Optimization

Simply speaking, an optimization problem is the problem of finding the best solution from all feasible solutions.

Generally, an optimization problem consists of a set of variables, an target function that maps total assignments to real numbers and an optimality criterion, which aims to find a total assignment that is optimal according to the optimality criterion.

As one of the typical examples of optimization problem, the traveling salesman problem is to find the cheapest way of visiting all of the cities and returning to your starting point by given a collection of cities and the cost of travel between each pair of them. As the stated above, we have a set of travelling routes as the variables, and the summery mileage of those routes that can be seen as the target function needing to be minimized, and a restraints requirement of returning to the starting point as the final step and not repeating the points even once.

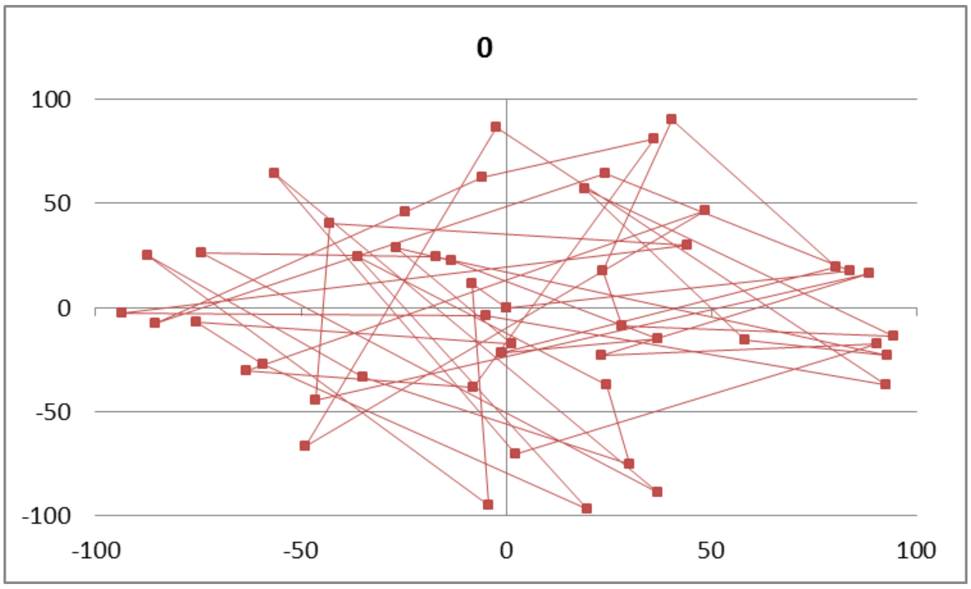

Fig.1 Initial calculation result of the travelling salesman problem in steepest descent

In order to calculate the travelling salesmen problem, various approaches are put forward, such as steepest descent, simulated annealing and genetic algorithm. In this report, I would like to focus on the former two approaches.

Steepest descent, also named as Gradient descent, is a way to minimize the target function by updating the parameters in the opposite direction of the gradient of the objective function. There is a fatal disadvantage in this approach. With a learning rate to depend how much it iterate in each step, the steepest descent algorithm may not reach the optimal or get into a local optimal.

