Simulation of Alloy Phase Transition by 2D Monte Carlo Method and Ising Mode with Python

Fengyi Li
Department of Materials
Royal School of Mines, Imperial College London
London, United Kingdom
fengyi.li18@imperial.ac.uk

***Abstract*—The simulation aims to investigate the effect of the temperature, alloy concentration and interaction energy on the system configuration, average order parameter, phase transition temperature and heat capacity of a given binary alloy. Metropolis Monte Carlo method and modified Ising model are implemented to achieve the purpose. Final configuration of the system becomes more dispersed as the temperature increases. Positive interaction energy gathers same type atoms together, and negative one separates them for decreasing the total energy. Phase transition temperature increases with the alloy fraction only for the system with negative interaction energy, and the phase transition temperature reaches the critical temperature when the alloy fraction is 0.5 of the system.**

***Index Terms*—Monte Carlo Method, Ising model, periodic boundary condition, binary alloy, phase transition temperature**

I. Introduction

The symmetry of the crystalline structure brings much simplification for us to discover the physical properties of materials. X-ray diffraction provides the evidence that in the *ordered* alloy scattered waves show one type of atoms are exactly in phase with each other and out of phase with the other type one [1]. However, in practical applications, most materials used are *disordered* or weakly ordered because of the existence of the defects, vacancies, dislocations and stacking faults. For example, the doped semiconductor shows an increase of disorder after possessing a higher number density of the charge carriers. In polycrystalline materials such as metals and alloys, grain boundaries cause much more disordered configuration. The disorder can also be produced by the rise of the temperature, which stimulates the vibration of the atoms in the lattice, thus the vibrational disorder being strengthened based on the Eq.(1),

$$S = k_B W \tag{1}$$

where S is the configurational entropy of the system, k_B is the Boltzmann constant, and W is the number of microscopic state with respect to the macroscopic thermodynamic state. In this simulation, the ideal alloy possesses mainly chemical or compositional disorder.

For an alloy, it contains a considerable amount of atoms and defects that can be analytically classified into discrete numbers due to their highly consistent properties for each type. This provides us with a great chance to predict the results or outcomes of the real alloy system based on the mathematical models such as the Monte Carlo method and Ising model. The computer simulation is faster and more accurate than the traditional pencil-and-paper work, and it is convenient to be used with small programs. Alloy phases and then mechanical properties can be predicted by the computer simulations before the practical experiments are carried out, which reduces time and money expense in reality. However, the computational cost should also be considered as an essential factor during the simulation process.

In contemporary metallurgy and manufacture industry, many metals are strengthened by distributing small second-phase particles in a ductile matrix, which is called *precipitation hardening* or *age hardening* [3]. The phase of the metallic solid solution is separated after quenched into the miscibility gap of the phase diagram [2], and this phase separation is predictable with the Monte Carlo method and Ising model. The Monte Carlo method is a relatively sophisticated method to simulate the binary alloy phase transition. It will not give identical results but yields values in the 'statistical error'. Monte Carlo method can calculate the thermal averages of many-particle systems in equilibrium statistical mechanics and give highly accurate results even on some complicated and realistic interaction models [4]. Since the accuracy of the Monte Carlo estimate depends on the completeness with the probed phase, the solution obtained can be as exact as possible with more numerical effort involved [4], [5]. Therefore, the improvement of the accuracy of the Monte Carlo method is possible both in the principle and practice [5]. In this simulation, the accurate results can be obtained by a large number of move steps. Additionally, Ising model is widely applied to simulate the configurational statistics of alloys on a basis using phenomenological atomic interaction parameters [5]. It is one of the most studied and best understood models in computational physics because of its extensive and computational infrastructure [6].

From this alloy simulation, some meaningful phenomenon and principles are expected to be learnt including the final configuration of the system, average order parameter, heat capacity, transition temperature given different combinations of conditions. However, the phase fraction and hence the

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

Fig. 1: A special case of identity 3×3 matrix with host atom as 0 and alloy atom as 1.

composition of each type of atom in phases are not available to be obtained. No effects of defects such as vacancies or interstitial atoms can be achieved in consequence of the perfect rigid grids system.

II. Methods

A. Model and Method

The model constructed in this report treats two types of atoms, *host atom* A and *alloy atom* B, as simple equivalent sites randomly distributed on a rigid $L \times L$ alloy lattice filled with $\{0, 1\}$ in Fig.1. The perfect alloy matrix in the simulation is used with the periodic boundary condition because it is efficient for determining the equilibrium properties with the (semi-)grand canonical ensemble. The implementation avoids degenerate two-phase equilibria and implies the least constraints on the relaxation process [5]. All of this can indicate to apply a useful and simple atomistic *Ising model* [2], and the kinetics are modelled by the direct exchange of atom A and B (Kawasaki dynamics [7]). Therefore, each configuration σ constructed by the two type atoms possesses a total energy or *Hamiltonian*, $H(\sigma)$,

$$H(\sigma) = \frac{1}{2} \sum_{i=0}^{L^2} \sum_{\langle i,j \rangle} J_{ij} \sigma(i) \sigma(j) \tag{2}$$

$$J_{ij} = \begin{cases} 1 & \sigma(i) \neq \sigma(j) \\ 0 & \sigma(i) = \sigma(j) \end{cases} \tag{3}$$

where J_{ij} is the interaction energy between two atoms [8]. Atom i and j are the nearest neighbours and $\sigma(i)$ represents the atom in the configuration σ at a specific position. The coefficient 0.5 is due to the energy doubled. Note that Eq.(2) is simplified to be used in a perfect alloy lattice without any terms related to the defects such as vacancies or interstitial atoms. Also, only the nearest neighbours' interaction is counted in for an accurate enough simulation due to the short range forces [10]. In this simulation, the maximum number of nearest neighbours is four.

By applying the *Metropolis importance sampling Monte Carlo method*, a random initial state is firstly generated in the form of a $L \times L$ matrix, and the initial total energy $H(\sigma)$ is calculated in Eq.(2). A site i is randomly chosen and one of its nearest neighbours is selected randomly for swapping the position by the following rule.

$$W_{n \to m} = \text{Min}\{e^{-\Delta E / k_B T}, 1\} \tag{4}$$

$W_{n \to m}$ is the *transition probability* for a successful change of state from n to m after the swap of two chosen atoms i and j. ΔE is the total energy difference between the m state and the n state. k_B is the Boltzmann constant with the unit eV and T is the temperature [2]. When ΔE is larger than zero, the transition probability is the one with the exponential form. If the transition probability is smaller than a random number between 0 and 1, then the swap will be failed. The simulation runs through the above scheme until the system reaches the equilibrium when the total energy remains nearly constant or oscillates around some fixed value.

Metropolis Monte Carlo method is applied here because it can generate the most likely states and configurations, hence obtaining appropriate thermal averages of interesting observables [5]. During the simulation, energy change of a small local region can be easily calculated to obtain the total energy change. Furthermore, each simulation step can be done completely independent, which ensures the accuracy for the simulation results.

B. Programming

1) Strategies:

The whole program follows a recipe in the Fig.2 to avoid making logic mistakes that hide deeply and cause difficulties for debugging. During the construction of the program, involved functions are finished on individual files for easing the debug sessions.

1) Initial state σ_0 & Initial Hamiltonian $H(\sigma_0)$
2) Random atom x with its four nearest neighbors xN4
3) Random atom y ∈ xN4 and its four nearest neighbors yN4
4) If ΔE < 0:
   → swap
   else if exp(-ΔE / k_BT) < random number r ∈ (0, 1):
   → no change
   else:
   → swap
5) Go to 2) and continue until steps reach a given value N
6) Data and graph generation

Fig. 2: Metropolis Monte Carlo recipe

2) Parameters:

Before the simulation starts, several input parameters should be set in the MAIN file. The dimension of the alloy matrix is $L \times L$ in the form of square. A total number of Monte Carlo moves, or variable *nSweeps*, is set at a large enough value for the system to reach the equilibrium after the system has run sufficient steps, *nEquil*, to have a completely atom contacting between each other. The alloy atom fractions, *f*, temperature range, *T_list*, and interaction energy are chosen to be reasonable and meaningful based on the real physical system. Figure plot should be wisely constructed with a frequency, *step_interval*, ($e.g.$, per 1000K) in order to avoid

a waste of computational efficiency.

