TI2736-A

Report Assignment 3

A Robot To Do Your Groceries

Delft University of Technology

Group 19
Rick Molenaar, 4280814
Matthijs Klaassen, 4273796
Daniël Brouwer, 4288297

Friday 30th October, 2015
Version 1.0

Contents

1 Introduction

For the grand challenge, the robot needs to navigate through a supermarket that is modeled by a maze. The robot should find the fastest route picking up several products on its way. This problem is usually referred to as the Traveling Salesmen Problem (TSP) and can be solved with a Genetic Algorithm.

Before the robot can do this, it first needs to determine the shortest paths from start point to the products and from every product to all other products. To determine the shortest path Swarm Intelligence more specifically an Ant Colony Optimization (ACO) algorithm will be implemented.

This report contains answers to the questions given with this assignment. The questions give an impression on how the ACO and TSP algorithms are implemented. Both the ACO and TSP algorithms were made using the Java programming language.

2 Ant Preparation

1

- Large open areas: An ant might not always take the shortest path here because it won't know the position of the exits.
- Dead ends: An ant might take a path with a dead end. It would be smart to "fill" these paths, so no other ant will take it.

2

Pheromone is dropped on a path as an indication for the length of the path. More pheromone means a shorter path. The algorithms purpose is to find an optimal (or as close to optimal) path. We used a slightly adjusted variant of the pheromone dropping equation, which worked better for our implementation. The formulas are show below with (1) the pheromone to add and (2) the new pheromone amount on a path

$$\Delta pheromone_{ij} = pathlength_{ij} * Q/distance \tag{1}$$

$$pheromone_{ij} = pheromone_{ij} + \Delta pheromone_{ij} \tag{2}$$

- distance - total distance of the path that the Ant found.
- $pathlength_{ij}$ - the length of the path between 2 nodes i and j (The algorithm we implemented transforms the maze into a graph containing nodes).
- $\Delta pheromone_{ij}$ - the pheromone to be added on that path.
- Q = an estimation of 'distance'.

3

We assume as the algorithm progresses that each iteration forms a shorter path than it's predecessors. Thus newly dropped pheromone probably indicates a shorter path, therefore pheromone evaporates slowly to make an distinction between older and less optimal paths and newer better paths. In our implementation, if one of the Ants reaches the endpoint, the covered route its pheromone gets updated. And afterwards all paths in the whole maze will be evaporated (thus also the newly pheromone added paths). For this implementation the following equation is used: see (3).

$$pheromone_{ij} = (1 - evaporationconstant) * pheromone_{ij} \tag{3}$$

3 Implementing Swarm Intelligence

4

A short pseudo code description of the implementation:

1. Set variables (alpha,beta,evaporationconstant,Q,amountofants,amountofwinners) according to the chosen maze difficulty.
2. Spawn 'amountofants' ants into the maze.
3. Each Ant chooses a path based on the path probability which depends on the pheromone and path length.

4. WHILE(amount of winners < amountofwinners)
5. let the Ants move.
6. IF(an ant reaches the end coordinate)
7. THEN update the pheromone on the path & evaporate ALL paths.
8. AND place all Ants back to the starting position.
9. Repeat and keep executing WHILE loop till we have 'amountofwinners' Ants reaching the endpoint.
10. Release a SuperAnt which always chooses the path with the highest path probability.
11. Save the directions of the SuperAnt.

4 Upgrading Ants with intelligence

5

Before letting any Ant run through one of the mazes, the idea was to first convert the maze into a graph representation. This is done by making nodes of 'crossings', endpoints, the start and end-coordinate. Between every node there is a path each having an amount of pheromone and a path length. Figure 1 shows an example of this graph representation. The X is a node, a dot is a path and a full block is a wall. This 'simplification' of the maze is done because it will reduce the amount of computations needed and therefore speed up the algorithm.

