

CASM Tutorial

Brian Puchala²,
John C. Thomas¹
Anton Van der Ven¹

PRISMS Workshop
August 4-5, 2025

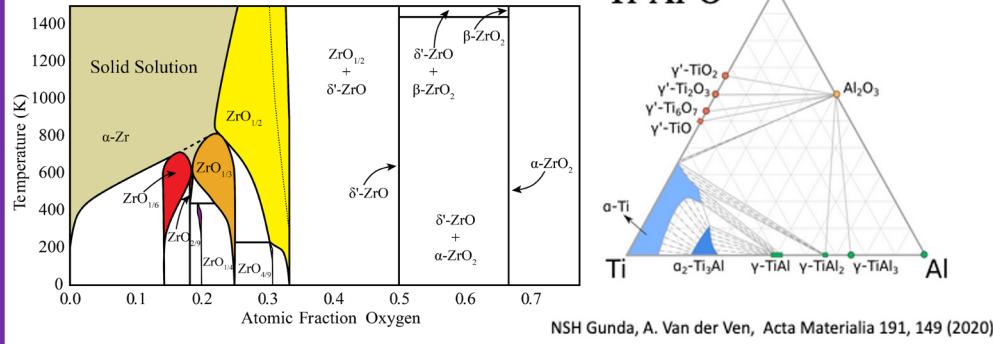
¹Materials Department, University of California Santa Barbara

²Department of Materials Science & Engineering, University of Michigan

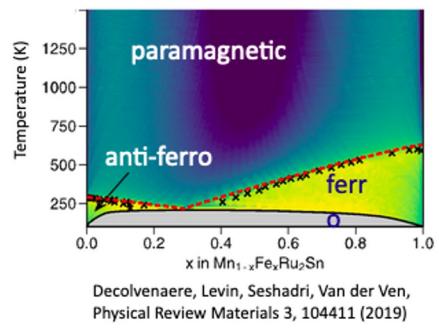
First-Principles Statistical Mechanics

Thermodynamics

Phase Diagrams

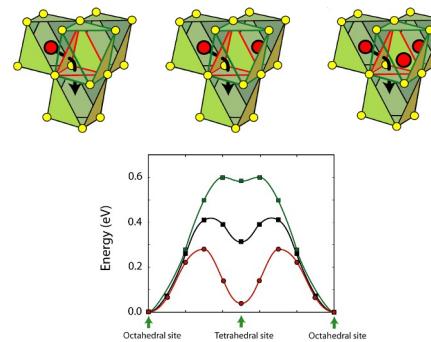


Chemo-Magnetic phase stability

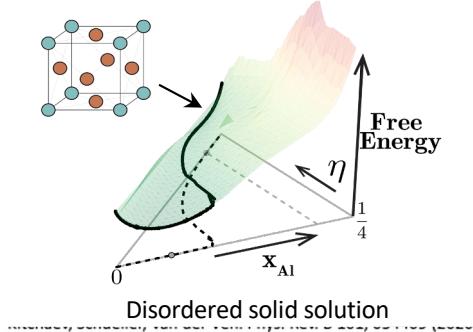


Kinetics

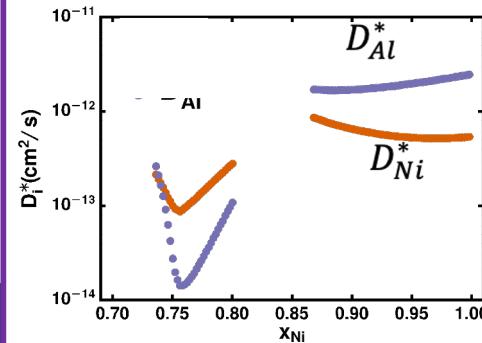
Diffusion in disordered alloys



Free Energies

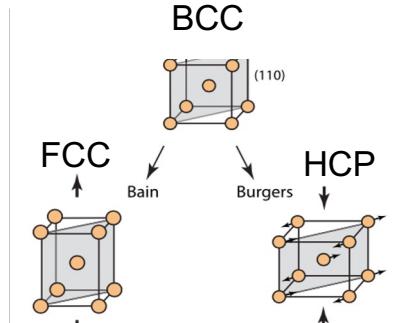


Diffusion Coefficients

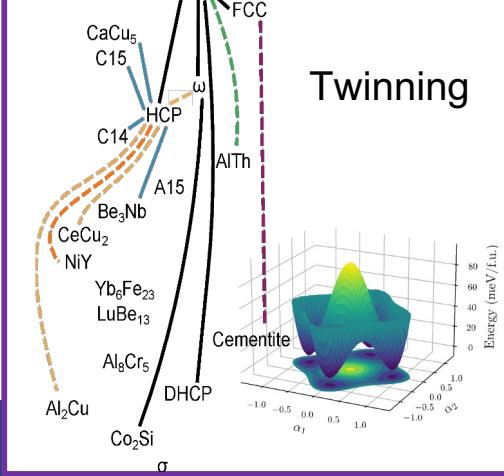


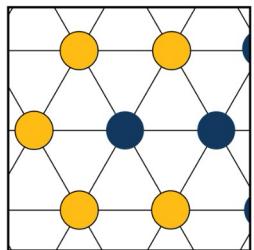
Chemo-Mechanics

Structural transformations



Twinning





CASM

A Clusters Approach to Statistical Mechanics

Puchala, et al. Comp. Mater. Sci. 217, 111987 (2023).

Contents lists available at [ScienceDirect](#)
Computational Materials Science
journal homepage: www.elsevier.com/locate/commatsci

Full length article

CASM — A software package for first-principles based study of multicomponent crystalline solids

Brian Puchala ^{a,*¹}, John C. Thomas ^{b,*¹}, Anirudh Raju Natarajan ^b, Jon Gabriel Goiri ^b, Sesha Sai Behara ^b, Jonas L. Kaufman ^b, Anton Van der Ven ^{b,*}

^a Department of Materials Science and Engineering, University of Michigan, Ann Arbor, United States of America

^b Materials Department, University of California, Santa Barbara, United States of America



Office of
Science

PRISMS



CASM v2 Overview:

https://prisms-center.github.io/CASMcode_pydocs/overview/latest/

CASM Workflow

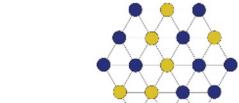
CASM Project

Parent Crystal Structure

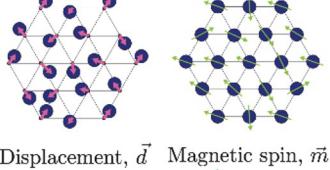
Lattice vectors, $\vec{L} = (\vec{l}_1, \vec{l}_2, \vec{l}_3)$

Basis site coordinates, $B = (\vec{b}_1, \vec{b}_2, \dots)$

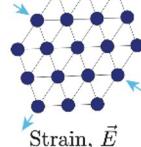
Degrees of Freedom (DoF)



Occupation, \vec{s}



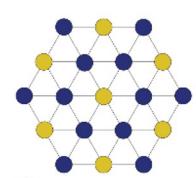
Displacement, \vec{d} Magnetic spin, \vec{m}



Strain, \vec{E}

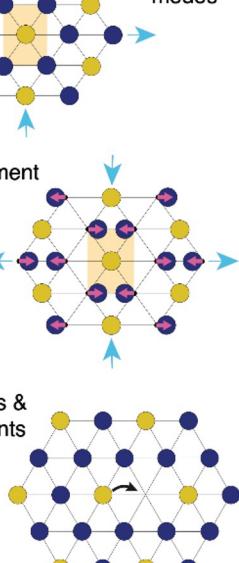
Derivative Structure Enumeration

Discrete occupation ordering

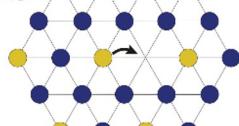


+ Strain modes

+ Displacement modes

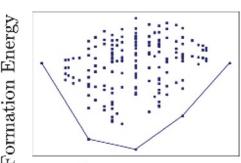


Migration event types & environments



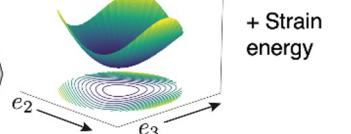
First Principles Calculations

Discrete occupation ordering energy



Formation Energy

Concentration



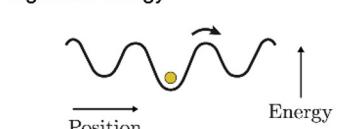
+ Strain energy

+ Displacement energy



+ Relaxation strain, displacement, magnetic spin, orientation, ...

Migration energy



Position Energy

Cluster Expansion Effective Hamiltonian

$$E(\vec{s}, \vec{d}, \vec{m}, \vec{q}, \vec{E}) = \sum_i m_i V_i \Gamma_i(\vec{s}, \vec{d}, \vec{m}, \vec{q}, \vec{E})$$

Symmetry-adapted basis functions, Γ_i
Expansion coefficients, V_i
Cluster multiplicity, m_i

Monte Carlo Calculations

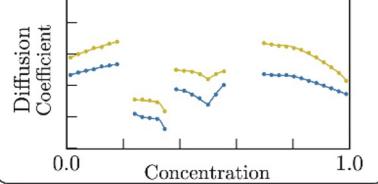
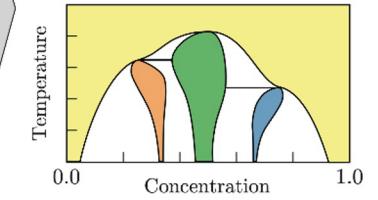
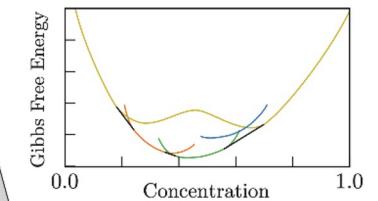
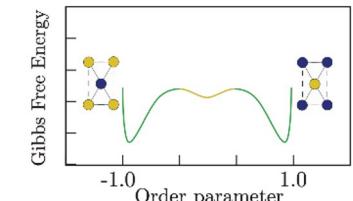
Metropolis, N-fold way, Hamiltonian

Kinetic Monte Carlo Calculations

Local-cluster Expansion of Diffusion Barriers

$$\Delta E_{\delta}^{KRA}(\vec{s}) = \sum_i m_i V_i \Gamma_i(\vec{s})$$

Finite Temperature Properties



CASMv2: About CASM

- CASM v2 consists of several Python packages
 - Easier to learn, to use, and to develop
- Python packages (libcasm-) with C++ extensions for core features
 - Fast symmetry application for enumeration
 - Efficient Monte Carlo calculations
 - Python interface using pybind11
- Pure python packages (casm-) for ease of use
 - casm-bset: Generate symmetry-adapted functions
 - casm-flow: (coming soon) Manage Monte Carlo calculations
 - casm-tools: CLI tools and helper methods
 - casm-project: Interact with project data **← Users start here**

CASMv2: Installation for Si-Ge tutorials

```
# ~~ Install ~~  
# load python3.11 and gcc/11.3.0  
module load python/3.11  
module load gcc/11.3.0  
  
# install casm-project  
git clone https://github.com/prisms-center/CASMcode_project.git  
cd CASMcode_project  
pip install .  
pip install -r notebooks_requirements.txt  
  
# ~~ Setup environment and test ~~  
export PATH=/home/<username>/.local/bin:$PATH  
export CASM_PREFIX=$(python -m libcasm.casmglobal --prefix)  
casm-calc -h  
python -m casm.bset --test  
  
# ~~ Launch Jupyter ~~  
jupyter lab
```

CASMv2: Getting Started

- Run Si-Ge cluster expansion tutorial notebooks in CASMcode_project:
 - notebooks/SiGe_occ_part1.v2.ipynb
 - CASM project initialization
 - Configuration enumeration
 - Calculation
 - Import and mapping
 - notebooks/SiGe_occ_part2.v2.ipynb
 - Cluster expansion construction & fitting
 - Monte Carlo calculation

CASMv2: Tutorials

The screenshot shows a Jupyter Notebook interface with two tabs open:

- SIGE_OCC_PART1.V2.IPYNB**: A sidebar menu listing various sections of the project:
 - Si-Ge cluster expansion workflow - part 1
 - Project initialization
 - Specify the "prim"
 - Initialize a CASM project
 - Check prim symmetry
 - Enumeration
 - Introduction
 - Supercell enumeration
 - Enumerating supercells by volume
 - Enumeration Data
 - SupercellSet
 - Storing multiple enumerations
 - Configuration enumeration
 - Enumerating configurations by supercell
 - ConfigurationSet
 - Conversion to structure
 - Order in ConfigurationSet
 - Configuration list
 - Selecting configurations
 - Clearing configurations
- SiGe_occ_part1.v2.ipynb**: The active notebook tab, titled "Si-Ge cluster expansion workflow - part 1".

This is a CASM project tutorial to generate a phase diagram using a Si-Ge binary alloy cluster expansion fit to DFT calculations. The overall workflow is split into two parts.

Topics covered in part 1:

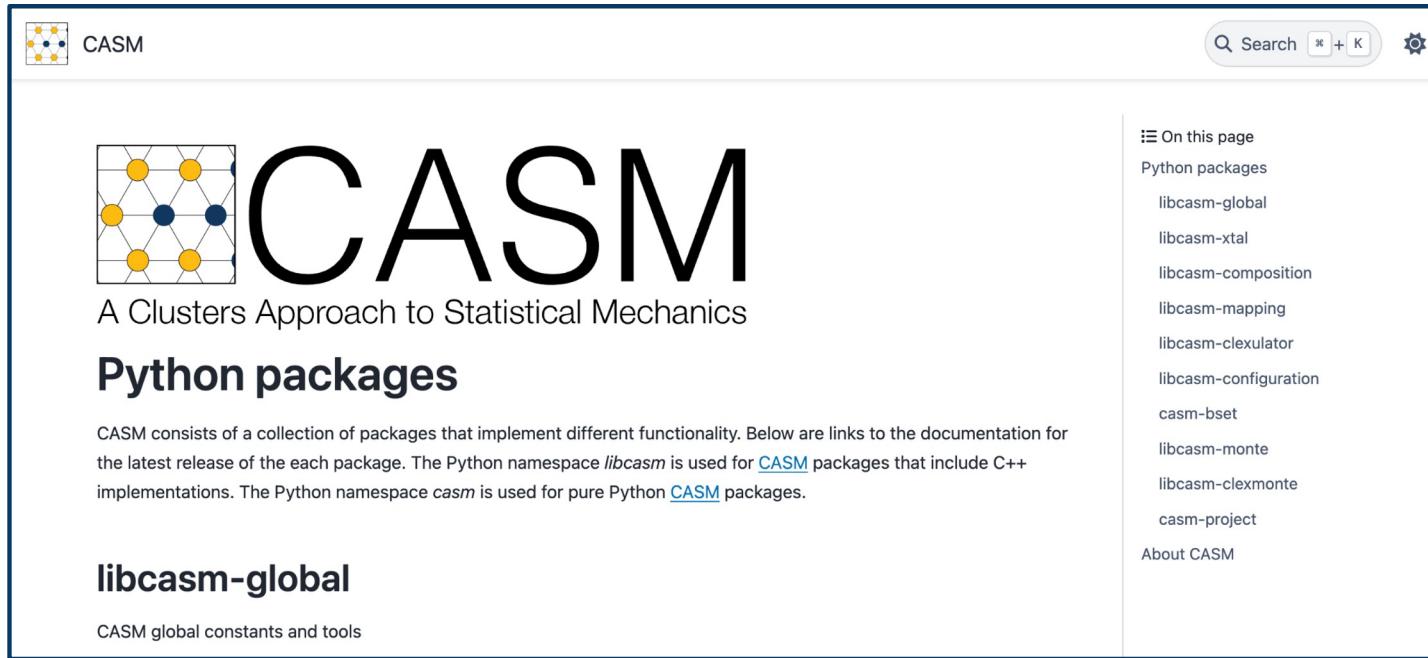
 - Project initialization:** Define the primitive crystal structure and allowed atoms on each crystal site
 - Enumeration:** Enumerate crystal structures which are symmetrically distinct orderings of the atoms allowed by the prim occupation DoF
 - Calculation:** Calculate the energies of the enumerated structures using DFT
 - Import and mapping:** Import calculation results, mapping to orderings on the prim
 - Set reference states:** Choose reference states to define a formation energy for each structure
 - Query:** Query calculation properties

```
[1]: import pathlib  
  
import libcasm.xtal as xtal  
from casm.project import Project  
from casm.project.json_io import safe_dump
```

CASMv2: Docs

CASMv2 Python package docs:

- https://prisms-center.github.io/CASMcode_pydocs/overview/latest/



The screenshot shows the homepage of the CASMv2 Python package documentation. At the top left is the CASM logo, which consists of a grid of colored circles (yellow and blue) and the word "CASM". The main title "CASM" is in large, bold, black font. Below it is the subtitle "A Clusters Approach to Statistical Mechanics". A section titled "Python packages" is present, with a sub-section for "libcasm-global". On the right side, there is a sidebar with a search bar and a "On this page" menu containing links to various Python packages: "Python packages", "libcasm-global", "libcasm-xtal", "libcasm-composition", "libcasm-mapping", "libcasm-clexulator", "libcasm-configuration", "casm-bset", "libcasm-monte", "libcasm-clexmonte", "casm-project", and "About CASM".

Project initialization

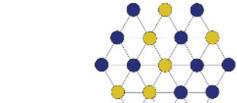
CASM Project

Parent Crystal Structure

Lattice vectors, $\vec{L} = (\vec{l}_1, \vec{l}_2, \vec{l}_3)$

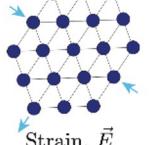
Basis site coordinates, $B = (\vec{b}_1, \vec{b}_2, \dots)$

Degrees of Freedom (DoF)



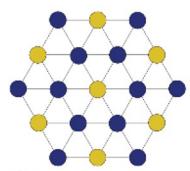
Occupation, \vec{s}

Displacement, \vec{d} Magnetic spin, \vec{m}



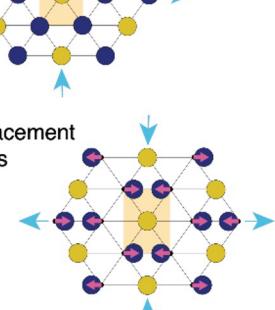
Derivative Structure Enumeration

Discrete occupation ordering

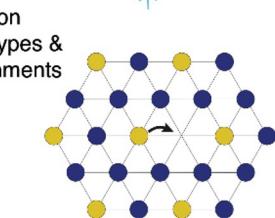


+ Strain modes

+ Displacement modes

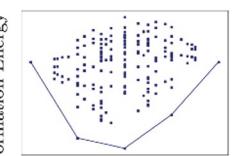


Migration event types & environments



First Principles Calculations

Discrete occupation ordering energy



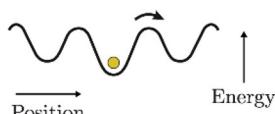
+ Strain energy

+ Displacement energy



+ Relaxation strain, displacement, magnetic spin, orientation, ...

Migration energy



Cluster Expansion Effective Hamiltonian

$$E(\vec{s}, \vec{d}, \vec{m}, \vec{q}, \vec{E}) = \sum_i m_i V_i \Gamma_i(\vec{s}, \vec{d}, \vec{m}, \vec{q}, \vec{E})$$

Symmetry-adapted basis functions, Γ_i
Expansion coefficients, V_i
Cluster multiplicity, m_i

Monte Carlo Calculations

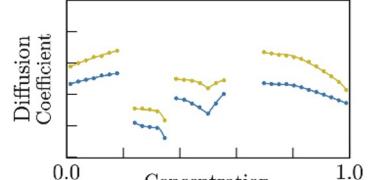
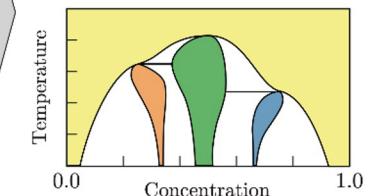
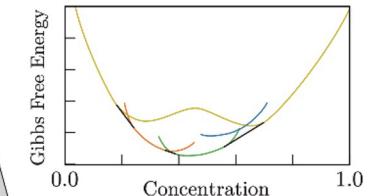
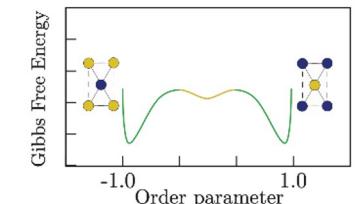
Metropolis, N-fold way, Hamiltonian

Kinetic Monte Carlo Calculations

Local-cluster Expansion of Diffusion Barriers

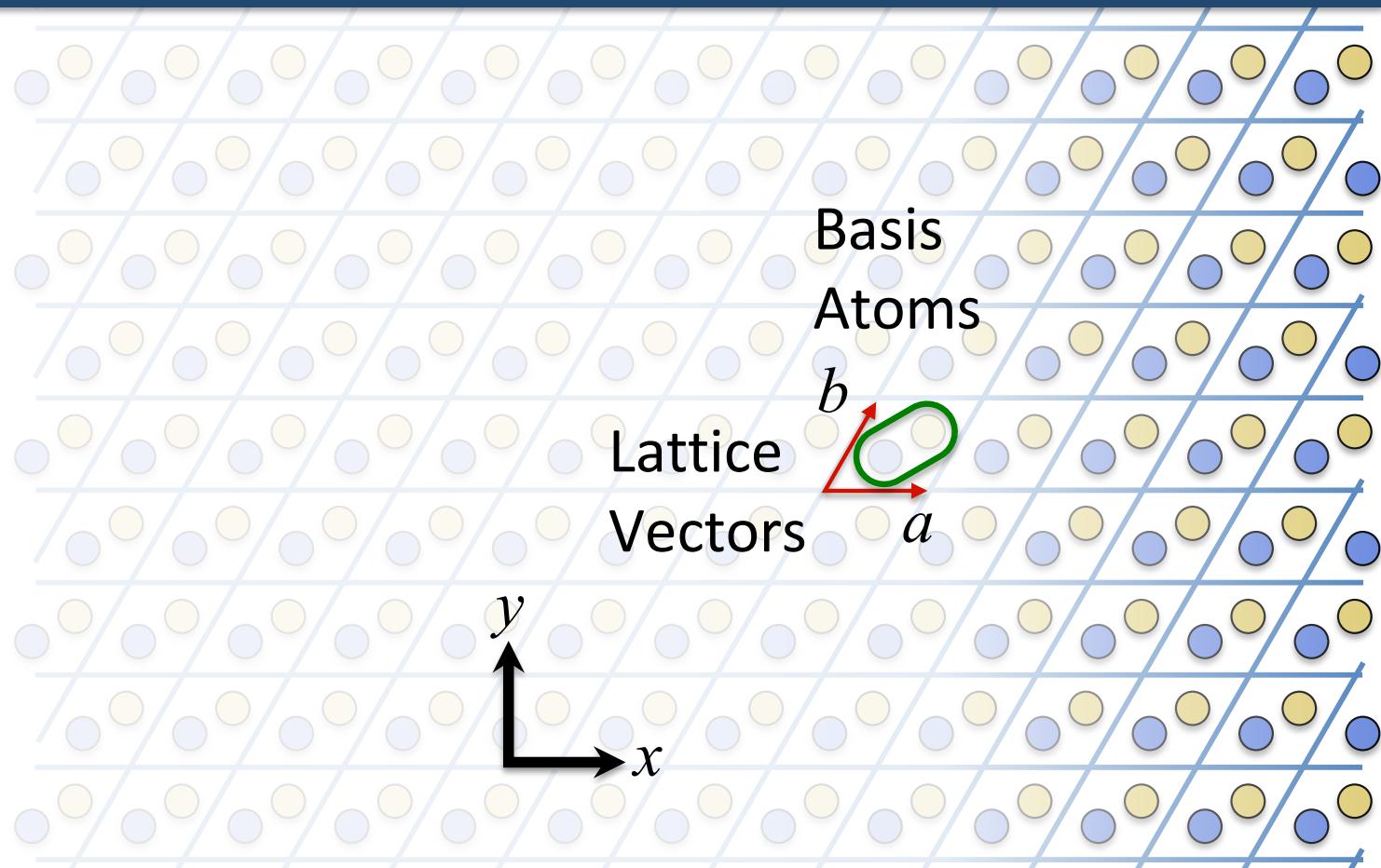
$$\Delta E_{\delta}^{KRA}(\vec{s}) = \sum_i m_i V_i \Gamma_i(\vec{s})$$

Finite Temperature Properties



Project Initialization

Crystal structure

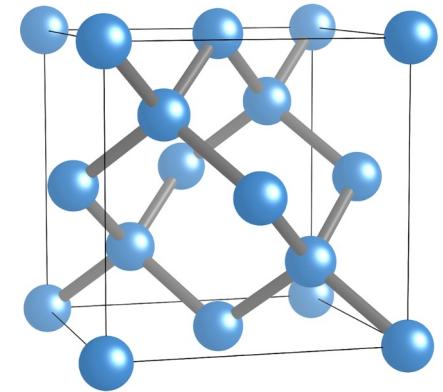
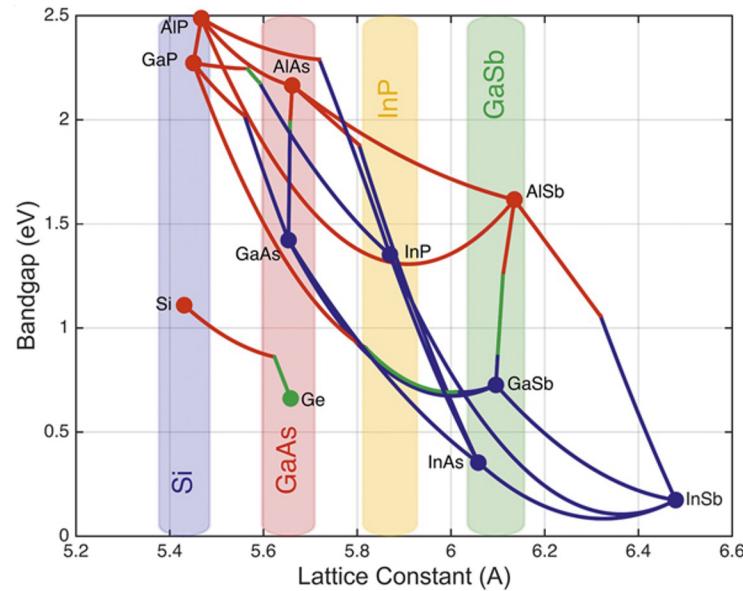


PRiSMS

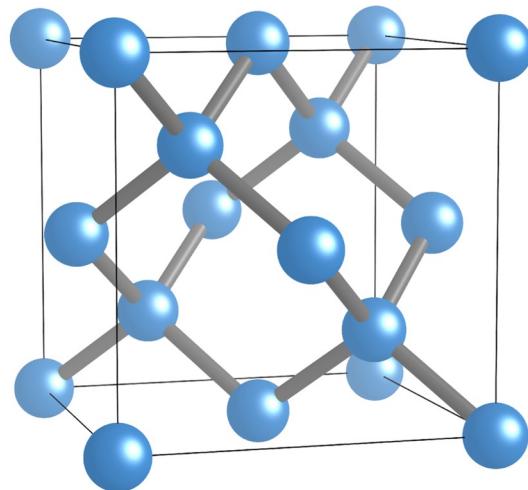
Tutorial Project: $\text{Si}_{1-x}\text{Ge}_x$ binary system

Important semiconductor alloy with tunable bandgap and lattice parameter.

