# Travelling Salesman Problem: Is it possible to find the optimal tour of a network with a new algorithm based on Ant Algorithm?



## Introduction

When I graduate from secondary school, I want to go on a trip around the United States of America, where I visit my German relatives who migrated to the USA at around 1900 due to famine. During my time, I would like to visit thirteen cities in the shortest timespan. I therefore became interested in learning the different algorithms that can be applied to a network to find the shortest combination of journeys possible to visit every place. After a research on the internet, I found intelligent algorithms, which simulate nature, such as Simulated Annealing Algorithm (SAA) and Ant colony optimisation. These algorithms solve problems within discrete mathematics and applied mathematics through combinatorial optimisation, where the aim is to to find an optimal object from its set.

## Aim

The aim of this work is to compare the capabilities of the Nearest Neighbour Algorithm (NNA) and the Ant Algorithm, in terms of finding an optimal route, for my 13 city network. In order to do this, a deep understanding of the Ant Algorithm must be gained, and how the parameters influence the results of the Ant Algorithm must be investigated.

## Rationale

Nowadays, with online shopping services, and the increase in the usage of telecommunication devices, the TSP is becoming more important. Products are delivered by drivers from companies such as UPS and Amazon, whose routes are generated by an optimisation algorithm on a computer, and can only be changed slightly by individual drivers. The TSP is also found within autocatalytic processes (Colorni, Dorigo and Maniezzo, 1992), the Vehicle Routing Problem, telephone networks (AntNet), and the Quadratic Assignment Problem (Kopp, Leßmann and Kranstedt, 2003). The reason why the topic for this investigation was chosen was so that next time I visit the USA, I can see as much of the USA as possible, even with a short amount of time.

## Planning the exploration

In order to achieve the aim, this exploration requires a logical development, as the TSP is a large area of mathematics, so a lack of logic could lead to an incoherant exploration. To create a logical development, careful planning is needed: firstly, the NNA will be explored. The result will compared to the optimal solution, which can only be found using a brute force program.

Afterwards, a decision will be made on how to approach the Ant Algorithm efficiently. Afterwards, solutions will be found using programs based on Ant Algorithm. By structuring the exploration in this way, a reader can gain a thorough understanding of the mathematical concepts explored, which is important when fulfilling the aim.

**A short introduction of networks and graphs and useful terminology**

A graph are ordered pairs:

$$G := (V_G, E_G) \text{ with } n \in \mathbb{N} \;\; V_G = \{v_1, ....v_n\} \text{ set of vertex (node) and edges } E_G \subseteq \binom{V_G}{2} \quad \textbf{(1)}$$

An instance is the set of objects (towns or cities) which are the input for a heuristic. In this case, these are the algorithms, and its dimension is the cardinality of this set. In this example, I have 13 towns, so the the dimension is 13.

In an undirected graph, the edges have no orientation, so:

$$\{v_n, v_{n+1}\} = \{v_{n+1}, v_n\} \quad \textbf{(2)}$$

A graph is a complete graph $K_n$, when:

$$E_G = \{(u, v) : u, v \in V_G \quad u \neq v\} \quad \textbf{(3)}$$

A complete graph has

$$\textstyle\sum_{k=1}^{n} k = 1 + 2 + 3...n = \frac{n(n+1)}{2} = \binom{n+1}{2} = |E_G| \quad \textbf{(4)}$$

edges. This generates triangular numbers. If:

$$\{(u, v)\} \in E_G \quad \textbf{(5)}$$

$u$ and $v$ are adjacent neigbours. The vertex's number of neighbours is its degree: $deg(u)$. In our example, a vertex has 12 neighbours, as my problem is modelled as a complete graph. If $v \in V_G$ and $e \in E_G$ with $v \in e$, then $e$ is incident with $v$.

A graph is a cycle graph if:

$$C_G = (V_G, E_G) \text{ with} \{v_i, v_{i+1}\} \in E_G \wedge \{v_n, v_1\} \in E_G \quad \textbf{(6)}$$

A graph is a weighted graph if a number (the weight here is in kilometres) is assigned to each edge. A Hamiltonian path or cycle is where all nodes are visited only once. A graph which contains a Hamiltonian Cycle is called a Hamiltonian Graph. I am interested in finding the optimal Hamiltonian Cycle with the lowest total weight. Here, the deterministic objective function is the sum of all distances between the nodes of a tour and the constraint that every city is visited only once and that the tour is a cycle (Theobald, 2007).

