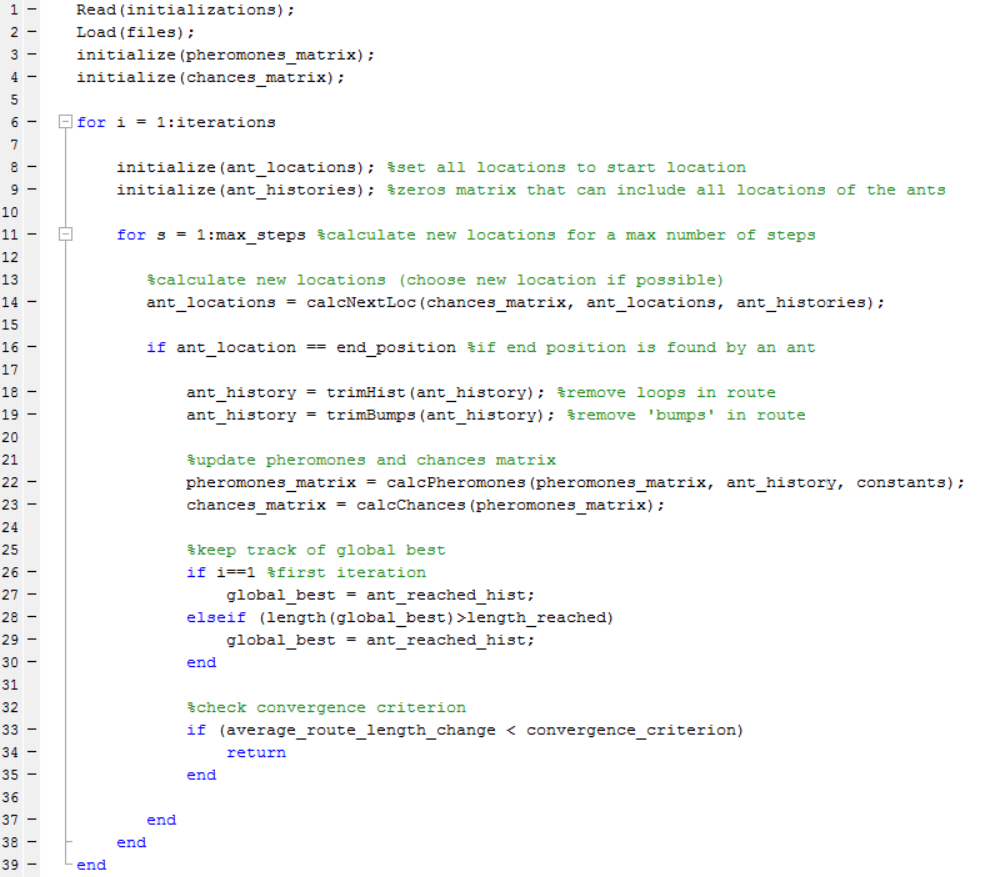
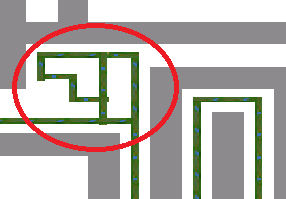
Assignment 3

1. (large) open areas, irregular paths (open areas with lots of isles or straight paths with inlets).
2. Formula for the amount of dropped pheromone:  
   Pheromone is dropped to mark routes that are frequently visited. Shorter routes get more pheromone added because Q is divided by a smaller length . Therefore, over time, high concentrations of pheromone will mark a short route from A to B. However, local optima are possible, so it is not always the shortest route that is marked.
3. Formula for the amount of pheromone on the path from ‘i’ to ‘j’. ‘k’ represents an ant, ‘m’ is the total amount of ants:  
    is the evaporation constant. With every iteration, a factor of the previous amount of pheromone is used in the following amount of pheromone. The evaporation constant is meant to introduce a certain amount of ‘forgetting’ in the algorithm. This will slowly undo pheromone on a route that turns out to be too long.
4. The figure below contains the pseudo code of our main function. Please note that this version of the pseudo code is already the version with ‘special features’, these features will be explained in the next exercise. The ants simulated in this code only drop pheromone on the route they passed when they find a path to the end (unlike ‘real’ ants, which always drop pheromone). This reduces the amount of iterations because only the quickest route found in an iteration is marked.