3) Expectations: Based on the configuration shown in the form of rigid grids, the effect of temperature, alloy concentration and interaction energy on the system properties can be observed in the simulation. The *average order parameter* is calculated in the Eq.(5)

$$\bar{n} = \frac{\sum_{n=0}^{Z} nN_n}{\sum_{n=0}^{Z} N_n} \tag{5}$$

where n is the number of unlike neighbours, Z is the maximum unlike nearest neighbours and N_n is the number of sites with n unlike neighbours. The *heat capacity* of each atom is calculated in the Eq.(6)

$$C_V = \frac{\langle E^2 \rangle - \langle E \rangle^2}{k_B T^2} \tag{6}$$

where $\langle E^2 \rangle$ is the average square energy and $\langle E \rangle^2$ is the square average energy of each atom at temperature T. Therefore, the relationship between the heat capacity of each atom and the temperature change can be also obtained. The transition temperature can be found from curves made of the average order parameter and the heat capacity of each atom.

III. Results

A. Program

The program contains following components including different functions to complete the simulation. The integrated codes can be checked in the Appendix. The *unit testing* is applied in the following content, which proves the correctness of working for each component.

- SET_UP: An $L \times L$ matrix is generated when the type of the atoms and alloy fraction are provided. Fig.3 shows the schematic configuration of atoms initially generated.

- GET_NEIGHBOURS: Four nearest neighbours' coordinates around a chosen atom are found and returned in an array when provided with the coordinate of that atom. The boundary condition is applied here as mentioned in the Method section. For instance, the four nearest neighbours of the yellow box, atom x marked by the red five-point star in the Fig.3, should have the respective coordinates $\Lambda \in$ {Left: (8,9); Right: (8,1); Bottom: (7,0); Top: (9,0)}, which is exactly shown by GET_NEIGHBOURS component. Example output is shown in the Fig.11 in the Appendix for checking the correctness.

- SWAP_INFO: One of the nearest neighbours of atom x in Fig.3 is randomly chosen ($e.g.$, top one, say atom y), and then its four nearest neighbours, $\Gamma \in$ {Left: (9,9); Right: (9,1); Bottom: (8,0); Top: (0,0)}, are also obtained

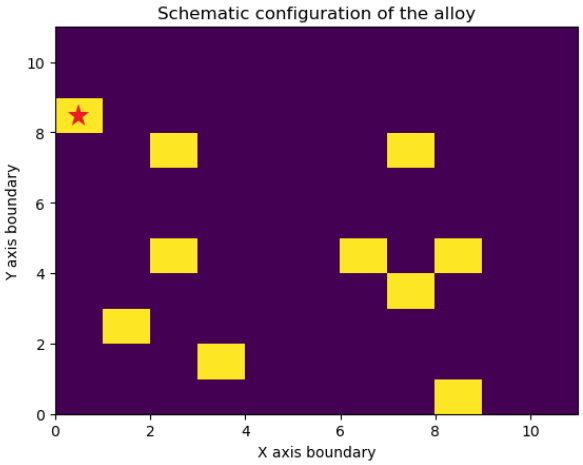


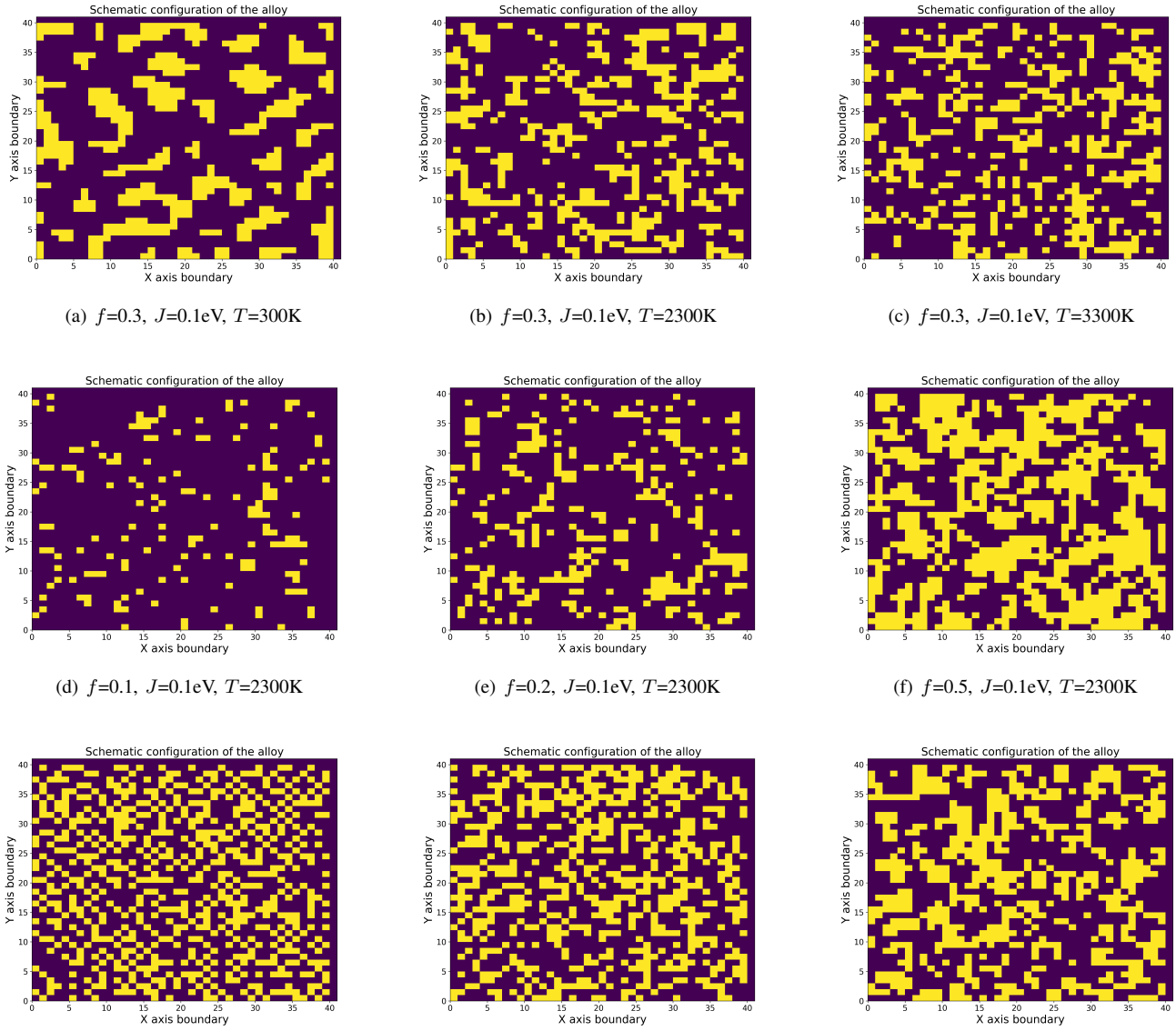
Fig. 3: Schematic configuration of the 10×10 alloy matrix with f as 0.1. Yellow box represents the alloy atom '1' and the purple box is the matrix alloy '0'. No specific meaning is related to the numbers assigned to each type of atom. The extra column at the right side and row at the top are caused by the figure setting, which is negligible with no effect on the simulation.

by using the function GET_NEIGHBOURS. The energy change if swapping two atoms is calculated in

$$\begin{aligned} \Delta E = J \sum_{\Gamma} [\sigma(x)\sigma(\Gamma) - \sigma(y)\sigma(\Gamma)] \\ + J \sum_{\Lambda} [\sigma(y)\sigma(\Lambda) - \sigma(x)\sigma(\Lambda)] \end{aligned} \tag{7}$$

Based on the total energy in Eq.(2) and (3) and transition probability in Eq.(4), the decision of a swap can be determined trivially by a set of *if* statements. For testing, when the J_{ij} is zero, the final configuration is randomly dispersed, which can be seen in the Fig.4(h). As J_{ij} is 0.1 eV, the distribution shows the tendency of nucleation for large grains like in the Fig.4(a), which is reasonable for decreasing the $H(\sigma)$. The example result of the energy change for a swap between atom x and atom y in the Fig.14 is zero as we expect. Therefore, it is a successful swap.

