Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Physics

Øystein Tormodsen Nygård

Kinetic Monte Carlo simulations of the early stages of precipitation in Al-Mg-Si alloys using ab initio based activation energies

Master's thesis in Applied Physics and Mathematics
Supervisor: Jesper Friis and Jaakko Akola
July 2020

NTNU
Norwegian University of
Science and Technology

Øystein Tormodsen Nygård

Kinetic Monte Carlo simulations of the early stages of precipitation in Al-Mg-Si alloys using ab initio based activation energies

Master's thesis in Applied Physics and Mathematics
Supervisor: Jesper Friis and Jaakko Akola
July 2020

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Physics

NTNU
Norwegian University of
Science and Technology

I dedicate this work to the loving memory of my grandfather Sverre Nygaard.
He always supported me, and helped me pursue and fulfil all the different project ideas I had, both as a child and adult.
I will always remember you fondly.

Abstract

Kinetic Monte Carlo simulations have been performed using two different expressions for the activation energies for the purpose of getting better insight into the kinetics of the early stages of precipitation in Al-Mg-Si alloys at an atomic level. The simulation system consists of $25 \times 25 \times 25$ face centred cubic unit cells and the alloy contains $0.67\,\mathrm{at.\%}$ Mg and $0.77\,\mathrm{at.\%}$ Si. The first expression for the activation energies is from the literature and has been thoroughly tested and benchmarked against density functional theory (DFT) calculations in this work. The analysis show that the activation energies are not consistent with DFT, but the method consistently produces clusters with the $\mathrm{L1_0}$ structure at room temperature and tracks the total energy well. Other cluster structures have not been observed and the $\mathrm{L1_0}$ clusters dissolve at $T = 350\,\mathrm{K}$. The second expression, developed in this thesis, is based on cluster expansion and have been trained on DFT calculations to produce physically accurate activation energies. The implementation of the new expression is fast to evaluate and represents the training set well with an RMSE of only $1.8\,\mathrm{meV}$ and cross-validation score of $9.9\,\mathrm{meV}$. Simulation results show that this method with the current training set gives diffusivities close to the literature values, but it does not produce any large clusters and dissolve the $\mathrm{L1_0}$ structure at $T = 400\,\mathrm{K}$ and higher. The method is likely to produce better results with a larger training set that contains a representative selection of structures for all stages of the precipitation process. It is easy to include more lattice sites and add higher order terms to the expression, but this is currently not beneficial due to the limited amount of information in the training set.

Sammendrag

Kinetisk Monte Carlo simuleringer har blitt utført med bruk av to ulike uttrykk for aktiveringsenergiene med hensikt til å få bedre innsikt i kinetikken i den tidlige fasen av presipitering i Al-Mg-Si legeringer på atomnivå. Simuleringssystemet består av $25 \times 25 \times 25$ kubisk flatesentrert enhetsceller og legeringen inneholder $0.67\,\mathrm{at.\%}$ Mg og $0.77\,\mathrm{at.\%}$ Si. Det første uttrykket for aktiveringsenergiene er fra litteraturen og har i dette arbeidet blitt grundig testet og sammenlignet mot beregninger basert på tetthetsfunksjonalteori (DFT). Analysen viser at aktiveringsenergiene ikke stemmer overens med DFT beregningene, men metoden produserer konsekvent klynger med $\mathrm{L1_0}$ strukturen ved romtemperatur og følger totalenergien godt. Klynger med andre strukturer har ikke blitt observert og $\mathrm{L1_0}$ klyngene oppløses ved $T = 350\,\mathrm{K}$. Det andre uttrykket har blitt utviklet i denne avhandlingen, og er basert på klyngeekspansjon. Uttrykket har blitt trent på DFT beregninger for å gi fysisk korrekte aktiveringsenergier. Implementasjonen av metoden er rask å evaluere og representerer treningssettet godt med en RMSE på $1.8\,\mathrm{meV}$ og kryssvalideringsscore på $9.9\,\mathrm{meV}$. Simuleringsresultater viser at denne metoden med det nåværende treningssettet gir diffusiviteter som er nær verdiene i litteraturen, men den produserer ingen store klynger og løser opp $\mathrm{L1_0}$ klynger ved $T = 400\,\mathrm{K}$. Det er sannsynlig at metoden vil produsere bedre resultater dersom et større treningssett som inneholder et representativt utvalg av strukturer for alle steg i presipiteringsprosessen blir brukt. Det er lett å inkludere flere atomplasseringer og høyere ordens korreksjoner i uttrykket, men dette er ikke hensiktsmessig per nå grunnet den begrensede informasjonsmengden i treningssettet.

Preface

This work is written as a master's thesis for the degree Master of Science in Applied Physics at the Norwegian University of Science and Technology (NTNU) in Trondheim. The work is a mandatory part of the degree, corresponds to 30 ECTS and have been performed in the period 27.01.2020 to 22.06.2020. This master's thesis is a continuation of the work done during my specialisation project. During the specialisation project an existing codebase for kinetic Monte Carlo, written in the C language by Jesper Friis, was debugged, made partly modular and updated with new features, and the results of the article by Liang et al. [1] was recreated and extended. The focus for this thesis has been to develop and implement a new expression for the activation energies that can be trained from density functional theory calculations to get simulation results that are based on physically accurate activation energies. The code has also been updated with new features, modularity has been improved and a Python interface have been made to make the program easier to configure and use.

This master thesis is a part of the research project SumAl, which is a collaboration between NTNU, the research organisation SINTEF and several industrial partners from the Norwegian aluminium industry. The purpose of the SumAl project is to gain better insight into the precipitation processes in Al-Mg-Si alloys at different length scales. With this insight the goal is to be able to manipulate solute clusters to optimise the properties of the finished alloys. The main goal for this thesis work is to further improve the kinetic Monte Carlo simulations to gain better insight into the kinetics of the early stages of precipitation in Al-Mg-Si alloys at an atomic scale.

Acknowledgements

I want to express my deepest appreciation to my two supervisors, Professor Jaakko Akola, Department of Physics at NTNU, and Senior Research Scientist Jesper Friis, SINTEF Industry, Department of Materials and Nanotechnology. This work would not have been possible without the skilled guidance provided by my supervisors. A special thanks to Inga Gudem Ringdalen, Research Scientist at SINTEF Industry, Department of Materials and Nanotechnology, for doing the labour-intensive work of executing and organising hundreds of DFT calculations needed for training the model. I am also very grateful to the PhD student David Kleiven who have provided expert help and guidance with the cluster expansion. I would also like to thank Professor Normand Mousseau for fruitful discussions and for sharing his expertise on kinetic Monte Carlo. Lastly, I want to thank my family, and especially my significant other, Mari, for care, encouragement and support throughout my studies.

Table of Contents

List of Tables

List of Figures

Abbreviations

AA	=	Artificial ageing
ASE	=	Atomic simulation environment
CD	=	Cluster dynamics
CE	=	Cluster expansion
CLEASE	=	Cluster expansion for atomic simulation environment
CV	=	Cross-validation
DFT	=	Density functional theory
ECI	=	Effective cluster interaction
FCC	=	Face centred cubic
HAADF	=	High-angle annular dark-field imaging
HF	=	Hartree Fock
k-ART	=	Kinetic activation-relaxation technique
KMC	=	Kinetic Monte Carlo
KRA	=	Kinetically resolved activation barrier
LMOCV	=	Leave-multiple-out cross-validation
LOOCV	=	Leave-one-out cross-validation
MC	=	Monte Carlo
MEP	=	Minimum energy path
NA	=	Natural ageing
NEB	=	Nudged elastic band
NN	=	Nearest neighbour
NNN	=	Next nearest neighbour
RNG	=	Random number generator
STEM	=	Scanning transmission electron microscopy
SS	=	Solid solution
SSSS	=	Supersaturated solid solution
SWIG	=	Simplified Wrapper and Interface Generator
TEM	=	Transmission electron microscopy
VASP	=	Vienna Ab initio Simulation Package

Chapter 1

Introduction

Aluminium alloys are versatile materials that have high corrosion resistance, can be extruded and cast, have a high strength to weight ratio and are energy efficient to recycle [2]. One important use for aluminium alloys is the automotive industry, and other modes of transportation, as the high strength to weight ratio lowers the total weight of the finished vehicle, which lowers the fuel usage and thereby emissions. The properties of an aluminium alloy highly depend on the composition of the alloy and the heat-treatment it undergoes. Impurities are added to the pure aluminium to create an alloy with beneficial properties, the added alloying elements are referred to as solutes. This work will study alloys where the solutes are magnesium and silicon, which will be called Al-Mg-Si alloys, and these alloys are part of the 6xxx series of aluminium alloys. These alloys are commonly used for automotive parts, both in sheet form for body panels and extrusions for load bearing parts [3], and for architectural constructions [2].

Not all alloys are heat treatable but the Al-Mg-Si are and can get a significant increase in hardness due to the heat treatment. The added strength is caused by precipitates, which are small coherent structures that usually have a high solute concentration. Precipitates give the material increased hardness due to the interface strain between the precipitates and the surrounding aluminium [4]. Heat treatment usually consist of several stages at different temperatures and the time spent in each stage also varies. The size, type and number density of precipitates that form depend on how the heat treatment was performed, and this affects the properties of the finished alloy.

The precipitation process in Al-Mg-Si alloys have been studied using both experimental and numerical techniques at different length scales to gain information about the precipitation sequence in the alloys and their influence on the material properties. There is however a lack of studies that have performed numerical simulations that give insight into the kinetics of solute clustering and precipitation in Al-Mg-Si alloys at an atomistic level. An algorithm that is well suited for studying kinetic processes in crystalline materials on long timescales is the rejection-free kinetic Monte Carlo algorithm. This method models the time evolution of stochastic systems through calculating transition rates to all adjacent states and choosing the next state for the system based on these rates. The transition rates

are usually calculated from the activation energy needed to go from the current state to the top of the saddle point between the adjacent- and the current state. This algorithm has been used to study precipitation in other aluminium alloys, such as the Al-Zr-Sc system [5], Al-Sc [6] and Al-Zn-Mg-Cu [7].

Early stages of precipitation in Al-Mg-Si alloys have been studied using KMC in an article by Liang et al. [1]. In the utilised model all atoms must reside at a face centred cubic lattice site, which is a good approximation in the early stages of precipitation. For each iteration one of 12 nearest neighbours of a vacant lattice site can jump from their current site to the vacant site. The expression they used for calculating the activation energies only take into account the nearest neighbours of the atom that jumps and the nearest neighbours of the vacancy. This method was implemented and used in my specialisation project [8] to recreate and extend the work presented in the original article. The outcome of KMC simulations are governed by the activation energies for atoms to jump to a vacant site. The expression used in [1] does not treat atoms at different sites, relative to the jump direction, differently and there are no interactions between the neighbours, only between neighbour and the atom that jumps and neighbour and the vacancy. The outcome of the simulations may therefore lack important information, which can result in the kinetics not being consistent with the real processes.

The main focus of this master thesis is to develop an expression for calculating the activation energy for an atom jumping to a neighbouring vacant lattice site that can easily be expanded to include more lattice sites and higher order corrections. The expression should be able to be trained from a set of structures with physically accurate activation energies. This expression will be based on the principle of cluster expansions and the coefficients of the expression, also known as effective cluster interactions, will be determined from the training set using Bayesian regression. The training set consist of structures where the activation energies have been accurately computed using density functional theory (DFT). The goal is that this will produce physically accurate activation energies that result in better and more realistic insight into the kinetics of the early stages of the precipitation process at an atomic level.

This thesis will start with an introduction to the precipitation process in Al-Mg-Si alloys, followed by an introduction to rejection-free KMC and a short introduction to the method used in the article by Liang et al. The introduction to KMC will cover the algorithm, requirements for good KMC moves and compare it to other numerical methods. The next chapter will go through cluster expansions in general and present the developed expression for the activation energies. It will also go through how to create the training set, how to determine the effective cluster interactions using Bayesian regression and discuss methods for expanding the training set in an efficient way. A chapter on the implementation of the algorithms follows, which goes through updates to the existing code, how the new expression for the activation energies have been implemented to get low memory usage and high efficiency and go through a new Python interface for the code.

The results will start with introducing and inspecting the training set and compare different methods for determining the optimal combination of effective cluster interactions. In the next part, an in-depth inspection and benchmark of both the cluster expansion based method and the method by Liang et al. is performed. Simulation results from both methods is then presented, including cluster evolution, snapshots of the lattice, diffusivities and total

energy evolution. Clustering rates for the method by Liang et al. is presented together with line fittings to the evaporation rates and compositional data for clusters. The last section discusses error sources for the training set, the modelling assumptions and error sources for the two methods of calculating the activation energies. The thesis is rounded off with a conclusion and suggestions for future work.

Chapter 2

Theory

This chapter will start with a short introduction to precipitation hardening of Al-Mg-Si alloys, covering the topics needed for this thesis. The next section will introduce the rejection-free kinetic Monte Carlo algorithm and compare it to other methods, both at a general level and for the purpose of simulating precipitation. After the comparison, the details of how kinetic Monte Carlo is used to simulate precipitation is presented. In the last section an introduction to an existing method for calculating the activation energies for the Al-Mg-Si system is presented.

2.1 Introduction to Al-Mg-Si alloys and precipitation hardening

This section will introduce Al-Mg-Si alloys and the process of precipitation hardening, also known as age hardening. Aluminium alloys are divided into different series based on the main alloying elements, and the Al-Mg-Si alloys belong to the 6xxx series of alloys. This series of alloys is commonly used for automotive applications [3], both in sheet form and extrusions, and for architectural purposes [2].

Heat treatment is an important step when producing parts out of an aluminium alloy since the heat treatment leads to precipitation, which is an important strengthening mechanism [2]. The precipitates are small coherent structures that cause a strength increase due to interface strain between the surrounding Al matrix and the precipitates [4]. A typical heat treatment process of an aluminium alloy is illustrated in Figure 2.1. The treatment starts at a temperature that is below the melting point, but high enough for the alloy to form a solid solution (SS) where the solutes are well dispersed, temperature is usually above $740\,\mathrm{K}$ [2]. Quenching, a rapid cooling of the alloy usually performed by submerging the part in water, follows and creates a supersaturated solid solution (SSSS) since the atoms do not have time to reach equilibrium during the fast cooling process. The SSSS will start to form clusters of solutes during natural ageing (NA), which is performed at room temperature. After some time of NA, ranging from minutes to weeks, the tempera-

ture is elevated to speed up the precipitation forming, a typical temperature for Al-Mg-Si alloys is $450\,\mathrm{K}$ [2]. The time and temperature for each step can be varied and affects the final properties of the alloy.

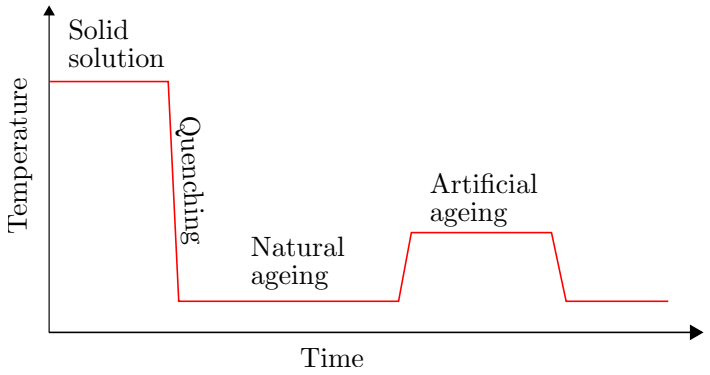


Figure 2.1: Illustration of a typical heat treatment procedure for Al alloys. The alloy is heated to a temperature below the melting temperature but high enough for the solutes to disperse evenly and form a solid solution, the alloy is then quenched, which creates a supersaturated solid solution. Then the alloy is aged at room temperature before the alloy is heated to an elevated temperature for artificial ageing [2].

During the heat treatment, the alloy undergoes several different structural transformations and many of the structures are metastable. The general sequence of precipitation in Al-Mg-Si alloys is [4]

$$\begin{aligned}\text{SSSS} \rightarrow \text{solute clusters} \rightarrow \text{spherical coherent Guinier Preston zones} \rightarrow \\ \text{needle shaped } \beta'' \rightarrow \beta', \mathrm{U1}, \mathrm{U2}, \mathrm{B}' \rightarrow \text{plate shaped } \beta\end{aligned}$$

The peak hardness of the material usually contain a high fraction of the metastable β'' structure [9]. This thesis focuses on the early stages of precipitation, mainly during NA, hence only the steps up to β'' are relevant for this work.

Pure aluminium at room temperature have the face centred cubic (FCC) structure. The FCC unit cell is as the name implies cubic and have one lattice site at each corner, shown as white spheres in Figure 2.2, and at the centre of each face of the cube, shown as red spheres in the figure, which results in a total of 4 lattice sites per unit cell. Alloys with low percentage of alloying elements, often called lean alloys, are assumed to have a structure close to FCC during supersaturated solid solution. Later in the precipitation process when larger clusters start to form this assumption is no longer good, since the large clusters often have a different crystal structure than the Al host matrix. This new crystal structure is energetically favourable for the bulk of the cluster, but due to the interfacial energy towards the Al matrix the clusters must become a certain size before the new structure has lower total energy.

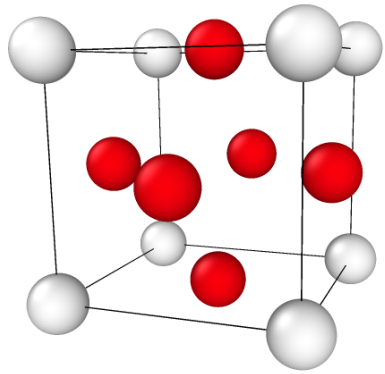


Figure 2.2: Illustration of the lattice sites in a face centred cubic (FCC) unit cell. In the figure the corner, or cubic, lattice sites are white spheres and the face centred sites are shown in red.

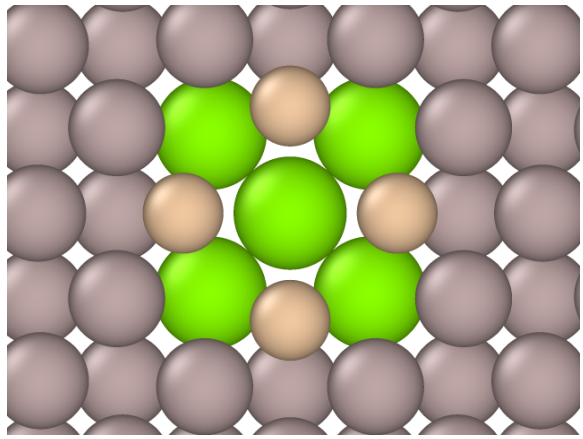


Figure 2.3: The figure shows a β'' "eye" consisting of a layer of 4 Si atoms, sand coloured, with a Mg atom, green, in the middle of them. The layer below has 4 Mg atoms and the rest of the structure is Al. The structure is a slab that is only 2 layers thick, but due to periodic boundary conditions it is an infinite column. The atom positions have been relaxed using DFT by Research Scientist Inga G. Ringdalen at SINTEF Industry, Department of Materials and Nanotechnology.

The structure and form of the important β'' precipitates is of great interest and have been studied in detail in several studies. The composition was found to be $Mg_4Al_3Si_4$ using high-angle annular dark-field scanning transmission electron microscope (HAADF-STEM) but DFT indicated that $Mg_5Al_2Si_4$ is the most stable structure [10]. The β'' precipitates are coherent and elongated along a $\langle 100 \rangle$ direction of the Al matrix and have a typical cross section of 1-15 nm^2 and length 30-100 nm [4]. The unit cell for the β'' is monoclinic [10], but is closely related to the FCC structure. In Figure 2.3 a slab with a β'' "eye" that have been relaxed with DFT is shown. The sand coloured Si atoms are pushed

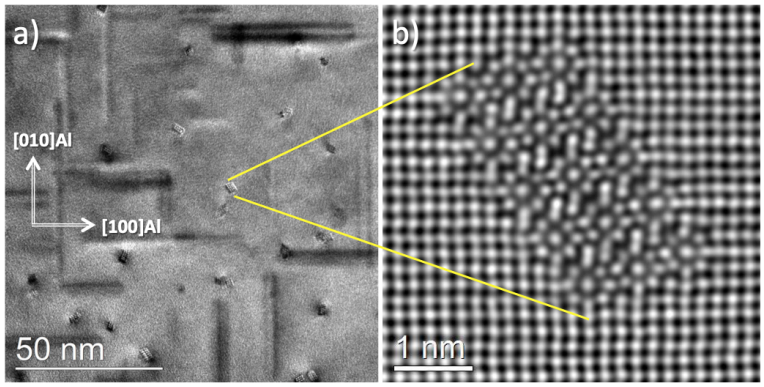


Figure 2.4: Figure a) shows a bright field TEM image with a typical β'' microstructure in a peak aged ultrapure 6082-like Al-Mg-Si alloy. Figure b) shows a HAADF-STEM image of a β'' cross-section with atomic resolution. These images have been provided by Calin D. Marioara, Senior Research Scientist at SINTEF Industry, Department of Materials and Nanotechnology.

outwards compared to their FCC positions due to the green Mg atom that is in the middle of the Si atoms in the same layer. A β'' precipitate can consist of many of these eyes. A transmission electron microscopy image of the precipitate structure in a peak aged Al-Mg-Si alloy with a typical β'' microstructure is shown in subfigure a) of Figure 2.4. Subfigure b) shows a HAADF-STEM image of the cross-section of one of the β'' precipitates, where the "eyes" are clearly visible.

The β'' can not be represented in the KMC simulations by only FCC lattice sites. The sand coloured Si atoms in Figure 2.3 and the 4 green Mg atoms forming a square can be represented by their FCC positions even when they are slightly displaced since the KMC does not use their exact positions for calculating activation energies. The Mg atom in the centre of the 4 Si atoms can be represented by adding an octahedral lattice site to each FCC unit cell. The octahedral site is in the centre of the FCC unit cell.

2.2 Rejection-free Kinetic Monte Carlo

This section will introduce the rejection-free kinetic Monte Carlo (KMC) algorithm, which was originally introduced by Bortz et al. as the n-fold way [11]. Some of the figures and some explanations for the governing equations are from the introduction of the KMC algorithm given in my specialisation project [8]. The KMC algorithm is a very versatile algorithm as the only requirement is that there must be a way obtain a set of possible transitions, with corresponding transition rates, for every state of the system. KMC has been used to study surface kinetics and catalysts, see review article [12], precipitation and diffusion in metal alloys, for example Fe systems [13, 14] and Al alloys [5, 7, 15, 16], and biological processes such as protein folding [17] and crystallisation [18]. The use in this work will be to study precipitation in the Al-Mg-Si system on an atomic scale by having a single vacancy in a FCC lattice where one of the nearest neighbours of the vacancy will jump to the vacant lattice site.

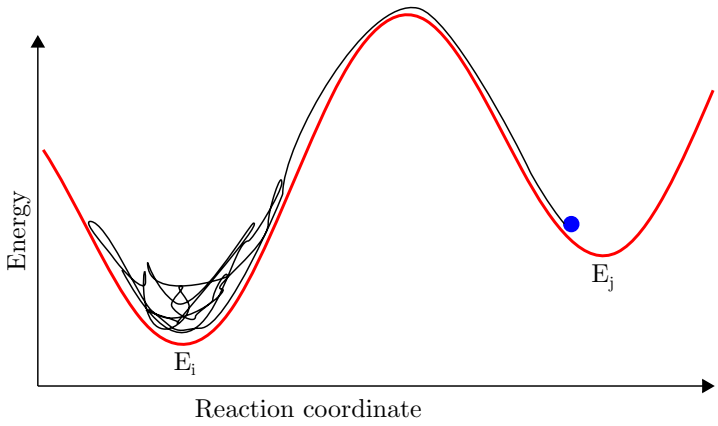


Figure 2.5: Illustration of the energy landscape for the reaction going from an initial state, state i, to an adjacent state, state j. The system may spend a long time in state i before escaping, especially when the temperature of the system is low, KMC is efficient for such problems since it calculates the escape time instead of simulating all the failed attempts at transitioning into an adjacent state.

The main idea behind the KMC algorithm is that instead of waiting for the system to change state, it instead calculates the escape time from the current state and evolves the system into a neighbouring state in a single iteration, see Figure 2.5. This is very beneficial for systems where the escape time is large, which is the case for most systems when the temperature of the system is low. To be able to simulate a system with KMC each state of the system must have a set of possible transitions, which can either be calculated during runtime or be predefined, into adjacent states, and a way to obtain the transition rate for each of these. Which transition that is chosen each time is random, but they are chosen in such a way that on average each transition is chosen proportional to their transition rate.

For a system in state i the transition rate for transitioning into the adjacent state j is given by

$$\Gamma_{ij} = \nu \exp\left(- \frac{E_{ij}^{\mathrm{act}}}{k_{\mathrm{B}} T} \right), \tag{2.1}$$

where ν is an attempt frequency, k_{B} is the Boltzmann constant, T is the temperature of the system and E_{ij}^{act} is the activation energy for the transition, often referred to as the energy barrier for the transition. In Figure 2.6 the energy landscape for a transition between two adjacent states is shown together with the definition of the activation energy for both the forward and backward reaction.

To choose which of the N_{e} available transitions from state i that will be carried out, all the partial sums of the transition rates need to be calculated

$$R_{i,k} = \sum_{j=1}^{k} \Gamma_{ij}. \tag{2.2}$$

Then a random number $u \in [0, 1)$ is drawn from a uniform distribution to determine which

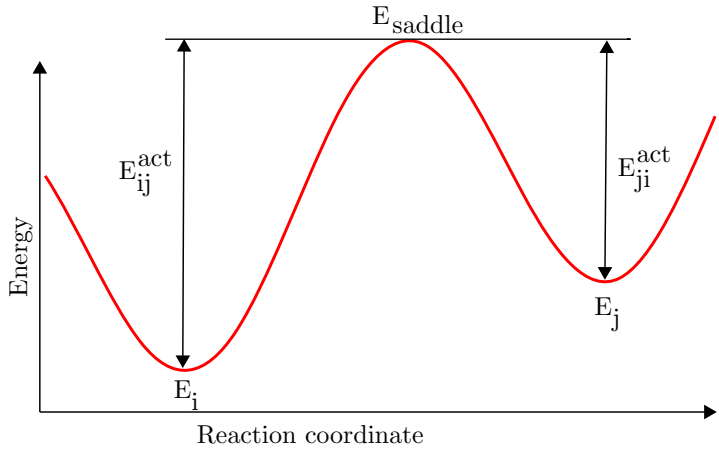


Figure 2.6: Illustration of the energy landscape for the system going from state i to state j. The activation energy for the transition from state i to state j is given by E_{ij}^{act}, and the reverse reaction has activation energy E_{ji}^{act}.

event to carry out. The chosen event, event k, is the one that fulfills

$$R_{i,k-1} \leq uR_{i,N_e} < R_{i,k}. \tag{2.3}$$

For a graphical representation of the selection process see Figure 2.7. This process ensures that any event can happen at any iteration, no matter how unlikely it is, but on average each event will be chosen proportional to their transition rate.

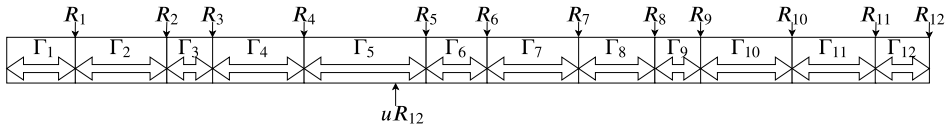


Figure 2.7: Illustration of the selection process of which event to carry out in a KMC step, here illustrated with 12 different possible events. The width of each box is proportional to the rate, $\Gamma_i j$, of the event, the sums of rates are given by $R_{i,k}$ and $u \in [0, 1)$ is a random number from a uniform distribution. The subscript i have been left out in the Figure. The event that will be carried out is the one that uR_{12} points to. The figure is a reprint from my specialisation project [8].

For each iteration the time is increased by Δt, which represents the time the system spent in state i before escaping to one of the adjacent states [19]. The escape time can be calculated as

$$\Delta t = \frac{1}{C\sum_{j=1}^{N_e}\Gamma_{ij}} = \frac{1}{CR_{i,N_e}}, \tag{2.4}$$

where C is dimensionless constant that scales the time of the system. The original algorithm by Bortz et al. included a factor of $\ln(1/u')$ where u' is a new random number

from the same distribution as u was drawn from. This factor is often omitted since the expectation value of the factor is 1, some examples of this are [1, 5, 13, 19]. Omitting this factor will make the escape time less stochastic, which will affect the time evolution between single jumps, but it will not affect the average time evolution.

2.2.1 Requirements for good KMC moves

Both regular Monte Carlo and kinetic Monte Carlo simulations need to fulfill the criteria of being Markovian, ergodic and obey detailed balance to correctly sample the configurational space of the system. The introduction to these requirements is based on the book by Newman and Barkema [19].

For the process of generating a new state j from the current state i to be Markovian, the transition probability, $P(i \to j)$, should only depend on information about the current state and the new state. The history of how the system got to state i should therefore not impact the chosen escape path, which is a common assumption for systems that spend a sufficient amount of time in a state before transitioning into the next state. Another criterion is that the transition probabilities should be time independent and the sum of all the transition probabilities from state i should be equal to unity. The transition probabilities from state i can in KMC simulations be calculated as

$$P(i \to j) = \frac{\Gamma_{ij}}{R_{i,N_\mathrm{e}}}. \tag{2.5}$$

As long as the transition rates fulfil the requirements of time independence and only use information about the current and possible new states, the transition probabilities will also fulfil the same requirements and the requirement of summing up to unity.

The criterion of being ergodic states that the system must be able to reach every state of the system from any starting state in finite time. This ensures that the whole configuration space of the system can be explored if the simulation is left running long enough. In a KMC simulation most of the transition rates from a state will be zero since normally only a small number of adjacent states can be transitioned to. This is not a problem since the requirement does not include any restrictions on the number of transitions needed to go from one state to the other. The requirement is therefore fulfilled if there always exist a chain of transitions that can bring the system from the current state to any other state in the configuration space.

The criterion of detailed balance, together with the requirement of being ergodic and Markovian, ensures that the configuration space is sampled correctly. Detailed balance can be written as

$$p_i P(i \to j) = p_j P(j \to i), \tag{2.6}$$

which means that the probability of being in state i, p_i, and transitioning to state j is the same as the probability of being in state j and transitioning to state i.

For a regular MC simulation the correct probability of observing the system in state i when the system is in equilibrium is proportional to the Boltzmann weight of state i. This weight is given by

$$p_i = \frac{\exp\left(-E_i/k_\mathrm{B}T\right)}{Z}, \tag{2.7}$$

where Z is the partition function of the system. By rearranging equation (2.6) and inserting equation (2.7) the criterion of detailed balance for MC becomes

$$\frac{P(j \to i)}{P(i \to j)} = \frac{p_i}{p_j} = \exp\left(\frac{E_j - E_i}{k_\mathrm{B}T}\right). \tag{2.8}$$

By combining (2.5) and (2.1) and inspecting Figure 2.6 it can be shown that for the KMC algorithm the ratio of the transition probabilities are

$$\frac{P(j \to i)}{P(i \to j)} = \frac{\Gamma_{ji}/R_{j,N_\mathrm{e}}}{\Gamma_{ij}/R_{i,N_\mathrm{e}}} = \frac{R_{i,N_\mathrm{e}}}{R_{j,N_\mathrm{e}}} \exp\left(\frac{\Delta E_{ij}^{\mathrm{act}} - \Delta E_{ji}^{\mathrm{act}}}{k_\mathrm{B}T}\right) = \frac{R_{i,N_\mathrm{e}}}{R_{j,N_\mathrm{e}}} \exp\left(\frac{E_j - E_i}{k_\mathrm{B}T}\right). \tag{2.9}$$

Which is a factor $R_{i,N_\mathrm{e}}/R_{j,N_\mathrm{e}}$ away from being the ratio of the Boltzmann weights as it would be for detailed balance for regular MC. It has proven difficult to find a proof in the literature of why this factor appears and if $p_i = R_{i,N_\mathrm{e}} \exp(-E_i/k_\mathrm{B}T)/Z$ is the correct probability of observing the system in state i. To get an explanation an email exchange with Professor Normand Mousseau, the leader of the research team behind the off-lattice KMC implementation called kinetic activation relaxation technique (k-ART) [20, 21], was initiated. According to Mousseau the extra factor R_{i,N_e} in p_i comes from the fact that KMC moves between energy basins that can contain several points of the configuration space. The sum of the transition rates, R_{i,N_e}, corresponds to the entropic weight of the basin, which is the proper weight to account for the local ensemble of points. Detailed balance is therefore fulfilled for the KMC algorithm and $p_i = R_{i,N_\mathrm{e}} \exp(-E_i/k_\mathrm{B}T)/Z$ should be interpreted as the probability of observing the system in one of the points in the basin which point i is part of.

