Simulated Annealing Monte Carlo and Genetic Algorithms


A protein with just two types of amino acid, hydrophobic (black) and hydrophilic (white) occupies the sites of a 2-D square lattice. The figure shows two configurations of the 20-residue protein with sequence BWBWWBWBWBWWBBWBBW.
The energy function is very simple

\[ V = - \sum_{\langle B_i B_j \rangle} \Delta(r_i - r_j), \quad \Delta(r) = \begin{cases} 1 & \text{if } |r| = 1 \text{ and bond is not covalent} \\ 0 & \text{otherwise} \end{cases} \]

The configuration on the left has energy $-4$. A trial move selects the arrowed site and rotates $180^\circ$ to give the folded configuration on the right with minimum energy $-9$.

The model was simulated using two types of algorithms:

- Genetic Algorithm: A population of 200 configurations was evolved over 300 generations
- Simulated Annealing: The simulations involved the same number of initial configurations as the GA algorithm. The MC simulation used the same number of energy evaluations as the GA. The “Long MC” and “Multiple MC” simulations were run much longer to improve the results.

They found that the GA algorithms performed much better than the MC.
The basic idea to find the absolute minimum energy folded configuration in a complicated landscape of mountains and valleys with many local minima is to start the system at a finite temperature $T$ large enough to allow it explore the landscape. Random moves are made with acceptance probability determined by $T$-dependent Boltzmann factor, and the system is gradually cooled as its energy tends towards the global minimum.

Eight sequences were tested. For each, the GA method is compared with several MC variants. We check a MC that uses as many energy evaluations as the GA, a much longer MC and a multiple MC running 100 different simulations. The GA results are very superior to those from the comparable MC which failed to find the optimal solution in all cases, and are also better than the longer MC runs. In the 36, 48 and 60-residue long sequences, the lowest energy conformation was found by the GA and was not found in any of the MCs. All the methods failed to find the lowest energy conformation for the longest sequence.

<table>
<thead>
<tr>
<th>Length</th>
<th>Optimal energy</th>
<th>GA</th>
<th>MC</th>
<th>Long MC</th>
<th>Multiple MC</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>-9</td>
<td>-9 (30,492)</td>
<td>-8</td>
<td>-9 (292,443)</td>
<td>-9 (41%)</td>
</tr>
<tr>
<td>24</td>
<td>-9</td>
<td>-9 (30,491)</td>
<td>-8</td>
<td>-9 (2,492,221)</td>
<td>-9 (19%)</td>
</tr>
<tr>
<td>25</td>
<td>-8</td>
<td>-8 (20,400)</td>
<td>-7</td>
<td>-8 (2,694,572)</td>
<td>-7 (100%)</td>
</tr>
<tr>
<td>36</td>
<td>-14</td>
<td>-14 (301,339)</td>
<td>-12</td>
<td>-13 (6,557,189)</td>
<td>-13 (5%)</td>
</tr>
<tr>
<td>48</td>
<td>-22</td>
<td>-22 (126,547)</td>
<td>-18</td>
<td>-20 (9,201,755)</td>
<td>-19 (3%)</td>
</tr>
<tr>
<td>50</td>
<td>-21</td>
<td>-21 (592,887)</td>
<td>-19</td>
<td>-21 (15,151,203)</td>
<td>-20 (1%)</td>
</tr>
<tr>
<td>60</td>
<td>-34</td>
<td>-34 (208,781)</td>
<td>-31</td>
<td>-33 (8,362,338)</td>
<td>-32 (7%)</td>
</tr>
<tr>
<td>64</td>
<td>-42</td>
<td>-37 (187,303)</td>
<td>-31</td>
<td>-35 (7,848,952)</td>
<td>-32 (2%)</td>
</tr>
</tbody>
</table>
Simulated Annealing Monte Carlo

The Monte Carlo algorithm of Metropolis et al. *J. Chem. Phys.* 21, 1087 (1953) is modified using a Simulated annealing schedule, as outlined in Section 3 of the article.

A stopping criterion is chosen, which can be some maximum number of steps, or a target minimum energy.

1. Start from a random coil configuration, for example an extended straight line.
2. Make a trial configuration change, for example by selecting a monomer at random and rotating the C-terminal part of the chain around this monomer.
3. If the new configuration is self-avoiding, compute the change in energy $\Delta E$ in this trial move:
   - If $\Delta E < 0$ accept the change
   - Otherwise accept conditionally using the Metropolis criterion
     \[ w = \exp \left[ -\frac{\Delta E}{c_k} \right] > u, \quad c_k \equiv k_B T \]
     where $u$ is a random uniform deviate. monomer, as was done in the figure.
4. If the stop criterion is not met, repeat at step 2., gradually reducing $T$ according to an annealing schedule. For example, with a stop criterion of 50,000,000 steps, the temperature is reduced from 2 to 0.15 by reducing $c_k \rightarrow c_{k+1} = \alpha c_k$ with $\alpha = 0.99$ every 200,000 steps.
Genetic Algorithms

Genetic algorithms try to model evolution by natural selection. In nature the genetic code is stored in DNA molecules as sequences of bases: adenine (A) which pairs with thymine (T), and cytosine (C) which pairs with guanine (G).

