Internal Report 2012-06 August 2012

Universiteit Leiden
Opleiding Informatica

Routing with Ant Colony
Algorithms in
Continuous Spaces

Jan-Paul van Osta

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

This paper is about ant colony optimization in continuous spaces. Ant colony algorithms are based on real world ants, this idea will be explained, as well as the discrete ant colony algorithms. Based on these ideas is the actual algorithm this paper describes. While describing the algorithm different topics will be discussed. The Kernel Density function is essential for the use of pheromones. The actual path finding will be discussed as well as the evaporation of the pheromones and the evaluation of the ants. Another important topic of the algorithm is momentum. Parameters for the algorithm have been produced and the algorithm was tested. Results will be presented in this paper.

Contents

1 Introduction

1.1 Real-world ants

Multiple optimization algorithms have been based on ant colonies. One ant on its own cannot achieve much, but a whole colony of ants can perform relatively complex tasks. This is why ant colony optimization is an interesting natural computing topic. To understand how ant colony optimization algorithms usually work it is important to first know how real-world ants do their job. An essential observation in this context is that ants communicate indirectly by leaving pheromones on their path. Other ants can follow these paths and eventually most ants will walk the strongest path.

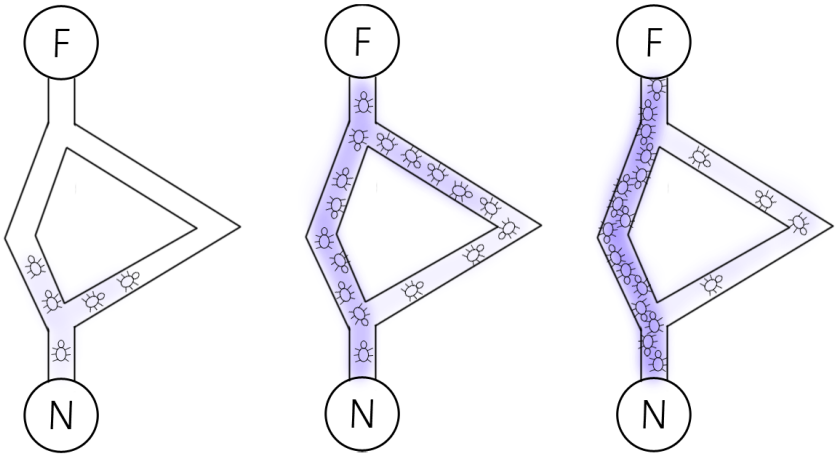

Figure 1: Ants walk a path from the nest (N) to the food (F) and back. More and more pheromones build up on the shortest path.

As is shown in figure 1 the ants have two possible ways to reach the food when starting at the nest. N is the nest, F is the food and the blue color represents the intensity of the pheromones. On the most left picture we see the ants start walking. Because there are no pheromones yet and the ants can't see which route is shortest the decision will be random. The chance of an ant taking the left route will be as big as it taking the right route. While walking back and forth on these paths the ants will leave their pheromones. Since the ants walking the shortest path will take less time to walk back and forth, the shortest path will be used more frequently. Eventually there will be more pheromones on the shortest path. This will make the chance of an ant choosing this path larger. Eventually most ants will choose this shortest path and an efficient path from the nest to the food is found.

1.2 Discrete ACO

As said before multiple ant colony optimization, or ACO, algorithms are based on ant colonies. These ant colony algorithms are mainly applied to discrete path finding problems. In this case solutions are sequences of vertices from a finite vertex set V. [1] Before explaining how to apply ACO on a continuous problem it is important to see how the discrete algorithms work. A basic structure of an Ant Colony Algorithm can be seen in algorithm 1.

Algorithm 1 A basic structure of an Ant Colony Algorithm. (based on fig. 2 of [3].)

```
initialize
while not terminate do
  send ants
  evaluate ants
  evaporate pheromones
  update new pheromones
end while
```