- ALLOY2D: The initial total energy, $H(\sigma)$, of the alloy matrix is calculated by Eq.(2) and (3). Then, SWAP_INFO operates for enough steps to mix the atoms completely with a change of $H(\sigma)$ by ΔE amount. After mixing sufficiently, the alloy matrix starts to run another amount of steps to reach the equilibrium, as figures of $H(\sigma)$, final configurations, order parameter distribution to temperature are plotted. When reaching the equilibrium, the simulation runs for few more steps, *test˙steps*, to obtain the data for processing the average order parameter and the heat capacity for each atom. All related figures will be shown in the Discussion part



(a) f=0.3, J=0.1eV, T=300K (b) f=0.3, J=0.1eV, T=2300K (c) f=0.3, J=0.1eV, T=3300K

(d) f=0.1, J=0.1eV, T=2300K (e) f=0.2, J=0.1eV, T=2300K (f) f=0.5, J=0.1eV, T=2300K

(g) f=0.4, J=-0.1eV, T=2300K (h) f=0.4, J=0.0eV, T=2300K (i) f=0.4, J=0.1eV, T=2300K

Fig. 4: Variation in the structure of the final configuration with temperature in figures a, b, c, with alloy fraction in figures d, e, f, and with interaction energy in figures g, h, i. X axis is along the horizontal direction of the system matrix, and the Y axis is the vertical direction. L=40, $nSweeps$=3000000, $nEquil$=2000000.

below. The correctness can be checked based on the general trends of the configuration in Fig.4 from figures g to i and energy curves in Fig.12 shows the $H(\sigma)$ oscillates around a fixed value after the system mixes well, thus the system approaching to the equilibrium finally.

- ORDERRAMDON: The random order parameter distribution is calculated in the Eq.(8)

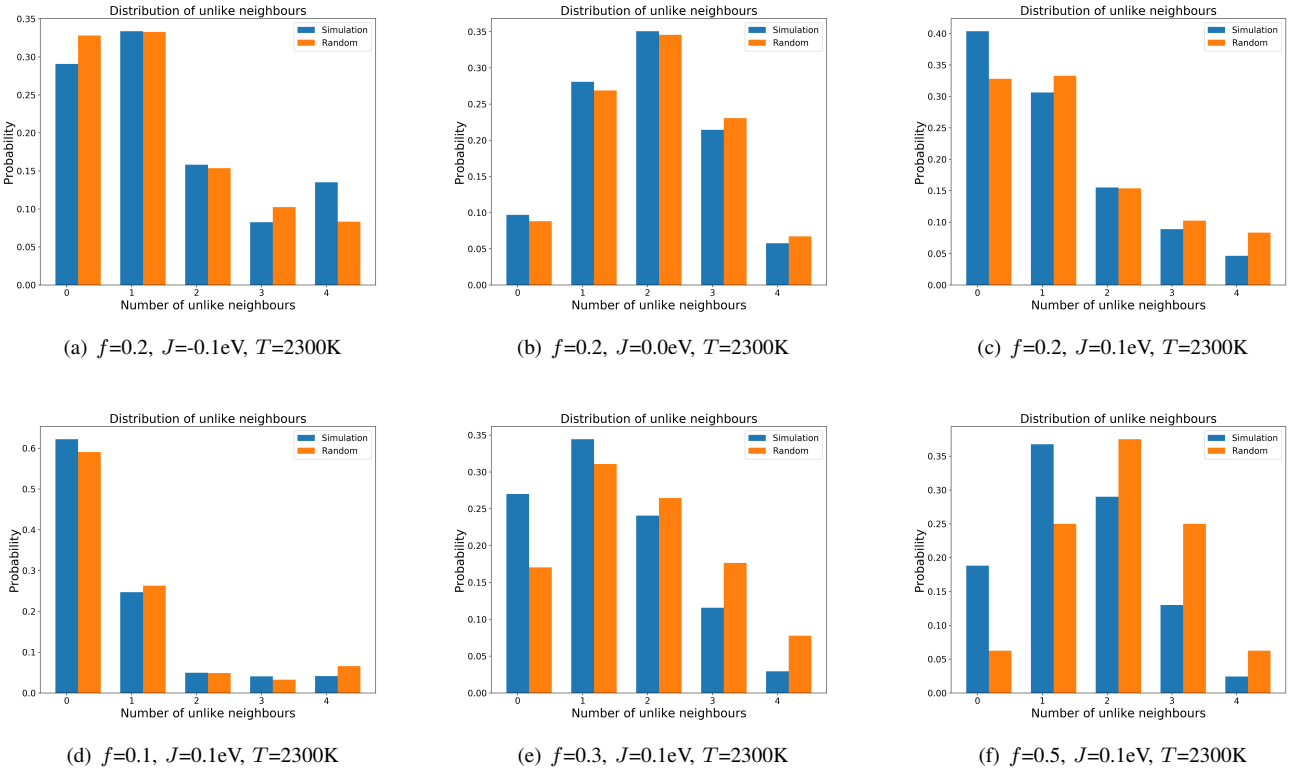
$$P_n = {}^{Z}x_n[ff^{Z-n}(1-f)^n + (1-f)f^n(1-f)^{z-n}] \tag{8}$$

The result of the computer work is shown in the Appendix Fig.13, which is the same as the pencil-and-work result calculated.

- ORDER2D: The number of unlike neighbours will be counted for each step, and assigned to the respective order parameter. Then, the order parameter distribution is calculated in the Eq.(9)

$$P_n = \frac{N_n}{\sum_{n=0}^{Z} N_n} \tag{9}$$

The correctness can be checked from the situation when



(a) f=0.2, J=-0.1eV, T=2300K (b) f=0.2, J=0.0eV, T=2300K (c) f=0.2, J=0.1eV, T=2300K

(d) f=0.1, J=0.1eV, T=2300K (e) f=0.3, J=0.1eV, T=2300K (f) f=0.5, J=0.1eV, T=2300K

Fig. 5: Probability distribution of the order parameters for the alloy matrix as the alloy fraction or the interaction energy changes. Blue bars represent the simulation result and the orange bars represent the randomly distributed result. X axis is the probability for each order parameter and Y axis is the number of unlike neighbours. L=40, $nSweeps$=3000000, $nEquil$=2000000.

J is zero in the Fig.5(e). The probability distribution produced by the simulation nearly matches the one by RANDOM2D, which proves ORDER2D works properly.

The *integration testing* shows the correct result in the Fig.15 where some details of each move step are displayed.

B. General results

The variance of the final configurations under different T with fixed f and J are plotted in the Fig.4. Yellow grids represent the alloy atoms and purple ones stand for the host atoms. Along the first row, figures a, b and c shows the variation of the final configuration when the temperature rises with a fixed combination of f and J. The second row provides the effect of the alloy fraction on the final configuration with the given T and J. The last row displays the relationship between the interaction energy and the final configuration.

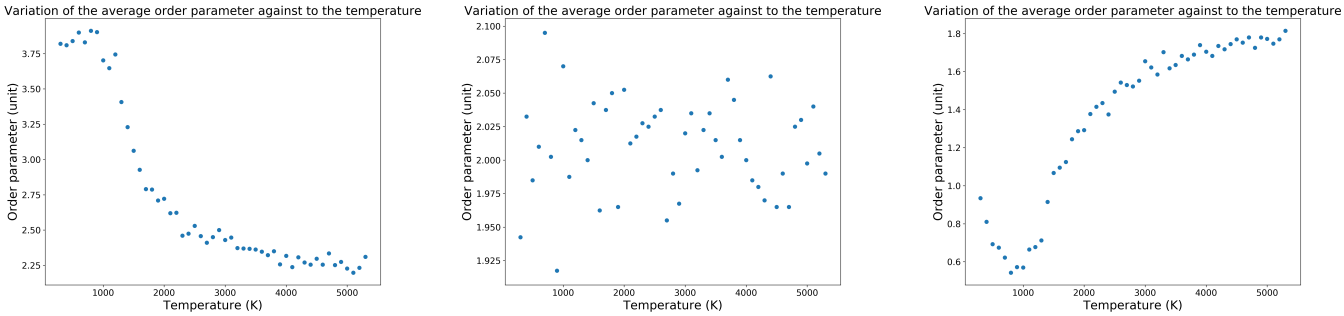
In the Fig.5, the probability distribution for the average order parameter are shown from figure a to c with respect to the change of J, and from figure d to$f$ against the f change. The blue bars represent the simulation results and the orange bars represent the random situation. Figure b is two system with interaction energy equal to 0, which means that the atoms randomly swap their position during each step.

In the Fig.6, the variation of the order parameter against the temperature with given f and J is shown in the figure from a to c. It is straightforward to see that different interaction energy will cause large difference on the pattern of the order parameter with the temperature. Due to tiny and unclear difference with alloy fraction change, the figures here are not shown.

