



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises:
Modelling and Simulating Social Systems with MATLAB

Project Report

**Solving the Travelling Salesman Problem
by Using an Artificial Ant Colony**

Raphaela Wagner & Giandrin Barandun

Zurich
May 2014

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Raphaela Wagner

Giandrin Barandun



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

SOLVING THE TRAVELLING SALESMAN PROBLEM
BY USING AN ARTIFICIAL ANT COLONY

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):

WAGNER
BARANDUN

First name(s):

RAPHAELA
GIANDRIN

With my signature I confirm that

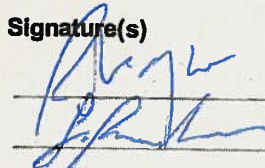
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

ZURICH, 13.05.14

Signature(s)



For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Contents

1	Abstract	5
2	Individual contributions	5
2.1	Raphaela Wagner	5
2.2	Giandrin Barandun	5
3	Introduction and Motivations	5
4	Description of the Model	7
4.1	Choosing a City	7
4.2	Moving Forward and Updating	8
4.3	Modifications	8
5	Implementation	9
5.1	Main program	9
5.2	Choose the Next City	10
6	Simulation Results and Discussion	10
6.1	Length of shortest tour	11
6.2	Model adaptations	13
7	Summary and Outlook	13
8	References	13

1 Abstract

2 Individual contributions

2.1 Raphaela Wagner

With the aim of achieving a good model for solving the travelling salesman problem by the use of artificial ants Raphaela helped the group understanding the underlying paper and the included model. She contributed a great amount of explanations and ideas how to approach the whole project.

In a second step she took care of how to implement the raw data from the TSP-library into MATLAB and transform it to a usable form. Further more she coded the functions "eta.m", "global_pheromene_update.m", "test_funktionen.m" and "update.m" and helped improving and correcting the main program.

After the code was written she did a lot of testing with different problem sets and compared the solution of the program to known solutions.

When the group got stuck and did not see a way out of a specific problem she was the one to bring along a hot chocolate and cheer the group up again.

2.2 Giandrin Barandun

The paper which is thought to be reconstructed on the following pages was selected and suggested to the group by Giandrin and during the whole process he tried to have some influence on the project with his wide technical understanding of the problem.

The codes for the functions "prob_dist.m", "calc_Lnn.m", "choose_city.m", "main_initialize_system.m", "coordinates.m" and "calc_dist.m" are his contributions as well as the collaboration on the main program. He searched the internet for known TS-problems and their solutions and put all data in a readable form. A lion's share for the program working at the end was his bug fixing in all the functions and programs and combining them to the running model.

3 Introduction and Motivations

Observing crawling ants how they manage to find a shortest path from a food source to their nest arises the question of how to model such a biological phenomena. It is known that the way ants organize their transporting system is based on a secreted chemical called pheromone. While ants move on a track they deposit a certain amount of pheromone. Since real ants prefer choosing lines of a high pheromone concentration, this messenger ensures that ants follow their members on a certain trail. To illustrate the effect of pheromone on an ant trail consider Figure 1. Real

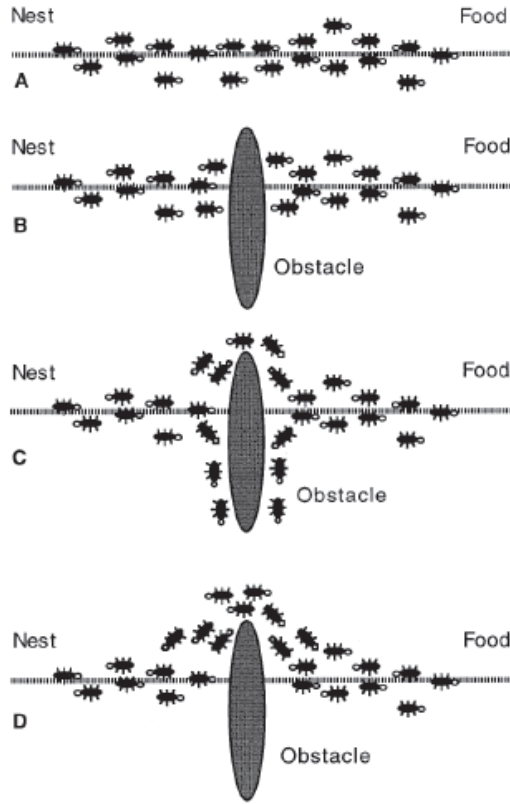


Figure 1: (A): Ants following a trail between food source and their nest. (B): An obstacle is placed to interrupt the track of the ants. (C): The column of ants splits into two groups each choosing a different way to circumvent the obstacle. (D): Due to the higher concentration of pheromone all ants have chosen the shortest path.

ants follow a path between a food source and their nest (Fig. 1 A). Placing an obstacle on the trail forces the ants to find a way of restoring the interrupted track (Fig. 1 B). One expects half of the ants to turn right and half of them to turn left. In the beginning both ways around the obstacle are enriched with approximately the same amount of pheromone (Fig. 1 C). Since ants that have chosen the shortest path need less time to pass by the obstacle the number of ants per time is bigger compared to those who have chosen the longer path. Consequently the shorter path contains a higher concentration of deposited pheromone than the longer one. This follows from the assumption that all ants secrete the same amount of pheromone and move approximately at the same speed.

After a certain time more ants prefer the shorter path containing more pheromone until the longer one is completely neglected to circumvent the obstacle (Fig. 1 D).

Consulting the literature ? one finds an interesting paper which uses an ant colony system (ACS) to solve a traveling sales man problem (TSP). Artificial ants, also called agents are successively moving on a TSP graph between different cities. In the course of this they are following the constraint to visit each city once and

return to their starting point. After all ants have completed their tour, the shortest one is rewarded by increasing the weight of the according tracks. This corresponds to a higher concentration of pheromone on the chosen tour.

