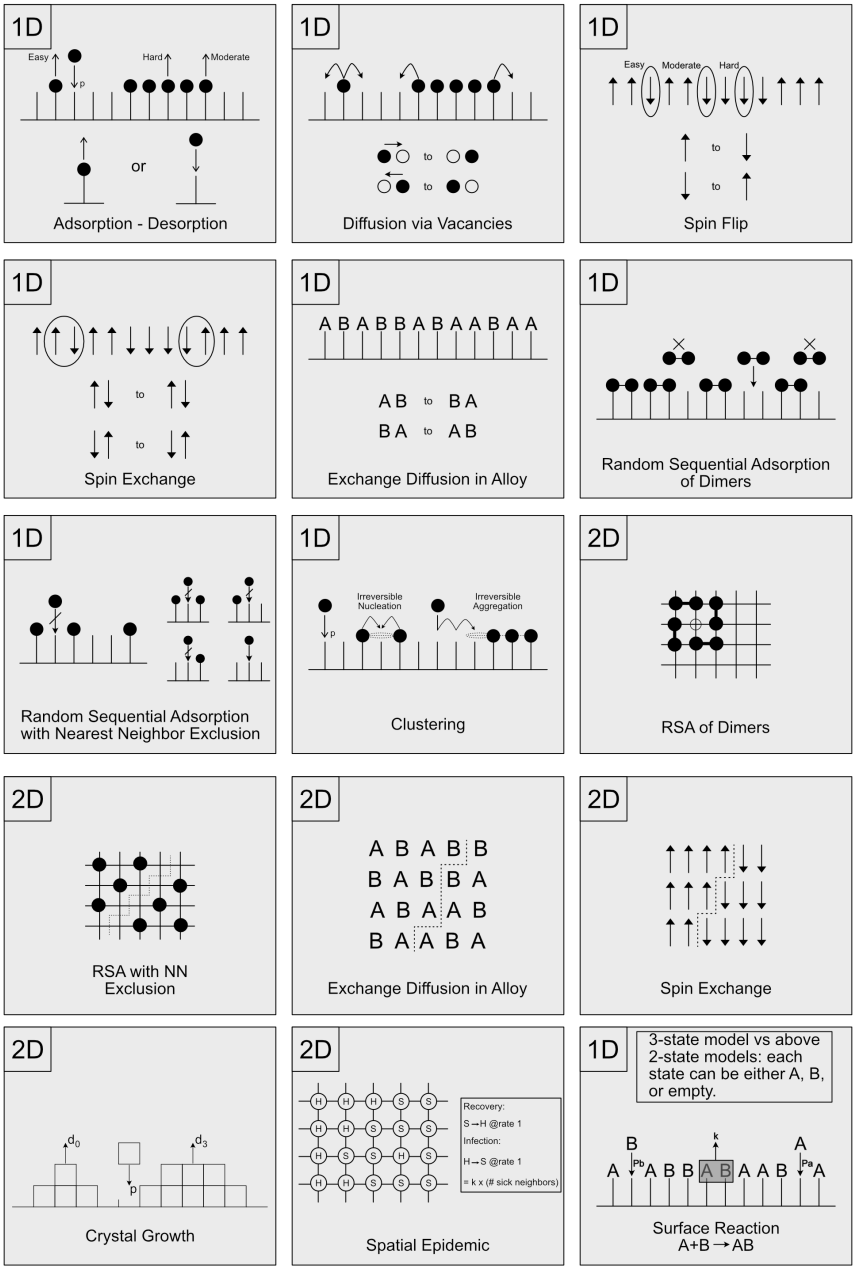
Complex Stochastic Models: Interacting Particle Systems



Contents

1 Flow Charts for KMC Simulation Algorithm

STANDARD ALGORITHMS: "PICK A SITE"

1.1 Random Sequential Adsorption with Nearest Neighbor Exclusion

Random sequential adsorption (RSA) with nearest neighbor (NN) exclusion on a linear lattice with N sites with periodic boundary conditions.

