1 Algorithms

The focus of the following algorithm is on speeding up MC simulations of a binary alloy system using the Ising Model with a coupling constant of $J > 0$. The algorithm can be generalized for more complex systems.

Algorithm 1: Important Variables

```
**The following are variables that are constant in the code**
// Base Probability corresponding to boundary-boundary moves
$prob\_bb\_base = some\_value$
// Base Probability corresponding to boundary-interior moves
$prob\_bi\_base = some\_value$
// Base Probability corresponding to interior-interior moves
$prob\_ii\_base = some\_value$
// Expansion probability of adding neighbors when the proposed initial move is a
    boundary-boundary pair
$prob\_bb\_expansion = some\_value$
// Expansion probability of adding neighbors when the proposed initial move is a
    boundary-interior pair
$prob\_bi\_expansion = some\_value$
// Expansion probability of adding neighbors when the proposed initial move is an
    interior-interior pair
$prob\_ii\_expansion = some\_value$
**All the following variables are updated at each iteration of the simulation**
// List of boundary atom lattice positions of specie1
$Boundary\_Atoms\_Specie1$
// List of interior atom lattice positions of specie1
$Interior\_Atoms\_Specie1$
// List of boundary atom lattice positions of specie2
$Boundary\_Atoms\_Specie2$
// List of interior atom lattice positions of specie2
$Interior\_Atoms\_Specie2$
**The following four hash tables are used to achieve O(1) insert and delete into its
  corresponding list pair**
// Hash table that hashes the lattice positions to its corresponding index in
    Boundary_Atoms_Specie1
$Boundary\_Atoms\_Specie1\_Hash$
// Hash table that hashes the lattice positions to its corresponding index in
    Interior_Atoms_Specie1
$Interior\_Atoms\_Specie1\_Hash$
// Hash table that hashes the lattice positions to its corresponding index in
    Boundary_Atoms_Specie2
$Boundary\_Atoms\_Specie2\_Hash$
// Hash table that hashes the lattice positions to its corresponding index in
    Interior_Atoms_Specie2
$Interior\_Atoms\_Specie2\_Hash$
```

```
// Lengths of the lists
$numb\_specie1 = len(Boundary\_Atoms\_Specie1)$
$numi\_specie1 = len(Interior\_Atoms\_Specie1)$
$numb\_specie2 = len(Boundary\_Atoms\_Specie2)$
$numi\_specie2 = len(Interior\_Atoms\_Specie2)$
// Number of boundary-boundary pairs
$num\_bb = numb\_specie1 * numb\_specie2$
// Number of boundary-interior pairs
$num\_bi = numb\_specie1 * numi\_specie2 + numi\_specie1 * numb\_specie2$
// Number of interior-interior pairs
$num\_ii = numi\_specie1 * numi\_specie2$
/* Normalizing factor to compute the swap-type probability of the initial pair    */
$Norm = prob\_bb\_base * num\_bb + prob\_bi\_base * num\_bi + prob\_ii\_base * num\_ii$
// probability that the initial pair is a boundary-boundary pair
$prob\_bb\_Initial = prob\_bb\_base * num\_bb/Norm$
// probability that the initial pair is a boundary-interior pair
$prob\_bi\_Initial = prob\_bi\_base * num\_bb/Norm$
// probability that the initial pair is an interior-interior pair
$prob\_ii\_Initial = prob\_ii\_base * num\_bb/Norm$
```

Algorithm 2: Propose Initial Pair of Atoms to Swap

Input : $prob\_bb\_Initial$, $prob\_bi\_Initial$, $prob\_ii\_Initial$, swap-type probabilities for initial pair

Input : $Boundary\_Atoms\_Specie1$, $Interior\_Atoms\_Specie1$, $Boundary\_Atoms\_Specie2$, $Interior\_Atoms\_Specie2$, Lists of different types of lattice positions

Input : $num\_bi, numb\_specie1, numi\_specie2$, number of boundary-interior pairs along with the number of boundary atoms of specie1 and number of interior atoms of specie2