The analog of DNA in a digital genetic algorithm is a sequence of binary digits (0) and (1).

In nature, the genetic code describes a genotype, which is translated into an organism, a phenotype, by the process of cell division.

Digital genetic algorithms can be used to solve a problem, such as finding the global minimum of a complicated energy landscape. A famous example is the Traveling Salesman Problem.

The phenotype in a genetic algorithm is some state of the model: strings of binary digits are mapped to the states of the model to be solved.

Evolution by natural selection is driven in part by changes to the genetic code:

**Mutations:** Random changes can occur, for example caused by radioactivity or cosmic rays damaging a DNA molecule. Mutations of the digital genotype can be modeled by choosing a random bit in the string and changing it $1 \rightarrow 0$ or $0 \rightarrow 1$.

**Recombination or Crossover:** During sexual reproduction the offspring inherit DNA from each of the
parents. This can be simulated by taking two strings and exchanging two substrings.

**Survival of the Fittest:** There is some criterion of *fitness* such that when mutations or recombinations take place, the mutants or offspring either survive and reproduce or die out.

These simple ingredients can be used to construct a very wide variety of genetic algorithms. A simple algorithm which can be applied to an energy landscape problem is illustrated by the random Ising model:

\[
E = - \sum_{\langle ij \rangle} T_{ij} s_i s_j ,
\]

where \( s_i = \pm 1 \) are Ising spins, and the coupling constants \( T_{ij} \) between nearest neighbors are chosen randomly to be \( \pm 1 \). This is a model of a *spin glass* which has a very complicated energy landscape with numerous local minima.

What is a genotype for this model? Suppose we have a 2-D lattice of spins with \( i, j = 0, 2, \ldots, (L - 1) \), then we can order the spins linearly using the formula \( n = iL + j = 0, 1, \ldots, (L^2 - 1) \) for example. A configuration is of spins is mapped to a genotype of \( L^2 \) bits by setting the bit with index \( n \) to 0 or 1 if \( s_{ij} = \pm 1 \).

Since we are seeking the global energy minimum, the *fitness* of a particular genotype can be taken to be \( 2L^2 - E \), since the minimum and maximum possible values for the energy are \( \pm 2L^2 \) for a 2-D square lattice and periodic boundary conditions. (Recall that the number of bonds is then twice the number of spins.)

The following is one possible evolution protocol:
Start with a population of a fixed number \( N_0 \) of strings initialized in some way, for example by setting the string bits randomly.

Repeat the following “generations”:

- Allow some number of mutations. For example, choose 20% of the strings at random, and mutate a random bit (flip a random spin) in each string.
- Choose some number of pairs of strings at random and have them “reproduce” as follows: each pair produces two offspring which differ from the parents by exchange of a randomly chosen substring.
- The size of the population has now increased from \( N_0 \) to \( N \) due to reproduction, and the parents and children are competing for the same limited natural resources. Select \( N_0 \) fittest survivors as follows:
  - Construct a cumulative histogram

\[
H_k = \sum_{i=1}^{k} (2L^2 - E_i) , \quad k = 1, 2, \ldots, N ,
\]

where \( k \) labels the strings in the population.

- Repeat \( N_0 \) times:
  - Choose a random \( H \) between 0 and the maximum \( H_N \).
  - Select the smallest \( k \) such that \( H_k > H \).

After many generations the population should converge to the global energy minimum configuration!
Genetic Algorithm for the Unger-Moult Model

Figure 2. The genetic algorithm. The process starts with a population of fully extended structures. Each structure undergoes a MC stage followed by a crossover stage. In the crossover stage, pairs of structures are randomly (based on their energies) cut and pasted. In this example the cutpoint was randomly chosen to be after residue 14. Joining the first 14 residues of (A) with the last 6 residues of (B) and applying a randomly chosen 270° rotation at the joint achieves the compact structure in (C). In this case, the energy value of the hybrid (C) is $-9$, lower than the energies $-5$ and $-2$ of its "parents". The hybrid is always accepted if its energy is lower than the averaged energies of its parents, or non-deterministically accepted according to its energy increase.
■ The binary strings are taken to be proteins, with hydrophobic and hydrophilic amino acid residues assigned binary 0 or 1.

■ The process starts with $N$ extended linear chains.

■ Each mutation is a single Metropolis Monte Carlo step subject to the Metropolis acceptance criterion at temperature $T$.