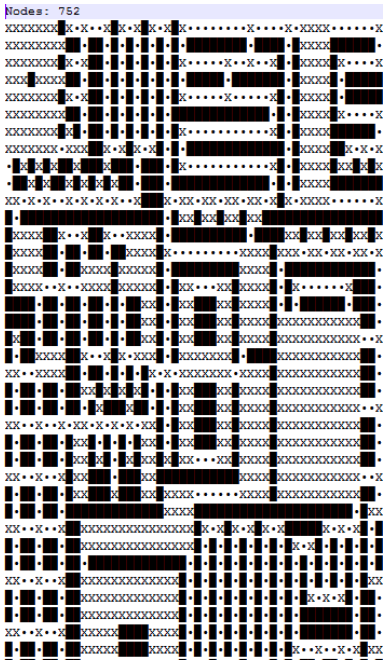

Figure 1: Graph representation of a part of the medium maze

Different techniques were tried for releasing the Ants. The first approach was to just release all Ants at once and stop the algorithm once a certain value of ants reached the end coordinate. Afterwards a SuperAnt was released which always chooses the path with the highest probability. This first idea failed miserably because loops in the system caused certain paths to have more pheromone and therefore SuperAnt got stuck everytime.

The second approach was to release one Ant at a time and only release a new Ant if the previous Ant has reached the endpoint. This idea worked better but it still didn't account for loops

in the system. Also releasing the Ants one by one caused a big performance decrease.

The third and best approach was to release all Ants at once and once an Ant reaches the endpoint, every Ant gets reseted to the starting point and released again. Moreover the loop problem was addressed by removing all directions between a node that is visited twice and thus only the pheromone gets updated of the 'direct' route a particular Ant found. In section 2 the equations are show which we used for updating the pheromone on the path and for evaporating all paths.

To optimize the parameters of the system, we saved the information about each individual ant to a .txt file which we later analyzed using Matlab to increase the performance of the ACO algorithm. More about this topic in section section 5.

The only problem left was the problem depicted in figure 2. In large open areas, our Ants will sway left to right. Unfortunately due to time constraints, we could not implement a solution for this problem before the deadline.

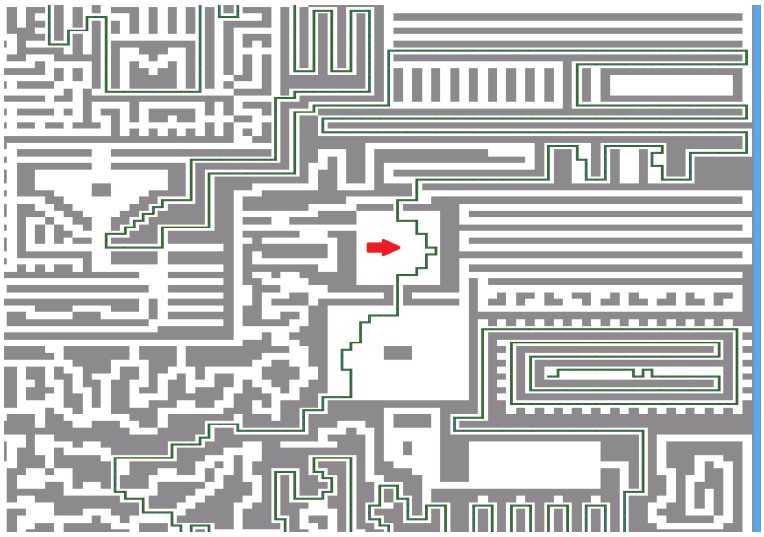

Figure 2: Problem in open areas

5 Parameter Optimization

6

The Ant colonization optimization algorithm depends on a few variables:

- Alpha - a constant used in the 'path probabilities' formula which increases the impact that the pheromone on the path has.
- Beta - a constant used in the 'path probabilities' formula which increases the impact that the path length has.
- AmountOfAnts - The amount of ants that will be released and need to find the endpoint.
- Q - controls the amount of pheromone that is dropped by the ant. Q needs to be an 'good' estimate of the path length.

By varying each parameter individually while keeping the other parameters constant and comparing the convergence of the solutions, theoretically the 'optimum' parameters can be found. Because probably each parameter relies on each other, when one optimum parameter is found (e.g. Q) the constant parameter Q will be updated to the optimum value, and than the next optimum parameter will be found (e.g. alpha).

Because the amountOfAnts generally does not improve the rate of convergence it is hold to a 'high' constant initial value of 100. Afterwards, when the optimum parameters are found, the convergence point determines the amount of ants that needs to be released to increase the speed of the program.

The following parameters were used as a 'starting point':

- Alpha - 1
- Beta - 0.5
- AmountOfAnts - 100
- Q - absolute value of $(s_x - e_x)$ + absolute value of $(s_y - e_y)$ with (s_x, s_y) the start coordinate and (e_x, e_y) the end coordinate. This is chosen as a rough estimation of the path length.