During my research, I found a mathematical description of the Travelling Salesman Problem, which I will use as the basis for my description.

$$MinTSP = (I, Sol, w, goal) \quad \textbf{(7)}$$

with

$$I := \{\ \langle (d_{i,j})_{1 \leq i,j \leq n} \rangle | n, d_{i,j} \in \mathbb{N}\} \quad \textbf{(8)} \qquad Sol(D) := S_n \quad \textbf{(9)}$$

$Sol(D)$ is the set of all tours in $D$. $S_n$ is the set of all permutations of $n$ elements. $\pi$ is the tour, where $\pi_i$ is the order the towns are visited in. The length of a tour $w$:

$$w_{(D,\pi)} = \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)} \quad \textbf{(10)} \quad goal = min \quad \textbf{(11)}$$

The decision problem with border of length $B$ (Nöhring, 2007):

$$TSP = \{\ \langle (d_{i,j})_{1 \leq i,j \leq n}, B \rangle | n, d_{i,j}, B \in \mathbb{N} \wedge \pi \in S_n. \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)} \leq B\} \quad \textbf{(12)}$$

Below, a map of all the cities to be visited is displayed. A is Washington D.C.. B is New York. C is Chicago. D is San Francisco. E is Cincinnati. F is Boston. E is Los Angeles. F is Seattle. G is Philadelphia. H is Houston. I is Denver. J is Austin. K is New Orleans.

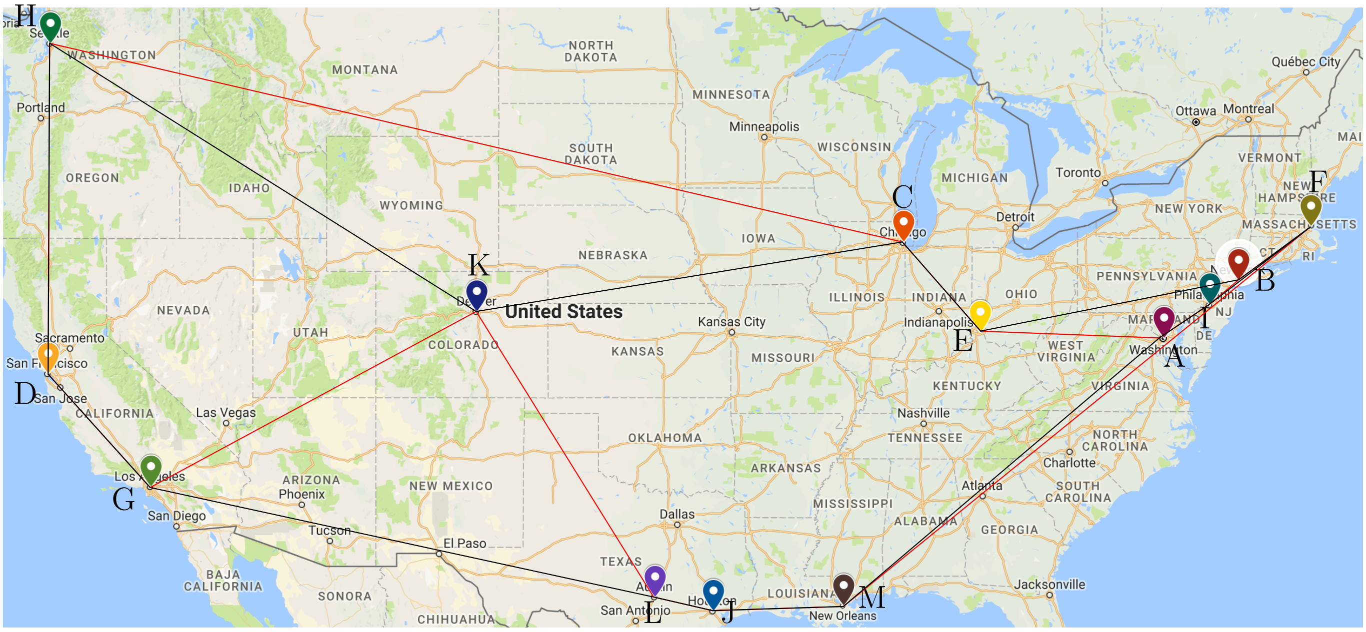



**Figure 1:** Towns to be visited

**Nearest Neighbour Algorithm**

The Nearest Neighbour Algorithm is a heuristic algorithm, which allows someone to find an upper bound. The Nearest Neighbour Algorithm yields a short tour where all nodes in a network have been visited at least once, but the tour is not usually optimal. The upper bound must be the smallest possible value. This algorithm consists of the following steps:

1. Use each node as starting points.
2. Go to the nearest unvisited node.
3. Repeat step 2 until all nodes have been visited, and return to the starting node using shortest route.
4. After all nodes have been used as starting points in the different routes, select the shortest route as the upper bound (Jameson, 2010).

