Simulated Annealing-The Implementation and Performance Analysis

Luezhen Yuan(袁略真)*

Abstract: Simulated annealing is a metaheuristic algorithm used to solve combinatorial optimization problem. However, the choose of annealing schedule is not discussed in many textbooks. This article implement the simulated annealing algorithm in C++ in a generic style, and test the program on the Traveling Salesman Problem(TSP) for 4 annealing plans and each one runs 10 configurations. The results show that a slow temperature decreasing plan is likely to find a good optimization solution, however the time cost is too big. Setting the r =0.99999 and running 10 configurations can give a good solution and the time cost is affordable.

Keywords: Simulated Annealing; Annealing schedule; Combinatorial optimization; Traveling salesman problem; Generic programming; Object-oriented programming

1. Introduction

The simulated annealing is a metaheuristic method to find the global minimum of a combinatorial optimization problem[3]. The problem is often an NP-complete problem, which means finding the exact solution will cost polynomial time. This method is a Markov Chain Monte Carlo(MCMC) based method, and allows worse solutions in the Markov Chain. The method is also featured by its use of the decreasing temperature, which is an efficient improvement of the Metropolis procedure.

The simulated annealing is introduced by S. Kirkpatrick[1] in 1983. The most procedure in the simulated annealing algorithm is the same as Metropolis's algorithm, but the use of deceasing temperature. Metropolis's algorithm is a Markov chain Monte Carlo sampling method that used for simulation used in statistical mechanics. The simulated annealing, however, is an heuristic method used to solve combinatorial optimization problem.

This article implements the simulated annealing in C++ in a generic style. The generic programming is an efficient method to separate algorithms from data structures. In C++, these methodology is characterized by the use of parametric polymorphism through the use of templates. In the generic programming, an important term is the *concept*, which is a set of requirements that describe the interface and semantic behaviour of an unknown data structure used in a generic algorithm. This article use a abstract base class to store the concept.

2. Simulated Annealing and Implementation

2.1 Procedure in Simulated Annealing

The simulated annealing method simulate the annealing procedure, which starts from a high temperature, and then slowly decreases the temperature of a system. The slowly decreasing temperature is a central feature of the method. When the temperature is high, the system can almost free from changing to the other states. As the temperature decrease, it's more likely to change to a lower energy state than climb up the energy mountain. Climbing up is valuable for jumping out of a local minimum valley. When the temperature is decreased closely to zero, the system is reluctant from climbing up the mountain. And now, the state is likely to be a global minimum.

*Student ID: 3130103964. Major: Bioinformatics. Email: 3130103964@zju.edu.cn

2.2 Terms in Simulated annealing

The simulated annealing is a Markov Chain Monte Carlo(MCMC) method, and many terms mentioned following are the same as other MCMC.

The simulated annealing procedure is a Markov Chain because the *neighbours of a state* and the *transition probability* only depend on the state. The neighbours of a current state is a subset of all possible states which can be *moved* to from the current state. These moves usually defined under simple rules, and can be seen as *a small random displacement*.

The states of a system can be described quantitatively using the *energy*. The energy, in the optimization problem, is the target function that you want to minimize. Two states can be compared using the Boltzmann probability factor, $\pi(s) = exp(-E(s)/k_BT)$[1], where s is a state of the system, and $E(s)$ is the energy of this state. Under the condition of the existence of stationary distribution in Markov Chain, the transition probability should follows, $\pi(s)P(s \to s') = \pi(s')P(s' \to s)$. Metropolis choose the following transition probability:

$$P(s \to s') = \begin{cases} e^{-\Delta E/k_BT} & , \Delta E > 0 \\ 1 & , \Delta E \leqslant 0 \end{cases} \tag{1}$$

The terms mentioned above, actually, is the Metropolis algorithm. The feature of the simulated annealing algorithm, which different from the Metropolis's one, is the decreasing temperature. An *annealing schedule* is a predefined rule that quantify how the temperature will decrease. For example, $T(t_{n+1}) = rT(t_n)$, where n is the simulation time step, T is the temperature at that step, r quantify how fast the temperature will decrease.