This extra factor in the detailed balance requirement reflects that for getting the kinetics correct it is not only important that the energy difference between states are correct, but also the energy of the top of the barrier need to be correct, E_{saddle} in Figure 2.6. The energy difference between states can be correct even if the value for E_{saddle} is incorrect if both activation energies are too low or high by the same amount. This affects the transition rates and can lead to an incorrect time evolution, even though it would fulfil the detailed balance criterion for regular MC simulations.

2.2.2 Comparison to other methods

This subsection will compare KMC to other well-known numerical methods, both generally and for the purpose of studying precipitation in aluminium alloys. The methods it will be compared to is regular Monte Carlo (MC), Molecular Dynamics (MD) and an off-lattice KMC implementation known as kinetic activation relaxation technique (k-ART).

Monte Carlo simulations are used to sample the equilibrium distribution of the configurational space of a system. An iteration of a Monte Carlo simulation starts with drawing a random number to generate a trial move, which is often non-local. For the purpose of studying precipitation the trial move would normally be to swap two random atoms in the lattice. To determine if the move should be accepted or rejected a new random number $u \in [0, 1)$ is drawn from a uniform distribution and compared to an acceptance criterion, A, if $u < A$ the move is accepted. There are many different methods for calculating the acceptance criterion, and this will affect the efficiency of the method. One of the most

commonly used methods is the Metropolis-Hastings acceptance rule where A, for a move from state i to state j, is given by [19]

$$A(i \to j) = \begin{cases} \exp(-(E_j - E_i)/k_{\mathrm{B}}T) & \text{if } (E_j - E_i) > 0 \\ 1 & \text{otherwise} \end{cases}. \tag{2.10}$$

If the move is rejected the original state is recounted in the sampling.

Regular MC simulations can be considered as a set of disordered non-local events and can therefore only give thermodynamic information about the system and no kinetics. If the goal is only to study thermodynamic quantities at equilibrium, regular MC is usually computationally more efficient than KMC since it allows non-local moves and only computes the energy of one trial configuration instead of calculating the transition rate for each possible transition. If the temperature of the system is low, the acceptance ratio will also become low and the KMC method can be more efficient for these cases.

For studying precipitation, MC is useful for finding ground state structures of a system through simulated annealing [22]. This method starts with a high temperature, such that almost any move is accepted, and then decreases the temperature until almost only moves where the energy decreases is accepted. MC is also often used for creating phase diagrams, for example in [23]. It is not possible to study how the clusters in the material evolve and eventually turn into precipitates with MC, since there is no kinetics.

Molecular dynamics evolves the system in time by numerically integrating Newton's equations of motion. This means that all the particles in the system can move by a small amount each timestep. To successfully integrate the equations of motion, each timestep must be very small, typically on the order of 1 fs [24]. Due to the high computational cost only short time scales are reachable, usually on the order of some µs [25], but purpose-built supercomputers have been able to reach timescales of 1 ms [26]. Compared to KMC that only calculates the escape time from the current state, MD will simulate all the failed attempts in Figure 2.5, which uses a lot of computation time. MD is not suited for direct atomistic simulations of the precipitation process in most alloys since the reachable time scale is low. It is however useful for studying diffusion and other important aspects of precipitation, one example is a study of the strengthening effect of precipitates in Al-Cu alloys [27].

A new algorithm, based on the KMC algorithm, called the kinetic Activation Relaxation Technique (k-ART) have been developed by a group led by Normand Mousseau [20]. One of the main differences is that the k-ART algorithm is an off-lattice model, while the regular KMC model used in this work is an on-lattice model where each atom must reside at a lattice site. The advantage of an off-lattice model is that it is more realistic and can include deformations and long-range elastic interactions. Since there is no fixed lattice, it is not possible to use a fixed set of possible transitions. To solve this, k-ART uses ART nouveau, see the detailed description of the algorithm in [20], to find the possible transitions. The corresponding energies are calculated using empirical or ab initio potentials. The possible transitions with corresponding rates are stored in a catalogue together with a topological classification of the local environment. This makes it possible to reuse them if the simulation encounter a topologically equivalent local environment. An advantage of searching for possible transitions during the simulation is that it removes the possibility of introducing a bias through having a fixed set of possible transitions.

The similarity between regular KMC and k-ART is that they both calculate the rate for all possible transitions, chooses an event to carry out and increase the time the same way. The advantages of the off-lattice model do however come with significantly increased computational cost due to the event search, topological classification and rate calculations. It will therefore take a lot more computational time and resources to reach the same timescale with the off-lattice model as with the on-lattice model.

From this short introduction and comparison to other numerical methods it is evident that all of them are very useful for their own purposes. For studying equilibrium properties MC is often the best choice and MD is useful for getting accurate kinetics for short timescales. For studying the kinetics of precipitation in aluminium alloys only the KMC on-lattice and off-lattice, k-ART, models are suited for the job. The on-lattice model has superior computational speed and can reach longer time scales, while the off-lattice model includes a more realistic description since it includes deformations, long-range interactions and creates the set of possible events at runtime.

2.2.3 KMC for atomistic simulations of precipitation in Al-Mg-Si alloys

The use of KMC in this thesis is to study the early stages of precipitation in Al-Mg-Si alloys at an atomic scale. This subsection will go through the details of how the KMC algorithm is used for simulating precipitation of alloys, with Al-Mg-Si as an example system. The simulations are performed in the canonical ensemble, also known as the NVT ensemble, which means that the number of atoms, the volume of the simulation cell and the temperature of the system are constant.

The simulation system consists of a FCC lattice where every lattice site except one is occupied by an Al, Mg or Si atom. This FCC lattice restriction does not allow β'' to be formed. The reason for only allowing FCC sites is that the simulations will start from SSSS in FCC, and it is therefore important that the behaviour of the KMC simulations for the FCC lattice works well, before additional sites are added. Adding octahedral sites to allow β'' to form is a natural next step after this thesis work. For each iteration of the KMC simulation on the FCC lattice, the set of possible events is that one of the 12 nearest neighbours of the vacancy can jump to the vacant site. This set of events ensures that any atom can move from its current position to any other position in the lattice through a series of jumps to the vacant lattice site. It is possible to reach all possible configurations of the lattice if given enough time since any atom can occupy any site, the method is therefore ergodic. For the process to be a Markov process, the transition rates, and therefore also the calculation of the activation energies, must be time independent and only depend on the current state and the considered new state, in addition to following the KMC algorithm for choosing which event to carry out. The transition rates, see equation (2.1), also depend on the jump attempt frequency, which is assumed to be constant but can have a different value for different atom types. KMC was shown to fulfil detailed balance for the general case in subsection 2.2.1, given that the activation energies are correct.

In alloys the vacancy concentration will vary with temperature and the total number of vacancies in the real system will scale with system size. In the simulations there is only one vacancy, but this can be compensated for by dividing the escape time by the

total number of vacancies the system would have in reality. This is equal to setting $C = NC_{\mathrm{V}}$ in equation (2.4), where N is the number of atoms in the system and C_{V} is the vacancy concentration. To estimate the vacancy concentration at different temperatures it is assumed that divacancies can be neglected and the concentration of monovacancies can be written as an Arrhenius term [28]

$$C_{\mathrm{V}}(T) = \exp\left(-\frac{G_{\mathrm{V}}^{\mathrm{F}}}{k_{\mathrm{B}}T}\right), \tag{2.11}$$

where $G_{\mathrm{V}}^{\mathrm{F}}$ is the Gibbs free energy of formation for monovacancies. The free energy can be calculated from [28]

$$G_{\mathrm{V}}^{\mathrm{F}}(T) = H_{\mathrm{V}}^{\mathrm{F}} - TS_{\mathrm{V}}^{\mathrm{F}}, \tag{2.12}$$

where $H_{\mathrm{V}}^{\mathrm{F}}$ is the enthalpy of formation and $S_{\mathrm{V}}^{\mathrm{F}}$ is the entropy of formation. Both the enthalpy and entropy of formation are assumed to be temperature independent.

2.3 Existing expression for activation energies in Al-Mg-Si alloys

In my specialisation project [8] the method for calculating the activation energies was from an article by Liang et al. [1]. An introduction to this method has been included in this work, since the new method will be benchmarked against Liang et al. It will also be used to illustrate some new features of the program. This introduction is based on the introduction given in the specialisation project [8] and on the article by Liang et al. [1].

The expression for the activation energy for an atom jumping from site i to site j , E_{ij}^{act}, is in [1] given by

$$E_{ij}^{\mathrm{act}} = E_{X_i}^{\mathrm{d}} - E_{\mathrm{V}}^{\mathrm{f}} - \left(\sum_{k\in NN_j}\epsilon_{X_k\mathrm{V}} + \sum_{k\in NN_i}\epsilon_{X_kX_i}\right) + \left(\sum_{k\in NN_i}\epsilon_{X_k\mathrm{V}} + \sum_{k\in NN_j}\epsilon_{X_kX_i}\right), \tag{2.13}$$

where $E_{X_i}^{\mathrm{d}}$ is the theoretical diffusion activation energy for the atom type at site i, which is denoted X_i and referred to as the occupation of site i, and $E_{\mathrm{V}}^{\mathrm{f}}$ is the formation energy of the vacancy in pure aluminium. The subscript V is used to indicate the vacancy, NN_i is the set of lattice sites that are nearest neighbours for site i and the ϵ are interaction energies. The value for all of the variables in equation (2.13), as well as the jump attempt frequency for each atom type, is given in Table 2.1[1]. All interaction energies with aluminium have been set to zero.

The first two terms in equation (2.13) represents the activation energy for a jump in pure aluminium. The first pair of sums effectively count the number of bonds between the atom at site i and its neighbours and the vacancy and its neighbours before the jump. For the second set of sums the vacancy and the atom have switched places and represents the number of bonds after the jump. The two pair of sums will adjust the activation energy

[1]The sign of the interaction energies, ϵ, have been switched compared to the tabulated values in [1] since there seems to be a misprint in the original article. With the original signs no clustering happens, while the article clearly shows clustering in the snapshots of the system.

Table 2.1: The value for all the variables in equation (2.13) taken from the article by Liang et al. [1].

Variable	Value
ν_{Al}	$1.66 \cdot 10^{13}\,\mathrm{s}^{-1}$
ν_{Mg}	$1.86 \cdot 10^{13}\,\mathrm{s}^{-1}$
ν_{Si}	$1.57 \cdot 10^{13}\,\mathrm{s}^{-1}$
$E^{\mathrm{d}}_{\mathrm{Al}}$	$1.29\,\mathrm{eV}$
$E^{\mathrm{d}}_{\mathrm{Mg}}$	$1.27\,\mathrm{eV}$
$E^{\mathrm{d}}_{\mathrm{Si}}$	$1.15\,\mathrm{eV}$
$E^{\mathrm{f}}_{\mathrm{V}}$	$0.63\,\mathrm{eV}$
$\epsilon_{\mathrm{Mg-V}}$	$-0.015\,\mathrm{eV}$
$\epsilon_{\mathrm{Si-V}}$	$-0.025\,\mathrm{eV}$
$\epsilon_{\mathrm{Mg-Si}}$	$-0.04\,\mathrm{eV}$
$\epsilon_{\mathrm{Mg-Mg}}$	$0.04\,\mathrm{eV}$
$\epsilon_{\mathrm{Si-Si}}$	$0.03\,\mathrm{eV}$

based on the nearest neighbours for both site i and j. In Figure 3.1 in subsection 3.2 the sites included in the sum over NN_i are numbered 0-10 and the sum over NN_j includes sites 7-10 and 15-21 in the figure.

Chapter 3

Cluster expansion

This chapter will give a general introduction to cluster expansions and show how it have been adapted to calculate activation energies for KMC simulations. How to create and expand the training set, as well as fitting of the model to the training set will also be presented in detail. Cluster expansions are used to create an expansion for the value of a property of a crystalline material based on the state of set of sites. This method has become a very valuable tool for numerical simulations, especially for computing thermodynamic quantities for alloys [29]. One of the most common uses is to create a link between DFT calculations and the computation of the total energy of a system when performing Monte Carlo simulations [30]. There are numerous examples of using CE together with MC simulations on different systems listed in [29, 30], one of the uses has been to study precipitation in Al-Mg-Si alloys [22, 31]. There are fewer examples of using CE for kinetic Monte Carlo than for regular Monte Carlo, but some of the uses have been diffusion in Li_xCoO_x [32], Al-Zn alloys [16], Fe-Cr alloys [14], Ag [33], zirconia [34] and catalysts [35].

3.1 General cluster expansion

This introduction to the concept of cluster expansions are based on the article by Mueller et al. [36] and the work of Sanchez et al. [37, 38]. Cluster expansions are used to create an expansion for the value of a property of a crystalline material based on the state of set of sites. The single site state variable will in this work be the type of atom occupying the site, but other variables such as spin, used in Ising models, can also be used. There are usually a very large number of different configurations of the site variables and it would not be possible to calculate all of these using computationally intensive methods such as DFT. This is one of the main advantages of cluster expansions, they create an expression that can calculate the chosen property for any configuration of the site variables after being fitted to a training set, computed by for example DFT. Another advantage of cluster expansions is that they can be truncated to increase computational speed at the trade-off of reducing the accuracy.

It is important to note that a cluster in a cluster expansion is a set of lattice sites, and

has nothing to do with clusters forming in aluminium alloys. A cluster can consist of one or more sites and can have different shapes. To form a complete basis, a set of single site basis functions are needed. Basis function number b for site j is denoted by $\Theta_{b,j}$. To compute the value of the property of the crystalline material, F, the following expression is used

$$F(\vec{s}) = \sum_{\vec{b}} V_{\vec{b}} \prod_{j} \Theta_{b_j,j}(s_j). \tag{3.1}$$

Here s_j is the value of the site variable for site j, $\vec{s}$ is a vector of all the site variables and $\vec{b}$ contains which basis function to use for each site. The effective cluster interactions (ECI) are denoted by $V_{\vec{b}}$ and the sum is over all possible $\vec{b}$. Basis function number 0 can be set to be constant, and in this case it is set to 1. This means that to represent a single site cluster all but one element of $\vec{b}$ will be nonzero, for a pair cluster two elements will be nonzero and so on. When using $\Theta_{0,j} = 1$ the product over j in equation (3.1) only needs to be taken over elements that have nonzero value for b_j. The basis function for a cluster can then be written as

$$\Phi_{\vec{b}}(\vec{s}) = \prod_{j|b_j!=0} \Theta_{b_j,j}(s_j). \tag{3.2}$$

All symmetrically equivalent cluster functions should have the same ECI. The set of all symmetrically equivalent $\vec{b}$ can be denoted by α. By adopting the notation of α and using equation (3.2), the main equation for the cluster expansion, equation (3.1), becomes

$$F(\vec{s}) = \sum_{\alpha} V_{\alpha} \sum_{\vec{b}\in\alpha} \Phi_{\vec{b}}(\vec{s}). \tag{3.3}$$

When using cluster expansions it is common to truncate the expression to increase the computational speed at the cost of precision. Truncation lowers the number of ECIs, which is an advantage when fitting the model as the size of the training set is usually limited. One way to truncate the expression is to assume that there is a finite interaction range and remove any clusters larger than a set size [38]. It is also common to assume that the ECI for clusters decrease rapidly as the number of sites included in each cluster increases [29], therefore only clusters that have fewer sites than a set threshold are included.

The main challenges when creating a cluster expansion are often related to the creation of the training set, finding which structures to expand the training set with to get the best accuracy increase per calculation and how to fit the model parameters. A more thorough introduction to each of these challenges, as well as the proposed solutions, will be covered in separate sections later in this chapter.

3.2 Adapting cluster expansion for use in KMC

One of the main goals of this thesis is to develop a systematic and easily expandable expression for the activation energies associated with an atom jumping to a vacant position in the FCC lattice for use in KMC simulations. The expression should be able to be fitted to a training set calculated by DFT to get physically accurate activation energies. This section will show the details of how the general cluster expansion method have been

adapted for the purpose of calculating these activation energies. The reason for needing a way to go from a training set of some selected structures to a general expression is that the number of possible local environments can become very high, and DFT has a high computational cost. If only considering the nearest neighbours of site i, atom 0-10 in Figure 3.1, and only considering single site clusters, there are 2 groups of single site clusters that each has 4 clusters, 1 that has 2 clusters and 1 that only have 1 cluster. For the group with only 1 cluster there are 3 different configurations, for a group with two clusters there are 6 different configurations and for the groups with 4 clusters there are 15 different configurations. The number of configurations do not take into account the order of the atoms since this information is not included in single site clusters. With 3 different atom types that can jump, the number of different local environments becomes $3 \cdot 3 \cdot 6 \cdot 15^2 = 12150$. If all the sites in Figure 3.1 is included, and still only considering single site clusters, the number of configurations become $3 \cdot 3^2 \cdot 6^6 \cdot 15^3 \approx 4.3 \cdot 10^9$. Adding pairs or even higher terms will dramatically increase the number of configurations from this already large number of configurations.

The site variable for the cluster expansion is the occupation, X, which is the type of atom occupying the site. For the single site basis functions Kronecker delta functions have been chosen as these make it easy to see what each ECI corresponds to. The cluster functions are the product of these delta functions. The activation energy for the same local environment will be different depending on which type of atom that jumps, X_i. Due to this fact all ECIs have been chosen to depend on X_i to include this effect, even when only single site clusters are included. This results in effectively 3 separate cluster expansions for the Al-Mg-Si system, one for each atom type that jumps. Symmetry will be limited to rotational symmetry around the jump direction, which increases the number of ECIs compared to a regular CE for the total energy of a system.

The energy barriers are assumed to only depend on the local environment and the atom type that jumps, therefore only lattice sites up to a certain cutoff will be included. The selection of which sites to include have been done symmetrically around the centre of the jump to include the same sites for both the forward and backward jump. This is done to avoid introducing a bias for either of the jump directions. In this work only the nearest and next-nearest neighbours of lattice site i and j have been included due to limitations of the training set. The included sites are illustrated in Figure 3.1, where all the sites have a colour based on which group of single site clusters they belong to.

The cluster expansion based expression for the activation energy for an atom at site i to jump to the vacancy at the neighbouring site j is given by

$$E_{ij}^{act} = V^{X_i} + \sum_{\alpha} \sum_{\vec{q}} V_{\alpha}^{\vec{q},X_i} \sum_{\vec{k} \in S_{\alpha ij}} n_{\vec{q}}(X_{\vec{k}}). \tag{3.4}$$

The first term, V^{X_i}, is an effective cluster interaction (ECI) that only depends on the occupation, the atom type X_i occupying site i. This constant is equal to the activation energy for a jump in pure aluminium. The remaining ECIs, denoted by $V_{\alpha}^{\vec{q},X_i}$, are for single- and multi-site clusters and depend on the occupation of site i, X_i, a decoration vector, $\vec{q}$ and α. Each α contains a set of symmetrically equivalent clusters, the same as in the introduction to general cluster expansions, but here they need to be symmetrically equivalent with respect to the jump direction. The decoration vector is a vector with the same length as the

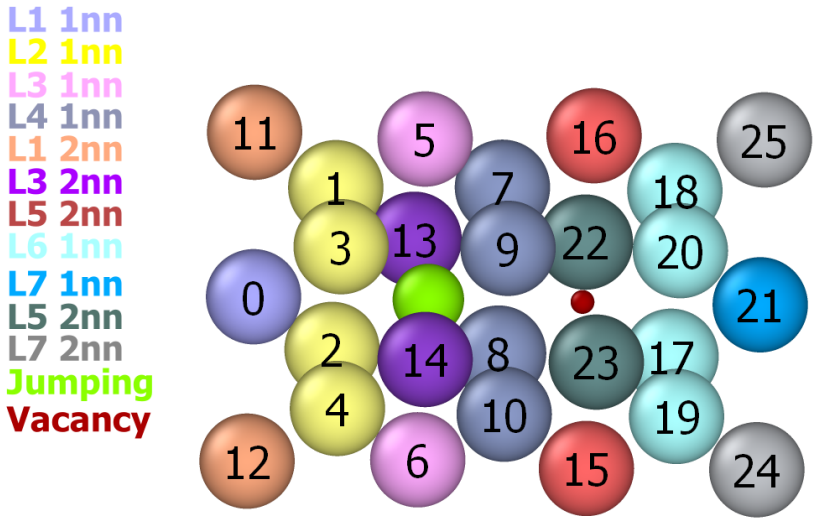


Figure 3.1: An illustration of the lattice sites included in the cluster expansion for the activation energies. The green atom is jumping along the 110 direction of the lattice to the vacant position indicated by the small red sphere. All neighbours of the same colour belong to the same group of symmetrically equivalent single site clusters. The legend at the left specifies which 110 layer the atoms of a given colour is part of, and whether it is nearest neighbour (1nn) or next nearest (2nn) to the closest one of the green atom and the vacancy. The numbering of the atoms is the order that the atom types are listed in the filenames of the output files from the DFT calculations. Figure have been provided by Research Scientist Inga G. Ringdalen at SINTEF Industry, Department of Materials and Nanotechnology.

number of sites in each of the clusters contained in α, and each element of $\vec{q}$ is either Mg or Si for the Al-Mg-Si system. The reason for not including Al is explained in the next paragraph. $S_{\alpha ij}$ contains vectors of the lattice site indices for all clusters in α for a jump from site i to site j, and $\vec{k}$ is a vector of site indices for one of these clusters. The ECI is multiplied by the sum over all $\vec{k}$ in $S_{\alpha ij}$, where the cluster function $n_{\vec{q}}(X_{\vec{k}})$ is given by

$$n_{\vec{q}}(X_{\vec{k}}) = \begin{cases} 1 & \text{for } \vec{q} = X_{\vec{k}} \\ 0 & \text{otherwise} \end{cases}. \tag{3.5}$$

Each element of the decoration vector does not need to take on the value of all the different atom types in the system for all information to be included, one atom type will always be redundant [29]. For the Al-Mg-Si system the elements of $\vec{q}$ have been chosen to only have the value Mg or Si. If the atom type does not match either Mg or Si it must be Al since there is only one vacancy in the system and the lattice site of the vacancy is not included in any cluster. If the system would have included more than one vacancy it

would be necessary for the elements of $\vec{q}$ to take on three different values instead of two, since each site now has four possible occupations. With the exclusion of Al from $\vec{q}$, the number of ECIs for a single site cluster in the Al-Mg-Si system will be 6 since X_i has three different values and $\vec{q}$ is length one and therefore have two different values. For a pair cluster there will be 12 ECIs, which reduces to 9 if the two sites that form the pair have the same distance and offset.

For a jump from lattice site i to j all clusters that are rotationally symmetric around the jump direction, $\vec{r_{ij}}$, are symmetrically equivalent clusters. For a single site cluster, a first order term, the criterion for a cluster to be part of a given α need to contain two independent coordinates since one coordinate is left free due to rotational symmetry. For a single site cluster, lattice site k, α is represented in the program by the distance from site i to site k, $D_{ik} = |\vec{r_{ik}}|$ and the projection of this distance onto the jump direction $O_k = (\vec{r_{ik}} \cdot \vec{r_{ij}})/|\vec{r_{ij}}|$, which is reffered to as offset. A two-site cluster, also referred to as a pair cluster, consisting of site k and l is represented by the distance and offset for each of the two atoms as well as the distance between them, $D_{kl} = |\vec{r_{kl}}|$. See figure 3.2 for a graphical illustration of the stored quantities for a pair descriptor.

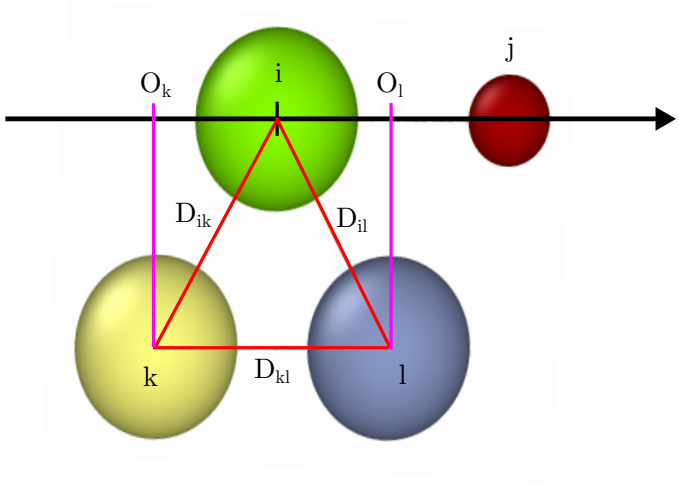


Figure 3.2: Illustration of how a pair term is represented in the implementation of the cluster expansion for activation energies for a jump from lattice site i to site j. The distances, D_{ik}, D_{il} and D_{kl}, are measured from centre to centre and the offsets, O_k and O_l, are measured along the jump direction and have positive values to the right of lattice site i. The jump direction is the 110 direction.

For clusters of size two and larger, there may exist several decoration vectors $\vec{q}$ that are symmetrically equivalent to each other. In principle it is possible to leave these as separate ECIs and the fitting procedure should be able to find out that they should have the same value, but this will need a very large training set to happen in practice. It is therefore beneficial to reduce all symmetrically equivalent decorations for a given cluster to a single ECI. For a pair cluster, the two sites can be interchanged only if both the distance and offset are equal for both sites. A pair cluster of this kind will only have one ECI for $\vec{q} = [\mathrm{Mg}, \mathrm{Si}]$ and $\vec{q} = [\mathrm{Si}, \mathrm{Mg}]$, and equation (3.5) will give 1 if $X_{\vec{k}}$ matches either one of them.

3.3 Creating the training set

The first step in the workflow for creating, training and improving the cluster expansion shown in Figure 3.3, is to calculate the activation energy for a set of structures using density functional theory (DFT). The activation energies should be calculated by an accurate method, such as DFT, to ensure that the cluster expansion is trained on physically correct activation energies. All the calculations for the training set have been performed by Research Scientist Inga G. Ringdalen at SINTEF Industry, Department of Materials and Nanotechnology. This section will therefore only give a brief overview of the utilised methods.

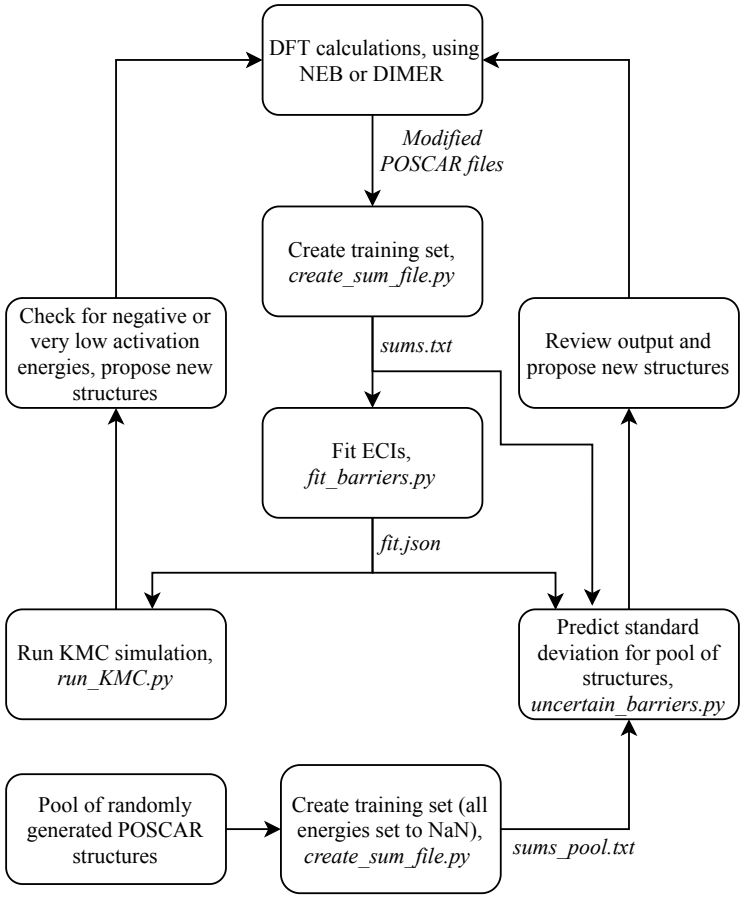


Figure 3.3: Illustration of the workflow for creating, training and improving the cluster expansion for activation energies for KMC simulations. Filenames for the Python scripts that are used for each step as well as the output filenames are included in italic.

Computing the activation energies consist of two main tasks, the first one is to calculate the energy of the initial state and the other one is to find and calculate the energy of

the highest energy point of the minimum energy path (MEP), E_{saddle} in Figure 2.6. For computing the energies DFT have been used, which calculates properties such as the energy of the system from the electron density. The field of DFT computations is large, but an introduction to the theory with comparisons to other methods, such as Hartree Fock, is given in the book by Leach [24]. DFT is often referred to as *ab initio* calculations, which, in this context, means from first principles. Despite being called *ab initio*, the exchange and correlation energy is approximated, and it is therefore important to choose a functional that is well suited for the system that is being studied [39].

To identify the saddle point for a given transition, two different methods have been used: The Nudged Elastic Band (NEB) method [40] and the DIMER method [41]. Both the DIMER and NEB methods only use the first derivative of the energy and are well suited methods to pair with plane wave formulated DFT, which is well suited for calculations on crystalline structures due to the periodicity of the system.

The NEB method finds the minimum energy path (MEP) between an initial and a final state using only first derivatives of the energy [40]. For finding the MEP, NEB creates a series of intermediate images and the force acting on each image is minimised. To ensure that the final path is continuous each adjacent image of the system is coupled with a spring term, which is the origin of the elastic band part of the name. To find the highest point of the MEP it is necessary to have enough images to get good resolution near this point. The advantage of NEB is that it does not require prior knowledge about the MEP and the result shows the energy curve for the whole transition.

The DIMER method is a method that only use first derivatives to find saddle points on high dimensional potential surfaces and was introduced by Henkelman and Jónsson in [41]. Instead of creating a series of images and finding the full MEP like NEB calculations does, the DIMER method only use two images of the system, which is the origin of the name of the method. These two images must be given a starting point and start direction. The method can be started from the initial state of the system or from an initial guess of the saddle point, which can speed up the process significantly if the guess is close. The two images are moved uphill towards the top of the barrier and the dimer is rotated on the way to calculate the lowest curvature mode for each location of the dimer, similar to the force minimisation for each image in the NEB method.

For the purpose of finding the activation energies only the energy of the initial configuration and the top of the barrier, and also the final configuration if the barrier for the reverse reaction is wanted, are of interest. By running a NEB calculation for a jump in pure Al for the different atom types, data about the position and direction of the top of the barrier can be collected. This data can be used as the starting point for the DIMER method, which can give a significant speedup compared to starting in the initial state, especially for structures that have few solute neighbours, since the energy landscape will be similar to the jump in pure Al.

3.4 Determination of effective cluster interactions

To determine the effective cluster interactions in equation (3.4) a Bayesian approach is used to be able to introduce physical insight into the fitting process. The introduction to this method is based on the article by Mueller and Ceder [36].

Bayes theorem, adapted for the fitting of ECIs, can be written as

$$P(\vec{V}|\boldsymbol{\mathcal{X}},\vec{y}) = \frac{P(\vec{y}|\vec{V},\boldsymbol{\mathcal{X}})P(\vec{V}|\boldsymbol{\mathcal{X}})}{P(\vec{y}|\boldsymbol{\mathcal{X}})}, \tag{3.6}$$

where $\vec{V}$ is a vector, of length N_{V}, that contains the value for all the ECIs in equation (3.4). The training set is represented by $\vec{y}$ and $\boldsymbol{\mathcal{X}}$. The activation energies are stored in $\vec{y}$, which has length equal to number of structures in training set, N_{s}, and the matrix $\boldsymbol{\mathcal{X}}$, size $N_{\mathrm{s}} \times N_{\mathrm{V}}$, contains the value of the rightmost sum in equation (3.4) for all ECIs for each of the structures. The denominator in equation (3.6) can be considered a constant since it is the probability of the activation energies found from DFT given the structures in the training set. The left factor in the numerator is the probability of the training set activation energies given a set of ECIs and the training set structures, which is what is maximised in a regular least squares regression. The right term is the probability of the ECIs given the training set structures, this is also known as the prior distribution and is where the physical insight of the system is introduced in the fitting process. The goal of the fitting procedure is to find the combination of ECIs, $\vec{V}_{\mathrm{optimal}}$, that is most likely given the training set, which is equivalent to maximising $P(\vec{V}|\boldsymbol{\mathcal{X}},\vec{y})$.