Beginning with parameter Q, the parameter will be multiplied by an increasing factor x. Every iteration the amount of steps that each Ant has to walk, is plotted. The value of Q which results to the fastest convergence will be chosen taking into account that the convergence speed depends on the amount of steps of the first Ant. The results of the easy, medium, hard and insane mazes are displayed in figure 3, 4, 5 and 6.

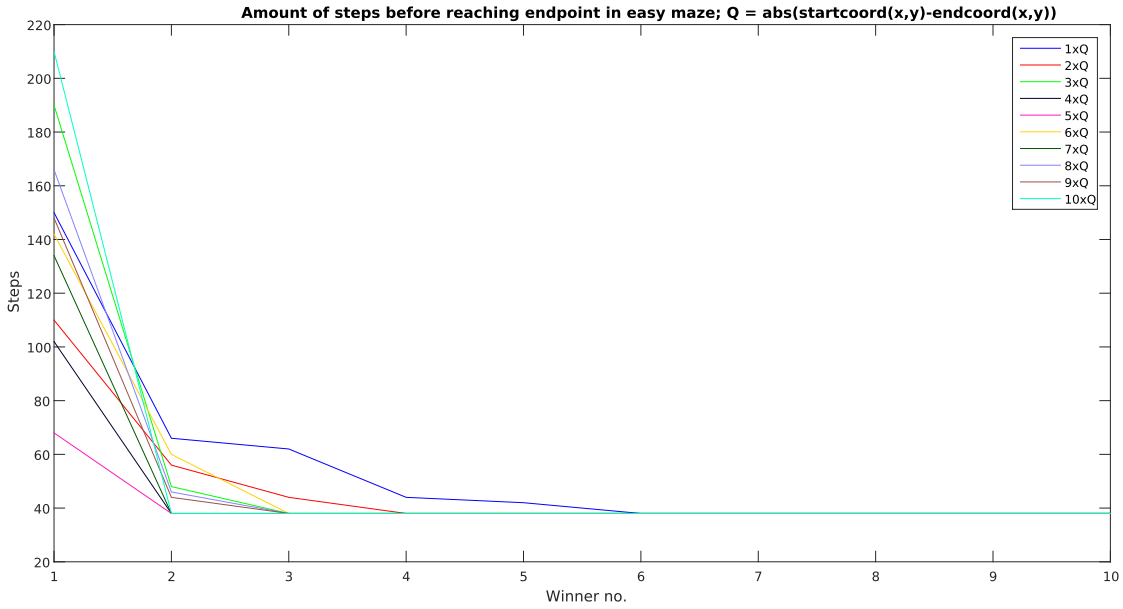

Figure 3: Convergence in easy maze as a function of Q

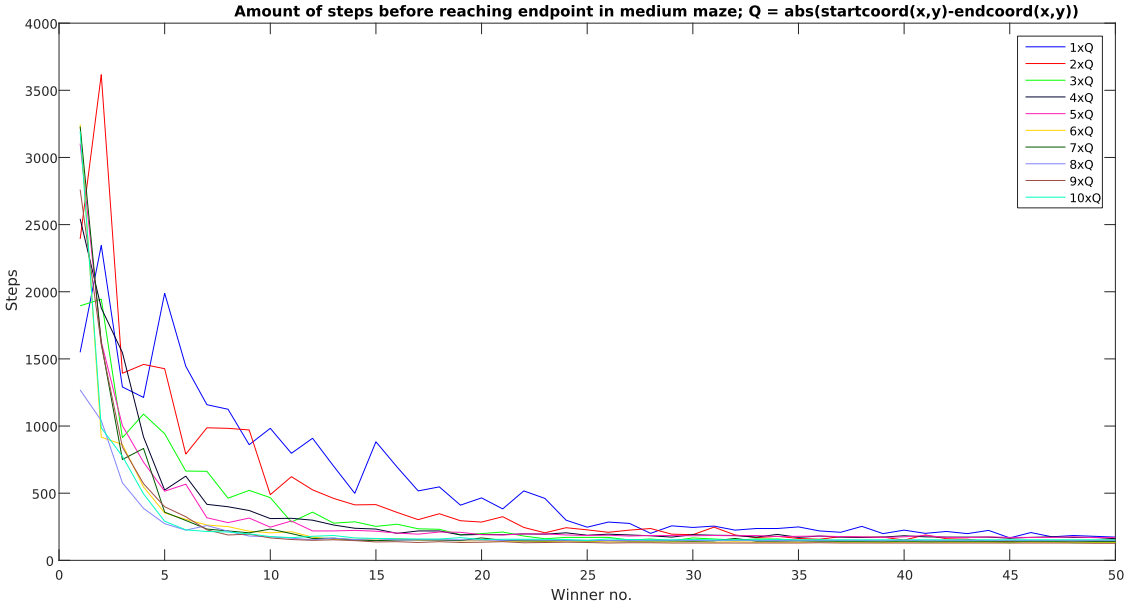

Figure 4: Convergence in medium maze as a function of Q

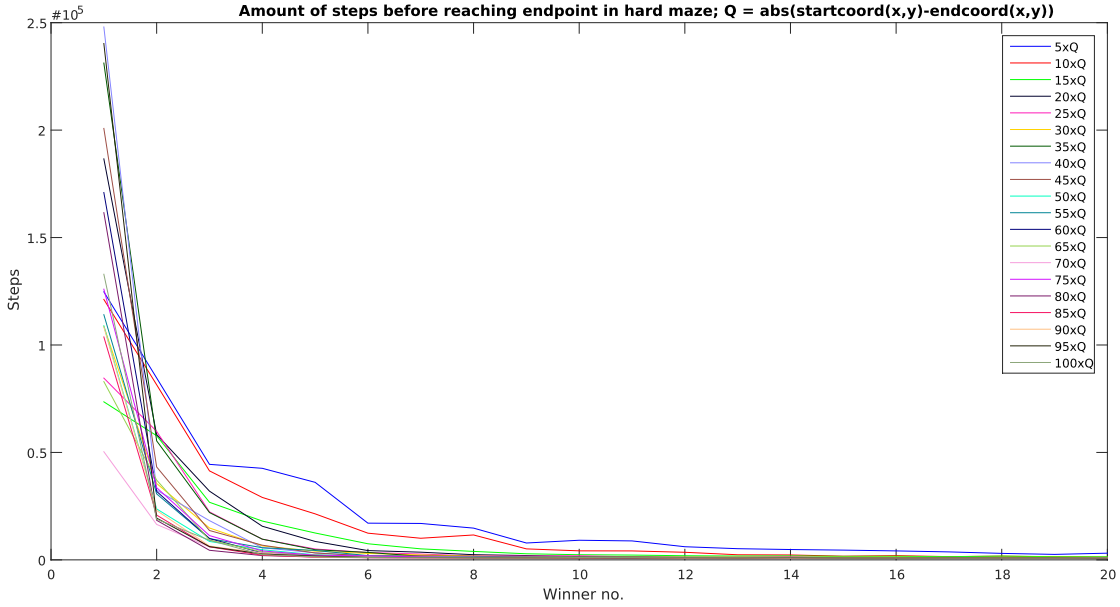

Figure 5: Convergence in hard maze as a function of Q

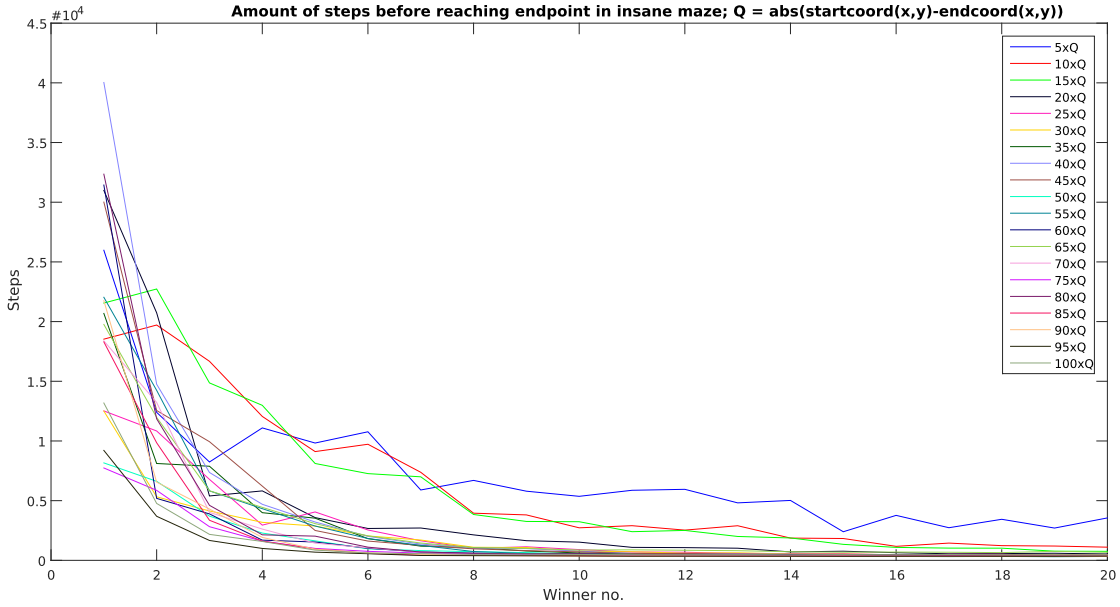

Figure 6: Convergence in insane maze as a function of Q

From the graphs is is shown that the initial estimate of the path length was quite bad. The graphs also shows that a higher Q results in a faster converging system to a certain extend. The find the optimum Q, a Q is chosen for each maze, that converges fast and has the smallest value possible because choosing a Q that is very high, may result in Ants dropping to much pheromone which causes Ants to always follow a certain path without discovering other possibilities and therefore the algorithm will converge to a local optimum.

The following values for Q are deduced from the graphs:

- Easy - 3Q
- Medium - 9Q
- Hard - 30Q
- Insane - 40Q

The same procedure has been followed for Alpha and Beta only this time the variables are incremented from 0 to 1 with steps of 0.1f. The results are show in figure 7, 8, 9 and 10.