Heat capacity of the atom changes in different ways with the interaction energy, which can be seen in the Fig.7. Figure a and c shows the heat capacity change when alloy atom fraction is different. However, the general trend keeps the same and only value of the heat capacity changes at some levels. The effective interaction energy is defined in the Eq.(10)

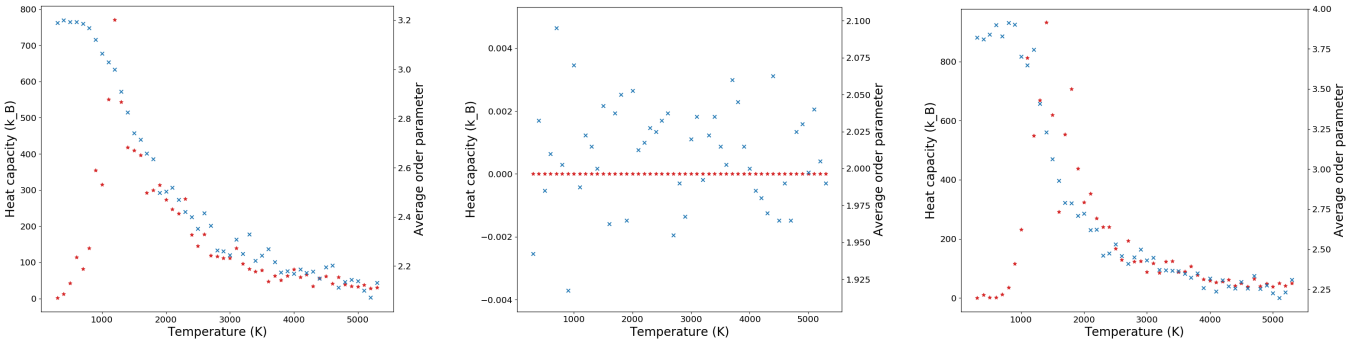
$$U = 2J_{AB} - J_{AA} - J_{BB} \tag{10}$$

where J is the interaction energy between different atoms. If U is positive, it is impossible to have the transition occur, which means that systems with the positive interaction energy cannot find any phase transition temperature [9]. In the Fig.16, the heat capacity curve does not show a clear trend compared with the system with negative J. When U is zero, the system is in homogeneous state, which means that the entropy, S,



(a) f=0.5, J=-0.1eV (b) f=0.5, J=0.0eV (c) f=0.5, J=0.1eV

Fig. 6: Variation of the average order parameter with the temperature provided fixed alloy atom fraction and the interaction energy. X axis represents the temperature and the Y axis represents the average order parameter. L=40, $nSweeps$=3000000 and $nEuil$=2000000.



(a) f=0.4, J=-0.1eV (b) f=0.5, J=0.0eV (c) f=0.5, J=-0.1eV

Fig. 7: Heat capacity variation and the order parameter change with the temperature for the atom in the alloy matrix with given alloy atom fraction and the interaction energy. X axis represents the temperature and Y axis represents the heat capacity for each atom. L=40, $nSweeps$=3000000 and $nEuil$=2000000.

of the system is independent of the temperature by observing from the Fig.7(b) [11]. Therefore, according to the definition of the heat capacity in the Eq.(11),

$$C = T\frac{\partial S}{\partial T} \tag{11}$$

the heat capacity equals to zero, and no transition temperature can be found. In conclusion, only system with negative J has the transition temperature.

The figure with the most concise points and clearest trend line to find the transition temperature is the Fig.7(a) with the estimated transition temperature as 1250K.

IV. Discussion

According to the Fig.4(a), 4(b) and 4(c), as the temperature increases the final configuration presents a more and more dispersed and random uncorrelated distribution with given alloy fraction and interaction energy. Initially, as the temperature is low, the configuration is fully correlated. When the temperature is slightly larger than the transition temperature, the configuration is random but still correlated, showing a *short-range order*. In principle, with a positive interaction energy, atoms with the same type should nucleate to decrease the $H(\sigma)$ in the system because the larger number of unlike bonds will contribute more total interaction energy, which increases the $H(\sigma)$, as we can see it from the Fig.4(a) that isolated alloy atoms are in the small amount. However, when the temperature rises, based on the Eq.(4), the transition probability also increases for a successful swap, thus a state with higher $H(\sigma)$ being more available for the system to accept, resulting in a less patterned configuration. In the Fig.4(c), the configuration shows more and more randomness compared with the Fig.4(b), and the $H(\sigma)$ oscillation in the Fig.12 can also show a numerical trend for the increase of $H(\sigma)$. The larger and more oscillation will be caused as the temperature rises, and this can be explained by the transition probability.

Based on the Fig.4(d) to 4(f), the final configuration becomes clearer to discover a pattern when the alloy fraction

increases. Initially, with only 10% alloy atoms in the Fig.4(d), it is not easy to observe a nucleation for alloy atoms. Most parts of the configuration are occupied by the host atoms. When the fraction increases, an obvious pattern can be noticed that alloy atoms nucleate into large grains to decrease $H(\sigma)$ because the number of unlike pairs will contribute higher $H(\sigma)$ based on the Eq.(2) and (3). When the alloy fraction increases, the system is naturally obtained a rise in the number of alloy atoms. In virtue of common sense, the probability for a host atom to have unlike pairs will increase due to the rising number of alloy atoms under the unchanged conditions. Therefore, given the same L and J larger than zero, the probability of unlike neighbours larger than zero increases as f increases.

Interaction energy determines whether the atoms will tend to nucleate or disperse in the matrix. According to the Fig.4(g) to 4(i), the interaction energy changes from negative to positive with a mid-value as zero. The general trend displayed shows that negative J makes atoms separate as individuals with more unlike neighbours due to a decrease of $H(\sigma)$, deduced from the Eq.(2) and (3). On the opposite side, the system with positive J tends to form large grains consisting same type atoms to diminish the number of unlike neighbours. When the interaction energy is equal to zero, the configuration will be in a random distributing state. In the other words, ΔE is a constant zero and atoms are randomly distributed in the system because of an unity $W_{n\to m}$. As we can see in the Fig.4(h), there are some grains consisting more alloy atoms than the Fig.4(g) but less than the Fig.4(i), and more individual alloy atoms than the Fig.4(i) but less than the Fig.4(g).

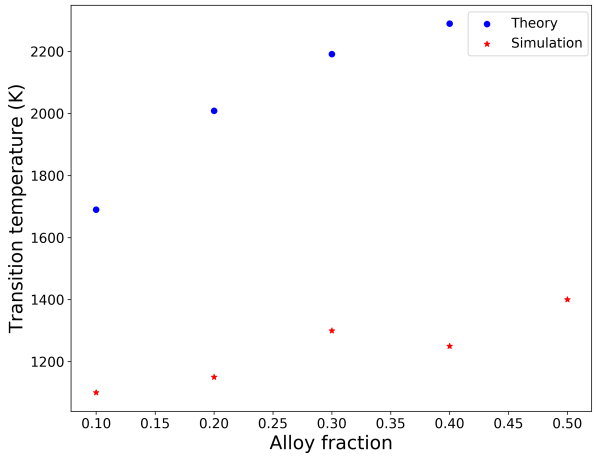


Fig. 8: Transition temperature of simulation and literature against to the alloy fraction. The blue points represent the literature values, and the red stars represent the simulating values. L = 40, $nSweeps$ = 4000000, $nEquil$ = 000000, J = -0.1 eV

From the Fig.6(a) to 6(c), the general trend of the change of the average order parameter against the temperature differs dramatically with different interaction energy, but follows

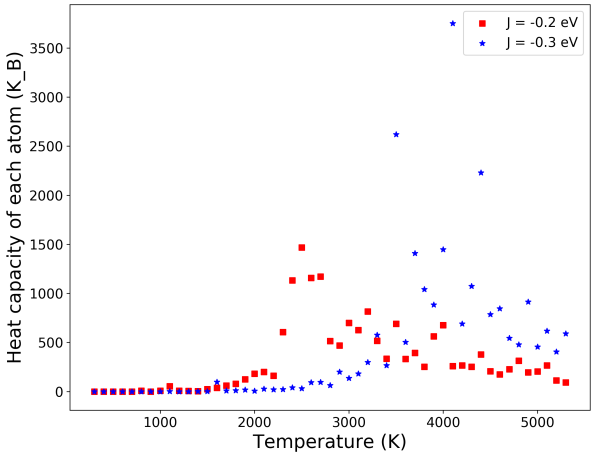


Fig. 9: Heat capacity of each atom against the temperature change with different interaction energy. L = 40, $nSweeps$ = 4500000, and $nEquil$ = 4000000.

the exactly same principle mentioned above in the General results. Fig.5(a) to 5(c) and Fig.4(g) to Fig.4(i) also support the explanation. Negative interaction energy tends to separate the atoms individually without forming large grains, which results a homogeneous mixed solute. Randomly distributed values of the average order parameter matches what we expect theoretically.