Output: $Initial\_Atom1$, $Initial\_Atom2$, lattice positions of proposed initial pair of atoms to swap (Note: the two atoms have different specie type)

```
// Create pmf(probability mass function) for the swap-type probabilities of the
   initial pair
$pmf = [prob\_bb\_Initial, \quad prob\_bi\_Initial, \quad prob\_ii\_Initial]$
// create cdf(cumulative distribution function) from the pmf
$cdf = [prob\_bb\_Initial, \quad prob\_bb\_Initial + prob\_bi\_Initial, \quad 1]$
// Used to draw a move
$draw = Null$
$r = \texttt{random}(0, 1)$
if $r <= cdf[0]$ then
    $draw = 0$
else
    for $i$ in $range(1, \ len(cdf))$ do
        if $r > cdf[i-1]$ and $r <= cdf[i]$ then
            $draw = i$
        end if
    end for
end if
// Proposed Move is a boundary-boundary pair
if $draw == 0$ then
    // Draw a pair from Boundary_Atoms_Specie1 and Boundary_Atoms_Specie2
end if
// Proposed Move is a boundary-interior pair
if $draw == 1$ then
    $u = (numb\_specie1 * numi\_specie2)/num\_bi$
    $r = \texttt{random}(0, 1)$
    if $r < u$ then
        // Draw a pair from Boundary_Atoms_Specie1 and Interior_Atoms_Specie2
    else
        // Draw a pair from Boundary_Atoms_Specie2 and Interior_Atoms_Specie1
    end if
end if
// Proposed Move is an interior-interior pair
if $draw == 2$ then
    // Draw a pair from Interior_Atoms_Specie1 and Interior_Atoms_Specie2
end if
return $Initial\_Atom1, \ Initial\_Atom2$
```

Algorithm 3: Expanding Swaps

Input : *Initial_Atom1*, *Initial_Atom2*, lattice positions of proposed initial pair of atoms to swap

Input : *prob_bb_expansion*, *prob_bi_expansion*, *prob_ii_expansion*, expansion probabilities

Output: *changed_atoms_list*, List of lattice positions of atoms that will undergo a change in specie

```
// List of lattice positions of atoms that will undergo a change in specie
$changed\_atoms\_list = [\,]$
// This variable is set to the appropriate expansion probability based on the
   swap-type of the initial pair of atoms
$prob\_expansion = 0.0$
// swap-type of Initial pair is boundary-boundary
if SwapType($Initial\_Atom1, Initial\_Atom2$) == $bb$ then
    $prob\_expansion = prob\_bb\_expansion$
end if
// swap-type of Initial pair is boundary-interior
else if SwapType($Initial\_Atom1, Initial\_Atom2$) == $bi$ then
    $prob\_expansion = prob\_bi\_expansion$
end if
// swap-type of Initial pair is interior-interior
else if SwapType($Initial\_Atom1, Initial\_Atom2$) == $ii$ then
    $prob\_expansion = prob\_ii\_expansion$
end if
// Append the lattice positions of the initial pair of atoms into changed_atoms_list
$changed\_atoms\_list.append(Initial\_Atom1)$
$changed\_atoms\_list.append(Initial\_Atom2)$
// Initialize two empty queues, one corresponds to specie1 and the other to specie2
$queue1 = [\,]$
$queue2 = [\,]$
// Initialize an empty hash table or python dictionary
$visited\_list = \{\,\}$
// Append the lattice positions of the initial pair of atoms into their
   corresponding queues, based on their specie type
$queue1.append(Initial\_Atom1)$
$queue2.append(Initial\_Atom2)$
// Mark both atoms as visited (i.e.  place them in the hash tables)
$visited\_list[Initial\_Atom1] = True$
$visited\_list[Initial\_Atom2] = True$
```

```
while $len(queue1) \mathrel{!}= 0$ do
    // Initialize two empty neighbor lists
    $nei\_list1 = [\,]$
    $nei\_list2 = [\,]$
    for each position in $queue1$ do
        // Add the lattice positions of all the nearest neighbors of the atom at this
           lattice position into nei_list1 that are of the same atomic specie and
           have not been marked as visited
        // Mark each neighbor as visited after adding it into nei_list1
    end for
    for each position in $queue2$ do
        // Add the lattice positions of all the nearest neighbors of the atom at this
           lattice position into nei_list2 that are of the same atomic specie and
           have not been marked as visited
        // Mark each neighbor as visited after adding it into nei_list2
    end for
    // Empty both the queues
    $queue1 = [\,]$
    $queue2 = [\,]$
    if $nei\_list1$ or $nei\_list2$ is empty then
        // Break and Exit from the While loop
    end if
    // Give the neighbor lists a random permutation
    $nei\_list1 = \texttt{permute}(nei\_list1)$
    $nei\_list2 = \texttt{permute}(nei\_list2)$
    // Find the minimum size between the two neighbor list
    $min\_size = \texttt{min}(len(nei\_list1),\ len(nei\_list2))$
    // Number of atoms to swap using the neighbor lists
    $num\_to\_swap = 0$
    for $i$ in $range(min\_size)$ do
        $r = \texttt{random}(0, 1)$
        if $r < prob\_expansion$ then
            $num\_to\_swap \mathrel{+}= 1$
        end if
    end for
    if $num\_to\_swap > 0$ then
        // Add the first num_to_swap lattice positions in nei_list1 and nei_list2 into
           changed_atoms_list
        // Also add these same positions into queue1 and queue2 according to their
           atomic specie type
    end if
end while
return $changed\_atoms\_list$
```