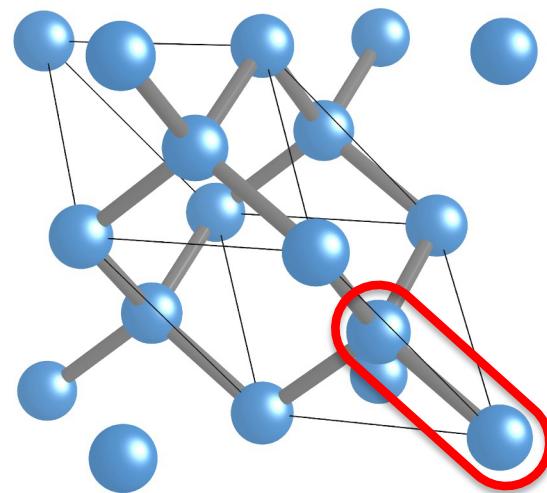
Model alloying of Si and Ge on sites of diamond cubic crystal



Tutorial Project: $\text{Si}_{1-x}\text{Ge}_x$ binary system



Conventional cubic cell

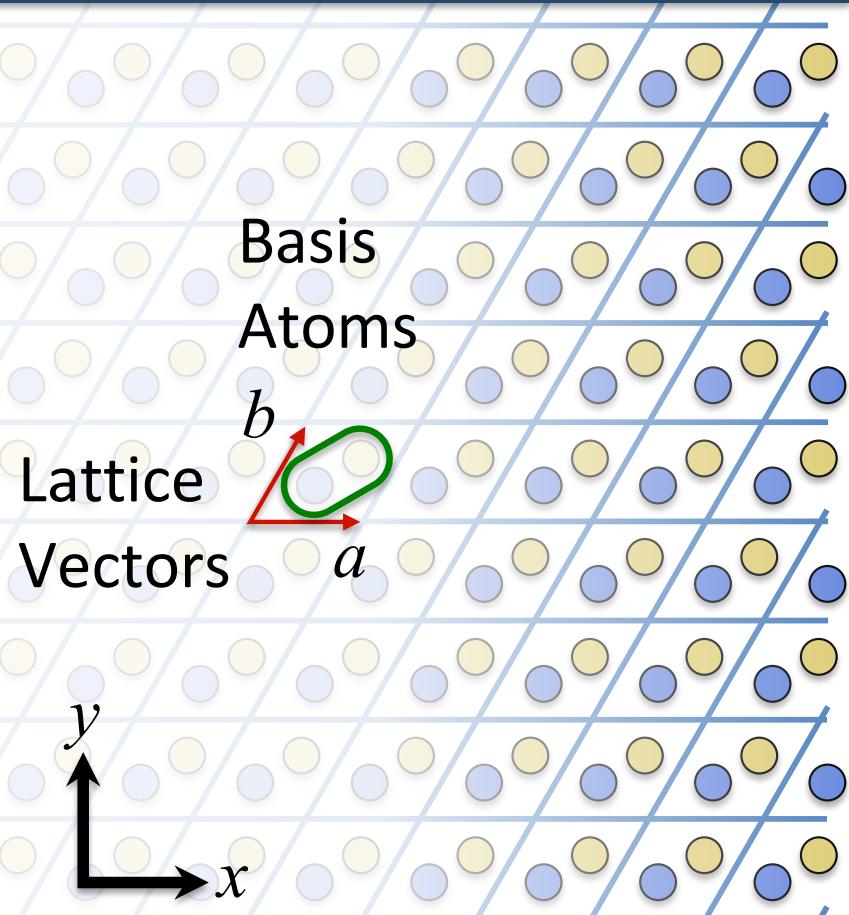


Primitive cell

Project Initialization

prim.json:

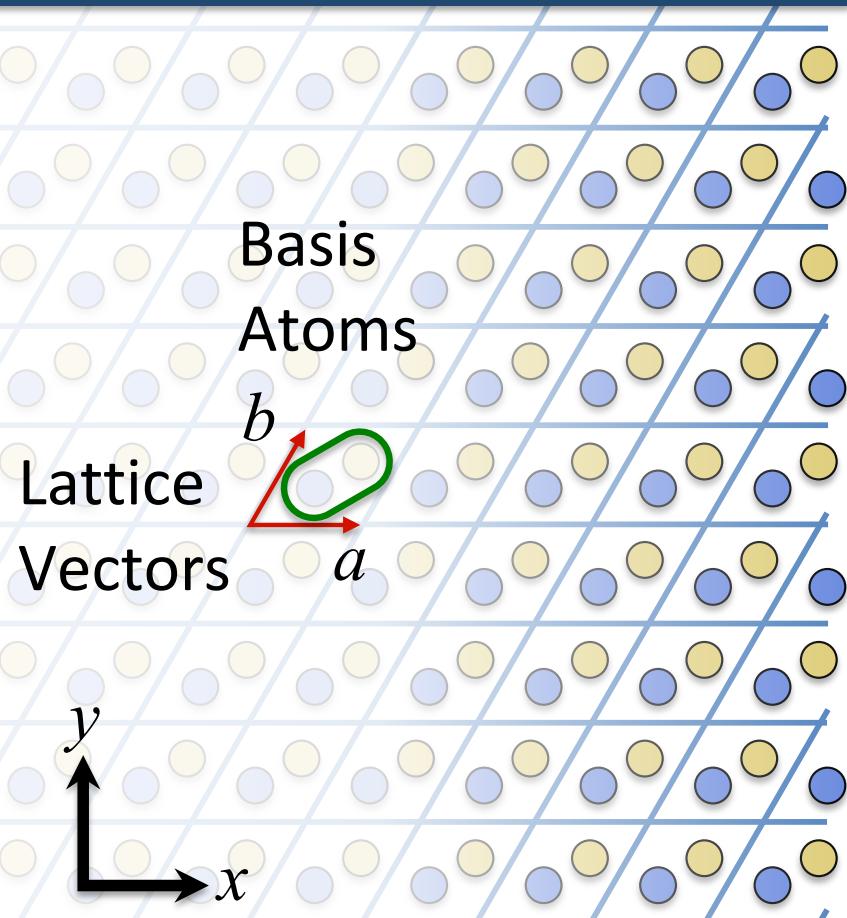
```
{  
  "title" : "SiGe",  
  "description" : "Si/Ge on diamond cubic crystal",  
  "coordinate_mode" : "Fractional",  
  "lattice_vectors" : [  
    [ 0.0000, 2.8000, 2.8000 ],  
    [ 2.8000, 0.0000, 2.8000 ],  
    [ 2.8000, 2.8000, 0.0000 ]  
  ],  
  "basis" : [  
    {  
      "coordinate" : [ 0.0000, 0.0000, 0.0000 ],  
      "occupant_dof" : [ "Si", "Ge" ]  
    },  
    {  
      "coordinate" : [ 0.2500, 0.2500, 0.2500 ],  
      "occupant_dof" : [ "Si", "Ge" ]  
    }  
  ]  
}
```



Project Initialization

prim.json:

```
{  
  "title" : "SiGe",  
  "description" : "Si/Ge on diamond cubic crystal",  
  "coordinate_mode" : "Fractional",  
  "lattice_vectors" : [  
    [ 0.0000, 2.8000, 2.8000 ],  
    [ 2.8000, 0.0000, 2.8000 ],  
    [ 2.8000, 2.8000, 0.0000 ]  
  ],  
  "basis" : [  
    {  
      "coordinate" : [ 0.0000, 0.0000, 0.0000 ],  
      "occupant_dof" : [ "Si", "Ge" ]  
    },  
    {  
      "coordinate" : [ 0.2500, 0.2500, 0.2500 ],  
      "occupant_dof" : [ "Si", "Ge" ]  
    }  
  ]  
}
```



Project Initialization

The screenshot shows a Jupyter Notebook interface with two tabs: "SiGe_occ_part1.ipynb" and "Terminal 2". The left sidebar contains a tree view of a project structure:

- Si-Ge cluster expansion workflow - part 1
 - Project initialization
 - Specify the "prim"
 - Initialize a CASM project
 - Check prim symmetry
 - Enumeration
 - Introduction
 - Supercell enumeration
 - Enumerating supercells by volume
 - Enumeration Data
 - SupercellSet
 - Storing multiple enumerations
 - Configuration enumeration
 - Enumerating configurations by supercell
 - ConfigurationSet
 - Conversion to structure
 - Order in ConfigurationSet
 - Configuration list
 - Selecting configurations
 - Clearing configurations
 - Filtered enumeration
 - Other enumeration methods
 - Note on configuration names
 - Equivalence of configurations
 - Acting on enumeration data
 - Get an enumeration by id
 - List all enumerations
 - Copy an enumeration
 - Merge enumerations
 - Remove an enumeration

Enumeration

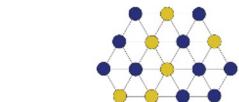
CASM Project

Parent Crystal Structure

Lattice vectors, $\vec{L} = (\vec{l}_1, \vec{l}_2, \vec{l}_3)$

Basis site coordinates, $B = (\vec{b}_1, \vec{b}_2, \dots)$

Degrees of Freedom (DoF)



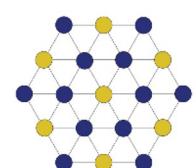
Occupation, \vec{s}

Displacement, \vec{d} Magnetic spin, \vec{m}

Strain, \vec{E}

Derivative Structure Enumeration

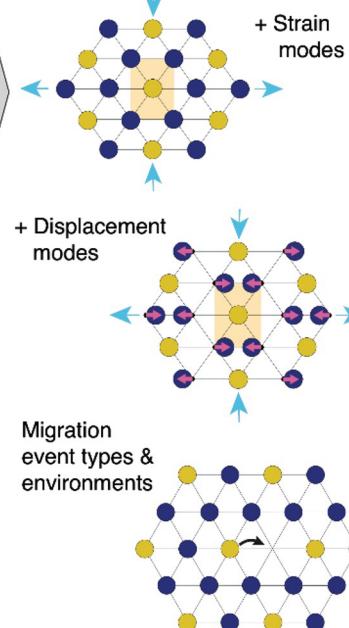
Discrete occupation ordering



+ Strain modes

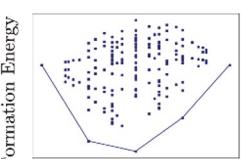
+ Displacement modes

Migration event types & environments

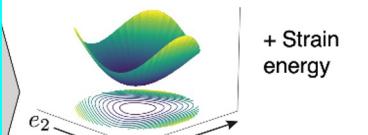


First Principles Calculations

Discrete occupation ordering energy



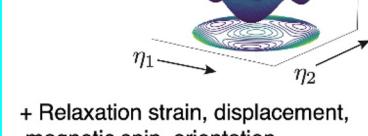
Concentration



+ Strain energy

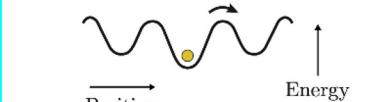


+ Displacement energy



+ Relaxation strain, displacement, magnetic spin, orientation, ...

Migration energy



Position Energy

Cluster Expansion Effective Hamiltonian

$$E(\vec{s}, \vec{d}, \vec{m}, \vec{q}, \vec{E}) = \sum_i m_i V_i \Gamma_i(\vec{s}, \vec{d}, \vec{m}, \vec{q}, \vec{E})$$

Symmetry-adapted basis functions, Γ_i
Expansion coefficients, V_i
Cluster multiplicity, m_i

Monte Carlo Calculations

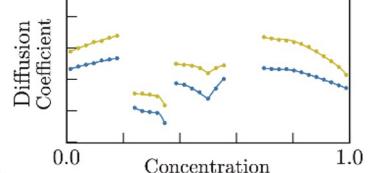
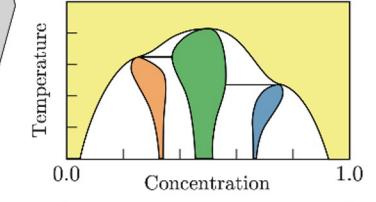
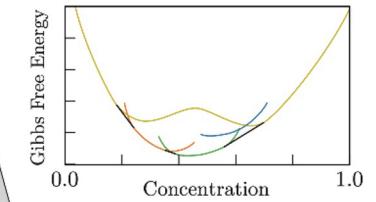
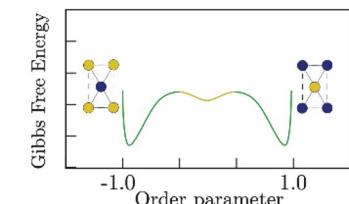
Metropolis, N-fold way, Hamiltonian

Kinetic Monte Carlo Calculations

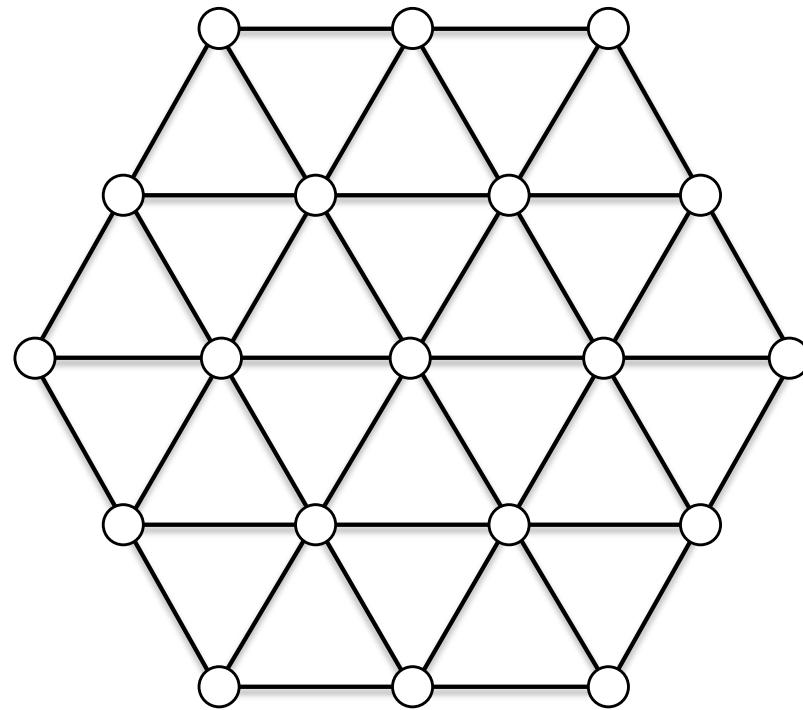
Local-cluster Expansion of Diffusion Barriers

$$\Delta E_{\delta}^{KRA}(\vec{s}) = \sum_i m_i V_i \Gamma_i(\vec{s})$$

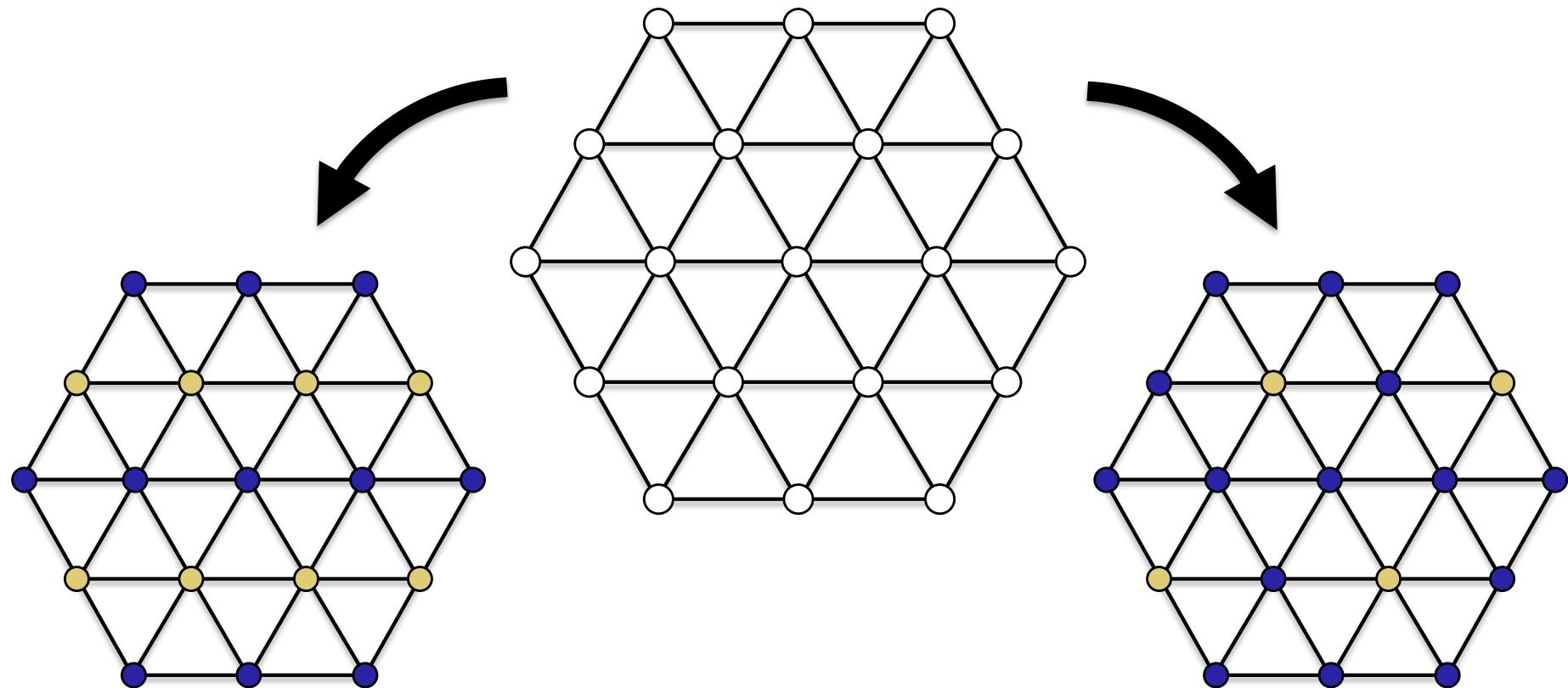
Finite Temperature Properties



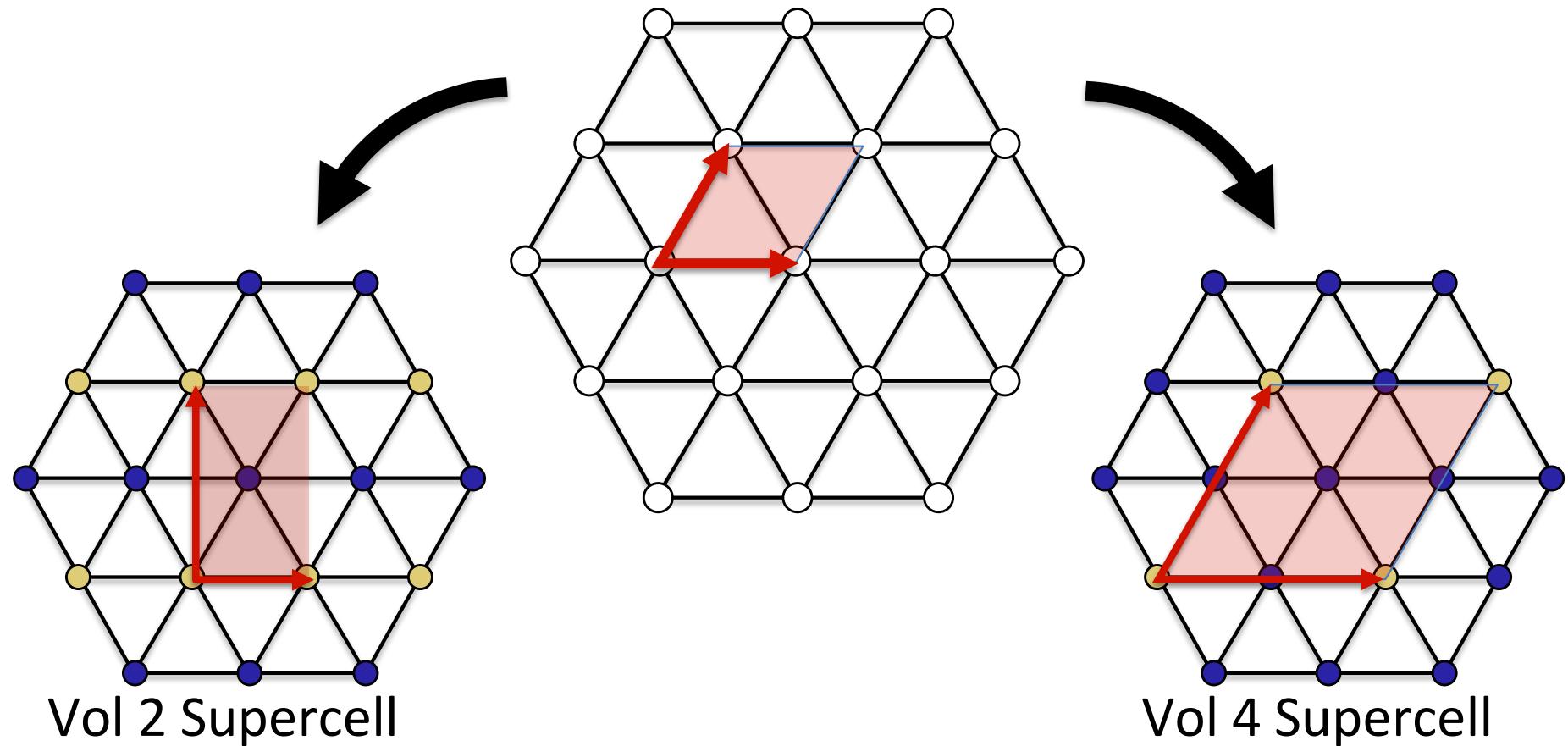
Crystal configurations



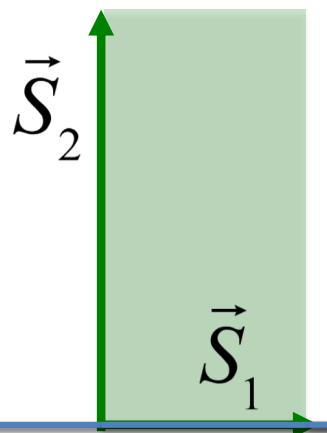
Crystal configurations



Crystal configurations

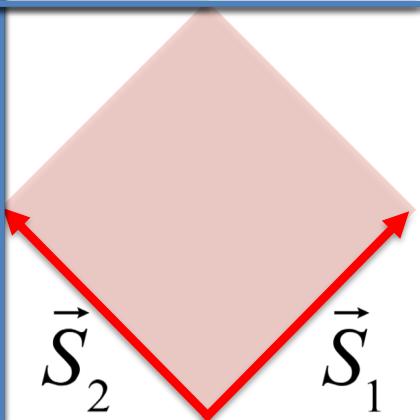
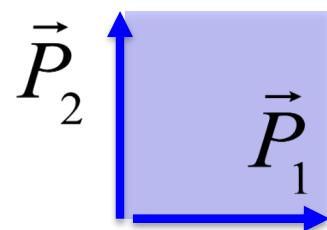


Unique supercell enumeration



$$\begin{aligned}\vec{S}_1 &= \vec{P}_1 \\ \vec{S}_2 &= 2\vec{P}_2\end{aligned}$$

Supercells - linear combinations of primitive cell lattice vectors



$$\begin{aligned}\vec{S}_1 &= \vec{P}_1 + \vec{P}_2 \\ \vec{S}_2 &= -\vec{P}_1 + \vec{P}_2\end{aligned}$$

Supercell enumeration

The screenshot shows a Jupyter Notebook interface with two tabs: "SiGE_OCC_PART1.V2.ipynb" and "SiGe_occ_part1.v2.ipynb". The "SiGe_occ_part1.v2.ipynb" tab is active, displaying a section titled "Supercell enumeration" with the sub-section "Enumerating supercells by volume". The text explains how to construct an enumeration with id="supercells_by_volume.1" using project.enum.get, which returns an EnumData object. It notes that the method EnumData.supercells_by_volume enumerates symmetrically distinct supercells from a minimum to a maximum volume, specified as integer multiples of the prim unit cell volume. It also lists additional options described in the reference documentation.