Define the quantities:

$$N_{\text{atts}} \equiv \text{Adsorption attempts.}, \qquad N_{\text{ads}} \equiv \text{Adsorbed particles.}, \quad \rightarrow \quad N_{\text{ads}} \leq N_{\text{atts}}$$

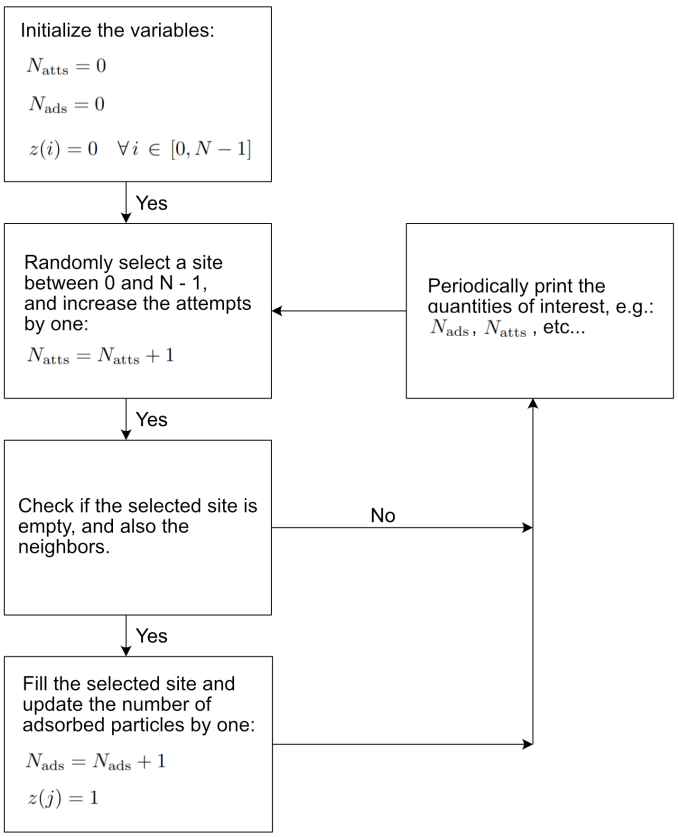
$$\Theta \equiv \frac{N_{\text{ads}}}{N} \equiv \text{Fraction of filled sites, i.e, coverage,} \qquad k \equiv \text{Rate per site,} \qquad R_{\text{tot}} = kN \equiv \text{Total rate,}$$

$$t = \frac{1}{k}\frac{N_{\text{atts}}}{N} \equiv \text{Physical time,} \qquad \delta = \frac{1}{R_{\text{tot}}} \quad \rightarrow \quad \delta = -\frac{\ln(x)}{R_{\text{tot}}} \equiv \text{Physical time per attempt,}$$

where x is a random number in the interval $(0, 1)$.

Let z be the array of particles:

$$z = \underbrace{[0, 0, 0, \ldots, 0]}_{\substack{N \text{ sites with} \\ \text{particles (1)} \\ \text{or empty} \\ \text{sites (0)}}} \equiv \text{Array of particles, intially empty.}$$



1.2 Glauber Spin Flip Dynamics

Glauber spin flip dynamics on a linear lattice with N sites with periodic boundary conditions.

Define the quantities:

$$N_{\text{atts}} \equiv \text{Spin flip attempts}, \qquad \omega_j \equiv \text{Spin flip rate for site } j,$$

$$\omega_i = \begin{cases} \omega_0 & \alpha, \\ \omega_\pm & \alpha \cdot (1 \pm (2\beta J)) \end{cases}, \qquad \omega_{\max} = \max(\omega_+, \omega_-) \equiv \text{Total maximum rate},$$

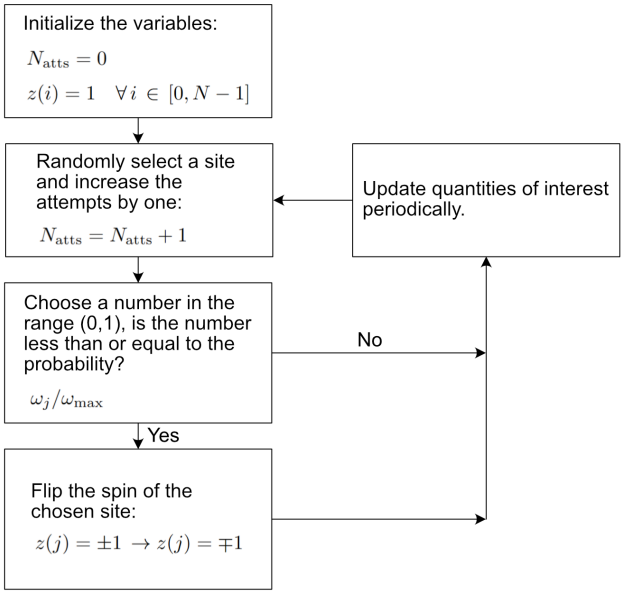
$$R_{\text{tot}}^{\max} = \omega_{\max} N \equiv \text{Total maximum rate}, \qquad t = \frac{1}{\omega_{\max}} \frac{N_{\text{atts}}}{N} \equiv \text{Physical time},$$

$$\delta = \frac{1}{R_{\text{tot}}^{\max}} \quad \rightarrow \quad \delta = -\frac{\ln(x)}{R_{\text{tot}}^{\max}}.$$

Where x is a random number in the interval $(0, 1)$.