Maximising $P(\vec{V}|\boldsymbol{\mathcal{X}},\vec{y})$ is the same as minimising $-\ln P(\vec{V}|\boldsymbol{\mathcal{X}},\vec{y})$ since the natural logarithm is a monotonically increasing function. This can be written as

$$\vec{V}_{\mathrm{optimal}} = \min_{\vec{V}}[-\ln\left(P(\vec{y}|\vec{V},\boldsymbol{\mathcal{X}})\right) - \ln\left(P(\vec{V}|\boldsymbol{\mathcal{X}})\right)]. \tag{3.7}$$

It is assumed that each activation energy have a normal distributed error with mean 0 and variance σ^2, which gives that

$$P(\vec{y}|\vec{V},\boldsymbol{\mathcal{X}}) \propto \prod_k \exp\left(-\frac{(y_k - \vec{\mathcal{X}}_k \cdot \vec{V})^2}{2\sigma_k^2}\right), \tag{3.8}$$

where $\vec{\mathcal{X}}_k$ is row k of $\boldsymbol{\mathcal{X}}$. By inserting equation (3.8) into equation (3.7) the first term of equation (3.7) becomes

$$\sum_k \frac{(y_k - \vec{\mathcal{X}}_k\vec{V})^2}{2\sigma_k^2}. \tag{3.9}$$

The variance for each structure in the training set is unknown, but it can either be set to a constant value to weight all structures equally or it can be set individually to weight some structures more than others. This inverse variance can be stored for each structure in a diagonal weighting matrix, $\boldsymbol{W}$, of size $N_{\mathrm{s}} \times N_{\mathrm{s}}$.

The prior distribution is set by the user and should be able to include information about the value of each of the different ECIs as well as the value of one ECI relative to another ECI. This can be achieved by setting the prior to be such that $-\ln P(\vec{V}|\boldsymbol{\mathcal{X}}) = \vec{V}^T\boldsymbol{Q}\vec{V}/2$ where $\boldsymbol{Q}$ is a matrix of size $N_{\mathrm{V}} \times N_{\mathrm{V}}$. This way of representing the prior is chosen because it fulfils all the wanted requirements and gives a practical expression for the solution for $\vec{V}_{\mathrm{optimal}}$, see Appendix in [36] for a full derivation. If $\boldsymbol{Q}$ is diagonal the fitting procedure becomes the same as ridge regression, L_2 regularisation or Tikhonov regularisation [36].

Due to this similarity the matrix $\boldsymbol{Q}$ will be referred to as the regularisation matrix. The maximum likelihood estimate for $\vec{V}$ can then be calculated as [36]

$$\vec{V}_{\text{optimal}} = (\boldsymbol{\mathcal{X}}^T \boldsymbol{W} \boldsymbol{\mathcal{X}} + \boldsymbol{Q})^{-1} \boldsymbol{\mathcal{X}}^T \boldsymbol{W} \vec{y}. \tag{3.10}$$

The choice of prior distribution will affect the outcome of the fitting procedure, which can improve the fitting, but it can also make it worse if the introduced bias is not consistent with the true value of the ECIs. In the priors used for fitting the ECIs for expression (3.4) only diagonal elements of $\boldsymbol{Q}$ are used and all ECIs with the same α are treated equally. It can be difficult to estimate how to vary the elements of $\boldsymbol{Q}$ to get a good result, but the user often has some insight of what factors that could play a role. One of the insights for this system is that clusters with more sites should have a lower value than clusters with fewer sites. Another insight is that the further away the lattice sites are from either the middle of the jump or the initial site of the jump, the lesser impact they should have on the activation energy. To incorporate these insights without knowing the exact way the value should depend on these two factors, it is useful to use hyperparameters and perform a hyperparameter optimisation. Hyperparameters are parameters for the prior and are not connected to the system being analysed. The prior for the two aforementioned insights can be written as

$$Q_{\alpha\alpha} = aD_\alpha^b + d\gamma_\alpha^e + C, \tag{3.11}$$

where $Q_{\alpha\alpha}$ is a diagonal element of $\boldsymbol{Q}$, γ is the number of sites included in each cluster in α, $D_\alpha = 1/\gamma \sum_k D_{ik}$ is the average distance for the γ lattice sites in each cluster in α and a, b, C, d and e are hyperparameters.

When performing the hyperparameter optimisation, and the fitting in general, it is important that the resulting set of ECIs are as robust as possible for predicting the activation energy for structures that are outside of the training set. This is the same as avoiding overfitting of the model, which is a relevant problem for underdetermined systems. To avoid overfitting the model, the hyperparameters of the prior are optimised with regard to getting the lowest cross-validation score, which is an estimate of how accurate the model will be for structures outside of the training set. The cross-validation score can be computed by using leave-one-out cross-validation (LOOCV) or leave-multiple-out cross-validation for lower computational cost.

Cross-validation is performed by partitioning the total training set into N_p equally sized partitions, if $N_\text{p} = N_\text{s}$ the method is equal to LOOCV. One partition is set aside and not included in the calculation of $\vec{V}_{\text{optimal}}$, this is the validation partition. After calculating the ECIs, the RMSE for the structures in the validation set is computed and stored. The next step is to pick one of the other partitions as validation partition and perform the fit to the other partitions and calculate the error for the validation set. This is repeated N_p times such that each structure is used for validation a single time, and the cross-validation (CV) score is the average of all the errors that where computed for the validation structures. If there are fewer partitions than structures, leave multiple out cross-validation (LMOCV), the CV score will vary slightly depending on which structures are grouped together in the partitions. If LOOCV is used, the score will be the same each time it is computed for the same prior with the same hyperparameters since there is only 1 structure in each partition. The implementation of the fitting procedure gives the user the option to choose the number of partitions to use for calculation of the cross-validation score. After calculating the CV

score from doing multiple fits with partitioned data the final fit is performed using all the structures in the training set.

An alternative to using cross-validation to find the optimal fit is to plot the L-curve for the fitting and identify the corner of the curve [42]. An L-curve is a log-log plot of the solution norm, $||\vec{V}||_2$, against the residual norm, $||\boldsymbol{\mathcal{X}}\vec{V} - \vec{y}||_2$. A typical shape for the L-curve is shown in Figure 3.4. When performing the fitting both the residual norm and the solution norm should be kept small. If the fit minimises the solution norm it can end up with an over regularised solution which represents the data set poorly, and if the residual norm is minimised the solution can become under-regularised which gives a large solution norm and poor performance on data outside the training set. The regularisation corresponding to the corner of the L-curve is supposed to be the optimal trade-off between a small residual norm and a small solution norm [42].

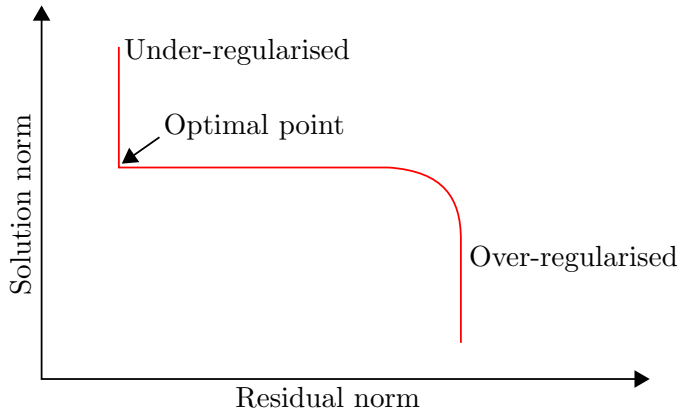


Figure 3.4: Illustration of the typical form of the L-curve plotted in log-log scale [42]. The solution norm is calculated as $||\vec{V}||_2$ and the residual norm as $||\boldsymbol{\mathcal{X}}\vec{V} - \vec{y}||_2$. The amount of regularisation applied, which is applied through $\boldsymbol{Q}$ in equation (3.10), increases from the start of the curve at upper left corner to the end of the curve. The amount of regularisation that gives the best trade-off between small residual norm and small solution norm is located at the corner of the L-curve.

3.5 Expanding the training set efficiently

DFT calculations are computationally expensive and it is therefore important to identify structures that will make the greatest impact on the fitting of the cluster expansion. How to identify which structures to expand the training set with is one of the challenges of creating a cluster expansion and there exist many different approaches.

One method used in this work is to create a large set of possible structures that are not calculated by DFT and try to estimate the uncertainty in the prediction of the activation energies. The first step in this method is to create N_{train} different training sets from the complete training set that is available. Each of these N_{train} sets are made by drawing a random structure from the complete training set and repeating this N_{s} times, each time drawing from the complete set. The next step is to perform the fitting procedure on each

of these N_{train} training sets by using the optimal hyperparameters stored in the output file of the original fitting procedure. Each model is used to predict the activation energy for each of the possible structures. Then the standard deviation for the N_{train} predictions is calculated for each structure. The assumption behind this method is that a structure where the activation energy is very sensitive to changes in the training set composition is poorly predicted by the model trained on the complete training set. These structures are therefore good candidates for expanding the training set.

Another approach is to calculate the activation energy for possible structures using the model trained on the complete training set and inspecting if any of the values are abnormally high or low. Structures that are predicted to have negative activation energies are good candidates for expanding the training set since the prediction must either be wrong or the state is not stable since it is not in a local minima, either way it contains valuable information about the system. From testing it seems like the structures that are predicted to have negative barriers are also predicted to have amongst the highest standard deviations using the other method.

It is important that the activation energy for the forward and backward reaction match the energy difference between the two states, see Figure 2.6 for an illustration of the energies. A third method to expand the training set that focuses on uncovering pairs of activation energies that match poorly is to run a KMC simulation and log the difference in energy for each iteration using an observer and write the lattice configuration to file every iteration. Then the energy difference between the states can be calculated using a regular cluster expansion for the whole lattice and compare it to the logged energy difference. This requires that the user have access to a regular cluster expansion for the whole lattice that is trained for the same system as the one being studied. This is also possible to do on structures generated manually or randomly as long as pairs of initial and final structures are produced.

Chapter 4

Implementation

During the work of this master thesis a lot of the time that was not spent on the physics of the problem have gone into implementing new features, making the code more user friendly and make it easier to maintain and expand. This have been done through extensive use of version control, restructuring the code to become more modular, creating good documentation and implementing automated tests. The first section of this chapter will go through all the smaller changes to the existing code and new observers. The second section will go through how the most important new feature, the cluster expansion based way of calculating the activation energies, have been implemented to achieve high computational speed and low memory usage. The last section will go through the newly created Python interface, which makes the program easier to use without sacrificing performance. For user manuals for running the code, as well as documentation of the Python interface, see Appendix B. The chosen standard format for representing lattices in the project are the POSCAR format from the Vienna Ab initio Simulation Package (VASP). Support for other formats can easily be added by making a small Python script.

4.1 Implementation of the KMC algorithm

The goal for the implementation of the KMC algorithm have been to create a computationally fast code, to be able to reach long timescales, as well as making the code modular to make it easy to implement new features and configure the code to the user's needs. For computational speed the algorithm has been implemented in C. The original codebase was not modular in design, but during my specialisation project the logging was made modular by introducing the concept of observers [8]. In Figure 4.1 a flowchart of the implemented KMC algorithm is shown.

4.1.1 Introduction to observers

The goal of running a KMC simulation is usually to study the time evolution of one or more quantities or record certain events. A simulation often consists of billions of iterations and

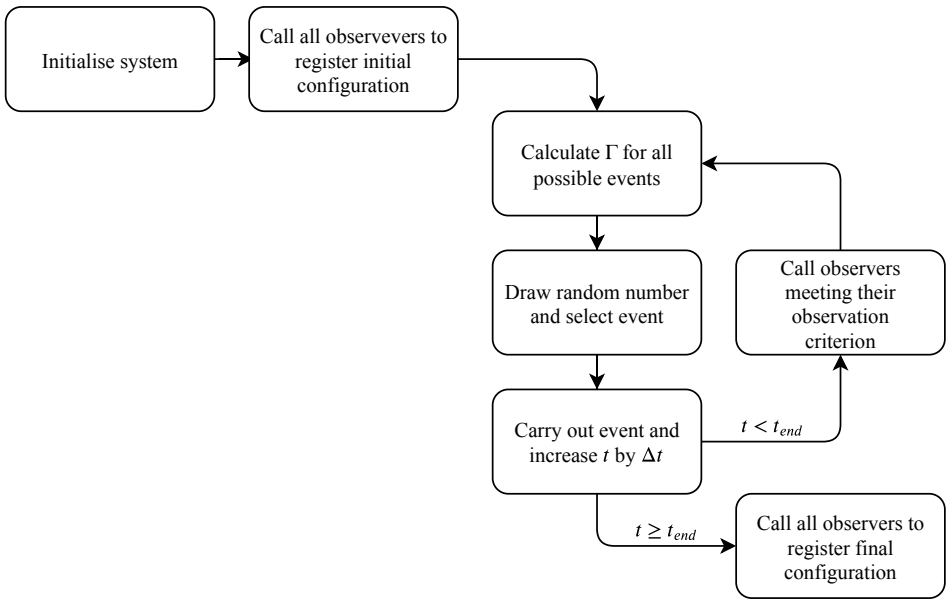


Figure 4.1: Program flow in the implementation of KMC. Observers are functions used for logging different kinds of data during the KMC simulation, which makes it easy to setup a logging scheme that fits the user's needs. The figure is a reprint from my specialisation project [8].

it is therefore important that the user can easily setup what to log and when to log it. As an example, if the user wants to log a double precision number and decides to store it every iteration for a simulation with 1 billion iterations the disk space used will be 8 GB. The chosen solution for getting a modular logging scheme is called observers and was thoroughly introduced in the specialisation project [8], but a short introduction will be given here. All observers are based on the observer struct, see listing 4.1, which ensures that the KMC code has a consistent way of interacting with all the different observers. Through setting the variables `use_time`, `factor` and `fixed_offset` it is possible to adjust how often the observer should be called to log. If `use_time` is positive the logging is based on the time in the KMC simulation, otherwise it is based on the iteration count. If the user wants to log with fixed interval `factor` is set to unity and `fixed_offset` is set to the wanted quantity. For logarithmically spaced logging `fixed_offset` is set to 1 and `factor` is set to larger than 1. More details on setting the logging parameters as well as an overview of the available observers is given in Appendix B.2.

```
1 struct observer{
2     /* When observe function should be called next. */
3     uint64_t next_obs;
4     /* Counter that keeps track of how many times the
5        observe function has been called. Starts at 0. */
6     uint64_t called;
7
8     /* All variables below are kept constant after
9        initialisation. */
```

```
    /* Fixed integer added to next_obs each time observe
       is called */
    uint64_t fixed_offset;
    /* Factor used for updating next_obs logarithmically,
       multiplies next obs with factor and rounds down
       to an integer. */
    double factor;
    /* Bool to determine if observer should use time or
       jumps to determine when to call observe.
       0 corresponds to iteration number,
       positive integer corresponds to time. */
    int use_time;
    /* Function pointer to the observe function. */
    void (*observe) (KMC *kmc, observer *self);
    /* Pointer to a struct which contain all data the
       observer need. */
    void *data;
    /* Frees all data for the observer. */
    void (*free) (observer *self);
};
```

Listing 4.1: Declaration of the observer struct.

4.1.2 Total energy logging

An observer for logging of the total energy of the system relative to the start configuration have been implemented. The only information available for the KMC code are the activation energies, the change in total energy must therefore be calculated as the difference between the activation energy for the forward and backward reaction. For an iteration where the system goes from state i to state j, or an atom jump from site i to site j, the change in total energy is

$$\Delta E_{ij} = E_{ij}^{\text{act}} - E_{ji}^{\text{act}}. \tag{4.1}$$

To be able to keep track of the total energy it must be updated every single iteration, which means that the value for the logging variables in the observer struct must be set accordingly. The energy observer therefore have a struct stored in the `data` field of the observer where a separate version of `use_time`, `factor`, `fixed_offset` and `next_obs` is stored to be able to set when the total energy should be written to file. Each row in the output file contains the iteration number and the value for the total energy.

In addition to making it easy to visualise the time evolution of the total energy of the system the energy observer also opens up for checking if the energy difference calculated from the barriers are consistent with other more precise methods. The first step is to enable both the total energy observer and the POSCAR observer, which writes the current lattice configuration to file with POSCAR format, and set them to observe at the same time. The total energy of the lattice configurations given in the POSCAR files can then be calculated by for example a cluster expansion for the total energy of a lattice. If the logging is performed every iteration it is possible to find particularly bad activation energy pairs, otherwise it will just give an indication of the overall error.

4.1.3 Update on logging of clustering rates

During the specialisation project [8] an observer for logging condensation and evaporation rates was implemented. The condensation rate, denoted as β_n, is the rate at which a cluster of size n receives a single solute atom and becomes size $n+1$. Evaporation rate, α_n, is the rate for a cluster of size n to lose a single solute and become size $n-1$. These rates are calculated as the inverse of the time between the last size changing event of a cluster and the current size changing event, and can be used as input for cluster dynamics (CD). For an introduction to the concept of CD see Appendix A. In the current implementation of this observer clusters only consist of solutes and only solutes that are NNs are considered to be in the same cluster. This observer has been updated to track the composition of each cluster and include it in the output file as well as log which atom type that joined or left the cluster. From this data it is possible to study the composition of clusters and the effect that cluster composition has on the clustering rates for the different solute types in the system.

4.1.4 Minor updates

A bug was found and corrected in the code for logging the diffusivity of the different atom types. The diffusivity for atom type X is given by [24]

$$D_X = \lim_{t\to\infty} \frac{\langle |\vec{r}_X(t) - \vec{r}_X(0)|^2 \rangle}{2t\mathcal{D}}, \tag{4.2}$$

where the average is taken over all atoms of the same atom type, $\vec{r}(0)$ is the initial position of the atom, $\vec{r}(t)$ is the position after time t and $\mathcal{D}$ is the dimensionality of the system, which in this case is $\mathcal{D} = 3$. The bug was related to the averaging in equation (4.2), the code checked atom type at site i of the lattice but calculated the diffusion distance for the atom that originally started at site i instead of the one that is currently at site i. This effectively led to that the average that should be over only atoms of type X included a random set of atoms instead.

Other minor improvements have been made to the KMC code, mainly to accommodate the newly implemented cluster expansion based way of calculating the activation energies. The neighbourlist have been updated to include any neighbour up to a user defined radius instead of only including nearest neighbours. Code for setting up an empty lattice and add sites manually have been implemented to make it easier to setup lattices from POSCAR files. Since these functions already are available adding support for initialising lattices from other file formats, such as XYZ, can be done by just writing a small Python script. The main KMC struct has also gotten two new function pointers, one for the rate calculation and one for the activation energy calculation. Since the rate and energy calculators might need some data, the KMC struct has also gotten a data pointer that can be used by these two functions. The pointer to the activation energy calculator is needed for the total energy observer and the rate calculator pointer makes it very easy to change between different calculators when setting up the simulation.

4.2 Implementation of cluster expansion adapted for use in KMC

The implementation of the cluster expansion based way of calculating the activation energies, introduced in subsection 3.2, have to be fast to calculate and it should be easy to adjust which sites and terms to include in the expansion. To be able to make the evaluation as fast as possible it is important that the program is able to know which lattice sites that makes up each cluster for each i and j without having to do any searches or calculations. In this subsection it will be shown how this is achieved through having a sorted neighbourlist, an orientation matrix and site groups.

Each site in the lattice has an array of indices to neighbouring sites, which in the previous version of the program only contained nearest neighbours. In the new implementation the neighbourlist includes all sites that are within a user-defined distance. The neighbourlist for each site is then sorted first on distance between the the neighbour and the central atom, then on relative x, y and z coordinate. All the sorting is performed with a tolerance since the coordinates are floats. This ensures that neighbour number k in the neighbourlist of a site always have the same relative position to the original site for all sites in the lattice. The program needs a way to know which neighbours it can jump to, which in the current version is the nearest neighbours sites. This is done by setting an integer n_{nn} that indicates that the first n_{nn} neighbours are nearest neighbours and can be jumped to, which is why the neighbourlist is sorted on distance first.

The expression given in equation (3.4) use groups of clusters that are symmetrically equivalent relative to the jump direction. This means that which neighbours in the neighbourlist that is in each cluster will be different for each of the n_{nn} jump directions. This problem is solved by making an $\mathrm{orientation\_matrix}$ that maps the neighbourlist in such a way that the remapped neighbourlist is always sorted the same way relative to the jump direction. To accomplish this, the $\mathrm{orientation\_matrix}$ must have n_{nn} rows and n columns, where n is the number of neighbours in the neighbourlist. Since all sites have the same order for their neighbourlist, the program only need one instance of the $\mathrm{orientation\_matrix}$, which saves a lot of memory usage compared to each site having to store n_{nn} versions of the neighbourlist.

Row k of the $\mathrm{orientation\_matrix}$ contains the mapping of the neighbourlist for a jump to neighbour number k in the neighbourlist. The first step in setting up row k is to calculate the vector from the initial site, site i, to neighbour number k in the neighbourlist of site i. Then a rotation matrix R, which rotates the system such that the vector for the jump direction is mapped to the x-axis, is made using Rodrigues rotation formula, see [43] for details on the derivation of the formula. The Rodrigues rotation formula effectively rotates the system π radians around a vector in the middle between the x-axis and the jump direction. This rotation is not sufficient to ensure that the system is oriented the same way each time since the rotation around the x-axis will not be the same after the initial mapping. Another rotation matrix which rotates the system around the x-axis such that atom number 5 and 6 in Figure 3.1 are in the xz-plane is therefore applied. The next step is to calculate and sort the neighbours according to distance and relative coordinates in the rotated system. If a site is neighbour number l in the rotated neighbourlist, for row k, and neighbour number m in the original neighbourlist, then $\mathrm{orientation\_matrix}[k][l] =$

m.

For each group of clusters, α, a struct called `site_group`, shown in listing 4.2, is allocated and initialised. The `index_array` stores the index to all the elements of the neighbourlist that are part of each cluster in the group. The number of terms in the sum for a given site group is `n_sites/order`. To describe the criteria for a cluster to be part of the `site_group` a struct called `descriptor`, see listing 4.3, is used. This struct contains a `type` field to indicate the type of descriptor, a free function, a void pointer, `data`, to store the data needed for the descriptor and a `get_str` function which returns a string with all the information about the descriptor. The struct also contains a `check` function which checks if a given combination of neighbours fulfils the descriptor. The remaining fields of the `site_group` struct, `n_coeffs`, `coeffs`, `atomcombs` and `n_equivalent`, are used to store the effective cluster interactions together with all the symmetrically equivalent decoration vectors and the occupation of site i. Row k of `atomcombs` corresponds to the ECI `coeffs`$[k]$ and have length `n_equivalent`$[k]\cdot$`order` $+1$. The first column contains the atomic number for X_i and the remaining columns stores the atomic numbers for all the equivalent decoration vectors.

```
/* Contains all data for a group of sites defined by the descriptor. */
typedef struct _site_group {
  /* Stores an array of indices for all clusters included in this group.
     The indices are the position of the lattice sites in the neighbour
     list of site i.
  The list is stored as a 1D array, so if the order is for example 2, then
     element 0 and 1 is the first cluster, 2 and 3 is the second cluster
     and so on.*/
  int *index_array;
  /* Number of elements in index_array. */
  int n_sites;
  /* Number of elements allocated for index_array. */
  int size;
  /* Order of the term, the number of lattice sites in each cluster.*/
  int order;
  /* Pointer to a descriptor that describes the conditions to be a part of
     this site_group. */
  descriptor *descriptor;

  /* Number of stored coefficients for this site_group. */
  int n_coeffs;
  /* Stores coefficients. */
  double *coeffs;
  /* Stores atomic numbers for use with coefficients.
  It is a 2D arrray with n_coeffs rows and varying number of columns,
  number of columns for row k is 1+n_equivalent[k]*order.
  The first column is the atom type that jumps, the remaining are all
  equivalent atom combinations for the neighbours. */
  int **atomcombs;
  int *n_equivalent;
}  site_group;
```

Listing 4.2: Declaration of the site_group struct.

```
/* General descriptor struct to have a consistent way of freeing and
    checking different descriptors. */
typedef struct descriptor descriptor;
```

```
struct descriptor{
  /* String describing the type of descriptor */
  char *type;
  /* Free all data for descriptor and the descriptor itself. */
  void (*free)(descriptor *self);
  /* Number of neighbours that need to be in the neighbours list sent to
     the check function. */
  int n_neighbours;
  /* Check if neighbours fulfills the requirements of the descriptor */
  int (*check)(const descriptor *self, const Lattice *lattice, int i, int
     j, int *neighbours);
  /* Function to get string with info about descriptor. */
  char *(*get_str)(const descriptor *self);
  /* Pointer to all data needed by descriptor. */
  void *data;
};
```

Listing 4.3: Declaration of the descriptor struct.

To store all the data for calculating the activation energies during a KMC simulation the `Eact_data` struct is used, see listing 4.4. This stores a list of `site_group` pointers, an instance of the `orientation_matrix`, the number of different elements in the lattice and the jump attempt frequency for each element. When calculating the sum for a `site_group` for a jump from site i to site j, which is neighbour number k for site i, the program finds the site index corresponding to element l in the `index_array` in the following way `si.neighbours[orientation_matrix[k][index_array[l]]]`, where `si` is a pointer to site struct for site i. This implementation with the presented structs fulfil the goal of being easy to configure, fast to evaluate and is very memory efficient.

```
typedef struct _Eact_data {
  /* Array containing all site_groups for the potential. */
  site_group **site_groups;
  /* Number of site_groups. */
  int n_groups;
  /* Allocated size of site_groups. */
  int size;
  /* Number of different elements in the lattice, excluding the vacancy.
     */
  int n_elem;
  /* Array of length n_elem that contains the jump attempt frequency for
     the different elements. Same order as lattice->Z. */
  double *jump_freq;
  /* An array with size n_nn X n_neighbours.
     The first dimension is the number of jump possibilities for an atom,
     the second dimension is the number of elements in the neighbour list
     of the atom.
     The array contains indices to elements in the neighbour list.
     If an atom jumps to site number j in the original sites neighbourlist,
      row j of this array should be used when accessing elements.
     neighbours[orientation_matrix[j][other_neighbour_index]]. */
  orientation_matrix *orientation_matrix;
} Eact_data;
```

Listing 4.4: Declaration of the Eact_data struct.

4.3 Python interface

During the work of this master thesis the KMC code and the newly implemented cluster expansion based code have been interfaced to Python to make it easier to setup simulations and access C functions from Python, see Appendix B.5 for documentation of the interface. Without the interface, the code had to be recompiled if any change to the simulation setup was needed, except for a few settings that was available from the command line. The KMC simulations still operate entirely in C after the setup has finished and the simulation has been launched from Python, there is therefore no noticeable impact on the performance of the code.

Another important reason for interfacing the C code to Python was to avoid having any duplicates of code related to the cluster expansion. It is crucial that the sums in equation (3.4) are calculated the same way when creating training sets as during the KMC simulations. It is therefore better to make an interface and use the same code for both instead of having one C version and one Python version. Another benefit is that it makes setting up the wanted combination of site groups when creating the training set easier as it is done from Python. After the combination of site groups have been set during the creation of the training set, the setup of site groups and ECIs are done automatically from the information in the output file of the fitting procedure. The setup procedure also checks that the correct number of sites was added to each site group.

The KMC simulations can now be started from a POSCAR file instead of a random configuration by specifying the filename as a command line option. This can for example be used to restart a simulation from the last logged POSCAR if a simulation was aborted or if the user decides to run the simulation longer after the original simulation have finished. It has also proven to be very useful to check if a given lattice configuration, for example one with large clusters that are known to be stable from other studies, is stable in the KMC simulation at different temperatures. This would not be possible to check without this option since there might be situations where a random start configuration does not lead to any clustering due to small clusters being unstable while a large cluster is stable.

The interface has been created using Simplified Wrapper and Interface Generator (SWIG)[1], which allows structures to be extended to become class like objects. Most of the key structs in the program have been extended, for a detailed overview of the interface see appendix subsection B.5. One challenge of interfacing C code to Python with SWIG is memory handling. Memory allocated in C can be exposed to Python, which is very powerful since it allows the memory to be modified in place from Python, but can give problems related to ownership and deallocation of the object. This can be solved by reference counting, i. e. incrementing and decrementing counters manually that indicate if the memory is in use by another thread. To minimise the risk of memory related problems the interface uses *get* functions, that return a copy of the object, and *set* functions instead of exposing the memory allocated by C.

[1] See swig.org for more information

Chapter 5

Results and discussion

This chapter will first present and discuss the DFT results, then different fitting strategies will be compared and a final fit to use for the rest of the chapter will be chosen. The next part of the chapter will do an in-depth inspection of both the cluster expansion based way of calculating the activation energies and the method by Liang et al. [1]. The two methods will be benchmarked against DFT calculations, stress tested on a pool of different structures and the energy difference between states, calculated from pairs of activation energies, will be compared to energy differences calculated from a CE of the total lattice. Runtime for the different methods will also be included. Simulation results, including monomer concentration, average cluster size, diffusivities and total energy evolution, will be shown and discussed for two different runs at different temperatures for both methods. The new features of the updated clustering rates observer will be shown for an average over many realisations of the system using the activation energy expression from Liang et al. The last section will go through error sources and discuss the overall results.

For all KMC simulations the following settings, which are mostly the same as the ones used in [1], are used, except for cases where other values are stated explicitly. The simulation system consists of $25 \times 25 \times 25$ face centred cubic (FCC) unit cells, corresponding to $62\,500$ lattice sites. Periodic boundary conditions are used since the goal is to study precipitation processes in bulk material. Lattice spacing is set to $a = 4.05\,\text{Å}$, this value does not affect the physics of the system since the calculation of the transition rates do not depend on this parameter, but it will affect diffusivities and other quantities that use the spacing or the volume of the system. All lattice sites except one, which is left vacant, are occupied by an Al, Mg or Si atom. All presented runs use the default alloy of the program that contains $0.67\,\text{at.}\%$ of Mg and $0.77\,\text{at.}\%$ of Si, which is close in composition to the lean 6060 alloy without any trace elements. This is the same composition as used in the article by Liang et al. [1]. Since the simulations are only of the early stages of precipitation the default temperature is room temperature $T = 293.15\,\text{K}$, which corresponds to natural ageing.

The vacancy concentration of the system is not related to the actual vacancy concentration of the simulation system, since the simulation always contain just one va-

cancy, but only used to scale the time evolution as explained in subsection 2.2.3. In the presented results for the cluster expansion based transition rates the vacancy concentration have been calculated using equations (2.11) and (2.12). The value for the enthalpy of formation for a vacancy is set to $H_{\mathrm{V}}^{\mathrm{F}} = 0.66\,\mathrm{eV}$ and the corresponding entropy is $S_{\mathrm{V}}^{\mathrm{F}} = 1.6k_{\mathrm{B}}$ [28]. These values are equilibrium values and give a vacancy concentration of $C_{\mathrm{V}}(T = 293.15\,\mathrm{K}) = 2.23 \cdot 10^{-11}$ at room temperature. For all results for the method by Liang et al. the vacancy concentration have been set to the one used in the article [1], which is $C_{\mathrm{V}}(T = 293.15\,\mathrm{K}) = 1.41 \cdot 10^{-4}$. For calculating the vacancy concentration for other temperatures for the runs with the Liang method it have been assumed that the Gibbs free energy G is temperature independent since they have not stated in the article which values for $H_{\mathrm{V}}^{\mathrm{F}}$ and $S_{\mathrm{V}}^{\mathrm{F}}$ they used to get this value[1]. The vacancy concentration will be discussed in subsection 5.6.2

The jump attempt frequencies have been set to the values given in Table 2.1, which are the same as used in the article by Liang et al. [1] and are originally from [44]. Corresponding jump frequencies were calculated in [45] for Al self-diffusion and Si diffusion in Al using transition state theory and first principles molecular dynamics. They found ν_{Al} to be between $6 \cdot 10^{12}\,\mathrm{s}^{-1}$ and $3.8 \cdot 10^{13}\,\mathrm{s}^{-1}$ and ν_{Si} was found to be between $4 \cdot 10^{12}\,\mathrm{s}^{-1}$ and $5 \cdot 10^{12}\,\mathrm{s}^{-1}$. The values seem to be in the correct order of magnitude and the transition rates given by equation (2.1) only scale linearly with the jump attempt frequency.