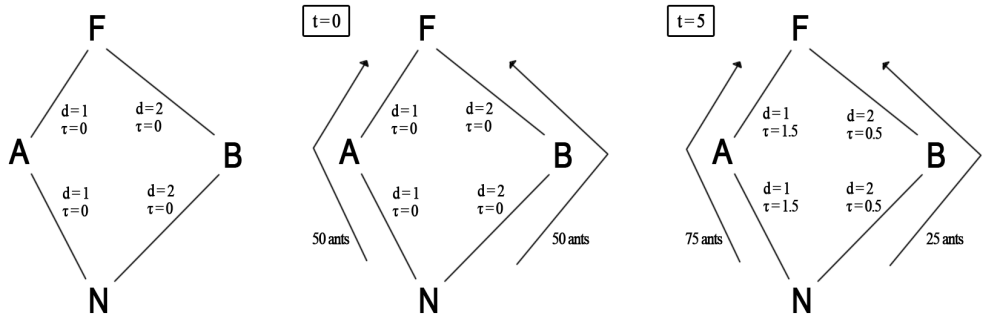

Figure 2: A simple ACO problem visualized in a graph.

In figure 2 we see the same problem as in figure 1, this time visualized as a graph. Where d is the value of the distances and where τ represents the pheromone values. N is the nest, F is the food and A and B are the points on the paths within the graph. A 100 ants will walk these paths. At $t = 0$ we see the pheromone values are all equal, the values are all 0. Because of this the ants will choose a path at random. An average of 50% will choose path NAF and another 50% will choose path NBF. After a number of rounds, say at $t = 5$ we will see the pheromone levels have shifted. The ants are more likely to choose the shortest path. Pheromones in a discrete ACO algorithm can be placed in different ways. For instance an elitist ant system where the best found solution places more pheromones. [7] Or a rank-based ant system where ants with a better fitness (shorter route) leave higher amounts of pheromones. [7] According to [1] the differences between real-world ants and artificial ants are:

- artificial ants will have some memory,
- they will not be completely blind,
- they will live in an environment where time is discrete.

1.3 Continuous ACO

This research evolves around an ACO algorithm designed for a continuous plane. An important difference between these continuous problems and discrete problems is that there is an infinite number of possible solutions for it. Not as much research has been done for continuous ACO algorithms as for discrete ACO algorithms. Most ACO algorithms for continuous problems are designed for minimization or maximization a certain function or for finding local optima.

The algorithms in [3] and [4] are designed for finding optima on a continuous plane. This algorithm is based on a number of starting vectors. Ants will walk these vectors and look at its fitness for each step. Paths that result in improvement will leave pheromones. The vectors will be adjusted until ending conditions are met.

The algorithm in [2] is designed for an objective function $f : S \to \mathbb{R}_0^+$ to be minimized on a search space S defined over a finite set of continuous decision variables and a set Ω of constraints among variables. This is also a function optimize (minimize) algorithm. This algorithm uses the Kernel Density Estimation to look at the density of the pheromones. The algorithm described in this paper does this too.

2 The algorithm

Algorithm 2 Continuous routing ACO algorithm

```
initialize plane
initialize colony
while not terminate do
  for every ant do
    while not found food do
      select a number of random possible steps
      evaluate pheromone values and momentum for steps
      choose and do step
      store route so far
    end while
    calculate route lenght
  end for
  for every pheromone do
    evaporate
    if evaporated then
      destroy pheromone
    end if
  end for
  for every ant do
    evaluate route
  end for
  place new pheromones
end while
```

2.1 The main concept

Algorithm 2 describes the algorithm used in this research. This algorithm has certain similarities with the discrete ACO algorithms mentioned before. One important similarity is that the algorithm will work with a random factor. Both in this algorithm and in the discrete ACO algorithm there is a probability an ant will choose a certain path. Real-world ants will start walking around randomly until they find pheromones or food. The ants

in this algorithm will start walking in a random direction too and will have a preference if pheromones are nearby, just like the real ants. After going their way, following any pheromones, they will reach the food. The path is stored and remembered and the length is measured. When all ants have walked the path evaporation will take place, the pheromones which have been around for too long will disappear. After this a number of ants with the best fitness, the shortest path, will leave new pheromones on their path. Discrete ACO algorithms have their pheromone values stored in every line (or point) of the graph, since this algorithm is for a continuous plane, this will not be possible. Every pheromone will have a coordinate. As in [2] the Kernel Density Estimation function is used for measuring these points. However this algorithm will use the Kernel Density Estimation to help path finding, not to optimize a function.