Algorithm 4: Computing Forward and Reverse Probabilities

Input : *changed_atoms_list*, List of lattice positions of atoms that will undergo a change in specie

Input : $Initial\_Atom1$, $Initial\_Atom2$, lattice positions of proposed initial pair of atoms to swap

Input : *num_bb*, *num_bi*, *num_ii*, number of pairs for each swap-type

Input : $prob\_bb\_Initial$, $prob\_bi\_Initial$, $prob\_ii\_Initial$, swap-type probabilities for initial pair

Input : *prob_bb_expansion*, *prob_bi_expansion*, *prob_ii_expansion*, expansion probabilities

Input : *compute_only_partial*, boolean value that tells the function to compute the probability of the forward or reverse move treating $Initial\_Atom1$ and $Initial\_Atom2$ as the only possible initial pair if True, otherwise it computes the probability of the forward or reverse move using all possible initial pairs between the two sets of atoms of different specie type in *changed_atoms_list*

Output: *total_prob*, Probability of the forward or reverse move

```
1  // Set of lattice positions with atoms of specie1
2  $Atom\_set\_specie1 = [\,]$
3  // Set of lattice positions with atoms of specie2
4  $Atom\_set\_specie2 = [\,]$
5  // Add lattice positions into Atom_set_specie1 and Atom_set_specie2 based on the
      boolean value of compute_only_partial
6  if $compute\_only\_partial == True$ then
7      $Atom\_set\_specie1.append(Initial\_Atom1)$
8      $Atom\_set\_specie2.append(Initial\_Atom2)$
9  else
10     for each position in changed_atoms_list do
11         if position has an atom of specie1 then
12             $Atom\_set\_specie1.append(position)$
13         else
14             $Atom\_set\_specie2.append(position)$
15         end if
16     end for
17 end if
18 // Probability of the forward or reverse move
19 $total\_prob = 0.0$
```

```
for $initial\_atom1$ in $Atom\_set\_specie1$ do
    for $initial\_atom2$ in $Atom\_set\_specie2$ do
        $prob\_expansion = 0.0$
        // swap-type of initial pair is boundary-boundary
        if SwapType($initial\_atom1,\ initial\_atom2$) $==$ $bb$ then
            $prob\_expansion = prob\_bb\_expansion$
        end if
        // swap-type of initial pair is boundary-interior
        else if SwapType($initial\_atom1,\ initial\_atom2$) $==$ $bi$ then
            $prob\_expansion = prob\_bi\_expansion$
        end if
        // swap-type of initial pair is interior-interior
        else if SwapType($initial\_atom1,\ initial\_atom2$) $==$ $ii$ then
            $prob\_expansion = prob\_ii\_expansion$
        end if
        // The variable below is used to compute the forward or reverse probability
           using the above initial pair of atoms (i.e.  initial_atom1 and
           initial_atom2)
        $prob\_partial = 1.0$
        // Compute the probability of picking this initial pair (i.e.  initial_atom1
           and initial_atom2)
        // swap-type of initial pair is boundary-boundary
        if SwapType($initial\_atom1,\ initial\_atom2$) $==$ $bb$ then
            $prob\_partial = prob\_partial * prob\_bb\_Initial * \frac{1}{num\_bb}$
        end if
        // swap-type of initial pair is boundary-interior
        else if SwapType($initial\_atom1,\ initial\_atom2$) $==$ $bi$ then
            $prob\_partial = prob\_partial * prob\_bi\_Initial * \frac{1}{num\_bi}$
        end if
        // swap-type of initial pair is interior-interior
        else if SwapType($initial\_atom1,\ initial\_atom2$) $==$ $ii$ then
            $prob\_partial = prob\_partial * prob\_ii\_Initial * \frac{1}{num\_ii}$
        end if
        // Initialize two empty queues, one corresponds to specie1 and the other to
           specie2
        $queue1 = [\ ]$
        $queue2 = [\ ]$
        // Initialize an empty hash table or python dictionary
        $visited\_list = \{\ \}$
        // Append the lattice positions of the initial pair of atoms into their
           corresponding queues, based on their specie type
        $queue1.append(initial\_atom1)$
        $queue2.append(initial\_atom2)$
    // Mark both atoms as visited (i.e. place them in the hash tables)
    $visited\_list[initial\_atom1] = True$
    $visited\_list[initial\_atom2] = True$
    // keeps track of the number of atoms undergoing a change in specie
    $num\_added = 2$
    while $len(queue1) \, != 0$ do
        // Initialize two empty neighbor lists
        $nei\_list1 = [\,]$
        $nei\_list2 = [\,]$
        for each position in $queue1$ do