5.1 DFT calculations of activation energies

The activation energies for the training set have been computed using DFT together with the NEB or DIMER method for locating the saddle point. For performing the DFT calculations the Vienna Ab initio Simulation Package (VASP) have been used [46, 47]. All the calculations presented in this section have been performed within the SumAl project by research scientist Inga Gudem Ringdalen at SINTEF Industry, Department of Materials and Nanotechnology.

The DFT calculations provided by Inga have been performed at the PBE-GGA level [48] with a plane wave energy cut-off of $400\,\mathrm{eV}$. Convergence criterion for the total energy was set to $1 \cdot 10^{-5}\,\mathrm{eV}$, force convergence was set to $0.01\,\mathrm{eV}$ during ionic relaxation and a Gamma sampling of 0.08 k-point per Å was used to model the Brillouin zone. The simulation cell for the DFT calculations consisted of $3 \times 3 \times 3$ FCC unit cells, with periodic boundary conditions and lattice parameter $a = 4.0396$ Å. All the calculated structures only have one vacant site out of the 108 lattice sites, to be consistent with the situation in the KMC simulations. The atom positions have been relaxed when calculating the energy, but the volume of the simulation cell have been kept constant. For all the calculated structures all atoms in the simulation cell that are not shown in Figure 3.1 have been set to Al. This unfortunately means that the set does not contain any situations where the vacancy is close to a larger cluster, which will affect the jumps in the DFT calculations.

For most of the calculations the DIMER method has been used for finding the saddle point as this method is faster than NEB due to calculating fewer images of the system. This method have proved itself to be good for structures with few solute neighbours where the

[1] They have given the values $H_{\mathrm{V}}^{\mathrm{F}} = 0.67\,\mathrm{eV}$ and $S_{\mathrm{V}}^{\mathrm{F}} = 0.7k_{\mathrm{B}}$, but these values inserted into equation (2.11) and (2.12) does not give $C_{\mathrm{V}}(T = 293.15\,\mathrm{K}) = 1.41 \cdot 10^{-4}$, which is the value they state was used.

energy landscape is similar to the one for a jump in pure Al, which means that the initial guess based on NEB simulation in pure Al is close to the real position of the saddle point. The DIMER method struggled for some of the structures with many solute neighbours where the energy landscape is more complex. NEB have therefore been used on structures that have many solute neighbours and on structures where DIMER was attempted but failed.

The calculated activation energy for the structures in the training set are shown in Figure 5.1 and the maximum, minimum, average and pure Al activation energy are listed in Table 5.1. The black horizontal lines in the figure show the activation energy for a jump in pure aluminium. Many of the calculations have both the forward and backwards barrier to ensure that the information about the energy difference of the two states are available to the CE. It is relatively cheap to obtain the backwards barrier when the forward barrier is known since only the final state needs to be calculated when E_{saddle}, defined in Figure 2.6, is already known.

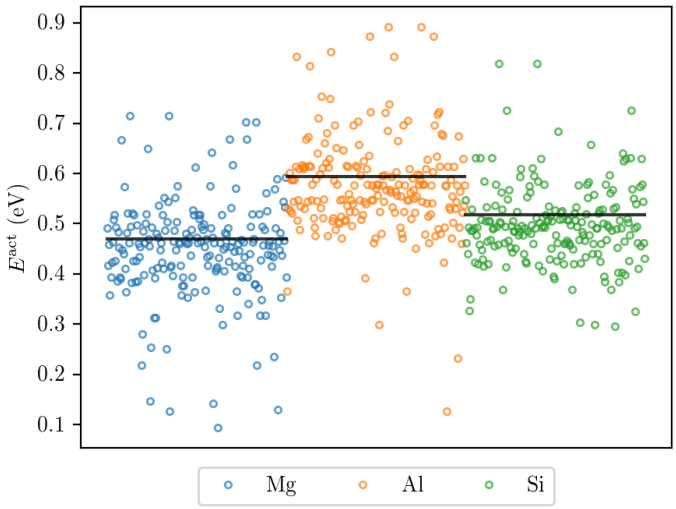


Figure 5.1: The calculated activation energy for jumping from one lattice site to a vacant nearest neighbour site. The type of atom that jumps, X_i, is indicated by the colour of the marker and the black horizontal line indicates the activation energy for a jump in pure Al. The different calculations are distributed along the horizontal axis to improve viewing clarity.

The quantity referred to as activation energy, energy barrier or E^{act} in this thesis is often referred to as the migration barrier or ΔH_m in the literature. Migration barriers for Mg and Si in pure Al as well as Al self-diffusion have been calculated by Mantina et al. using DFT with the local density approximation [44]. The migration barriers for Al self-diffusion and Si in pure Al have also been studied by Løvvik et al. where they used first principles molecular dynamics and transition state theory at two different levels of precision [45]. All of these values are listed in Table 5.1 together with the values found by

DFT in this work. The barriers found in this thesis are slightly higher for Mg and Al and slightly lower for Si compared to the values by Mantina et al. For the results of Løvvik et al. the range for both Si and Al include the results found in this work.

Table 5.1: Maximum, minimum and average of the activation energies calculated by DFT presented in Figure 5.1, all quantities are given in eV. The activation energy for a jump in pure Al found in this thesis, indicated by the black lines in Figure 5.1, are listed in the pure Al column. Literature values by Løvvik et al. [45] and by Mantina et al. [44] for jumps in pure Al are listed in the two rightmost columns.

Jump type	Min	Max	Average	Pure Al	Løvvik et al.	Mantina et al.
Mg	0.093	0.715	0.449	0.470	-	0.42
Al	0.127	0.892	0.575	0.594	0.49-0.62	0.58
Si	0.296	0.818	0.497	0.518	0.42-0.55	0.55

In Table 5.2 the number of structures in the training set for each jump type is listed together with how many of the numbered sites in Figure 3.1 that are solutes. The Table shows that most of the structures only have 1 or 2 solute neighbours. This is suboptimal for the purpose of training the CE since only a few of the rightmost sums in equation (3.4) will be nonzero for each structure. The reason for having many structures with few solutes is that these were calculated early in the project when there was free capacity at the utilised supercomputers and the CE was not ready to test fitting yet. The structures with more solute neighbours have taken more work since the potential energy landscape deviates more from the jump in pure Al than the structures with few solutes. Some of the structures have therefore been calculated with NEB instead of DIMER. Getting more calculations with more solutes have also been difficult both due to calculation problems and a lack of available computational resources[2].

Table 5.2: The leftmost column in the table is the total number of solutes occupying the numbered lattice sites in Figure 3.1, and the three remaining columns are how many structures there are in the training set, presented in Figure 5.1, that have the given number of solutes in the structure for Mg, Al and Si jumps.

Number of solutes	$X_i = \mathrm{Mg}$	$X_i = \mathrm{Al}$	$X_i = \mathrm{Si}$
0	1	1	1
1	22	24	24
2	105	104	105
3	4	4	4
4	9	8	8
5	6	4	4
6	0	2	1
7	2	2	2
8	4	4	4
9	0	0	2

[2]The utilised supercomputers were filled up with computations from Norwegian Institute of Public Health for a longer period due to the current world pandemic

5.2 Pool of possible structures

To better visualise the spread of the activation energies and test the limits of the different methods, a pool of possible structures with a wide compositional span have been created. Each structure in the pool is made by starting with Al matrix and then for each of the numbered positions shown in Figure 3.1 a random atom type, Al, Mg or Si is assigned to the site. To capture a wide variety of structures and try to find the outer limits of the activation energies, the set includes many structures and the probability for each atom type was varied. The detailed settings used for producing the pool of structures are listed in Table 5.3. The pool includes the same amount of structures for each type of atom that jumps, and the field # structures in the table is per jump type.

Table 5.3: Table show parameters used for generating the pool of random structures used for testing different methods for calculating the activation energy. The # structures column is the number of structures made for each of three atom types that can jump. Each numbered site in Figure 3.1 is set to Mg with probability P(Mg), Si with probability P(Si) and Al with P(Al).

# structures	P(Mg)	P(Al)	P(Si)
1000	0.10	0.80	0.10
1000	0.20	0.60	0.20
1000	0.33	0.33	0.33
500	0.30	0.60	0.10
500	0.10	0.60	0.30
500	0.50	0.00	0.50
1	0.00	1.00	0.00

5.3 Fitting of expression to the training set

The training set, presented in section 5.1, have 444 nonzero columns in $\boldsymbol{\mathcal{X}}$, out of 822 total, when pairs with pair distance up to 4th NN are included. All the columns corresponding to single site clusters are nonzero, which means that there are 378 ECIs for pair clusters that do not appear in any of the structures in the training set. All the ECIs that correspond to an empty column in $\boldsymbol{\mathcal{X}}$, as well as the column itself, are removed before the fitting procedure begins. The fitting procedure have been done by using two different priors, and for one of the priors both the CV score method and the L-curve method have been used to optimise the ECIs. The three different combinations of prior and optimisation method are listed in Table 5.4, from now on they will be referred to as fit I, II and III. The table also shows the value for the hyperparameters that gave the best fit according to the chosen optimisation method. For fit I and II the hyperparameter was varied between $1 \cdot 10^{-20}\,\mathrm{eV}^{-2}$ and $5\,\mathrm{eV}^{-2}$, represented by 300 log-spaced values. For fit III the initial fit started with a wide allowable range for each of the hyperparameters, and 5000 random combinations were tested. The second fitting, used as the final result, was performed the same way as the initial fit with a narrower range of hyperparameter values, centred around the optimal values from the initial fit. The range for b and e for the final fit was the integers

from 0 to 5, for d it was 100 logarithmically spaced values from $1 \cdot 10^{-18}\,\mathrm{eV}^{-2}$ to $1\,\mathrm{eV}^{-2}$ and for a and C it was 50 log-spaced values from $1 \cdot 10^{-20}\,\mathrm{eV}^{-2}$ to $1 \cdot 10^{-10}\,\mathrm{eV}^{-2}$.

Table 5.4: The table shows the three different fitting methods used to fit the ECIs in equation (3.4) to the dataset introduced in section 5.1. The method is how the fitting have determined the optimal regularisation and the values for $Q_{\alpha\alpha}$, which are used in equation (3.10) and are the same as the prior, are given in eV^{-2}. For explanation of the variables in the expression for $Q_{\alpha\alpha}$ for fit III see equation (3.11).

Fit	$Q_{\alpha\alpha}$	Method	Optimal $Q_{\alpha\alpha}$
I	C	LOOCV	$2.05 \cdot 10^{-2}$
II	C	L-curve	$3.46 \cdot 10^{-9}$
III	$aD_\alpha^b + d\gamma_\alpha^e + C$	LOOCV	$1.05 \cdot 10^{-19} D_\alpha^1 + 1.52 \cdot 10^{-2}\gamma_\alpha^2 + 6.55 \cdot 10^{-20}$

5.3.1 L-curve criterion compared to CV minimisation

The L-curve for fitting the ECIs with the constant prior, fit I and II, is shown in Figure 5.2. Fit I is shown as the blue dot and fit II is shown as a black dot and the inset plot is zoomed in on the corner of the L. The CV optimised fit has around 7 orders of magnitude higher regularisation than fit II, but fit I is still relatively close to the corner and hard to separate from the black dot in the full-size plot. Since fit I has more regularisation than fit II it leads to fit I having a slightly higher residual norm and slightly lower solution norm compared to fit II, as can be seen from Table 5.5.

The L-curve method can be a lot faster than the CV method since the CV calculation requires the same amount of fits as the number of partitions that the training set is split into for each of the hyperparameter combinations. For LOOCV the number of fits performed per hyperparameter combination is equal to the number of structures in the training set. The L-curve only requires one fit performed on the complete training set per hyperparameter combination since only the residual and solution norm needs to be calculated.

The L-curve also have some disadvantages, the most important one is the problem of identifying which point that is the corner of the ”L”. It is possible in many cases to determine the corner by visual inspection, but this is not a robust method for automated fitting. If the L-curve is continuous, the criterion is that the corner is where the curvature of the L-curve has it maximum. This can be numerically approximated for the case of fit I and II, but for fit III where there are more than one hyperparameter and the value of $Q_{\alpha\alpha}$ is not the same for all ECIs, it is not well defined how to approximate the curvature. The problem of identifying the corner of a discrete L-curve in a robust way is discussed in [49], where a two-step pruning algorithm is proposed as the best solution. This solution has not been implemented since the CV method is already implemented, is robust and gives similar results as the L-curve. If considerably larger training sets and/or more ECIs are included, then it should be considered to implement this method to save time when fitting the ECIs. It is also possible to save a lot of time by using fewer partitions for the initial fitting when using the CV criterion, and then do a second fitting with a narrower range for the hyperparameters with more partitions.

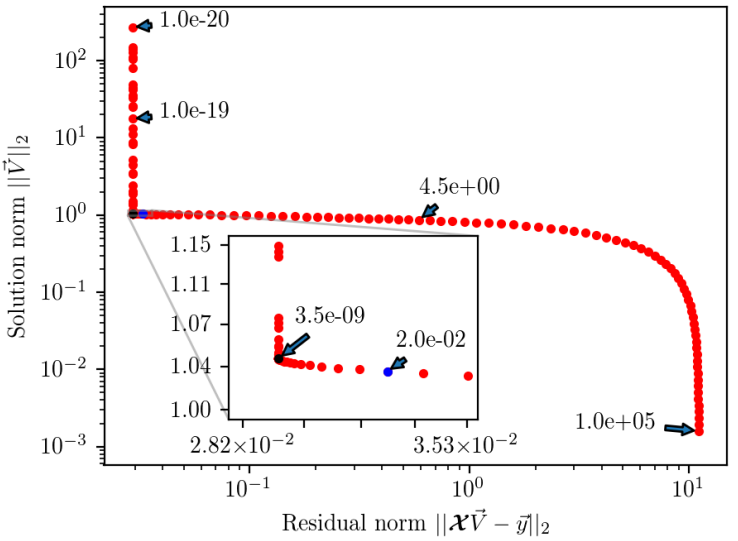


Figure 5.2: L-curve for fitting of ECIs to the training set presented in section 5.1 by setting $\boldsymbol{Q} = \lambda I$. The annotations give the value for λ and the inset plot shows a better view of the corner of the plot. The value at the corner of the curve is indicated by a black dot and corresponds to fit II, the point found by CV optimisation, fit I, is shown as a blue dot.

5.3.2 Fitting results and comparison of fits

The resulting ECIs, for single site clusters and pairs, for the three different fits listed in Table 5.4 are shown in Figure 5.3. The constant ECI for each jump type, V^{X_i}, have not been included in the Figure since they naturally have much higher values, but they are listed in Table 5.5. All the fits have V^{X_i} that are close to the value for jump in pure Al found by DFT in Table 5.1, but fit II and III are closer than fit I. The table also includes the residual norm, solution norm, LOOCV score and RMSE for the three different fits. All of the fits have similar results with very low RMSE but fit II has slightly lower residual norm and higher solution norm compared to fit I and III that used the LOOCV minimisation.

The ECIs for the single site clusters and pair clusters are not as easy to benchmark, since these cannot be measured directly in the same way as V^{X_i}. Even if a DFT calculation is performed on a structure that only have one nonzero sum over $\vec{k}$ in equation (3.4), the value of the ECI should not necessarily be the value computed from this DFT calculation. This is because the contribution of a solute in that group of single site clusters may vary with the configuration of the rest of the structure. This average contribution of each cluster is what the fitting tries to optimise. The value of an ECI, for example an ECI for a single site cluster, may also vary depending on which other terms that are included in the cluster expansion, since these other terms will catch some of the variations that the ECI must average over if the other terms are left out.

For fit III 5 out of the 6 largest ECIs, in absolute value, of range A in Figure 5.3 are

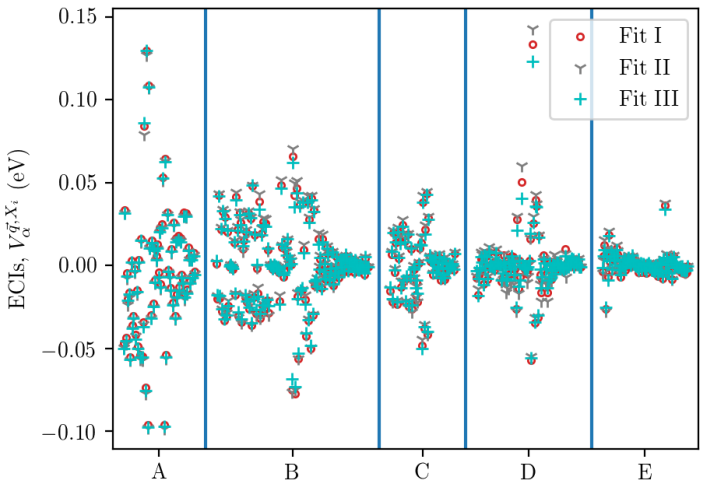


Figure 5.3: The value of all the ECIs for the three different fits. Range A contains the ECIs for single site clusters, range B, C, D and E contain ECIs for pairs that have pair distance equal to 1st NN, 2nd NN, 3rd NN and 4th NN respectively. The value for jumps in pure Al, V^{X_i}, are not included in the figure but are listed in Table 5.5.

for L4 1nn in Figure 3.1. In range B, pairs with pair distance equal to NN distance, the 4 largest ECIs are for pairs in the L4 1nn layer, the pairs are atom 7 and 8 and 9 and 10. For range C, pairs with second nearest neighbour pair distance, the 6 largest ECIs are all for pairs between atom 7 and 9 and 8 and 10 in layer L4 1nn. In range D, 3rd nearest neighbour pair distance, 7 out of 8 are for diagonal pairs between the 4 atoms in L4 1nn, and the last one of the 8 is for the 4 pairs between site 1-9, 2-10, 3-7 and 4-8 in Figure 3.1. The last range for pairs with 4th nearest neighbour pair distance have three different site groups at the top. The largest one is for Si75SiSi, which means that $X_i = \mathrm{Si}$, $\vec{q} = (\mathrm{Si}, \mathrm{Si})$ and site group is number 75, which is the pair 15-16 in the figure. The second largest is Al65MgMg and is for the pairs 1-10, 2-9, 3-8, 4-7, and the last one is Si67SiSi which is for the pair 5-6.

Table 5.5: Residual norm, solution norm, LOOCV score, RMSE and the value for the three V^{X_i} for the three different fits defined in Table 5.4. The values for V^{X_i} are given in eV.

Fit	$\|\|\boldsymbol{\mathcal{X}}\vec{V} - \vec{y}\|\|_2$	$\|\|\vec{V}\|\|_2$	LOOCV	RMSE	V^{Mg}	V^{Al}	V^{Si}
I	32.6 meV	1.033 eV	10.26 meV	1.51 meV	0.467	0.592	0.514
II	29.2 meV	1.044 eV	10.79 meV	1.35 meV	0.470	0.595	0.517
III	38.2 meV	1.031 eV	9.86 meV	1.77 meV	0.469	0.594	0.516

The average absolute value of the ECIs for each section of Figure 5.3 are listed in Table 5.6. From the table it is apparent that the average ECI is the highest for fit II which

Table 5.6: The average absolute value of the ECIs in the different ranges in Figure 5.3 for the different fits from Table 5.4.

Fit	$\langle\lvert\mathrm{A}\rvert\rangle$	$\langle\lvert\mathrm{B}\rvert\rangle$	$\langle\lvert\mathrm{C}\rvert\rangle$	$\langle\lvert\mathrm{D}\rvert\rangle$	$\langle\lvert\mathrm{E}\rvert\rangle$
I	52.02 meV	16.62 meV	12.34 meV	8.85 meV	4.22 meV
II	52.62 meV	16.86 meV	12.81 meV	10.12 meV	4.28 meV
III	52.46 meV	15.97 meV	11.77 meV	7.99 meV	3.70 meV

was optimised with the L-plot method, which is reasonable since the other two fits have a lower solution norm since they on average have more regularisation, see Table 5.4 and 5.5. It is also evident that the prior of fit III succeeds in making the average for second order terms, $\gamma_\alpha = 2$, range B, C, D and E, lower than the first order terms, $\gamma_\alpha = 1$. This is also visible in Figure 5.3 where fit III have the lowest value of the three fits for the two largest ECIs in range D. It is not as easy to see the effects of the distance term, D_α, from this table since the ranges are only split based on order and pair distance, not the distance from site i which is what D_α is based on. The prefactor for this term of the prior is however very small and will affect the fitting by only a small amount compared to the term based on γ_α.

Another way of gaining some information about the different fits is to stress test them on the pool of structures created in section 5.2. This test will show the distribution of the activation energies and try to find the upper and lower limits of the activation energies. The results of calculating the activation energies for the pool of structures with the three different fits are presented in Table 5.7. Unfortunately, all of the fits have some negative activation energies, which means that the predicted activation energy is either wrong or inaccurate or the initial state of the jump is not a stable state since it is not an energy minimum. If one of the possible transitions have a negative or very low activation energy it will get a very high transition rate and dominate the selection process in the KMC algorithm, which can lead to the vacancy getting stuck for a long time. The very high transition rates also lead to the time increase defined in equation (2.4) becoming very low. Since the KMC simulations are mostly dominated by the low barriers, it is more important that the low barriers are close to their correct value than the high barriers.

Table 5.7: Statistics from calculating the activation energies of the structures in the pool introduced in section 5.2 with the three different fits introduced in Table 5.4. Min, max and average are given in eV.

Fit	X_i	Min	Max	Avg	# negative
	Mg	−0.271	1.267	0.422	85
I	Al	−0.275	1.280	0.506	58
	Si	−0.311	1.584	0.457	85
	Mg	−0.224	1.289	0.421	72
II	Al	−0.271	1.292	0.505	44
	Si	−0.320	1.649	0.458	77
	Mg	−0.264	1.266	0.418	77
III	Al	−0.240	1.255	0.505	39
	Si	−0.225	1.509	0.453	56

The cause for the negative predicted barriers might be that the training set is missing important structures, which can lead to predicting these structures poorly since it is effectively extrapolating from the knowledge available in the training set. The LOOCV score is supposed to give an indication of how well the model fits structures outside the training set, but this estimate should be considered as a lower bound estimate since this is predicted from partitioning of the training set. During the project several predicted negative barriers have been computed using DFT and have been found to be low barriers, but positive. One example is a structure where $X_i = \mathrm{Mg}$ and site 7-10 in Figure 3.1 is Si, the remaining sites were occupied by Al atoms. This structure was predicted to be $-0.07\,\mathrm{eV}$ with an earlier fit, when the training set only had structures with 1 and 2 solutes, and was found to be $0.129\,\mathrm{eV}$ when calculated with DFT. It is possible that some of the predicted negative barriers actually are unstable configurations and will not converge to the FCC structure if calculated by DFT. These will be highly interesting as they might be jumps that start the structure for β'' or other precipitates that do not have the FCC structure. Currently these types of events are not implemented in the code, but the plan is to include at least the octahedral sites to allow β'' to form. Out of all the negative predicted barriers found from fit III and the structure pool, the structure with the lowest number of solutes had 10 solutes out of the 26 numbered positions in Figure 3.1. The training set still have few structures with many solute neighbours, which can be seen from Table 5.2. An important step towards improving the problem with negative predictions will be to include more structures with many solutes in the training set. A temporary solution for circumventing the problem of low and negative activation energies, for the purpose of being able to run simulations with the current fit, will be presented in the last subsection of this section.

5.3.3 Details of the chosen fit

For the rest of the result chapter only fit III, presented in Table 5.4, will be used. The choice is based on the information available from Table 5.5, 5.6 and 5.7 and Figure 5.3. Fit III have the lowest LOOCV score, the lowest solution norm and have values for V^{X_i} that are close to the true value in the training set given in table 5.1. The fit also have a lower average ECI value for second order terms than the two other fits, which is consistent with the physical insights presented in [36]. The last factor for choosing fit III is that it has fewer negative barrier predictions than fit I and II and also have a higher minimum barrier for Al and Si than the other two fits.

In Figure 5.4 the DFT activation energies are plotted against the activation energies calculated with fit III of the cluster expansion. All the circles are close to the black line where $E^{\mathrm{act}}_{\mathrm{CE}} = E^{\mathrm{act}}_{\mathrm{DFT}}$, which is expected since the RMSE listed in Table 5.5 is only $1.77\,\mathrm{meV}$. The activation energies for the pool of structures introduced in section 5.2 are shown in Figure 5.5. From the figure it seems like there are many structures in the pool that have predicted barriers that are outside of the range of the barriers in the training set, listed in Table 5.1. Some of the predictions might be wrong, especially the negative values, but it also indicates that the training set does not represent the whole range of activation energies well.

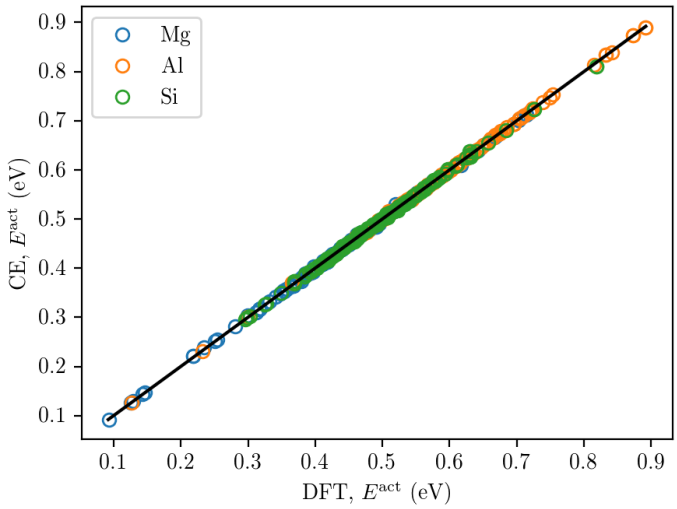


Figure 5.4: Comparison between the activation energies computed by DFT, shown in Figure 5.1, and activation energies computed by equation (3.4) with ECIs fitted to the same DFT training set using fit III in Table 5.4. The optimal result is that all the circles lay on the straight line as this is where $E^{\text{act}}_{\text{CE}} = E^{\text{act}}_{\text{DFT}}$.

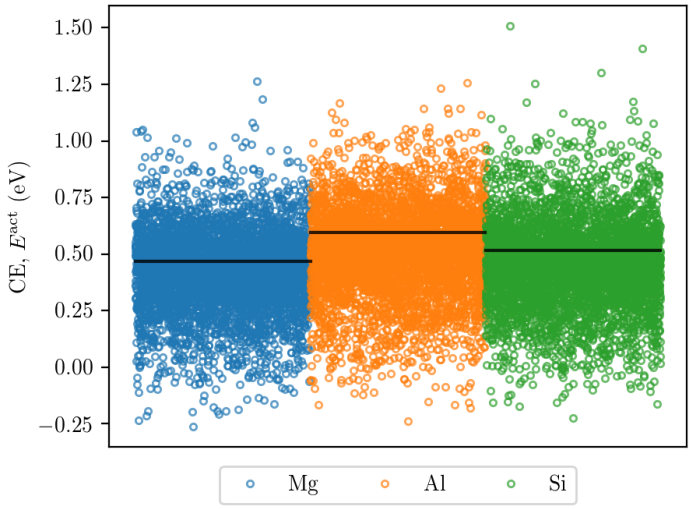


Figure 5.5: The activation energy for the pool of structures, created using the parameters in Table 5.3, calculated using equation (3.4) with fit III from Table 5.4. The different calculations are distributed along the horizontal axis to improve viewing clarity and the colour of the circles indicates the atom type that jumps. The black horizontal lines indicate the activation energy for a jump in pure Al.

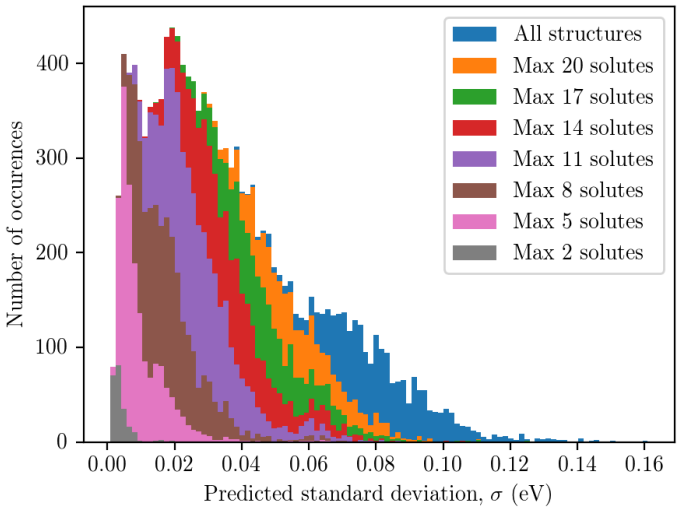


Figure 5.6: The predicted standard deviation for the pool of structures made with parameters given in Table 5.3. The prediction is made by fitting the ECIs to 100 different training sets made by the method described in section 3.5 with the optimal $Q_{\alpha\alpha}$ values from fit III in Table 5.4. The number of solutes referred to in the legend is how many of the numbered sites in Figure 3.1 that are occupied by a solute atom.

The standard deviation for the predicted activation energies shown in Figure 5.5 have been approximated by fitting the model to 100 different training sets and calculate the standard deviation of the predictions for each structure. The details of this procedure are described in the second paragraph of section 3.5. The predicted standard deviations are shown in Figure 5.6 as a histogram. From the figure it is evident that the structures with few solutes have on average lower standard deviation than the structures with many solutes. Predicted standard deviation will naturally be larger for structures with more solute neighbours since more of the sums over $\vec{k}$ in equation (3.4) will be nonzero, and therefore get larger variations since more coefficients are used to calculate the barriers. It is therefore important that the further expansion of the training set focuses on structures with many solutes. In the previous subsection it was found that the structures with negative activation energies have at least 10 solute neighbours, which in Figure 5.6 have relatively large standard deviation compared to the thermal energy at room temperature $25.3\,\mathrm{meV}$. This means that both the method of calculating the energies of the pool of structures and identifying the most negative and positive activation energies and the method of predicting the standard deviation for finding new structures to add to the training set proposes some of the same structures. It is therefore a good idea to expand the training set with the structures that both methods predict to be good additions to the set.

5.3.4 Shifting of activation energies

The cluster expansion trained on the final training set presented in section 5.1 results in some negative and very low activation energies as shown in Figure 5.5 and Table 5.7. Both the low and the negative activation energies will lead to very large transition rates and will therefore be picked by the KMC algorithm almost every iteration if available. This can lead to the vacancy getting stuck in a loop of jumps with high transition rates, which results in simulations where nothing interesting happens. This subsection will present a temporary solution that will be used to shift the activation energies up until the training set becomes more complete.

The problem of having two or more adjacent states with a very low activation energy can happen even if all the activation energies are accurate, this is also known as flickering states [21]. Even if this trapping of the vacancy happens in the real materials, it is still a problem for KMC simulations since all the CPU time is wasted on simulating the same loop of events repeatedly. In the k-ART implementation of KMC they handle flickering states by using an algorithm called basin-autoconstructing Mean Rate Method that calculates the average escape time from the basin of flickering states [21]. It should be considered to implement this method when the training set is larger and more complete, and the cluster expansion does not give low or negative barriers as often as it does now.

For the current situation the simulations encounter very low and negative energy barriers at a regular basis, especially if there are any clusters in the system. In an attempt at getting better simulation results activation energies below a set threshold are shifted up. To not alter the difference in energy between states, both the forward and the corresponding backward activation energy must be shifted by the same amount. This means that when the KMC requests the activation energy E_{ij}^{act}, the cluster expansion will need to calculate both E_{ij}^{act} and E_{ji}^{act} to check if either of them are below a set threshold.