2.2 The Kernel Density Estimation

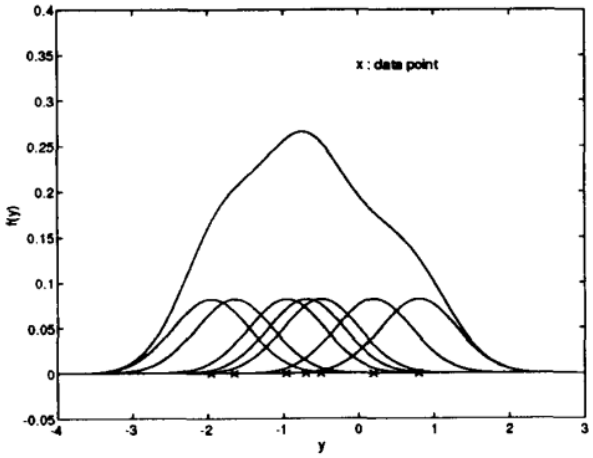

Figure 3: A KDE example: multiple points form one smooth density function. [6]

The Kernel Density Estimation, or KDE, is a function which can turn collections of different points into one smooth function. The more data points there are in the vincinity of a point in a certain area the higher the function value at this point. An example using Gaussian kernels for points is given in Figure 3. This is also possible for three dimensions. As described in [5] and [6] the Kernel Density Estimation formula looks like equation 1.

$$f(x) = \frac{1}{n} \sum_{i=1}^{n} K\left(\frac{x - x(i)}{h}\right) \tag{1}$$

Where n is the number of points, in this case pheromones. x is the estimated point and $x(i)$ is one of the data points, or pheromones. This means $x - x(i)$ is the difference between these two points. Since this value will later be squared, it can also be seen as simply the distance between the two points. h is a parameter which will decide on the smoothing of the curves. The higher h is, the smoother the curves will be. K is the chosen function for the Kernel Density Estimation. Different formulas for K can be chosen, like triangle, cosinus or Gaussian kernel. This algorithm uses the Gaussian formula, which is equation 2. Where u is the input as shown in the KDE formula. (equation 1) So when h is 1 this

input will simply be the distance between the different data points and the estimated point.

$$f(x) = \frac{1}{\sqrt{2\pi}} exp(-\frac{1}{2}u^2) \tag{2}$$

In this algorithm there are two differences from the common KDE. The ants are dealing with obstacles. This is why every KDE value calculated on the plane will only use the pheromones which are visible in a straight line from that point, with no obstacles blocking the view. Also, only pheromones within a $3h$ radius are considered, for complexity reasons. Nearest-neighbor was considered, but this would have made the algorithm even more complex. Despite of these differences equation 1 still stands.

2.3 Momentum

A discrete ACO algorithm can state to never visit one line twice. This will prevent the path from getting unnecessarily long. Because our continuous problems have an infinite amount of possible solutions this is not the case. This is why the algorithm has a momentum factor. This will make the ant less likely to turn around when the path ahead is not blocked. The chance of choosing a possible next step will be multiplied with this momentum factor. The formula for this momentum factor is seen in equation 3. Where a is the difference between the original direction and the new one. This is a value between 0 and 2π. The original direction is the average position of the last steps. And the constant b is the momentum multiplier. This is a parameter. The other constants combined with the min and max are there so the is no difference between straight forward (or backward) and slightly forward (or backward), thus allowing it more to still make some turns.

$$f(a) = 1.0 + \min(\max((a - 0.5), 0.0), 2.2) * b \tag{3}$$

2.4 The path of the ant

Every round every ant will try to find its way to the target, or food. An ant will always start walking from the nest. From there it will try to reach the target. The ant will make a variable amount of vectors (this is a parameter) as possible choices to walk. These vectors will be random, based on a random length and random direction. This length is a number in between two parameters. The random direction will be between 0 and 2π. These vectors have a random direction and length so every possible position on the plane can be reached. After making these vectors the ant will have to choose which one to use as a next step. This will be random, however the vectors with a high KDE value and a high momentum factor at the point of their destination will have a bigger probability to be chosen. This decision will be done using roulette selection. Also a small value between 0 and 1 will be added to every KDE value, so a point will never have a 0 probability to be chosen. This value will be a parameter. The only time a point can not be chosen is when there will be an obstacle or border in its way. When no possible new steps are found the ant will stay in its current position. These described steps will be taken for a number of times. Taking new steps will continue until the ant reaches the target or it has reaches the maximum number of steps. This number is a parameter. When the ant reaches his target, it will temporarily store its path into its memory and the next ant will start moving. A

variable amount (this is a parameter) of ants will move every round. After these are done, they will be evaluated.