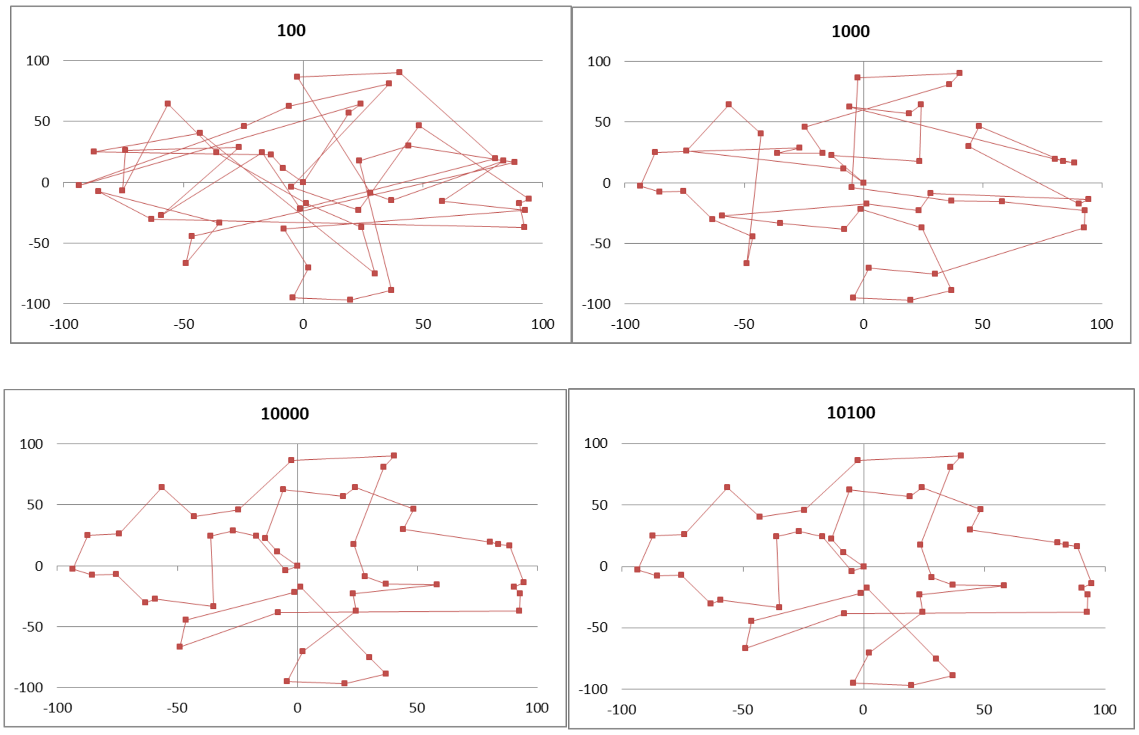

Fig.2 The calculation result in some certain steps with steepest descent

As one kind of random-search technique, the Simulated Annealing takes the probability of the stochastics into account to reach a global optimal. More specifically, we first randomly select an initial point and a decent direction. After the comparison of their calculation result of the target function, we will accept the new one if the calculation result is smaller and use Exp function to get its probability if being larger. With the decrease of the T, the probability P will decrease as well and reach stable which is the end of the calculation.