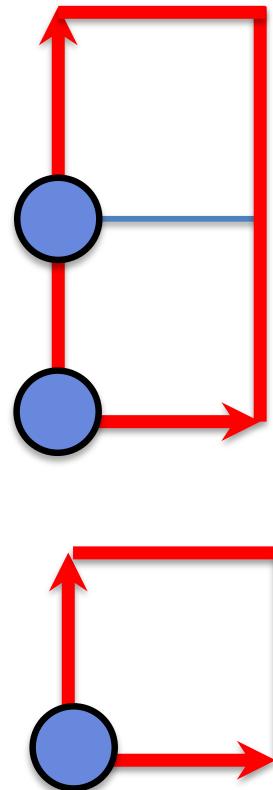
The code cell [5] contains the following Python code:

```
# Enumerate supercells with volume 1 to 4
enum = project.enum.get("supercells_by_volume.1")
enum.supercells_by_volume(
    max=4,
    min=1,
    verbose=True,
)
```

The output of the code cell shows the generated supercell configurations:

```
-- Begin: Enumerating supercells by volume --
Generated: SCEL1_1_1_0_0_0
Generated: SCEL2_2_1_1_0_1_1
Generated: SCEL2_2_1_1_0_0_1
Generated: SCEL3_3_1_1_0_2_2
Generated: SCEL3_3_1_1_0_2_1
Generated: SCEL3_3_1_1_0_0_2
Generated: SCEL4_4_1_1_0_0_0
Generated: SCEL4_4_1_1_0_1_0
Generated: SCEL4_4_1_1_0_0_2
Generated: SCEL4_4_1_1_0_0_3
Generated: SCEL4_4_1_1_0_2_1
Generated: SCEL4_2_2_1_0_1_0
```

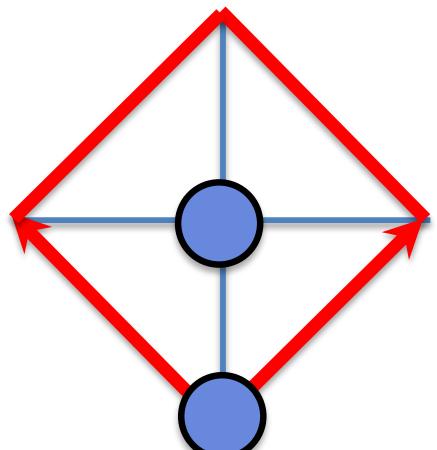
Unique configuration enumeration



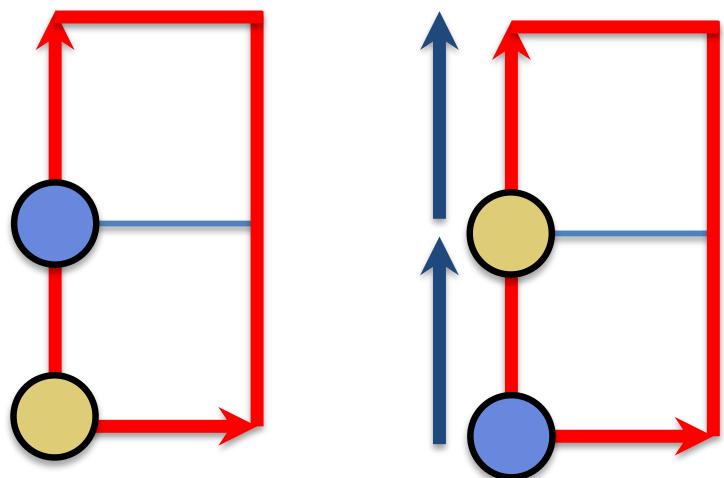
Equivalent configurations:

A:
B:

- Supercells of the volume 1 configuration



Unique configuration enumeration

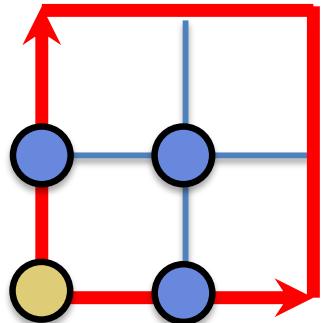


Equivalent configurations:

- A:
- B:

- Translation maps one configuration onto the other

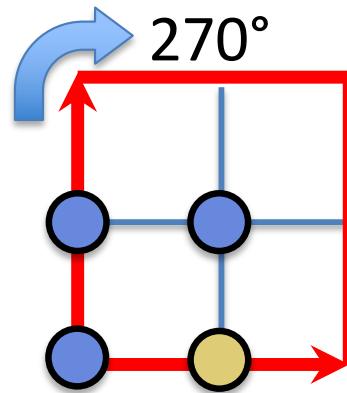
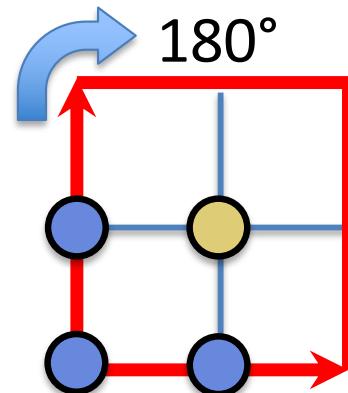
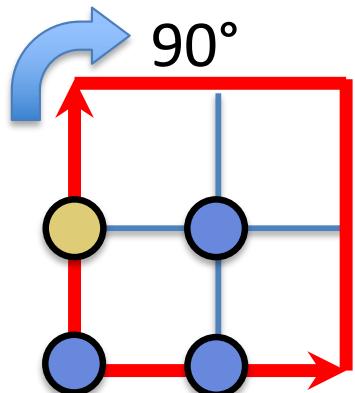
Unique configuration enumeration



Equivalent configurations:

A:
B:

- Rotation or translation maps occupants



Configuration enumeration

The screenshot shows a Jupyter Notebook interface with two tabs: "SiGE_OCC_PART1.V2.ipynb" and "Terminal 2". The notebook tab displays a section titled "Configuration enumeration" with the following content:

Configuration enumeration

Enumerating configurations by supercell

The method `occ_by_supercell` enumerates all occupations in supercells ranging from a minimum to a maximum volume. All configurations are guaranteed to be in a canonical supercell. By default it:

- only outputs primitive configurations,
- only outputs configurations in canonical form (the configuration that compares greatest to all configurations in a supercell that can be mapped by symmetry operations).

With these defaults, if enumeration proceeds without skipping supercells, all symmetrically distinct configurations will be enumerated.

As with `enum.supercells_by_volume` it also has additional options, described in the reference documentation, for more complex use cases:

- Enumerate occupations in supercells of another supercell
- Enumerate occupations in 1d or 2d supercells
- Enumerate occupations in supercells with a fixed shape but different sizes

Warning: The number of possible occupations in a n -component alloy with m sites is n^m . Take care not to request too large of an enumeration.

In the code cell [10]:

```
# Enumerate configurations in supercells with volume 1 to 4
enum = project.enum.get("occ_by_supercell.1")
enum.occ_by_supercell(max=4, min=1)
```

The terminal tab shows the output of the code execution:

```
-- Begin: Enumerating occupations by supercell --
Enumerate configurations for: SCEL1_1_1_0_0_0
3 configurations (3 new, 0 excluded by filter)

Enumerate configurations for: SCEL2_2_1_1_0_1_1
4 configurations (4 new, 0 excluded by filter)
```

At the bottom, the status bar indicates "Mode: Command" and "Ln 1, Col 5".

Calculation / Import and Mapping

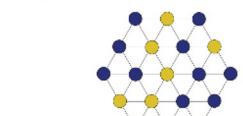
CASM Project

Parent Crystal Structure

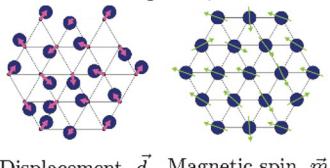
Lattice vectors, $\vec{L} = (\vec{l}_1, \vec{l}_2, \vec{l}_3)$

Basis site coordinates, $\vec{B} = (\vec{b}_1, \vec{b}_2, \dots)$

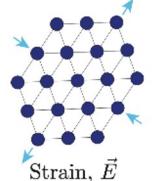
Degrees of Freedom (DoF)



Occupation, \vec{s}



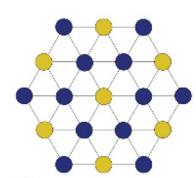
Displacement, \vec{d} Magnetic spin, \vec{m}



Strain, \vec{E}

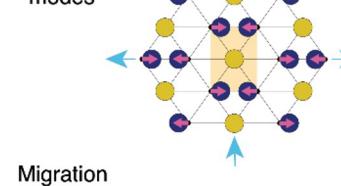
Derivative Structure Enumeration

Discrete occupation ordering

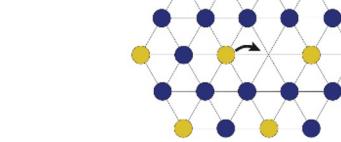


+ Strain modes

+ Displacement modes

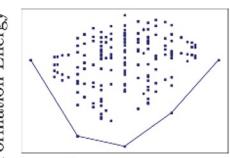


Migration event types & environments

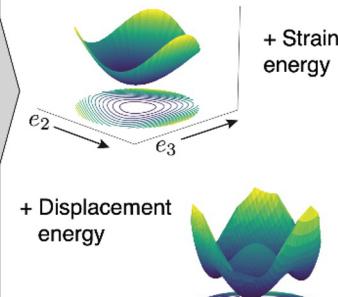


First Principles Calculations

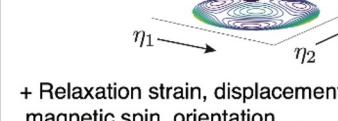
Discrete occupation ordering energy



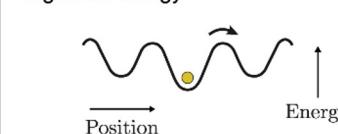
Concentration



+ Displacement energy



Migration energy



Position

Energy

Cluster Expansion Effective Hamiltonian

$$E(\vec{s}, \vec{d}, \vec{m}, \vec{q}, \vec{E}) = \sum_i m_i V_i \Gamma_i(\vec{s}, \vec{d}, \vec{m}, \vec{q}, \vec{E})$$

Symmetry-adapted basis functions, Γ_i
Expansion coefficients, V_i
Cluster multiplicity, m_i

Monte Carlo Calculations

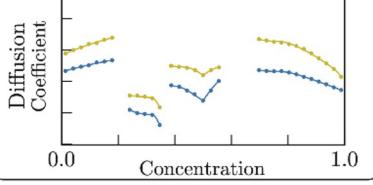
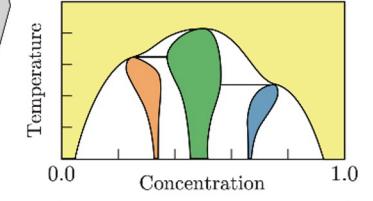
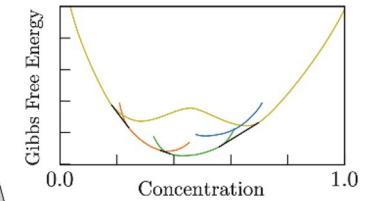
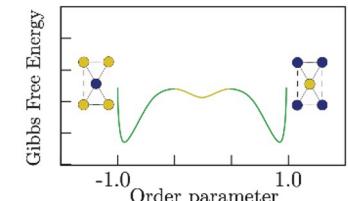
Metropolis, N-fold way, Hamiltonian

Kinetic Monte Carlo Calculations

Local-cluster Expansion of Diffusion Barriers

$$\Delta E_{\delta}^{KRA}(\vec{s}) = \sum_i m_i V_i \Gamma_i(\vec{s})$$

Finite Temperature Properties



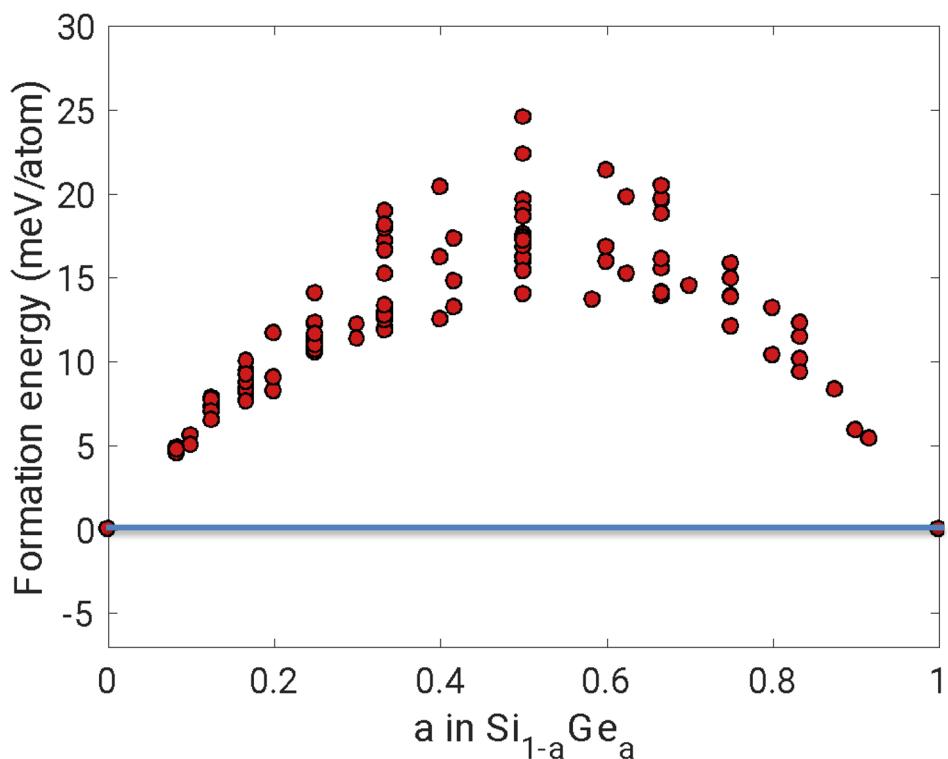
Calculation / Import and Mapping

CASM Configuration

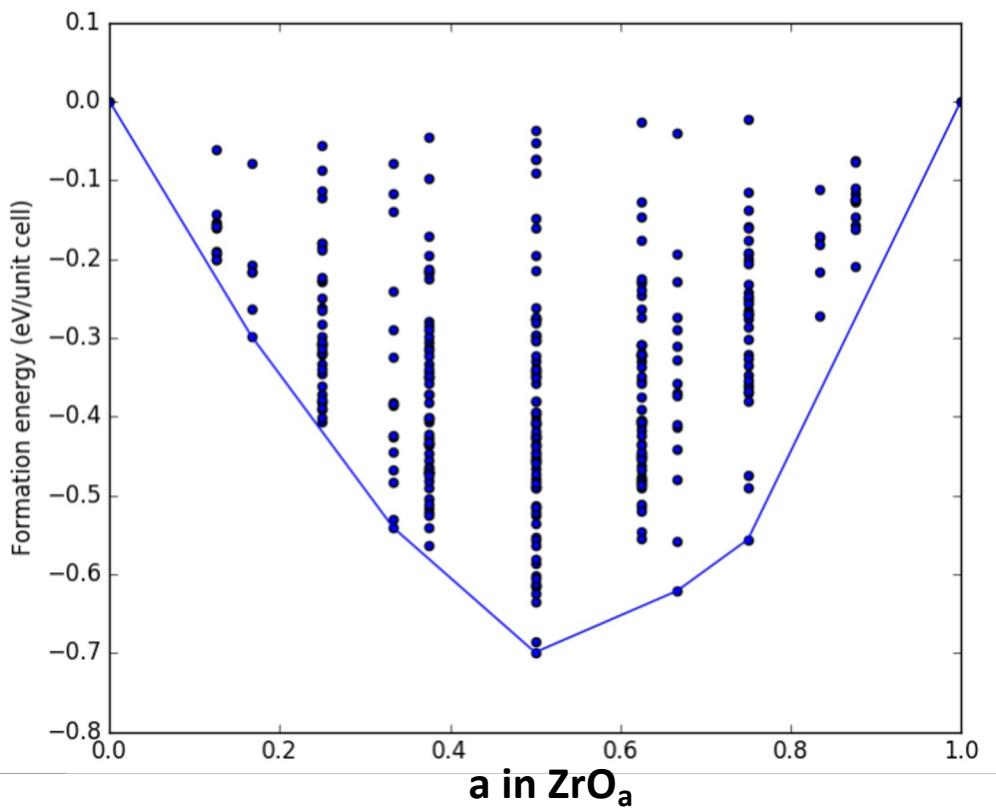
- CASM Structure
- DFT Calculation Input (VASP INCAR, KPOINTS, POSCAR, POTCAR)
- DFT Calculation Output (VASP OUTCAR)
- Import → CASM Structure w/ properties
- Mapping → CASM Configuration w/ properties

Typical DFT results

Miscibility Gap



Ordered groundstates



Calculation

The screenshot shows a Jupyter Notebook interface. On the left, there is a sidebar with a navigation menu titled "SIGE_OCC_PART1.V2.IPYNB". The menu items include:

- Selecting configurations
- Clearing configurations
- Filtered enumeration
- Other enumeration methods
- Note on configuration names
- Equivalence of configurations
- Acting on enumeration data
 - Get an enumeration by id
 - List all enumerations
 - Copy an enumeration
 - Merge enumerations
 - Remove an enumeration
- Calculation
 - Calculation directory structure
 - Calculation status file
- Calculation settings
 - Calculation settings for ASE + VASP
- Setup calculations
 - Select configurations
 - Generate VASP input files
 - Check input files
 - Checking calculation status
 - Generate Slurm job submission scripts
- Run calculations
 - Select configurations
 - Submit jobs: casm-calc submit
 - Check job status
 - Make CASM structures with properties
- Import and Mapping
 - Overview
 - Collect calculated structures

The main notebook area has tabs for "SiGe_occ_part1.v2.ipynb", "Terminal 2", and "Markdown". The "SiGe_occ_part1.v2.ipynb" tab is active. The content of the notebook includes:

Calculation

There are many different workflow tools for managing DFT calculations that could be interfaced with CASM. The calculation process implemented here involves the following conversions:

```
libcasm.configuration.Configuration  
-> libcasm.xtal.Structure  
-> ase.Atoms  
-> VASP input files (POSCAR, KPOINTS, INCAR, POTCAR)
```

- CASM provides a standard directory structures for saving input files. Using this structure also enables some CASM features for managing jobs on a compute cluster.
- CASM provides a very simple integration to [The Atomic Simulation Environment \(ase\)](#) which can be customized for a particular use case.
- Users can customize the input file setup and output file parsing to work with other DFT codes or more complicated workflows.

Calculation directory structure

The calculation directories created will be at:

```
calc_dir = <enum_dir>/training_data/calctype.<calctype_id>/<configname>/
```

Where

```
enum_dir = <project>/enumerations/enum.<enum_id>/
```

Here:

- calctype_id* - The calculation type ID used to organize calculations with different parameter sets or calculating different quantities (i.e. energy vs band structure).
- configname* - The configuration name used to identify the configuration in the enumeration. Can be obtained from

Mode: Command Ln 1, Col 1 SiGe_occ_part1.v2.ipynb 0 ⚡

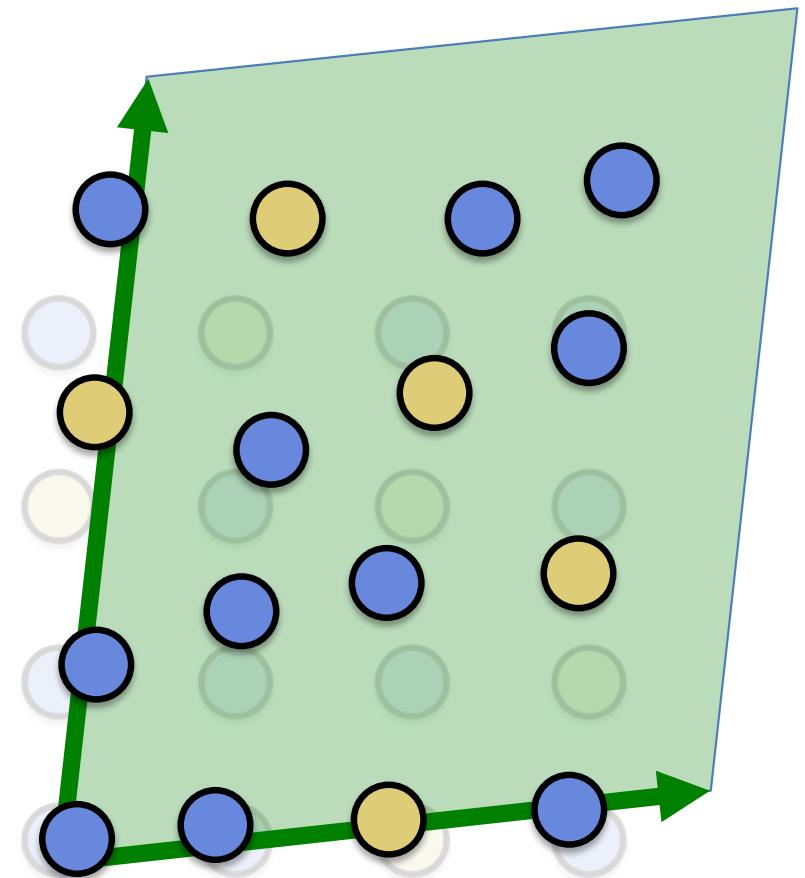
Import and Mapping

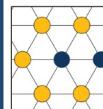
Very large relaxations may change the configuration.

There is no single valid mapping from a deformed structure and configuration.

CASM searches for mappings and scores them based on lattice deformation and atomic displacement.

Conflicts can be resolved using either min-energy or min-deformation rule





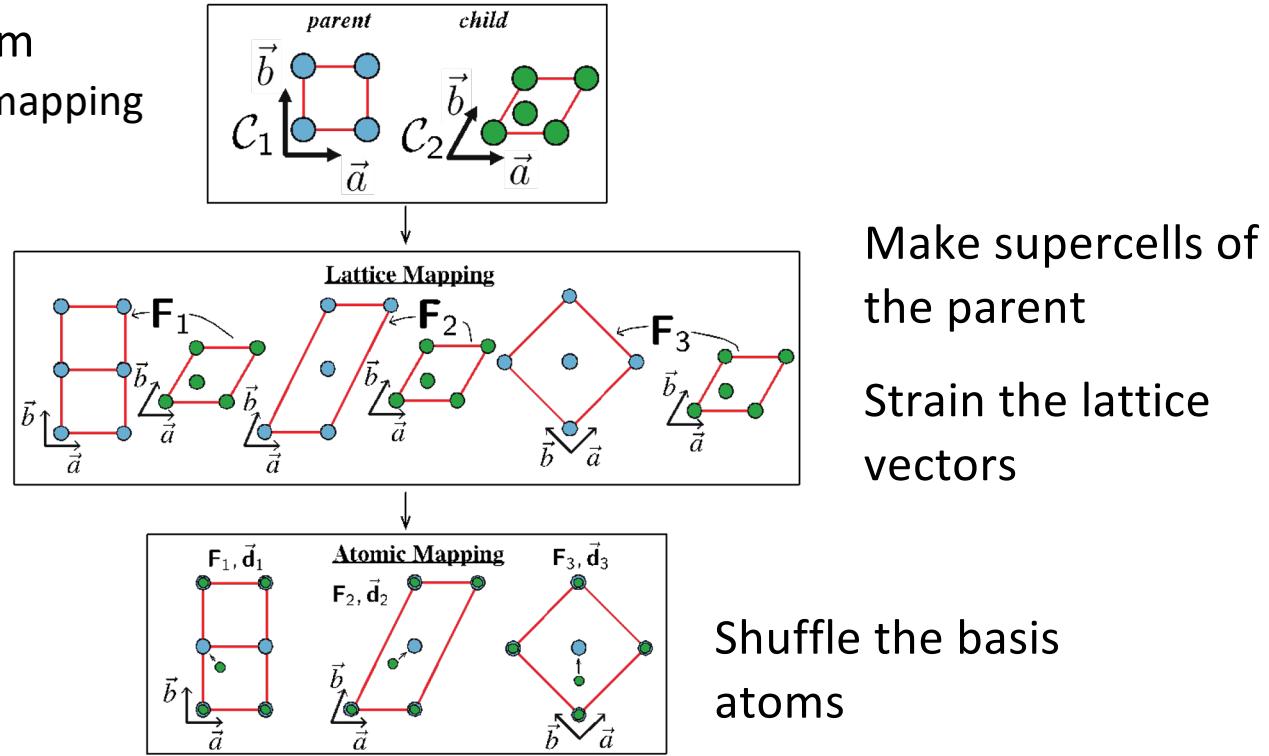
CASM

A Clusters Approach to Statistical Mechanics

CASM mapping algorithm

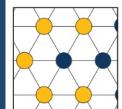
CASM mapping algorithm

- Available in libcasm-mapping



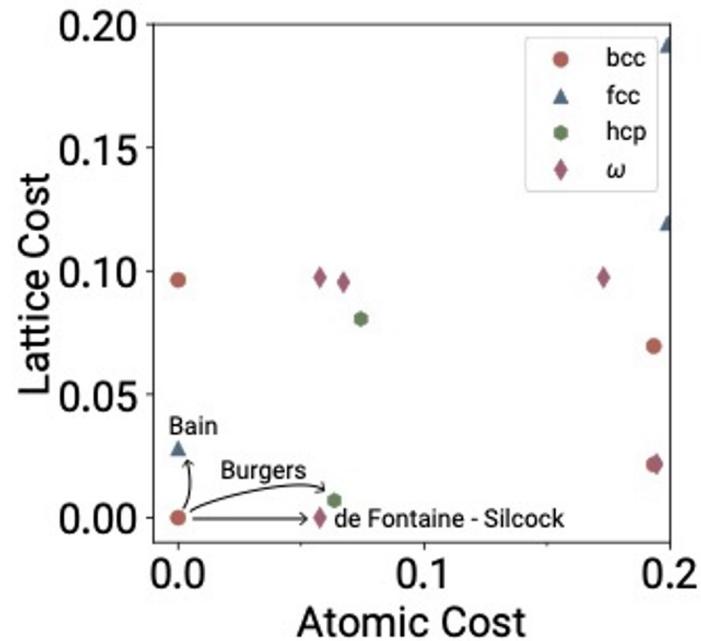
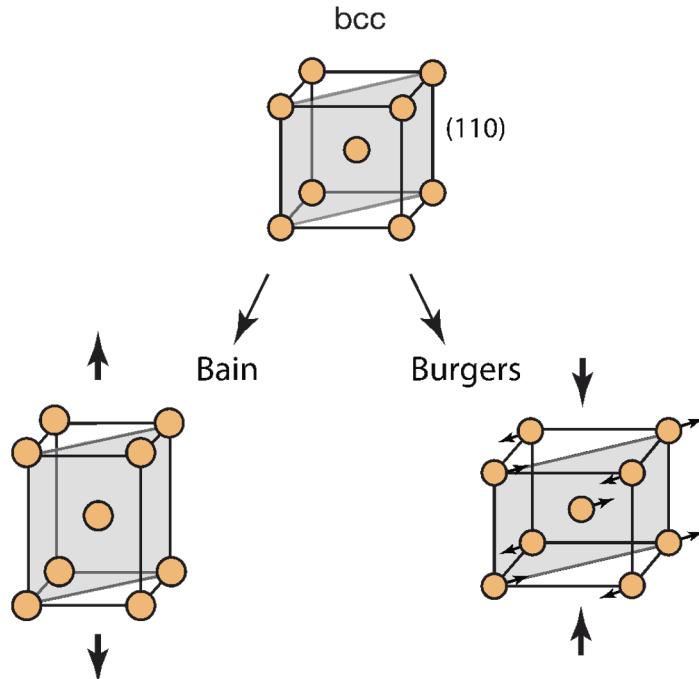
Make supercells of
the parent
Strain the lattice
vectors