2.5 Evaluation, updating and evaporation

A variable amount of ants will be the best, or elite, ants. This amount can be anywhere between one and all of the ants. These will be the ants with the best fitness, so the shortest route. Only these ants will leave pheromones on their path. Every ant will drop the same amount of pheromones, this number is a parameter. Because of this shorter paths will have a higher concentration of pheromones. These dropped pheromones will have an x, y and duration variable. This duration variable will decrease every round. When this variable is 0 the pheromone will evaporate and will no longer exist. This means it will not be used anymore in the KDE calculation. The initial duration variable value is a parameter. These new pheromones will help the next generation of ants find their way to the target.

2.6 Optimization

Algorithm 3 Optimization of a path

```
for every step (i) in path do
  for every previous step (j) do
    if no obstacles between i and j then
      for every step in between i and j do
        make that step the same as step i
      end for
    end if
  end for
end for
for every step in path do
  if step is the same as previous step then
    remove step
  end if
end for
recalculate path length
```

Since the ants are walking back and forth in a semi-random manner, they will not necessarily walk in the most efficient way. Especially when dealing with short step lengths it is very probable there will be some unnecessary turns in an ant's path. By cutting away these unnecessary steps we can improve the path's fitness. This concept is also seen in [3] and [4] and can be seen in figure 4. By removing steps from the path that will only make it longer, we can make the entire path shorter. This algorithm is described in algorithm 3.

2.7 Multiple targets

The algorithm described so far is based on a path finding problem with one starting point and one target. It is also possible to have a problem involving multiple targets. In algorithm 2 the line "while not found food" should be changed to "while not found

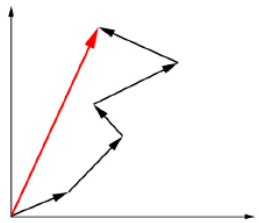

Figure 4: Making multiple vectors into one will improve the ant's fitness. [4]

all food". Also, it should be made sure the same food is not found twice. These two adjustments make it possible for an ant to make a path visiting multiple targets.

3 Methods

The previously described algorithm was implemented using C++. OpenGL and GLUT libraries were used to visualize the continuous plane. A class was made for the plane, containing a nest, a target, a width and height, pheromones and ants. For storing the pheromones a linked list class was made. For the ants another class was made, containing x and y coordinates and a memory to store its path. This piece of software was used to test the algorithm by using different parameters and planes. The results are shown in the next chapter.

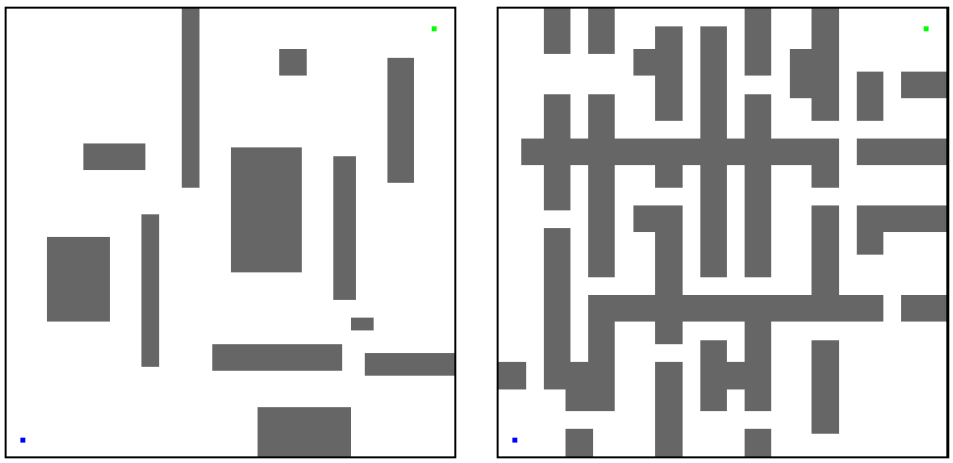

Figure 5: These planes were used. Left: Blocks, Right: Maze

4 Results

The algorithm has been tested on two different planes, as shown in figure 5. One relatively simple problem, referred to as "blocks", and one more complex problem, referred to as "maze" can be seen. As seen in figure 6 the ants left a pheromone trail and found routes to the target.