            // Add the lattice positions of all the nearest neighbors of the atom
               at this lattice position into nei_list1 that are of the same atomic
               specie and have not been marked as visited
            // Mark each neighbor as visited after adding it into nei_list1
        end for
        for each position in $queue2$ do
            // Add the lattice positions of all the nearest neighbors of the atom
               at this lattice position into nei_list2 that are of the same atomic
               specie and have not been marked as visited
            // Mark each neighbor as visited after adding it into nei_list2
        end for
        // Empty both the queues
        $queue1 = [\,]$
        $queue2 = [\,]$
        if $nei\_list1$ or $nei\_list2$ is empty then
            // Break and Exit from the While loop
        end if
        // $num\_to\_swap\_list1$ and $num\_to\_swap\_list2$ keeps track of the number of
           atoms of each specie type that has to undergo a swap for the reverse or
           forward move to happen given in the current iteration of the while loop
        $num\_to\_swap\_list1 = 0$
        $num\_to\_swap\_list2 = 0$
        // $Atoms\_to\_add\_list1$ and $Atoms\_to\_add\_list2$ keeps track of the lattice
           positions of atoms of each specie type that has to undergo a change in
           specie type (i.e. a swap) for the reverse or forward move to happen
           given in the current iteration of the while loop
        $Atoms\_to\_add\_list1 = [\,]$
        $Atoms\_to\_add\_list2 = [\,]$
```

```
            // A hash table is used to make the below searches in changed_atoms_list
               O(1)
            for $each\ position\ in\ nei\_list1$ do
                if $position\ exist\ in\ changed\_atoms\_list$ then
                    $num\_to\_swap\_list1 += 1$
                    $Atoms\_to\_add\_list1.append(position)$
                end if
            end for
            for $each\ position\ in\ nei\_list2$ do
                if $position\ exist\ in\ changed\_atoms\_list$ then
                    $num\_to\_swap\_list2 += 1$
                    $Atoms\_to\_add\_list2.append(position)$
                end if
            end for
            // If num_to_swap_list1 is not equal to num_to_swap_list2, that means that
               not all the atoms in changed_atoms_list will be picked with the chosen
               initial pair of atoms, in other words the forward or reverse move is
               not possible with the chosen initial pair of atoms
            if $num\_to\_swap\_list1\ != \ num\_to\_swap\_list2$ then
                $prob\_partial = 0.0$
                // Break from the While loop
                $Break;$
            end if
            $min\_size = \mathtt{min}(len(nei\_list1),\ len(nei\_list2))$
            $s = num\_to\_swap\_list1$
            $p = prob\_expansion$
            // Compute probability of picking s number of swaps
                $$prob\_s = \frac{min\_size!}{s!(min\_size - s)!} p^s (1-p)^{min\_size - s}$$
            // Compute probability of picking correct atoms for both neighbor lists
            $N = len(nei\_list1)$
            $M = len(nei\_list2)$
            // probability of picking correct atoms in nei_list1
                $$prob\_N = \frac{s!(N-s)!}{N!}$$
            // probability of picking correct atoms in nei_list2
            $$prob\_M = \frac{s!(M-s)!}{M!}$$
            // Update partial probability of this initial pair
            $prob\_partial = prob\_partial * prob\_s * prob\_N * prob\_M$
            // Update the queues
            for $position$ in $Atoms\_to\_add\_list1$ do
                $queue1.append(position)$
            end for
            for $position$ in $Atoms\_to\_add\_list2$ do
                $queue2.append(position)$
            end for
            // Update the variable num_added
            $num\_added = num\_added + 2 * s$
        end while
        // Required Sanity Check
        if $prob\_partial > 0$ then
            if $num\_added\ != \ len(changed\_atoms\_list)$ then
                $prob\_partial = 0.0$
            end if
        end if
        // Update total_prob by adding the probability of the forward or reverse move
           using this initial pair (i.e.  the current positions of initial_atom1 and
           initial_atom2)
        $total\_prob = total\_prob + prob\_partial$
    end for
end for
return $total\_prob$
```