Shuffle the basis
atoms



CASM mapping algorithm

Example: BCC parent





CASM

A Clusters Approach to Statistical Mechanics

libcasm-mapping

- Map structures
- Map lattices
 - Allow any orientation, or do not allow re-orientation
- Map atoms to sites
- Find k -best mappings
- Make interpolated structures after finding a mapping
- *libcasm.mapping.mapsearch*: Build your own mapping search algorithm, with custom constraints on the search

libcasm-mapping

Installation Usage Reference Bibliography

Search

On this page

libcasm-mapping

About CASM

License

Documentation

Show Source

CASM

A Clusters Approach to Statistical Mechanics

libcasm-mapping

The libcasm-mapping package is the CASM structure mapping module. This includes:

- Methods for searching for low-cost lattice, atom, and structure mappings, taking into account symmetry, based on the approach described in the paper [Thomas, Natarajan, and Van der Ven, *npj Computational Materials*, 7 \(2021\), 164.](#)
- Methods for generating interpolated structures based on mapping results
- Methods for generating symmetrically equivalent mappings
- Data structures and methods for creating custom mapping searches

https://prisms-center.github.io/CASMcode_pydocs/libcasm/mapping/2.0/index.html



CASM

A Clusters Approach to Statistical Mechanics

LatticeMapping

```
class libcasm.mapping.info.LatticeMapping(  
    self: LatticeMapping,  
    deformation_gradient: numpy.ndarray[numpy.float64[3, 3]],  
    transformation_matrix_to_super: numpy.ndarray[numpy.float64[3, 3]],  
    reorientation: numpy.ndarray[numpy.float64[3, 3]],  
)
```

A lattice mapping has the form:

$$FL_1TN = L_2,$$

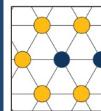
where:

- L_1 is a shape=(3,3) matrix with columns containing the reference “parent” lattice vectors
- L_2 is a shape=(3,3) matrix with columns containing the “child” lattice vectors
- F is the parent-to-child deformation gradient tensor, a shape=(3,3) matrix. F can be decomposed as:

$$F = QU = VQ$$

into a pure rigid transformation component, Q , (called an isometry) and either a right stretch tensor, U , or a left stretch tensor, V . U and V are both positive definite and symmetric and include all of the non-rigid lattice deformation. Isometries, Q , have the property that $Q^{-1} = Q^T$, where T is used to indicate matrix transpose.

- T is an integer transformation matrix that generates a superlattice of L_1
- N is a unimodular reorientation matrix that generates a lattice equivalent to L_1T with reoriented lattice vectors



AtomMapping

```
class libcasm.mapping.info.AtomMapping(  
    self: AtomMapping,  
    displacement: numpy.ndarray[numpy.float64[m, n]],  
    permutation: list[int],  
    translation: numpy.ndarray[numpy.float64[3, 1]],  
)
```

An atom mapping is defined in the context of an existing [LatticeMapping](#). An atom mapping has the form:

$$F(\vec{r}_1(i) + \vec{d}(i)) = \vec{r}_2(p_i) + \vec{t}$$

where:

- $\vec{r}_1(i)$ is the Cartesian coordinates of the i -th atom in the parent superstructure. The parent superstructure can be constructed using `libcasm.xtal.make_superstructure`:

```
import libcasm.xtal as xtal  
parent_superstructure = xtal.make_superstructure(  
    T * N, parent_structure)
```

where T , and N are from a [LatticeMapping](#). Then the i -th atom coordinate, $\vec{r}_1(i)$, is equal to:

```
parent_superstructure.atom_coordinate_cart()[:, i]
```

- $\vec{r}_2(i)$ is the Cartesian coordinates of i -th atom in the child structure.
- F is the parent-to-child deformation gradient tensor from a [LatticeMapping](#).
- p_i is a permutation vector, specifying which atom in the child structure (p_i) is mapped to the i -th site of the parent superstructure. Values of p_i greater than the number of atoms in the child structure indicate inferred vacancies in mappings to [Prim](#) with vacancies allowed.
- \vec{t} is a translation vector, in Cartesian coordinates, usually chosen so the average displacement is zero.
- $\vec{d}(i)$: **The Cartesian displacement associated with the atom**
at the i -th site in the parent superstructure.

Import and Mapping

The screenshot shows a Jupyter Notebook interface with two tabs open: 'SiGe_OCC_PART1.V2.ipynb' and 'Terminal 2'. The left sidebar contains a tree view of the notebook's structure, with the 'Import and Mapping' section expanded. The main area displays the content of the 'Import and Mapping' section.

Import and Mapping

Overview

The import and mapping process is essentially the reverse of the calculation setup process:

```
VASP output files (vasprun.xml)
-> ase.Atoms
-> libcasm.xtal.Structure
-> libcasm.configuration.Configuration
```

However, while the Configuration -> Structure conversion is deterministic, the relaxed Structure -> Configuration mapping is not.

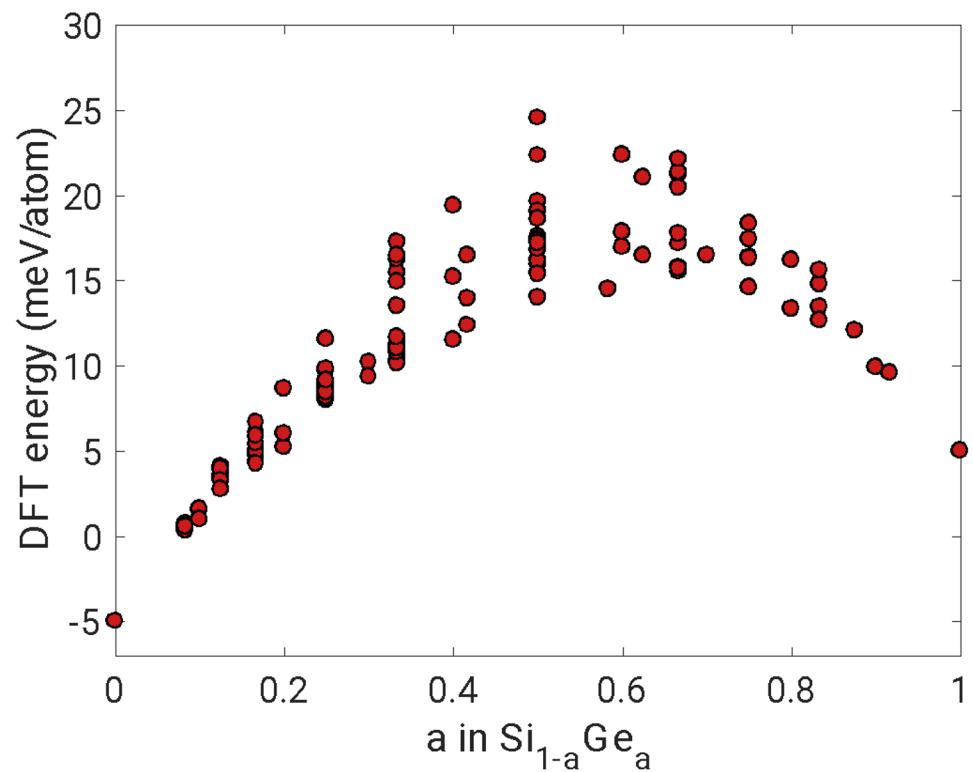
When performing mappings, the term "parent structure" is used for ideal superstructures of the prim (the Configuration being mapped to), and "child structure" is used for the (possibly relaxed) structure being mapped.

CASM provides several methods to systematically check and score possible mappings:

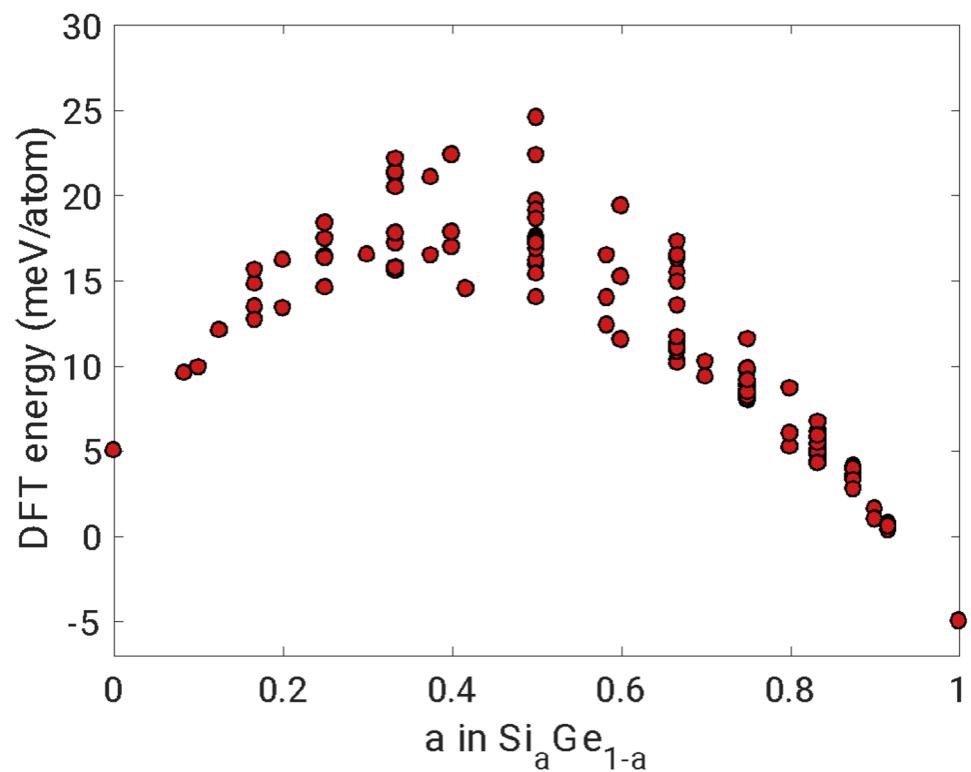
- **map_structures:**
 - A very general structure mapping method.
 - Proposes and checks [StructureMapping](#) of parent structures to a child structure for a range of parent supercell volumes and lattice vector reorientations, allowing combinations of rigid rotation, translation, lattice strain, and atom displacement.
 - Roughly, the approach is to first propose and check lattice mappings, and then propose and check atom mappings.
 - Mappings are scored using a weighted sum of lattice strain cost and atomic displacement cost metrics.
- **map_lattices:**
 - A lattice mapping method for when the parent superstructure lattice is not known.
 - Proposes and checks [LatticeMapping](#) from parent superstructures to the child structure considering reorientations of the parent lattice vectors.
 - Scores mappings using a lattice strain cost metric.

Simple 2 8 1 Python 3 (ipykernel) | Idle Mode: Command ✓ Ln 1, Col 1 SiGe_occ_part1.v2.ipynb 0 🔍

Composition Axes

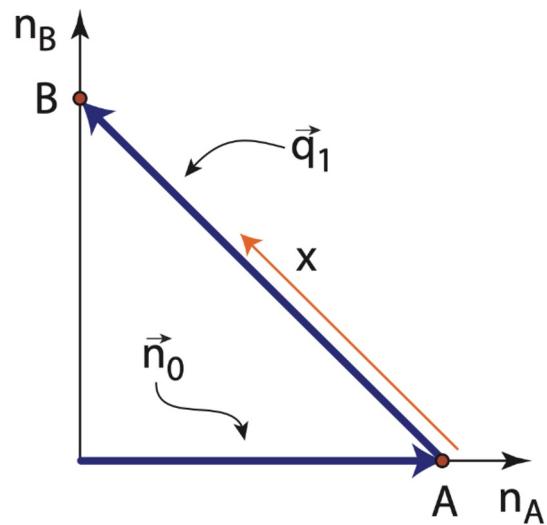


Pure Si as a=0 reference

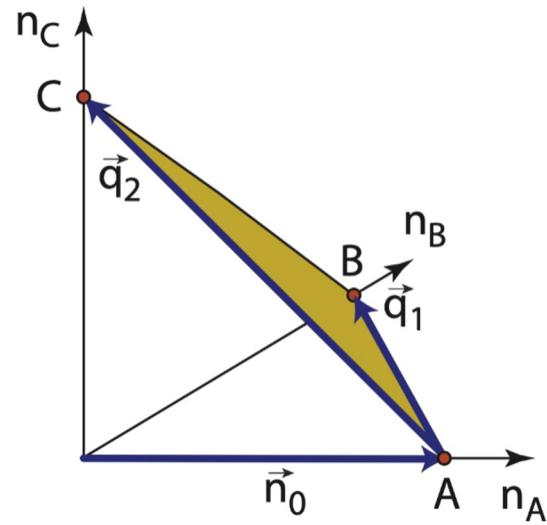


Pure Ge as a=0 reference

Composition Axes



Binary



Ternary

Selecting composition Axes

```
# select axes with <key>, unset with None
project.chemical_composition_axes.set_current_axes(1)

# print possible axes and formulas for current choice
print(project.chemical_composition_axes)

# commit current choice
project.chemical_composition_axes.commit()
```

KEY	ORIGIN	a	GENERAL FORMULA
0	Ge(2)	SiGe	Si(a)Ge(2-a)
1	Si(2)	SiGe	Si(2-a)Ge(a)

Currently selected composition axes: 1

Parametric composition:
comp(a) = -0.5*(comp_n(Si) - 2) + 0.5*comp_n(Ge)

Composition:
comp_n(Si) = 2 - 1*comp(a)
comp_n(Ge) = 1*comp(a)

Parametric chemical potentials:
param_chem_pot(a) = -chem_pot(Si) + chem_pot(Ge)

Composition Axes

The screenshot shows a Jupyter Notebook interface with a sidebar on the left and a main content area on the right.

Sidebar (Left):

- File Edit View Run Kernel Tabs Settings Help
- SIGE_OCC_PART1.V2.IPYNB
- Calculation directory structure
- Calculation status file
- Calculation settings
- Calculation settings for ASE + VASP
- Setup calculations
- Select configurations
- Generate VASP input files
- Check input files
- Checking calculation status
- Generate Slurm job submission scripts
- Run calculations
- Select configurations
- Submit jobs: casm-calc submit
- Check job status
- Make CASM structures with properties
- Import and Mapping
- Overview
- Collect calculated structures
- Map structures to configurations
- Convert mapped configurations with properties
- Plot lattice and strain mapping costs
- Composition axes
- Introduction
- Print standard parametric composition axes
- Select default parametric composition axes
- Formation energy
- Get properties
- Plot energy per unitcell vs composition
- Calculate formation energy
- Plot formation energy per unitcell vs composition

Main Content Area (Right):

Terminal 2: SIGe_occ_part1.v2.ipynb

Notebook: Python 3 (ipykernel)

Composition axes

Introduction

In a crystal with a fixed number of sites, the number of species occupying the same sublattice are not independent. In general, a crystal occupied by s component species will have $k < s$ independent compositions. CASM converts between compositions expressed as number of species per unit cell and compositions in terms of independent "parametric composition axes" using:

$$\mathbf{n} = \mathbf{n}_0 + Q\boldsymbol{\varpi}$$
$$\boldsymbol{\varpi} = R^T(\mathbf{n} - \mathbf{n}_0)$$

where:

- \mathbf{n} : Vector of shape=(s ,), the number of each component species per unit cell (`mol_composition`).
- $\boldsymbol{\varpi}$: Vector of shape=(k ,), The composition along each composition axis when referenced to the origin composition (`param_composition`).
- \mathbf{n}_0 : Vector of shape=(s ,), The origin in composition space, as the number of each component species per unit cell.
- Q : Matrix of shape=(s , k), with columns representing the change in composition per unit cell going one unit distance along each independent composition axis.
- R : Matrix of shape=(s , k), such that $R^T Q = Q^T R = I$.

The "parametric composition axes" are the columns of Q . Due to preservation of the number of sites per unit cell, $\sum_i Q_{ij} = 0$. If vacancies are allowed, they are included as a component species.

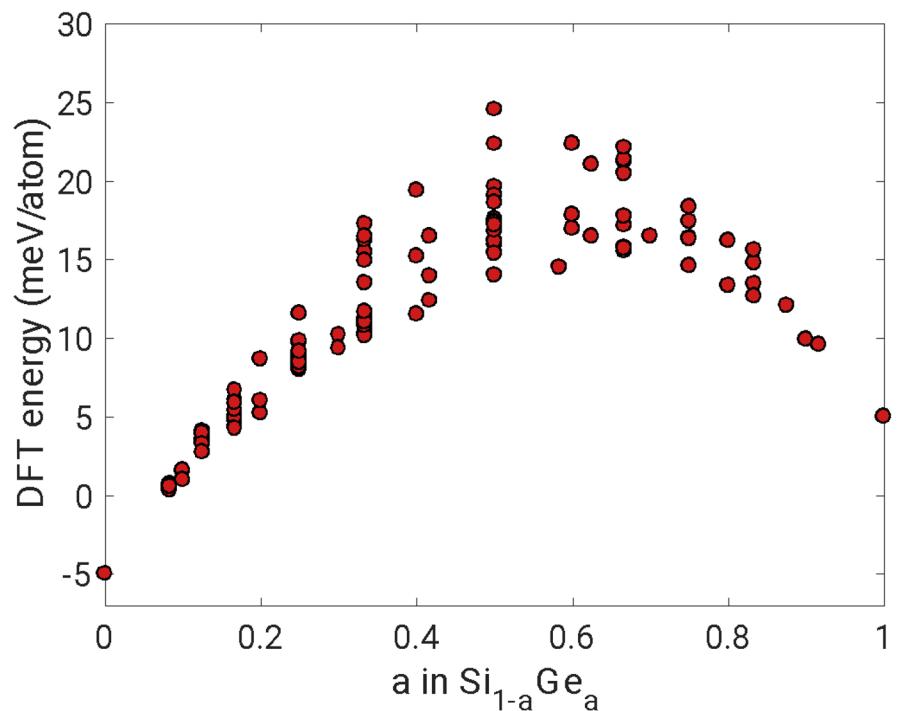
Print standard parametric composition axes

When a CASM project is initialized, a set of standard choices for the parametric axes are determined and stored in the `[Project.chemical_composition_axes]` attribute. Printing the `chemical_composition_axes` results in a table summarizing the possible choices:

```
[42]: # print possible axes
```

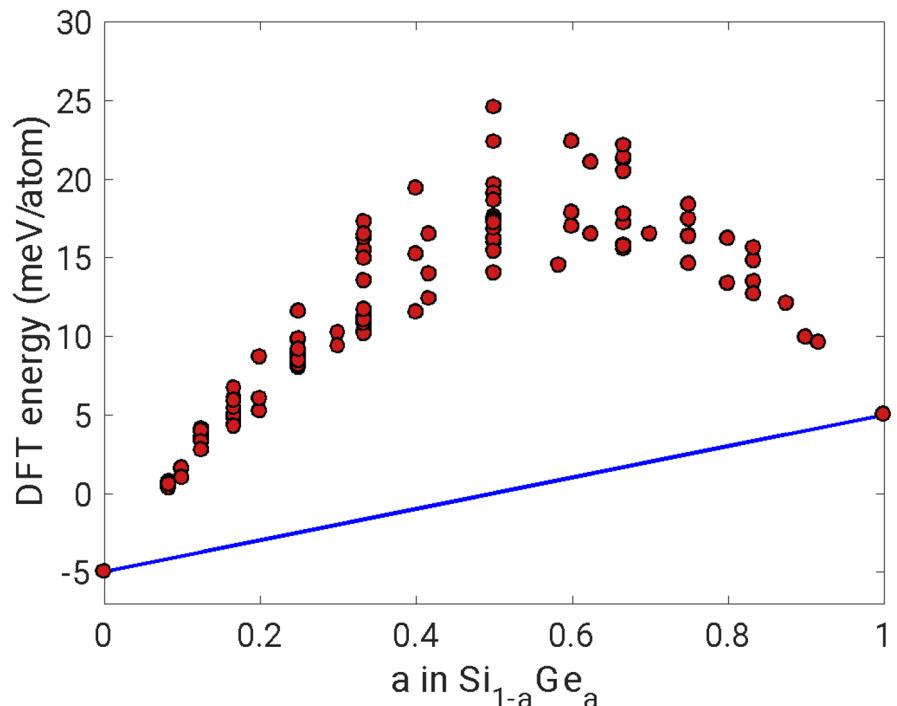
Mode: Command Ln 5, Col 213 SIGe_occ_part1.v2.ipynb 0