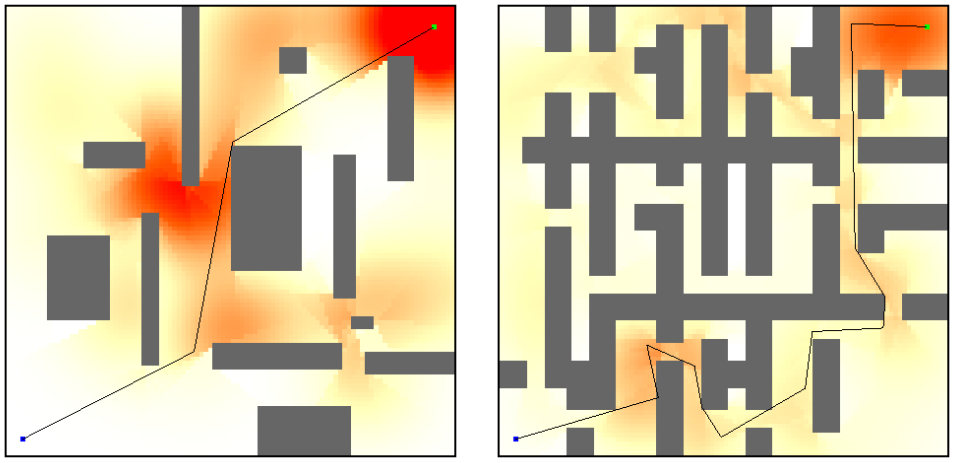

Figure 6: Paths found with the help of pheromones.

4.1 Setting the parameters

Before testing a number of parameters had to be set. Initial parameters were made by doing tests on some rough parameter values. This was done to test what value every parameter should approximately have. The output of these values were evaluated on the best found solution and the average number of steps for every ant. The initial parameters can by found in table 7. These parameter values were used in the fine tuning of the parameters. The process of the fine tuning was as follows: The blocks plane was tested with the parameters from table 7 (initial), but with one of the parameters changing. For every different value this was done ten times using six iterations of ants. The parameters were evaluated on the average number of steps. This because it says something about every ant, it will tell how fast an ant can reach its target. Also, it is harder to evaluate on path length, since every best found path turned out to be relatively good after a number of iterations, with too less of a clear difference too evaluate.

In table 11 we see the results of "pheromone dropped per ant" parameter. With an average of 158.95 steps fourteen is the best value for this parameter. h is a parameter of the KDE function. In table 12 we can see that 0.75 performed slightly better than 1.0. Not that this will differ when dealing with planes of different sizes. In table 13 we can see the duration before evaporation of the pheromones did not have a very large influence on the results. However, 4 had the best results. As seen in table 8 64 directions had the best results. However, the difference with 16 to 45 is not notably large. For complexity reasons the parameter value will stay 32. In table 9 we see the performance for each number of ants per iteration. We see that the more ants are used, the better the performance. For complexity reasons this amount will stay at 20. In table 10 we see the results for different percentages of elite ants. This is the percentage of (best) ants which will leave pheromones. 90% had the best result. (90% is in this case of course 18 ants, since 20 ants were used) The supplement KDE is the value added to the KDE value so there is never a chance of 0 to reach a certain point. As we can see in table 14 the smallest value is the best choice. The momentum multiplier and the KDE multiplier are factors which improve the influence of the momentum and the KDE value respectively. In table 15 we can see

the best value for the momentum multiplier is 1.75. In table 16 we can see the best value for the KDE multiplier is 2.0. The parameters for minimum and maximum step length turned out to be optimal when as low as possible and as high as possible, respectively. However, average number of steps might not be a good evaluator for these parameters, since good solutions will be harder to find when taking steps which are too big. In table 7 the new optimized parameters are found. Note that different parameter values may have influence over the performance of other parameters. Also note that these parameters are not perfect and might perform different on other planes. However, these approximations do work for the earlier mentioned planes.