Figure 1: Pseudo-code of the main algorithm



1. The algorithm in the previous exercise was then equipped with three main features.  
   The first involved the *calcNextLoc* function to also look at the history of an ant, and choose randomly (based on the chance of taking that route, of course) only between all new possibilities. If there are no new locations available, the algorithm will simply pick at random (again, based on the chance of picking that route). This feature improves the exploring abilities of the ants, while also enabling them to keep going forward when confronted with long hallways, for example.  
   The second feature was the function *trimHist*, this function is used on the history of an ant that has managed to find a route to the end. This function loops through all the locations an ant has been, starting at the first history element. In every loop, the function checks for the next duplicate of this location, and then deletes all the locations up to and including the found duplicate. This feature removes all loops (see Figure 2), including dead-ends, making the final route of an ant shorter. Quick tests show that the route usually becomes over 60% shorter using this function, with some routes becoming over 90% shorter.  
   The third feature is a function called *trimBumps*. This function is an alternative form of *trimHist*, and looks for the next duplicate of all the neighbouring locations. If a duplicate is found, the locations in between are removed. The result is a much smoother route. Figure 3 shows an example of a bump removed by *trimBumps*.



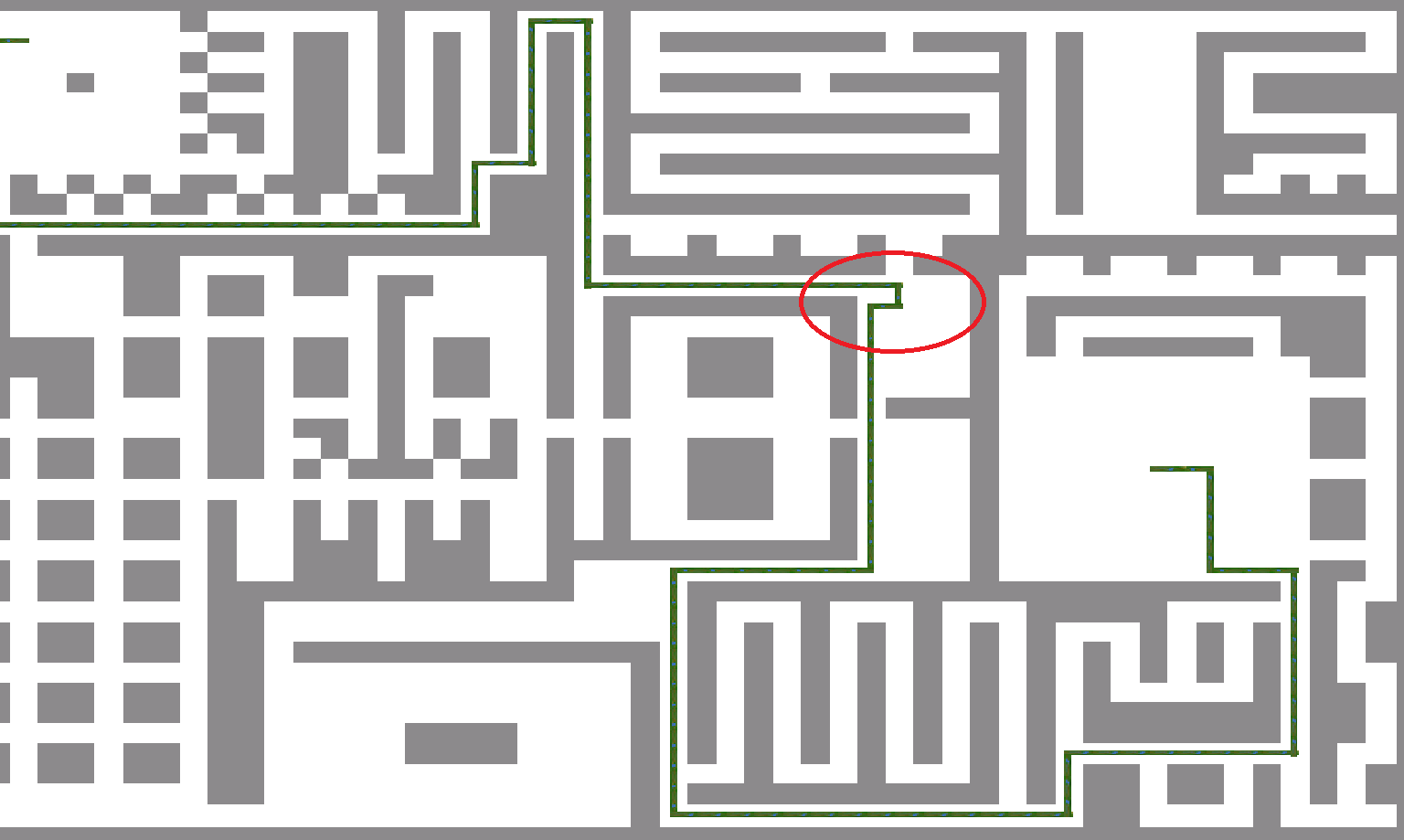


Figure 3: a typical bump removed by *trimBumps*

Figure 2: a typical loop removed by *trimHist*

By including *trimHist* and *trimBumps* in every iteration, the code performance was improved, decreasing calculation times in the medium maze by almost 30% on average. The functions also had a small improvement on the final route length, decreasing the medium routes by roughly 4 steps on average. Performance on the calculation of new locations was not measured, because there was too little time to make and measure a ‘basic’ location picker.

Figure 4: performance increases as result of the trim functions