Setting an energy reference

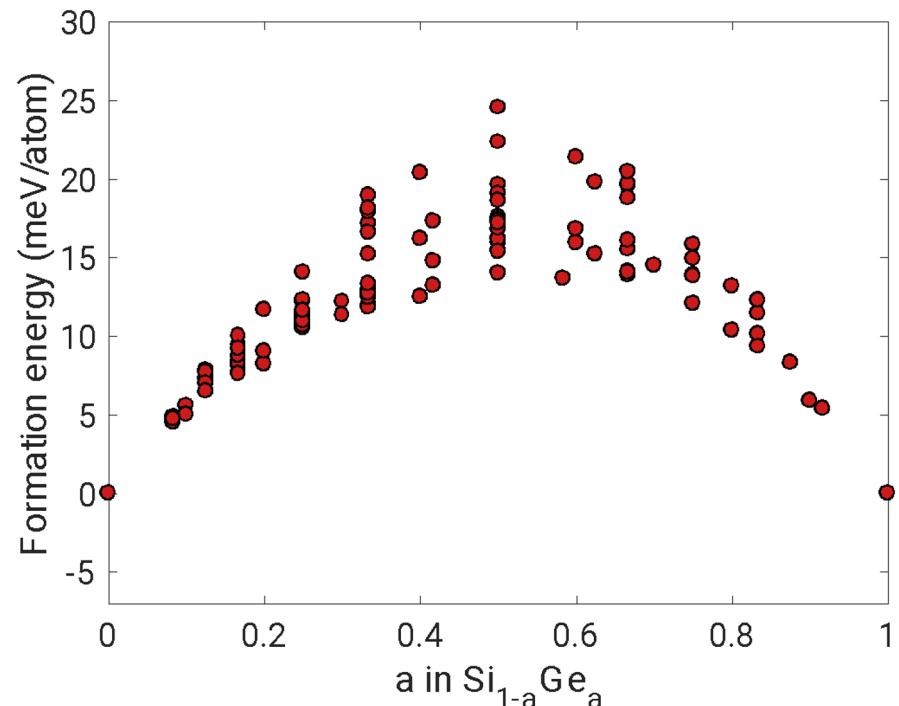


Chemical energy can only be measured relative to reference compounds

Setting an energy reference

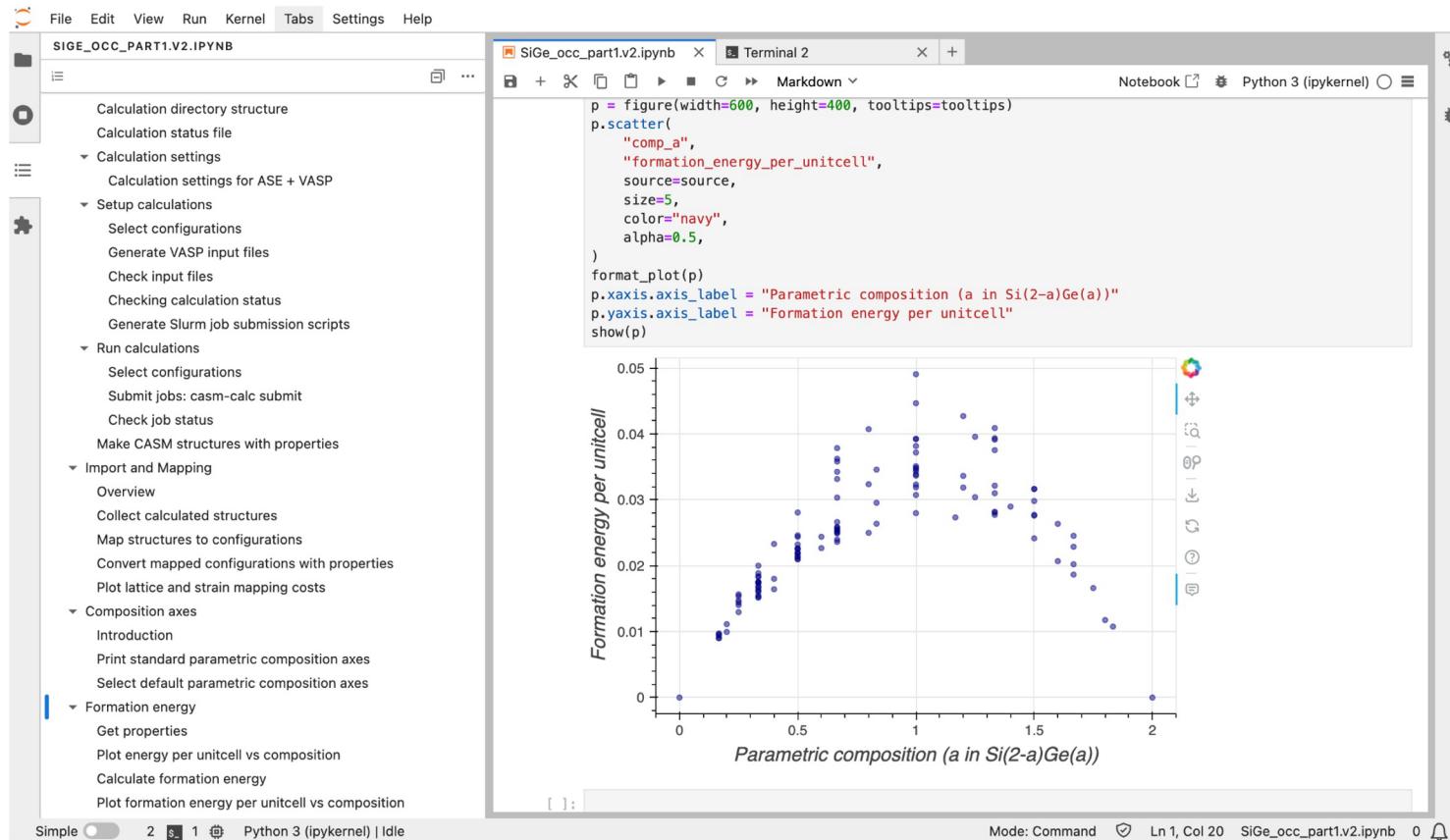


DFT energy



Formation energy

Formation energy



Cluster expansion construction & fitting

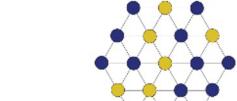
CASM Project

Parent Crystal Structure

Lattice vectors, $\vec{L} = (\vec{l}_1, \vec{l}_2, \vec{l}_3)$

Basis site coordinates, $\vec{B} = (\vec{b}_1, \vec{b}_2, \dots)$

Degrees of Freedom (DoF)



Occupation, \vec{s}

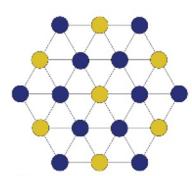
Displacement, \vec{d}

Magnetic spin, \vec{m}

Strain, \vec{E}

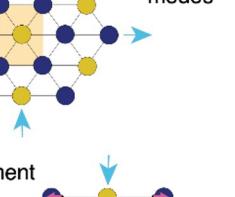
Derivative Structure Enumeration

Discrete occupation ordering

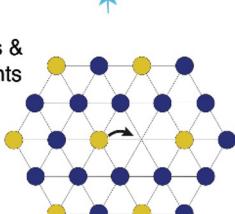


+ Strain modes

+ Displacement modes

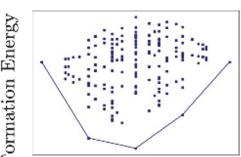


Migration event types & environments



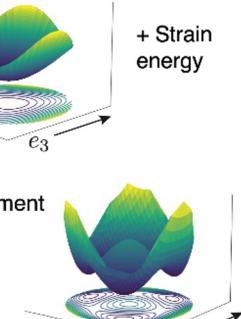
First Principles Calculations

Discrete occupation ordering energy



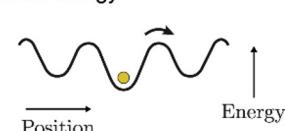
+ Strain energy

+ Displacement energy



+ Relaxation strain, displacement, magnetic spin, orientation, ...

Migration energy



Cluster Expansion Effective Hamiltonian

$$E(\vec{s}, \vec{d}, \vec{m}, \vec{q}, \vec{E}) = \sum_i m_i V_i \Gamma_i(\vec{s}, \vec{d}, \vec{m}, \vec{q}, \vec{E})$$

Symmetry-adapted basis functions, Γ_i
Expansion coefficients, V_i
Cluster multiplicity, m_i

Monte Carlo Calculations

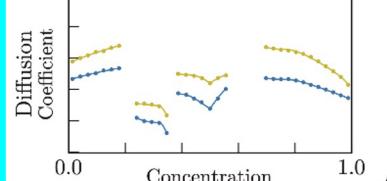
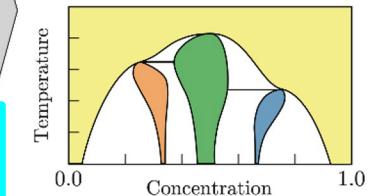
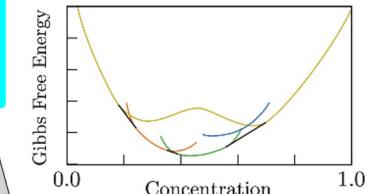
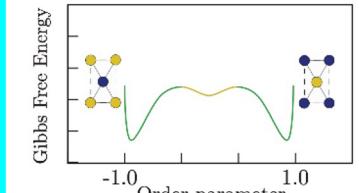
Metropolis, N-fold way, Hamiltonian

Kinetic Monte Carlo Calculations

Local-cluster Expansion of Diffusion Barriers

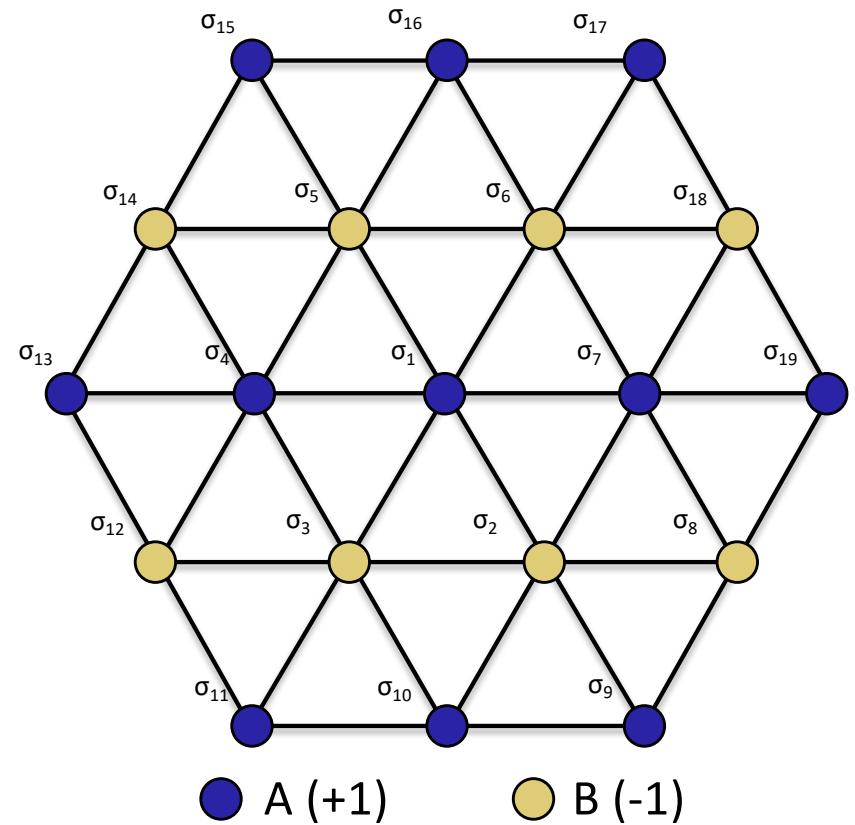
$$\Delta E_{\delta}^{KRA}(\vec{s}) = \sum_i m_i V_i \Gamma_i(\vec{s})$$

Finite Temperature Properties



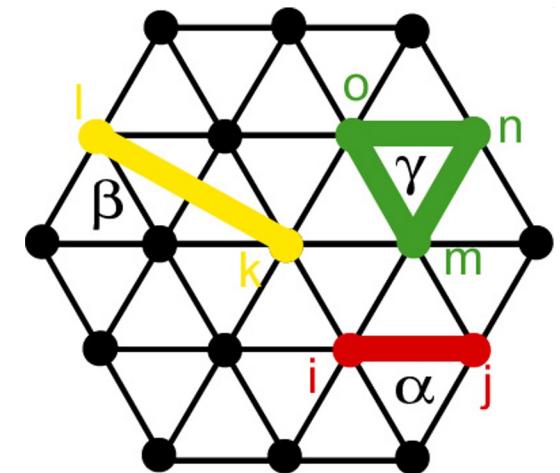
Cluster expansion

A particular ordering is tracked
with occupation variables σ_i
assigned to each site i



Cluster expansion

$$E(\vec{\sigma}) = V_o + \sum_i V_i \sigma_i + \sum_{i,j} V_{i,j} \sigma_i \sigma_j + \sum_{i,j,k} V_{i,j,k} \sigma_i \sigma_j \sigma_k + \dots$$



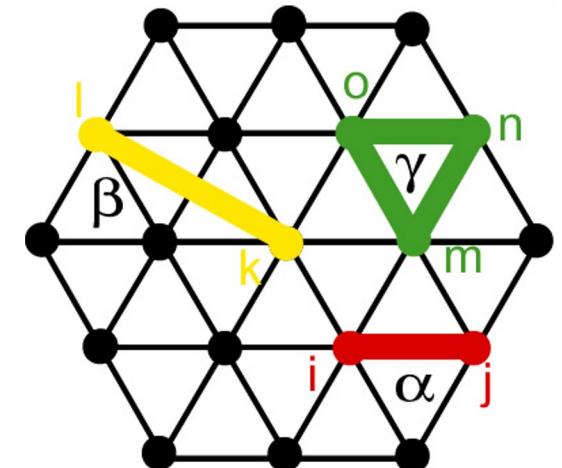
Cluster expansion

$$E(\vec{\sigma}) = V_o + \sum_i V_i \sigma_i + \sum_{i,j} V_{i,j} \sigma_i \sigma_j + \sum_{i,j,k} V_{i,j,k} \sigma_i \sigma_j \sigma_k + \dots$$

Can be written more compactly as

$$E(\vec{\sigma}) = V_0 + \sum_{\alpha} V_{\alpha} \phi_{\alpha}(\vec{\sigma})$$

$$\phi_{\alpha}(\vec{\sigma}) = \prod_{i \in \alpha} \sigma_i$$



Cluster expansion

$$E(\vec{\sigma}) = V_o + \sum_i V_i \sigma_i + \sum_{i,j} V_{i,j} \sigma_i \sigma_j + \sum_{i,j,k} V_{i,j,k} \sigma_i \sigma_j \sigma_k + \dots$$

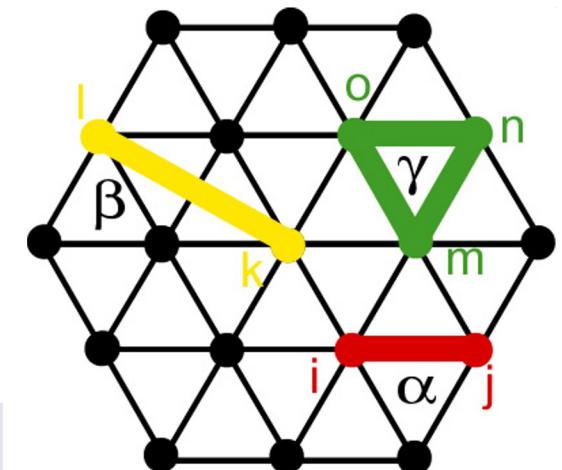
Can be written more compactly as

$$E(\vec{\sigma}) = V_0 + \sum_{\alpha} V_{\alpha} \phi_{\alpha}(\vec{\sigma})$$

$$\phi_{\alpha}(\vec{\sigma}) = \prod_{i \in \alpha} \sigma_i$$

Two clusters α and δ that are equivalent by symmetry have the same expansion coefficient

$$V_{\alpha} = V_{\delta}$$



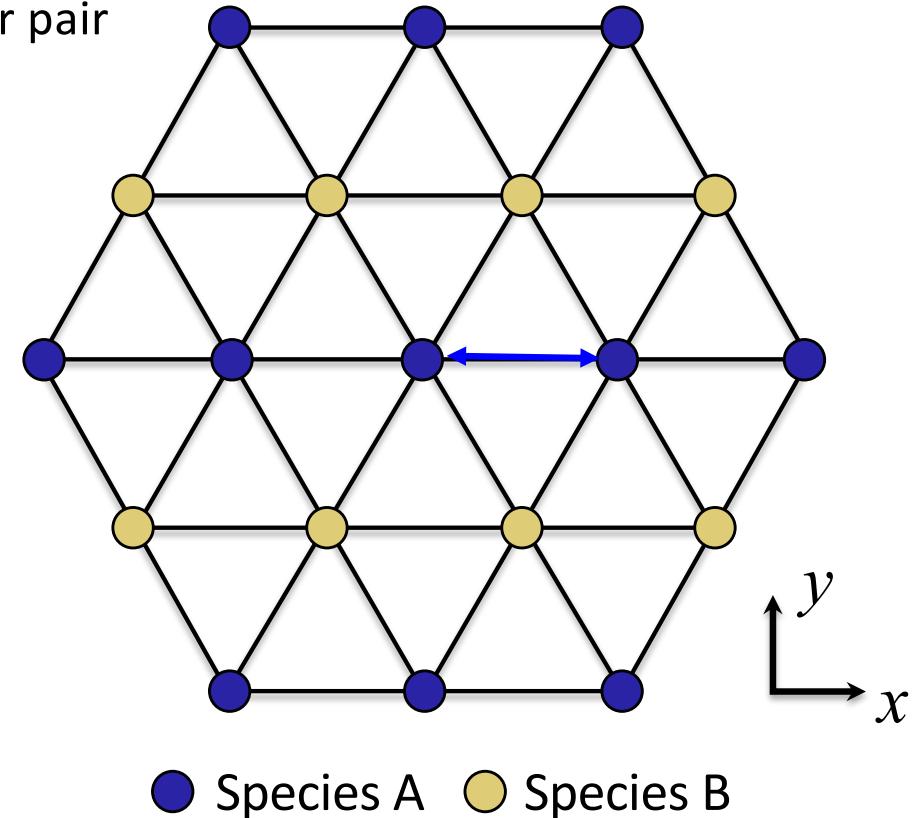
Cluster expansion

Example of the nearest neighbor pair

$$E(\vec{\sigma}) = V_0 + \sum_{\alpha} V_{\alpha} \phi_{\alpha}(\vec{\sigma})$$

$$\phi_{\alpha}(\vec{\sigma}) = \prod_{i \in \alpha} \sigma_i$$

All these pair clusters will have the same expansion coefficient V



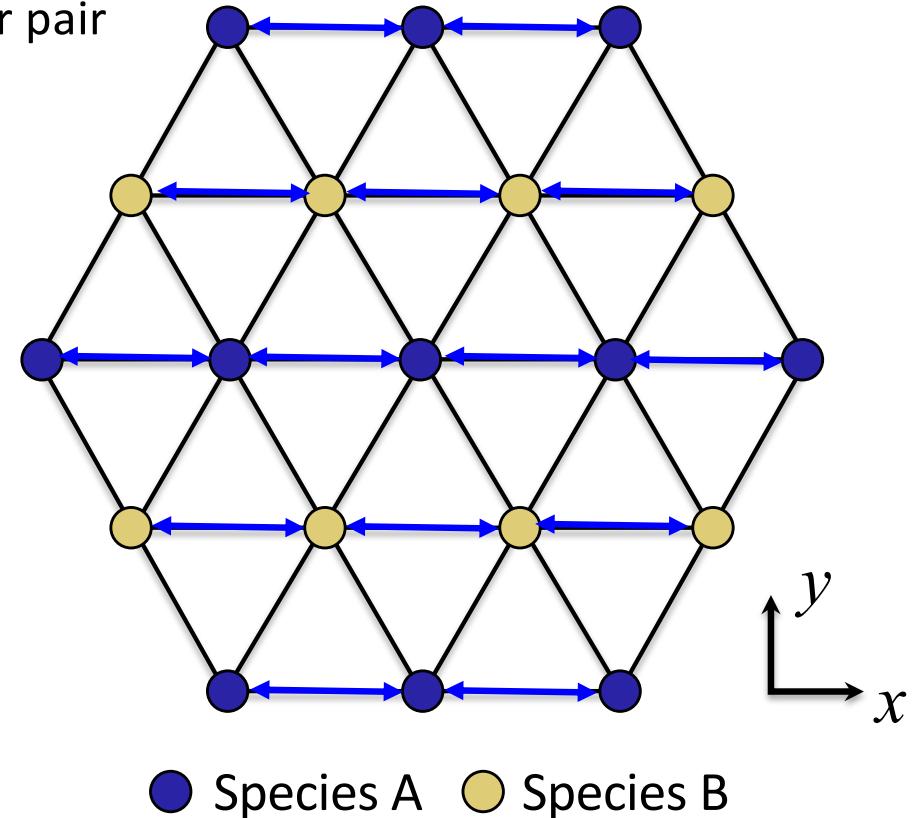
Cluster expansion

Example of the nearest neighbor pair

$$E(\vec{\sigma}) = V_0 + \sum_{\alpha} V_{\alpha} \phi_{\alpha}(\vec{\sigma})$$

$$\phi_{\alpha}(\vec{\sigma}) = \prod_{i \in \alpha} \sigma_i$$

All these pair clusters will have the same expansion coefficient V



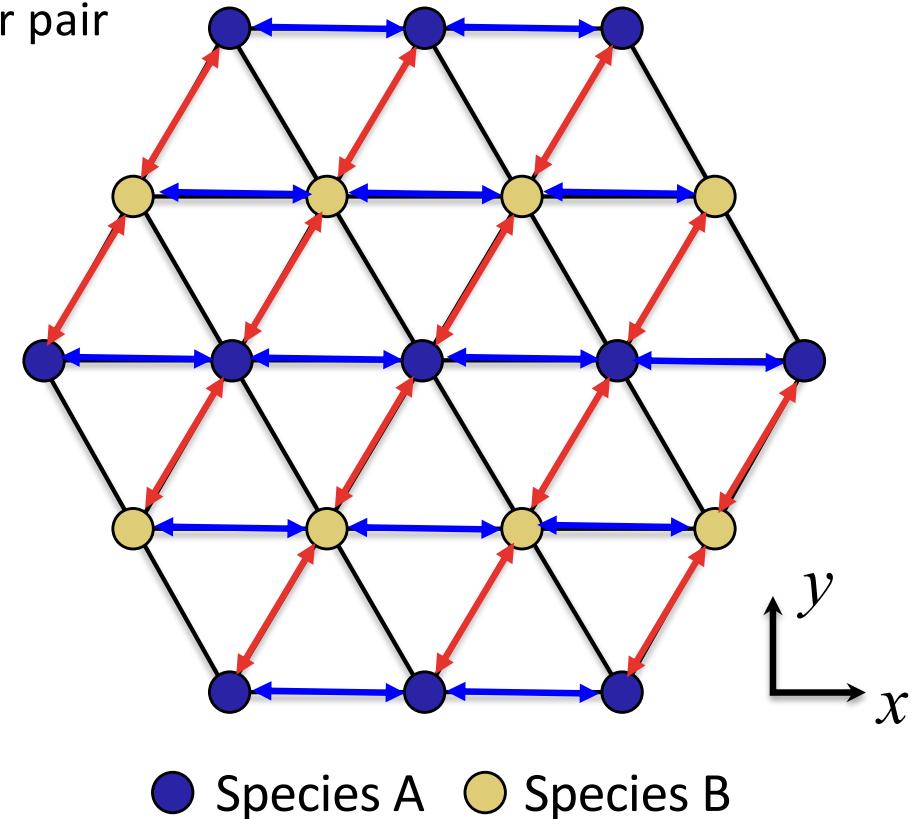
Cluster expansion

Example of the nearest neighbor pair

$$E(\vec{\sigma}) = V_0 + \sum_{\alpha} V_{\alpha} \phi_{\alpha}(\vec{\sigma})$$

$$\phi_{\alpha}(\vec{\sigma}) = \prod_{i \in \alpha} \sigma_i$$

All these pair clusters will have the same expansion coefficient V



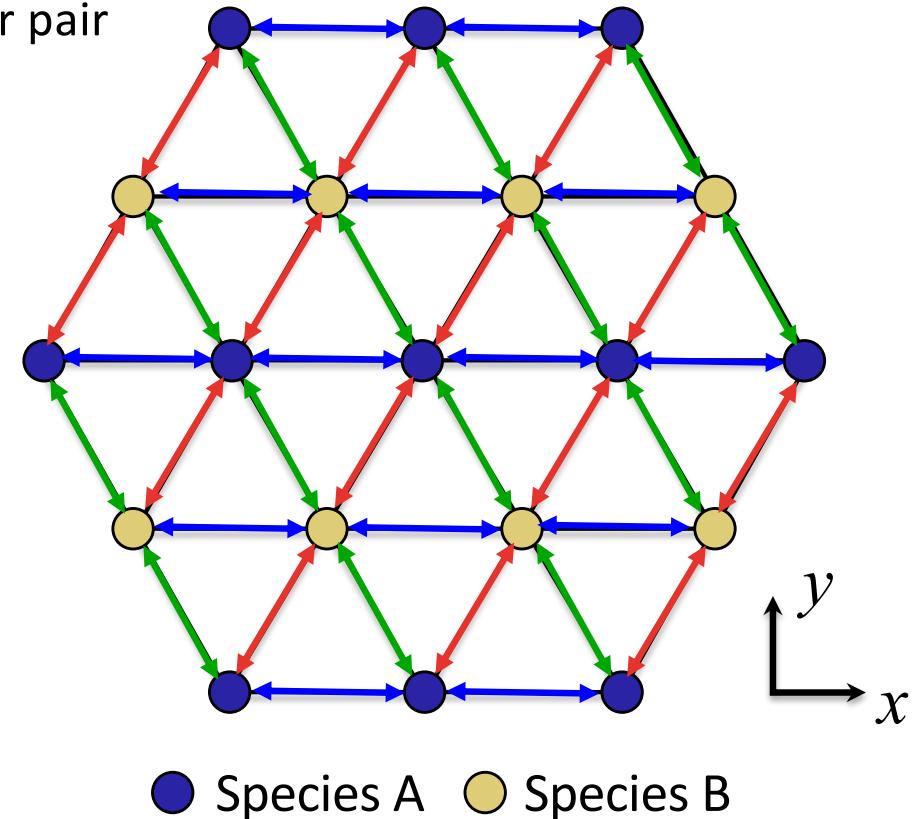
Cluster expansion

Example of the nearest neighbor pair

$$E(\vec{\sigma}) = V_0 + \sum_{\alpha} V_{\alpha} \phi_{\alpha}(\vec{\sigma})$$

$$\phi_{\alpha}(\vec{\sigma}) = \prod_{i \in \alpha} \sigma_i$$

All these pair clusters will have the same expansion coefficient V



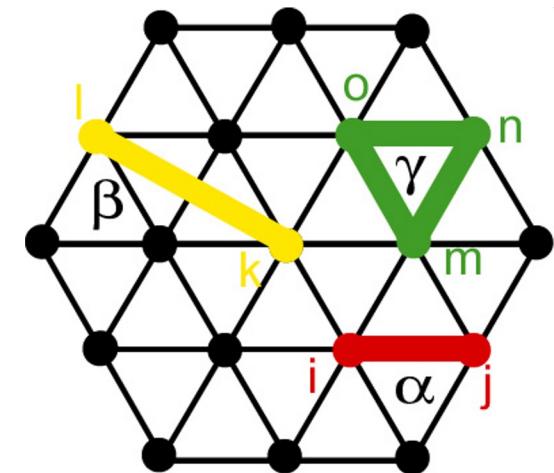
Symmetry equivalence of clusters lets us simplify the cluster expansion

$$E(\vec{\sigma}) = V_0 + \sum_{\alpha} V_{\alpha} \phi_{\alpha}(\vec{\sigma}) \quad \phi_{\alpha}(\vec{\sigma}) = \prod_{i \in \alpha} \sigma_i$$

Using symmetry

$$e(\vec{\sigma}) = v_0 + \sum_{\alpha} v_{\alpha} \Gamma_{\alpha}(\vec{\sigma})$$

$$\Gamma_{\alpha} = \frac{1}{Nm_{\alpha}} \sum_{\beta \in \Omega_{\alpha}} \phi_{\beta}(\vec{\sigma})$$