With this, I start with town A. I start by looking for the smallest number column A, which is 222, which is in row I. Row A is deleted. I look for the smallest number in column I, which is 155, which is in row B. Row I is deleted. I look for the smallest number in column B, which is 347, which is in row F. Row B is deleted. I look for the smallest number in column F, which is 1392, which is in row E. Row F is deleted. I look for the smallest number in column E, which is 474, which is in row C. Row E is deleted. I look for the smallest number in column C, which is 1482, which is in row M. Row C is deleted. I look for the smallest number in column M, which is 557, which is in row J. Row M is deleted. I look for the smallest number in column J, which is 259, which is in row L. Row J is deleted. I look for the smallest number in column L, which is 1466, which is in row K. Row L is deleted. I look for the smallest number in column K, which is 1626, which is in row G. Row K is deleted. I look for the smallest number in column G, which is 613, which is in row D. Row G is deleted. I look for the smallest number in column D, which is 1291, which is in row H. Row D is deleted. I look for the smallest number in column H.

By starting at A, the route is $\pi_{NNA_{(A)}} = (A, I, B, F, E, C, M, J, L, K, G, D, H, A)$.

(A,I)=222, (I,B)= 155, (B,F)= 347, (F,E)= 1392, (E,C)= 474, (C,M)= 1482, (M,J)= 557, (J,L)= 259, (L,K)= 1466, (K,G)= 1626, (G,D)= 613, (D,H)= 1291, (H,A)=4432

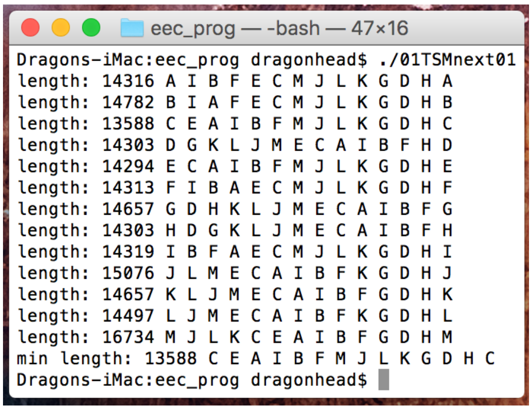
The length of this route is: $222 + 155 + 347 + 1392 + 474 + 1482 + 557 + 259 + 1466 + 1626 + 613 + 1291 + 4432 = 14316$

The full calculation of this example, which includes all tables, can be seen in Appendix A1. A table which summarises these calculations can be seen in Figure 1:

| | **A** | B | C | D | E | F | G | H | I | J | K | L | M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | - | ~~360~~ | ~~1117~~ | ~~4570~~ | ~~834~~ | ~~701~~ | ~~4254~~ | ~~4432~~ | ~~222~~ | ~~2253~~ | ~~2682~~ | ~~2437~~ | ~~1738~~ |
| B | ~~360~~ | - | ~~1262~~ | ~~4650~~ | ~~1022~~ | ~~347~~ | ~~4443~~ | ~~4544~~ | **155** | ~~2605~~ | ~~2845~~ | ~~2789~~ | ~~2090~~ |
| C | ~~1117~~ | ~~1262~~ | - | ~~3410~~ | **474** | ~~1574~~ | ~~3022~~ | ~~3302~~ | ~~1214~~ | ~~1715~~ | ~~1605~~ | ~~1792~~ | ~~1482~~ |
| D | ~~4570~~ | ~~4650~~ | ~~3410~~ | - | ~~3814~~ | ~~4958~~ | **613** | ~~1291~~ | ~~4600~~ | ~~3085~~ | ~~2005~~ | ~~2813~~ | ~~3640~~ |
| E | ~~834~~ | ~~1022~~ | ~~474~~ | ~~3814~~ | - | **1392** | ~~3480~~ | ~~3725~~ | ~~915~~ | ~~1678~~ | ~~1907~~ | ~~1806~~ | ~~1290~~ |
| F | ~~701~~ | **347** | ~~1574~~ | ~~4958~~ | ~~1392~~ | - | ~~4773~~ | ~~4886~~ | ~~310~~ | ~~2957~~ | ~~3154~~ | ~~3141~~ | ~~2442~~ |
| G | ~~4254~~ | ~~4443~~ | ~~3022~~ | ~~613~~ | ~~3480~~ | ~~4773~~ | - | ~~1816~~ | ~~4342~~ | ~~2477~~ | **1626** | ~~2205~~ | ~~3174~~ |
| H | ~~4432~~ | ~~4544~~ | ~~3302~~ | **1291** | ~~3725~~ | ~~4886~~ | ~~1816~~ | - | ~~4514~~ | ~~3710~~ | ~~2093~~ | ~~3392~~ | ~~4138~~ |
| I | **222** | ~~155~~ | ~~1214~~ | ~~4600~~ | ~~915~~ | ~~310~~ | ~~4342~~ | ~~4514~~ | - | ~~2475~~ | ~~2765~~ | ~~2659~~ | ~~1960~~ |
| J | ~~2253~~ | ~~2605~~ | ~~1715~~ | ~~3085~~ | ~~1678~~ | ~~2957~~ | ~~2477~~ | ~~3710~~ | ~~2475~~ | - | ~~1646~~ | ~~259~~ | **557** |
| K | ~~2682~~ | ~~2845~~ | ~~1605~~ | ~~2005~~ | ~~1907~~ | ~~3154~~ | ~~1626~~ | ~~2093~~ | ~~2765~~ | ~~1646~~ | - | **1466** | ~~2077~~ |
| L | ~~2437~~ | ~~2789~~ | ~~1792~~ | ~~2813~~ | ~~1806~~ | ~~3141~~ | ~~2205~~ | ~~3392~~ | ~~2659~~ | **259** | ~~1466~~ | - | ~~818~~ |
| M | ~~1738~~ | ~~2090~~ | **1482** | ~~3640~~ | ~~1290~~ | ~~2442~~ | ~~3174~~ | ~~4138~~ | ~~1960~~ | ~~557~~ | ~~2077~~ | ~~818~~ | - |

**Figure 2:** Example 1: Nearest Neighbour Algorithm (start at town A)

This process would be carried out repeatedly by starting at town B, C, and so on. Afterwards, all resulting tours are compared, and the shortest route is chosen as the upper bound. However, as the Nearest Neighbour Algorithm is prone to mistakes, and is time-consuming, I chose to write a program, whose source code can be seen in Appendix **A1**. The output of this program is displayed in Figure 3. The shortest tour here is $\pi_{NNA} = (C, E, A, I, B, F, M, J, L, K, G, D, H, C)$, which has a weight of $w_{\pi_{NNA}} = 13588$ (length of 13588km). I became interested in how far this result is from the shortest route possible. To determine the shortest route possible, brute force must be applied.

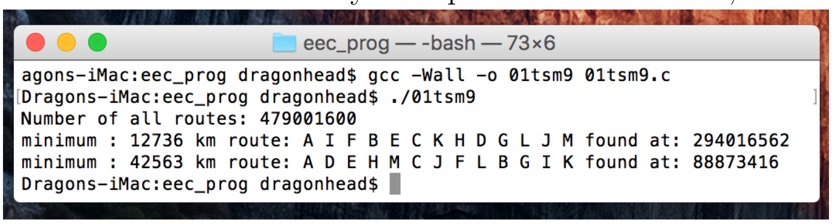


```
eec_prog — -bash — 47×16
Dragons-iMac:eec_prog dragonhead$ ./01TSMnext01
length: 14316 A I B F E C M J L K G D H A
length: 14782 B I A F E C M J L K G D H B
length: 13588 C E A I B F M J L K G D H C
length: 14303 D G K L J M E C A I B F H D
length: 14294 E C A I B F M J L K G D H E
length: 14313 F I B A E C M J L K G D H F
length: 14657 G D H K L J M E C A I B F G
length: 14303 H D G K L J M E C A I B F H
length: 14319 I B F A E C M J L K G D H I
length: 15076 J L M E C A I B F K G D H J
length: 14657 K L J M E C A I B F G D H K
length: 14497 L J M E C A I B F K G D H L
length: 16734 M J L K C E A I B F G D H M
min length: 13588 C E A I B F M J L K G D H C
Dragons-iMac:eec_prog dragonhead$
```