$$P = \mathrm{e}^{(-\Delta f/T)}$$

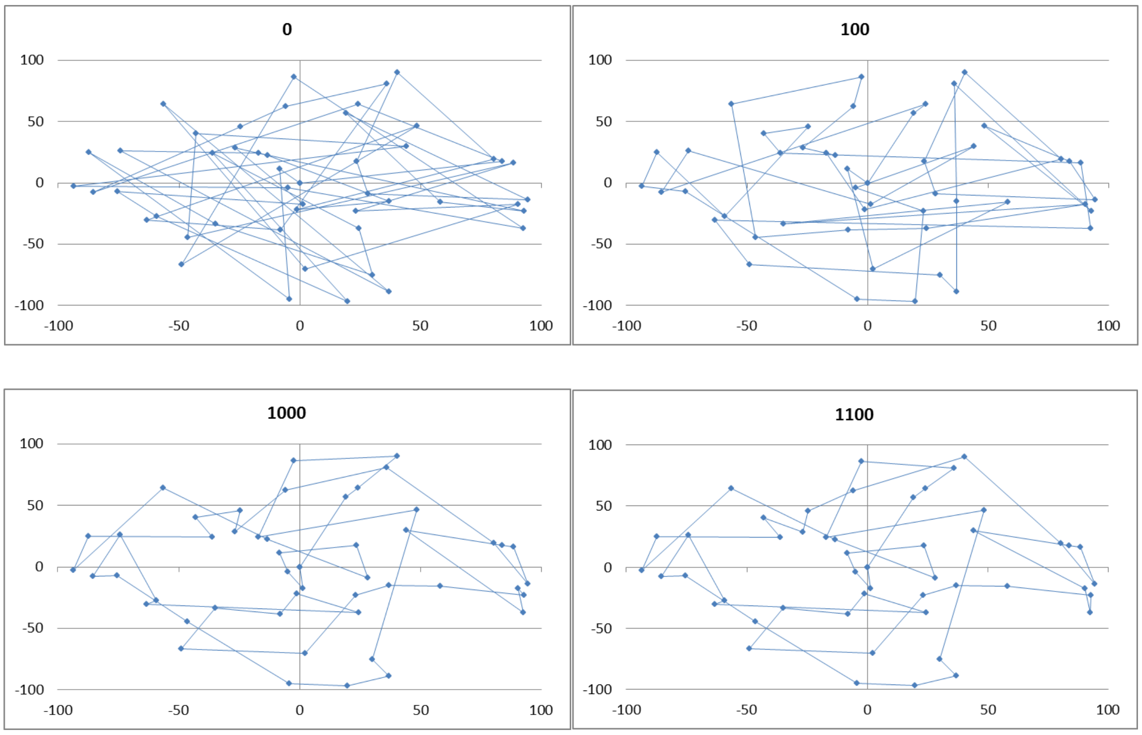

Fig.3 The calculation result in some certain steps with simulated annealing

As we have gotten the calculation results of these two approaches, we can easily find that the gradient descent approximately reaches the optimal point in the 10000 step and the simulated annealing is in the 1000 step, which means the simulated annealing is more efficient than the gradient descent.

Here is a question about the simulated annealing: under my understanding, the value of T will affect the speed of reaching optimal. However, it seems not correct from the calculation of distance, in which I change the T from 250 to 1000.

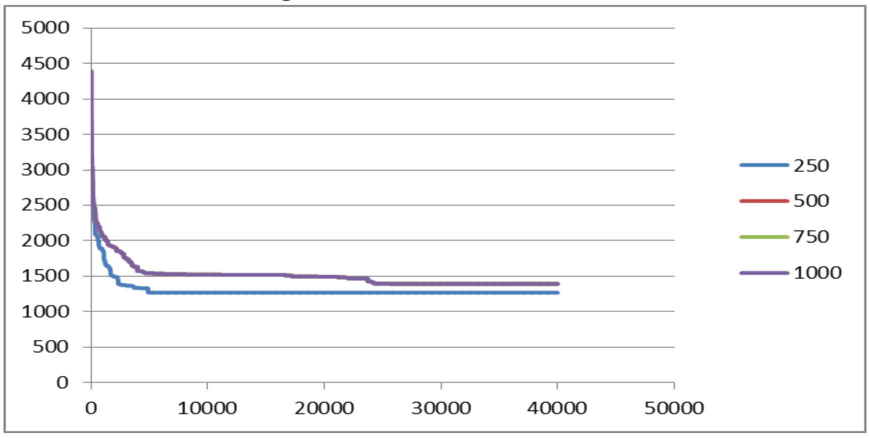

Fig.4 the calculation result of distance with different value of T

```
int main()
{
          int step = 0;
          int i, j;
          double temperature = 200.0;
          double distance, dtemp, prob;
          SITE stemp;
          FILE *fdist;

        fdist=fopen("dist_sa.dat","w");
        distance=calc_dist( );
        output(step,distance);

        for (step = 1; step <= TOTAL; step++) {
        PICKUP_I:
            i = (NSITE - 1)*myrand();
            if ((i == 0) || (i == NSITE - 1)) goto PICKUP_I;
        PICKUP_J:
            j = (NSITE - 1)*myrand();
            if ((j == 0) || (j == NSITE - 1)) goto PICKUP_J;
            if (i == j) goto PICKUP_J;
           exchange(i, j);
           dtem = calc_dist();
            if (dtem<distance) { distance = dtemp;}
            else { prob = exp(-(dtem - distance) / temperature);
                   if (myrand() < prob){distance = dtem; }
                   else{exchange(i, j); }}

            printf("%10d %10.5f¥n", step, distance);
            fprintf(fdist, "%10d %10.5f¥n", step, distance);

            if (step%OUTSTEP==0) {output(step, distance);}
            temperature *= 0.5; }

    fclose(fdist);
             return 0;}
```