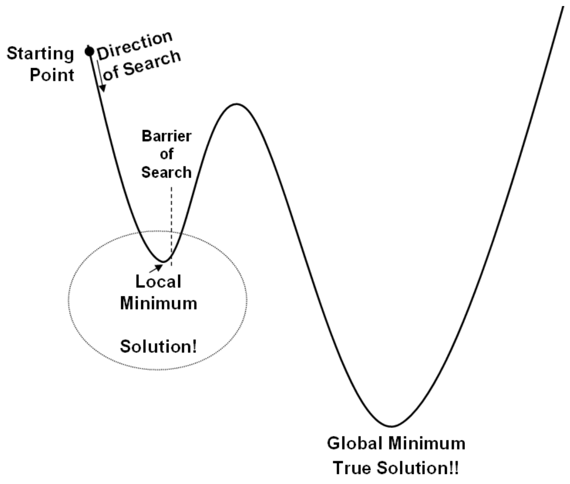
S&T-1st C Programming Lab 10

Bonus – Simulated Annealing (模拟退火式的算法)

-- 6:45pm



In solving the optimization problem, the runtime of the program can grow incredibly long for a large problem size as we need to search through an enormous number of possible permutations to find the best solution. Heuristic approach may not be the best alternative because the algorithm may be trapped in a local optimum. The example on the right is a hill climber's heuristic to find the minimum of a problem space where the solution was trapped in a local optimum. The hill climber can accept a new solution provided the new solution is better than the current solution. Since the next step at the local minimum is not better, the heuristic stops but the true solution has not really been found yet.

A way to prevent the algorithm from being trapped in a local optimum is by accepting the solution which is worse than the current solution and subsequently to proceed from there. But this will have to be done in a controlled manner and this is how simulated annealing works. As learnt from the Internet (http://www.theprojectspot.com/tutorial-post/simulated-annealing-algorithm-for-beginners/6), the simulated annealing algorithm was originally inspired from the process of annealing in metal work. Annealing involves heating and cooling a material to alter its physical properties due to the changes in its internal structure. As the metal cools, its new structure becomes fixed, consequently causing the metal to retain its newly obtained properties. In simulated annealing we keep a temperature variable to simulate this heating process. We initially set the temperature high and then allow it to slowly 'cool' as the algorithm runs. When this temperature variable is high the algorithm will be allowed, with more frequency, to accept solutions that are worse than our current solution. This gives the algorithm the ability to jump out of any local optimums. When the temperature is lowered so is the chance of accepting worse solutions, therefore allowing the algorithm to gradually focus on an area of the search space in which hopefully a close to optimum solution or the optimum solution itself can be found. This gradual 'cooling' process is what makes the simulated annealing algorithm remarkably effective at finding a close to optimum solution when dealing with large problem spaces which contain numerous local optimums.

More specially, during the iterations the simulated annealing algorithm will have to keep track of the best solution so far. The algorithm will randomly perturb the existing solution to create a new solution. This new solution will be <u>unconditionally accepted</u> if it is better than the current best and will be used as the reference for the subsequent iteration. If the new solution is worse than the current solution (which may not necessary be the current best), the new solution will still be accepted if the weighted differential cost is within a statistical threshold. The details are as follows:

<u>differential cost</u> = <u>new cost</u> - <u>existing cost</u>
<u>weighted differential cost</u> = <u>differential cost</u>/temperature

The algorithm will then draw a random number *r* in [0..1], and if $r < e^{\text{- weighted differential cost}}$, the worse solution will still be accepted to replace the current solution and will serve as the reference for the next iteration.

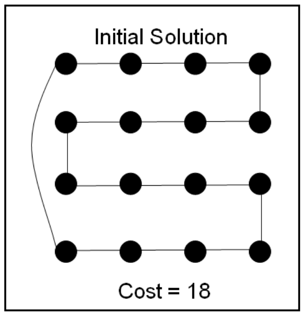
The cost function will be the objective function of the problem space and it is defined by the programmer. For examples, the cost can be the total distance of travel, length of loop, or the number of objects on a fixed area etc.

On the use of the weighted differential cost, it is clear that the higher the temperature at the starting stage of the algorithm, the chance of accepting a worse solution will be higher. When the temperature has a low value during the ending stage, the chance of accepting a worse solution will be low thus the ending journey towards the direction of optimum will be a relatively steady process but it is still in probabilistic manner. It is also worthwhile to understand the characteristics about the exponential function as follows:

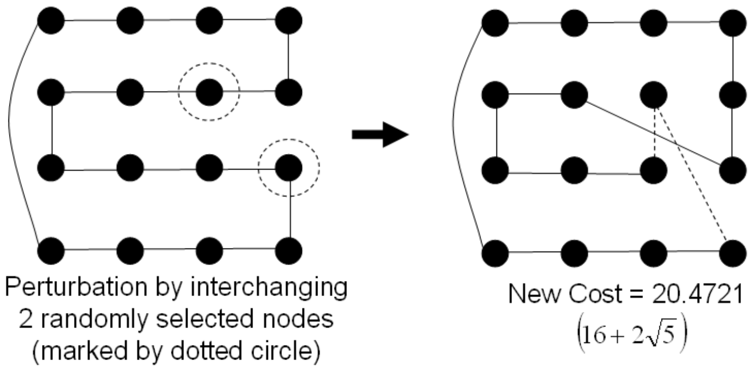
(i) $\int_0^\infty e^{-T}\,dT = 1$ (Total probability property) , and

(ii) Prob ($\dfrac{T > s+t}{T > s}$) = $\dfrac{\int_{s+t}^\infty e^{-T}\,dT}{\int_s^\infty e^{-T}\,dT} = \dfrac{e^{-(s+t)}}{e^{-s}} = e^{-t}$ = Prob ($T > t$) (Memorylessness property of exponential distribution). This means that any weighted differential costs can happen at anytime (because it is memoryless) during the annealing process thus this requires a careful control when accepting the worse solution.