The goal of this project is to implement the given model (see chapter 4 on page 7) from ? and to calculate the shortest tour for different city environments. Those are obtained from the TSPLIB (<http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp>) and correspond to the data used in the reference paper ?. The aim is to figure out whether our code is able to produce the same length of the shortest tour or not. Next to that, the variation of parameters in the model is analysed. Precisely, these simulations try to answer questions like: How does the shortest tour length depend on the rewarding, i.e. on the amount of pheromone deposited? How fast is the decay in the shortest tour as a function of completed rounds? Moreover, the influence of the number of agents on the time needed to find the shortest tour is investigated.

4 Description of the Model

In this model a number of k ants is sent on a network of m cities with every ant starting at the same city. The next ant only starts when the previous ant has finished its tour which means there never are two ants in the network.

4.1 Choosing a City

Before moving on to the next city an ant has to decide where it wants to go. For this purpose it chooses randomly a number q between zero and one and if this number is smaller or equal than a certain parameter q_0 ($q \leq q_0$) looks for the city s which fulfils the following formula:

$$s = \arg \max \{ [\tau(r, u)] \cdot [\eta(r, u)]^\beta \}, \quad u \notin M_k. \quad (1)$$

The current city where the ant stays is denoted by r and only cities can be chosen which are not yet in the ants memory M_k which means the ant has not visited these cities. The matrix τ stores the information about the amount of pheromone on the edge between city r and city u and the function η gives the inverse of the distance between the two cities.

If the random number is bigger than q_0 ($q > q_0$) then the ant randomly chooses one of the remaining unvisited cities and accepts or rejects it according to the probability p_k :

$$p_k(r, s) = \frac{[\tau(r, s)] \cdot [\eta(r, s)]^\beta}{\sum_{u \notin M_k} [\tau(r, u)] \cdot [\eta(r, u)]^\beta} \quad (2)$$

This probability basically contains the same formula of τ and η as the one above but is now normalized with the sum over these relations of every unvisited city. One can clearly see that in the beginning the sum is big and the probabilities are small but favouring the edges with more pheromone and lower distance. At the end when only one city is left the sum equals the term in the nominator and the probability becomes one for the last remaining city.

4.2 Moving Forward and Updating

Once the city has been chosen the ant moves along the edge and the pheromone on the path is updated according to:

$$\tau(r, s) = (1 - \alpha) \cdot \tau(r, s) + \alpha\tau_0 \quad (3)$$

The newly introduced parameters α and τ_0 are explained in chapter 6.2. This update reduces the amount of pheromone on the chosen edge and helps avoiding very strong edges which would be taken by all the ants.

At the time the first ant has finished the tour the second one can start while the first ant still keeps in mind the trajectory of its tour which means the sequence of the city it has visited and the length of its tour but deletes its memory such that it is ready for a new tour. When all ants have completed one tour the shortest one is rewarded with pheromone according to the formula:

$$\tau(r, s) = (1 - \alpha) \cdot \tau(r, s) + \alpha\Delta\tau(r, s) \quad (4)$$

This update is intended to give the edges along the shortest path a little head start in the following round. With this step the first round is complete and the second round can start.

4.3 Modifications

In the above model every ant started at city number 1 and waited for his predecessor to finish his tour completely before it started its own tour. In the paper^[1] underlying this report the model was slightly different. All agents are placed randomly on any city in the network and move together. When the first agent chose his second city and moved there he waits there until every ant has found its next city. So in every time step all agents move for one city. The difference to the model before is that every ant feels the pheromone on the edges of all the other ants in the network. Whereas in the previous model the second ant only saw the trace the first ant left but did not feel any influence of the still following ants.

Another small change which was tested is to increase the award for the shortest path

which was found after one round. This means to add a constant amount of pheromone to each edge along the shortest path in the global update formula (equation 4).

$$\tau(r, s) = (1 - \alpha) \cdot \tau(r, s) + \alpha \Delta \tau(r, s) + \mathbf{0.1} \quad (5)$$

5 Implementation

5.1 Main program

In the previous section the theoretical understanding of the model was tried to be imparted to the reader. Following in this section is an overview of how the model was implemented in MATLAB.

To start with a main program (*main_initialize_system.m*) was written which reads the data of a specific TSP and puts it into an upper triangle matrix form which contains the distances between the cities as elements. Further more all parameters like number of rounds and number of contributing ants (agents) can be adjusted in this main program. At the end it activates the function *main_main.m* with the purpose to find the shortest route.

In the main function there are two for-loops, one for the number of rounds and one for the number of ants (agents). For every agent we have a memory of the visited cities (M_k) and a trajectory vector which saves the sequence of how the cities were visited. In the first step every ant chooses the next city with help of the function *choose_city.m* which is described below. The city is added to the memory of the ant and the trajectory vector and the pheromone on the edge is reduced according to the formula on page 75 in the paper. As long as there are unvisited cities this procedure is repeated for the agents and at the end the way to the start city is updated (pheromone, trajectory) and the memory reset. Then the second agent starts his tour.

After every agent has finished his route the shortest path is detected and the edges along this route are rewarded with pheromone with help of the trajectory vector and the function *global_pheromone_update.m*. At the end of this step the trajectory vector is reset for all ants and the second round can be initialized. After every round the shortest route of the current round is compared to the overall shortest route found until now and then kept or rejected depending on the result of the comparison.

When all rounds have been calculated the function gives the shortest path found in any of the rounds.