1. Using rough estimations (10 ants, 100 pheromones, 0.1 evaporation), the code was run on the easy maze first. The results of 5 tests can be found in Table 1. On average, the calculation took less than 2.4 seconds, and every calculation resulted in the shortest route from begin to end, which was 39 steps. Since the algorithm always found the fastest route in a relatively short time, the variables were considered ‘decent’ for this maze, and were therefore not altered any further.

Table 1: results of the easy maze

|  |  |  |  |
| --- | --- | --- | --- |
| Test # | Time (s) | Iterations | Length |
| 1 | 2,850 | 17 | 39 |
| 2 | 1,854 | 13 | 39 |
| 3 | 2,585 | 14 | 39 |
| 4 | 1,993 | 14 | 39 |
| 5 | 2,593 | 18 | 39 |
| AVG | 2,375 | 15 | 39 |

The code was then tested on the medium level maze. Results on this maze can be found in Figure 5, Figure 6 and Figure 7. All measurements are an average of 5 tests. From these results it was concluded that the following variables were considered ‘good’: 10 ants, 300 pheromones per iteration and evaporation 0.05.

Figure 5: performance on the medium maze varying in the amount of ants (300 pheromones, 0.1 evaporation)

Figure 6: performance on the medium maze varying in the amounts of dropped pheromone per iteration (10 ants, 0.1 evaporation)

Figure 7: performance on the medium maze with different evaporation constants (10 ants, 300 pheromones)

The code was then tested on the hard level maze. Results on this maze can be found in Figure 8, Figure 9 and Figure 10. All measurements are an average of 3 tests. From these results it was concluded that the following variables were considered ‘good’: 10 ants, 400 pheromones per iteration and evaporation 0.1. An evaporation constant of 0.2 and 800 pheromones was also considered ‘good’ for faster calculations (These results can be found in Figure 10).

Figure 8: performance on the hard maze varying in number of ants (800 pheromones, 0.1 evaporation)

Figure 9: performance on the hard maze varying in the amounts of dropped pheromone per iteration (20 ants, 0.1 evaporation)

Figure 10: performance on the hard maze with different evaporation constants (20 ants, 800 pheromones)

The code was then tested on the INSANE level maze. Results on this maze can be found in Figure 11, Figure 12 and Figure 13. All measurements are an average of 5 tests. From these results it was concluded that the following variables were considered ‘good’: 20 ants, 500 pheromones per iteration and evaporation 0.05.