Let z be the array of particles:

$$z = \underbrace{[1, 1, 1, \ldots, 1]}_{\substack{N \text{ sites with} \\ \text{spin up } (1) \\ \text{or spin} \\ \text{down } (-1)}} \equiv \text{Array of particles, intially with spins up.}$$



Considerations:

- In both cases, sites are chosen at random, and the system configuration is updated after each successful attempt (to deposit or spin flip).
- RSA simulation is efficient, except for long times where few adsorption sites remain → most adsorption attempts fail; alternative, only track adsorption sites.
- Glauber simulation is efficent unless $\omega_+ >> \omega_0$, ω_- and the system has evolved to large clusters of aligned spins.

1.3 Irreversible Island Formation: Clustering

For clustering, the following processes occur:

- Atoms are deposited at rate F per site.
- Adsorbed atoms with no neighbors hop left or right at rate h.
- Adjacent adatoms are permanently bound together.

For a lattice with N sites, define:

$$F + 2h \equiv \text{Maximum rate per site}, \qquad R_{\text{tot}}^{\max} = N \cdot (F + 2h) \equiv \text{Total maximum rate},$$
$$\delta = \frac{1}{R_{\text{tot}}} \quad \rightarrow \quad \delta = -\frac{\ln(x)}{R_{\text{tot}}} \equiv \text{Physical time per attempt},$$

where x is a random number in the interval $(0, 1)$.

Algorithm:

1. Pick a site randomly.
2. Attempt to deposit with probability $q_{\text{dep}} = F/(F + 2h)$, provided the site is empty.
3. Attempt to hop with probability $q_{\text{hop}} = 2h/(F + 2h)$, provided the site contains a particle and has no neighbors. If the conditions are met, choose the left or right side at random and only jump if empty.

Inefficiency: Typical h/F is in the range $10^6 - 10^9 \rightarrow$ mainly attempt to hop, but density of active isolated particles is very low.

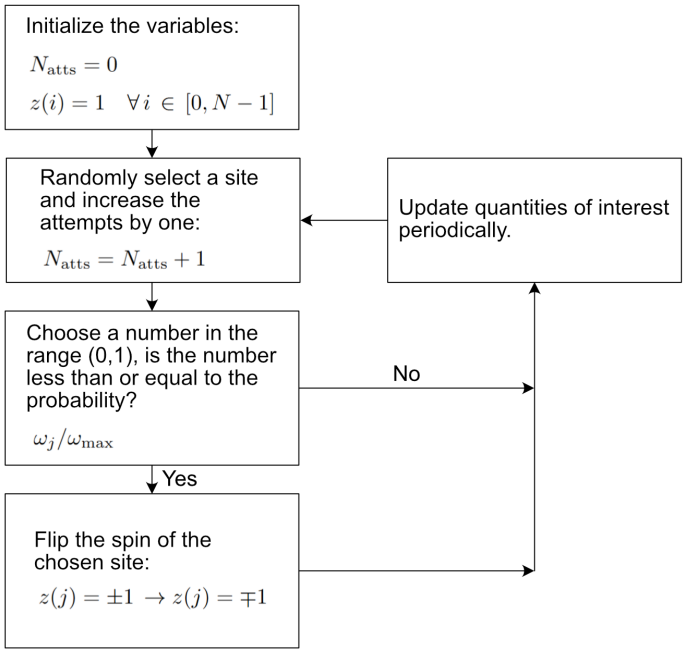
Alternative: Use a "Bortz Algorithm", i.e., tracks all isolated active particles.