<u>**Your assignment-1**</u> is to find the shortest length of a close loop to join all the point in a 4x4 grid. You can start with any initial solution. An example with the initial cost of 18 is shown on the right.



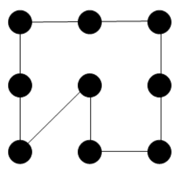
Subsequently you can perturb two random points on the loop and a new cost will be computed. One example of the perturbation is shown as follows:

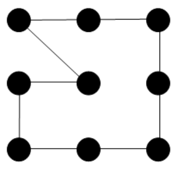

Perturbation by interchanging
2 randomly selected nodes
(marked by dotted circle)
New Cost = 20.4721
$(16+2\sqrt{5})$

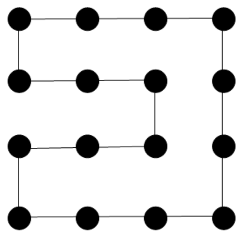
If the starting temperature is 500 and the cooling rate is 0.001, you will now compute the exponential value and to decide to accept the worse answer or not. The iteration stops when the temperature reaches the lowest temperature of 0.000001. The main algorithm is as follows:

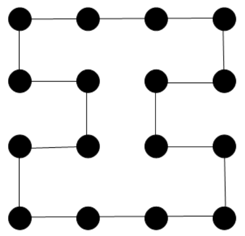
```
set current loop and current length to initial loop and initial length respectively;
set best loop and best length to initial loop and initial length respectively;

set temperature to starting temperature;
 while (temperature >lowest temperature)
 {
      randomize two points in the current loop to obtain a new loop;
      compute the length of the new loop;

      if (new length is shorter than the current length)
           accept the new loop for subsequent reference;
      else
       {
           draw a random number in [0..1];
           compute the acceptance threshold by exponential function;
           decide to accept the worse solution or not based on the threshold;
       }

      if (length of new loop is shorter than the length of best loop)
           updated the best loop and best length;

      decrease the temperate by the cooling rate;
   }
```

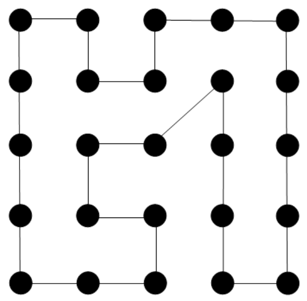
The proposed solutions from my programs are not unique:

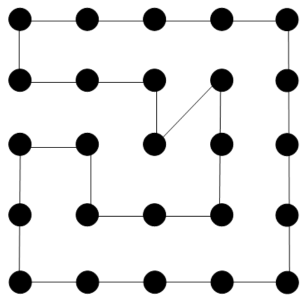

Cost = 9.4142


Cost = 9.4142


Cost = 16


Cost = 16

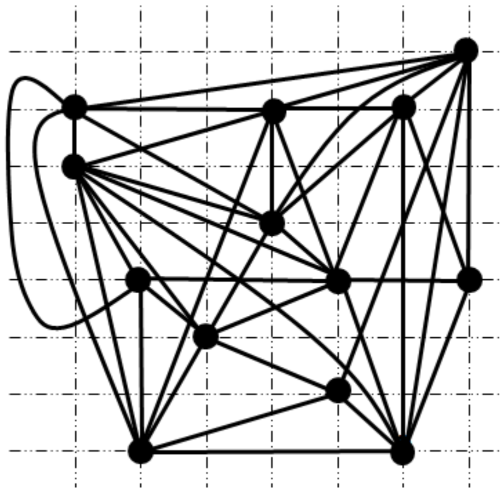

Cost = 25.4142


Cost = 25.4142

Use debugger whenever in doubt!

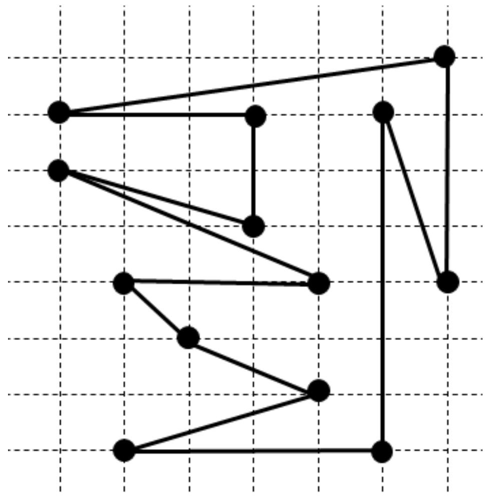
Please take note that the simulated annealing method does not guarantee the best solution in a finite number of iterations. The advantages of this method is that (i) it has a chance to jump out from a solution which is a local optimum but is not a global optimum, (ii) its runtime is a constant, i.e., the time complexity is of O(1) regardless of the problem size!

The Finale



Your **<u>assignment-2</u>** is slightly more challenging as the connection of the grid points is not systematic. You are going to carry out route planning with 13 towns which are <u>**not** fully connected</u>.

The 13 towns represented by dots are modeled on a 2-D grid where the grid lines are equally spaced out as shown on the right. These 13 towns are not fully connected and their connectivity is specified by the edges joining the dots. You are to provide a routing plan to achieve a reasonably short journey to cover all the 13 towns where each town can be visited only once.



You can start with the initial solution as shown on the right and use simulated annealing method to answer this question.

Use debugger whenever in doubt!!