**Supervisor meeting:**

4 points:

Testing.

Requirements.

What the classes should do.

Input output

It is importaint to consider what input the program should have, what parameters and so on, THIS SHOULD BE IN THE START OF THE REPORT.

You should at least have the best tour length and time used when the program ends.

It is importaint to have requirement specifikations – it is importaint to remember what variables you need.

You don’t need 4 classes you just need 3, write in the main class with the control.

InitializeData is just input. Make it as a class instead of a file inside control.

The problem with this is that you have to send input between the files.

You have decided that the ants shouldn’t do anything but they should. It is the ant that does the tour contruction and if you want to implement threads you have to give it the constructtour function.

You could If you had time make it more object orientated like having city as an object and such.

**Hints:**

The main method is just starting the ACOTSP and provide the ARGS argument.

**ASOTSP:**

First you parse the parameters to the program and starts ACOforTSP.

**ACOforTSP:**

The program runs as that it first initialize the data then it adds the ants to a ant list.

It is just as simple as in the book – done exactly as in it.

You need a package gnu.getopt.getopt, it can be found on the internet and it is needed to part the data.

**Parseargs:**

Start with making an object of getOpt which is a new getopt, it needs a string(the name of the program), the args and the i:

If you open the getOpt package there is a description of what it does

What it does is that it makes it possible to read one option at a time.

It is used to make the option list.

You make a while loop that reads all the options from the g.getopt() object.

The parseArgs should always return true.

Now we are only parsing strings, to part an int like the ants use integer.parseint(g.getOptarg())

The way it should look in the parameters in intellij if we want to parse the ants is: –m6

What the initializedata should do is.

1. readData();
2. computeDistances();
3. computenearestneighborlists();
4. initializeants();
5. computechoiceinformation();
6. initializeParameters();
7. initializeStatistics();

Instead of using the nearest neighbor algorithm to initialize pheromone you can use just very small numbers.

ReadData is pretty much as we did it.

Compute distances

What the NNlist contains in the number of nearest neighbors to the ants at a specific city. For an example you could have the 20 nearest neighbors.

So if you have 280 cities and only search for the 20 nearest neighbors it is a speedup of 280/20 = 14!.

Initialize ants is much like we have it but by calling the ant array list.

Pheromones is initialized to 1e-10.

The pheromoneupdate is:

Evaporate();

For (ant a : ants)

Deposit pheromone

End for

Computechoiceinformation.

You should use the random class instead of the math function to geneate a random number.

For the elitist ant you should just let each ant find their own tour.

At the variable used to update at the deposit pheromone the 1/tourlength must NOT be 1 but 1.0.

We must remember that we only consider objects within Euclidean space.

If you know the optimum you can hardcode how far it is away from it.

To not get 0 at the dist set the heuristic factor to 1/dist[i][j]+0.1.

At the ants there is provided at tour and a tourlength.

What the ant should have is.

Tour array

Pos array (is the index, the position at the tour)

Visited array.

To call the ASDdecisionRule random number use ACOTSP.rand.nextDOuble() \* sumprobabilities

**2-opt**

The index in the 2 opt is called don’t look, checks which cities we have already tried to make 2-opt on.

In 2 opt in the ants you can make a shuffled array which you can use to make 2-opt on.

The way it goes in the 2-opt is: a = prev(b), d = prev(c) and c= nextd).

The prev of two opt is, if it the number is more than 0 in the index then the position[v] – 1 else it equals to n – 1;

If not done like this then the 2-opt

All cities are enumerated fro 0 to n-1.

Also remember to swap both the tours and the index (pos) when finding a 2-opt move.

Pos is simply just a way to specify that when you find a city find the next position of the tour – it is an index.

You also need to use the position in the flip move.

In 2-opt you can use tourlength-=gain when finding a better tour – it is SAFE.

Pos[tour[i]] = i **REMEMBER**

**ConstructionSolution:**

Starts with initiating visited[i] to false for all cities.

Then set ievery ant to start at a random city by setting it to tour[step] = ACOTSP.rand.nextint(n);

While step < n do neighborlistasdecisionrule

To go back to the starting city set tour[n]=tour[0]

EXPECT ABOUT 15813!.

You can make a class to check the tour by checking for whether there are no duplicates(if so it isn’t a tour).

**Threads:**

one way to implement threads with ants is to have it implement runnable in the public ant class.

Make a public void run() { which calls construction solution.

}

Then make a thread for each ant in the program and start it.

The problem now is to check if the threads are alive. to do this we need to make an array/list containing the threads and then TRY use the method join for each ant(remember to use a try catch for the join) .

What the join method do is to check for each ant whether it is finished.

1. Thread A = new thread (ant)
2. A.start();
3. A.join();

The way to specify a list is list<thread> threads = new arraylist<threads>(); if it should be an arraylist

Without the thread list the thread keeps running and not waiting for all the ants to be finished.

The only other thing that could be speeded up with threads might be the pheromone evaporation which is

To give a different starting position for all ants use the hashcode() function in the random number.

**Commandoprompt when in the program:**

Man javac – can be used in commandprompt to profile the program-

Top – can be used to see the running time for the CPU.

**Report:**

Documentation of the program

Requirement specifications.

Testing the program

Running it with examples.

If you don’t manage to make it object orientated enough in the time left then write in the conclusion how well you have furfilled your requirements and say how you could have done it more object orientated and say what you will do in the future(perspectivation).

Make an introduction that doesn’t go into the code or theory but just explain what you have done and explain the project in a simple way.

Remember to make a descriptive simple title.

A preface is just like in a book that tells what is in the report and under which circumenstances the report has been made(it’s a bachelor project and who is the supervisor feks.) and tell something how the report is structured and how it should be read. Also tell here what is the prerequisites for reading the report (such as knowing java). It should simply be clear from the beginning what it is all about.

You should also have an abstract and a preface.

If you need pictures of the tour for the report you can save the tour as a png or jpg file.

**Program:**

Remember at each class write in // what the class does, there needs to be description of your program.

**Testing:**You should always somehow check that the code is working.

If you see that the tours are reasonable it should be good enough testing for you.

**2 opt:**

For each b you know that the a is the predecessor of b and that d is the predecessor of c.

Gain= Dist[a][b]Dist[d][c] - Dist[a][d]Dist[b][c]

Defining the right primitives the code becomes simple, these are prev, swap and flip.

**For the next supervisor meeting:**

Include the ipr and iws besides the iml when you send your program for a supervisor meeting or else keld will not be able to compile it with commandoprompt.

If you sit 2 hours+ then consider sending a well explained problemformulation of the problem to keld and he may be able to help you over mail really quick.