5.2 Choose the Next City

To choose which city the ant will go next two different methods are implemented according to the paper and one of them is selected based on a certain probability. The first way to decide which city to go next is to optimize a function which depends on the amount of pheromone on the edge between the current city and the chosen one and its length. This method chooses the edge with the highest amount of pheromone and the shortest length whereas there are parameters to weight these two variables relatively to each other.

On the other hand a probability was assigned to every unvisited city again depending on the amount of pheromone and the length of the edge between the cities. Then a city was randomly chosen and accepted or rejected with its assigned probability. At the end the function *choose_city.m* gives the number of the next city to visit back to the main function.

6 Simulation Results and Discussion

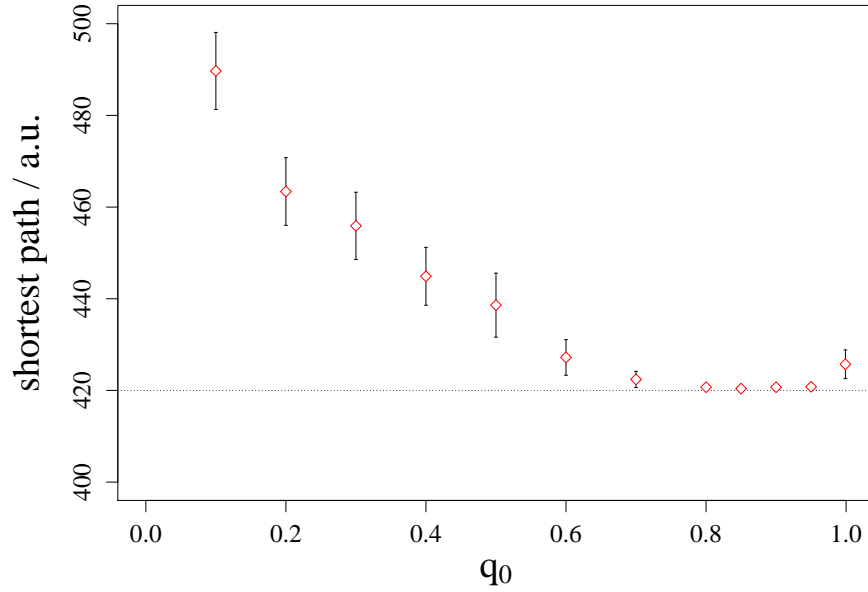


Figure 2: sdfasd

6.1 Length of shortest tour

In a first attempt the dependency of the shortest tour length on different parameters is analysed. Thereby the number of completed rounds each agent finishes is varied. Moreover the value of the update rate α is optimized and the relative weighting of deposited pheromone and closeness is examined by changing the parameter β .

The shortest tour length is calculated for a 30 and 51 cities problem (called *oliver30* and *eil51* respectively) and averaged over 10 trials. The error bars in the following plot always refer to the standard deviation from those repeated measurements.

The distances between the cities are determined using the given coordinates on a two dimensional euclidean plane. The distance $d_{i,j}$ between city i and j is then calculated by

$$d = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}. \quad (6)$$

For *oliver30* the distances are rounded to the next integer values, whereas in *eil51* double distances are used. This leads to slightly different numbers for the shortest tour length.

The trajectory vector of the shortest tour is defined as the sequence of city numbers which yields the optimal tour length. For the city environment *oliver30* the solution found in ? is given by [1 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 25 24 26 27 28 29 30 2]. Using equation (6) the total tour length is calculated to $shortest\ path_{solution} = 420$. Comparing it to the sequence obtained from own simulations one observes a somewhat different sequence: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 25 24 26 27 28 29 30 1] where the agents don't move from city 30 to 2 but to 1. However, the length of this shortest path also results in $shortest\ path_{simulation} = 420$. Obviously, there exist two solution to this symmetric TSP using rounded distances.

In *eil51* the trajectory vector found using the model from ? blabalbal. The length is balbalba $shortest\ path_{solution} = 426$.

In Figure 3 and 4 the dependency of the shortest path length on the number of completed rounds is shown. As expected the average length decreases asymptotically towards the value of the optimal path. For both city environments the program achieved to reach the averaged shortest path known from literature ? and ?. In *oliver30* $\langle shortest\ path \rangle$ is in fact equal the best result found using 400 rounds, whereas in *eil51* a lot more rounds would have been needed to achieve this. Anyhow, the best result obtained for *eil* in a single run is $shortestpath = 429.48 \pm 1.86$. This shows that

	ACS average ? and ?	ACS best result ? and ?	ACS average own simulations	ACS best result own simulations
<i>eil51</i>	433.87	429	433.7 ± 3.2	429.4
<i>oli30</i>	(N/A)	420	420.5 ± 0.5	420

Table 1: Simulation results of two city environments compared to values from literature ? and ?.

In Figure 3 three models have been implemented and tested. The first (white circle dots) correspond exactly to the one described in ? where all agents are moving at the same time step. The red squares represent the result obtained by a slightly modified model. The same mathematics has been used but the agents are exploring a new trail one by one. As can be seen, this change has little effect on the value of the shortest tour length and its error.

Adapting the pheromone update equations (??) and (4) on page 7 by adding a constant of 0.1, results in a steeper fall of the average shortest path. At 2000 rounds the final value does not yet reach the averaged shortest path length from ?. One concludes that the added constant was chosen too high such that the global and local update only had a neglecting influence on $\tau(r, s)$.

Figure 5 illustrates that the averaged shortest path length is optimized for α between 0.1 and 0.3. This is the region where the length of the tour and also the error of the shortest path is minimized. The reason why the shortest path is highest close zero and one, can be understood considering the pheromone update formulas (??) and (4) on page 7. For $\alpha = 0$ the local and global pheromone concentration stays constant, no update is made. Hence the cities are only favored by closeness and not by the pheromone concentration anymore. On the other hand, if α is close to one the amount of pheromone after the local update has changed a lot. Randomly chosen trails which deviate from the shortest path will be weighted too much. Therefore the optimal update rate α is expected to be closer to zero.

