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 Optimisation. I’ve also fixed design and memory errors and made the code look neater.

# Simulated Annealing

Simulated annealing is a stochastic search technique in which a randomly generated potential solution, N, to a problem is compared to an existing solution, O. The probability of N being accepted for investigation depends on the proximity of N to O. If N is accepted, its suitability as a solution is evaluated according to a swap probability function and it may be chosen to replace O. Both the acceptance and swap probability functions depend on a temperature parameter T, which reduces in value as the algorithm proceeds.

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.

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 sample 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.

The other way, that was tested, is to run a simulation one time with a high temperature which starts dropping since the first iteration.

# Particle Swarm Optimisation

Particle Swarm Optimisation is an iterative optimisation technique inspired by the biological behavior of a swarm of birds or bees. Unlike evolutionary optimization techniques such as Genetic Algorithms, it is not based on the idea of the survival of the fittest. Instead, it is a collective method in which members of the population cooperate to find a global optimum in a partially random way and without any selection. Members of the population with the lower fitness functions are not discarded but do survive and can potentially be the future successful members of the swarm.

The position of each particle within the search space is defined as a sequence of vertices on an input graph. Each particle receives information from other members of the population about their best position to date and will also remember its own best position. The particle then calculates its velocity and position based on these parameters: the particle’s own velocity, moving towards the particle’s own best position so far and moving towards the best position of its best informer. I defined particle’s velocity as a sequence of swap operators over its position. After each iteration the program appends the velocity to the best particle’s local solution and swarm’s global solution to the velocity of the current particle with learning factors *alpha* and *beta*. *Alpha* is a probability that all swap operators from velocity to local best are included, and *beta* is for global best. Swap sequence length is capped by max velocity, which is basically a maximum size it can grow to. If it is getting longer than max size, the oldest swap operators will be removed. This is needed to avoid slow-downs if the number of iterations is chosen to be high.

# SA vs PSO

SA and PSO main difference lies in their working principles. SA attempts to escape local minima with a chance accepting worse solution on a certain step, while PSO uses what can be called “swarm intelligence” when there are a lot of starting particles at different positions and every one of them can find its own local minima of which the best is selected. In my opinion, SA will slip to a local minimum more often than PSO because it has only one starting location. I tried to solve this by launching it several times with different starting location. But this obviously affects the execution time.

There’s not much of a difference in terms of quality of answers provided by those two algorithms. Timewise, PSO appeared to be considerably faster, however this difference probably can be tuned down by finer selection of parameters.

# SA Testing

## Test1

S = 5 - number of simulations per launch  
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.

## Test 2

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.

# PSO Testing

## 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.

# Design Patter and UML

I have used factory pattern for the implementation of all of the algorithms. I selected this pattern because it works well with the algorithms that we have implemented in this lab. The managing of this pattern is super easy in my case and different interfaces can be managed easily using this pattern. In my code AlgoHandler.h is one of the factories that I have developed in the game. This factory constructs a specific algorithm object. My factory includes several smaller modules as well such as File reader. There is a graph class which helps creating Graphs with the help of input files with information of edges and weights. Interface for TSP is inherited by Algorithms.h which is further used to implement various algorithms needed for the search and selection.

