Simulated annealing/Metropolis and genetic optimization

Eugeniy E. Mikhailov

The College of William & Mary

Lecture 18

Nature's way ti find a minimum energy

- We see that probing full space permitted by combinatorics is not practical even for a reasonably small size problem.

Nature’s way ti find a minimum energy

- We see that probing full space permitted by combinatorics is not practical even for a reasonably small size problem.
- However nature seems to handle the problem of the energy minimization without any trouble.

Nature’s way ti find a minimum energy

- We see that probing full space permitted by combinatorics is not practical even for a reasonably small size problem.
- However nature seems to handle the problem of the energy minimization without any trouble.
- For example, if you heat up a piece of metal and then slowly cool it i.e. anneal, then the system will reach the minimum energy state.

Simulated annealing/Metropolis algorithm

Metropolis and coworker suggested in 1953 the following heuristic algorithm based on this observation, and the Boltzmann energy distribution law.

1. set the temperature to a high value so kT is larger then typical energy (merit) function fluctuation.
    - This requires some experiments if you do not know this a priori.
2. assign a state $\vec{x}$ and calculate the energy at this point E.
3. change somehow old $\vec{x}$ to generate a new one $\vec{x}_{new}$
    - $\vec{x}_{new}$ should be somewhat close/related to old optimum $\vec{x}$
4. calculate the energy at new point $E_{new} = E(\vec{x})$
5. if $E_{new} < E$ then $x = x_{new}$ and $E = E_{new}$
    - i.e. we move to new point of the lower energy
6. otherwise move to the new point with probability $p = exp(-(E_{new} - E)/kT)$
    - this resembles the Boltzmann energy distribution probability
7. decrease the temperature a bit i.e. keep annealing
8. repeat from the step 3 for a given number of cycles
9. $\vec{x}$ will hold the local optimal solution

Simulated annealing/Metropolis algorithm facts

In finite time (limited number of cycles) the algorithm guaranteed to find only local minimum.

Simulated annealing/Metropolis algorithm facts

In finite time (limited number of cycles) the algorithm guaranteed to find only local minimum.
There is a theorem which states:

The probability to find the best solution goes to 1, as we run algorithm for a longer time with a slow rate of cooling.

Simulated annealing/Metropolis algorithm facts

In finite time (limited number of cycles) the algorithm guaranteed to find only local minimum.
There is a theorem which states:

The probability to find the best solution goes to 1, as we run algorithm for a longer time with a slow rate of cooling.

Unfortunately, this theorem is of no use since it does not give a recipe of how long to run the algorithm. It is even suggested that it will need more cycles than the brute force combinatorial search.

Simulated annealing/Metropolis algorithm facts

In finite time (limited number of cycles) the algorithm guaranteed to find only local minimum.
There is a theorem which states:

The probability to find the best solution goes to 1, as we run algorithm for a longer time with a slow rate of cooling.

Unfortunately, this theorem is of no use since it does not give a recipe of how long to run the algorithm. It is even suggested that it will need more cycles than the brute force combinatorial search.
However, in practice very good solutions can be found in quite short time with quite small number of cycles.

Simulated annealing/Metropolis algorithm facts

In finite time (limited number of cycles) the algorithm guaranteed to find only local minimum.
There is a theorem which states:

The probability to find the best solution goes to 1, as we run algorithm for a longer time with a slow rate of cooling.

Unfortunately, this theorem is of no use since it does not give a recipe of how long to run the algorithm. It is even suggested that it will need more cycles than the brute force combinatorial search.
However, in practice very good solutions can be found in quite short time with quite small number of cycles.
The Metropolis algorithm method is not limited to the discrete space problems, and can be used for the problems accepting real values of the $\vec{x}$ components.

Simulated annealing/Metropolis algorithm facts

In finite time (limited number of cycles) the algorithm guaranteed to find only local minimum.
There is a theorem which states:

The probability to find the best solution goes to 1, as we run algorithm for a longer time with a slow rate of cooling.

Unfortunately, this theorem is of no use since it does not give a recipe of how long to run the algorithm. It is even suggested that it will need more cycles than the brute force combinatorial search.
However, in practice very good solutions can be found in quite short time with quite small number of cycles.
The Metropolis algorithm method is not limited to the discrete space problems, and can be used for the problems accepting real values of the $\vec{x}$ components.

- the main challenge is to find a good way to choose new $\vec{x}$ to probe.

Backpack problem with Metropolis algorithm

- The main challenge is to find a good routine to generate new candidate for the $\vec{x}_{new}$. We do not want to randomly jump to an arbitrary position of problem space.
- Recall that $\vec{x}$ generally looks like $[0, 1, 1, 0, 1, \cdots, 0, 1, 1]$ so lets just randomly toggle/mutate some choices/bits
    - note that such a random mutation could lead to overfilled backpack
- The rest is quite straight forward, as long as we remember, that we are looking for the maximum value in the backpack, while Metropolis algorithm is designed for merit function minimization. So we choose our merit function to be negative value of all items in the backpack. Also we need to add a big penalty for the case of the overfilled backpack.
- See the realization of the algorithm in the `backpack_metropolis.m` file.
- it will find quite good solution for the “30 items to choose” problem within a second instead of 13 hours of combinatorial search.

Genetic algorithm

Idea is taken from nature which usually able to find optimal solution via natural selection.
This algorithm has many modification but the main idea is the following

1. Generate a population (set of $\vec{x}$)
   - how big is up to you and your resources
2. Find the fitness (merit) function for each member of the population
3. Remove from the pool all but the most fitted
   - how many should stay is up to heuristic tweaks
4. from the most fitted (parents) breed new population (children) to the size of the original population
5. repeat several times starting from step 2
6. Chose the fittest member of your population to be the solution

How to generate children from parents

As usual the most important question is generation of an new $\vec{x}$ from the older ones.
Let's use recipe provided by nature. Let's refer to $\vec{x}$ as chromosome or genome.

1. chose two parents randomly
2. crossover/recombine parents chromosomes i.e. take randomly gens ($\vec{x}$ components) from either parent and assign to new child chromosome
3. mutate (change) randomly some gens

Some algorithm modifications allow parents to be in the new cycle of selection, some eliminate them (to hope away from local minima).

What to expect with genetic algorithm

To find a good solution you need large populations. Since this lets to explore larger parameter space. Think microbes vs human. But this in turn leads to longer computational time for every selection cycle.
Algorithm is not guaranteed to find global optimum in finite time.
Nice part about algorithm though that it suits the parallel computation paradigm: you can evaluate fitness of each child on different CPU and then compare their fitness.