2.3 Implementation Using the Generic Programming Method

Input: *start_state*, *max_iteration*, *r*
Output: *fineal_state*
temperature ← 100;
state ← *start_state*;
for *t* ← 0 **to** *max_iteration* **do**
  energy$_1$ ← current *stage* energy;
  state_tmp ← a neighbour of the *state*;
  energy$_2$ ← *stage_tmp* energy;
  transition_probability ← $e^{-\Delta E/k_BT}$;
  p ←random number generator[0,1);
  if *p* < *transition_probability* **then**
    state ← *state_tmp*;
  end
  else
    drop *state_tmp*;
  end
  temperature ← *temperature* ∗ *r*;
end
return *state*;

Algorithm 1: Simulated Annealing Algorithm

To implement a generic simulated annealing algorithm, I design the header of the template function *Simulated_Annealing*:

```
template <class State>
State Simulated_Annealing(Simulated_Annealing_Problem<State>* pSAP,int iteration=100000,double
    downratio=0.999, string file_name="Simulated_Path.txt",void m_free(State pst)=my_free,
  double m_next(double current_temperature,double downratio)=my_next,double Acceptance(
  Simulated_Annealing_Problem<State>* pSAP,State st1, State st2, double temperature)=
  my_Acceptance );
```

Here, the *Simulated_Annealing_Problem* is an abstract base class, storing the concept asked in the template function *Simulated_Annealing*. The concept is some member functions used in the generic algorithm.

```
class Simulated_Annealing_Problem{
public:
virtual State initial_state()=0;
virtual double energy(State state)=0;//[0 1]
virtual State neighbour(State state)=0;
};
```

3. Test Performance on the Traveling Salesman Problem(TSP)

3.1 Introduction to the TSP

The TSP is a combinatorial optimization problem, and also belongs to the class of NP-complete problem, which need polynomial time to find the exact solution. TSP tells this story: A salesman want to find a shortest route to travel N given cities.

3.2 Derived Class for TSP

To solve the TSP using the generic simulated annealing algorithm, I design a derived class of the *Simulated_Annealing_Problem*(an abstract base class, mentioned in the previous section.) for TSP.

```
class TSP:public Simulated_Annealing_Problem<int*>{
double* m_distance;
double max_length;
public:
int m_size;
TSP(string data_file="TSPdata.txt");//Input file. The coordinate of each city.
int* initial_state();
double energy(int* state);//[0 1]
int* neighbour(int* state);
double& distance(int i, int j);//A[i][j]=distance[(2*m_size-i)(i-1)/2+j-i];
};
```

3.3 Test Cases and Results

Figure 1 shows the simulated annealing procedure under r =0.999.

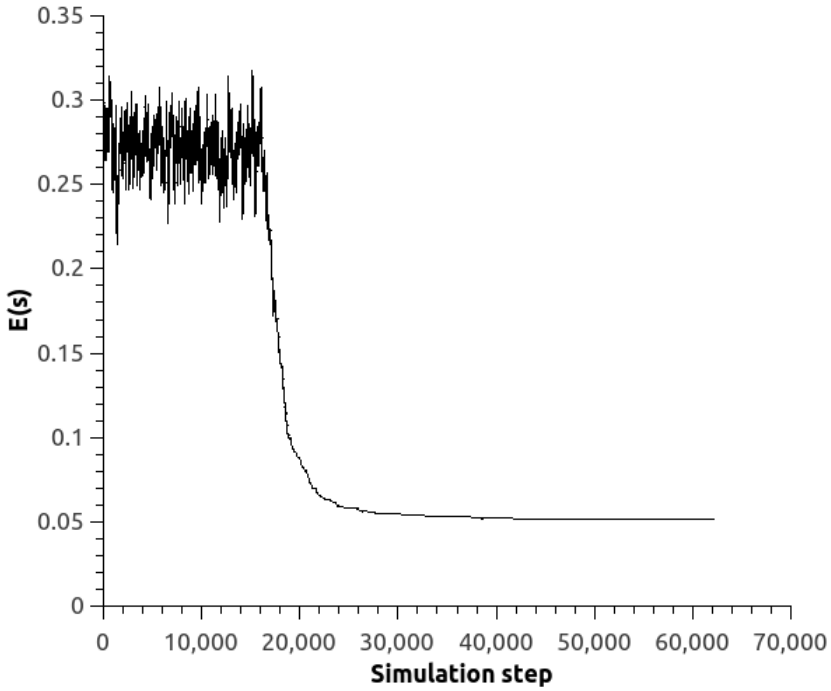

Figure 1. Common simulated annealing procedure. The simulation is performed under r =0.999.

To test the performance of the generic algorithm and study the annealing schedule, I change the value of the important parameter r. Let r=0.999,0.9999,0.99999, and 0.999999. Each runs 10 configurations(10 times). The figure 2,3, and 4 shows the results. The data of 100 cities and one starting city comes from [4].

Figure 2 shows the best solution for each r in 10 configurations. Figure 3 shows the solution distribution on on each r. Figure 4 shows the time cost on each r. The time cost is related to the iteration times in one run. However, when the system is trapped in a state for a long time(20% of the *iteration*), the iteration will stop, even if the max iteration time(here *iteration*) isn't reached. Here, the iteration time in one run is calculated by the simple formula(100 is the start temperature.):

$$t_{low} = 100 * r^{iteration}$$

$$iteration = \frac{log(t_{low}/100)}{log(r)}$$

To compare the time costs on each configurations, I set the $t_{low} = 10^{-25}$.