For the parameter β a similar analysis has been done for a thirty city environment and ten agents. The motivation for setting $\beta = 2$ in ? is barely understood from figure 6. The decrease of the error for higher values of β is explained by the fact that the criteria of closeness between the cities is weighted stronger than the pheromone concentration (see equation (2) and (1)). Thus, the exploration of new paths is suppressed and the deviation from the averaged path length is small. In the region of $\beta \approx 2$ the bigger error bars indicate that the relative importance of pheromone and closeness is chosen such that the ants are forced to try new trails. The probability of getting stuck in a local minimum is reduced.

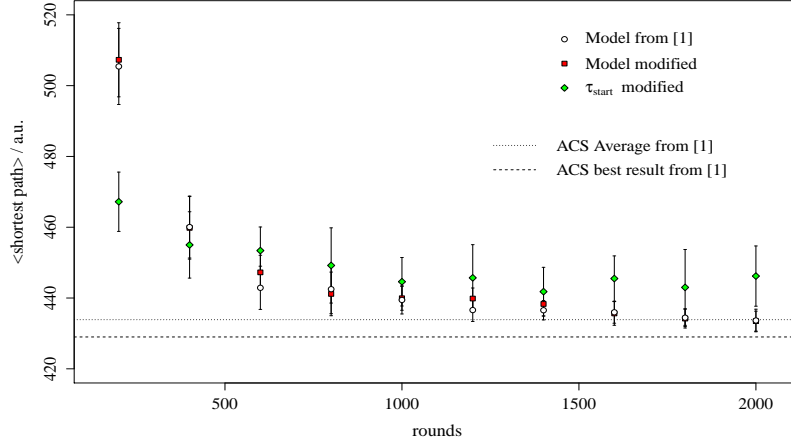


Figure 3: Shortest tour length averaged over 10 runs for different numbers of rounds and 10 agents with the corresponding standard deviation as error bars. Thereby two slightly different implementations of the model from literature [1] are done (white circle dots and red squares). The green diamonds correspond to a model where the update formula $\tau(r, s)$ is changed. The values approximate the optimal solution for ≈ 1200 rounds. The two different lines represent the best shortest tour and the average value measured according to [1]. The parameters are chosen as follows $\alpha = 0.1$, $\beta = 2$, $q_0 = 0.9$, $\tau_{\text{start}} = 0.1$.

6.2 Model adaption

7 Summary and Outlook

8 References

- [1] *Ant Colonies for the Travelling Salesman Problem*; Marco Dorigo, Luca Maria Gambardella; Lugano, Switzerland; 24.10.1996
- [2] <https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>

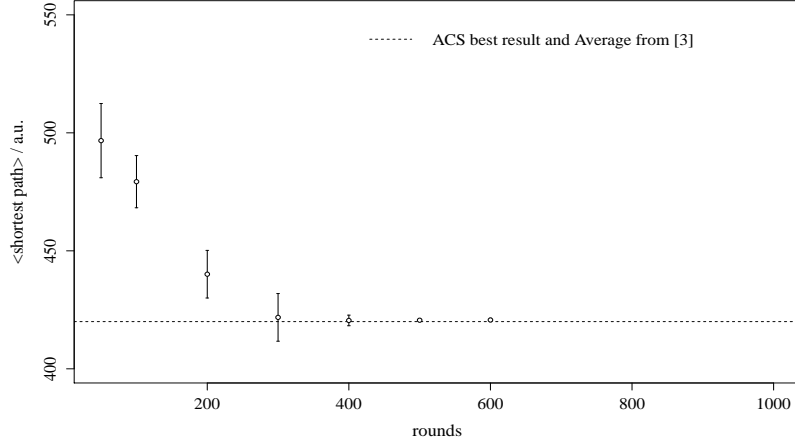


Figure 4: Shortest tour length averaged over 10 runs for different numbers of rounds and 10 agents with the corresponding standard deviation as error bars. The value approximates the optimal solution for ≈ 400 rounds. For this simulation the model and its implementation are done in the same way as in ?.

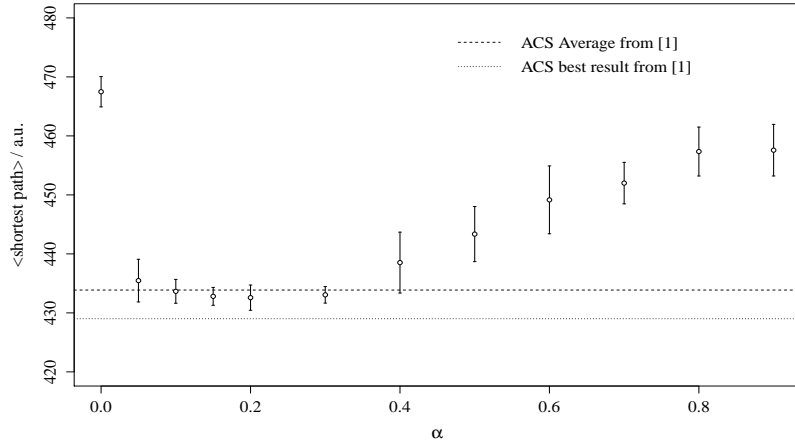


Figure 5: The update rate parameter α changes the averaged shortest tour length. The simulation is done for a fifty city problem using ten agents which all completed 2000 rounds. Other parameters are set to $\beta = 2$, $q_0 = 0.9$ and $\tau_{ij} = 0.1$. The values are averaged over ten runs. By setting $\alpha = 0.1$ the length of the averaged shortest path is comparable to the value found in literature ? and the change in pheromone update is held low.