In the Fig.7, the transition temperature of a system with the given f and J is the peak of the heat capacity curve represented in red stars. Simultaneously, the transition temperature can be also found from the average order parameter curve at the position with infinite gradient. By observing the figures, the inconsistency between the peak and the infinite gradient position is caused due to the insufficient move steps to reach the fully equilibrium. The two positions should possess the same phase transition (separation) temperature. From the Fig.7, it is clear to see lots of noise that disturbs looking for the peak. One of the reasons haven been mentioned, and another one could be the oscillation of $H(\sigma)$ with increase of the temperature. This is available to be proved by the Fig.12 where higher temperature leads to larger energy oscillation around a fixed value even after 4000000 steps.

Based on the two dimensional binary alloy model in the form of a strictly regular solution, the *Helmholtz free energy* of the system, F_m, can be obtained in the Eq.(12)

$$F_m = Nsf(1-f)U + NkT[f\ln f + (1-f)\ln(1-f)] \quad (12)$$

where N is the total number of atoms in the system, and s is the number of nearest neighbours. From the F_m, the phase separation energy, T_{ps}, is derived by differentiating and then equating it to zero in the Eq.(13)

$$T_{ps} = -\frac{sU(1-2f)}{2k\ln\frac{1-f}{f}} \quad (13)$$

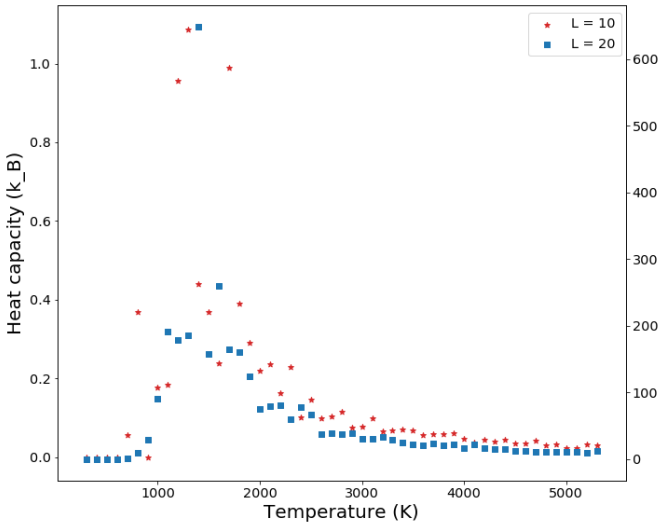


Fig. 10: Heat capacity variation with the temperature for the atom in the alloy matrix with given f and J. X axis represents the temperature and Y axis represents the heat capacity for each atom. Left axis is for the system with L = 10, and the right one is for L = 20. J = -0.1 eV, $nSweeps$ = 4500000 and $nEuil$ = 4000000, and f = 0.5.

When the f is equal to 0.5, the T_{ps} will be the critical temperature, T_c, of the system by taking the limit of f in the Eq.(13). The T_c is in the Eq.(14).

$$T_c = -\frac{sU}{4k} \tag{14}$$

It is trivial to see that the T_{ps} reaches the maximum when it is equal to T_c, which means that the transition temperature of the system should be maximum when the f is 0.5.

In the Fig.8, the transition temperature changes with the alloy fraction in the system with a negative interaction energy which can be correctly deduced from the Eq.(13). The literature values show the theoretical trend of the transition temperature for the regular binary alloy. The large difference between two set of values due to the lack of move steps to reach the sufficient equilibrium, which can be observed from some oscillations along the general trend from the Fig.7(a) to the Fig.7(c). It is reasonable to obtain the maximum transition temperature when the fraction is 0.5 because the first order transition turns into the second order transition at this point [11]. In the Fig.9, the interaction energy physically increases from -0.2 to -0.3eV, and the transition temperature shows a trend of increase, which is also implied by the Eq.(13).

Between the Fig.10 and the Fig.7(c),the peak of the heat capacity curve sharpens, and less noise can be found under the peak with smaller oscillation in the Fig.7(c). The probability distribution for order parameters is improved with larger system size based on the increase of the sample size, therefore the result approaching to the value under the infinite size rigid lattice situation. In conclusion, more accurate transition temperature can be found from both curves when the system size is large enough, but the transition temperature will not change with the system size, which can be observed from the Fig.10 where both peaks are at the nearly same temperature in the acceptable statistical error. Compared with the heat capacity, the smooth and consistent average order parameter curve is more feasible to be achieved with the same amount of move steps, thereby it being a better way to check the transition temperature when the move steps are not sufficient to obtain a clear heat capacity curve. When the system size is small ($i.e$ L=10), the system can achieve the equilibrium in a reachable move steps. For a system with large size ($i.e$ L=100), a huge number of move steps will be required to reach the equilibrium. In principle, the system with larger size will give a result of sharper peak for the transition temperature. The reason that we cannot see it here is due to the computational limitations, which can be solved with high performance computer for more move steps. However, this will cause a high expense of computational time.

ACKNOWLEDGEMENT

Part of the computational works is performed on the computer in the Department of Materials, Imperial College London. The support of the access was provided by Professor Andrew Horsfield.

REFERENCES

[1] T. Muto and Y. Takagi, "The theory of order-disorder transitions in alloys," in Solid State Physics, F. Seitz and D. Turnbull, Eds. 1955,DOI: https://doi.org/10.1016/S0081-1947(08)60679-7. pp. 193.

[2] P. Fratzl et al, "Coarsening in the Ising model with vacancy dynamics," Physica A: Statistical Mechanics and its Applications, vol. 279, (1), pp. 100-109, 2000. DOI: 10.1016/S0378-4371(99)00527-0.

[3] Dieter, George E. "Mechanical Metallurgy". 3rd ed. New York ; London: McGraw-Hill, 1986. Print. McGraw-Hill Ser. in Materials Science and Engineering, pp. 212.

[4] D. P. Landau and K. Binder, "Introduction," in A Guide to Monte Carlo Simulations in Statistical Physics, 4th ed., Cambridge: Cambridge University Press, 2014, pp. 1–6.

[5] W. Schweika 1956-, "Disordered Alloys : Diffuse Scattering and Monte Carlo Simulations". 1998, pp. 1-3.

[6] C. Huang and J. Marian, "A generalized Ising model for studying alloy evolution under irradiation and its use in kinetic Monte Carlo simulations," Journal of Physics. Condensed Matter : An Institute of Physics Journal, vol. 28, (42), pp. 425201, 2016. DOI: 10.1088/0953-8984/28/42/425201.

[7] K. Kawasaki, "Diffusion Constants near the Critical Point for Time-Dependent Ising Models. I," Phys. Rev., vol. 145, (1), pp. 224-230, 1966. DOI: 10.1103/PhysRev.145.224.

[8] R. Cerf, "The wullf crystal in lsing and percolation models," Lecture Notes in Mathematics, vol. 1878, pp. 1-264, 2006.

[9] Kittel, Charles., and McEuen, Paul. Introduction to Solid State Physics. 8th ed. New York, ; Chichester: Wiley, 2005. Print, pp. 629-631.

[10] Anonymous "A simple model of binary alloys," in Introduction to the Theory of Metastable and Unstable States, J. D. Gunton and M. Droz, Eds. 1983, DOI: 10.1007/BFb0035333.

[11] Department of Physics, "Binary mixtures", Royal Holloway University of London. Website:http://personal.rhul.ac.uk/UHAP/027/PH4211/PH4211˙files/slides7.pdf. Access: 02/03/2020.

V. Appendix

```
[3]▸ import numpy as np...

     [[8 9]
      [8 1]
      [7 0]
      [9 0]]
```

Fig. 11: Example result of coordinates of neighbours around the chosen atom by GETNEIGHBOURS function. L=10.