**Figure 3:** Output from NNA Program

### Determining the shortest tour with brute force

A brute-force approach involves systematically checking whether each candidate for a solution satisfies the statement of a problem (Rouse, 2006). Here, a brute-force approach will be employed in order to systematically check whether each tour to find the shortest one. My program starts at town A, and the next destination can be any of the unvisited towns (B to M). If B is visited, then the unvisited towns C to M are the only next possible destinations, and so on.



```
eec_prog — -bash — 73×6
agons-iMac:eec_prog dragonhead$ gcc -Wall -o 01tsm9 01tsm9.c
Dragons-iMac:eec_prog dragonhead$ ./01tsm9
Number of all routes: 479001600
minimum : 12736 km route: A I F B E C K H D G L J M found at: 294016562
minimum : 42563 km route: A D E H M C J F L B G I K found at: 88873416
Dragons-iMac:eec_prog dragonhead$
```

**Figure 4:** Output from Brute Force Program

My program contains 12 loops, and is written as a monolithic program, to save runtime. The source code is displayed in Appendix **A2**, and the output of the program is displayed in Figure 4. For 13 towns, there are 12! = 479001600 possible routes to be checked. The program found that the shortest route is $\pi_{BFmin} = (A, I, F, B, E, C, K, H, D, G, L, J, M, A)$, which has a weight of $w_{\pi_{BFmin}} = 12736$. This was the $294,016,562^{th}$ combination.

To compare: the longest route was $\pi_{BFmax} = (A, D, E, H, M, C, J, F, L, B, G, I, K, A)$ and had a weight of $w_{\pi_{BFmax}} = 42563$. This was the $88,873,416^{th}$ combination. Of course, there are only $\frac{12!}{2}$ combinations, as routes can occur in both directions. For example, a route (A,B,C) can also go in the direction (C,B,A). I checked 12! possibilities, as I would have to program the computer to recognise if a route already exists in the other direction, but this costs a lot more runtime, than if just 12! possibilities were checked. My program runs already for several minutes. If there were more towns, such as 15 towns, the runtime of the program would be too long (NP-problem).

**Reflection on the results from the Nearest Neighbour Algorithm and from brute force**

It is interesting to compare the shortest length found using brute force (this is the black curve in Figure 1) with the upper bound found using the Nearest Neighbour Algorithm (the red curve in Figure 1). Certain sequences are present in both routes. For example, (A,I), (B,F), (E,C), (H,D), (D,G), (L,J), and (J,M) are found in both routes. Some of these pairs, such as JM, are found in both lines, but are swapped around in one of the lines MJ. Some interesting is that the combinations of the letters in the shortest tour (A,I,F,B) is swapped in the tour found using the Nearest Neighbour Algorithm (A,I,B,F). The distance between I and F is 310, but the distance between I and B is 155. Another more obvious example to demonstrate this problem is (H,K,C) (brute force) and (L,K,G) (Nearest Neighbour Algorithm). The Nearest Neighbour Algorithm does not find (H,K,C), as K was already visited in LKG. This shows a great problem within the Nearest Neighbour Algorithm. This algorithm is of a greedy nature. This means that the algorithm ensures that the shortest distances are chosen at each stage, regardless of whether this is the optimal solution or not (Jameson, 2010).

When comparing the result of the NNA and brute force, it can be seen that the result from NNA is $\frac{13588 - 12736}{12736} \cdot 100 = 6.69\%$ larger than the optimal result. It is also interesting to mention that both curves develop a Jordan cycle, which means that none of the edges cross each other. This is the case, even when the distances between the nodes are calculated in an Euclidean space, and also not in a real space. This result may be quite acceptable for many applications, but in my task, this percentage error is unacceptably large. Therefore, a more sophisticated method, such as the Ant Algorithm, should be used.

**Introduction to Ant Algorithm**

Ant System and Ant Colony Optimisation simulate the behaviour of ants in order to find a solution to the Travelling Salesman Problem. Ants cannot see very well, and do not have a great thinking capacity. Despite this, they are still able to find the shortest distance between the nest and the food source. Unlike the classic algorithms, not only is one individual on its way, but rather several ants are running at the same time. If an ant finds an obstacle it could choose to go right or left around the obstacle to reach the food source. If one way is longer than the other, the shorter route is in favour. The ants releases pheromones as it moves, which are detectable by other ants. The more pheromones are on a tour, the more likely this tour is preferred by the following ants. These pheromones have a rate of evaporation.