Define:

$$N_h \equiv \text{Number of isolated particles},$$
$$R_{\text{dep}} = FN \equiv \text{Maximum total deposition rate}, \qquad R_{\text{hop}} = 2h \cdot N_h,$$
$$P_{\text{dep}} = \frac{R_{\text{dep}}}{R_{\text{dep}} + R_{\text{hop}}} \equiv \text{Deposition probability}, \qquad P_{\text{hop}} = \frac{R_{\text{hop}}}{R_{\text{dep}} + R_{\text{hop}}} \equiv \text{Hopping probability}.$$

Let z be the array of particles:

$$z = \underbrace{[0, 0, 0, \ldots, 0]}_{\substack{N \text{ sites with} \\ \text{particles (1)} \\ \text{or empty} \\ \text{sites (0)}}} \equiv \text{Array of particles, intially empty.}$$


Initialize the variables:
$N_{\text{atts}} = 0$
$z(i) = 1 \quad \forall i \in [0, N-1]$
Randomly select a site and increase the attempts by one:
$N_{\text{atts}} = N_{\text{atts}} + 1$
Update quantities of interest periodically.
Choose a number in the range (0,1), is the number less than or equal to the probability?
$\omega_j / \omega_{\max}$
No
Yes
Flip the spin of the chosen site:
$z(j) = \pm 1 \rightarrow z(j) = \mp 1$

2 Example Code: RSA with Nearest Neighbor Exclusion

This program compiles with the C++ GNU compiler, in Linux, and uses the C++17 standard.