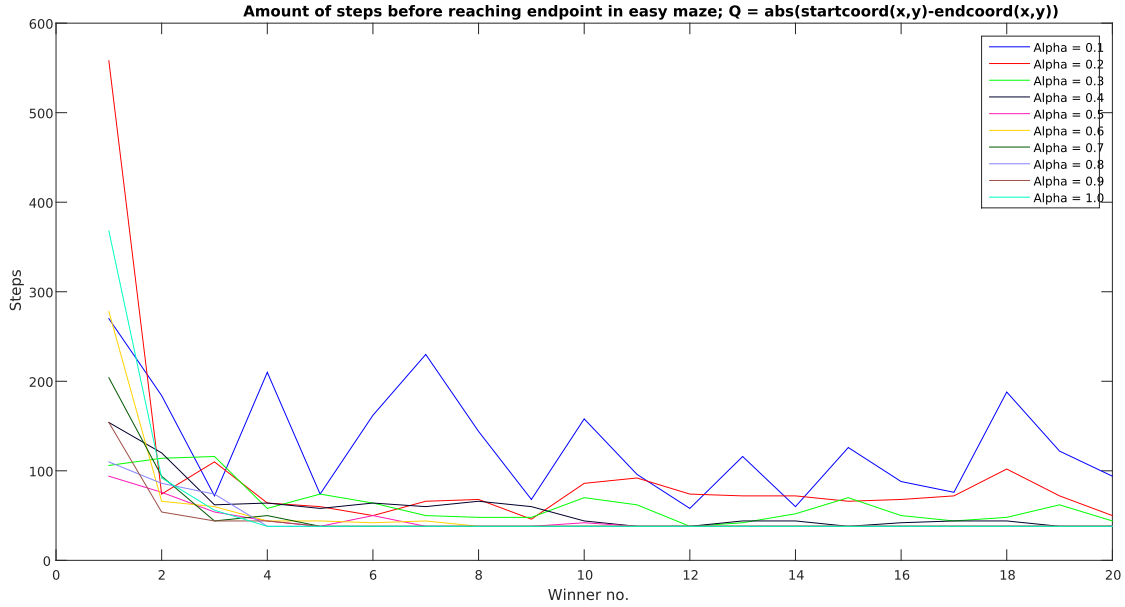

Figure 7: Convergence in easy maze as a function of alpha

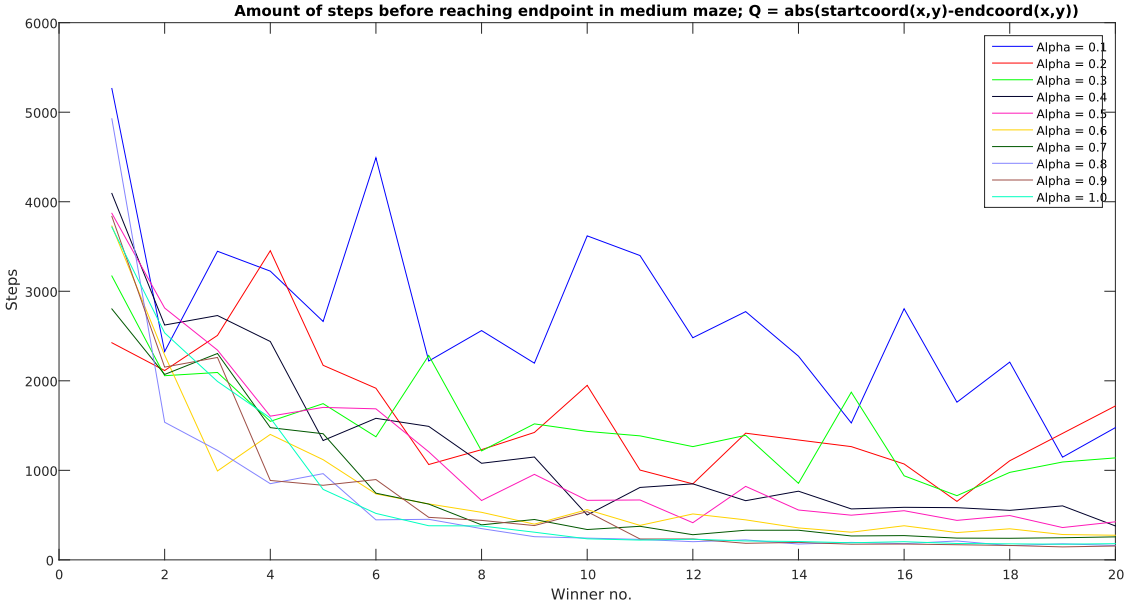

Figure 8: Convergence in medium maze as a function of alpha

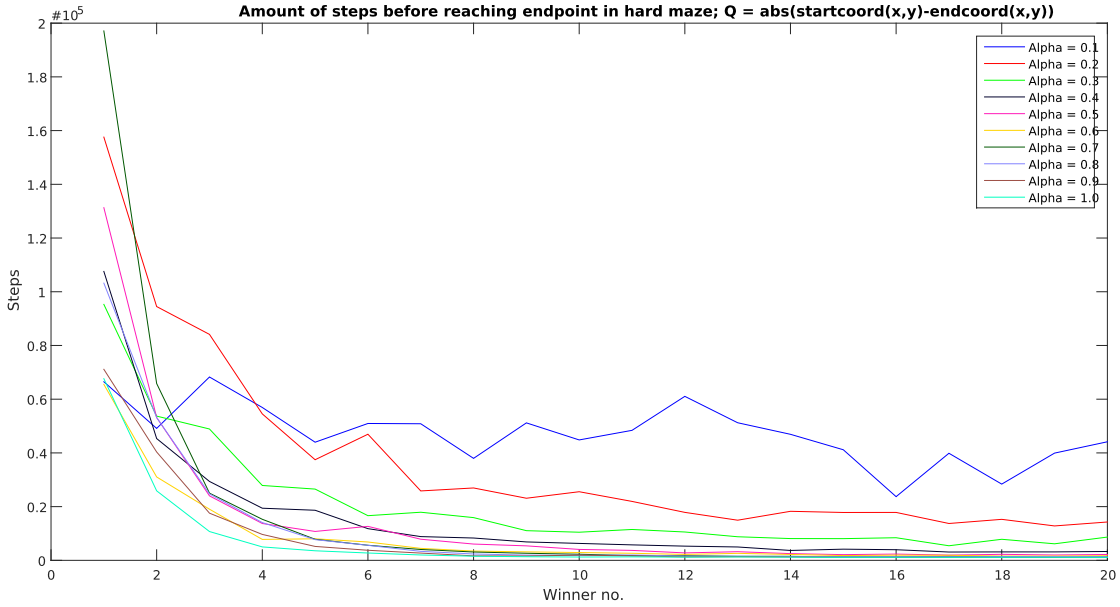

Figure 9: Convergence in hard maze as a function of alpha

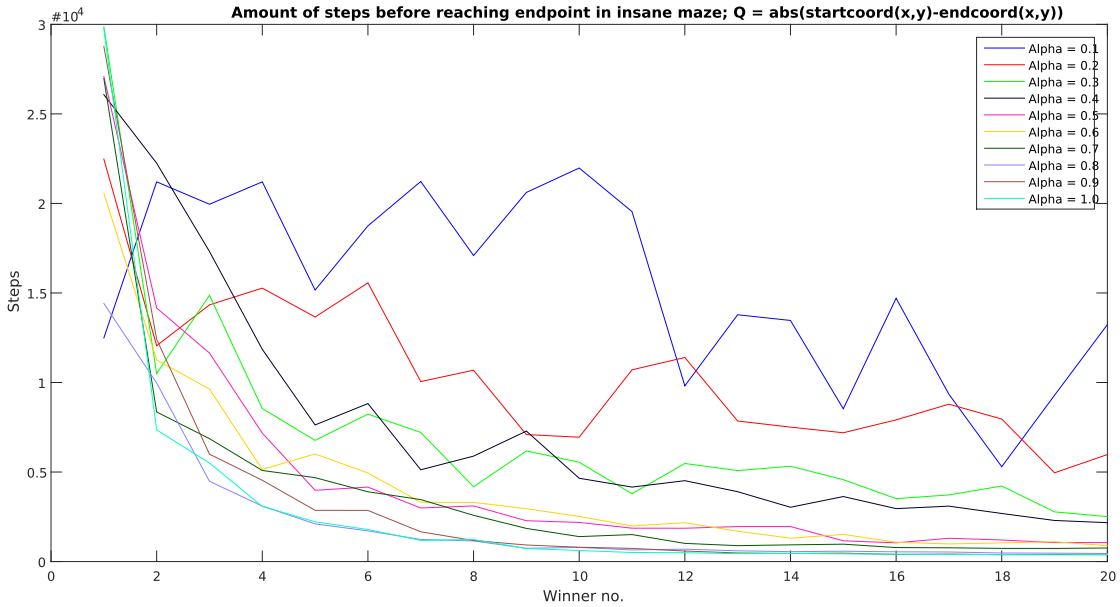

Figure 10: Convergence in insane maze as a function of alpha

From the graphs is is shown that the value of alpha should be 1 for all mazes. All values different from 1 show no improvement in amount of steps nor improvement in convergence.