The shifting has been done in such a way that all activation energies below E_{high} is shifted up such that it is between E_{low} and E_{high}. The added energy to E_{ij}^{act}, which is only added if E_{ij}^{act} or E_{ji}^{act} are below E_{high}, is called E_{ij}^{shift} and is calculated as

$$E_{ij}^{\text{shift}} = E_{ji}^{\text{shift}} = E_{\text{low}} - \min(E_{ij}^{\text{act}}, E_{ji}^{\text{act}}) + \max(a(E_{ij}^{\text{act}} + E_{ji}^{\text{act}})/2 + b, 0), \quad (5.1)$$

where E_{low}, a and b are parameters. Parameter a is set according to $a = (E_{\text{high}} - E_{\text{low}})/(E_{\text{min}} - E_{\text{high}})$, where E_{min} is the value of the lowest expected activation energy, and $b = -aE_{\text{high}}$. For the results in this work $E_{\text{high}} = 0.30\,\text{eV}$, $E_{\text{low}} = 0.20\,\text{eV}$ and $E_{\text{min}} = -0.30\,\text{eV}$. These values have been chosen based on simulations getting stuck when E_{low} is set lower, E_{high} has been set to $0.10\,\text{eV}$ higher than the lower bound to give some variation to the shifted barriers. The final parameter, E_{min}, is set slightly lower than the lowest recorded value for fit III in Table 5.7.

This solution does not conserve R_{i,N_e} and therefore not detailed balance given by equation (2.9). The solution does however conserve the ratio of the Boltzmann weights since both E_{ij}^{act} and E_{ji}^{act} are shifted up by the same amount. The runtime will increase to almost the double, since the amount of activation energy calculations are doubled. This is not a good long-term solution, but it is necessary at the time due to the properties of the fit to the current training set.

5.4 Benchmarking of the method by Liang et al. against DFT

This section will show a benchmark of the Liang method, presented in section 2.3, against the DFT calculations presented in section 5.1. In Figure 5.7 the activation energy computed by the Liang method is compared to the activation energies found by DFT. The black line indicates where $E^{\text{act}}_{\text{Liang}} = E^{\text{act}}_{\text{DFT}}$, the optimal result is that all the circles lie on this line. It is evident that the Liang method is far away from being consistent with the barriers calculated by DFT. The method is especially far off for the lowest and highest barriers. If a circle is to the left of the black line the Liang value is higher than the DFT value and opposite if it is to the right of the line. Most of the activation energies for the Mg jumps, blue circles, seem to be overestimated compared to DFT, this also seems to be the case for Al and Si jumps, but not to the same extent. The root mean square error (RMSE) for this method compared to the DFT calculations is $151\,\text{meV}$, which is high compared to the thermal energy at room temperature $k_{\text{B}} \cdot 293.15\,\text{K} = 25.3\,\text{meV}$.

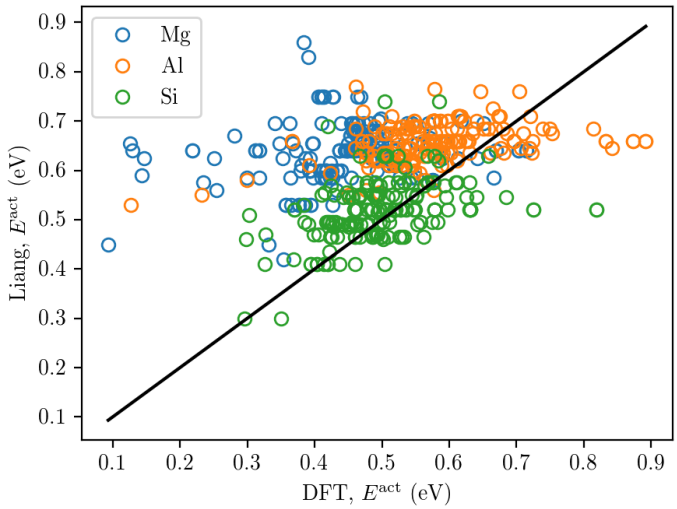


Figure 5.7: Comparison between the activation energies computed by DFT, shown in Figure 5.1, and activation energies computed by equation (2.13) with values as listed in Table 2.1. The colour of the circle indicate the atom type that jumps to the vacancy, and all the circles should lie on the black line if the two methods give the same results.

The spread of the activation energy calculated with the Liang method for the pool of structures is shown in Figure 5.8. From the figure it is evident that the Al jumps have a much narrower range of possible values for the activation energy than for Mg and Si. The DFT calculations, however, do not have a narrower range for the Al jumps than for Mg and Si. By inspecting equation 2.13 and the values for the interaction energies in Table 2.1 the reason is that since every interaction with Al is set to 0, only the interactions

with the vacancy alters the barrier. This means that the expression predicts that solutes surrounding an Al atom that jumps does not affect the energy landscape for the Al atom. In cluster expansions it is normal to set the interactions with one atom type to 0 since each site must be occupied by either Al, Mg or Si, so only two interactions are needed since the third is the baseline. The choice of which atom type that is redundant, set to zero, should not affect the outcome of the activation energies. With the method presented by Liang et al. the atom type that is set as the baseline will automatically get a narrower range of activation energies. One of the premises for setting all pair interactions that contains one or more atoms of the redundant type to zero in a cluster expansion is that all sub-clusters of the pair are already included in the expression [29]. The expression by Liang et al. only includes pair interactions, and even though the expression is not a cluster expansion it is still a plausible reason for why the choices they made give such a narrow range for Al jumps.

Another thing to note from Figure 5.8 is that there are no negative activation energies and there are very few low barriers. The minimum value for both Mg and Al is a lot higher than the minimum value found by DFT in the training set, which are listed in Table 5.1. This ensures that the vacancy does not get stuck for any extended period at room temperature or higher temperatures.

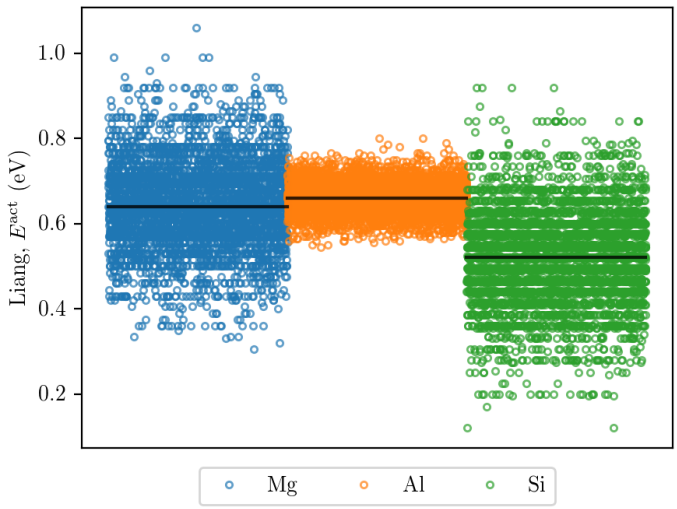


Figure 5.8: The activation energy for a pool of structures, created using the parameters in Table 5.3, calculated using equation (2.13) with values listed in Table 2.1. The different calculations are distributed along the horizontal axis to improve viewing clarity and the colour of the circle indicates the atom type that jumps. The black horizontal lines indicate the activation energy for a jump in pure Al.

5.5 Runtimes

An important goal for the in-house developed KMC implementation that have been used and extended during the work with this master thesis have been to achieve high computational speed to enable long time scales to be simulated. This is the main reason for writing all the simulation code in C instead of Python or other high-level programming languages. Python can be very fast if most of the computation time is spent in library functions, such as matrix computations, since these are written in other languages and are highly optimised. The KMC algorithm, however, is inherently based on a loop and have many low-cost computations that need to be done each iteration, see flowchart of the implementation of the algorithm in Figure 4.1. The computational speed will therefore be much higher in a compiled language such as C than in for example Python.

The time per iteration for the KMC implementation is listed in Table 5.8 for a selection of different activation energy calculators. The listed runtimes have been obtained by running the KMC simulation without any observers on a single core of an i7-8700 CPU. Cluster expansion with only the constant and single site clusters give 69 ECIs and is approximately 8 times slower than the method by Liang et al. [1]. The reason for this is that for each ECI, the sum over symmetrically equivalent clusters must be computed. The Liang method has effectively 3 different constants and 2 different site groups, given by NN_i and NN_j, and have 5 interaction energies per site group. This can roughly be considered as $3+2\cdot 5 = 13$ "ECIs", which is five times fewer than the CE with only single site clusters. The CE with pair clusters is a factor 9 slower than the CE with only single site clusters. All of the CE runtimes are measured without the use of the shifting described in equation (5.1), with the shifting the runtime is close to double for the CE with most ECIs since the activation energy calculations use most of the computation time.

Table 5.8: Single core performance on an i7-8700 processor without any observers for the method by Liang et al. [1] and the cluster expansion given by equation (3.4). The CE with 69 ECIs are for only single site clusters, while the one with 444 includes pair clusters up to 4th nearest neighbour pair distance, see Figure 3.2 for definition. The iteration count refers to the number of complete KMC iterations, which all includes 12 transition rate calculations.

Method	Number of ECIs	Time per iteration	Iterations per second
Liang	-	$0.60\,\mu s$	$1.65 \cdot 10^6$
CE	69	$4.64\,\mu s$	$2.16 \cdot 10^5$
CE	444	$40.2\,\mu s$	$2.49 \cdot 10^4$

When running the code without any observers the calculation of the transition rates uses most of the time, at least when using the CE method. In an attempt to speedup the program the loop that calculates all the transition rates for one iteration, 12 in this case, was parallelised using OpenMP[3]. Since all of the rate calculations are independent from each other and do not alter the KMC struct no race conditions can occur[4]. A race condition

[3] OpenMP enables parallelisation of code on shared memory systems by using compiler directives, see openmp.org for more information.

[4] In the current implementation, if the shifting described by equation (5.1) is enabled the transition rate calculation changes the KMC struct temporarily for calculating the reverse transition rate, which will cause race conditions, and should therefore not be used together with parallelisation.

is when the outcome of a parallel process can have different outcomes based on the execution order of the different threads or processes. The parallelised code was approximately 80 times slower than the serial code on a quad core processor with hyperthreading, with transition rates calculated by CE with just under 400 ECIs. The reason for the code becoming slower even though it has access to more computational power is related to overhead and the short time per iteration. For each iteration the master thread will fork several slave threads and distribute the calculations to all of the threads. When the slave threads are finished calculating the transition rates that was designated to them, they will join the master thread again. This process of forking and joining threads add some overhead, which in this case is large compared to the calculation time needed for the 12 transition rates, and the program therefore runs slower.

It is likely that a more advanced parallelisation scheme that uses several worker threads that are kept alive during the entire simulation can give a speedup. This requires a bit more communication, but should be a lot faster since the overhead of forking and joining is removed. Results from simulations of statistical systems should usually be averaged over many realisations of the system to get good data, and any parallelisation will always add some overhead due to communication and/or forking and joining of threads. It is therefore not much to gain by parallelising the code, even if it would give a significant speedup, unless the user has access to more processor cores than the number of realisations of the system that is needed. The best solution is currently to run many instances of the same serial program, with different seeds, at the same time.

5.6 Simulation results

This section will present and compare results from simulations performed with the cluster expansion based transition rates and the transition rates calculated by the method by Liang et al. [1]. Since the CE method unfortunately does not produce any real clustering yet, there will not be presented results where the results have been averaged over many realisations of the system. The focus has instead been on inspecting the behaviour of single runs. For averaged results created using the method of Liang et al. see the result chapter in my specialisation project [8].

5.6.1 Cluster evolution and snapshots

Two different simulations that use the method by Liang et al. will be presented in this section, and they will be referred to as Liang run A and Liang run B throughout the section. Run A is started from a random initial configuration, use all the default settings and have an end time of $t = 1 \cdot 10^8$ s. The monomer concentration, C_1, and the average cluster size for run A is shown in Figure 5.9. In this work the monomer concentration will refer to the ratio of single solutes to the total number of solutes in the system, which is 900. Clusters are here defined as all solutes that are connected through NN interactions with each other and have cluster size of 2 or larger. The threshold for being a cluster would normally be higher to avoid including random fluctuations as clusters, but since the CE method shows very little clustering this convention have been used to be able to present both methods in

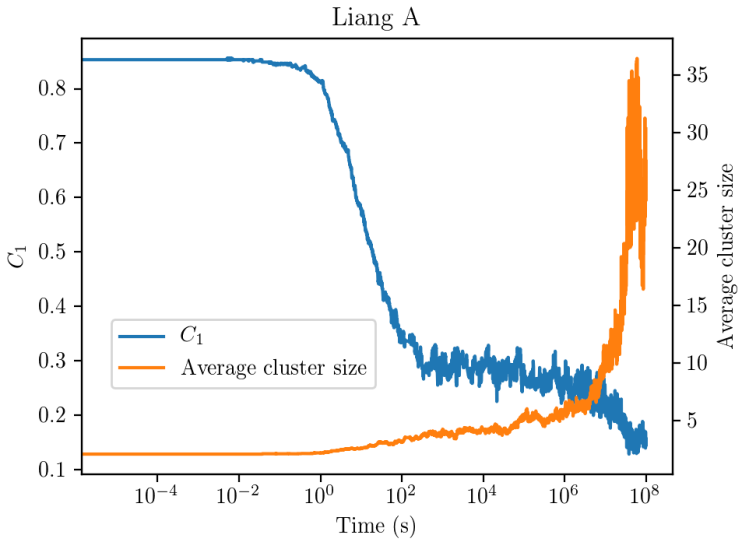
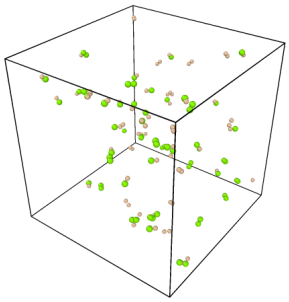


Figure 5.9: Monomer concentration, C_1, and average cluster size from a KMC simulation starting from randomly dispersed solutes, referred to as Liang run A. The transition rates are calculated using the method of Liang et al. [1].

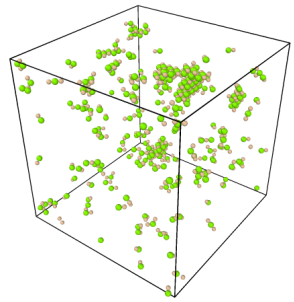
the same way. Results with the Liang method where the criterion has been set to size 30 was shown in my specialisation project [8].

In Figure 5.10 snapshots of the position of all solutes that are part of a cluster of size 2 or larger are shown for different times for Liang run A and run B, which will be introduced in the next paragraph. The initial random configuration of the lattice for run A is shown in subfigure 5.10a. After 20 h at room temperature some larger clusters start to form as shown in subfigure 5.10b. The final configuration shown in subfigure 5.10c contains mainly one large cluster and one smaller cluster that looks like two small clusters but are connected through the periodic boundaries. The large cluster have alternating layers of Mg and Si, where the layers are in the 100 family of planes. This structure is known as the $L1_0$ structure and was studied by DFT and MC in the article by Kleiven and Akola [31]. They found that this structure is a stable configuration for Mg and Si within the Al host matrix. They also studied the interfacial energy for the side of the structure consisting of alternating bands of Mg and Si and found that this interface has approximately half the interfacial energy of the average interfacial energy of pure Al-Mg and Al-Si interfaces. This is consistent with the elongated shape of the final cluster along the plane normal direction, since this increases the percentage of the surface area of the cluster that is of the alternating Mg Si type.

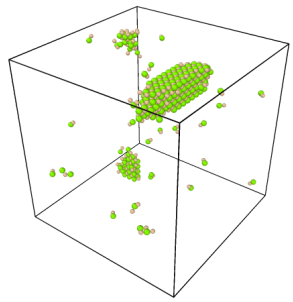
To check the stability of the $L1_0$ structured cluster, a simulation referred to as Liang run B was started from the end configuration of Liang run A, subfigure 5.10c, at a slightly elevated temperature of $T = 350\,\mathrm{K}$. A plot of the monomer concentration and average cluster size is shown in Figure 5.11. The figure shows that the large cluster dissolves and



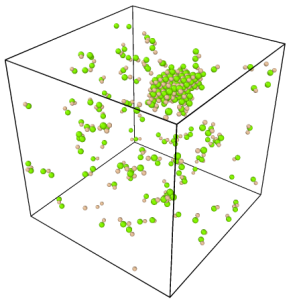
(a) Liang A, CE A: $t = 0\,\mathrm{s}$



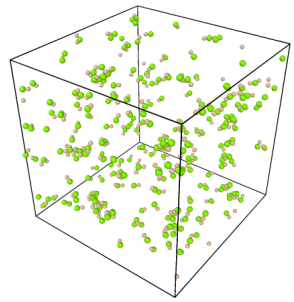
(b) Liang A: $t = 20\,\mathrm{h}$



(c) Liang A: $t = 3.2\,\mathrm{yr}$, Liang B: $t = 0\,\mathrm{s}$



(d) Liang B: $t = 1.1\,\mathrm{h}$



(e) Liang B: $t = 2.4\,\mathrm{h}$

Figure 5.10: Snapshots of the position of all solute atoms that is part of a cluster of size 2 or larger for Liang run A and B. The start position of CE run A is the same as for Liang run A. The Mg atoms are green and the Si atoms are sand coloured.

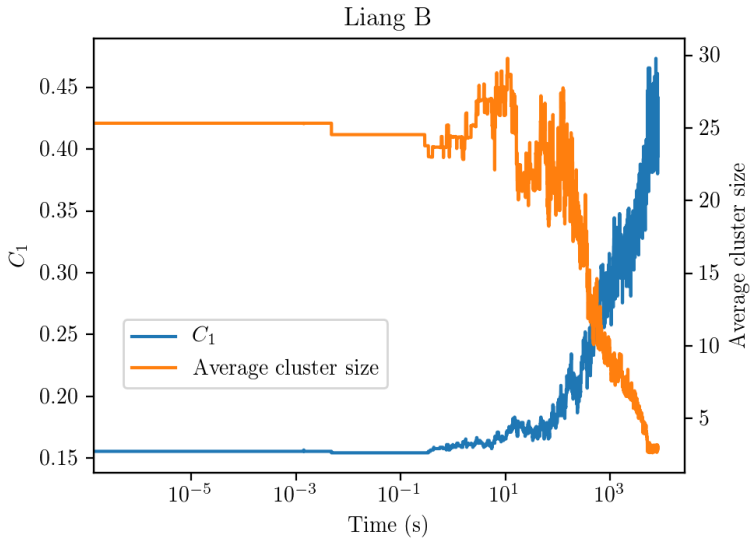


Figure 5.11: Monomer concentration and average cluster size from KMC simulation with transition rates calculated by the method presented by Liang et al. [1]. The simulation was started from the final configuration of Liang run A shown in subfigure 5.10c and the temperature was set to $T = 350\,\mathrm{K}$.

the end configuration contain many smaller clusters as the average cluster size is around 3 and C_1 fluctuates between 0.40 and 0.45. The snapshot in subfigure 5.10d is taken after $1.1\,\mathrm{h}$ and show that the large cluster has decreased in size and become more spherical compared to the initial structure. The final configuration shown in subfigure 5.10e show that there are only very small clusters left, but there are a lot more of them compared to the random distribution in subfigure 5.10a. Several other temperatures have also been tested, higher temperatures lead to the cluster dissolving faster. A lower temperature of $T = 325\,\mathrm{K}$ was also tested, for this temperature the cluster was stable for the duration of the simulation. Why the clusters dissolve will be discussed in subsection 5.6.3.

The results from two simulation runs where the cluster expansion have been used to calculate the transition rates will be presented, and are referred to as CE run A and B. The transition rates have been calculated using the cluster expansion described in equation (3.4) with ECIs from fit III in Table 5.4 and a shift have been added to low barriers according to equation (5.1). The monomer concentration and average cluster size for CE run A, which starts from the same random initial configuration as Liang run A, see snapshot in subfigure 5.10a, is shown in Figure 5.12. From the Figure it is apparent that no large clusters form during the simulation run at $T = 293.15\,\mathrm{K}$ since the average cluster size is never above 2.5 and C_1 stays above 0.70. This can also be seen from the snapshot of the final state of the simulation in subfigure 5.13a. A run without any shifting was also tested, but this produced similar results until the vacancy got stuck, no clusters larger than a few solutes were observed.

In Figure 5.14 the average cluster size and monomer concentration is shown for a run

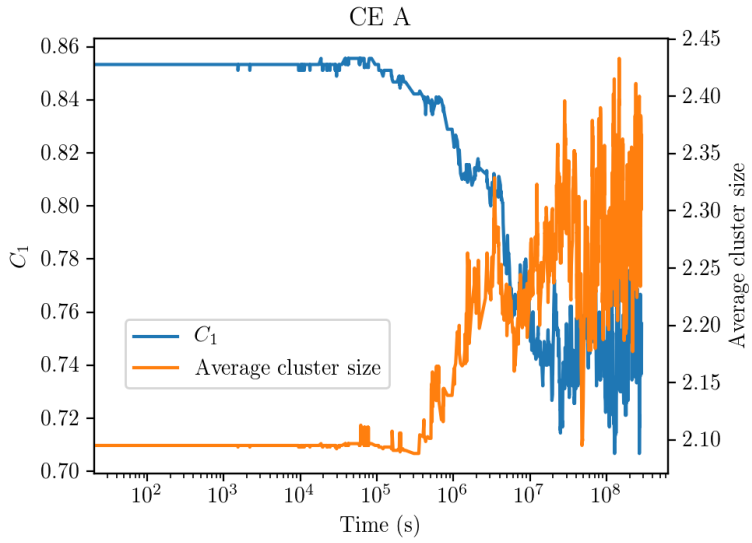
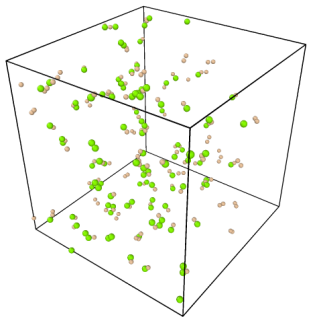
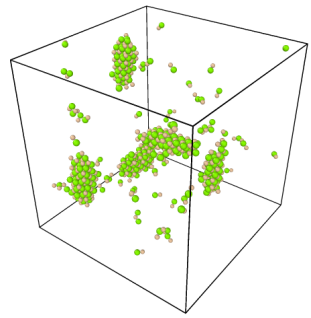


Figure 5.12: Monomer concentration and average cluster size for a simulation starting from the same random initial configuration as Liang run A, see snapshot in subfigure 5.10a. The transition rates are calculated using fit III from Table 5.4 with the shifting of low barriers as described by equation (5.1).

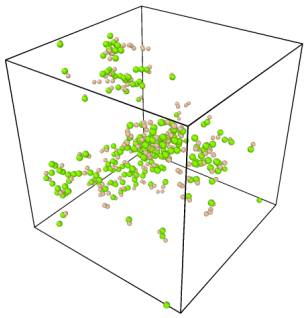
starting from the clustered configuration shown in subfigure 5.13b, this will be referred to as CE run B. The transition rates are calculated the same way as for CE run A and the temperature have been set to $T = 450\,\mathrm{K}$. The figure shows that very little happens to either of the quantities before it rapidly dissolves the clusters. From visual inspection of the lattice during the simulation it seems like the vacancy reorganises the $L1_0$ structure to a disordered cluster with more aluminium inside the structure. A snapshot of an intermediate state where the clusters are close to dissolved is shown in subfigure 5.13c. Then the solutes that are sticking out of the cluster are picked off one by one, dissolving the clusters. Towards the end of run B the system is back to solid solution, but C_1 is slightly higher and the average cluster size slightly lower compared to the solid solution of CE run A, which is as expected since the temperature is higher for run B. A snapshot of this solid solution is shown in subfigure 5.13d. From the figure it seems like there also are some dimers consisting of only Si or only Mg, which is not common to see in runs with the Liang method. Lower temperatures, $T = 293\,\mathrm{K}$, $T = 350\,\mathrm{K}$ and $T = 400\,\mathrm{K}$, was also tested. If given enough time, around one week of CPU time running on a single core, the clusters will dissolve at $T = 400\,\mathrm{K}$. At the lower temperatures the vacancy seems to stay inside one of the large clusters and move solutes around, but the clusters does not get dissolved.



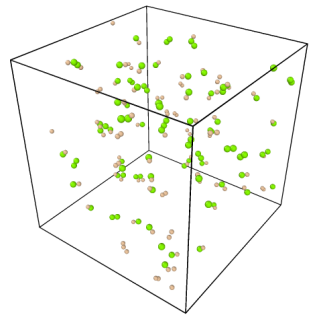
(a) CE A: $t = 9.7\,\mathrm{yr}$



(b) CE B: $t = 0\,\mathrm{s}$



(c) CE B: $t = 3.5\,\mathrm{s}$



(d) CE B: $t = 0.9\,\mathrm{h}$

Figure 5.13: Snapshots of the position of all solute atoms that is part of a cluster of size 2 or larger for CE run A and B. Snapshot of the initial configuration for CE run A is shown in subfigure 5.10a. The Mg atoms are green and the Si atoms are sand coloured.

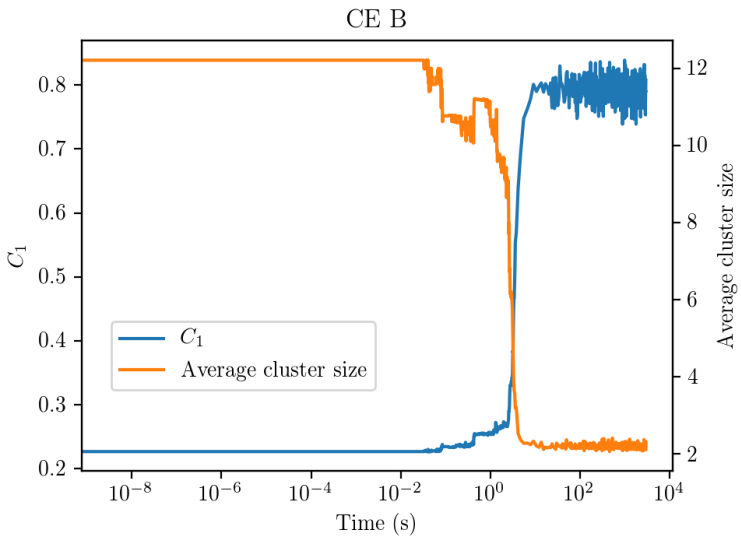


Figure 5.14: Monomer concentration and average cluster size for a simulation starting from the clustered state shown in subfigure 5.13b at $T = 450\,\mathrm{K}$. The transition rates are calculated using fit III from Table 5.4 with the shifting of low barriers as described by equation (5.1).

5.6.2 Diffusivity

In Figure 5.15 the diffusivities for the different elements have been plotted for Liang run A. The left subplot is for the old diffusion logging that had a bug in it as mentioned in subsection 4.1.4, and the right subplot is the fixed implementation of the logging. The problem with the old implementation was that it checked which atom type that occupies the lattice site now, but calculated the diffusion distance for the atom that started at the site instead of the one that is there now. This resulted in that the averaging over all atoms of the same type in equation (4.2) effectively became just the average over a random selection of atoms after all the atoms had been moved around, which explains why the diffusivities are all grouped together after some time.

The diffusivities in Figure 5.15 decrease over time, this can have several causes and was studied during my specialisation project [8]. The initial decrease and sporadic data points are due to the calculation being based on very few moves per atom. The continued decrease is related to that as clusters start to form, some of the atoms on the inside of the clusters become close to stationary. Another contribution is that the time increase for each iteration, given by equation (2.4), increases throughout the simulation. In the specialisation project the cause for this effect was not clear, but this might partly be explained by Figure 5.8, which shows that the activation energies for Al have a narrow range and no low barriers. Many of the solute jump activation energies are low and will cause the time to increase slowly since they have large transition rates. Since less solutes are reachable for the vacancy as clusters start to form fewer iterations with very low Δt will occur and the time will on average increase faster, which leads to lower diffusivities.

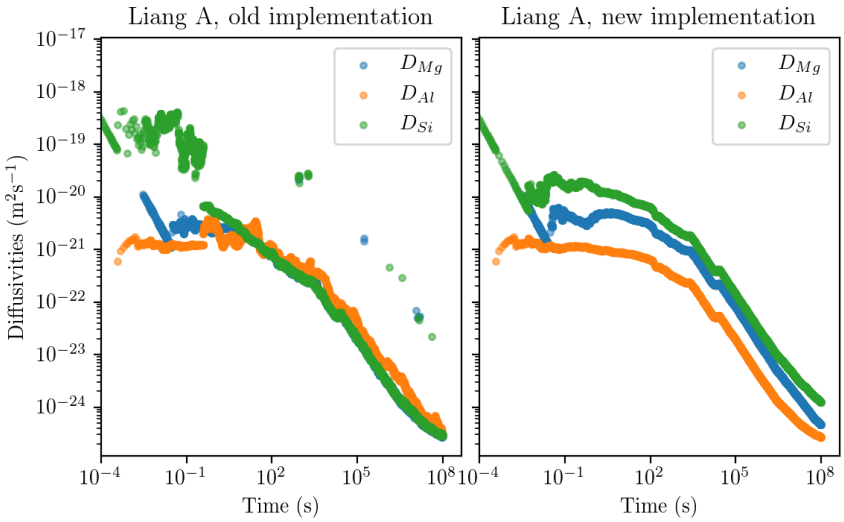


Figure 5.15: Comparison of the old implementation, which had a bug, and the new implementation for logging diffusivities. The old implementation did not include the correct atoms in the average in equation (4.2), which is why they almost merge together in the end. The data for both subplots are from Liang run A.

The diffusivity for Si is the highest, followed by Mg and then Al in Figure 5.15, and the spread is approximately one order of magnitude. In the article by Løvvik et al. [45], they found $D_{\mathrm{Al}} = 8.47 \cdot 10^{-27}\,\mathrm{m^2\,s^{-1}}$ and $D_{\mathrm{Si}} = 2.12 \cdot 10^{-28}\,\mathrm{m^2\,s^{-1}}$ at $T = 293.15\,\mathrm{K}$. In an older article by Mantina et al. [44] they report the values for Mg and Si to be $D_{\mathrm{Si}} = 6.21 \cdot 10^{-26}\,\mathrm{m^2\,s^{-1}}$ and $D_{\mathrm{Mg}} = 1.75 \cdot 10^{-27}\,\mathrm{m^2\,s^{-1}}$ at $T = 293.15\,\mathrm{K}$. This means that the results obtained by the KMC with the Liang method are 2-4 orders of magnitude too high, when using the diffusivities near the end of the simulation, and the ratio between Si and Al diffusivity is not consistent with [45]. If only the order of magnitude would have been different, it would be possible to just scale the time evolution by adjusting the vacancy concentration, but this will just shift all of them down and not alter the order of them. The true vacancy concentration is difficult to find since the alloy will probably have a higher concentration than the equilibrium concentration, at least in the beginning, because of the quenching step in Figure 2.1. This rapid cooling can ”freeze-in” vacancies since the alloy is not in equilibrium during the cooling. The value used in the article by Liang et al. [1], corresponds to the equilibrium concentration at $T = 732\,\mathrm{K}$ when using the values $H_{\mathrm{V}}^{\mathrm{F}} = 0.66\,\mathrm{eV}$ and $S_{\mathrm{V}}^{\mathrm{F}} = 1.6k_{\mathrm{B}}$ from [28].

For Liang run B the diffusivities are shown in Figure 5.16. This shows that the initial decrease is present in this simulation also, but when the clusters dissolve the values increase again. The initial decrease and fluctuation might be due to the diffusion distance being based on very few jumps per atom, as stated earlier. The expression used for calculating the diffusivities is given in equation (4.2), which shows that the limit of $t \to \infty$ should be taken to get the correct values. The calculated diffusivities should therefore be

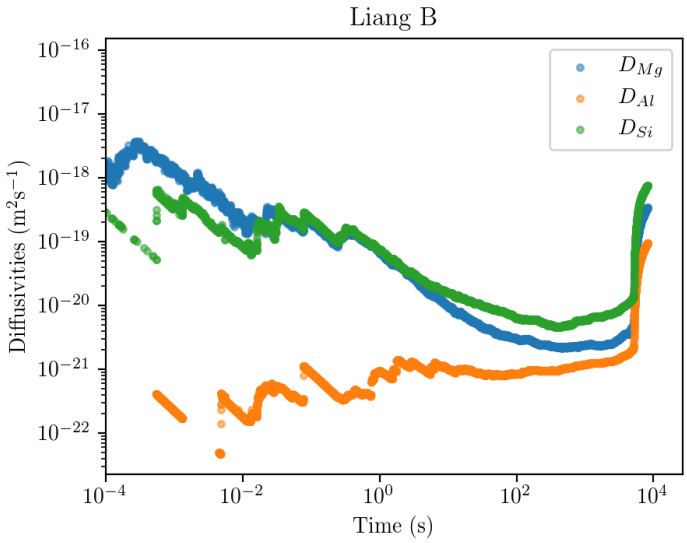


Figure 5.16: Diffusivities for Liang run B calculated from equation (4.2).

given some time to settle. The increase when the clusters dissolve is consistent with the decrease shown in Figure 5.15 when the system is clustering.