4.2 Evaluation the algorithm

To test the performance of the algorithm a number of tests were performed for every plane. The algorithm was tested against an equally large group of ants which walked randomly. These random ants had the same step lengths as the algorithm ants. The results of the algorithm can be found in table 1 and table 3. The results of the random ants are shown in table 2 and table 4. Table 5 and table 6 show the differences between the random and algorithmic ants. The table also shows the standard deviations of the result, to get a clearer look at the severeness of these differences. From these six tables we can see that the algorithm does come with better results than random walking ants. The random ants only perform slightly better before optimization on the blocks plane. On the more complex maze plane however, the algorithm performs definitely better. On both planes the algorithm performs clearly better after optimization. We see a difference of a few times the standard deviations. The biggest difference however is seen in the most right columns, the average number of steps. The random ants can come up with reasonably working solutions after a few iterations, but the average found solution still has a lot more steps than the average found solution when using the algorithm. It is clear that the algorithmic ants use a lot less steps to come to a solution.

4.3 Momentum vs. Pheromones

An interesting observation was the large improvement of the algorithm's performance when the momentum was introduced. This raised the question whether momentum or the (KDE) pheromones have the biggest impact on the algorithms performance. This was tested by lowering the momentum factor to 0.1 and testing the algorithm ten times on both planes. The same was done with the KDE multiplier. table 17 shows the results (average number of steps) in comparison with the regular (using both) results. We see the momentum obviously has the bigger impact. However, the difference with the impact of pheromones is smaller when looking at the more complex maze. We can conclude that the pheromones become more important when path finding within a more complex problem. Eventually we can also see the best result is achieved when using both methods.

5 Conclusion

5.1 Conclusion of the results

As could be seen in the results the algorithm performed better than random agents trying to solve the same problem. The algorithm found paths of an average length of 74.7% and 85.4% of the lengths found by the random agents on the blocks and maze plane respectively. The average amount of steps was a 14.9% and 28.6% of the average amount of steps done by the random agents, on the blocks and maze plane respectively. By using indirect communication with pheromones the ants could find solutions to path finding problems. Also, global parameters were found to maximize the results. The ants were given momentum which proved to be useful. The momentum brought the average amount of steps back to around 15.4% to 37.5% of the amount without momentum, for blocks and maze respectively. This paper has proven that this continuous ant colony optimization algorithm can be used to solve path finding problems on a continuous plane, since it can find solutions. Existing continuous ACO algorithms either did not use Kernel Density or were not designed for path finding. The algorithm has proven to be smarter than random ants, however it should be considered that it does not necessarily find its solutions faster, due to high complexity. It has also not been tested against existing path finding algorithms, which might perform better. For instance, [8] describes an algorithm for routing in polygonal spaces which connects the corners of obstacles. This could be used in both example planes as well.

5.2 Future research

Continuous ACO algorithms are still an unexplored field. There is still a lot to research to be done in this domain. The next step in this algorithm would be to go further than just obstacles. It could be possible to make areas with resistance which will make the traveled distance longer. We could compare this with hills or rough terrain. It would also be interesting to experiment with anti-pheromones, which will lower the chance of an ant moving to a certain area, like an area near a wall, or a dead end.

References

[1] Dorigo M, Maniezzo V, Colorni A. The Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and CyberneticsPart B, Vol.26, No.1, pp.1-13. 1996.

[2] Socha K, Dorigo M. Ant colony optimization for continuous domains. IRIDIA, Universite Libre de Bruxelles, CP 194/6, Ave. Franklin D. Roosevelt 50, 1050 Brussels, Belgium. 2006.

[3] Bilchev, G., Parmee, I.C. The Ant Colony Metaphor for Searching Continuous Design Spaces. Lecture Notes in Computer Science 993. Plymouth. 1995.

[4] Kuhn L. Ant Colony Optimization for Continuous Spaces. PhD thesis, The Department of Information Technology and Electrical Engineering, The University of Queensland, October 2002.

[5] Kernel Density Estimators
http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0405/MISHRA/kde.html

[6] Hwang J, Lay S, Lippman A. Nonparametric multivariate density estimation: a comparative study. IEEE Transactions of Signal Processing, Vol.42, No.10, October 1994.

[7] Raghavendra GS, Prasanna Kumar N. Statistical Approach for Selecting Elite Ants. Annals. Computer Science Series. 9th Tome 2nd Fasc. 2011.