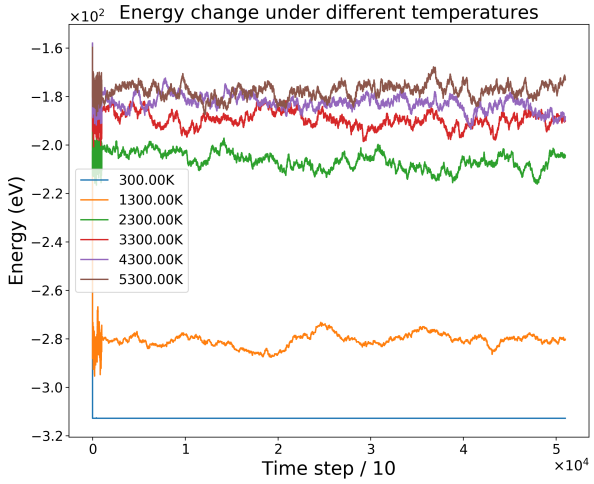


Fig. 12: Energy of the system for every 10 move steps under different chosen temperatures. L=40, $nSweeps$=4500000, $nEquil$=4000000, J=-0.1 eV.

```
[7]▸ import numpy as np...

     The number of neighbours, N, is: [0. 1. 2. 3. 4.]
     The probability distribution of N is [0.57 0.28 0.05 0.02 0.08]
```

Fig. 13: The number of neighbours and respective probability distribution for a binary alloy. L=10, f=0.1.

```
[11]▸ import numpy as np...

      0.0
```

Fig. 14: The energy change for a swap between the atom x in the Fig.3 and the top atom of it in the system with L=10 and J=-0.1.

```
----------------------------------------
The first chosen atom is: (5, 0)      Atom x
The second atom is: (4, 0)            Atom y
The energy change is: 0.000
The transition probability is: 0.9999999999999989
The random generated number is: 0.6435
The swap is successful.
----------------------------------------
The first chosen atom is: (2, 8)
The second atom is: (3, 8)
The energy change is: 0.000
The swap is successful.
----------------------------------------
The first chosen atom is: (5, 2)
The second atom is: (5, 1)
The energy change is: 0.000
The swap is successful.
----------------------------------------
The first chosen atom is: (5, 8)
The second atom is: (5, 7)
The energy change is: 0.000
The swap is successful.
----------------------------------------
The first chosen atom is: (5, 5)
The second atom is: (4, 5)
The energy change is: 0.200
The transition probability is: 0.0004366633291301925
The random generated number is: 0.7118
The swap is failed.
```

Fig. 15: Example results of simulation details for several move steps. L=10, $nSweeps$=20, $nEquil$=10, and J=-0.1 eV.

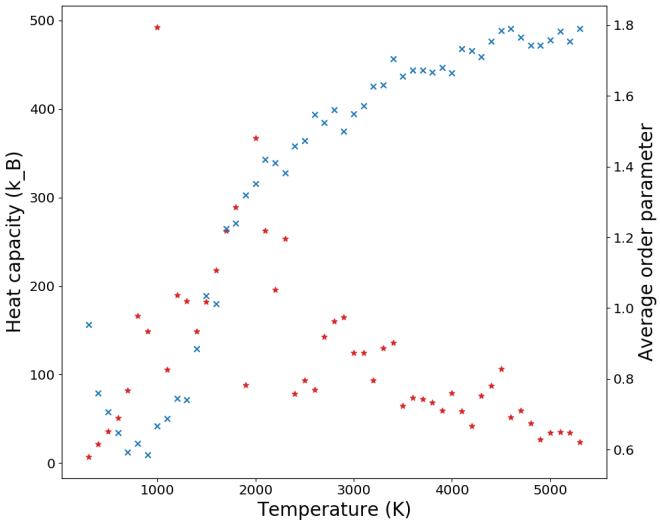


Fig. 16: Heat capacity variation and the order parameter change with the temperature for the atom in the alloy matrix with givenalloy atom fraction and the interaction energy. L=40, $nSweeps$=4500000, $nEquil$ = 4000000, and J=0.1 eV.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import matplotlib.pylab as pylab
import seaborn as sns
import gc
from scipy import constants

# Globle setting for the figure drawing
params = {'legend.fontsize': 'x-large',
          'figure.figsize': (10, 8),
          'axes.labelsize': 20,
          'axes.titlesize': 20,
          'xtick.labelsize': 'x-large',
          'ytick.labelsize': 'x-large',
          'patch.edgecolor': 'white'}
pylab.rcParams.update(params)

# Globle variables
cellA = 0
cellB = 1
k = constants.value(u'Boltzmann constant in eV/K')


def swapInfo(ixa, iya, dab, natoms, config, size, Eam, T):
    """
    Description:
    SWAPINFO returns the position of the neighbour and the energy change.

    Positional arguments:
    -> ixa:         X coordinate of first atom                                          int
    -> iya:         Y coordinate of first atom                                          int
    -> dab:         Direction of second atom relative to first. There are four possible
                    directions, so this takes values between 1 and 4. Together with ixa and
                    ixb, this allows the position of the second atom to be computed. This
                    calculation is done by getNeighbour.
                    0: Left; 1: Right; 2: Top; 3: Bottom                               int
    -> config:      The configuration of alloy atoms                    array(size, size)
    -> natoms:      Number of atoms                                                    int
    -> size:        Dimension of the matrix                                            int
    -> T:           Temperature in Klevin                                            float
    -> Eam:         Interaction energy between two types of atoms                    float

    Output arguments:
    -> ixb:         X coordinate of second atom
    -> iyb:         Y coordinate of second atom
    -> dE:          Energy change following swap
    """
    # Get the neighbour position for atom a
    neighbours_a = getNeighbour(size, ixa, iya)
    ixb, iyb = neighbours_a[dab[0]][0], neighbours_a[dab[0]][1]

    # Get all neighbours of the ataom a except atom b
    mask_a = mask(neighbours_a, ixb, iyb)
    neighbours_a_masked = np.extract(mask_a, neighbours_a).reshape(3, 2)

    # Get all neighbours of the ataom b except atom a
    neighbours_b = getNeighbour(size, ixb, iyb)
    mask_b = mask(neighbours_b, ixa, iya)
    neighbours_b_masked = np.extract(mask_b, neighbours_b).reshape(3, 2)

    # Initialize the original energy
    dE = 0

    # Calculate the original local energy within a defined matrix
    for pair in neighbours_a_masked:
        dE -= int(config[ixa][iya] + config[pair[0]][pair[1]] == 1) * Eam
        dE += int(config[ixb][iyb] + config[pair[0]][pair[1]] == 1) * Eam

    for pair in neighbours_b_masked:
        dE -= int(config[ixb][iyb] + config[pair[0]][pair[1]] == 1) * Eam
```

```
        dE += int(config[ixa][iya] + config[pair[0]][pair[1]] == 1) * Eam

    # Ckech if the energy decreases following the transition probability rule
    if dE <= 0:
        config[ixa][iya], config[ixb][iyb] = config[ixb][iyb], config[ixa][iya]

    elif np.exp(-dE / (k * T)) > np.random.uniform(0, 1):
        config[ixa][iya], config[ixb][iyb] = config[ixb][iyb], config[ixa][iya]

    else:
        dE = 0

    return ixb, iyb, dE


def mask(neighbour, x, y):
    """
    Description:
    Generate a mask to remove the atom a that is present in four neighbours of the atom b to increase the
    efficiency.

    Input parameters:
    -> neighbour:       neighbours of the atom a                                        array(4, 2)
    -> x:               X coordinate of the atom a                                          int
    -> y:               Y coordinate of the atom b                                          int

    Return:
    -> mask:            Array in two dimensions containing True and False               array(4, 2)
    """
    # Keep the atom in the neighbours of the atom a except atom b
    mask = (neighbour[:, 0] != x) | (neighbour[:, 1] != y)
    mask = mask.reshape(4, 1)

    return np.hstack([mask, mask])


def set_up(cellA, cellB, size, fraction):
    """
    Description:
    SET_UP initiates the simulation environment which is the alloy matrix.