**Colorni, Dorigo, and Maniezzo's modelling of the Ant System**

Let $b_i(t)$ be the number of ants in town $i$ (vertex $v_i$) at time $t$ with $i = (1, 2, 3...n)$. Let $m$ be the total number of ants:

$$m = \sum_{i=1}^{n} b_i(t) \quad \textbf{(14)}$$

This algorithm should start with the same number of ants as towns, as seen in in the original text. At each town, there should only be one ant. There are other possibilities, where the number of ants is larger or smaller than the number of towns, or the ants are not equally distributed over the towns. Let path$_{(ij)}$ (edge $(v_i, v_j)$) be the shortest path between towns i and j. In the original paper, the Euclidian distance of path$_{(ij)}$ is:

$$d_{(ij)} = \sqrt{(x_1^i - x_1^j)^2 + (x_2^i - x_2^j)^2} \quad \textbf{(15)}$$

Let $\Delta\tau^k_{(ij)}(t, t+1)$ be the quantity per unit of length of pheromones laid on path$_{(ij)}$ by the k-th ant between time $t$ and $t+1$. The sum of all quantities of pheromone laid by all ants on path$_{(ij)}$ between time $t$ and $t+1$ is:

$$\Delta\tau_{(ij)}(t, t+1) = \sum_{k=1}^{m} \Delta\tau^k_{(ij)}(t, t+1) \quad \textbf{(16)}$$

$\rho$ represents the evaporation coefficient, which determines the rate at which the pheromone evaporates. The evaporation of the pheromones makes the paths less attractive to ants.
With this, let $\tau_{(ij)}$ be the trail's intensity of the pheromones on path$_{(ij)}$ at time $t+1$. This is the actual amount of pheromone lying on the path, rather than the ant's perceived amount of pheromones:

$$\tau_{(ij)}(t+1) = \rho\tau_{(ij)}(t) + \Delta\tau_{(ij)}(t, t+1) \quad \textbf{(17)}$$

This consists of two summands: the first summand represents the amount of the remaining pheromones laid up to time $t$ after evaporation, and the second summand represents the new amount of pheromones laid by one or more ants on path$_{(ij)}$ between the time $t$ and $t+1$.
At $t = 0$, $\tau_{(ij)}(0)$ should be very small (or 0) on path$_{(ij)}$. Let $\eta_{(ij)}$ be the visibility. This is the modelled ant's ability to perceive the intensity of the pheromones. In this case, it is defined as:

$$\eta_{(ij)} = \frac{1}{d_{(ij)}} \quad \textbf{(18)}$$

With this, the transition probability $p_{ij}(t)$, which is the probability that an ant chooses the path from $i$ to $j$, is:

$$p_{(ij)}(t) = \frac{[\tau_{(ij)}(t)]^{\alpha}[\eta_{(ij)}]^{\beta}}{\sum\limits_{j=1}^{n}[\tau_{(ij)}(t)]^{\alpha}[\eta_{(ij)}]^{\beta}} \quad \textbf{(19)}$$

$\alpha$ and $\beta$ are parameters which allow users to control whether the actual amount of pheromones is more or less important than the perceived amount of pheromones on path$_{(ij)}$. Of course, for an ant at town $i$, there are several transition probabilities, as there are multiple towns that an ant can choose to go to. The ant chooses the path with the greatest transition probability, which is a fraction. The numerator is the perceived amount of the intensity of pheromones on path$_{(ij)}$. To avoid that ant visits a previously-visited town, a tabu list is created for every ant. This tabu list contains the towns which it had visited up to the time $t$.

**The three ways of modelling $\Delta\tau^{k}_{(ij)}(t, t+1)$**

The original text describes three ways of modelling $\Delta\tau^{k}_{(ij)}(t, t+1)$, which are Ant-quantity, Ant-density, and ant cycle.
In the Ant-quantity model, $Q_1$ is a constant quantity which represents the pheromones left on path$_{(ij)}$. $Q_1$ is independent of the length of the path$_{(ij)}$ and is not a function of $\mathrm{d}_{(ij)}$.

$$\Delta\tau^{k}_{(ij)}(t, t+1) = \left\{ \begin{array}{ll} \frac{Q_1}{d_{(ij)}} & \text{if k-th ant goes from i to j between time t ant t+1} \\ 0 & \text{otherwise} \end{array} \right\} \quad \textbf{(20)}$$

In the Ant-density model, $Q_2$ is a function of $\mathrm{d}_{ij}$ and gives the number of units of pheromones left on path$_{(ij)}$ for each unit of length.

$$\Delta\tau^{k}_{(ij)}(t, t+1) = \left\{ \begin{array}{ll} Q_2 & \text{if k-th ant goes from i to j between time t ant t+1} \\ 0 & \text{otherwise} \end{array} \right\} \quad \textbf{(21)}$$

In the Ant cycle model, $Q_3$ is a constant quantity which represents the pheromones left on path$_{(ij)}$. $Q_3$ is independent of the length of the path$_{(ij)}$, and is not a function of $\mathrm{d}_{(ij)}$.