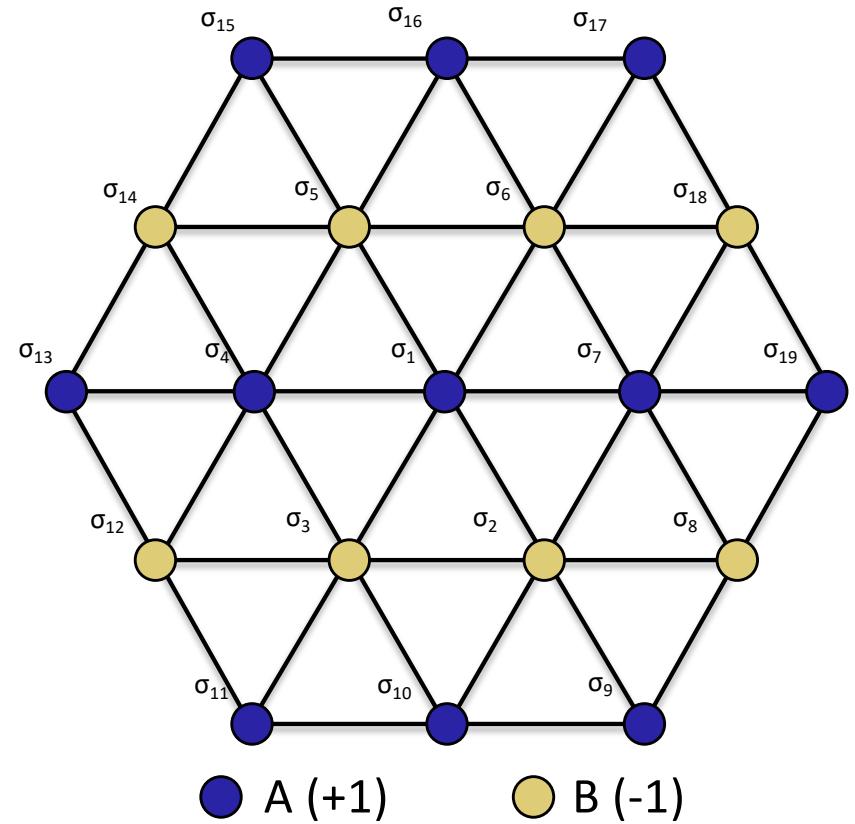


Cluster expansion

$$\Gamma_\alpha = \frac{1}{Nm_\alpha} \sum_{\beta \in \Omega_\alpha} \phi_\beta(\vec{\sigma})$$

$\sigma_i = +1$ if species A

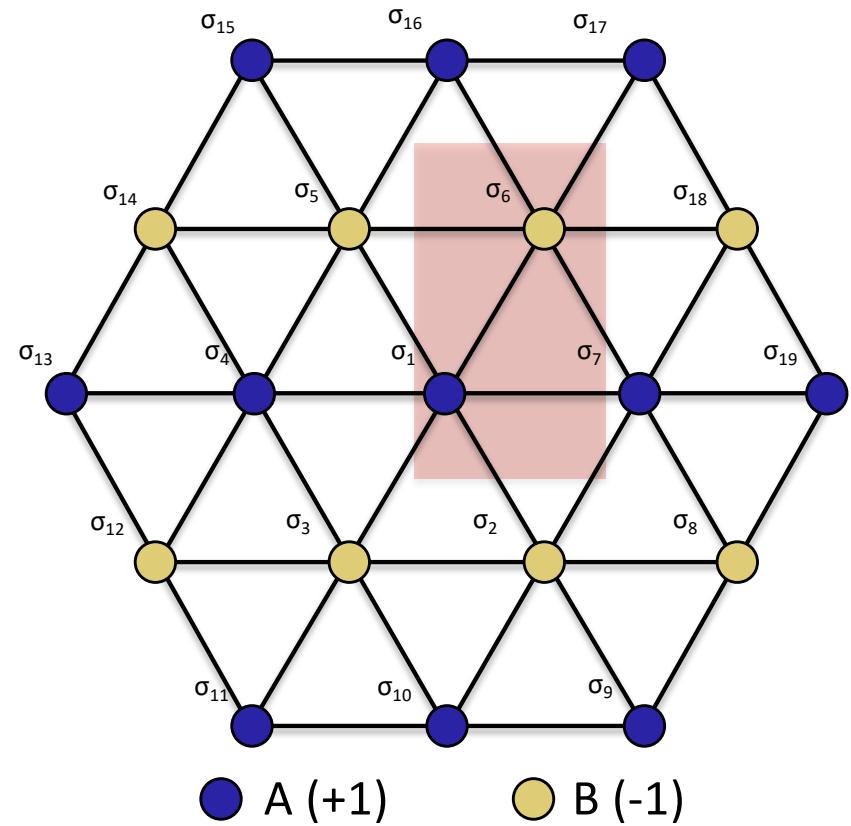
$\sigma_i = -1$ if species B



Cluster expansion

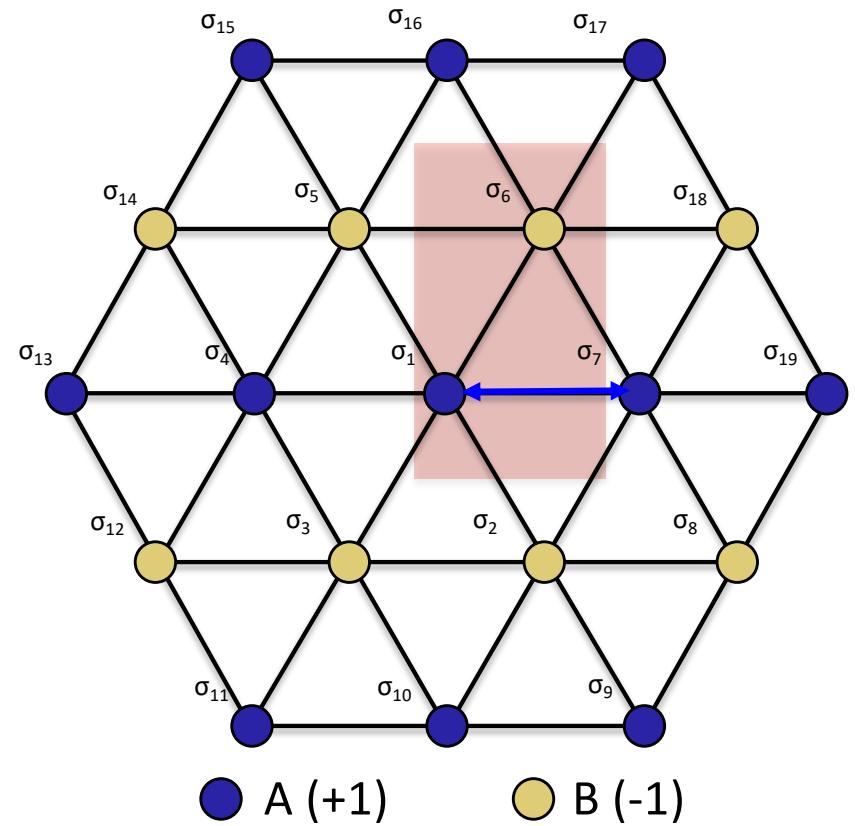
Correlation function
for the point cluster

Average site occupation is
 $\Gamma_1 = (\sigma_1 + \sigma_6)/2 = 0$



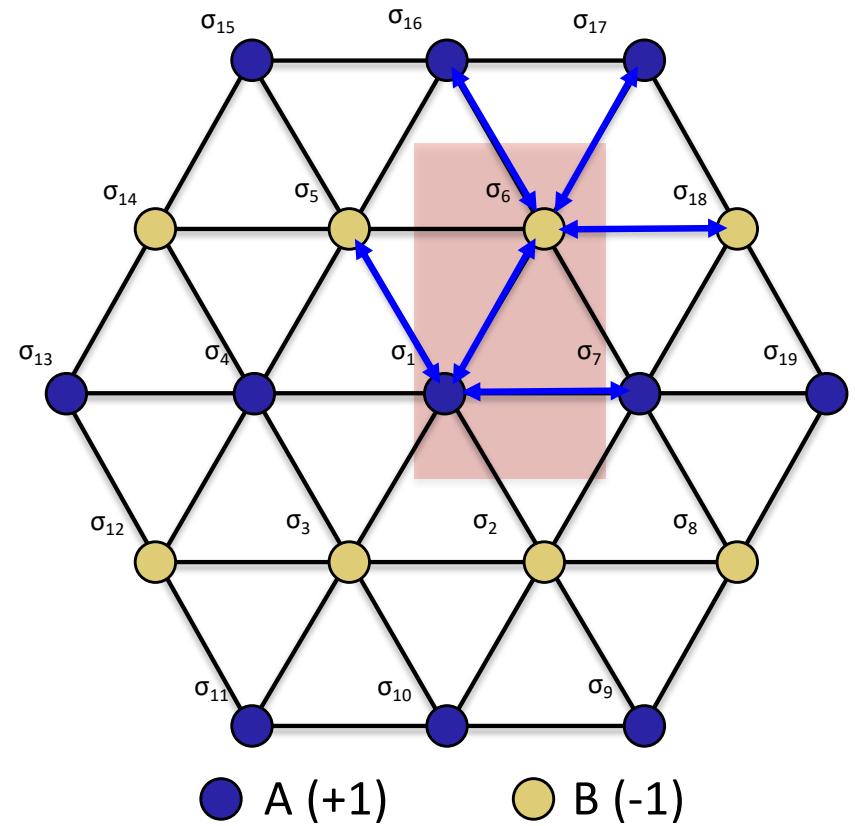
Cluster expansion

Correlation
function for the
nearest neighbor
pair cluster



Cluster expansion

Correlation
function for the
nearest neighbor
pair cluster

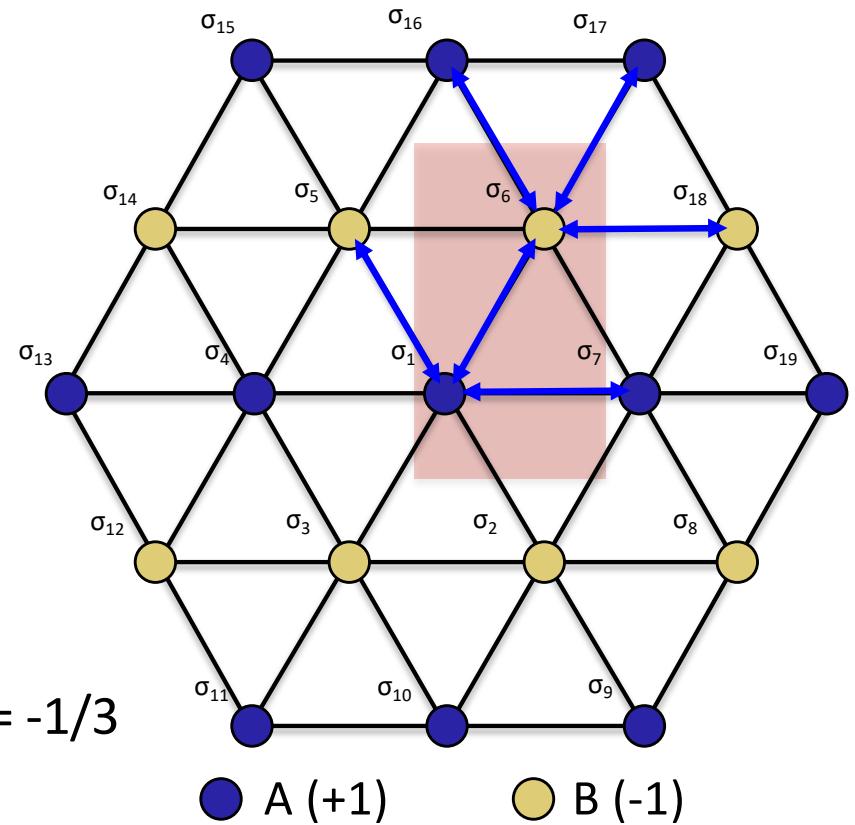


Cluster expansion

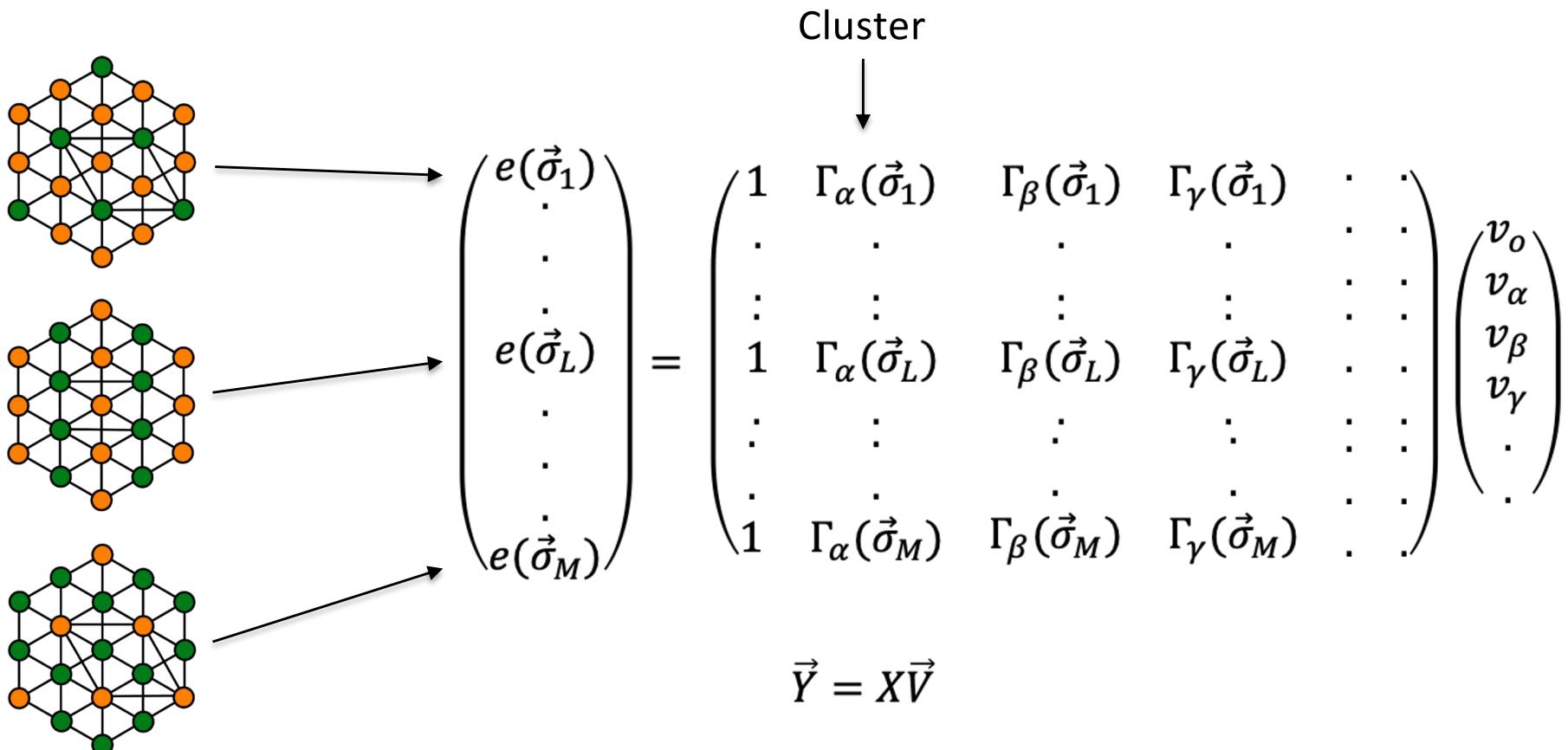
Correlation
function for the
nearest neighbor
pair cluster

Average NN pair correlation is

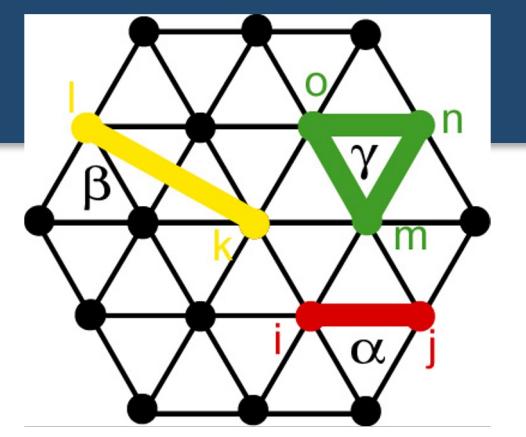
$$\Gamma_2 = (\sigma_1\sigma_7 + \sigma_1\sigma_6 + \sigma_1\sigma_5 + \sigma_6\sigma_{18} + \sigma_6\sigma_{17} + \sigma_6\sigma_{16})/6 = -1/3$$



Fitting Coefficients



Fitting Coefficients

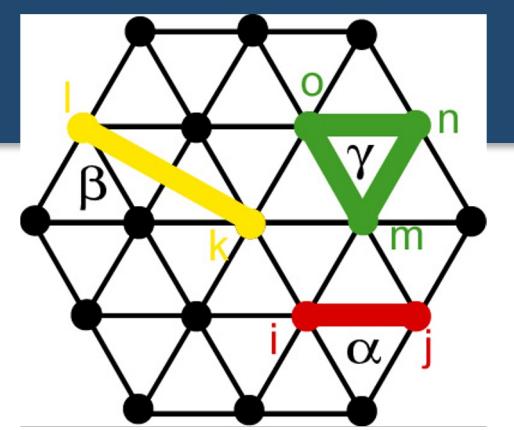
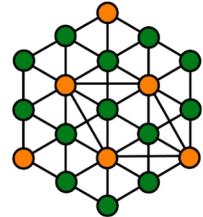
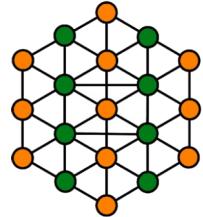
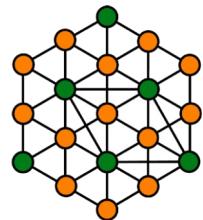


$$\frac{E(\vec{\sigma})}{N} = \frac{V_0}{N} + \sum_{\alpha} m_{\alpha} V_{\alpha} \left(\frac{1}{Nm_{\alpha}} \sum_{\delta \in \Omega_{\alpha}} \phi_{\delta}(\vec{\sigma}) \right)$$

↓ $\Gamma_{\alpha}(\vec{\sigma})$ Correlation function

$$e(\vec{\sigma}) = v_0 + \sum_{\alpha} v_{\alpha} \Gamma_{\alpha}(\vec{\sigma})$$

Fitting Coefficients

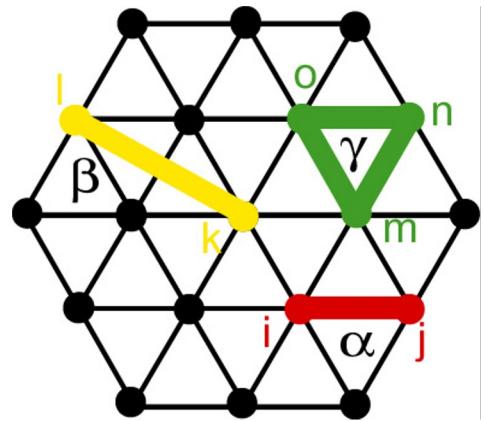


Fitting Coefficients

$$\begin{array}{ccc}
 \text{Cluster} & \downarrow & \\
 \begin{array}{c} \text{Diagram of a cluster with nodes labeled } l, o, n, \gamma, m, i, j, \alpha, \beta, k \\ \text{and edges connecting them.} \end{array} & & \begin{array}{c} \text{Three hexagonal clusters with nodes colored orange and green.} \end{array} \\
 \xrightarrow{\hspace{1cm}} & & \xrightarrow{\hspace{1cm}} \\
 \begin{array}{c} e(\vec{\sigma}_1) \\ \vdots \\ e(\vec{\sigma}_L) \\ \vdots \\ e(\vec{\sigma}_M) \end{array} & = & \begin{pmatrix} 1 & \Gamma_\alpha(\vec{\sigma}_1) & \Gamma_\beta(\vec{\sigma}_1) & \Gamma_\gamma(\vec{\sigma}_1) & \cdots & \\ \vdots & \vdots & \vdots & \vdots & \ddots & \\ 1 & \Gamma_\alpha(\vec{\sigma}_L) & \Gamma_\beta(\vec{\sigma}_L) & \Gamma_\gamma(\vec{\sigma}_L) & \cdots & \\ \vdots & \vdots & \vdots & \vdots & \ddots & \\ 1 & \Gamma_\alpha(\vec{\sigma}_M) & \Gamma_\beta(\vec{\sigma}_M) & \Gamma_\gamma(\vec{\sigma}_M) & \cdots & \end{pmatrix} \begin{pmatrix} v_o \\ v_\alpha \\ v_\beta \\ v_\gamma \\ \vdots \end{pmatrix}
 \end{array}$$

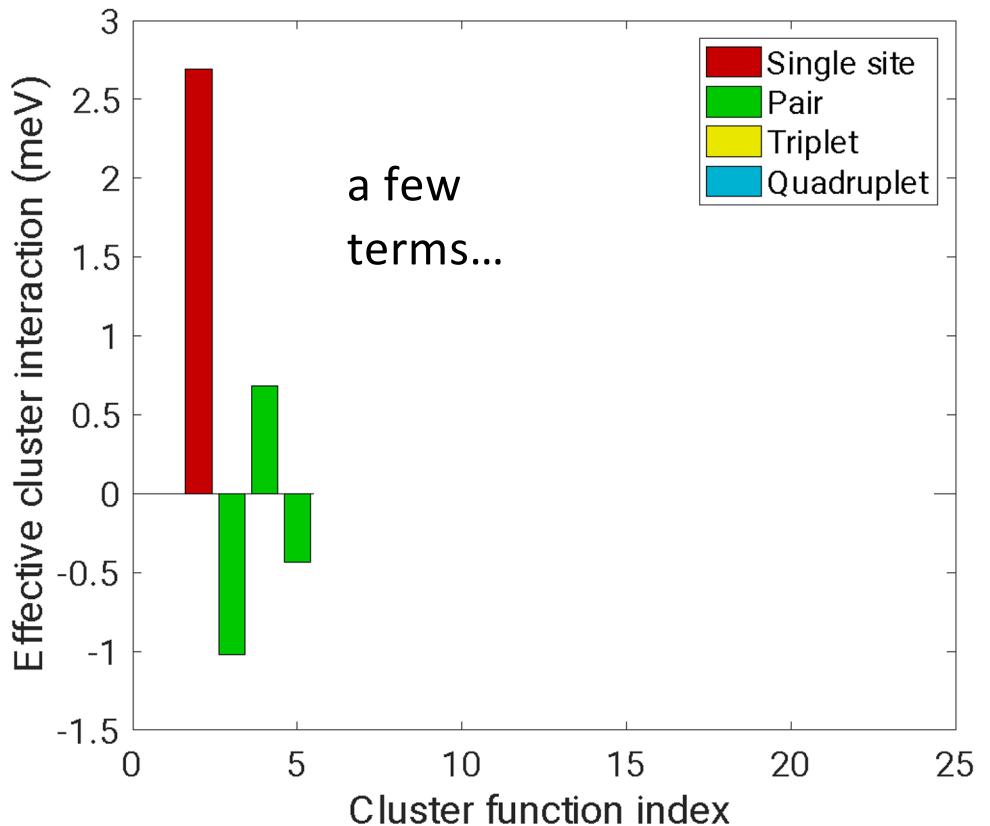
$\vec{Y} = X\vec{V}$

Exercise: Using a varying number of terms

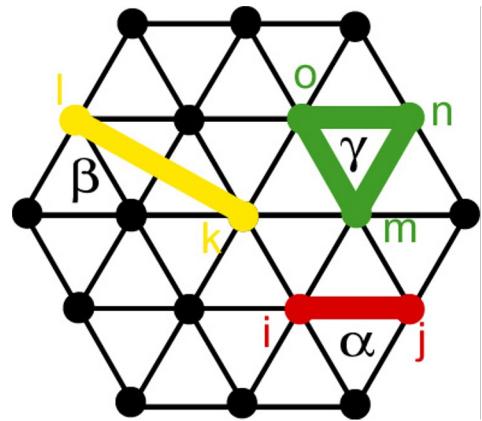


$$\begin{aligned}\Phi_{\alpha} &= \sigma_i \sigma_j \\ \Phi_{\beta} &= \sigma_l \sigma_k \\ \Phi_{\gamma} &= \sigma_m \sigma_n \sigma_o\end{aligned}$$

$$E(\vec{\sigma}) = V_o + \sum_i V_i \sigma_i + \sum_{i,j} V_{i,j} \sigma_i \sigma_j + \sum_{i,j,k} V_{i,j,k} \sigma_i \sigma_j \sigma_k + \dots$$