The diffusivities for CE run A are shown in Figure 5.17, which shows that they are relatively stable throughout the run since no large clusters form as seen from Figure 5.12. The magnitudes of the diffusivities are slightly closer to the literature than the results for the end of Liang run A and the order is the same as for Liang run A and B. This order is as discussed not consistent with the literature values in [45]. The vacancy concentration for this run is set to the equilibrium concentration, which is $C_{\mathrm{V}}(T = 293.15\,\mathrm{K}) = 2.23 \cdot 10^{-11}$ at room temperature. This concentration is 7 orders of magnitude lower than the one used in Liang run A, which means that if the same vacancy concentration as for Liang run A would be used here, the diffusivities would be shifted up by 7 orders of magnitude since $D \propto 1/t \propto C_{\mathrm{V}}$. The real vacancy concentration will probably lie somewhere between these two values and most likely change with time as it moves towards the equilibrium value.

The diffusivities for CE run B is shown in Figure 5.18. The aluminium diffusivity does not show a value in the beginning since the vacancy started close to a cluster and does not move any Al atoms in the beginning. Again the plot shows that the diffusivities need some time to get a stable estimate and it also shows that the diffusivities goes up when the cluster dissolves, but the change is much smaller than for Liang run B shown in Figure 5.16. The order of the diffusivities is also the same as for CE run A and Liang run A and B. Since the temperature is higher, and therefore have a higher vacancy concentration, the diffusivities are also higher.

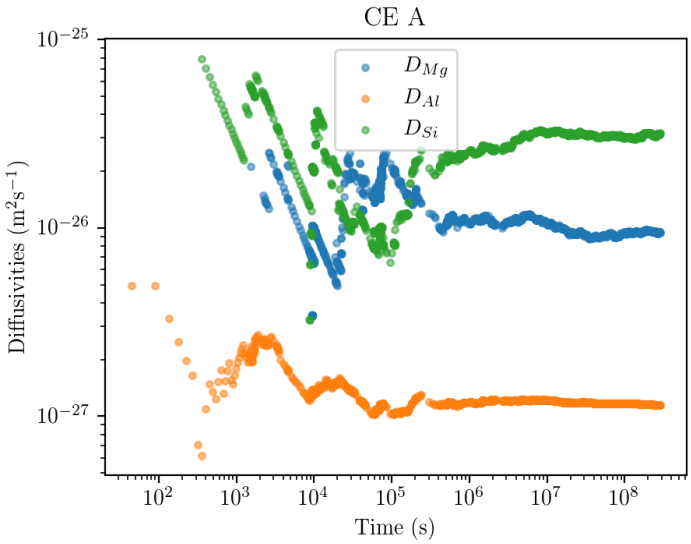


Figure 5.17: Diffusivities for CE run A calculated from equation (4.2).

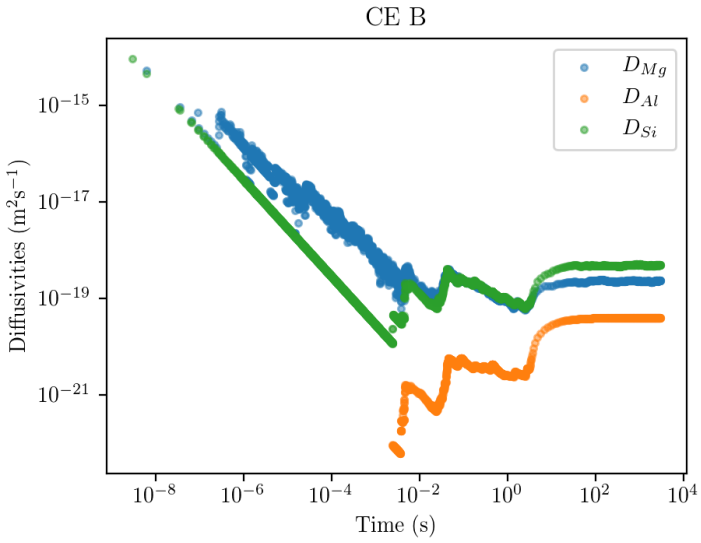


Figure 5.18: Diffusivities for CE run B calculated from equation (4.2).

5.6.3 Total energy evolution

In this subsection logs of the total energy relative to the initial lattice configuration is going to be presented for Liang run A and B and CE run A and B and compared to the total energy of the lattice computed by a cluster expansion of the whole lattice. In this setting the total energy is the zero-point energy of the system, since both the lattice CE and the activation energy CE have been trained on DFT calculations performed at $0\,\mathrm{K}$. The energy evolution logged from the KMC simulations are calculated from the backward and forward activation energies by using equation (4.1). To distinguish the CE for the total lattice from the CE for the activation energies, the lattice CE will be denoted as CE_MC since it is often used for regular MC simulations. The CE for the activation energies will be denoted as just CE and sometimes as CE_KMC.

Introduction and benchmark of CE_MC for lattice

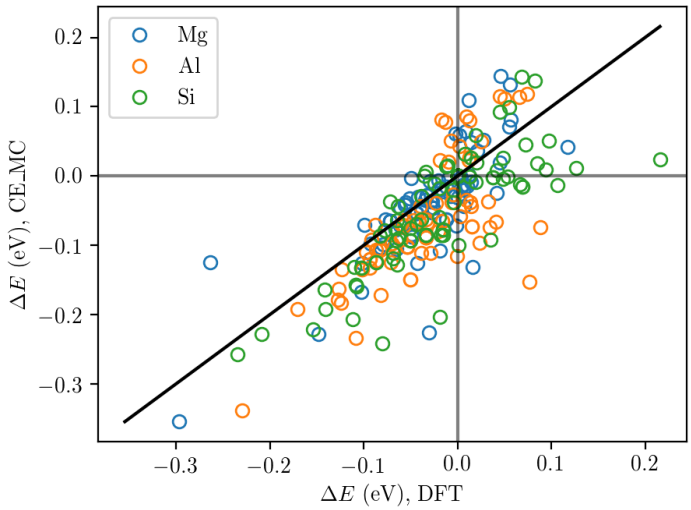


Figure 5.19: Comparison of the energy difference between states calculated from the DFT activation energies for the forward and reverse jump and the energy difference calculated from the CE_MC of the corresponding initial and final lattice configuration. The RMSE is $50.3\,\mathrm{meV}$.

The CE used for calculating the total energy of the lattice, referred to as CE_MC, have been created by PhD student David Kleiven using cluster expansion for atomic simulation environment (CLEASE) [50]. The CE_MC is trained for Al-Mg-Si-X system and have been trained on DFT calculations performed with the DFT Python code GPAW[5]. Calculations have been performed on a wide variety of configurations ranging from 0 to $100\,\mathrm{at.\%}$ of Mg, Al and Si and 0-20 $\mathrm{at.\%}$ of vacancies. The atom positions have been kept fixed, but

[5] See https://wiki.fysik.dtu.dk/gpaw/ for documentation.

the cell volume have been relaxed. The expansion includes all clusters up to 4 sites and a maximum cluster size of 5 Å. The trained model have a CV score of $6\,\mathrm{meV\,atom^{-1}}$, this is for the whole spectrum of possible structures.

In Figure 5.19 the energy difference between pairs of adjacent states have been compared for the CE_MC method and for the DFT calculated activation energies from section 5.1. The RMSE is $50.3\,\mathrm{meV}$, which is $0.47\,\mathrm{meV\,atom^{-1}}$ since the cells contain 108 lattice sites. This is well below the CV score of the CE_MC model and might be related to that it is energy differences between two very similar states that is calculated and not the total energy. Some of the differences may also be related to the DFT calculations the CE_MC have been trained on have been performed in a different program with different settings than the ones performed in this work.

Total energy evolution from simulations

The total energy evolution for Liang run A is shown in Figure 5.20. The logged total energy decreases faster than the CE_MC between $t = 1\,\mathrm{s}$ and approximately $t = 1 \cdot 10^4\,\mathrm{s}$, but then they drift together again and end up relatively close to each other. Fine details of the curves seem to match well throughout the whole run. The final configuration of the lattice is approximately $110\,\mathrm{eV}$ lower than the initial, which corresponds to a decrease in energy of $1.8\,\mathrm{meV\,atom^{-1}}$.

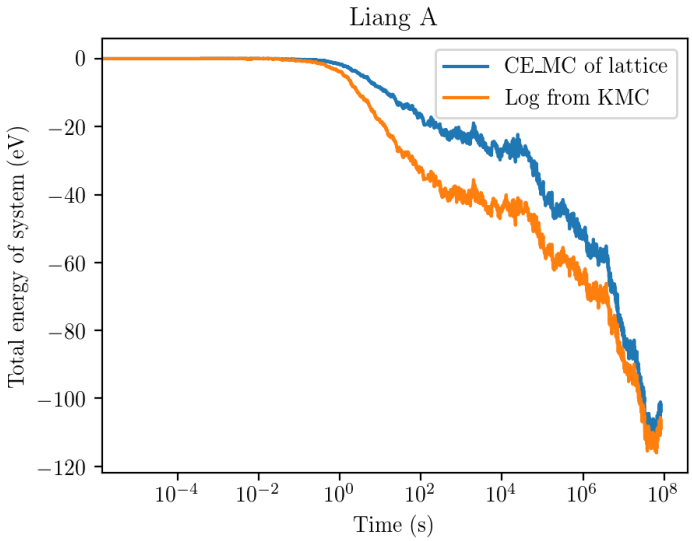


Figure 5.20: The total energy evolution for Liang run A, both from logging during the KMC simulation and from calculations of the total energy of the lattice using CE_MC. The energy is given relative to the initial configuration of the lattice.

The total energy evolution for Liang run B is shown in Figure 5.21. From the initial configuration of run B, which is the final state of run A, the energy increases by approximately $90\,\mathrm{eV}$, which is $1.4\,\mathrm{meV\,atom^{-1}}$. This means that the final configuration of B has

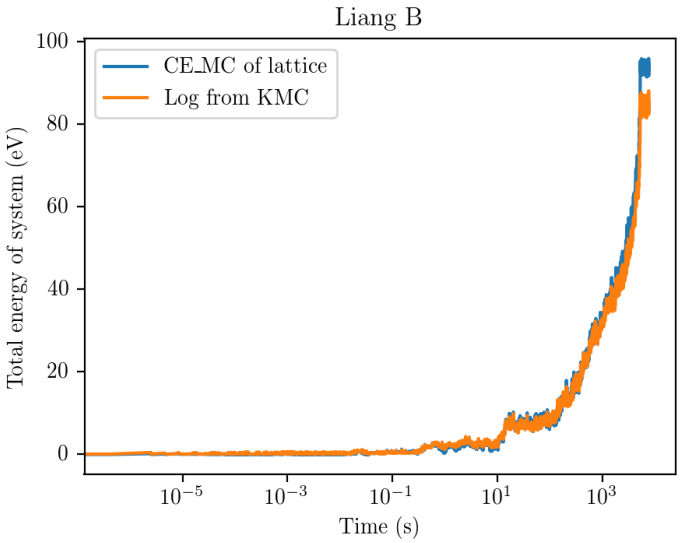


Figure 5.21: The total energy evolution of Liang run B, both from logging during the KMC simulation and from calculations of the total energy of the lattice using CE_MC. The energy is given relative to the initial configuration of the lattice.

a lower energy than the initial configuration of A. The energy curve is much steeper for the dissolving cluster, and it is therefore difficult to see if there is any drift between the two, but they end up at slightly different total energies.

The total energy of Liang run B increases throughout the run, which is unusual, especially since the temperature is only $57\,\mathrm{K}$ higher than run A that showed clustering. It is the transition rates that dictate the evolution of the system when running KMC, and increasing the temperature results in the transition rates for different activation energies become more similar than at lower temperatures. This indicates that the difference in the activation energies for clustering and dissolving events are close to each other, even though clustering is slightly lower. Free energy calculations are not available from the program in the current implementation, but it seems unlikely that the free energy decreases when the zero-point energy increases by approximately $90\,\mathrm{eV}$ and the temperature is only $T = 350\,\mathrm{K}$.

In Figure 5.22 the energy difference between adjacent states have been plotted for the 6000 first iterations for Liang run A and B against the energy difference found from CE_MC. The energy differences for Liang run B, which starts from the clustered configuration, have a much wider span than the differences for the random initial configuration of run A. The RMSE for the points from run A is $25.7\,\mathrm{meV}$ and for run B it is $102\,\mathrm{meV}$. Both of these are well below the CV score of the CE_MC since they correspond to $0.41\,\mu\mathrm{eV\,atom^{-1}}$ and $1.6\,\mu\mathrm{eV\,atom^{-1}}$. It is also visible that the discretisation of the method is relatively coarse, which causes bands of possible values. The differences for both run A and B seem to have approximately equally many differences on each side of the solid black line, which results in that they follow each other well on average.

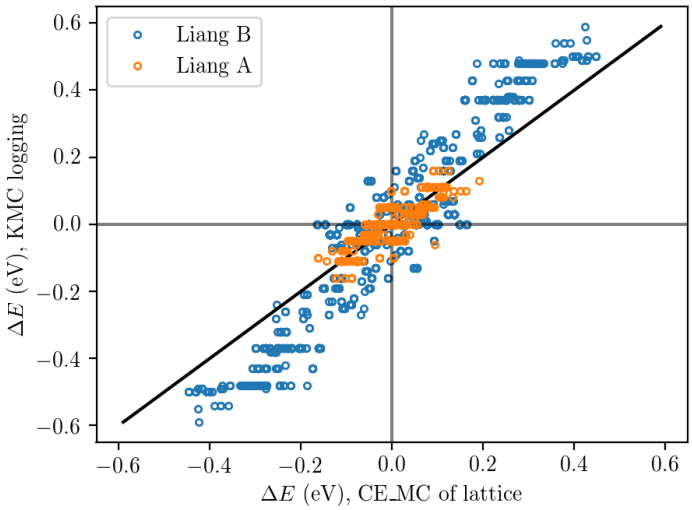


Figure 5.22: Benchmarking of the energy difference between adjacent states for the 6000 first jumps for Liang run A and B against the CE_MC. The RMSE for run A is 25.7 meV and for run B the RMSE is 102 meV.

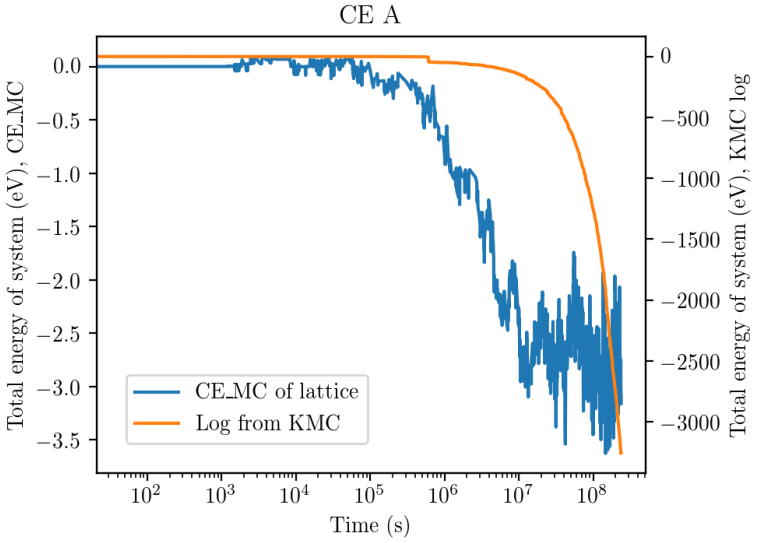


Figure 5.23: The total energy evolution of CE run A, both from logging during the KMC simulation and from calculations of the total energy of the lattice using CE_MC. The energy is given relative to the initial configuration of the lattice. Note the that the two time series are plotted on separate y-axes.

The total energy evolution for CE run A is shown in Figure 5.23. Note that the CE_MC of the lattice and the KMC log have been plotted on separate y-axes due to the large difference in size. It is apparent that the total energy logged from KMC with CE activation energies suffers from a large drift and ends up with a energy lower than $-3000\,\mathrm{eV}$ while the CE_MC ends up fluctuating around approximately $-3.0\,\mathrm{eV}$. The fine detail of the CE_MC is not reflected in the log from KMC. For the comparison of the energy difference between adjacent states in Figure 5.24 the RMSE is $32.2\,\mathrm{meV}$ for this run. This is not a lot higher than the method by Liang et al., but the differences might be larger later in the run.

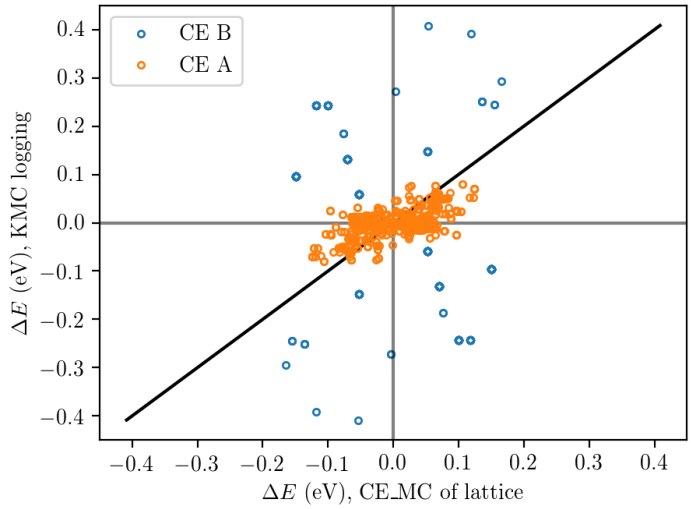


Figure 5.24: Benchmarking of the energy difference between adjacent states for the 6000 first jumps for CE run A and B against the CE_MC method. The RMSE for run A is $32.2\,\mathrm{meV}$ and for run B the RMSE is $227\,\mathrm{meV}$. The reason for few data points for run B is that many of the jumps are performed back and forth several times.

Total energy evolution for run B is shown in Figure 5.25. For run B the situation looks very similar to the situation in run A, but here the CE_MC energy increases, as expected when the clusters dissolves, but the logged energy still become negative. The size of the relative total energy is also very different between the two methods, CE_MC ends up at approximately $75\,\mathrm{eV}$ while the KMC log ends up close to $-3.5 \cdot 10^5\,\mathrm{eV}$. It also seems like the logged energy stabilises shortly after the clusters dissolve, but then after some time starts to decrease fast again. Why this happens have not been identified. The continued decrease after the CE_MC have stabilised indicates that just moving around in the solid solution without altering the monomer concentration or the average cluster size by any considerable amount, see Figure 5.14, there is a large negative drift in the total energy.

The problem of the total energy drifting severely can be solved by using the approach known as Kinetically Resolved Activation barrier (KRA) introduced by Van der Ven et al.

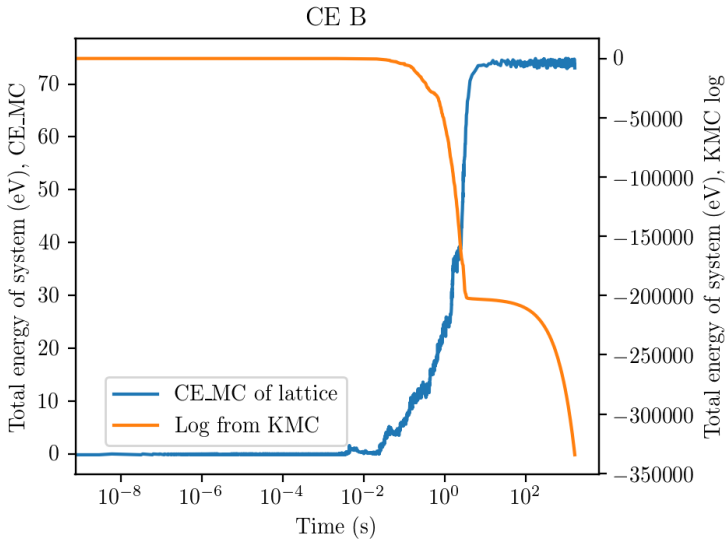


Figure 5.25: The total energy evolution of CE run B, both from logging during the KMC simulation and from calculations of the total energy of the lattice using CE_MC. The energy is given relative to the initial configuration of the lattice. Note the that the two time series are plotted on separate y-axes.

[32]. The KRA method uses a local cluster expansion to find the energy of the top of the saddle point minus the average of the energy of all the states reachable from the top of the barrier, which is the quantity they refer to as ΔE_{KRA}. This can be formulated as

$$\Delta E_{\mathrm{KRA}} = E_{\mathrm{saddle}} - \frac{1}{n}\sum_{k=1}^{n} E_k, \tag{5.2}$$

where E_k is the energy of the system in endpoint number k out of the n different possible endpoints from the top of the barrier. The activation energy from state i to state j can then be calculated from

$$\Delta E_{ij} = E_{ij}^{\mathrm{act}} = \Delta E_{\mathrm{KRA}} + \frac{1}{n}\sum_{k=1}^{n} E_k - E_i. \tag{5.3}$$

This method explicitly includes the energy difference between adjacent states and will therefore be much more robust when it comes to getting the total energy correct, than the approach introduced in this thesis where the energy differences are only included implicitly. The disadvantage of the KRA method is that the total energy for each of the n states reachable from the top of the barrier must be computed to obtain the activation energy for the one event. For the KMC, assuming that for each barrier top only two states are reachable, state i and j, the program must calculate the cluster expansion for ΔE_{KRA} plus the cluster expansion for the total energy of state i and state j for each of the possible

transitions. The energy of state i is of course common for all the possible transitions, but the method is still much more computationally intensive than the single cluster expansion given by equation (3.4).

It is possible to calculate ΔE_{KRA} from the existing cluster expansion for the activation energy for the forward and backward reaction instead of creating and training a new expansion for ΔE_{KRA} directly. By inspecting Figure 2.6 the following relation can be found

$$\frac{E_{ij}^{\mathrm{act}} + E_{ji}^{\mathrm{act}}}{2} = E_{\mathrm{saddle}} - \frac{E_i + E_j}{2} = \Delta E_{\mathrm{KRA}}. \tag{5.4}$$

Even though the KRA method from [32] most likely will fix the problem of the logged total energy drifting, it would still not ensure that the time evolution of the system becomes more correct, since this is governed by the barriers, not the energy differences. Considering that the KRA method is significantly more computationally expensive and does not ensure the correct time evolution it seems like it is a good idea to continue expanding and improving the training set to try to get the cluster expansion proposed in this thesis to work well.

5.7 Clustering rates

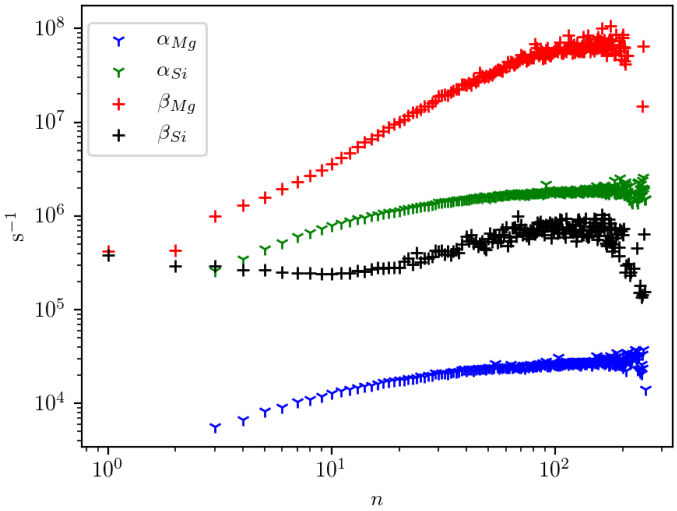


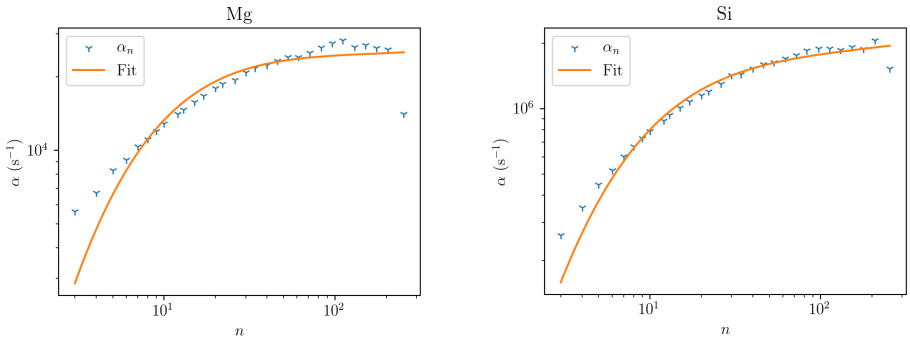
Figure 5.26: The condensation and evaporation rate for Mg and Si atoms as a function of cluster size. The results are averaged over 100 KMC simulations with end time $1 \cdot 10^6$ s and transition rates calculated by the method by Liang et al. [1]. Only data points with relative error below 0.30 have been included.

This section will show and discuss results for the clustering rates and cluster composition logged by the cluster rate observer from 100 different KMC simulation runs. The

transition rates have been calculated using the method by Liang et al. [1] and the end-time for each simulation is $1 \cdot 10^6$ s.

In Figure 5.26 the evaporation and condensation rate for Mg and Si is shown as a function of the number of solutes in the cluster, n. Only data points with relative error below 0.30 have been included. To calculate the relative error the average of each run has been treated as a single data point to avoid getting too low error estimates due to correlation between iterations in the KMC simulations. From the figure the condensation rate for Mg is the highest, followed by the evaporation rate for Si, condensation rate for Si and the evaporation rate for Mg. The Mg-Si ratio is close to 1 for the clusters, even though the condensation rate is much higher than the evaporation rate for Mg while they are similar for Si. One possible explanation for this is that not all size changing events are registered, only the ones where a single solute leaves or joins one cluster is registered due to the formulation of cluster dynamics. This means that there can be many ways to merge or split clusters that does not get registered.

5.7.1 Fitting of analytical expression to evaporation rates



(a) Mg: $A = 1.14 \cdot 10^3, c = -2.78, \bar{\sigma} = 0.078$ **(b)** Si: $A = 1.08 \cdot 10^5, c = -3.02, \bar{\sigma} = 0.067$

Figure 5.27: A nonlinear least squares fit of a log-spaced selection of the data from Figure 5.26 to the expression for the evaporation rate from cluster dynamics, given by equation (A.6). The average interfacial free energy for clusters, $\bar{\sigma}$, is given in J m^{-2}, A is given in s^{-1} and c is dimensionless. The value for A, c and $\bar{\sigma}$ for the two fits are listed in the caption of the subfigures.

The expression for the evaporation rate given in equation (A.6), which uses the capillary approximation and assumes spherical clusters, includes the average interfacial free energy for clusters. It is therefore possible to obtain an estimate for the interfacial energy by fitting the expression to the logged rates from the KMC simulations. The results from the fitting procedure, for both Mg and Si, is shown in Figure 5.27. The fitting has been performed on a log-spaced selection of cluster sizes to weight each interval of cluster sizes equally. As seen from the figure the smallest cluster sizes still have a higher evaporation rate than the fitting predicts, this is much more prominent if all the data points are used. The average interfacial free energy obtained from the fitting to the Mg rates is $\bar{\sigma}_{\mathrm{Mg}} = 0.078\,\mathrm{J\,m^{-2}}$ and for the Si rates it is $\bar{\sigma}_{\mathrm{Si}} = 0.067\,\mathrm{J\,m^{-2}}$. Both values are similar,

as they should be, even though the prefactor A differs by two orders of magnitude. The values are also close to the values found by DFT slab calculations on the $L1_0$ structure that was performed by Kleiven et al. in [31]. They found the interfacial free energy for the interface where the surface normal of the Mg and Si layers are parallel to the interface to be $\gamma_{\parallel} = 30\,\mathrm{mJ\,m^{-2}}$. For the calculations where the interface is perpendicular to the surface normals, the average of the interfacial free energy for the Al-Mg and Al-Si interfaces, the interfacial free energy is $\gamma_{\perp} = 75\,\mathrm{mJ\,m^{-2}}$. The average interface free energy found from the evaporation rates, $\bar{\sigma}$, will be a combination of both the normal and parallel interfaces since clusters consist of a combination of these two.

5.7.2 Cluster composition

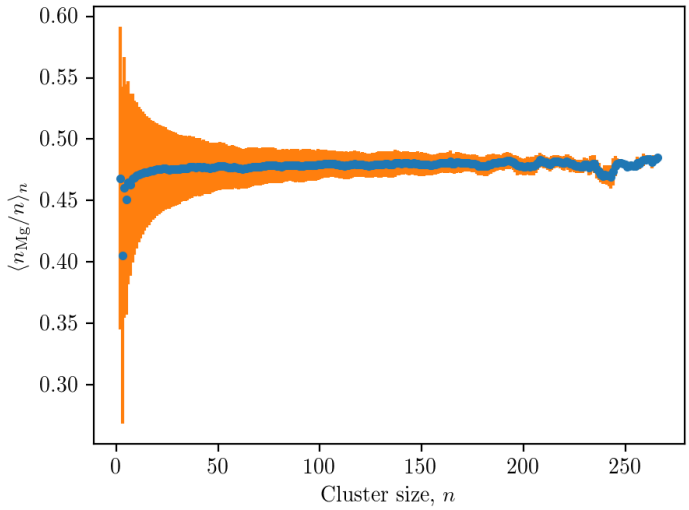


Figure 5.28: The average Mg percentage in clusters of size n is plotted as blue dots, and the orange lines are $\pm$ one standard deviation. Data is gathered from 100 different runs starting from different random configurations. End time for the simulations are $1 \cdot 10^6$ s and the transition rates have been calculated with the method of Liang et al. [1].

One of the new features of the cluster rate observer is that the composition of a cluster is logged every time a clustering event is logged. This information can be used to find the average cluster composition as a function of cluster size. The results for the average cluster composition, based on data from 100 KMC runs with the Liang et al. [1] method, is shown in Figure 5.28. The blue dots are the average values and the orange lines are $\pm$ one standard deviation. The standard deviation decrease fast in the beginning and the average value for the Mg content seems to settle at ≈ 0.47. The blue dot in the beginning that are much lower than the rest is for $n = 3$ and indicates that clusters with 2 or 3 Si atoms are more common than clusters with 2 or 3 Mg atoms a for clusters of size 3. From inspection

of the raw data, clusters of pure Si of size 2 is observed 87 times more often than clusters of pure Mg of size 2. For clusters of size 3 the same ratio is 128, for larger clusters pure Si or Mg is observed much more seldom, but pure Si remains more common than Mg.

5.8 Error sources and discussion of overall results

This section will discuss the overall results presented in this chapter and discuss some of the error sources caused by the model for the system, the calculations in the training set and the two different expressions for the activation energies.

5.8.1 Limitations imposed by the model

The two main limitations set by the model is that every atom must reside at an FCC lattice site and the only possible jumps are nearest neighbour jumps. These two limitations are coupled to each other, if the lattice representation changes the possible jumps should be re-evaluated. For the current FCC lattice, it would be possible to allow next nearest neighbours jump to the vacant site, but this would mean that 18 transition rates must be calculated for each iteration instead of 12. Another disadvantage of this is that the 6 additional rates would most likely need their own expression, which for the cluster expansion based method means that a new training set for next nearest neighbour jumps must be created. Adding these jumps will not alter the type of clusters that can be constructed since every atom still must reside at an FCC lattice site, but it is possible that it affects the kinetics.