[8] de Berg M, van Kreveld M, Overmars M, Schwarzkopf O. Computational Geometry. Springer: Berlin, 2000.

before optimisation		after optimisation		average nr of steps
path length	nr of steps	path length	nr of steps	
22,06	49	9.9	5	127.13
74,46	148	9.34	3	155.9
19,86	39	8.51	3	203.77
21,62	46	8.5	3	165.93
119,67	230	9.9	3	182.07
332,67	663	9.11	3	157.27
208,01	431	8.97	3	183.3
47,62	92	8.92	3	161.97
137,22	271	9.12	3	199.4
164,32	346	9.29	3	156.9
114,751	231.5	9.156	3.2	169.364

Table 1: Blocks: non-random

before optimisation		after optimisation		average nr of steps
path length	nr of steps	path length	nr of steps	
76,65	133	14.65	5	1049.33
90,89	158	12.31	5	1160.23
215,79	397	13.24	3	1021
99,71	178	12.11	4	1013.93
71,15	131	13.22	4	1103.33
84,36	160	10.7	5	1265.27
62,12	114	11.23	3	1155.47
48,54	89	13.04	3	1270.26
41,14	74	12.14	3	1096
101,35	193	9.94	5	1213.17
89,17	162.7	12.258	4	1134.799

Table 2: Blocks: random

before optimisation		after optimisation		average nr of steps
path length	nr of steps	path length	nr of steps	
404,9	1038	18.84	12	593.5
49,26	113	19.14	9	722.1
122,63	317	18.45	9	758.9
88,75	208	18.45	10	664.3
94,12	218	19.09	11	488.1
49,54	110	19.18	11	575
293,44	666	19.12	11	862.9
261,15	642	20.08	10	764
48,5	132	18.64	10	730.8
226,69	526	19.16	10	601.3
163,898	397	19.015	10.3	676.09

Table 3: Maze: non-random

before optimisation		after optimisation		average nr of steps
path length	nr of steps	path length	nr of steps	
313,66	739	23.3	14	2397.5
326,45	644	20.57	10	2559
349,95	801	20.32	11	1881.4
172,1	362	24.16	13	2345.1
167,02	384	21.4	10	2289.7
244,35	554	22.02	14	2370
151,05	323	21.96	11	1902.4
240	531	24.62	14	2525.2
166,81	373	20.33	12	2702.1
171,61	369	23.96	17	2667.9
230,3	508	22.264	12,6	2364.03

Table 4: Maze: random

	before optimisation		after optimisation		average nr of steps
	path length	nr of steps	path length	nr of steps	
st deviation non-random	100.87	203.27	0.48	0,63	23.03
st deviation random	48.86	90.45	1.38	0,94	94.25
difference	-25.58	-68.8	3.1	0,8	965.44

Table 5: Blocks: differences

	before optimisation		after optimisation		average nr of steps
	path length	nr of steps	path length	nr of steps	
st deviation non-random	124.73	310.43	0.47	0.95	111.91
st deviation random	75.97	172.82	1.65	2.22	283.57
difference	66.4	111	3.25	2.3	1687.94

Table 6: Maze: differences

parameter	initial	optimized
pheromones dropped per ant	18	14
h	1.0	0.75
pheromone duration	3	4
number of possible directions	32	32
number of ants	20	20
amount of elite ants	80 %	90 %
supplement KDE	0.001	0.001
momentum multiplier	2.0	1.75
KDE multiplier	2.25	2.0

Table 7: The parameter values, the initial ones and the optimized ones.

value	result
5	248.56
16	175.15
25	181.94
32	170.04
45	175.3
64	163.04

Table 8: Parameter: Number of possible directions per ant

value	result
5	200.6
10	171.65
15	173.83
20	170.04
50	148.5

Table 9: Parameter: Number of ants

value	result
10%	214.68
25%	198.52
50%	182.53
80%	170.04
90%	167
100%	181.48

Table 10: Parameter: Percentage of elite ants

value	result
4	173.35
8	181.87
14	158.95
18	170.04
22	181.8
30	168.72

Table 11: Parameter: Number of pheromones dropped per ant

value	result
0.1	298.11
0.25	267.28
0.5	210.69
0.75	168.68
1.0	170.04
1.25	174.2
2.0	184.45
3.0	207.91