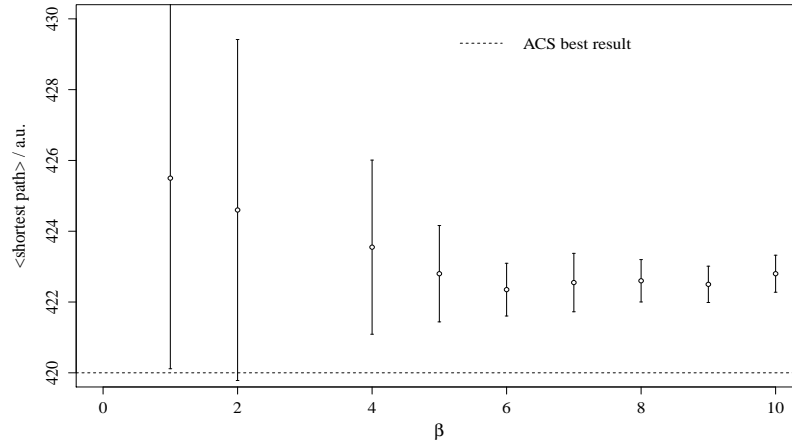


Figure 6: Variation of the parameter β for the oliver30 using ten ants going 400 rounds averaged over 20 runs. The parameters are chosen as $\alpha = 0.1, q_0 = 0.9$ and $\tau_{a??} = 0.1$.

Appendix

Main Program

```

1  %Hauptfunktion zum Whlen der Parameter und einlesen der Matrix mit Daten ...
   der Stdte und Strecken
2
3  %Fge Pfade zu den Datenstzen dazu (ANPASSEN!)
4  %addpath('C:\Users\Giandrin\Documents\GitHub\Solving-TSP-using-ACS\other');
5  addpath('C:\Users\Raphaela ...
   Wagner\Documents\GitHub\Solving-TSP-using-ACS\other');
6
7  %Stdtedaten einlesen
8  %cities.data = links obere Dreiecksmatrix mit Dimension: (no_cities-1) x
9  %(no_cities-1) mit Diagonaleintrgen
10
11 clear all
12 clc
13 close all
14
15 %Citydaten einlesen (2D euklidische Distanzen)
16 [filename, pathname] = uigetfile('*.txt', 'Please select a city environment');
17     if isequal(filename, 0)
18         disp('User selected ''Cancel''')
19
20     else

```

```

21     disp(['User selected ', fullfile(pathname, filename)])
22     delimiterIn = ' ';
23     headerlinesIn = 6;
24     cities = importdata(filename,delimiterIn,headerlinesIn);
25     data_set = coordinates(cities.data);
26
27     end
28
29 %     cities = importdata('eil51.txt',' ',6);
30 %     data_set = coordinates(cities.data);
31
32
33
34
35 %Citydaten einlesen (obere linke Dreiecksmatrix, zb city bayg29)
36 % [filename, pathname] = uigetfile('*.txt', 'Please select a city ...
    environment');
37 %     if isequal(filename, 0)
38 %         disp('User selected ''Cancel''')
39 %
40 %     else
41 %         disp(['User selected ', fullfile(pathname, filename)])
42 %         delimiterIn = ' ';
43 %         headerlinesIn = 8;
44 %         cities = importdata(filename,delimiterIn,headerlinesIn);
45 %         dim_data = length(cities.data);
46 %         %Neuordnung City-Matrix zu linker unterer Dreiecksmatrix (Eintrge
47 %         %oben rechts)
48 %         data_set = zeros(dim_data+1); ...
            %Initialisieren City-Matrix
49 %         for ii=1:dim_data
50 %             for jj=1:dim_data
51 %                 data_set(ii,jj+1) = cities.data(ii,dim_data-jj+1);
52 %
53 %             end
54 %
55 %         end
56 %
57 %         zero_nan = isnan(data_set);
58 %         data_set(zero_nan)=0;
59 %         data_set
60 %
61 %     end
62
63
64
65 alpha = 0.1;
66 beta_0 = 2;
67 no_agents = 10; %Wieviele Agents ...
    haben wir

```



```

68 rounds = 2000; %Wieviele Durchgange
69
70 q0 = 0.9;
71 tau_init = 0.1; %Pheromonmenge am ...
    Anfang
72
73
74 V = 2;
75
76
77 %-----
78 %Starte die Hauptfunktion
79 %-----
80
81
82 runs = 10; %shortest_path ...
    wird ber Anzahl runs gemittelt
83 global_shortest_path = zeros(runs,1);
84 %Fille Vektor mit shortest_path f r jeden Run
85 for ii=1:runs
86     [global_shortest_path(ii),tau_bild, global_shortest_trajectory] = ...
        main_main_agents_together(alpha, beta_0, no_agents, data_set, ...
            rounds, q0, tau_init);
87 end
88
89
90 V
91 rounds
92 disp('eil51')
93 global_shortest_path
94 global_shortest_trajectory;
95 global_shortest_path_average = sum(global_shortest_path)/runs ...
    %Gemittelter shortest_path
96 errors = std(global_shortest_path) %Standardabweichung shortest_path
97 figure
98
99 %-----

```

Main Function

```

1 %Hauptfunktion
2 %alle agents werden zu Beginn zufllig auf Stdte verteilt und bewegen sich ...
    dann gleichzeitig Schritt f r Schritt
3
4 function[global_shortest_path, tau_bild, global_shortest_trajectory]= ...
    main_main_agents_together(alpha, beta_0, no_agents, data_set, rounds, ...
        q0, tau_init)
5
6