The FCC lattice restriction is more important to lift since it does not allow the important β'' precipitates to form. By adding vacant octahedral lattice sites, the site in the middle of the FCC unit cell, it is possible to represent the β'' structure. Adding the octahedral sites means that the set of possible jump events must be updated to include jumps in and out of octahedral sites. It is important that this is done in a way that preserve the ergodicity of the simulation. With the new lattice sites the cluster expansion for regular nearest neighbour jumps must be updated and trained to include the effect of occupied octahedral sites close to the jump. A separate cluster expansion must be trained for calculating the activation energy for jumps in and out of the octahedral sites. This will require some changes to the implementation and new DFT calculations must be performed, but it is an important step towards getting better and more realistic precipitate structures in the simulations.

Both main limitations mentioned here could also be lifted by using the off-lattice KMC implementation k-ART [20, 21], which identifies the possible transitions based on the local environment during run time. Another advantage of the method is that it includes lattice deformations and long-range interactions, which are not possible to capture equally well in the on-lattice model. Some of the effects are included implicitly in the on-lattice model through the fact that the cluster expansion based method of determining the activation energies is trained on DFT calculations which include these effects. The predicted activation energies are therefore based on a relaxed lattice even though the lattice is represented by a fixed lattice in the KMC simulations. This will be discussed in more detail in the next subsection. The disadvantage of this method is the much higher computational cost, and it also needs to have a good force potential for the studied system to give good results.

Since long time scales are needed to study the precipitation process in Al-Mg-Si alloys it is beneficial to use the on-lattice method, given that the octahedral sites are successfully added to the model.

5.8.2 Error sources for the training set

The expression for the activation energies given by equation (3.4) is able to represent the training set very well as shown in Figure 5.4, but the simulations does not produce any larger clusters. This problem can have several causes, the training set is missing important information, error sources in the DFT calculations or poor fitting of the ECIs to the training set. This subsection will discuss the two causes regarding the training set.

The training set will never be complete because of the vast number of possible local environments, but it is important that the training set contain a wide variety of environments that the KMC simulation may encounter. Currently the training set only have a limited set of structures that contain more than a few solutes as can be seen from Table 5.2. When the simulations encounter structures with many solutes the calculated activation energy is only based on the information available from the structures with few solutes, which may give poor results. The energy differences between adjacent states calculated from CE_MC shown in Figure 5.22 have a much wider range of ΔE than the training set does, see Figure 5.19. This shows that the training set is missing information about activation energies between states that have a large energy difference. Some of the structures related to large energy differences are likely to include many solute neighbours since it is the simulation that was started from a clustered state that gave the largest energy differences in Figure 5.22. Structures with more solute neighbours should therefore be computed by DFT and added to the training set.

There are several choices that must be made when calculating the acitvation energy with DFT that can affect the activation energies to some degree. One of the choices is if the cell volume should be fixed or relaxed. For calculating the activation energies for the NVT ensemble it seems logical to keep the volume fixed, but it would also be possible to run the simulations in the NPT ensemble instead and relax the lattice volume in the DFT simulations. The lattice can still be fixed in the KMC since the barriers are based on the information in the training set and not the actual positions of the atoms in the KMC lattice, and no other changes would need to be done to the KMC since N and T is still constant. For the calculations presented in section 5.1 the cell volume was fixed but the lattice positions were relaxed. With more Mg atoms the pressure in the DFT simulations are observed to increase, due to the positive misfit volume of Mg compared to Al, which can cause the activation energy for some jumps to become lower since the jumping atom can be pushed towards the vacant site due to the increased pressure.

Another important factor is that the activation energies calculated by DFT does include long-range interactions and strain effects. This means that these effects are implicitly included in the cluster expansion even though the atoms that are causing these effects in the DFT not necessarily are on one of the numbered lattice sites in Figure 3.1. For the calculations of the training set presented in section 5.1, all the sites that are not part of the cluster expansion have been set to Al. This means that the cluster expansions ECIs does not represent an average over different environments outside the included sites since all of them are Al. It should be tested how much this affects the activation energy by doing some

test calculations with DFT. This might be extra important for jumps that are typically on the interface between Al and a cluster, and inside clusters, since the clusters usually induce some strain due to misfit. The reason for not including solutes outside the numbered sites was that the calculations were faster to perform with only Al since they were closer to the NEB calculation for jumps in pure Al.

The size of the simulation cell in the DFT calculations can also affect the activation energy. For calculating the training set for this work a cell size of $3 \times 3 \times 3$ was used, which saves a lot of time compared to using a larger cell, such as $4 \times 4 \times 4$. This will as stated earlier affect the activation energy, even if only Al atoms are added to make the simulation cell larger. Due to the periodic boundary conditions some of the different sites shown in Figure 3.1 will effectively be closer to each other than they appear in the figure, which can influence the results. A test of how much this affects the activation energies for some of the structures with the most solute neighbours have been requested but was not made available in time.

5.8.3 Error sources in the transition rates

This subsection will first discuss the effect of the parameters used in the transition rates that are common for the method by Liang et al. and the cluster expansion based method presented in this thesis. The vacancy concentration has already been discussed in subsection 5.6.2 that presented the diffusivities, since $\Delta t \propto 1/C_{\mathrm{V}}$ and $D \propto 1/t$. It can be difficult to find the true vacancy concentration since it can depend on the temperature history of the alloy and can also change over time. Since the parameter is only used to scale the time per iteration in the KMC simulation linearly and does not affect the outcome of the simulation it is not a big error source for other results than the diffusivities.

The jump attempt frequencies, given in Table 2.1, affect the transition rates linearly and therefore also affects the outcome of the simulations. It will also affect the order of the diffusivities indirectly since higher transition rates for one atom type will lead to that type being picked more often which can lead to higher diffusivity for that type. All of the jump attempt frequencies are in the order of $1 \cdot 10^{13}\,\mathrm{s}^{-1}$ and are based on the values from [44]. According to [21] the jump attempt frequency for all types of KMC simulations are usually in the order of $1 \cdot 10^{12}\,\mathrm{s}^{-1}$ to $1 \cdot 10^{13}\,\mathrm{s}^{-1}$.

The size of the simulation system is set to $25 \times 25 \times 25$ unit cells, this will set a restriction on the size of clusters and precipitates that can be observed. For the CE method this is not a concern yet, but for the method by Liang et al. the size of the largest cluster will be limited after running the simulation for some time. One example of this is the large cluster shown in subfigure 5.10c which contains 595 out of the 900 solutes in the system.

The method by Liang et al. [1] consistently produces the $\mathrm{L1}_0$ structure and follows the total energy very well. Tracking the total energy well does not necessarily imply that the activation energies are also good, since the total energy only depends on the difference between pairs of activation energies. The single activation energies have relatively poor agreement with the DFT calculations and the range of activation energies for Al jumps is much narrower than the range for Mg and Si jumps. The narrow range for Al jumps is not observed in the DFT calculations or in the cluster expansion method. At only $350\,\mathrm{K}$ the large $\mathrm{L1}_0$ clusters dissolve and the system returns to solid solution, this temperature seems low considering the increase in zero-point energy is more than $90\,\mathrm{eV}$. From running sev-

eral hundred simulations with the method during my specialisation project and this master thesis, all the observed large clusters have a $L1_0$ structure, it therefore seems unlikely that this method will produce any other structures of interest.

The method for calculating the activation energies that was developed during this master thesis is based on the principle of cluster expansions and is able to represent the training set well and have good computational efficiency. Several methods have been tested to avoid overfitting, which can be a problem since there are only 15 more unique structures in the training set than there are ECIs in the used expression. Both the L-curve criterion and LOOCV agree well and give similar results, which is a good indication that they find a close to optimal amount of regularisation. The L-curve in Figure 5.2 also shows that the amount of regularisation have been varied from under-regularised, high solution norm, to over-regularised, high residual norm during the hyperparameter optimisation. This is important since the best LOOCV score does not give the best fit if none of the attempted hyperparameter combinations are close to the optimal regularisation amount. The LOOCV score for the chosen fit is $9.86\,\mathrm{meV}$, which is relatively good considering this is for the total structure and not per atom. This score must however be treated as a minimum estimate since it is only calculated from structures in the training set and will probably give less accurate predictions for structures that are very different from any of the structures it is trained on.

The results from running the simulations are unfortunately not good enough yet and show no clustering when starting from randomly distributed solutes. It is however difficult to know how the correct clustering evolution will look like in KMC simulations since it is not guaranteed that the method by Liang et al. shows the correct evolution even though it produces $L1_0$ clusters. Regardless of how the correct clustering evolution will look like in KMC simulations, the CE method currently have a problem with a negative drift for the total energy, and it is present both when the total energy decreases and when it increases. The lack of clustering and drift of total energy might be caused by poor fitting of the ECIs, but this seems unlikely due to the measures taken to find the optimal fit. Another possible cause is that the training set is missing important information, or that the information in the training set is not representative for the structures encountered in the KMC simulation, which was discussed in more detail in the previous subsection.

Chapter 6

Conclusion

The goal of this master thesis has been to develop a new and more physically accurate way to calculate activation energies needed for KMC simulations of Al-Mg-Si alloys. The purpose is to get better insight into the kinetics of early stages of precipitation in Al-Mg-Si alloys at an atomic level.

During this master thesis a new method for calculating the activation energies needed in KMC simulations based on the concept of cluster expansion has been developed, presented and thoroughly tested. The expression have been trained on a set of structures with activation energies calculated by DFT, which resulted in a low RMSE of $1.77\,\mathrm{meV}$ and LOOCV score of $9.86\,\mathrm{meV}$. Both minimisation of the LOOCV score and the L-curve criterion have been tested to avoid overfitting the model and both gave similar results. The fitted expression predicts that some structures have negative activation energies. Most of these structures will likely have low, but positive, activation energies when calculated with DFT, but some of them may be negative. If an activation energy is negative it means that the initial structure is not a local energy minimum and therefore not stable, which indicates that the structure is not FCC and can therefore not be represented in the current version of the code.

A workaround to be able to use the current fit to run simulations that conserves the energy difference between states have been developed and used. Inspection of simulation results show that no large clusters form at room temperature and large clusters dissolve if the temperature is $T = 400\,\mathrm{K}$ or higher. The diffusivities are close to the literature values if the equilibrium vacancy concentration is used. Total energy relative to the initial configuration of the lattice have a negative drift compared to the CE_MC calculations. A probable cause for both the total energy drift and lack of clustering is that the training set has a fairly limited amount of variation in the structures it contains and is therefore missing important information.

The runtime of the method is low, and a single KMC iteration, which includes calculation of 12 transition rates, uses $40\,\mu\mathrm{s}$ when running on a single CPU core. This leaves room for including more lattice sites and higher order terms in the expansion and still be able to reach long timescales. Due to the limited amount of information in the training

set it is currently not possible to include more sites and not beneficial to include terms for clusters larger than two.

An in-depth inspection of the method by Liang et al. [1] have been performed during this work. Simulation results show that the method consistently produces large clusters of the $L1_0$ structure at room temperature, but dissolves them if the temperature is elevated to $T = 350\,\mathrm{K}$. This temperature is relatively low while the increase in zero-point energy due to dissolving the clusters is over $90\,\mathrm{eV}$, which indicates that the clusters should be stable at this temperature. It is the activation energies that determines the evolution of the KMC simulations, this indicates that the activation energies are not very good since they do not behave as predicted when the temperature is changed from room temperature. The diffusivities are a few orders of magnitude too high compared to the literature, but this can be related to the high vacancy concentration that is used. Total energy evolution is tracked well and only have a small drift compared to the total energy calculations using the CE_MC method. Clustering rates have been logged and the analytical expression for the evaporation rate have been fitted to the data. The average interfacial energy found from this line fitting is very close to literature values for the $L1_0$ structure. Benchmarking show that the method is not consistent with the activation energies calculated by DFT and testing on a large pool of structures show that the Al jumps have a suspiciously narrow range of activation energies.

Overall, the results show that the method by Liang et al. gives good results for room temperature and the $L1_0$ structure, but benchmarking show that the activation energies are not physically accurate. A way to calculate more physically accurate activation energies for the Al-Mg-Si system is needed to obtain information about other cluster structures than the $L1_0$ structure. The developed expression based on cluster expansion shows good potential, represents the training set very well and is a fast method considering the amount of terms included. In the current state it does not produce good simulation results, but a larger and more varied training set will likely improve this.

Chapter 7

Future work

This section will present suggestions for how the KMC simulations can be improved to give better simulation results.

7.1 Improving the activation energies

The current training set is accurately represented by the cluster expansion given by equation (3.4), but the expansion does not perform well in simulations. This indicates that the training set is missing important information that the expansion is not able to extrapolate well. A thorough check of the existing calculations should also be performed to ensure that the information available is representative for the local environments the KMC simulation encounters.

If more solutes are added to the DFT calculations it is easy to expand the cluster expansion to include more lattice sites. Higher order terms can also be implemented if the expansion is not able to represent the new training set well enough with second order terms.

7.2 Adding octahedral sites

Octahedral lattice sites should be added to the program to make it possible to form and represent the important β'' precipitates. Adding the lattice sites is easy and the octahedral sites can be given a different site tag in the program than the FCC sites to differentiate them. This will require some changes to the activation energy calculation, since a separate expression need to be used for jumps to and from octahedral sites. The addition of the octahedral sites will also require a change in the expression for the activation energies for the nearest neighbour jumps, since any occupied octahedral site close to the jump will affect the barrier. Both these changes to the activation energies will require new DFT calculations to the train the model on.

During this work it can be a good idea to make a modular system for event generation that generates the possible events and assigns the correct activation energy function to each of them. Care should be taken to ensure that the event generation leads to an ergodic system.

7.3 Other improvements

Implementing a method for handling flickering states efficiently should be considered since the DFT calculations have shown that there are some very low activation energies. The method used in the k-ART implementation, the basin-autoconstructing Mean Rate Method, calculates the average escape time from the basin of flickering states [21]. It should be investigated if this method would be a good solution for the on-lattice KMC program used in this thesis.

Precipitates in the Al-Mg-Si system can contain aluminium, this is the case for the β'' precipitates, but the current cluster identification only looks for solutes connected through NN interactions. This has been sufficient so far since the method by Liang et al. [1] only produces the $L1_0$ structure, which only consist of solutes. The cluster identification should therefore be changed to a method that is not purely based on solute NN interactions. Identification of different types of clusters and precipitates will also be a useful addition, since this would allow logged quantities to be dependent on the type of cluster or precipitate.

Bibliography

[1] Z. Liang, C. S. T. Chang, C. Abromeit, J. Banhart, and J. Hirsch. The kinetics of clustering in Al-Mg-Si alloys studied by Monte Carlo simulation. *International Journal of Materials Research*, 103(8):980–986, 2012. doi: https://doi.org/10.3139/146.110798.

[2] R. N. Lumley. *Fundamentals of Aluminium Metallurgy : Production, Processing and Applications.* Woodhead Publishing in Materials. Woodhead Publishing, 2011. ISBN 9781845696542.

[3] J. Hirsch. Automotive trends in aluminium - the european perspective. *Materials Forum*, 28:15, 01 2004.

[4] C. D. Marioara, S. J. Andersen, H. W. Zandbergen, and R. Holmestad. The influence of alloy composition on precipitates of the Al-Mg-Si system. *Metallurgical and Materials Transactions A*, 36(13):691–702, 2005. doi: 10.1007/s11661-005-1001-7.

[5] E. Clouet, M. Nastar, A. Barbu, C. Sigli, and G Martin. Precipitation in Al-Zr-Sc alloys: a comparison between kinetic Monte Carlo, cluster dynamics and classical nucleation theory. *arXiv preprint cond-mat/0507259*, 2005.

[6] A. de Moura and A. Esteves. Simulation of the nucleation of the precipitate Al_3Sc in an aluminum scandium alloy using the kinetic Monte Carlo method. In *2013 13th IEEE International Conference on Nanotechnology (IEEE-NANO 2013)*, pages 438–441, Aug 2013. doi: 10.1109/NANO.2013.6721003.

[7] G. Sha and A. Cerezo. Kinetic monte carlo simulation of clustering in an Al–Zn–Mg–Cu alloy (7050). *Acta Materialia*, 53(4):907 – 917, 2005. doi: https://doi.org/10.1016/j.actamat.2004.10.048.

[8] Ø. T. Nygård. Atomistic simulations of the kinetics of early stages of precipitation in Al-Mg-Si alloys using kinetic Monte Carlo. Specialisation project at NTNU, 2020.

[9] S.J. Andersen, H.W. Zandbergen, J. Jansen, et al. The crystal structure of the β'' phase in Al–Mg–Si alloys. *Acta Materialia*, 46(9):3283 – 3298, 1998. doi: https://doi.org/10.1016/S1359-6454(97)00493-X.

[10] P. H. Ninive, A. Strandlie, S. Gulbrandsen-Dahl, et al. Detailed atomistic insight into the β'' phase in Al-Mg-Si alloys. *Acta Materialia*, 69:126 – 134, 2014. doi: https://doi.org/10.1016/j.actamat.2014.01.052.

[11] A. B. Bortz, M. H. Kalos, and J. L. Lebowitz. A new algorithm for Monte Carlo simulation of Ising spin systems. *Journal of computational physics*, 17(1):10–18, 1975. doi: https://doi.org/10.1016/0021-9991(75)90060-1.

[12] M. Stamatakis and D. G. Vlachos. Unraveling the complexity of catalytic reactions via kinetic Monte Carlo simulation: Current status and frontiers. *ACS Catalysis*, 2 (12):2648–2663, 2012. doi: 10.1021/cs3005709.

[13] D. Molnar, R. Mukherjee, A. Choudhury, et al. Multiscale simulations on the coarsening of Cu-rich precipitates in α-Fe using kinetic Monte Carlo, molecular dynamics and phase-field simulations. *Acta Materialia*, 60(20):6961 – 6971, 2012. ISSN 1359-6454. doi: https://doi.org/10.1016/j.actamat.2012.08.051.

[14] M. Yu. Lavrentiev, D. Nguyen-Manh, and S. L. Dudarev. Cluster expansion models for Fe-Cr alloys, the prototype materials for a fusion power plant. *Computational Materials Science*, 49(4, Supplement):S199 – S203, 2010. ISSN 0927-0256. doi: https://doi.org/10.1016/j.commatsci.2010.04.033.

[15] M. Slabanja and G. Wahnström. Kinetic Monte Carlo study of Al-Mg precipitation. *Acta Materialia*, 53(13):3721 – 3728, 2005. ISSN 1359-6454. doi: https://doi.org/10.1016/j.actamat.2005.04.024.

[16] S. Müller, L-W Wang, and A. Zunger. First-principles kinetic theory of precipitate evolution in Al-Zn alloys. *Modelling and Simulation in Materials Science and Engineering*, 10(2), 2002. doi: https://doi.org/10.1088/0965-0393/10/2/303.

[17] D. E. Makarov and H. Metiu. A model for the kinetics of protein folding: Kinetic Monte Carlo simulations and analytical results. *The Journal of Chemical Physics*, 116(12):5205–5216, 2002. doi: 10.1063/1.1450123.

[18] J. S. Kwon, M. Nayhouse, P. D. Christofides, and G. Orkoulas. Modeling and control of crystal shape in continuous protein crystallization. *Chemical Engineering Science*, 107:47 – 57, 2014. ISSN 0009-2509. doi: https://doi.org/10.1016/j.ces.2013.12.005.

[19] M. E. J. Newman and G. T. Barkema. *Monte Carlo Methods in Statistical Physics*. Oxford Univerity Press, 2011.

[20] L. K. Béland, P. Brommer, F. El-Mellouhi, and N. Joly, J.F. Mousseau. Kinetic activation-relaxation technique. *Phys. Rev. E*, 84:046704, Oct 2011. doi: 10.1103/PhysRevE.84.046704.

[21] N. Mousseau, L. K. Béland, P. Brommer, et al. Following atomistic kinetics on experimental timescales with the kinetic Activation-Relaxation Technique. *Computational Materials Science*, 100:111 – 123, 2015. doi: https://doi.org/10.1016/j.commatsci.2014.11.047. Special Issue on Advanced Simulation Methods.

[22] N. Sandberg, M. Slabanja, and R. Holmestad. Ab initio simulations of clustering and precipitation in Al-Mg-Si alloys. *Computational Materials Science*, 40(3):309 – 318, 2007. ISSN 0927-0256. doi: https://doi.org/10.1016/j.commatsci.2007.01.001.

[23] D. Kleiven, O. L. Ødegård, K. Laasonen, and J. Akola. Atomistic simulations of early stage clusters in Al-Mg alloys. *Acta Materialia*, 166:484 – 492, 2019. ISSN 1359-6454. doi: https://doi.org/10.1016/j.actamat.2018.12.050.

[24] A. R. Leach. *Molecular Modelling Principles and Applications*. Prentice Hall, 2nd ed edition, 2001.

[25] J. L. Klepeis, K. Lindorff-Larsen, R. O. Dror, and D. E. Shaw. Long-timescale molecular dynamics simulations of protein structure and function. *Current Opinion in Structural Biology*, 19(2):120 – 127, 2009. doi: https://doi.org/10.1016/j.sbi.2009.03.004. Theory and simulation / Macromolecular assemblages.

[26] D. E. Shaw, R. O. Dror, J. K. Salmon, et al. Millisecond-Scale molecular dynamics simulations on anton. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 1–11, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605587448. doi: 10.1145/1654059.1654126.

[27] W. Verestek, A. P. Prskalo, M. Hummel, P. Binkele, and S. Schmauder. Molecular dynamics investigations of the strengthening of Al-Cu alloys during thermal ageing. *Physical Mesomechanics*, 20(3):291–304, 2017. doi: 10.1134/S1029959917030055.

[28] K. M. Carling, G. Wahnström, T. R. Mattsson, N. Sandberg, and G. Grimvall. Vacancy concentration in Al from combined first-principles and model potential calculations. *Phys. Rev. B*, 67:054101, Feb 2003. doi: 10.1103/PhysRevB.67.054101.

[29] X. Zhang and M. H.F. Sluiter. Cluster expansions for thermodynamics and kinetics of multicomponent alloys. *Journal of Phase Equilibria and Diffusion*, 37:44–52, 2016. doi: 10.1007/s11669-015-0427-x.

[30] Q. Wu, B. He, T. Song, J. Gao, and S. Shi. Cluster expansion method and its application in computational materials science. *Computational Materials Science*, 125:243 – 254, 2016. ISSN 0927-0256. doi: https://doi.org/10.1016/j.commatsci.2016.08.034.

[31] D. Kleiven and J. Akola. Precipitate formation in aluminium alloys: Multi-scale modelling approach. *Acta Materialia*, 195:123 – 131, 2020. ISSN 1359-6454. doi: https://doi.org/10.1016/j.actamat.2020.05.050.

[32] A. Van der Ven, G. Ceder, M. Asta, and P. D. Tepesch. First-principles theory of ionic diffusion with nondilute carriers. *Phys. Rev. B*, 64:184307, Oct 2001. doi: 10.1103/PhysRevB.64.184307.

[33] T. Rehman, M. Jaipal, and C. Abhijit. A cluster expansion model for predicting activation barrier of atomic processes. *Journal of Computational Physics*, 243:244 – 259, 2013. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2013.03.005.

[34] E. Lee, F. B. Prinz, and W. Cai. Ab initio kinetic Monte Carlo model of ionic conduction in bulk yttria-stabilized zirconia. *Modelling and Simulation in Materials Science and Engineering*, 20(6), 2012. ISSN 0021-9991. doi: https://doi.org/10.1088/0965-0393/20/6/065006.

[35] J. Nielsen, M. d'Avezac, J. Hetherington, and M. Stamatakis. Parallel kinetic Monte Carlo simulation framework incorporating accurate models of adsorbate lateral interactions. *The Journal of Chemical Physics*, 139(22):224706, 2013. doi: 10.1063/1.4840395.

[36] T. Mueller and G. Ceder. Bayesian approach to cluster expansions. *Phys. Rev. B*, 80: 024103, Jul 2009. doi: 10.1103/PhysRevB.80.024103.

[37] J. Sanchez. Cluster expansions and the configurational energy of alloys. *Physical review. B, Condensed matter*, 48:14013–14015, 12 1993. doi: 10.1103/PhysRevB.48.14013.

[38] J.M. Sanchez, F. Ducastelle, and D. Gratias. Generalized cluster description of multicomponent systems. *Physica A: Statistical Mechanics and its Applications*, 128 (1):334 – 350, 1984. ISSN 0378-4371. doi: https://doi.org/10.1016/0378-4371(84)90096-7.

[39] K. Burke and L. O. Wagner. Dft in a nutshell. *International Journal of Quantum Chemistry*, 113(2):96–101, 2013. doi: 10.1002/qua.24259.

[40] G. Henkelman and H. Jónsson. Improved tangent estimate in the nudged elastic band method for finding minimum energy paths and saddle points. *The Journal of Chemical Physics*, 113(22):9978–9985, 2000. doi: 10.1063/1.1323224.

[41] G. Henkelman and H. Jónsson. A dimer method for finding saddle points on high dimensional potential surfaces using only first derivatives. *The Journal of Chemical Physics*, 111(15):7010–7022, 1999. doi: 10.1063/1.480097.

[42] P. C. Hansen and D. P. O-Leary. The use of the L-Curve in the regularization of discrete ill-posed problems. *SIAM Journal on Scientific Computing*, 14(6):1487–1503, 1993. doi: 10.1137/0914086.

[43] J. E. Mebius. Derivation of the Euler-Rodrigues formula for three-dimensional rotations from the general formula for four-dimensional rotations. *arXiv Mathematics e-prints*, January 2007.

[44] M. Mantina, Y. Wang, L.Q. Chen, Z.K. Liu, and C. Wolverton. First principles impurity diffusion coefficients. *Acta Materialia*, 57(14):4102 – 4108, 2009. ISSN 1359-6454. doi: https://doi.org/10.1016/j.actamat.2009.05.006.

[45] O. M. Løvvik, E. Sagvolden, and Y. J. Li. Prediction of solute diffusivity in Al assisted by first-principles molecular dynamics. *Journal of Physics: Condensed Matter*, 26(2):025403, dec 2013. doi: 10.1088/0953-8984/26/2/025403.

[46] G. Kresse and J. Hafner. Ab initio molecular dynamics for liquid metals. *Phys. Rev. B*, 47:558–561, Jan 1993. doi: 10.1103/PhysRevB.47.558.

[47] G. Kresse and J. Furthmüller. Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set. *Computational Materials Science*, 6(1):15 – 50, 1996. doi: https://doi.org/10.1016/0927-0256(96)00008-0.

[48] J. P. Perdew, J. A. Chevary, S. H. Vosko, et al. Atoms, molecules, solids, and surfaces: Applications of the generalized gradient approximation for exchange and correlation. *Phys. Rev. B*, 46:6671–6687, Sep 1992. doi: 10.1103/PhysRevB.46.6671.

[49] P. C. Hansen, T. K. Jensen, and G. Rodriguez. An adaptive pruning algorithm for the discrete L-curve criterion. *Journal of Computational and Applied Mathematics*, 198 (2):483 – 492, 2007. doi: https://doi.org/10.1016/j.cam.2005.09.026. Special Issue: Applied Computational Inverse Problems.

[50] J. H. Chang, D. Kleiven, M. Melander, et al. CLEASE: a versatile and user-friendly implementation of cluster expansion method. *Journal of Physics: Condensed Matter*, 31(32):325901, may 2019. doi: 10.1088/1361-648x/ab1bbc.

[51] E. Clouet, A. Barbu, L. Laé, and G. Martin. Precipitation kinetics of Al_3Zr and Al_3Sc in aluminium alloys modeled with cluster dynamics. *Acta Materialia*, 53(8):2313 – 2325, 2005. ISSN 1359-6454. doi: https://doi.org/10.1016/j.actamat.2005.01.038.

Appendix A

Introduction to cluster dynamics

This chapter is an excerpt from my specialisation project [8] and introduces cluster dynamics. The introduction is useful for understanding the changes done to the cluster rate observer and for the discussion of the results presented in section 5.7.

A.1 Cluster dynamics

Cluster dynamics is a mesoscopic model for modelling time evolution of cluster size distribution in materials. This method has a very low computational cost compared to KMC and is suited to simulate larger systems for larger timescales. The limitation of the model is that it is not possible to study structures of precipitates and spatial distribution as only the cluster size distribution is available. The introduction of cluster dynamics given in this section is for a binary alloy and is based on the work by Clouet et al. [51]. The governing equations for cluster dynamics are

$$\frac{dC_n}{dt} = J_{n-1\to n} - J_{n\to n+1} \quad \forall n \geq 2 \tag{A.1a}$$

$$\frac{dC_1}{dt} = -2J_{1\to 2} - \sum_{n\geq 2} J_{n\to n+1}, \tag{A.1b}$$

where C_n is the probability of observing a cluster of size n and $J_{n\to n+1}$ is the flux of clusters going from size n to $n+1$. The fluxes can be calculated as

$$J_{n\to n+1} = \beta_n C_n - \alpha_{n+1} C_{n+1}, \tag{A.2}$$

where β_n is the condensation rate for clusters of size n and α_n is the evaporation rate. The condensation rate is the rate at which a cluster of size n receives a single solute from solid solution and becomes size $n+1$. The evaporation rate α_n is correspondingly the rate for a cluster of size n loose a single solute and becomes size $n-1$.

By solving the diffusion problem under the assumption of the clusters being spherical, the condensation rate can be obtained

$$\beta_n = 4\pi r_n \frac{D}{\Omega} C_1, \tag{A.3}$$

where D is the diffusivity for the solute in pure aluminium and Ω is the average volume per FCC lattice site.

The evaporation rate cannot be obtained directly. In [51] the expression for the evaporation rate is determined by assuming that the evaporation rate does not depend on the surrounding solid solution, then solving the governing equations, equation (A.1), with zero fluxes and divide the free energy of forming the cluster into a volume and surface contribution. The resulting expression for the evaporation rate is

$$\alpha_{n+1} = 4\pi r_n \frac{D}{\Omega} \exp\left[(36\pi)^{1/3} a^2 ((n+1)^{2/3}\sigma_{n+1} - n^{2/3}\sigma_n - \sigma_1)/k_{\mathrm{B}}T\right], \tag{A.4}$$

where σ_n is the interface free energy for a cluster of size n. To approximate σ_n an extension of the capillary approximation is used

$$\sigma_n = \bar{\sigma}(1 + cn^{-1/3} + dn^{-2/3}), \tag{A.5}$$

where c is a constant for the line contribution, d is a constant for the point contribution and $\bar{\sigma}$ is the isotropic average interface free energy. By assuming spherical clusters, $r_n \propto n^{1/3}$, and inserting equation (A.5) into (A.4) the expression for α_n can be written as

$$\alpha_n = An^{1/3} \exp\left[(36\pi)^{1/3} a^2 \bar{\sigma}\{(n+1)^{2/3} - n^{2/3} + c((n+1)^{1/3} - n^{1/3})\}/k_{\mathrm{B}}T\right], \tag{A.6}$$

where A is a temperature dependent constant given by

$$A = \left(\frac{4\sqrt{3}\pi}{\Omega}\right)^{2/3} D \exp\left[-(36\pi)^{1/3} a^2 \bar{\sigma}\{1 + c + d\}/k_{\mathrm{B}}T\right]. \tag{A.7}$$

Appendix B

User manual and documentation

This appendix includes some user manuals for how to get started with using the code. It will cover how to setup and run simulations, show the different available observers and how to implement new observers, how to setup a cluster expansion, including creation of training set and fitting of ECIs, a table of the interpretation of all the variables in equation (3.4) and documentation of the Python interface. The project is currently in a private repository on gitlab under NTNUMaterialsTheory/skmc. The project is setup such that gitlabs servers will automatically compile the code and run the implemented test cases when pushing commits, which ensures that the code is in a working condition at all times. When implementing new features, new tests should also be added to ensure that the code is always in a working condition. All Python scripts referred to in this appendix is stored in the folder scripts of the skmc repository while the C code is stored in the src folder.

B.1 Running KMC simulations

With the new Python interface, it is easier to setup KMC simulations, either by modifying the existing script run_KMC.py or by creating a script from scratch. The default script, run_KMC.py, have many command line options, which are listed in Table B.1 together with the default values. The command line options are very useful when performing many simulations with the same logging scheme since then they can all be launched manually from the terminal or from a simple bash script without altering the Python script.

For changing the logging scheme, it is necessary to edit the lines in run_KMC.py that adds observers to the KMC object. For more information on the available observers and the meaning of the different parameters for setting the logging schedule for each observer see the next subsection.