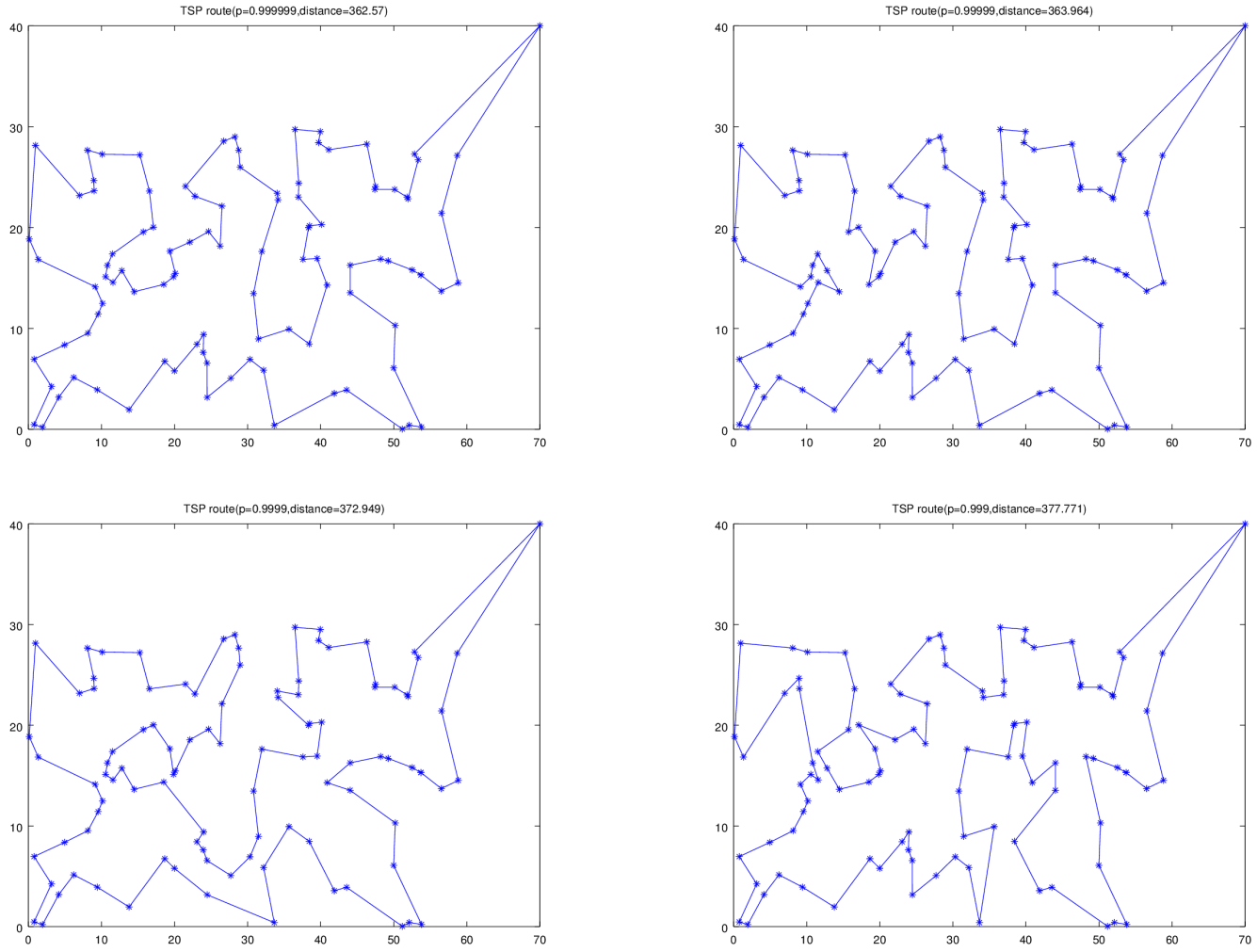

Figure 2. TSP routes on four rs. Top left: r=0.999999, distance=362.57; Top right: r=0.99999, distance=363.964; Bottom left: r=0.9999, distance=372.949; r=0.999, distance=377.771;

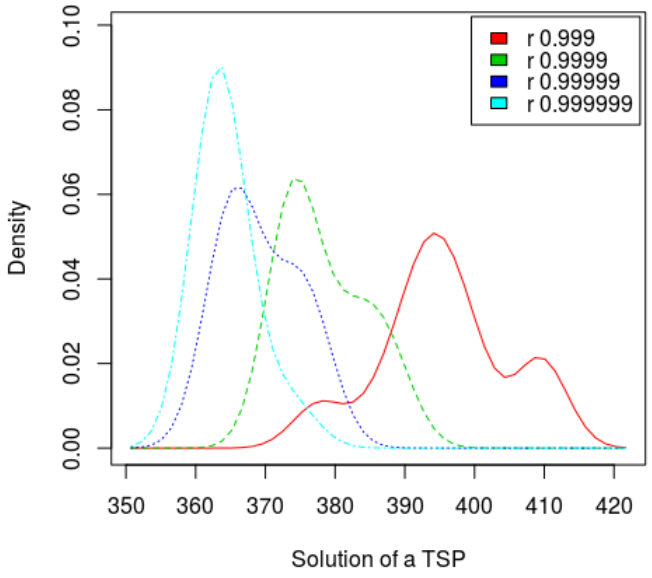

Figure 3. SA performance on finding the optimization.