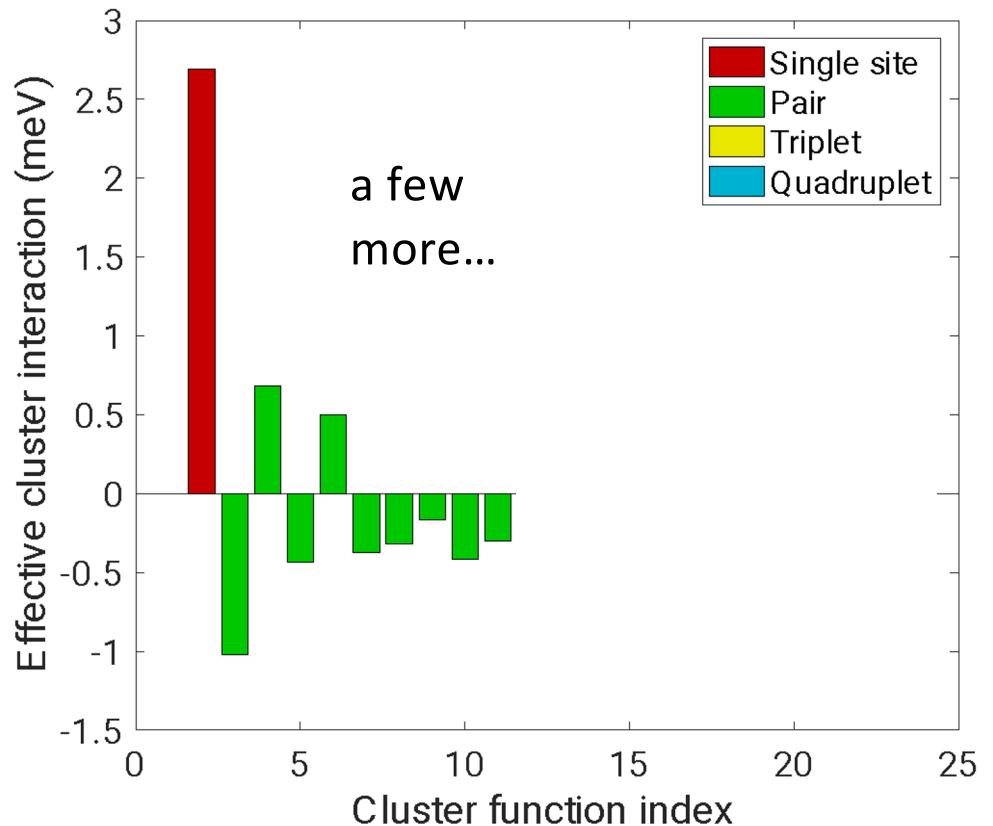


Exercise: Using a varying number of terms

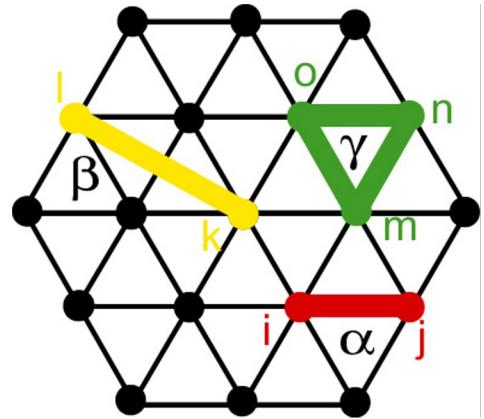


$$\begin{aligned}\Phi_{\alpha} &= \sigma_i \sigma_j \\ \Phi_{\beta} &= \sigma_l \sigma_k \\ \Phi_{\gamma} &= \sigma_m \sigma_n \sigma_o\end{aligned}$$

$$E(\vec{\sigma}) = V_o + \sum_i V_i \sigma_i + \sum_{i,j} V_{i,j} \sigma_i \sigma_j + \sum_{i,j,k} V_{i,j,k} \sigma_i \sigma_j \sigma_k + \dots$$

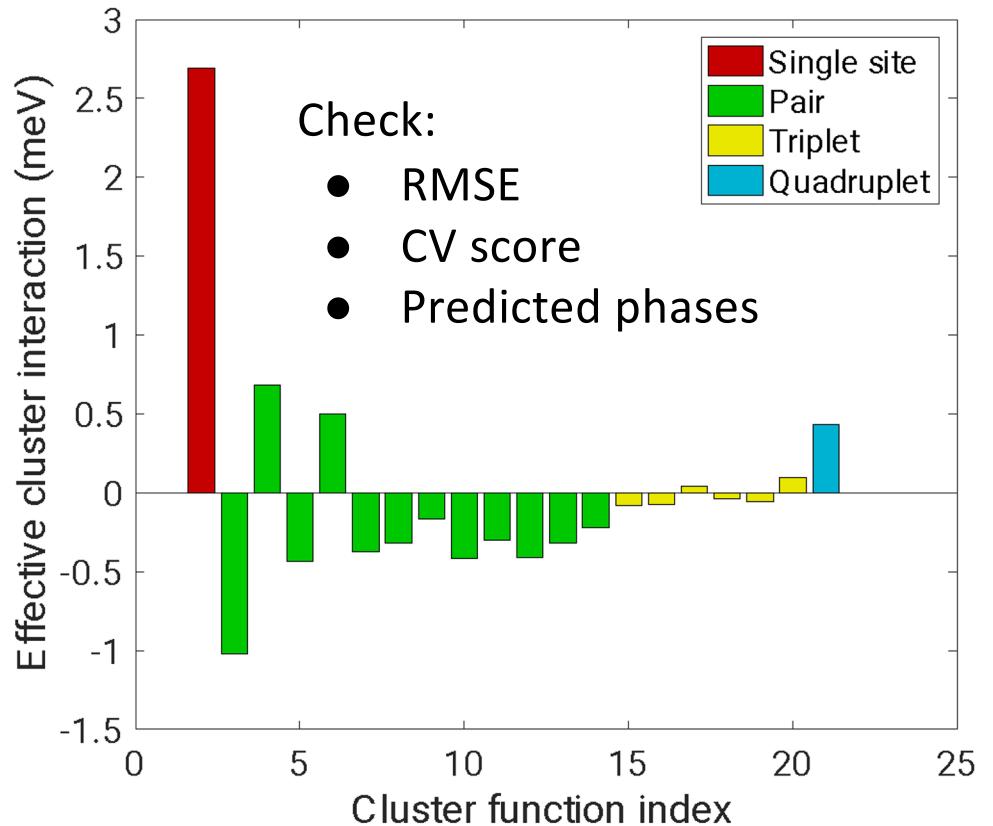


Exercise: Using a varying number of terms

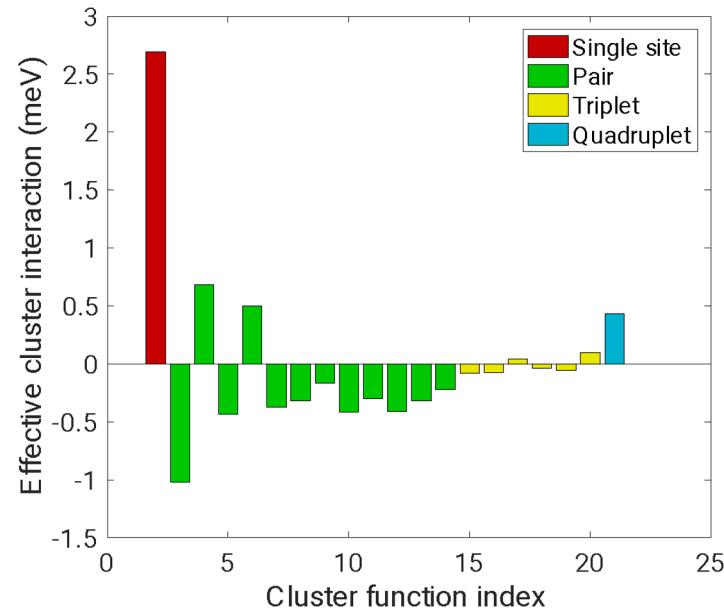


$$\begin{aligned}\Phi_{\alpha} &= \sigma_i \sigma_j \\ \Phi_{\beta} &= \sigma_l \sigma_k \\ \Phi_{\gamma} &= \sigma_m \sigma_n \sigma_o\end{aligned}$$

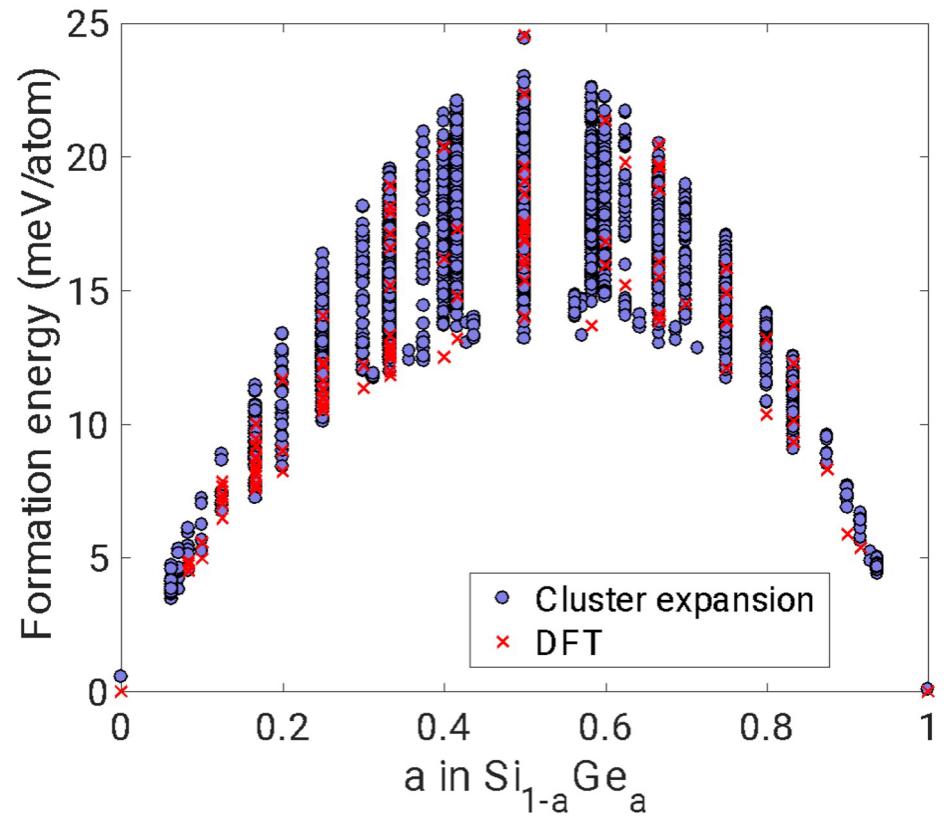
$$E(\vec{\sigma}) = V_o + \sum_i V_i \sigma_i + \sum_{i,j} V_{i,j} \sigma_i \sigma_j + \sum_{i,j,k} V_{i,j,k} \sigma_i \sigma_j \sigma_k + \dots$$



Exercise: Using a varying number of terms



$$E(\vec{\sigma}) = V_o + \sum_i V_i \sigma_i + \sum_{i,j} V_{i,j} \sigma_i \sigma_j + \sum_{i,j,k} V_{i,j,k} \sigma_i \sigma_j \sigma_k + \dots$$



Monte Carlo calculation & Finite Temp. Properties

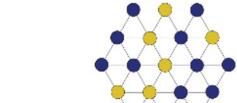
CASM Project

Parent Crystal Structure

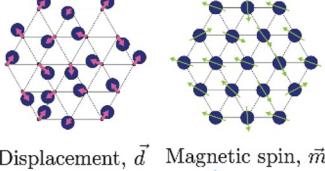
Lattice vectors, $\vec{L} = (\vec{l}_1, \vec{l}_2, \vec{l}_3)$

Basis site coordinates, $\vec{B} = (\vec{b}_1, \vec{b}_2, \dots)$

Degrees of Freedom (DoF)



Occupation, \vec{s}



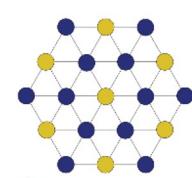
Displacement, \vec{d}

Magnetic spin, \vec{m}

Strain, \vec{E}

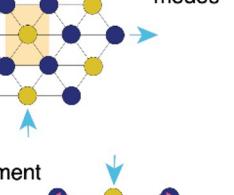
Derivative Structure Enumeration

Discrete occupation ordering

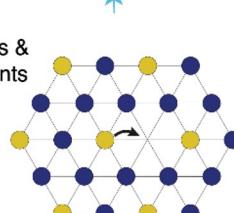


+ Strain modes

+ Displacement modes

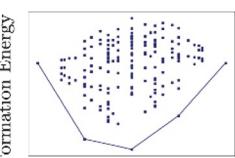


Migration event types & environments



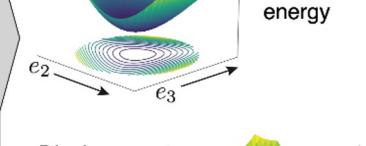
First Principles Calculations

Discrete occupation ordering energy

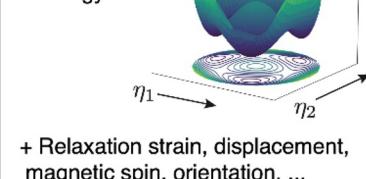


Concentration

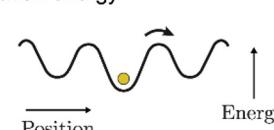
+ Strain energy



+ Displacement energy



Migration energy



Position Energy

Cluster Expansion Effective Hamiltonian

$$E(\vec{s}, \vec{d}, \vec{m}, \vec{q}, \vec{E}) = \sum_i m_i V_i \Gamma_i(\vec{s}, \vec{d}, \vec{m}, \vec{q}, \vec{E})$$

Symmetry-adapted basis functions, Γ_i
Expansion coefficients, V_i
Cluster multiplicity, m_i

Monte Carlo Calculations

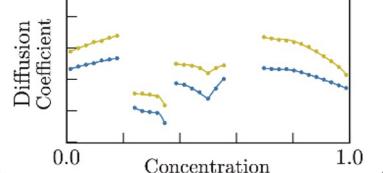
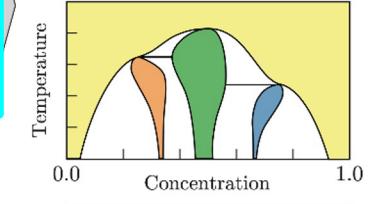
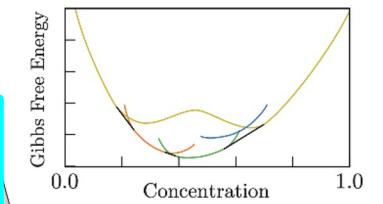
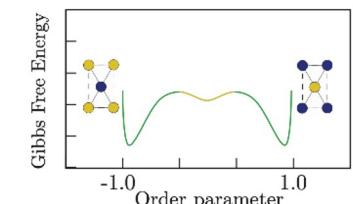
Metropolis, N-fold way, Hamiltonian

Kinetic Monte Carlo Calculations

Local-cluster Expansion of Diffusion Barriers

$$\Delta E_{\delta}^{KRA}(\vec{s}) = \sum_i m_i V_i \Gamma_i(\vec{s})$$

Finite Temperature Properties



Binary alloy, semi-grand canonical ensemble

Partition function: $Z = \sum_{\vec{\sigma}} e^{-\beta\Omega(\vec{\sigma})}$

$$\Omega(\vec{\sigma}) = E(\vec{\sigma}) - \Delta\mu N_B(\vec{\sigma})$$

Microstate formation energy

$$\Delta\mu = \mu_B - \mu_A$$

Free
energy:

$$\beta\Phi = -\ln(Z)$$

$$\Phi = G - \Delta\mu N_B$$

Binary alloy, semi-grand
canonical free energy

Gibbs free energy

Monte Carlo Output

Binary alloy, semi-grand canonical ensemble

Thermodynamic variables (Input): $\Delta\mu, T$

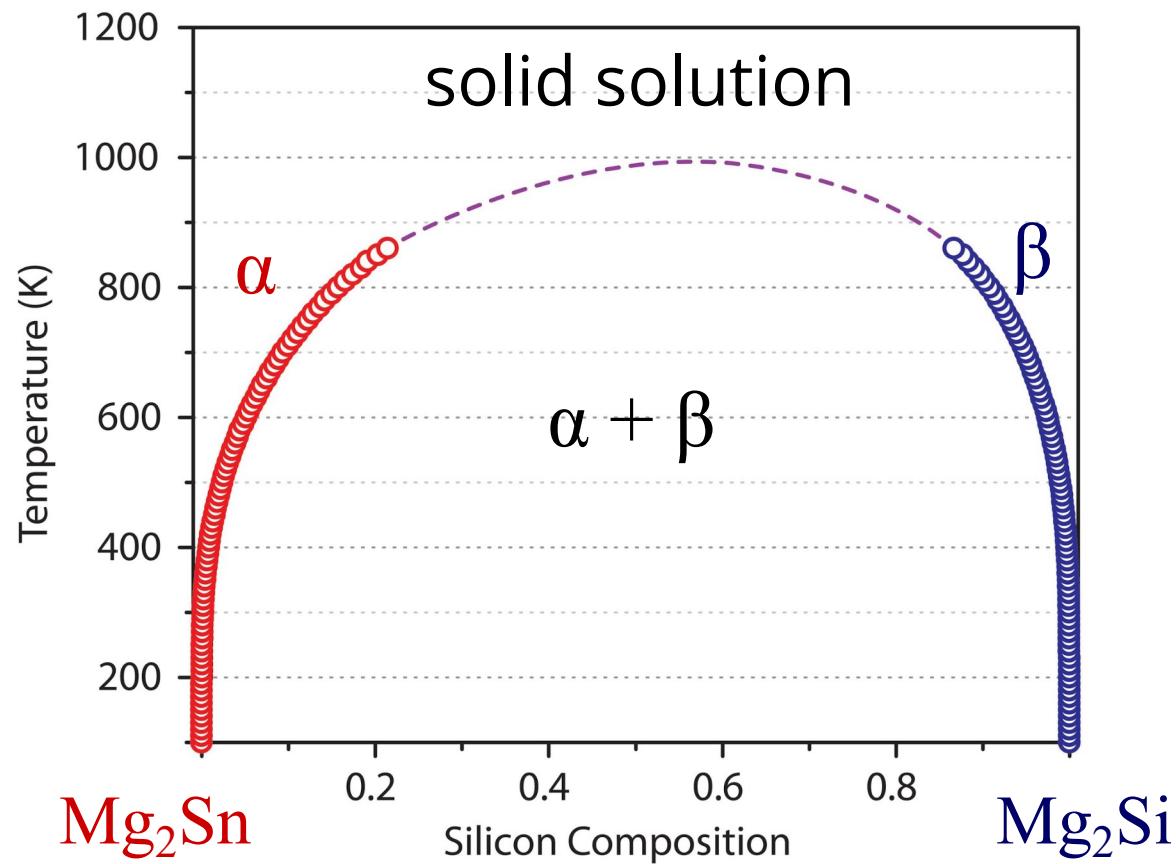
Ensemble averages (Output): $\bar{\Omega} = \sum_{\vec{\sigma}} P(\vec{\sigma})\Omega(\vec{\sigma})$

$$\bar{N}_B = \sum_{\vec{\sigma}} P(\vec{\sigma})N_B(\vec{\sigma})$$

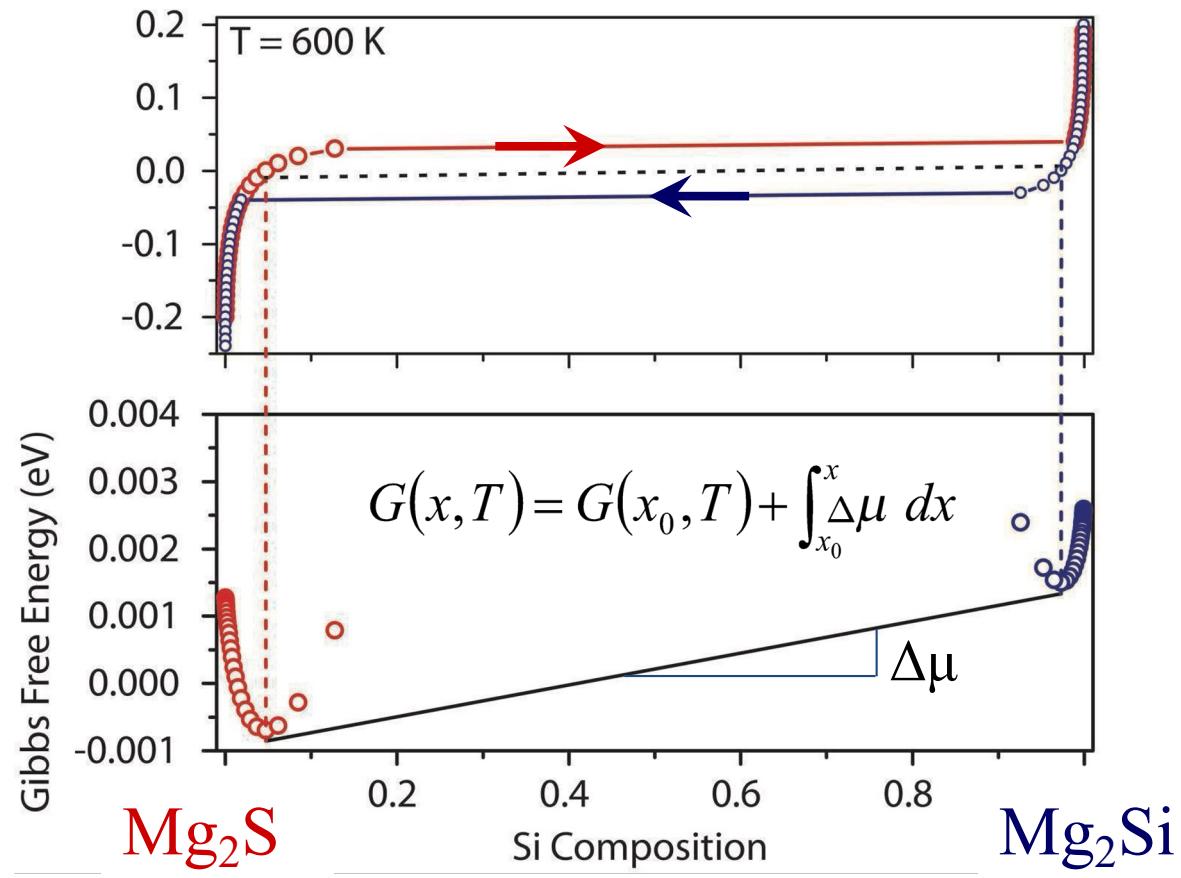
$$x = \bar{N}_B/N$$

Also measure correlations, order parameters, heat capacity, susceptibility, ...

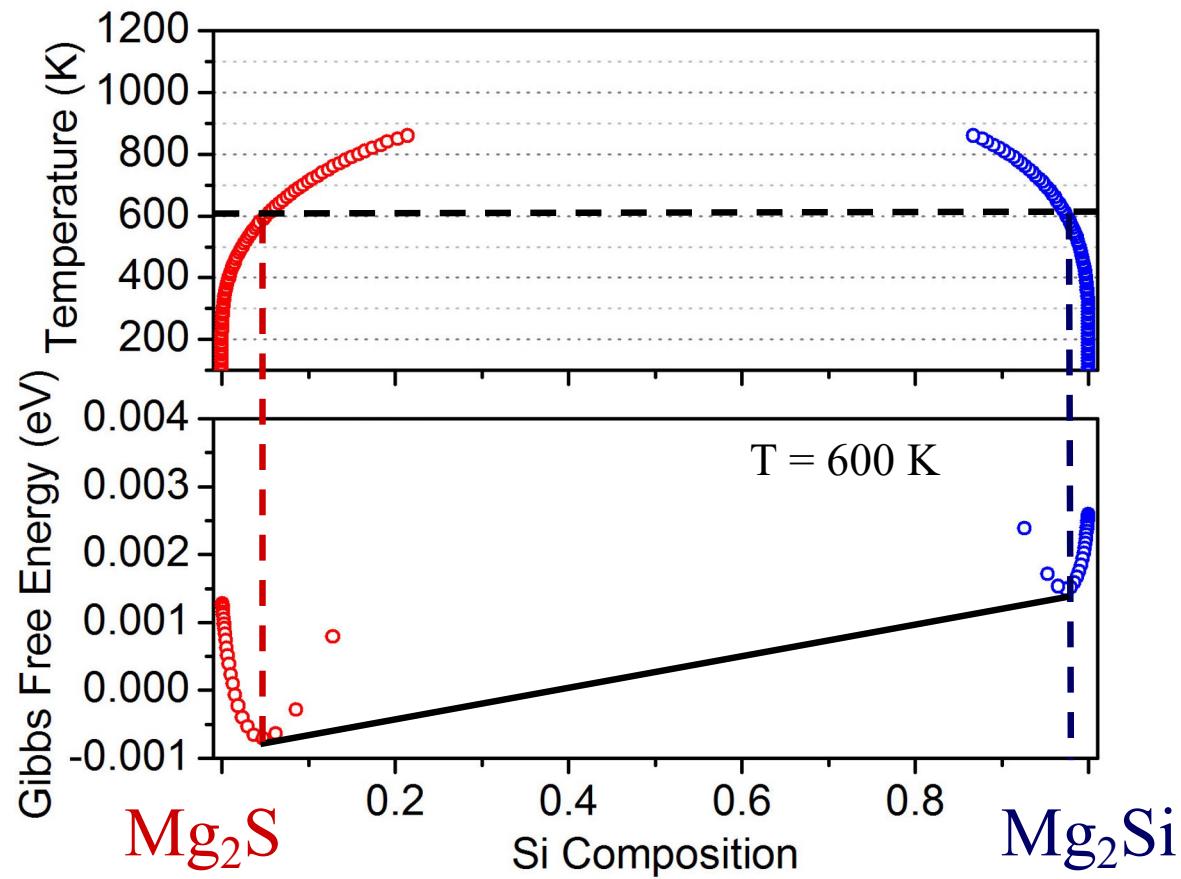
Example: miscibility gap $\text{Mg}_2\text{Sn}_{1-x}\text{Si}_x$