    Positional arguments:
    -> cellA:       Host atom (1)                                                       int8
    -> cellB:       Alloy atom (2)                                                      int8
    -> size:        Length of the matrix                                                int8
    -> fraction:    Composition of the alloy atom                                      float

    Return:
    -> Matrix:      Alloy matrix                                            array(side, side)
    """
    # Initiates the alloy matrix
    natoms = int(size**2)               # Total number of atoms
    nB = int(fraction * (natoms))       # The number of alloy atoms
    nA = int(natoms - nB)               # The number of host atoms

    # Concatenate host atoms and alloy atoms, and shuffle the positions
    matrix = np.concatenate(
        (np.zeros((1, nA), dtype='int'), np.ones((1, nB), dtype='int')),
        axis=None
    )
    np.random.shuffle(matrix)           # Randomly distribute atoms

    # Reshape into the required dimension
    return np.reshape(matrix, (size, size))


def orderRandom(Z, f):
    """
    Description:
    ORDERRANDOM produces a distribtion function of the order parameter for a random alloy. The order
    parameter is just the number of AB bonds around a site. The distribution is computed from the binomial
    distribution.

    Input arguments:
```

```
    -> Z:               The number of neighbouring sites per site                                   int8
    -> f:               The fraction of alloy atoms                                               float8

    Output arguments:
    -> N:               List of possible number of neighbours                                       list
    -> P:               The probability distribution of the order parameter                         list
    """
    # Initialise the probability distribution
    N = np.linspace(0,Z,Z+1)
    P = np.linspace(0,0,Z+1)

    # Run over allowed number of AB bonds around each site and compute the
    # probability distribution in the form of binomial distribution
    for n in N:
        Z_C_n = math.factorial(Z) / (math.factorial(n) * math.factorial(Z - n))
        P[int(n)] = Z_C_n * (f * (f**(Z-n)) * ((1 - f)**n)
                  + (1 - f) * (f**n) * (1 - f)**(Z - n))

    return N, P


def order2D(config):
    """
    Description:
    ORDER2D produces a distribtion function of the order parameter. The order parameter is just the number
    of AB bonds around a site.

    Input arguments:
    -> config:     The configuration                                                  array(size, size)

    Output arguments:
    -> N:          List of possible number of neighbours                                  array(1, 4)
    -> P:          The probability distribution of the order parameter                    array(1, 4)
    """
    size = config.shape[0]                                          # Scale of the alloy matrix
    Z = 4                                                           # Number of the nearest neighbours
    N = np.linspace(0, Z, Z+1)                                      # List of possible number of neighbours
    P = np.zeros(5)                                                 # Probability distribution

    # Count the number of each order paramenter
    for x in np.arange(config.shape[0]):

        for y in np.arange(config.shape[1]):                        # Pointwise atom from (0, 0)
            count = 0                                               # Number of unlike pairs
            neighbours = getNeighbour(size, x, y)                   # Neighbour array

            for pair in neighbours:                                 # Count unlike pairs
                if config[x][y] + config[pair[0]][pair[1]] == 1:
                    count += 1

            P[count] += 1                                           # Update the order parameters

    P = P / (size**2)                                               # Probability distribution

    return N, P


def getNeighbour (size, x, y):
    """
    Description:
    GETNEIGHBOUR returns the position of a neighbouring atom.

    Input arguments:
    -> size:                The size of the simulation box                                          int
    -> x:                   X coordinate of first atom                                              int
    -> y:                   Y coordinate of first atom                                              int

    Output arguments:
    -> Array:               Contains all nearest four neighbours of the atom                      array
    """
    # Get all four nearest neighbours around the chosen atom
    neighbours = np.array([
        [x, y - 1],     # Left
        [x, y + 1],     # Rright
```

```
        [x - 1, y],     # Top
        [x + 1, y]      # Bottom
    ])

    # Check whether the coordinates out of the boundary of the matrix
    # If out, change them as the periodic boundary condition
    # Return the new coordinates
    return periodic_boundary(neighbours, size)


def periodic_boundary(arr, size):
    """
    Description:
    Change the position of the atom that runs out of the boundary.

    Positional arguments:
    -> arr:                 Array of coordinates                                    array
    -> size:                Dimension of the matrix                                   int

    Return:
    -> x:                   Changed X coordinate                                      int
    -> y:                   Changed Y coordinate                                      int
    """
    # Check whether the X and Y coordinates out of the matrix dimension
    for pair in arr:
        if pair[0] > (size - 1):      # Larger than the upper bound
            pair[0] = 0               # Change to the lower bound

        elif pair[0] < 0:
            pair[0] = size - 1

        if pair[1] > (size - 1):
            pair[1] = 0

        elif pair[1] < 0:
            pair[1] = size - 1

    return arr


def alloy2D(size, fAlloy, nSweeps, nEquil, T, Eam, job, T_list):
    """
    Description:
    ALLOY2D Performs Metropolis Monte Carlo of a lattice gas model of an alloy. A random alloy is
    represented as a 2 dimensional lattice gas in which alloying atoms can exchange position with matrix
    atoms using the Metropolis algorithm. The purpose is to show how alloys become more random as the
    temperature increases.

    Input arguments:
    -> size:    The dimension of the matrix                                         int
    -> fAlloy:  The fraction of sites occupied by alloying atoms                  float
    -> nSweeps: The total number of Monte Carlo moves                               int
    -> nEquil:  The number of Monte Carlo moves used to equilibrate the system      int
    -> T:       The temperature (K)                                               float
    -> Eam:     Alloy-matrix interaction energy (eV)                              float
    -> job:     Name or number given to this simulation.                         string
                Useful for creating file names
    -> T_list:  Temperature list                                                   list

    Output arguments:
    -> nBar:    The average number of unlike neighbours                             int
    -> Ebar:    The average energy                                                float
    -> C:       The heat capacity                                                 float
    -> Etable:  Energy to move steps under a fixed temperature                     list
    """
    # Set up the matrix
    config = set_up(cellA, cellB, size, fAlloy)

    Eo = 0                  # The initial total energy of the matrix
    step = 0                # The initial step for energy record
    Etable = []             # Record the energy per 1000 step
    natoms = size**2        # Number of total atoms
    step_interval = 1000    # Step for recording the data

```

```
    # Calculate the initial energy
    for x in np.arange(config.shape[0]):

        for y in np.arange(config.shape[1]):
            neighbours = getNeighbour(size, x, y)    # Four nearest neighbours

            for pair in neighbours:
                Eo += int(config[x, y] + config[pair[0], pair[1]] == 1) * Eam

    Eo = Eo / 2             # Doubled energy by calculating each bond energy twice
    Etable.append(Eo)       # Initial energy

    # Randomly generate the number equal to nSweeps of positions to make swaps
    positions, directions = generator(nSweeps, size)

    # Swap the atoms based on the energy change
    for step in np.arange(nSweeps):
        ixb, iyb, dE = swapInfo(positions[0][step], positions[1][step],
                                directions[step], natoms, config,
                                size, Eam, T)

        # Eneregy change
        Eo += dE

        # Record the data per 1000 step
        if step >= nEquil and step % step_interval == 0:
            Etable.append(Eo)


    # After reaching the equilibrium, run more steps for the
    # phase transition temperature
    test_steps = 50000      # More steps for finding the transition temperature

    # Randomly generate two arrays for swapping atoms
    positions, directions = generator(test_steps, size)
    nStat = test_steps

    # Run more steps
    for step in np.arange(test_steps):
        ixb, iyb, dE = swapInfo(positions[0][step], positions[1][step],
                                directions[step], natoms, config,
                                size, Eam, T)

        Eo += dE
        Etable.append(Eo)


    # Choose some states to draw the figure
    # The interval is adjustable for request
    T_figure = T_list[::10]

    if T in T_figure:
        # Plot the configuration
        # Put extra zeros around border so pcolor works properly.
        config_plot = np.zeros((size+1, size+1))
        config_plot[0:size, 0:size] = config
        fig = plt.figure(dpi=300)
        ax = fig.add_subplot(111)
        ax.pcolor(config_plot)
        ax.set_title("Schematic configuration of the alloy")
        ax.set_xlabel("X axis boundary")
        ax.set_ylabel("Y axis boundary")
        fig.savefig(r'...\Config\{}---config.png'.format(job))
        plt.close(fig)
        gc.collect()

        diff = nSweeps - nEquil
        # Plot the energy change with fixed fraction and interaction energy under a temperature
        fig = plt.figure(dpi=300)
        ax = fig.add_subplot(111)
        ax.plot(Etable[0: int(diff/step_interval)], linewidth=1.5)
        ax.set_title ("Energy change to the move steps of an alloy")
        ax.set_xlabel("Time step / {}".format(step_interval))
        ax.set_ylabel("Energy (eV)")
```