Figure 11: performance on the INSANE maze varying in number of ants

Figure 12: performance on the INSANE maze varying in number of dropped pheromones

Figure 13: performance on the INSANE maze with different evaporation constants

In the end, the final length of a route was not very dependent on the number of ants in any maze. The calculation time, however, was. 10 ants was normally sufficient to solve a maze in an acceptable time. Fewer ants either meant a long final route or a long calculation time, and more ants usually meant a much longer calculation time. The amount of pheromones had a similar effect, too few pheromones meant a long calculation time or even no convergence, too much pheromone resulted in fast calculations but also longer routes. We found that a good rough estimation for the amount of pheromone was as follows: . Finally, a higher evaporation generally decreased the calculation time, but at the cost of the length of the fastest route.

1. Although the code seemed quite robust at converging decently with different parameters, choosing the right parameters for the fastest calculation time and/or the best route proved to be quite tricky. Picking the wrong parameters can mean that the code does not converge (for example: 20 ants, 100 pheromones and 0.1 evaporation on hard does not converge) or it can mean a huge increase in the calculation time (Figure 8, for example). In general, getting results is not too difficult. Getting the best results as quickly as possible, however, is difficult. ‘decent’ parameters were obtained more quickly on less complex mazes, because of relatively short calculation times. However, the dependency of the parameters on calculation time and final route length seems more or less equal across the mazes medium and higher.
2. The TSP is a problem of a salesman trying to visit multiple cities or ‘nodes’ while traveling the least amount of distance. A good example of an answer to a TSP can be found in Figure 14. This is the shortest route to visit all the reached German cities and returning to the starting city (which can be any of these cities).

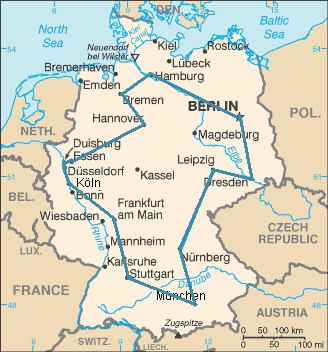


Figure 14: example of an answer to a TSP

1. The nodes to reach are connected to each other by multiple routes of different lengths. The distances are also unknown.
2. Because there are too many ways to connect these nodes. It is simply too much calculation to do it either by hand or by random calculations. The calculation needs to be more intelligent for it to solve the problem in a reasonable time.
3. A gene will be a number between 1 and 20. Number 19 and 20 are reserved for the starting and ending point, and will always be the first and last gene respectively. Chromosomes are 20 genes long and consist of all numbers between 1 and 20. This ensures that nodes are visited once and only once.
4. The fitness will be 1 divided by the length of the route. A shorter length means a higher fitness which would be a better result.
5. This algorithm will use a promotional selection to stimulate convergence while also trying to prevent local minima.
6. The TSP code will have both cross-over and mutation. Cross-over is relatively complex, because we want the children to still include all nodes. This is accomplished by ‘fixing’ the children, which involves changing values of genes outside of the cross-over area. If there is a duplicate number outside of the cross-over area, this value will change to the value the corresponding value in the cross-over area used to be before the cross-over. For example:

If we cross-over element 4 through 6, we get:

Replacing duplicates outside the cross-over area results in the following children:

This results in new duplicates, so we do the fix again:

And again:

Which now includes only one of every number.

For mutation, the algorithm simply picks two values at random and then switches these values.

1. By limiting the chromosome length to 20 and having them include all numbers once and only once ensures us that the route visits each node only once. The cross-over and mutation function also ensure that this criterion is kept (see previous exercise).
2. Local minima are prevented by including mutation and promotional selection. Cross-over can also have a positive effect on preventing local minima.
3. Elitism ensures that the best route is always kept in the population, unaltered. It ensures that the solution quality does not decrease from one generation to the next. The code of this group did not implement elitism in the code, because it was thought that it would hinder exploration. A global best, however, is kept in memory to ensure that the best route is always remembered.