Algorithm 5: Accept or Reject Move

```
Input  : $changed\_atoms\_list$, List of lattice positions of atoms that will undergo a change in specie
Input  : $Initial\_Atom1$, $Initial\_Atom2$, lattice positions of proposed initial pair of atoms to swap
Input  : $Boundary\_Atoms\_Specie1$, $Interior\_Atoms\_Specie1$, $Boundary\_Atoms\_Specie2$,
         $Interior\_Atoms\_Specie2$, Lists of different types of lattice positions
Input  : $prob\_bb\_Initial$, $prob\_bi\_Initial$, $prob\_ii\_Initial$, swap-type probabilities for initial pair
Output: $Proposed\_State$, Boolean value that indicates that the proposed move is accepted if True, otherwise it is False
1  // Lengths of the list of different lattice position types
2  $numb\_specie1 = len(Boundary\_Atoms\_Specie1)$
3  $numi\_specie1 = len(Interior\_Atoms\_Specie1)$
4  $numb\_specie2 = len(Boundary\_Atoms\_Specie2)$
5  $numi\_specie2 = len(Interior\_Atoms\_Specie2)$
6  // Number of boundary-boundary pairs
7  $num\_bb = numb\_specie1 * numb\_specie2$
8  // Number of boundary-interior pairs
9  $num\_bi = numb\_specie1 * numi\_specie2 + numi\_specie1 * numb\_specie2$
10 // Number of interior-interior pairs
11 $num\_ii = numi\_specie1 * numi\_specie2$
12 // Set to desired value
13 $compute\_only\_partial = False$
14 // Compute Forward Probability
15 $forward\_probability =$
     ComputeForwardProbability($changed\_atoms\_list$, $Initial\_Atom1$, $Initial\_Atom2$,
     $num\_bb$, $num\_bi$, $num\_ii$, $prob\_bb\_Initial$, $prob\_bi\_Initial$, $prob\_ii\_Initial$,
     $prob\_bb\_expansion$, $prob\_bi\_expansion$, $prob\_ii\_expansion$, $compute\_only\_partial$)
16 // In order to compute the reverse probability, we need to know the new lengths of
     the four lists of different lattice types (i.e.  Boundary_Atoms_Specie1,
     Interior_Atoms_Specie1, Boundary_Atoms_Specie2, and Interior_Atoms_Specie2) if the
     forward move is executed
17 // We need to record the set of all lattice positions that may need to be modified
     (i.e.  inserted or deleted) in the four lists of different lattice types if the
     forward move is executed
18 // This hash table or python dictionary stores the set of all lattice positions
     that may need to be modified in the four lists of different lattice types
19 $Changed\_lattice\_pos\_types = \{ \}$
```

```
20 for $position\ in\ changed\_atoms\_list$ do
21     $Changed\_lattice\_pos\_types[position] =\ True$
22     // The nearest neighbors of the position might also need to be modified in the