■ A crossover operation is performed by selecting a protein depending on its energy $E$ with probability

$$p(S_i) = \frac{E_i}{\sum_{j=1}^{N} E_j}$$

Thus the lower energy configurations have a higher chance of being selected.

■ For a pair of selected proteins a random point is chosen along the sequence, and the N-terminus portion of the first protein and connected to the C-terminus portion of the second protein.

  • There are three ways to do this, with angles of $0^\circ$, $90^\circ$, or $270^\circ$, which are tested in random order to find one that is self-avoiding.

  • If none of the three ways is valid, another pair of structures is selected.

■ The energy of the valid crossover structure is compared to the average of the energies of its two parents and conditionally accepted based on the Metropolis Boltzmann ratio criterion

$$w = \exp \left[ -\frac{(E - \langle E \rangle)}{c_k} \right] > u , \quad c_k \equiv k_B T$$
The crossover operation is repeated until $N - 1$ new accepted hybrid structures have been constructed.

There are numerous resources on genetic algorithms on the web, see for example Wikipedia Genetic Algorithm.

**Python Simulated Annealing Algorithm Code**

```python
import copy
import math
import random

def new_extended_coil(dna):
    """ input: dna sequence of 'B' (hydrophopic) and 'W' (hydrophilic) residues
    output: extended linear conformation
    ""
    L = len(dna)
    assert L == dna.count('B') + dna.count('W')
    return [ [0, i] for i in range(L) ]

def E(s, dna):
```

```python```
input: conformation, dna sequence
output: E = -(number of non-covalent BB nearest neighbors)

L = len(s)
n_BB = 0
for i in range(L - 2):
    for j in range(i + 2, L):
        if dna[i] == 'B' and dna[j] == 'B':
            dx = s[i][0] - s[j][0]
            dy = s[i][1] - s[j][1]
            if abs(dx) + abs(dy) == 1:
                n_BB += 1

return -n_BB

def MMC_mutation_step(s, dna, c_k):
    input: conformation, temperature
    output: changed conformation, whether mutation was self-avoiding

    L = len(s)
    s_trial = copy.deepcopy(s)  # for trial conformation
    i = random.randrange(1, L - 1)  # choose a random interior amino acid
    # rotation matrices for 90, 180, and 270 degrees
rotation_matrices = 
    [ [ [0, -1], [1, 0]],
     [ [-1, 0], [0, -1]],
     [ [ 0, 1], [-1, 0]] ]

rot_mat = rotation_matrices[random.randrange(3)]
# rotate the C-terminal (j < i) portion about amino acid i
for j in range(i):
    dr = [ s[j][k] - s[i][k] for k in range(2) ]
    for k in range(2):
        s_trial[j][k] = s[i][k]
        for l in range(2):
            s_trial[j][k] += rot_mat[k][l] * dr[l]
# check whether this is a valid self-avoiding conformation
for j in range(i):
    for k in range(i + 1, len(s)):
        if s_trial[j] == s[k]:
            return False
# no overlaps - update conformation using Metropolis et al. algorithm
dE = E(s_trial, dna) - E(s, dna)
w = math.exp(- dE / c_k)
if random.random() < w:
    for j in range(i):
        for k in range(2):
for k in range(2):
    s[j][k] = s_trial[j][k]
return True

def simulated_annealing_run(s, dna, mmc_steps):
    """ input: conformation s, dna, number of Metropolis Monte Carlo steps
    output: lowest energy, step found, fraction self-avoiding conformations
    """
    c_k = 2.0  # initial temperature
    alpha = 0.99  # cooling factor
    valid_confs = 0
    E_min = E(s, dna)
    step_min = 0
    for step in range(1, mmc_steps + 1):
        if MMC_mutation_step(s, dna, c_k):
            valid_confs += 1
            E_step = E(s, dna)
            if E_step < E_min:
                E_min = E_step
                step_min = step
                print(" E =", E_min, "at step", step)
        if step % 200000 == 0 and c_k > 0.15:
            # code here
c_k *= 0.99
valid_fraction = valid_confs / float(mmc_steps)
return E_min, step_min, valid_fraction

print(" Folding of a Simple Model Protein on a Square Lattice")
print(" using Simulated Annealing and Genetic Algorithms")
print(" ------------------------------------------------------")

# sample protein with 20 amino acids
dna = "BWBWWBBCBWWBBWWBBWWB"
p20 = new_extended_coil(dna)
L = len(p20)
print(" Initialized sequence", dna, "L =", L, "fully extended")
mmc_steps = 50 * 10**6
print(" Simulated annealing with", mmc_steps, "steps")
E_min, step_min, valid_confs = simulated_annealing_run(p20, dna, mmc_steps)
print(" E_min =", E_min, "at step", step_min,
      "fraction of valid configurations =", valid_confs)