$$\Delta\tau^{k}_{(ij)}(t, t+1) = \left\{ \begin{array}{ll} \frac{Q_3}{L^k} & \text{if k-th ant goes from i to j between time t ant t+1} \\ 0 & \text{otherwise} \end{array} \right\} \quad \textbf{(22)}$$

In the Ant-quantity and Ant-density models, $\Delta\tau^{k}_{(ij)}(t, t+1)$ is recalculated after each step is completed, so the transition probabilities also change after each step. In the Ant cycle model, $\Delta\tau^{k}_{(ij)}(t, t+1)$ and the transition probabilities are only recalculated at the end of a cycle. $L^k$ is the shortest tour. $Q_3$ is a constant.

This is a big difference, as the transition probabilities stay the same throughout a cycle. Whilst this does not represent nature, it was shown to be more efficient, therefore it was preferred by the authors. Now as all equations are established, a small amount of pheromone is distributed on all arcs between all nodes and at each town or node (i) an ant is placed. This town is written in the ant's tabu list. In the first step, the ants decide (as seen in the NNA) which path or arc to cross, according to equation **(19)**. $Q_1$ is placed on the path and the transition probability is calculated using equations **(16)**, **(17)**, and **(19)**. The ants make now the next decision, and this is repeated until all ants have visited all the towns, which results in an full tabu list. One cycle is finished. Now the ants are put back in their starting positions and then make their first decision. The transition probabilities are calculated. This is also repeated until the tabu list is filled. In the original work several hundred cycles had to be completed (Colorni, Dorigo and Maniezzo, 1992).

**Which influence does $\rho$, $\eta_{(ij)}$, $Q_1$, $\alpha$, and $\beta$ have on $p_{(ij)}(t)$?**

To demonstrate the influences each of the factors have, I chose to rewrite equation **(19)**:

$$p_{ij}(t) = \frac{[\tau_{(ij)}(t)]^{\alpha}[\eta_{(ij)}]^{\beta}}{\sum_{j=1}^{n}[\tau_{(ij)}(t)]^{\alpha}[\eta_{(ij)}]^{\beta}} = \frac{[\rho\tau_{ij}(t-1) + \frac{Q_1}{d_{ij}}]^{\alpha}[\frac{1}{d_{ij}}]^{\beta}}{\sum_{j=1}^{n}[\rho\tau_{ij}(t-1) + \frac{Q_1}{d_{ij}}]^{\alpha}[\frac{1}{d_{ij}}]^{\beta}} \quad \textbf{(23)}$$

With this equation, it will be much easier to establish the influences of each of the factors listed.

Influence of $\rho$:

The evaporation coefficient indicates the influence of the previously-laid pheromones on the transition probability. As $\rho$ increases, $p_{ij}(t)$ increases. This can result in a deadlock. A deadlock means, in this case, that an ant is unable to leave a non-optimal tour, as its ability to learn by trial and error is greatly reduced. If $\rho$ is very small, then the first summand of the first multiplicand of numerator approaches 0. This means that the ants would not take into account which tours were previously chosen, meaning the ant cannot remember which paths were optimal, and therefore chooses its paths randomly.

Influence of $\alpha$:

As $\alpha$ is an exponent the user can change the influence of $\rho\tau_{(ij)}(t)$ on the transition probability. If $\alpha = 0$ the ant would have no ability to learn and the decision only depends on the distance of two towns, as seen in the NNA.

Influence of $\eta_{ij}$:

In this case, as $\eta_{ij}$ increases, $p_{ij}(t)$ increases, as it is a multiplicand. Something important to note is that $\eta_{ij}$ is the reciprocal of the distance. As the distance increases, $p_{ij}(t)$ decreases. Here the distance between two towns has an influence on the decision to choose the next path.

Influence of $\beta$:

As seen with $\alpha$, $\beta$ weakens the influence of $\eta_{ij}$ on the transition probability.

Influence of $Q_1$:

As $Q_1$ is the numerator of the second summand of the multiplicand, which models the learning ability, it has an strong influence on the ants' ability to learn. If $Q_1$ increases, the influence of $\rho\tau_{(ij)}(t)$ decreases, meaning that the ability to learn from the past is reduced. If $Q_1$ is small the ant's decision to choose a path has no influence on the swarm's intelligence. Its decision becomes less important with a smaller $Q_1$.

The denominator weakens all decisions of individual ants.