RSA with Nearest Neighbor Exclusion

```
/*
    Random sequential adsorption of particles with nearest neighbor exclusion,
    on a one dimensional periodic lattice.
*/

// Imports.
#include <iostream>
#include <random>
#include <cmath>

// ------------------------------------------------------------------------------
// Constants.
// ------------------------------------------------------------------------------

// Set the lattice length.
const unsigned int latticeLength = 50;

// The total rate of the system.
const long double maximumTime = 5.0l;

// The total rate of the system.
const long double singleCellRate = 1.0l;

// The total rate of the system.
const long double totalRate = singleCellRate * (long double) latticeLength;

// ------------------------------------------------------------------------------
// Lattice variables.
// ------------------------------------------------------------------------------

// Initialize the array with no particles.
bool lattice[latticeLength]{false};

// ------------------------------------------------------------------------------
// Random Generator Variables.
// ------------------------------------------------------------------------------

// Create a pseudo random number generator.
std::random_device rd;
long int seed = rd();
std::mt19937_64 generator(seed);
std::uniform_int_distribution<int> intDistribution(0, latticeLength-1);
std::uniform_real_distribution<long double> longDblDistribution(0.0, 1.0);

// ------------------------------------------------------------------------------
// Other variables.
// ------------------------------------------------------------------------------
```

```
48
49 // Elapsed time; i.e., the physical simulation time.
50 long double elapsedTime = 0.01;
51
52 // Number of attempts that have been made.
53 unsigned int attempts = 0;
54
55 // Number of adsorbed particles.
56 unsigned int adsorbed = 0;
57
58 // Number of triple consecutive sites in the periodic lattice.
59 unsigned int triplets = 0;
60
61 // ------------------------------------------------------------------------
62 // Function specification.
63 // ------------------------------------------------------------------------
64
65 // Determines if the given site can adsorb a particle.
66 bool canAdsorb(int);
67
68 // Generates a lattice index given any unsigned integer index.
69 int generateLatticeIndex(int);
70
71 // Generates an integer in the range [0, latticeLength).
72 int generateLatticeSite();
73
74 // Generates a real number in the interval (0,1).
75 long double generateRealNumber();
76
77 // Attempts to adsobr a particle.
78 void adsorb();
79
80 // Gets the number of triplets available for adsorption.
81 void getTriples();
82
83 // Increments the time by a given random amount.
84 void updateTime();
85
86 // Prints the statistics of the simulation.
87 void printStatistics(bool);
88
89 // ------------------------------------------------------------------------
90 // Main program.
91 // ------------------------------------------------------------------------
92
93 /*
94     Runs the program.
95 */
96 int main()
97 {
98     // Print the header for the statistics.
99     printStatistics(true);
```

```
100
101     while(elapsedTime < maximumTime)
102     {
103         // Update the number of triplets.
104         getTriples();
105 
106         // Attempt to adsorb a particle.
107         adsorb();
108 
109         // Increase the attempt.
110         attempts += 1;
111 
112         updateTime();
113 
114         // Periodically print the statistics.
115         printStatistics(false);
116     }
117 }
118 
119 // ------------------------------------------------------------------------------
120 // Get functions.
121 // ------------------------------------------------------------------------------
122 
123 /*
124     Updates the number of sites that can adsorb particles; i.e., the number of
125     empty sites that do not have neighbors.
126 */
127 void getTriples()
128 {
129     // Always reset the number of triplets.
130     triplets = 0;
131 
132     // Evaluate if the sites can adsorb.
133     for(int i = 0 ; i < latticeLength; i++)
134     {
135         if(canAdsorb(i)) triplets += 1;
136     }
137 }
138 
139 // ------------------------------------------------------------------------------
140 // Generate random values functions.
141 // ------------------------------------------------------------------------------
142 
143 /*
144     Generates a real number in the interval (0,1).
145 
146     :return: A long double random number in the (0,1) interval.
147 */
148 long double generateRealNumber()
149 {
150     // Auxiliary variables.
151     long double randomNumber = longDblDistribution(generator);
```

```
152
153     // Make sure the number is in the correct range.
154     while(randomNumber >= 1.01 || randomNumber <= 0.01)
155     {
156         randomNumber = longDblDistribution(generator);
157     }
158
159     return randomNumber;
160 }
161
162 /*
163     Converts the given index to an index in the lattice.
164
165     :param int index: The index to convert into a lattice index.
166
167     :return: An integer in the range [0, latticeLenght).
168 */
169 int generateLatticeIndex(int index)
170 {
171     // Make sure the index is greater than zero.
172     while(index < 0) index += latticeLength;
173
174     return index % latticeLength;
175 }
176
177 /*
178     Generates an integer in the range [0, latticeLength).
179
180     :return: A random integer in the range [0, latticeLength).
181 */
182 int generateLatticeSite()
183 {
184     // Auxiliary variables.
185     randomNumber = intDistribution(generator);
186
187     return randomNumber;
188 }
189
190 // ------------------------------------------------------------------------------
191 // Printing functions.
192 // ------------------------------------------------------------------------------
193
194 /*
195     Prints the statistics.
196 */
197 void printStatistics(bool printHeader)
198 {
199     // Print the header if needed.
200     if(printHeader)
201     {
202         printf("Elapse Time\t Coverage \tFraction of triplets\n");
203         return;
```

```
204     }
205
206     // Print the statistics peridically.
207     if(!((10 * attempts) % latticeLength == 0)) return;
208
209     // Auxiliary variables.
210     long double numberSites = (long double) latticeLength;
211     long double tripleFraction = (long double) triplets / numberSites;
212     long double coverage = (long double) adsorbed / numberSites;
213
214     // Print the statistics.
215     printf("%0.11Lf\t%0.11Lf\t%0.11Lf\n", elapsedTime, coverage, tripleFraction);
216 }
217
218 // ------------------------------------------------------------------------------
219 // Time related functions.
220 // ------------------------------------------------------------------------------
221
222 /*
223     Increments the time by a given amount.
224 */
225 void updateTime()
226 {
227     // Auxiliary variables.
228     long double sites = (long double) latticeLength;
229
230     // Update the elapsed time of the system.
231     elapsedTime = (long double) attempts / sites;
232 }
233
234 // ------------------------------------------------------------------------------
235 // Validation functions.
236 // ------------------------------------------------------------------------------
237
238 /*
239     Determines if a given can adsorb a particle; i.e., the site is empty,
240     along with its neighbors, remembering that the lattice is periodic.
241
242     :param int index: An index within the lattice.
243
244     :return: A boolean flag indicating whether the lattice site at the given
245      index is empty, along with its neighbors. True, if the lattice site at
246      the given index is empty, along with its neighbors; False, otherwise.
247 */
248 bool canAdsorb(int index)
249 {
250     // Auxiliary variables.
251     int site0 = generateLatticeIndex(index - 1);
252     int site1 = generateLatticeIndex(index);
253     int site2 = generateLatticeIndex(index + 1);
254
255     return !(lattice[site0] || lattice[site1] || lattice[site2]);
```

```
}

// ------------------------------------------------------------------------------
// Other functions.
// ------------------------------------------------------------------------------

/*
    Attempts to adsorb a particle and updates the number of adsorbed particles.
*/
void adsorb()
{
    // Auxiliary variables.
    int site = 0;

    // Pick a random site within the lattice.
    site = intDistribution(generator);

    // If the particle cannot be adsorbed, no need to continue.
    if(!canAdsorb(site)) return;

    // Particle is adsorbed.
    adsorbed += 1;

    // Update the particle array.
    lattice[site] = true;
}
```