At last the procedure is repeated for beta. Because adjusting Beta, only resulted in small differences, the convergence was not significantly noticeable so the data is plotted in a different way. For all 100 ants that were released, the amount of steps of the best ant is plotted in a stem plot. In theory the beta value which results in the least steps, is the best. Because of random variations, the average is took over 2 times (more times would be better but unfortunately it takes to much time to accomplish).

The results are show in figure 11, 12, 13 and 14.

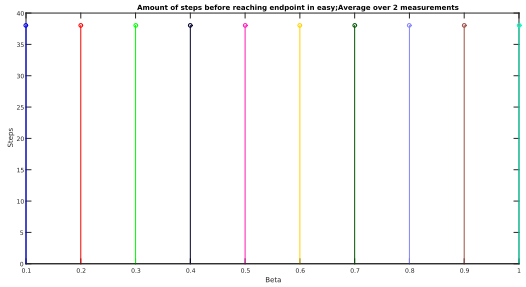

Figure 11: Minimum amount of steps in easy maze as a function of beta

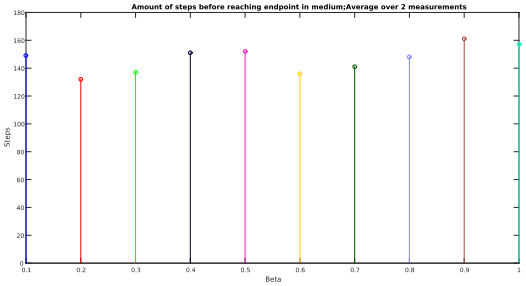

Figure 12: Minimum amount of steps in medium maze as a function of beta

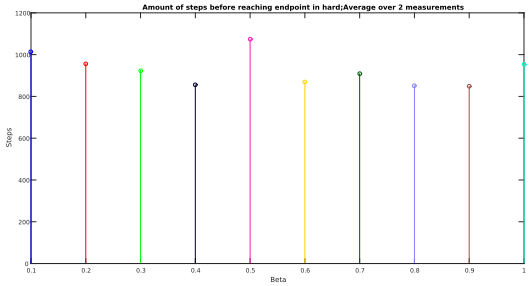

Figure 13: Minimum amount of steps in hard maze as a function of beta

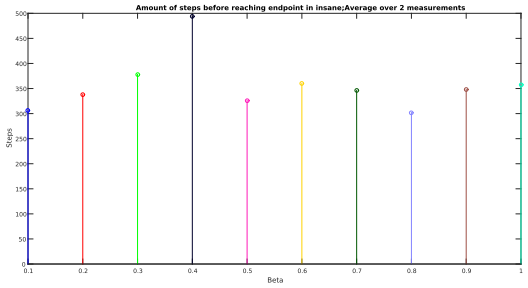

Figure 14: Convergence in insane maze as a function of beta

The following values for Beta are deduced from the graphs:

- Easy - doesn't have any effect therefore a random value is chosen: 0.5
- Medium - 0.6
- Hard - 0.9
- Insane - 0.8

7

- Parameter Q:

  The parameter Q really depends on both the complexity and the size of the maze, which is obvious because Q is really just an estimation of the route length and bigger and more complex mazes tend to have longer routes. Because the initial value of Q without multiplier takes into account both start- and end-coordinate, the ACO algorithm should work for different coordinates in the same maze too which is an important property for the grand challenge.

- Parameter Alpha:

  The parameter Alpha does not depend on the complexity nor size of the maze. The graphs show that a value other than 1 does not result in fewer steps and or faster convergence.

- Parameter Beta:

  The parameter Beta did depend on the complexity and size of the maze. In the easy maze, beta didn't have any effect but on the other mazes it had some noticeable effect. Visualizing the route in the visualizer, for example in the medium maze, different values for beta resulted in two different routes, a route clockwise and anticlockwise. Using the graphs beta values were deduced for all the mazes. The amount of data is to small (due to long execution times) to really draw any conclusion. Moreover the value Beta could depend on the start- and end coordinates which were not taken into account.

6 Problem Analysis

8

Given a list of nodes and distances between those nodes, what is the shortest possible path that visits each node exactly once and ends on the starting node.

9

Initially we don't know the distances of paths between the nodes. Also the paths are not symmetric. To be able to apply TSP, we let the ACO algorithm precompute all the paths from every product/node to every product/node and store the routes and path lengths. Using only the path lengths for the TSP algorithm, the problem can be solved like it is a classic TSP problem.

10

Exact algorithms take a lot of computational time, a heuristic approach like Computational Intelligence techniques might not give the exact minimum but provide a slightly less optimal solution in a much shorter time.

7 A Genetic Algorithm

11

Each chromosome contains 18 genes, with each gene representing one of the 18 products. For example a chromosome could have the following gene configuration: [9, 14, 4, 16, 1, 3, 2, 4, 6, 18, 17, 15, 5, 7, 8, 10, 11, 12, 13]. Because our chromosomes have a static length of 18, no duplicate genes are allowed in our algorithm.

12

A chromosomes fitness will be inversely proportional to the length of the path it contains: 1/[path length]. So the shorter the path length the higher the fitness. The path length here is the total number of actions it takes to take all the products in the order specified in the genes.

13

Using a roulette wheel implementation, chromosomes are selected for cross-over. The higher it's fitness the higher the probability it will be selected for cross-over.

14

We used two genetic operations, mutation and cross-over. In mutation two elements from the genes are switched randomly i.e. [9, 14, 4, 16] becomes [9, 16, 4, 14]. In cross-over two chromosomes are selected, using the roulette wheel, to create a new chromosome for the next generation. The two chromosomes are compared and for each element i in the genes ArrayList there is a .5 chance of getting the value from either parent. Because this can sometimes result in one product being in the list twice (and therefore one product not being in the list since there are always 17 elements) we implement a check for this that chances one of the doubles into the missing product.

15

Because as stated in 14 during the cross-over, the new chromosome is only allowed to contain a route, containing no duplicate products and because the algorithm tries to optimize the configuration of the products resulting in a short as possible route, the algorithm should in theory avoid visiting points twice. But because all the routes from each node to all other nodes are pre-computed, point being visited multiple times can still occur.

16

Local minima are avoided due to mutations in the current generation. Each chromosome in the current generation has a probability of lets say 0.1 that it is going to mutate. During a mutation it can occur that one of the chromosomes receive a high or higher fitness than the other chromosomes in the generation because it lands in another maybe higher local minima. Than this chromosome will have a higher chance of crossing over and thus has a higher chance to influence the next generation. Thus by tweaking the mutation probability value local minima can be avoided.

17

Elitism is avoided by the roulette wheel of choosing the 'to cross-over' chromosomes. Although the better chromosomes have a higher chance of crossing-over, it is not guaranteed that the best chromosomes will cross-over. Therefore less fit chromosomes also have a change to influence the next generation.

References

[1] M. Negnevitsky, *Artificial Intelligence A Guide to Intelligent Systems, Second Edition*, Pearson Education, Harlow, 2005.