```

```

7
8 %Auslesen der Anzahl St dte
9 no_cities = length (data_set(:,1)); %L nge der ...
    ersten Spalte der Matrix auslesen
10
11 %Memory der Ameise, Matrix mit Anzahl St dten x Anzahl Agents
12 %l heisst noch nicht besucht.
13
14 tau = zeros (no_cities) + tau_init; %tau als ...
    pheromenin-matrix mit dimension no_cities x no_cities, zu beginn alles null
15
16 %Berechnen von L.nn, bentigt f r tau0
17
18 L_nn = calc_Lnn(data_set, no_cities, 1); %Function ...
    calc_Lnn aufrufen um L_nn zu berechnen
19 tau0 = 1/(no_cities*L_nn);
20
21 start_city = zeros(no_agents,1); %Start_city ist ...
    f r jeden Agent unterschiedlich
22
23
24
25
26 global_shortest_path = L_nn; %Globaler ...
    shortest_path vergleicht shortest_path's von allen gegangenen rounds
27
28
29 %-----
30 %Start der Berechnung mit "rounds"-Durchgngen
31 %-----
32
33
34 for ii = 1:rounds
35
36     trajectory = zeros(no_cities, no_agents); ...
        %trajectory Matrix inizieren
37     current_city = zeros(no_agents, 1); ...
        %current_city Vektor inizieren
38     city_s = zeros(no_agents, 1); ...
        %s_city Vektro inizieren
39     path_length = zeros(no_agents,1); ...
        %Tourlngen-Vektor mit inizieren
40     M_k = ones(no_cities, no_agents);
41
42     for current_agent = 1:no_agents
43
44         start_city(current_agent) = randi([1, no_cities]); ...
            %Platziere die Agents zufllig in einer Stadt
45         M_k(start_city(current_agent), current_agent) = 0; ...
            %Memory f r Startstadt setzen

```

```

46     trajectory(1, current_agent) = start_city(current_agent);    ...
        %Die Startstadt jedes Agent als erste in der Trajectorymatrix ...
        setzen
47     current_city(current_agent) = start_city(current_agent);    ...
        %Startstadt bekannt geben
48
49
50 end %for current_agent
51
52
53
54 %Start der Durchgng, in jedem Durchgang machen alle Agents ...
    nacheinander einen Schritt, gehen also eine Stadt weiter
55 %es gibt soviele Durchgng wie es Stdte gibt und falls wir im letzten ...
    Durchgang sind, wird am Ende noch der Weg nachhause
56 %dazugerechnet.
57
58 for jj = 2:no_cities
59
60     traj_pos = jj;    ...
        %Position in Trajectorymatrix
61
62
63     %Alle Agents machen einen Timestep, gehen eine Stadt vor.
64     for current_agent = 1:no_agents
65
66         city_s(current_agent) = choose_city(tau, beta_0, ...
            M_k(:,current_agent), current_city(current_agent), ...
            no_cities, current_agent, q0, data_set);    ...
            %Whle eine Stadt
67
68         M_k(city_s(current_agent), current_agent) = 0;    ...
            %Memory dass stadt city_s besucht wurde
69
70
71     %-----pfadlnge und Pherominupdate-----
72     if current_city(current_agent) < city_s(current_agent)
73
74         path_length(current_agent) = path_length(current_agent) + ...
            data_set(current_city(current_agent), ...
            city_s(current_agent));
75         tau(current_city(current_agent), city_s(current_agent)) = ...
            (1-alpha)*tau(current_city(current_agent), ...
            city_s(current_agent))+alpha*tau0;    %Lokales ...
            Pheromenupdate
76
77     else
78
79         path_length(current_agent) = path_length(current_agent) + ...
            data_set(city_s(current_agent), ...

```

```

80         current_city(current_agent));
        tau(city_s(current_agent), current_city(current_agent)) = ...
            (1-alpha)*tau(city_s(current_agent), ...
            current_city(current_agent))+alpha*tau0;          ...
            %Lokales Pheromenupdate
81
82     end
83     %-----
84
85
86     trajectory(traj_pos, current_agent) = city_s(current_agent); ...
            ...
            %Trajectorymatrix updaten
87     current_city(current_agent) = city_s(current_agent); ...
            ...
            %Agent rckt eine Stadt vor
88
89
90     %-----Weg nach hause-----
91     if (jj == no_cities)
92
93         if current_city(current_agent) < start_city(current_agent)
94
95             path_length(current_agent) = path_length(current_agent) ...
                + data_set(current_city(current_agent), ...
                start_city(current_agent));
96             tau(current_city(current_agent), ...
                start_city(current_agent)) = ...
                (1-alpha)*tau(current_city(current_agent), ...
                start_city(current_agent))+alpha*tau0;          ...
                %Lokales Pheromenupdate fr heimweg
97
98         else
99
100             path_length(current_agent) = path_length(current_agent) ...
                + data_set(start_city(current_agent), ...
                current_city(current_agent));
101             tau(start_city(current_agent), ...
                current_city(current_agent)) = ...
                (1-alpha)*tau(start_city(current_agent), ...
                current_city(current_agent))+alpha*tau0;          ...
                %Lokales Pheromenupdate fr heimweg
102
103         end
104
105     end % end if
106     %-----
107
108 end %for current_agent, schleife der Agents in einem Timestep. ...
    Alle Agents bewegen sich um eine Stadt vor

