Lab 5 report

Lab 5 is an extension of the previous lab. In this assignment I’ve added two algorithms to existing Factory design pattern: Simulated Annealing and Particle Swarm Optimization. I’ve also fixed design and memory errors and made the code look neater.

# Simulated Annealing

Temperature: The key to SA's ability to find global optimums, and what differentiates it from simple hill-climbing algorithms, is that at each iteration it has a (decreasing) probability of moving to less fit states. It always moves to more fit states but, presented with the prospect of moving to a less fit state, it will do so with probability equal to Exp[-(dFitness)/T], where T is temperature. Theoretical analysis of SA indicates an optimal (maximum) cooling rate that ensures ergodicity and thus a positive probability of escaping from local optimums and eventually finding the global optimum. This optimal temperature schedule depends only on two measures of a given problem's graph ("r" and "L"). However, its applicability to implementation was rejected due to: (a) Irrelevance—The Main SA Theorem, though theoretically encouraging, is not applicable to implementations because it assumes we will let the chain run a lot longer than we can (infinitely long, in fact). In practice, we can't really hope to sample from the complete set of global optima, but only to get an optimum that is not very local. (b) Difficulty—In most problems we don't know what the topology of the state space is and so can't estimate the needed parameters of the graph to determine the optimal cooling schedule. In fact, if we could, it would probably also be possible to devise a more specific and efficient algorithm for the problem than SA.

Given that computing time is the primary constraint in implementation, I devised the following universally applicable temperature scheduling methodology: The first 1%, say, of the iterations of a run are used to "melt" the space—essentially sampling the graph at random to determine the minimum temperature at which all paths would be equally likely. From this the temperature decreases until we go through all iterations.

The whole simulation is executed several times in a cycle from different starting points to ensure that we find the best solution.

S = 5 - number of simulations  
k = 1000 – number of iterations  
Temperature\_init = 1000  
Temperature\_decrease\_per\_step = 10%

The project was build and tested for 2-200 nodes. But could go further.

The second approach is a steadily decreasing temperature from a melting point down, without repetition of the simulation.

S = 1   
k = 5000 – number of iterations  
Temperature\_init = std::numeric\_limits<float>::max()  
Temperature\_decrease\_per\_step = 5%

The runtime for 2-200 nodes was a bit slower. The results are sometimes worse than with the first method, since we do not test multiple starting positions.

# Particle Swarm Optimization

Test 1

Learning factors and other parameters:  
alpha = 0.5f – for particle best  
beta = 0.9f – for global best

max\_velocity\_size = 50 – max length of swap sequence

num\_of\_particles = 100

num\_of\_iterations = 10

These parameters provide similar results to Simulated Annealing algorithm

Test for 2-200 nodes:

Test 2

Learning factors and other parameters:  
alpha = 0.3f – for particle best  
beta = 1.0f – for global best

max\_velocity\_size = 10 – max length of swap sequence

num\_of\_particles = 100

num\_of\_iterations = 10

With these parameters the result is more likely to slip into a local minimum. Execution time is not affected that much by max\_velocity.