         four lists of different lattice types
23     $neighbors =$ GenNearestNeighbors($position$)
24     for $nei\ in\ neighbors$ do
25         $Changed\_lattice\_pos\_types[nei] =\ True$
26     end for
27 end for
28 // The function ComputeChangeProbAndPairs computes the new number of pairs of each
     swap-type and the new swap-type probabilities for the initial proposed pair if
     the forward move is executed (Note:  The function also takes in the current
     lengths of the four lists, but those values have been omitted for conciseness)
29 $num\_bb\_new,\ num\_bi\_new,\ num\_ii\_new,$
     $prob\_bb\_Initial\_new,\ prob\_bi\_Initial\_new,\ prob\_ii\_Initial\_new =$
     ComputeChangeProbAndPairs($Changed\_lattice\_pos\_types, ...$)
30 // The new values are used to compute the reverse probability (Note:  the function
     will first need to execute the forward move before computing the reverse
     probability, but this process is reversed before the function returns)
31 $reverse\_probability =$
     ComputeReverseProbability($changed\_atoms\_list,\ Initial\_Atom1,\ Initial\_Atom2,$
     $num\_bb\_new,\ num\_bi\_new,\ num\_ii\_new,\ prob\_bb\_Initial\_new,\ prob\_bi\_Initial\_new,$
     $prob\_ii\_Initial\_new,\ prob\_bb\_expansion,\ prob\_bi\_expansion,\ prob\_ii\_expansion,$
     $compute\_only\_partial$)
32 // Compute the change in energy from the forward move
33 $deltaE =$ ComputeEnergyChange($changed\_atoms\_list$)
34 // Compute the acceptance probability, here $\beta$ is the inverse temperature
35
     $$acceptance\_prob = min\left(1, exp(-\beta * deltaE)\frac{reverse\_probability}{forward\_probability}\right)$$
36 // Compute Proposed_Sate
37 // Proposed_State is False by default
38 $Proposed\_State =\ False$
```

```
if $acceptance\_prob == 1$ then
    $Proposed\_State = True$
    // Modify the four lists of different lattice types (i.e.
       Boundary_Atoms_Specie1, Interior_Atoms_Specie1, Boundary_Atoms_Specie2, and
       Interior_Atoms_Specie2) appropriately, also set the swap-type probabilities
       for the initial proposed pair to the new values
else
    $r = \texttt{random}(0, 1)$
    if $r < acceptance\_prob$ then
        $Proposed\_State = True$
        // Modify the four lists of different lattice types (i.e.
           Boundary_Atoms_Specie1, Interior_Atoms_Specie1, Boundary_Atoms_Specie2, and
           Interior_Atoms_Specie2) appropriately, also set the swap-type
           probabilities for the initial proposed pair to the new values
    end if
end if
return $Proposed\_State$
```