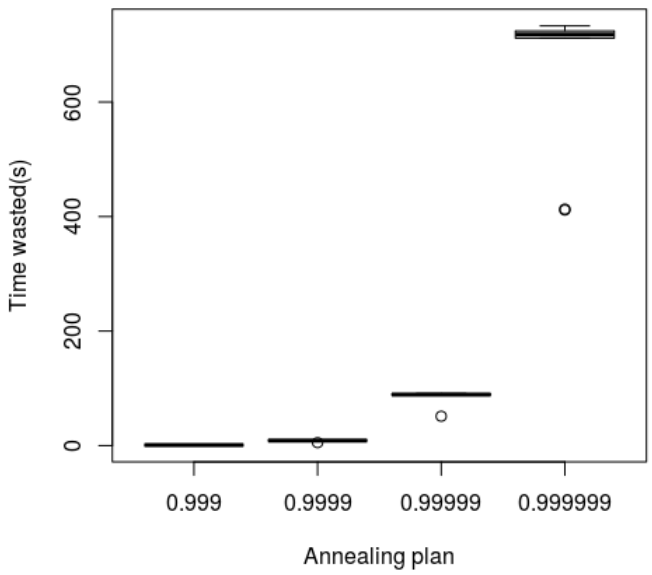

Figure 4. SA performance on finding the optimization.

Figure 3 shows that with the r increase(the decrease of temperature slower), the solution distribution shift left.

Figure 4 shows that with the r increase, the time cost grows extremely fast.

4. Discussions and Conclusions

The simulated annealing is a useful method for wide spread problems. In [2, 3], they conclude that if a problem follows the following conditions, they can be solved using the method:

1. Solution can be quantified by number.(Here, the energy function.)
2. An initial solution can be easily computed.(For example, generate a TSP route randomly)
3. There are simple adjustment rules that locally change a solution.(The solution means the state of a system.)
4. Every solution can be turned into every other solution by the adjustment rules.
5. Often used when the search space is discrete.

However, to solve these problems, we should set same parameters in the algorithm. From figure 3 and 4, the paradox of r appears. High r is good for finding the optimization solution, however the time cost is too big to afford. If we set r=0.99999, the time cost is around 150(s), however the solution distribution is not a single peak, which means we should run many configurations and find the best solution. Figure 2 top right shows that choosing r=0.99999 and run 10 configurations is likely to appear a good solution. If we set r =0.9999 or 0.999, even if we run many configurations, the solution is not good(See figure 2 bottom panel, and figure 3).

In conclusion, r=0.99999 and running 10 configurations is a good choice of the simulated annealing method, and the time cost is affordable..

5. Acknowledgement

Prof. Yizhi Tan(谈之奕)(School of Mathematical Sciences) is the course director of *Applied Operational Research(应用运筹学)* in Zhejiang University, Hangzhou, China. In the course, I learnt linear programming, integer programming, graph theory, combinatorial optimization(NP-complete problem, TSP, and so on), scheduling, and game theory. In computational biology, bioinformatics, and computational systems biology, there are some problems that have been solving using the methods mentioned in this course. In metabolic network analysis, the flux balance analysis is a linear programming problem. In molecular phylogeny, the branch-and-bound method also used in constructing some phylogenetic trees. Thanks to Prof. Yizhi Tan.

Prof. Dafang Zheng(郑大昉)(Department of Physics) is the course director of *Computational Physics(计算物理学)* in Zhejiang University, Hangzhou, China. The Monte Carlo simulation, Metropolis Markov Chain Monte Carlo Simulation is lectured in this course. Thanks to Prof. Dafang Zheng.

6. Supplement

The source code, this document, the figures, and program's output data are accessible in.

References

[1] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," Science, vol. 220, no. 4598, pp. 671–680, May 1983.

[2] B. Vöcking, H. Alt, M. Dietzfelbinger, R. Reischuk, C. Scheideler, H. Vollmer, and D. Wagner, Eds., Algorithms Unplugged. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[3] Simulated annealing, Wikipedia, the free encyclopedia. 07-Apr-2016.

[4] 司守奎, 数学建模算法与应用. 北京: 国防工业出版社, 2011.