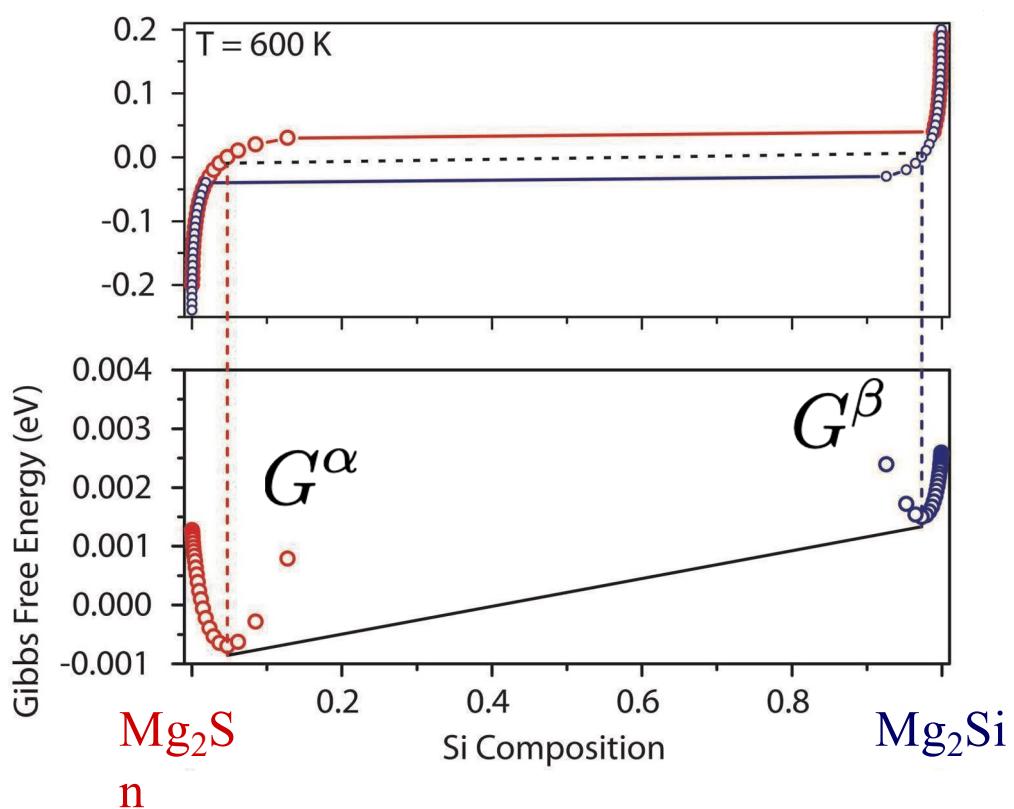
Chemical Potential and Free Energy



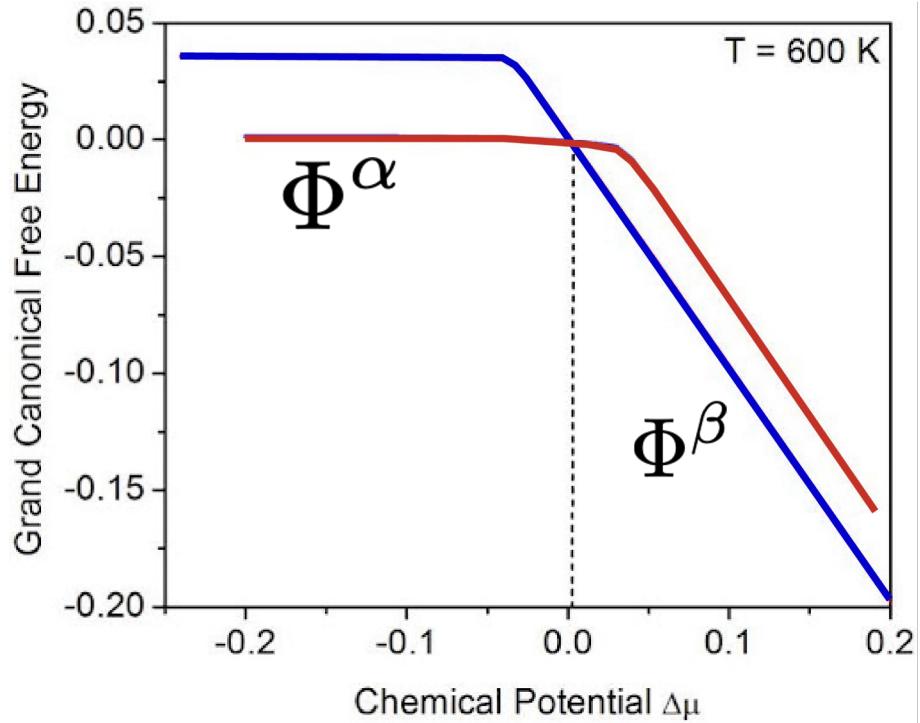
Drawing the Phase Diagram



In terms of grand canonical free energies

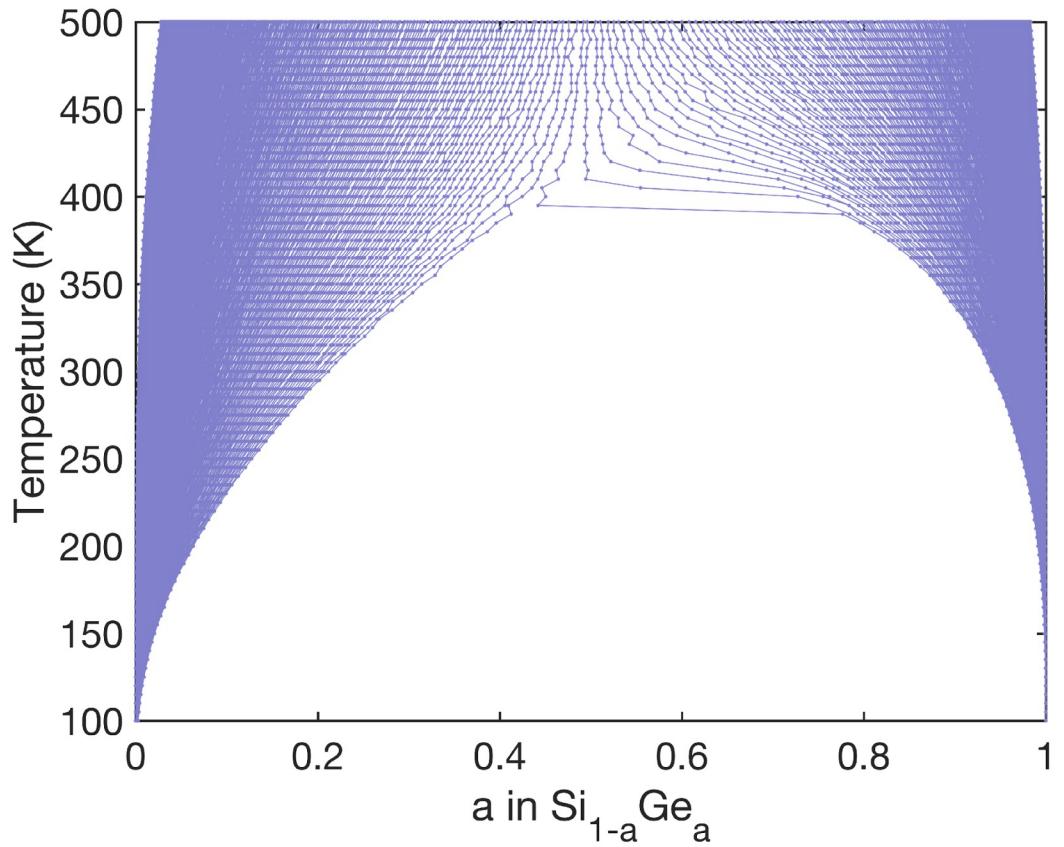


$$\Phi = G - \Delta\mu N_{Si}$$

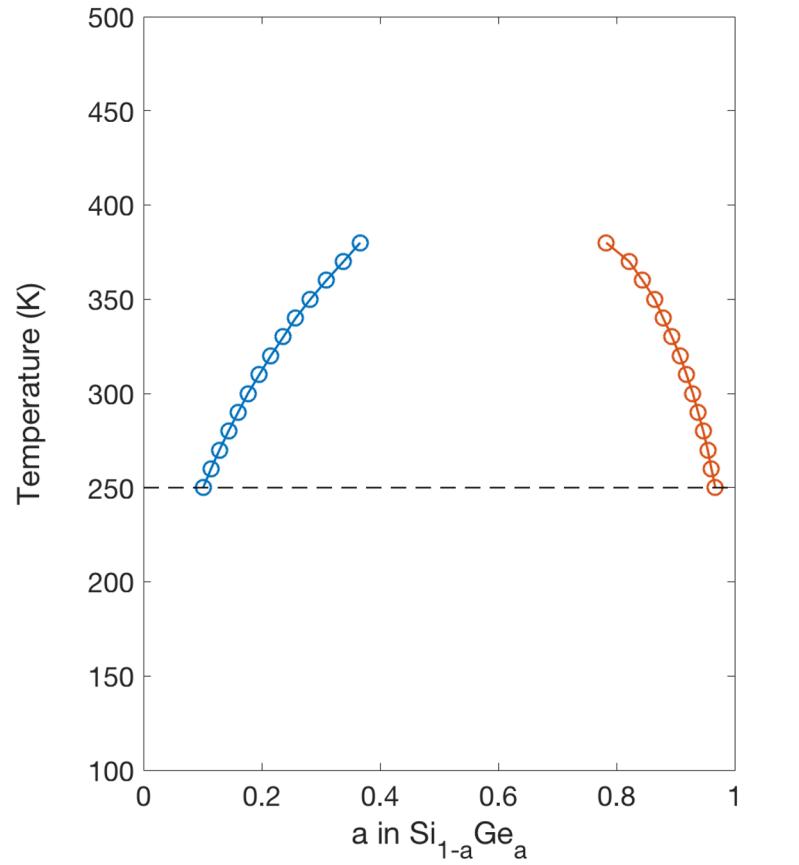
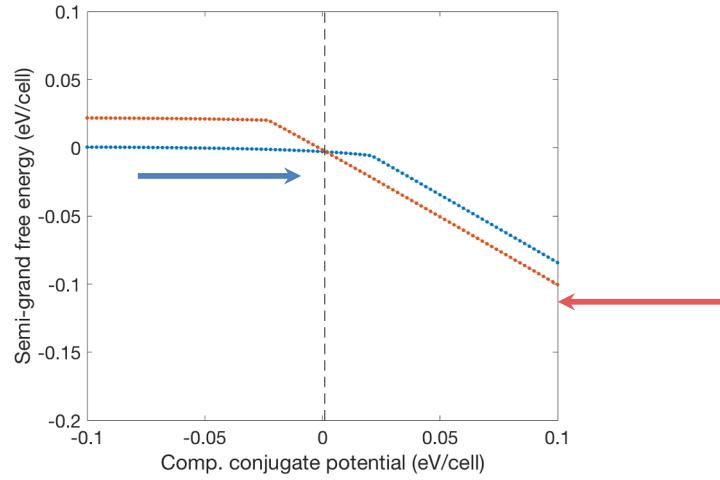
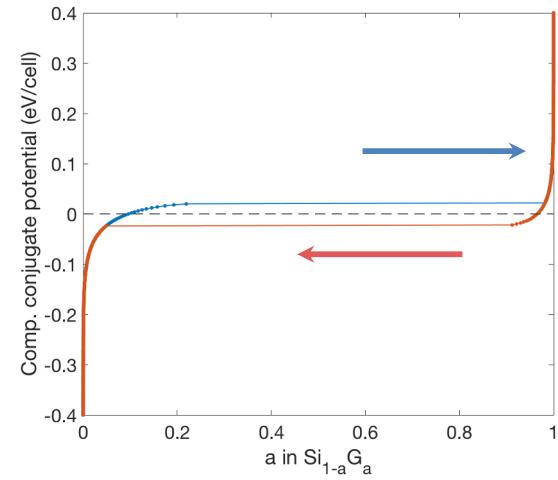


Phase diagram construction

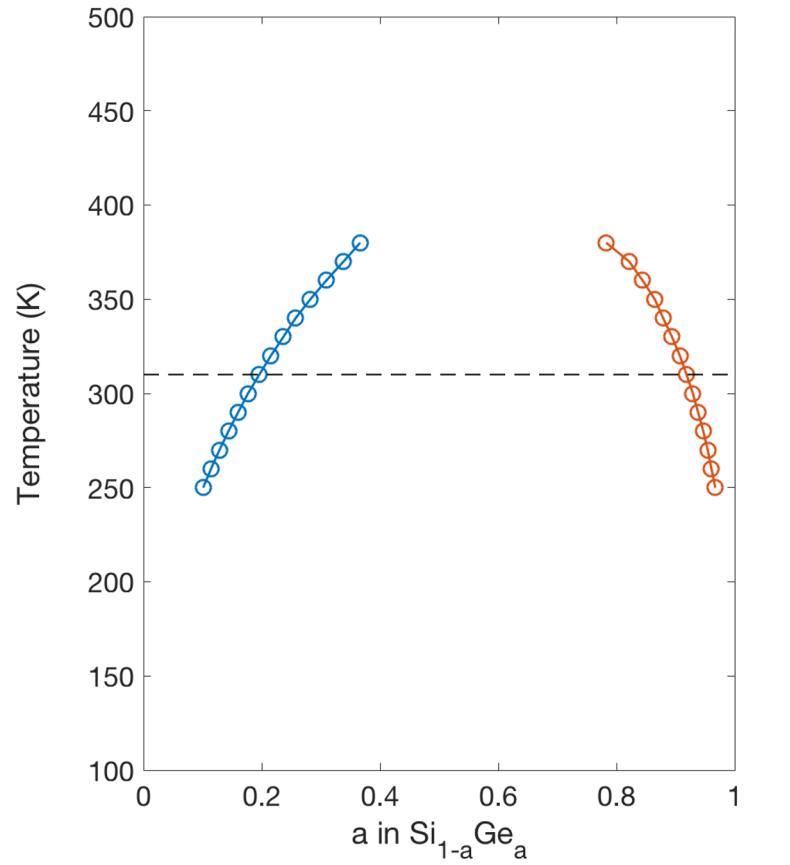
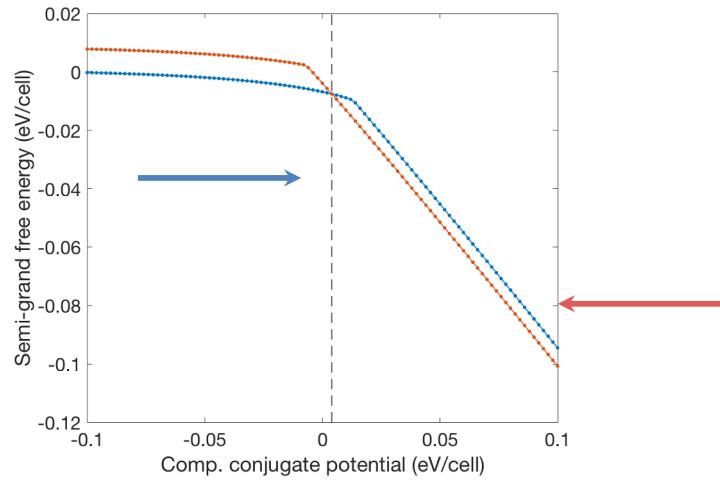
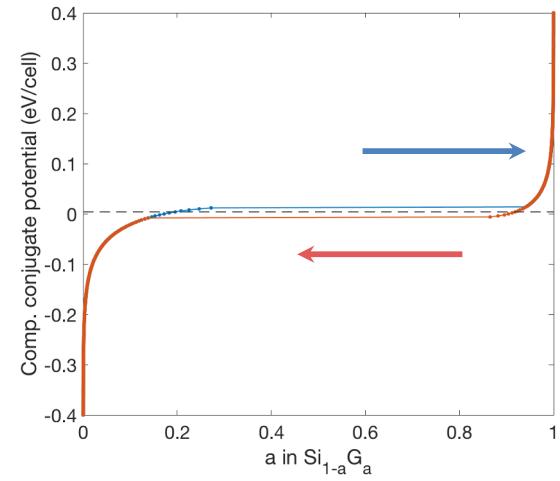
Cooling runs at constant chemical-potential reveal rough shape of miscibility gap



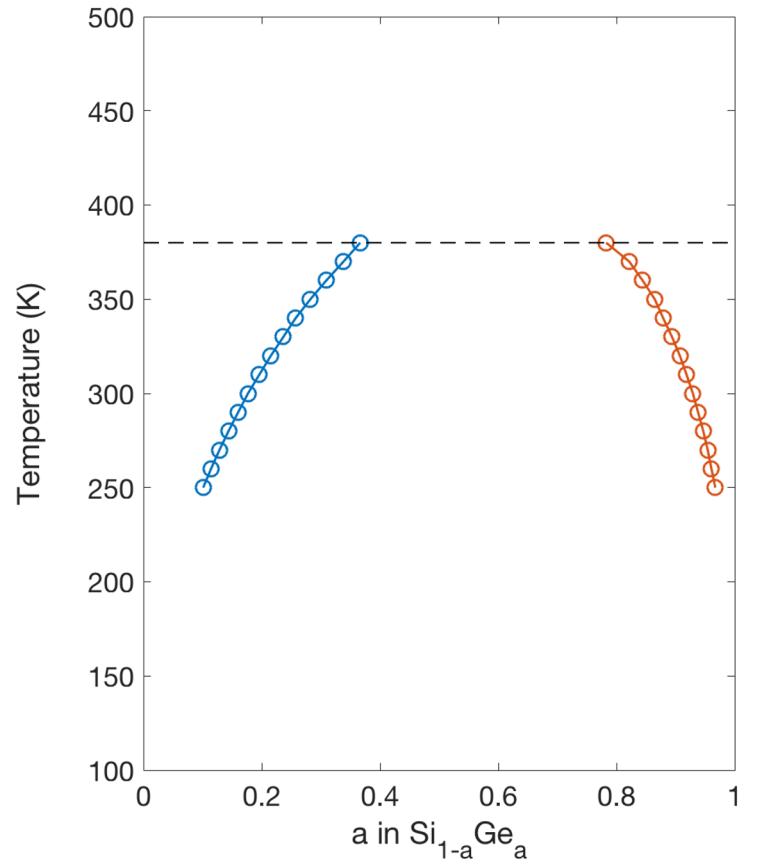
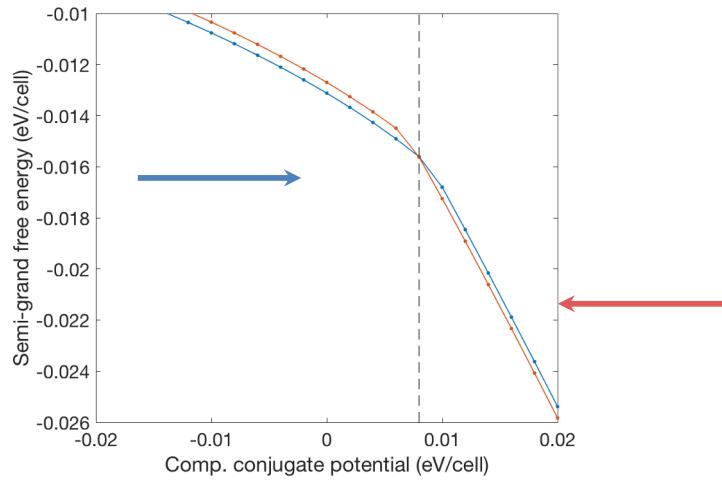
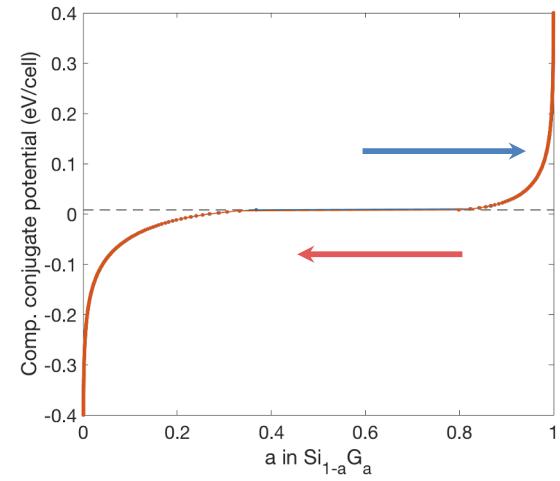
Phase diagram construction



Phase diagram construction

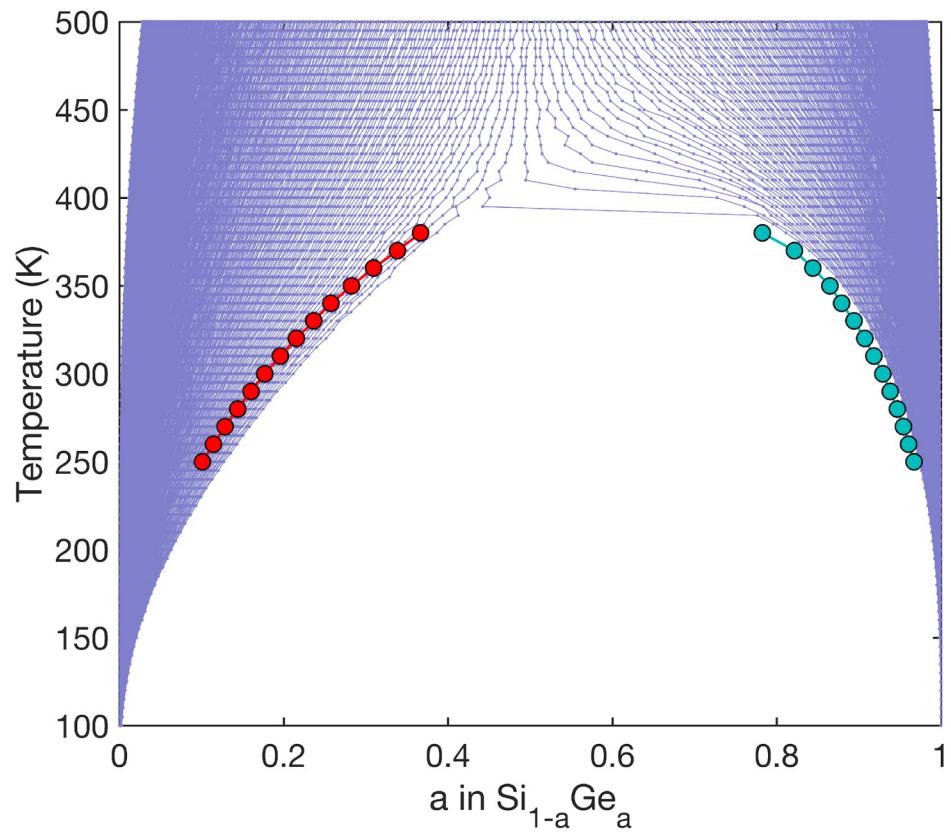


Phase diagram construction

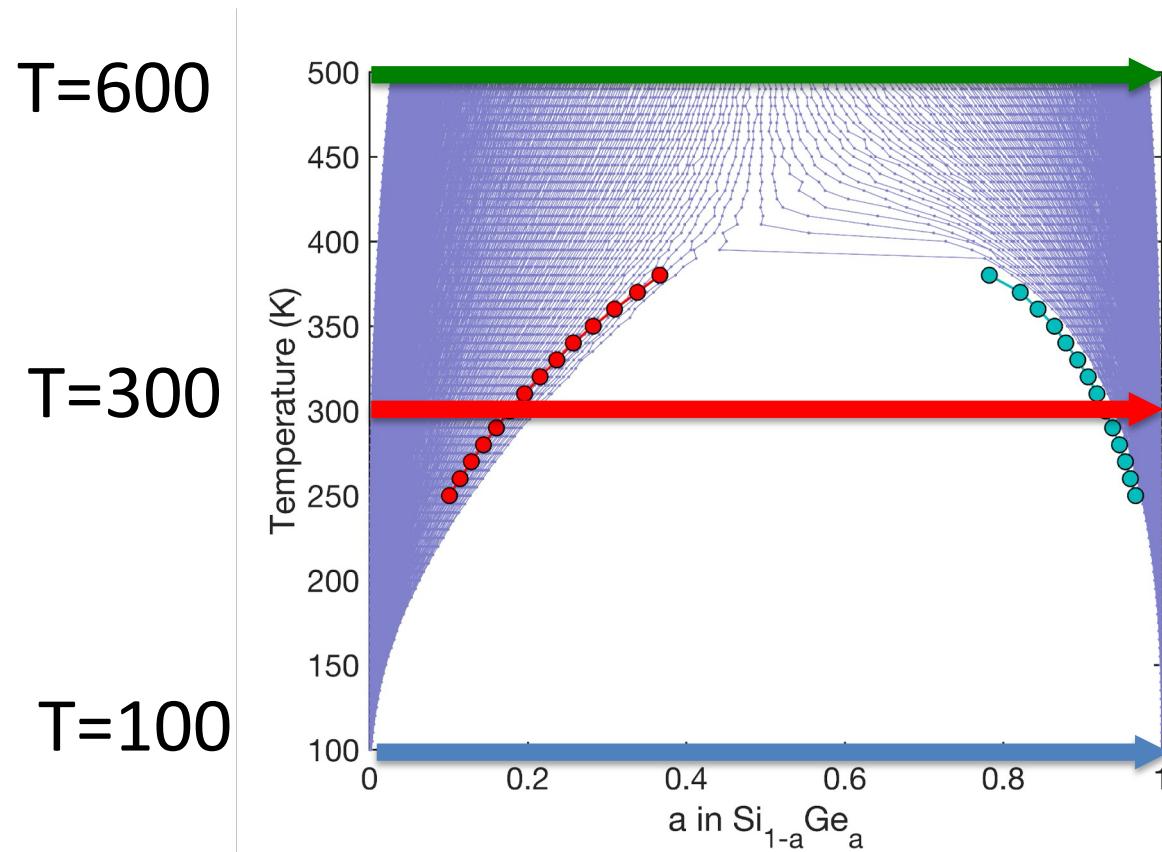


Phase diagram construction

Cooling runs at constant chemical-potential reveal rough shape of miscibility gap



Exercise: Run at T=100, 300, 600 K



Monte Carlo Results