```
        ax.ticklabel_format(axis='x', style='sci', scilimits=(0,0))
        ax.ticklabel_format(axis='y', style='sci', scilimits=(0,0))
        ax.xaxis.major.formatter._useMathText = True
        ax.yaxis.major.formatter._useMathText = True
        ax.xaxis.get_offset_text().set_fontsize(15)
        ax.get_xaxis().get_major_formatter().set_useOffset(True)
        ax.yaxis.get_offset_text().set_fontsize(15)
        ax.get_yaxis().get_major_formatter().set_useOffset(True)
        fig.savefig(r'...\Energy\{}---energy.png'.format(job))
        plt.close(fig)
        gc.collect()

        # Plot the final neighbour distribution
        N, P = order2D(config)
        N0, P0 = orderRandom(4, fAlloy)
        fig = plt.figure(dpi=300)
        ax = fig.add_subplot(111)
        bar_width = 0.35
        ax.bar(N, P, bar_width, label="Simulation")
        ax.bar(N0+bar_width, P0, bar_width, label="Random")
        ax.set_title ("Distribution of unlike neighbours")
        ax.set_xlabel("Number of unlike neighbours")
        ax.set_ylabel("Probability")
        ax.legend(loc=0)
        fig.savefig(r'...\order\{}-order.png'.format(job))
        plt.close(fig)
        gc.collect()


    # Average order parameter
    N, P = order2D(config)
    nBar = np.dot(N, P)

    # Average energy for each atom
    Etable = np.asarray(Etable)
    E_unit_bar_square = ((sum(Etable[-nStat:]) / nStat)**2)

    # Average energy square for each atom
    E2_unit_bar = sum(Etable[-nStat:]**2) / (nStat)

    # Heat capacity for each atom in the unit per K_B
    C = (E2_unit_bar - E_unit_bar_square) / ((k**2) * (T**2))

    print('')
    print('Heat capacity = {0:7.7f}'.format(C),' kB')
    print('The average number of unlike neighbours is = {0:7.3f}'.format(nBar))

    # Return the statistics
    return nBar, E_unit_bar_square, C, Etable


def generator(N1, size):
    """
    Description:
    Ramdonly generate two arrays with the chosen atoms and the direction pointing to the second atom.

    Positional arguments:
    -> N1:      limit for the number of atoms required to be swapped                                int8
    -> size:    The scale of the alloy matrix                                                       int8

    Return:
    -> positions:       coordinates of the first atom                                       array(2, N1)
    -> directions:      number indicating the second atom                                   array(N1, 1)
    """
    # Randomly generate the number equal to nSweeps of positions to make swaps
    positions = np.random.randint(0, size, (2, N1), dtype='int')

    # Randomly generate the directions for each swap
    directions = np.random.randint(0, 4, (N1, 1), dtype='int')

    return positions, directions


def main():
```

```
    """
    Description:
    Main function to simulate the phase transition of the alloy with several parameters, make some related
    and useful plots.

    Positional arguments:
    None

    Return:
    None
    """
    # Define the simulation parameters
    size = 60
    nEquil = 200000
    nSweeps = 300000
    fAlloy_list = [0.5, 0.4, 0.3, 0.2, 0.1]
    T_downlim = 300
    T_uplim = 5300
    T_interval = 100
    Eam_list = [-0.1, 0.0, 0.1]
    step_interval = 1000

    # Temperature list
    nT = int((T_uplim - T_downlim)/ T_interval) + 1
    T_list = np.linspace(T_downlim, T_uplim, nT)

    # Open file to save the statistics
    file = open ("stats.csv", "w")
    file.write('Job number, Alloy fraction, Temperature (K), Unlike bond energy (eV), Average number of
    unlike neighbours, Average energy (eV), Heat capacity (kB)\n')

    # Loop over values
    for fAlloy in fAlloy_list:

        for Eam in Eam_list:
            orders = list()             # Data of the average order parameters
            C_list= list()              # Data of the heat capacity
            E_assemble = list()         # Data of the total energy for each temperature

            for T in T_list:
                # Simulation details
                job = '{}f_{}eV_{:4.2f}K'.format(fAlloy, Eam, T)

                # Echo the parameters back to the user
                print ("")
                print ("Simulation ", job)
                print ("----------------")
                print ("Cell size = ", size)
                print ("Alloy fraction = ", fAlloy)
                print ("Total number of moves = ", nSweeps)
                print ("Number of equilibration moves = ", nEquil)
                print ("Temperature = ", T, "K")
                print ("Bond energy = ", Eam, "eV")

                # Run the simulation
                nBar, Ebar, C, Etable = alloy2D(size, fAlloy, nSweeps, nEquil, T, Eam, job, T_list)

                # Store the data
                E_assemble.append(Etable)
                C_list.append(C)
                orders.append(nBar)

                # Write out the statistics
                file.write('{0}, {1:6.4f}, {2:8.2f}, {3:5.2f}, {4:6.4f}, {5:14.7g}, {6:14.7g}\n'.format(job
    , fAlloy, T, Eam, nBar, Ebar, C))


            # Plot the heat capacity against the temperature
            name = '{0}_{1}'.format(fAlloy, Eam)

            fig = plt.figure(dpi=300)
            ax = fig.add_subplot(111)
            ax.scatter(T_list, C_list)
            ax.set_title("Effect of temperature change to\n the heat capacity per atom")
```

```
            ax.set_xlabel("Temperature (K)")
            ax.set_ylabel("Heat capacity (/k_B)")
            ax.ticklabel_format(axis='y', style='sci', scilimits=(0,0))
            ax.yaxis.major.formatter._useMathText = True
            ax.yaxis.get_offset_text().set_fontsize(15)
            ax.get_yaxis().get_major_formatter().set_useOffset(True)
            fig.savefig(r'...\C\{}---T_C.png'.format(name))
            plt.close(fig)
            gc.collect()

            # Plot the variation of the average order parameter against the temperature
            fig = plt.figure(dpi=300)
            ax = fig.add_subplot(111)
            ax.scatter(T_list, orders)
            ax.set_title("Variation of the average order parameter against to the temperature")
            ax.set_xlabel("Temperature (K)")
            ax.set_ylabel("Order parameter (unit)")
            fig.savefig(r'...\T_order\{}---T_order.png'.format(name))
            plt.close(fig)
            gc.collect()

            # Plot several energy curve under different temperature for general trends
            fig = plt.figure(dpi=300)
            ax = fig.add_subplot(111)
            diff = nSweeps - nEquil
            for i in range(0, len(E_assemble), 10):
                ax.plot(E_assemble[i][0:int(diff/step_interval)], label='{:.2f}K'.format(300 + T_interval*i
    ), linewidth=1.5)


            ax.legend(loc=0)
            ax.set_title ("Energy change under different temperatures")
            ax.set_xlabel("Time step / {0:d}".format(step_interval))
            ax.set_ylabel("Energy (eV)")
            ax.ticklabel_format(axis='x', style='sci', scilimits=(0,0))
            ax.ticklabel_format(axis='y', style='sci', scilimits=(0,0))
            ax.xaxis.get_offset_text().set_fontsize(15)
            ax.yaxis.get_offset_text().set_fontsize(15)
            ax.xaxis.major.formatter._useMathText = True
            ax.yaxis.major.formatter._useMathText = True
            ax.get_xaxis().get_major_formatter().set_useOffset(True)
            ax.get_yaxis().get_major_formatter().set_useOffset(True)
            fig.savefig(r'E:\Coding\alloy_phase_transition\Energy\{}---energy.png'.format(name))
            plt.close(fig)
            gc.collect()

            # Plot C and n on the same figure for convenience
            fig, ax1 = plt.subplots()

            # Plot C
            color = 'tab:red'
            ax1.set_xlabel('Temperature (K)')
            ax1.set_ylabel('Heat capacity (k_B)')
            ax1.scatter(T_list, C_list, color=color, marker='*')

            ax2 = ax1.twinx()  # For plotting n

            color = 'tab:blue'
            ax2.set_ylabel('Average order parameter')         # we already handled the x-label with ax1
            ax2.scatter(T_list, orders, color=color, marker='x')

            fig.tight_layout()  # otherwise the right y-label is slightly clipped
            fig.savefig(r'...\Order_C\{}_{}---Order_C.png'.format(fAlloy, Eam))
            plt.close(fig)
            gc.collect()


    # Close the file
    file.close()

    # Sign off
    print('')
    print ("Simulations completed.")

```

```

if __name__ == "__main__":
    main()
```

Listing 1: Simulation codes