```

```

109
110
111     end % for jj, schleife ber die Anzahl stdte
112
113     %Ermitteln des krzesten Pfades-----
114     shortest_path = path.length(1);                                     ...
115     %Pfad des 1. Agents als krzester Pfad gesetzt, zu Vergleichzwecken
116     shortest_path_index = 1;
117     for ll = 2:no_agents
118         if (path.length(ll) < shortest_path)
119
120             shortest_path = path.length(ll);                         ...
121             %Neuer krzester Pfad gefunden
122             shortest_path_index = ll;                                   ...
123             %Index zum Agent mit dem krzesten Pfad
124
125         end %if
126
127     end %for ll, zum ermitteln des krzesten Pfads
128
129     %-----
130     %Globales Update
131     %-----
132     tau = global_pheromene.update(trajectory(:, shortest_path_index), tau, ...
133         shortest_path, no_cities, alpha, trajectory(1, shortest_path_index) );
134
135
136     %Vergleiche den krzesten Pfad dieser Runde mit dem krzesten Pfad von ...
137     %allen
138     if shortest_path < global_shortest_path
139         global_shortest_path = shortest_path;
140         global_shortest_trajectory = trajectory (shortest_path_index);
141
142     end %if shortest path Vergleich
143
144
145
146 %
147
148 %     if mod(ii,50) == 0                                             ...
149 % %Ausgabe global_shortest_path nach jeder 50. round
150 %
151 %         global_shortest_path
152 %
153 %     end %if Ausgabe von global_shortest_path

```

```

153
154 end %for ii, ber die rounds
155 global_shortest_path
156 trajectory(:, shortest_path_index)
157     tau.bild = tau;

```

Coordinates to Distancematrix

```

1  %INFO: Funktion liest Vektor mit Koordinaten ein und gibt Distanzmatrix ...
    (als obere Dreiecksmatrix)
2  %
3  %
4  %INPUT: koordinatenvektro in der form (Nr., x-coord, y-coord)
5  %
6  %OUTPUT: Distanzmatrix
7
8
9  function[data_set] = coordinates(koordinaten)
10
11     no_cities = length (koordinaten);
12     data_set = zeros(no_cities);
13
14     for haupt=1:(no_cities-1)
15
16         for jj = (haupt+1):no_cities
17
18             data_set(haupt, jj) = sqrt( (koordinaten(haupt, 2) - ...
                koordinaten(jj, 2))^2 + (koordinaten(haupt, 3) - ...
                koordinaten(jj, 3))^2 );
19
20         end %for jj
21
22     end %for haupt
23
24 end

```

Calculate Nearest Neighbour Heuristic Distance

```

1  %Berechnet Lnn, welches die nearest neighbour heuristic tour length ist,
2  %also tourlength falls man immer zum nächsten nachbarn geht
3
4  function[total_length] = calc_Lnn(data_set, no_cities, start_city)
5
6     longest_distance = max(max(data_set))+1;
7
8     %eins mehr als die ...
9     %längste distanz, wird also nie gewählt
10    current_city = 1;

```

```

8     city_memory = zeros(no_cities,1);
                                     %Erinnerung ob wir in einer ...
        Stadt schon waren
9     city_memory(start_city) = 1;
10    distances = data_set + data_set' + ...
        diag([ones(no_cities,1)]*longest_distance;
                                     ...
        %Bildet die komplette Matrix mit berlangen distancen ...
        auf der diagonale, damit die nicht gew hlt werden
11    total_length = 0;
12
13    for ii = 2:no_cities
                                     %no_cities-1 durchg nge f hrt ...
        zu letzten stadt
14
15
16        test = 1;
                                     %initialisiere Test variabel ...
        distance_vector = distances(:, current_city);
                                     ...
                                     %Alle distancen in andere ...
        St dte von der jetzigen Stadt aus
18
19    while(test)
20
21        [shortest_length, current_city]=min(distance_vector);
                                     ...
                                     %Gibt als shortest ...
        length den kleinsten Eintrag und als current_city die position
22
23        if (city_memory(current_city) == 0)
                                     ...
                                     %Falls Stadt noch nicht ...
        besucht
24
25        total_length = total_length + shortest_length;
                                     ...
                                     %Distance zu dieser Stadt dazu addieren
26        test = 0;
                                     ...
                                     %und aus ...
        while-Schleife aussteigen
27        city_memory(current_city) = 1;
28
29    else
                                     ...
                                     %Falls Stadt schon ...
        besucht mach die Distanz l nger als die lngste,
30
31
32        distance_vector(current_city) = longest_distance;

```

```

33         end
34
35
36     end
37
38
39 end %ii
40
41     total_length = total_length + distances(1, current_city); ...
                                     %addiere noch die distanz nach ...
42     hause dazu
43 end

```

Choose a City Function

```

1  %Mit Wahrscheinlichkeit q0 whlt Agent eine neue Stadt nach dem Modell in
2  %Gleichung (1). Mit Wahrscheinlichkeit (1-q0) geht er neuen Weg zu S.
3  %
4  %Input:      Pheromen-Matrix fr alle Standorte, Vektor
5  %            in Memory-Matrix fr aktuellen Agenten, aktueller Standort,
6  %            totale Anzahl Cities, aktueller Agent, Parameter
7  %
8  %Output:     Neu gewhlte City s
9
10 function [city-s] = choose_city (tau, beta_0, M_k, current_city, no-cities, ...
    current_agent, q0, data-set)
11
12     q = rand(); %Zufallszahl zur Bestimmung ...
    ob Model in Gleichung (1) oder nicht
13
14     if (q ≤ q0)
15         city-s = 0; %Ausgabe Stadt als Null ...
            initialisieren
16         comp_arg = 0; %Vergleichsargument zur ...
            Entscheidung ob Stadt nehmen oder nicht
17
18         %Start for-loop um alle Stdte zu checken
19         for ii = 1:no-cities
20
21
22             if (M_k(ii) == 1) %Falls city ii ...
                von current_agent noch nicht besucht (M_k(ii) = 1)
23
24
25                 %Berechne das argument nach Formel (1) im Paper
26                 if (current_city > ii)
27                     argument = tau(ii, current_city)*eta(current_city, ...
                        ii, data-set)^beta_0;