Table 12: Parameter: h

value	result
1	177.14
2	176.3
3	170.04
4	169.99
5	175.07

Table 13: Parameter: Pheromone duration

value	result
0.001	170.04
0.002	192.03
0.005	185.46
0.01	238.01
0.05	285.58

Table 14: Parameter: KDE supplement

value	result
0.5	249.83
1.0	187
1.5	163.7
1.75	161.6
2.0	170.04
2.25	162.47
2.5	175.37
3.0	179.56

Table 15: Parameter: Momentum multiplier

value	result
0.5	272.18
1.0	215.35
1.5	197.58
1.75	188.97
2.0	158.85
2.25	170.04
2.5	172.61
3.0	171.96

Table 16: Parameter: KDE multiplier

	blocks	maze
mostly momentum based	376.54	1235.1
mostly pheromone based	1097.97	1801.02
using both	169.36	676.09

Table 17: Momentum vs. Pheromones (values are average steps per ant)

A Software manual

A.1 Introduction

This short manual will explain the attached software. The software will allow the user to execute the algorithm using different parameters and different planes.

A.2 Installation

To run the software unzip `continuous_aco_exe.zip`. The executable will not run on its own. First install Microsoft Visual C++ 2010 Redistributable Package. (`www.microsoft.com/en-us/download/details.aspx?id=5555`) This will make sure all the right dll files etc. are installed. Now the executable can be run.

It is also possible to compile the files in `continuous_aco_cpp.zip`. The correct software and packages for compiling C++ with OpenGL and GLUT should be installed on the computer.

A.3 The program

When starting up the program shows two screens. A terminal for input and textual output and a screen which will show the plane and the ants' progress. Select the terminal and press ENTER to begin. Every iteration the terminal will show the progress of the ants by printing an 0 for an ant that found a path or a . character if an ant did not find a path. When all ants are done the length and step-size of the best path will be shown in the terminal. The visual screen will show the new pheromone values on the plane. When all iterations are completed the best found route will be showed.

If the demo mode is activated every first ant of an iteration will show its path. Keep pressing ENTER to see every step. If the user wants to stop watching these steps, press S and ENTER.

A.4 Parameters

The parameters of the algorithm and the software can be changed in the document `default_prefs.txt`. This file has to be in the same folder as the executable. The following variables and parameters can be edited:

- Height and width of the plane. Coordinates will be between 0 and these values. These values should be doubles.
- X and Y coordinates of the nest. Ants will start walking from here. These values should be doubles within the plane.
- X and Y coordinates of the target. Ants will find this goal. These values should be doubles within the plane.
- The number of extra targets. Additional targets can be added to the first one. When this number is bigger than 0 the program will ask for the coordinates of these additional values. This value should be an integer above 0.

- The number of rounds/iterations. The ants will walk the plane a number of times before terminating. This is the amount of these rounds. This value is an integer.
- The number of ants. The number of ants which will walk the plane every iteration. This is an integer.
- H. This is the h value in the KDE formula. The higher h is, the smoother the KDE function will be. This is a double.
- Maximum duration of a pheromone. After this amount of iterations a pheromone will evaporate. This is an integer.
- Number of elite ants used. The best ants will leave their pheromones. This value should be an integer between 1 and the number of ants.
- The number of pheromones each elite ant will leave behind. This should be an integer.
- Maximum number of steps per ant before giving up. When reaching this amount of steps the ant will no longer search for the target. This is an integer.
- The number of possible random directions an ant can choose from each step. This is an integer.
- Minimum and maximum step length. These values determine the length of the ant's steps. These are doubles.
- The supplement KDE, a small value to avoid the probability of steps being 0. This should be a double bigger than 0.
- How close an ant should be to the target so it will finish it's path. When an ant will be in this proximity it will automatically find its target. This should be a double bigger than 0.
- The KDE multiplier, this will increase the influence of the pheromones on the ants' decisions. This is a double.
- The Momentum multiplier, this will increase the influence of momentum on the ants' decisions. This is a double.
- The momentum boolean, is momentum used? 1 for yes, 0 for no.
- The optimization boolean. If the algorithm should optimize paths during the complete algorithm: 1, otherwise 0.
- The demo boolean, watch every step of the first ant's path? 1 for yes, 0 for no.
- The obstacle input file. The obstacles on the plane will be loaded from this file. This file should be in the same folder as the executable. Each obstacle in this file is a rectangle made up by four doubles separated by commas. An example can be seen in `obstacles_blocks.txt` or `obstacles_maze.txt`.