Setting up the lattice is currently either done by the KMC script with a random start configuration or from a POSCAR file. The POSCAR file needs to include the vacancy position and mark the type of the atom as ”X” for the code to automatically find the position of the vacancy which is needed for running the simulation. Setup of a lattice from a POSCAR file is done by a script that uses the Python interface to make an empty lattice

with the correct cell size and then each lattice site with the correct atom type is added one by one. It is therefore easy to implement support for other lattice formats such as XYZ by implementing a short Python script similar to the one already implemented in read_poscar.py.

If the simulation is using the cluster expansion based rate calculator, selected by the ”-r CE” command line option, then the filepath to a json file needs to be specified with the option ”-j”. The setup of the cluster expansion is automatic except from the user needs to set ”--Ndist” to a large enough value manually to ensure that the neighbourlist contains all neighbours that are included in the clusters of the cluster expansion.

Table B.1: The default value and description for all the optional arguments to the script run_KMC.py. For most of the options there is a shortcut (-) in addition to the long option name (--). If vacancy concentration is not set manually it is calculated according to equations (2.11) and (2.12), with values given in the start of chapter 5.

Option	Default	Description
--seed, -s	42	Seed for random number generator (RNG)
--endTime, -e	1	End time for the KMC simulation (s)
--ratecalc, -r	liang	Rate calculator [CE, liang]
--jsonFname, -j	None	Relative path to json file from fit_barriers.py
--poscarInputLattice, -p	None	Relative path to POSCAR file
--temperature, -t	293.15 K	Temperature of system (K)
--Cv, -c	None	Set vacancy concentration manually
--MgAtPerc	0.67 at.%	Set at.% of Mg for random start config
--SiAtPerc	0.77 at.%	Set at.% of Si for random start config
--Ndist	4.45 Å	Cutoff distance for neighbourlist
--NNdist	3.0 Å	Cutoff distance for nearest neighbours

B.2 Observers and logging

Observers are used to log and observe different quantities during the KMC simulation, a flowchart that shows when observers observe function is called is shown in Figure 4.1. The selection of available observers, together with what they observe and the filename pattern for their output files, are listed in Table B.2. All observers are based around the observer struct shown in listing 4.1.

In addition to logging the initial and final state, the observers can also be set to log during the simulation. As mentioned in section 4.1 it is possible to set the logging interval to be fixed, logarithmic or a combination of the two. After each call to the observers `observe()` function `next_obs` is updated according to $\texttt{next\_obs} = \texttt{factor} \cdot \texttt{next\_obs} + \texttt{fixed\_offset}$. For fixed interval logging `factor` should be set to unity. For logarithmic logging `fixed_offset` should usually be set to 1 instead of 0 to ensure that even when $\texttt{factor} \cdot \texttt{next\_obs}$ is less than 1 the next observation will still be increased by at least one.

To implement a new observer, the first step is to find out what to observe and what kind of data that is needed to do the logging. The observe function have two parameters,

the observer struct, which is needed to access the observers information, and the KMC struct, which contain all data for the current state of the simulation, but no history except for information about the last jump. Any additional information that the observer needs to keep track of should be made into a struct that can be saved in the data pointer of the observer struct. The next step is to implement (i) a `free()` function, that frees the allocated data and the observer struct, (ii) the `observe()` function that performs the logging and (iii) a function for allocating and initialising the observer. For the observer to be visible to Python a binding function should be added to the SWIG file.

B.3 Setting up cluster expansion for KMC

The goal of this section is to go through the details of the workflow for going from a set of output files from DFT calculations to a finished fit that can be used as an input file for KMC simulations. For a graphical illustration of the workflow see Figure 3.3.

The only information that is needed from the DFT calculations are the activation energies, given in eV. These energies must be paired with the initial unrelaxed lattice for the jump as well as the index for site i and site j. The reason for having the unrelaxed lattice is that the KMC simulations are performed on a fixed FCC lattice. Furthermore, it is also necessary for setting up the site groups consistently for all the structures in the training set. All of this information can be stored in a slightly modified POSCAR file. The first line in a POSCAR file is a comment line, this line is used to store the activation energy, and has the form "DEnergy=x.xxxx". A regular POSCAR file does not store any vacancies, but since it is needed for setting up the site groups it is added to the POSCAR and marked with atom type "X". The last convention is that the first lattice site in the POSCAR is always the vacancy, $j = 0$, and the atom performing the jump is always the next lattice site, $i = 1$.

The next step is to create the training set consisting of $\boldsymbol{\mathcal{X}}$ and $\vec{y}$, as introduced in section 3.4, which is the input for the fitting procedure. The activation energies are stored in the vector $\vec{y}$ and each row in matrix $\boldsymbol{\mathcal{X}}$ contains the rightmost sum in equation (3.4) for all the ECIs in $\vec{V}$. The training set will be stored in a text file where the first section of the file contains information about all the different site groups and the rest contains one line per POSCAR. The first element of each line is the filename, the next elements are the sums for all the different site groups and combinations of X_i and $\vec{q}$, the last element is the activation energy. To create this text file the python script create_sum_file.py is used. The path to the folder containing the special POSCAR files is given as the first argument, the other options are to set a relative path for the output file, -o, print out the poscar and the site group data, -i, and creating a second file where no duplicate sum combinations are included.

It is during the creation of the training set that the combination of site groups is determined by the user, after the training set has been created the setup of site groups for calculations will be done automatically. Before running create_sum_file.py any changes to the site group combinations should be done by editing the function `setup_Eact_data_manual()` and the arrays used by this function. There are functions for creating all pair site groups with a given pair distance or creating all single site groups within a certain distance. It is however recommended to explicitly setup the site groups since the small DFT simulation cells can give some pairs that are only there because of the cell size and the periodic boundary conditions. If these are included in the training set the automatic setup of `Eact_data`

will fail since these site groups will be empty in the larger KMC simulation lattice. To get the correct site groups when NNN of site i and j are included, the cell size must be at least $4 \times 4 \times 4$ unit cells. A script to pad $3 \times 3 \times 3$ cells into $4 \times 4 \times 4$ is located in the scripts folder in the gitlab repository.

The text file created by create_sum_file.py is used as input to the fitting routine and the path to this file is the first input parameter for fit_barriers.py. The option --strategy is used to set the strategy for the hyperparameter optimization and can be set to grid or random, with random as the default value. If the random strategy is used --num_attempts can be used to set the number of attempts. For prior distributions with few combinations of hyperparameter values, it is wise to use the grid strategy since this will test all possible combinations once. When the number of combinations is large it can be useful to first use a random search and then do a grid search with values close to the ones found by the initial random search. The number of partitions used to calculate the cross-validation score can be set by using the option --partitions. The last option is --out and sets the path for the output file. This fitting creates a json file which contains all the information from the input file, the coefficients and the inverse variance used for the best fit.

The calculation of the $Q_{\alpha\alpha}$ term for each ECI, also referred to as inverse variance in the program, is implemented as classes in the file inv_var.py. The choice of inverse variance calculator, which is effectively the choice of prior distribution, is done by altering the corresponding lines of code in fit_barriers.py. Since the inverse variance calculators are implemented as classes it is easy to implement a new prior by making a new class for it.

Table B.2: Available observers, can be added by calling kmc.add_obs_"name", the output file names are on the form "basename""called"_"output extension", where "basename" is from the KMC struct and called is the number of times the observer has been called. The energy, append_statfile and CD_rates observers only use one output file and appends a new line each time the observe function is called.

Name	Function	Output extension
print_info	Prints current iteration, KMC time and CPU time to terminal	None
energy	Logs the total energy of the system relative to the start configuration.	tot_energy.txt
CD_rates	Logs rates for evaporation and condensation of single solutes. For each event start and end size for the cluster, cluster composition and time between events are logged.	.rates
write_XYZ	Writes lattice to XYZ format.	.xyz
write_POSCAR	Writes lattice to POSCAR format.	.POSCAR
write_POSCAR_noAl	Writes lattice to POSCAR format, Al is excluded.	.POSCAR-noAl
write_cube	Writes lattice to Gaussian cube format	.cube
write_KMC	Writes the KMC struct to file	.kmc
append_statfile	Logs KMC time, iterations, CPU time, number of clusters, average-, minimum- and maximum cluster size, percentage of solutes that are part of dimer or larger clusters and diffusivities	.stat
write_XYZ_clusters	Writes all solutes participating in clusters of size$>=2$ to XYZ format	clusters.xyz
write_cluster_hist	Writes histogram of cluster sizes to file	.hist
write_cluster_pos	Writes position of all solutes participating in clusters of size$>=2$ to file	.pos
write_neighbours	Write all sites in the neighbourlist of specified site to file first time its observe function is called. Option to use different atomic element for each site to visualise the sorting	Neighbours_site"site index".xyz

B.4 Table of variables for main equation of CE for KMC

Table B.3: Interpretation and implementation of all the variables in equation (3.4).

Variable	Interpretation	Implementation
E_{ij}^{act}	Activation energy needed for atom at site i to jump to site j	Return value of function `get_Eact_Eact_data`
X_i	Atom type at site i	Stored as first element of each row in `atomcombs`
V^{X_i}	ECI for a jump in pure aluminium	Stored as coefficients in a `site_group` with `order=0`
α	Contains a set of symmetrically equivalent clusters	The properties for a cluster to be part of α is stored in the `descriptor` struct of the `site_group`
$S_{\alpha ij}$	Contains the lattice site indices to all the clusters in α for a jump from site i to j	Indices to their position in neighbourlist of site i is stored in `index_array` [1]
$\vec{k}$	Vector of lattice site indices for a cluster from $S_{\alpha ij}$	See implementation field of $S_{\alpha ij}$
$\vec{q}$	Decoration vector, length equal to number of sites in cluster, each element is [Mg, Si]. Together with equation (3.5) it defines a cluster basis function	For each coefficient the corresponding decoration vector is stored in `atomcombs` [2]
$n_{\vec{q}}(X_{\vec{k}})$	Cluster basis function for cluster consisting of lattice sites $\vec{k}$ and decoration vector $\vec{q}$	Calculated by `caclulate_sum`, where the sum is over all $\vec{k} \in S_{\alpha ij}$ and any equivalent $\vec{q}$

[1] To get lattice site index for element l of the `index_array` use `si->neighbours[orientation_matrix[k][index_array[l]]]` where k is the position of site j in the neighbourlist of site i.

[2] The first element of `atomcombs` is reserved for X_i. If several decoration vectors are equivalent they will be automatically added to the row of `atomcombs` and the number of vectors in the row are stored in `n_equivalent`.

B.5 Documentation of Python interface for KMC

Contents

Module **skmc.skmc**

Python bindings to skmc.

Classes

Class **Basis**

```
class Basis(sitetag: int, symbol: char const *, sx: double, sy: double,
sz: double)
```

Proxy of C _Basis struct.

Represents a basis site in the unit cell - only used for creating the lattice.

Instance variables

Variable **sitetag** Application specific tag for this site

Variable **symbol** Symbol of element occupying this site (use "X" for vacancy)

Variable **sx** Scaled x-coordinate for this site

Variable **sy** Scaled y-coordinate for this site

Variable **sz** Scaled z-coordinate for this site

Variable **thisown** The membership flag

Class **Eact_data**

```
class Eact_data()
```

Proxy of C _Eact_data struct.

Struct for storing all data needed for calculation of activation energies using cluster expansion method.

Instance variables

Variable **site_groups** Array containing all site_groups for the potential.

Variable **n_groups** Number of site_groups.

Variable **size** Allocated size of site_groups.

Variable **n_elem** Number of different elements in the lattice, excluding the vacancy.

Variable **jump_freq** Array of length n_elem that contains the jump attempt frequency for the different elements.
Same order as lattice->Z.

Variable **orientation_matrix** An array with size n_possible_swaps X n_neighbours.
The first dimension is the number of swap possibilities for an atom, the second dimension is the number of elements in the neighbour list of the atom.
The array contains indices to elements in the neighbour list.
If swapping an atom to site number j in the original sites neighbourlist, row j of this array should be used when accessing elements.
neighbours[orientation_matrix[j][other_neighbour_index]].

Variable **thisown** The membership flag

Methods

Method **add_const_site_group**

```
def add_const_site_group(self) -> 'void'
```

Adds site group for constants in Eact_data.

Method **add_pair_site_group**

```
def add_pair_site_group(self, lattice: Lattice, dist1: double, dist2: double,
pr_dist: double, offset1: double, offset2: double, tol: double) -> 'int'
```

Set up a pair site group for the given parameters.

Method **add_pair_site_group_irow**

```
def add_pair_site_group_irow(self, lattice: Lattice, dist1: double,
dist2: double, pr_dist: double, offset1: double, offset2: double, tol: double,
i: int, row: int) -> 'int'
```

Set up a pair site group for the given parameters using specified i and row during setup.

Method **add_pair_site_groups**

```
def add_pair_site_groups(self, lattice: Lattice, pr_dist: double, tol: double)
-> 'void'
```

Will add all possible pairs with given pr_dist and tol from neighbouring sites that are part of a 'single' site group.

Method **add_pair_site_groups_irow**

```
def add_pair_site_groups_irow(self, lattice: Lattice, pr_dist: double,
tol: double, i: int, row: int) -> 'void'
```

Will add all possible pairs with given pr_dist and tol from neighbouring sites that are part of a 'single' site group. Uses specified i and row during setup.

Method **add_single_site_group**

```
def add_single_site_group(self, lattice: Lattice, distance: double,
offset: double, tol: double) -> 'int'
```

Set up a single site group for the given parameters.

Method **add_single_site_group_irow**

```
def add_single_site_group_irow(self, lattice: Lattice, distance: double,
offset: double, tol: double, i: int, row: int) -> 'int'
```

Set up a single site group for the given parameters using specified i and row during setup.

Method **add_single_site_groups**

```
def add_single_site_groups(self, lattice: Lattice, maxdist: double,
tol: double) -> 'void'
```

Set up site groups based on single_descriptors.All neighbours with distance <= maxdist, will be included in a site group.

Method **add_single_site_groups_irow**

```
def add_single_site_groups_irow(self, lattice: Lattice, maxdist: double,
tol: double, i: int, row: int) -> 'void'
```

Set up site groups based on single_descriptors.All neighbours with distance <= maxdist, will be included in a site group. Uses specified i and row during setup.

Method **get_header_for_sums**

```
def get_header_for_sums(self) -> 'char *'
```

Returns header for sums calculated using calculate_sums_from_lattice.

Method **get_jump_freq**

```
def get_jump_freq(self) -> 'obj_t *'
```

Returns a copy of array of jump attempt frequency for the different elements.

Method **get_n_groups**

```
def get_n_groups(self) -> int
```

Returns n_groups from Eact_data.

Method **get_orientation_matrix_row**

```
def get_orientation_matrix_row(self, row: int) -> 'obj_t *'
```

Returns specified row of the orientation matrix.

Method **get_poscar_fname**

```
def get_poscar_fname(self, lattice: Lattice, j: int, i: int) -> 'char *'
```

Returns poscar filename for the jump from i to j based on the single site groups in data.

Method **get_site_group**

```
def get_site_group(self, i: int) -> 'struct _site_group *'
```

Returns site group number i.

Method **resize_to_n_groups**

```
def resize_to_n_groups(self) -> 'void'
```

Resizes site_groups to the current number of groups.

Method **set_jump_freqs**

```
def set_jump_freqs(self, jump_freqs: obj_t *) -> 'void'
```

Sets the jump frequencies for the Eact_data. jump_freqs should be a numpy array of type double and length of n_elem, sorted same way as the n_elem first elements of Z in lattice.

Method **show**

```
def show(self) -> 'void'
```

Prints Eact data to terminal.

Method **write_site_groups_to_xyz**

```
def write_site_groups_to_xyz(self, lattice: Lattice, j: int, i: int,
filename: char const *) -> 'void'
```

Writes site groups in Eact_data to a xyz file with different element for each site group.

Class `KMC`

```
class KMC(lattice: Lattice, model: Model, end_time: double, j: int,
rate_function: char *)
```

Proxy of C _KMC struct.

Struct describing the complete KMC model.

Instance variables

Variable **`lattice`** Pointer to lattice structure.
This structure will not be changed.
Hence, the elements in lattice-sites corresponds to the initial configuration.

Variable **`time`** Current simulation time (s).

Variable **`end_time`** End time for the simulation (s).

Variable **`jumps`** Current number of KMC-steps performed.

Variable **`model`** Model parameters provided by the user.

Variable **`solutes`** Array of element numbers for solute elements.
Length: model.nsolutes

Variable **`perm`** Array with element permutations.
The current element at site `i` is given by `lattice->sites[perm[i]].elem`.

Variable **`iperm`** The inverse of `perm`.
Atoms are ordered according to their original positions in `lattice`.
A pointer to the current site of atom `i` can be obtained with `lattice->sites + iperm[i]`.

Variable **`dd`** Array of diffusion data for each atom.
The order of the elements in this array is fixed and corresponds to the initial configuration in lattice->sites.
Length: lattice->n

Variable **`diffusion_time`** Time at which we started to collect diffusion data (i.e. time since last call to resetDiffusion()) (s).

Variable **diffusion_jump** Same as diffusion_time but stores iteration number instead of time.

Variable **timescale** Time scale (1/s).

Variable **nnmax** Max number of nearest neighbors of any site.

Variable **R** Cumulative sum of rates; length: nnmax+1

Variable **observers** List of observers.

Variable **observers_len** Allocated length of 'observers'.

Variable **get_rate** Function pointer to the rate finder for the KMC.
Returns the jump rate for a jump (of the vacancy) from site `j` to `i`.

Variable **get_energy** Function pointer to the energy calculator for the KMC. Returns the activation energy for a jump (of the vacancy) from site `j` to `i`.

Variable **get_rate_data** Void pointer to all extra data needed by the get_rate function.

Variable **j** Index to site with vacancy.

Variable **old_j** Index to where vacancy was before most recent kmcStep.

Variable **thisown** The membership flag

Methods

Method **add_obs_CD_rates**

```
def add_obs_CD_rates(self) -> 'void'
```

Add CD_rates observer to KMC.

Method **add_obs_append_statfile**

```
def add_obs_append_statfile(self, fixed_offset: uint64_t, factor: double,
use_time: int, cl_data: struct _cluster_data *) -> 'void'
```

Add append_statfile observer to KMC. Appends a line of statistics in the statfile each time observe is called.

Method **add_obs_print_info**

```
def add_obs_print_info(self, fixed_offset: uint64_t, factor: double,
use_time: int) -> 'void'
```

Add print_info observer to KMC.

Method **add_obs_write_KMC**

```
def add_obs_write_KMC(self, fixed_offset: uint64_t, factor: double,
use_time: int) -> 'void'
```

Add write_KMC observer to KMC, writes KMC struct to file.

Method **add_obs_write_POSCAR**

```
def add_obs_write_POSCAR(self, fixed_offset: uint64_t, factor: double,
use_time: int) -> 'void'
```

Add write_POSCAR observer to KMC, writes lattice to file using the POSCAR format of VASP.

Method **add_obs_write_POSCAR_noAl**

```
def add_obs_write_POSCAR_noAl(self, fixed_offset: uint64_t, factor: double,
use_time: int) -> 'void'
```

Add write_POSCAR observer to KMC, writes lattice to file using the POSCAR format of VASP. Excludes all Al atoms

Method **add_obs_write_XYZ**

```
def add_obs_write_XYZ(self, fixed_offset: uint64_t, factor: double,
use_time: int) -> 'void'
```

Add write_XYZ observer to KMC, writes lattice to file using extended XYZ format.

Method **add_obs_write_XYZ_clusters**

```
def add_obs_write_XYZ_clusters(self, fixed_offset: uint64_t, factor: double,
use_time: int, cl_data: struct _cluster_data *) -> 'void'
```

Add write_XYZ_clusters observer to KMC. Writes lattice to file using extended XYZ format, but only atoms in clusters of size>=2 are included.

Method **add_obs_write_cluster_hist**

```
def add_obs_write_cluster_hist(self, fixed_offset: uint64_t, factor: double,
use_time: int, cl_data: struct _cluster_data *) -> 'void'
```

Add write_cluster_hist observer to KMC. Calculates histogram of cluster sizes and writes it to .hist file.

Method **add_obs_write_cluster_pos**

```
def add_obs_write_cluster_pos(self, fixed_offset: uint64_t, factor: double,
use_time: int, cl_data: struct _cluster_data *) -> 'void'
```

Add write_cluster_pos observer to KMC. Writes position of all cluster atoms to a .pos file.

Method **add_obs_write_cube**

```
def add_obs_write_cube(self, fixed_offset: uint64_t, factor: double,
use_time: int) -> 'void'
```

Add write_cube observer to KMC, writes lattice to file using Gaussian Cube format.

Method **add_obs_write_neighbours**

```
def add_obs_write_neighbours(self, i: int, differentElement: int) -> 'void'
```

Add write_neighbours observer to KMC, writes neighbours of site i at initial configuration to file. If differentElement is true the function will give each site a different element such that it is possible to show the sorting of the neighbours in for example ovito.

Method **add_obs_energy**

```
def add_obs_energy(self, fixed_offset: uint64_t, factor: double, use_time: int)
-> 'void'
```

Add energy observer to kmc, logs the change in total energy for each jump and writes the energy to file with set logging scheme.

Method **make_cl_data**

```
def make_cl_data(self) -> 'struct _cluster_data *'
```

Allocate and initialize cluster_data struct for given KMC

Method **run**

```
def run(self) -> 'void'
```

Run KMC until end_time is reached.

Method **set_get_rate_data**

```
def set_get_rate_data(self, data: void *) -> 'void'
```

Set the get_rate_data pointer for the KMC struct.

Class **Lattice**

```
class Lattice(nx: int, ny: int, nz: int, a: double, symbol: char const *,
maxdistMultiplier: double = 0.9)
```

Proxy of C _Lattice struct.

Represents a periodic lattice supercell of size nx * ny * nz. The default initiator creates a fcc lattice with lattice parameter a of element symbol. maxdistMultiplier is multiplied with a to set the cutoff radius for neighbour lists.

Instance variables

Variable **cell** Three vectors, cell[0], cell[1], cell[2], defining the simulation cell

Variable **volume** Volume of the simulation cell (Åˆ3)

Variable **orthorhombic** Whether the lattice is orthorhombic (cell vectors are perpendicular and along the Cartesian axis)

Variable **n** Number of sites

Variable **sites** Array of sites; Length: n

Variable **nele** Number of different atomic elements in the system

Variable **Z** Array of atomic numbers of the different elements in the system sorted in descending order; Length: nele

Variable **thisown** The membership flag

Methods

Method **addRandomSolute**

```
def addRandomSolute(self, n: int, symbol: char const *, sitetag: int = 0,
put_symbol: char const * = None) -> 'void'
```

Adds n solute atoms of thpe symbol to the lattice, replacing existing atoms. If sitetag is equal or greater than zero, solutes will only be added to sites with this sitetag. If elem is equal to or greater than zero, solutes will only be added to sites with this atom.

Method **addSolute**

```
def addSolute(self, i: int, symbol: char const *) -> 'void'
```

Adds solute atom symbol to site i.

Method **add_site**

```
def add_site(self, s: Site, symbol: char const *) -> 'void'
```

Adds a site to the lattice and sets elem according to specified atomic symbol, disregards elem field of the given site.

Method **calculate_energy_Eact_data**

```
def calculate_energy_Eact_data(self, data: Eact_data, j: int, i: int) ->
'double'
```

Calculate energy from current lattice configuration instead of using the perm array. This function will only be used for calculating energy for POSCAR files.

Method **calculate_energy_liang**

```
def calculate_energy_liang(self, j: int, i: int) -> 'double'
```

Returns the activation energy for a jump from site i to j.Calculated from the initial configuration of the lattice, not used for running kmc, only for calculating energy of jumps from poscar files.

Method **calculate_sums_from_lattice**

```
def calculate_sums_from_lattice(self, data: Eact_data, j: int, i: int) ->
'obj_t *'
```

Calculate sums from current lattice configuration instead of using the perm array. This function will only be used for calculating sums for precalculated energies.

Method **calculate_vector_from_iToj**

```
def calculate_vector_from_iToj(self, i: int, j: int) -> 'obj_t *'
```

Return the vector from lattice site i to lattice site j.

Method **copy**

```
def copy(self) -> 'struct _Lattice *'
```

Returns a deep copy of the lattice.

Method **defineNN**

```
def defineNN(self, maxdistNN: double, maxdist: double) -> 'void'
```

Defines nearest neighbours in the lattice, include all sites with `distance <maxdist`. Resulting neighbourlist is not sorted.

Method **elementNumber**

```
def elementNumber(self, symbol: char const *, add: int) -> 'int'
```

Returns element number of symbol, or -1 if add is zero and symbol is not already in lattice->Z. If add is non-zero and symbol is not already in lattice->Z, it is added to the lattice->Z.

Method **getMaxN**

```
def getMaxN(self) -> int
```

Returns the maximum number of neighbours of any site in the lattice.

Method **getMaxNN**

```
def getMaxNN(self) -> int
```

Returns the maximum number of nearest neighbours of any site in the lattice.

Method **get_Z**

```
def get_Z(self) -> 'obj_t *'
```

Returns a copy of array of atomic numbers of all elements in the system (including vacancy).

Method **get_cell**

```
def get_cell(self) -> 'obj_t *'
```

Returns a copy of the supercell vectors (cell vectors along rows).

Method **get_elements**

```
def get_elements(self) -> 'obj_t *'
```

Returns a copy of elem for each site.

Method **get_n**

```
def get_n(self) -> int
```

Returns number of sites in lattice.

Method **get_positions**

```
def get_positions(self) -> 'obj_t *'
```

Returns an Nx3 array with position of each site.

Method **get_site**

```
def get_site(self, i: int) -> 'struct _Site *'
```

Returns the site instance at index i.

Method **get_sitetags**

```
def get_sitetags(self) -> 'obj_t *'
```

Returns a copy of the tags for each site.

Method **make_empty_Eact_data**

```
def make_empty_Eact_data(self) -> 'struct _Eact_data *'
```

Creates and returns an empty Eact_data instance for lattice.

Method **sortAllNeighbourListsFCC**

```
def sortAllNeighbourListsFCC(self) -> 'void'
```

Sorts the neighbour list for all sites in the lattice, sorted by distance, relx, rely and relz, smallest first. This ensures that nearest neighbours are first, second nearest are next and so on.

Class **Model**

```
class Model(Cv: double, T: double, basename: char *)
```

Proxy of C _Model struct.

Struct filled in by user to describe the model.

Instance variables

Variable **Cv** Vacancy consentration (at. fraction). This may be changed during simulation, e.g. in response to temperature changes.

Variable **T** Temperature of the system in Kelvin.

Variable **nsolutes** Number of solute elements.

Variable **solute_names** Array of chemical symbols of solute elements; Length: nsolutes

Variable **basename** Base name of for output files.

Variable **thisown** The membership flag

Methods

Method **addSoluteName**

```
def addSoluteName(self, symbol: char *) -> 'int'
```

Add symbol to solute_names. Returns -1 if symbol is already in solute_names.

Method **show**

```
def show(self) -> 'void'
```

Prints model struct to terminal.

Class **Site**

```
class Site(elem: int, x: double, y: double, z: double, sitetag: int = 0,
n_nn: int = 0, n: int = 0, neighbours: int * = None)
```

Proxy of C _Site struct.

Represents a lattice site.

Instance variables

Variable **elem** Element index (lattice->Z[elem] gives atomic number)

Variable **sitetag** Application specific tag for this site

Variable **x** x-coordinate of site (Å)

Variable **y** y-coordinate of site (Å)

Variable **z** z-coordinate of site (Å)

Variable **n_nn** Number of nearest neighbours

Variable **n** Number of neighbours

Variable **neighbours** Array of neighbour indices; Length: n. First n_nn elements are nearest neighbours.

Variable **wrap** Array of wrap flags; Length: n

Variable **dists** Array of nearest neighbour distances (Å); Length: n

Variable **thisown** The membership flag

Methods

Method **get_dists**

```
def get_dists(self) -> 'obj_t *'
```

Returns a copy of the array of the distances to the neighbours.

Method **get_n**

```
def get_n(self) -> int
```

Returns number of neighbours.

Method **get_neighbours**

```
def get_neighbours(self) -> 'obj_t *'
```

Returns a copy of the neighbour indices.

Method **get_positions**

```
def get_positions(self) -> 'obj_t *'
```

Returns a copy of the position of this site in absolute coordinates.

Method **get_wrapflags**

```
def get_wrapflags(self) -> 'obj_t *'
```

Returns a copy of wrapflags array.

Class **descriptor**

```
class descriptor()
```

Proxy of C descriptor struct.

Instance variables

Variable **type** String describing the type of descriptor

Variable **free** Free all data for descriptor and the descriptor itself.

Variable **n_neighbours** Number of neighbours that need to be in the neighbours list sent to the check function.

Variable **check** Check if a neighbour fulfills the requirements of the descriptor

Variable **get_str** Function to get string with info about descriptor.

Variable **data** Pointer to all data needed by descriptor.

Variable **thisown** The membership flag

Class **pair_descriptor_data**

```
class pair_descriptor_data()
```

Proxy of C pair_descriptor_data struct.

Instance variables

Variable **dist1** Distance to site 1, measured from initial site of atom.

Variable **dist2** Distance to site 2, measured from initial site of atom.

Variable **offset1** The offset for site 1, measured along swap direction, with origin in the initial site of atom.

Variable **offset2** The offset for site 2, measured along swap direction, with origin in the initial site of atom.

Variable **pr_dist** Distance between the two sites in the pair.

Variable **tol** Tolerance used for checking.

Variable **thisown** The membership flag

Class **single_descriptor_data**

```
class single_descriptor_data()
```

Proxy of C single_descriptor_data struct.

init(self) -> single_descriptor_data

Instance variables

Variable **distance** Distance from the atom that is being swapped.

Variable **offset** The offset is measured along swap direction, with origin in the initial site of atom.

Variable **tol** Tolerance used for checking.

Variable **thisown** The membership flag

Class **site_group**

```
class site_group()
```

Proxy of C _site_group struct.

Stores all symmetrically equivalent clusters that fulfill the descriptor of the site group. Also stores all coefficients, with their respective atomic combinations, for this collection of clusters.

Instance variables

Variable **index_array** Stores an array of indices for all clusters included in this group. The indices are the position of the lattice sites in the neighbour list of site i.
The list is stored as a 1D array, so if the order is for example 2, then element 0 and 1 is the first cluster, 2 and 3 is the second cluster and so on.

Variable **n_sites** Number of elements in index_array.

Variable **size** Number of elements allocated for index_array.

Variable **order** Order of the term, the number of lattice sites in each cluster.

Variable **descriptor** Pointer to a descriptor that describes the conditions to be a part of this site_group.

Variable **n_coeffs** Number of stored coefficients for this site.

Variable **coeffs** Stores coefficients.

Variable **atomcombs** Stores atomic numbers for use with coefficients.
It is a 2D arrray with n_coeffs rows and varying number of columns, number of columns for row k is 1+n_equivalent[k]*order.
The first column is the atom type that jumps, the remaining are all equivalent atom combinations for the neighbours.

Variable **n_equivalent** Array that stores number of equivalent combinations for each row of atomcombs.

Variable **thisown** The membership flag

Methods

Method **add_coefficient**

```
def add_coefficient(self, coeff: double, atomcomb: obj_t *) -> 'int'
```

Adds a coefficient to a site group.atomcomb is a 1D array that wil be added as a column in atomcombs.If arr_size does not match order+1 it will abort and return -1.If the given atomcomb is already present it will change the value of coeff and return 2.If atomcomb does not already exist, both coefficient and atomcomb is added and return 1.

Method **get_index_array**

```
def get_index_array(self) -> 'obj_t *'
```

Returns a copy of the index array.

Method **get_n_sites**

```
def get_n_sites(self) -> int
```

Returns n_sites for a site_goup.

Method **get_order**

```
def get_order(self) -> int
```

Returns order of site_group.

Method **get_str**

```
def get_str(self) -> 'char *'
```

Returns string with info about site_goup.

Method **get_str_coeffs**

```
def get_str_coeffs(self) -> 'char *'
```

Returns string with info about the coefficients for a site_goup.

Method **print_coeffs**

```
def print_coeffs(self) -> 'void'
```

Prints site group coefficients to terminal.

Method **show**

```
def show(self) -> 'void'
```

Prints site group to terminal.

Øystein Tormodsen Nygård

Kinetic Monte Carlo simulations of precipitation in Al-Mg-Si alloys

NTNU
Norwegian University of
Science and Technology