```



```

28         else
29             argument = tau(current_city, ii)*eta(current_city, ...
30                 ii, data.set)^beta_0;
31         end
32         if (argument ≥ comp_arg) %Falls das ...
33             Argument das grsste bis jetzt, setze city_s neu
34
35             city_s = ii;
36             comp_arg = argument;
37         end
38
39         end %if M_k(ii)
40
41     end %for ii
42     city_s;
43     %nchste Stadt gefunden —> city_s
44
45     %Bestimme S falls nchste City nicht nach Modell in (1) gewhlt wird
46     else
47         while (1)
48             s0 = floor(rand()*no_cities)+1; %Generiere ...
49             Zufallszahl zwischen 1 und no_cities, floor generiert ...
50             zwischen 0 und no_cites-1, deshalb +1
51             p0 = rand(); %Genierere ...
52             weitere Zufallszahl um neue Stadt mit Wahrscheinlichkeit p0 ...
53             anzunehmen
54
55             %Agent whlt eine City mit Wahrscheinlichkeit p0, falls er sie ...
56             noch nicht besucht hat
57             if (M_k(s0) == 1 && p0 < prob-dist(tau, M_k, current_city, s0, ...
58                 no_cities, beta_0, data.set))
59                 city_s = s0;
60                 break; %Stadt city_s ...
61                 wurde gewhlt
62             end
63
64         end %end while
65     end %end if else
66 end %end function

```

Calculate the η -Function

```

1 %Funktion eta() welche in p_k gebraucht wird.
2 %Ist gegeben als das Inverse der Distanz zwischen current_city und

```

```

3 %next-city
4
5 %Input:      Aktueller und neuer Standort
6 %Output:     1/(Distanz der beiden Standorte)
7
8 function [distance] = eta(current_city, next_city, data_set)
9
10
11     if (next_city > current_city)
12         distance = 1/data_set(current_city, next_city);
13     else
14         distance = 1/data_set(next_city, current_city);
15     end
16 end

```

Probability p_k

```

1 %Funktion zur Berechnung von p_k aus Gleichung (2) im Paper
2 %
3 %INFO:      Funktion eta() ist extern definiert und muss NICHT als Input
4 %            gegeben werden! Kann innerhalb von prob_dist() aufgerufen werden.
5 %            Indem Variablen innerhalb einer Funktion als global definiert ...
6 %            werden (z.b. 'global X')
7 %            entfällt mehrfaches einlesen.
8 %
9 %Input:     Ganze Pheromen-Matrix und nicht nur Vektor f r aktuelle Stadt, da ...
10 %           nur Dreiecksmatrix, !!!Vektor!!! aus Memory-Matrix f r aktuellen
11 %           Agenten, Funktion eta = 1/(distance cities), aktueller Standort,
12 %           nchster Standort, totale Anzahl Cities, Parameter
13 %Output:    Wahrscheinlichkeit p_k mit welcher eine neue Stadt gewhlt wird
14
15 function [p_k] = prob_dist (tau, M_k, current_city, next_city, no_cities, ...
16                             beta_0, data_set)
17
18 %Bilde eine Summe des Gewichts aller besuchten Stdte
19 summe = 0;
20
21 for ii = 1:no_cities
22
23     if M_k(ii) == 1
24
25         %
26         if (ii < current_city)
27             summe = summe + tau(ii, current_city)*eta(current_city, ii, ...
28                 data_set)^beta_0;
29         end
30     end
31 end

```

```

29         else
30             summe = summe + tau(current_city, ii)*eta(current_city, ii, ...
31                 data_set)^beta_0;
32         end
33     end
34 end
35
36
37 %Berechne die Probability Distr fr die Stdte
38 if M_k(next_city) %Stadt noch nicht besucht
39
40
41     if (current_city > next_city)
42         p_k = (tau(next_city, current_city)*eta(current_city, ...
43             next_city, data_set)^beta_0)/summe;
44     else
45         p_k = (tau(current_city, next_city)*eta(current_city, ...
46             next_city, data_set)^beta_0)/summe;
47     end
48 else
49     p_k = 0;
50
51 end
52
53 end

```

Update Pheromone Globally

```

1  %Funktion fr globales Pheromenuupdate. Nachdem alle Agenten eine Tour
2  %absolviert haben wird die krzeste ausgesucht und auf jeder
3  %Verbindungsstrecke zwischen den einzelnen cities zustzlich Pheromen
4  %deponiert.
5  %
6  %Input:      Vektor der krzesten Tour von der trajectory-Matrix !!!ALS ...
7               VEKTOR!!, ganze
8  %           Matrix tau, Lnge der krzesten Tour, Anzahl Stdte, Parameter
9  %Output:     Aktuelle Pheromenwerte in tau
10
11 function [tau] = global_pheromene_update(trajectory, tau, shortest_path, ...
12     no_cities, alpha, start_city )
13
14     for ii = 1:no_cities-1
15         if (trajectory(ii) > trajectory(ii+1))
16             tau(trajectory(ii+1), trajectory(ii)) = ...
17                 (1-alpha)*tau(trajectory(ii+1),trajectory(ii))+alpha/shortest_path;

```

```

16
17         else
18             tau(trajectory(ii), trajectory(ii+1)) = ...
                (1-alpha)*tau(trajectory(ii),trajectory(ii+1))+alpha/shortest_path;
19
20         end
21
22     end
23
24     %Nach hause weg mit pheranomin versorgen
25     if start_city < trajectory(no_cities)
26         tau (start_city, trajectory(no_cities)) = ...
            (1-alpha)*tau(start_city,trajectory(no_cities))+alpha/shortest_path;
27     else
28         tau (trajectory(no_cities), start_city) = ...
            (1-alpha)*tau(trajectory(no_cities), ...
                start_city)+alpha/shortest_path;
29
30
31 end

```