**Ant colony optimisation after Colorni, Dorigo, and Maniezzo**

Ant Colony Optimisation is an algorithm, which was based on the Ant System (Ant cycle model). Four changes were made to the Ant cycle model to create Ant Colony Optimisation. Firstly, only the ant with the shortest tour places pheromones on the paths after a cycle is completed. This is more efficient, as ants with longer tours will have no influence on the learning process. Secondly, whenever an ant crosses an arc, the amount of pheromones on that path decreases.

This also quickens the learning process. In the original model, as the number of ants which use a path increases, the amount of pheromones on that path increases, and the amount of pheromones can only decrease through evaporation. This can result in a deadlock. This deadlock is avoided by the new model, as when a sub-optimal path is taken by several ants, the amount of pheromones on that path decreases. This means that new tours can be created more quickly. Thirdly, for every town, a tabu list, which contains the nearest towns, is generated. For a small number of towns, this results in additional runtime, but with a TSP with several hundred towns, this makes more sense, as not every town should be checked. This could also result in a deadlock, as shown above, since long arcs are not included. This results in a local search. Fourthly, rather than the transition probability, a more complex function with two new variables is used:

$$j = \left\{ \begin{array}{ccc} argmax_{u \in J_k^i}\{\tau_{(iu)}(t)[\eta_{(iu)}]^{\beta}\} & for & q \leq q_0 \\ J & for & q > q_0 \end{array} \right\} \quad \mathbf{(24)}$$

The variable $q$ is chosen randomly by the program and has a value in the interval $[0,1]$. $q_0$ is defined by the user and is also chosen from the interval $[0,1]$. It determines whether the first or second line is taken to determine the next town to be visited ($j$) by ant $k$ at town $i$. If $q_0 = 0.5$, both lines have the same chance of being taken. If $q > q_0$, $j$ is determined as seen in the Ant system equation **(19)**. If $q \leq q_0$ a local search starts. $argmax_{u \in J_k^i}\{\tau_{(iu)}(t)[\eta_{(iu)}]^{\beta}\}$ determines the next town to be visited, and is based on visibility and the quality of the previously-found solutions.

**A new Ant Algorithm:**

For my purpose, I did not use the euclidian distances as seen in equation **(15)**, but rather the real distances, which were found using Google maps, as this seems more reasonable. I also decided to create a new Ant Algorithm which uses a very simple term for the transition probability. The denominator is the sum of all perceived amounts of the intensities of pheromones on the paths to the reachable towns (in the computer program, these are the cities which have not yet been visited) and weakens only the influence of individual decisions on the swarm's intelligence. After some consideration, I avoided not only this division, but also the sum within the denominator which is runtime-consuming for a computer. Hence, my transition probability is defined as:

$$p_{(ij)}(t) = \frac{[\tau_{(ij)}(t)]^{\alpha}[\eta_{(ij)}]^{\beta}}{1} = [\tau_{(ij)}(t)]^{\alpha}[\eta_{(ij)}]^{\beta} \quad \mathbf{(25)}$$

Furthermore, I decide to let the influence of knowledge from the past and the distance (greed) be the same, which results in $\alpha = \beta$. I set them both to:

$$\alpha = \beta = 1 \quad \mathbf{(26)}$$

With this, the transition probability is:

$$p_{(ij)}(t) = \tau_{(ij)}(t)\eta_{(ij)} \quad \mathbf{(27)}$$

The problem with having a constant initial concentration of pheromones before any ant starts is of great importance, especially since this problem is has already been seen in the Nearest Neighbour Algorithm. If the ants' first decision is based on greed, this leads to a non-optimal decision, which has a great impact on the other ants' decision, because the pheromones must first evaporate before this mistake is 'forgotten'. Under certain circumstances, it is not possible for the swarm to find the shortest tour. In my algorithm the first decision is not based on the length of a path but on random choice. This enables the swarm to find shorter tours more quickly, since the first decisions were not made according to the transition probabilities, which are based on distances between the towns, which results in the same deadlock problem found in the Nearest Neighbour Algorithm. All of these alterations have been made to make a very fast program block, which will be repeated several times.

**How to implement the Ant Algorithm into a program**

It is clear that the Ant Algorithm cannot be carried out by hand, as it is prone to mistakes, and it would probably take years to solve the TSP for thirteen towns, as in our task. Therefore, a program was written and altered several times to our desire. Each town's name is replaced by a number from 0 to 12. Each ant's name is also a number from 0 to 12. In the initial block, ant 0 is placed at town 0, ant 1 is placed at town 1, and so on. My program is very versatile, and does need not need any pheromones at the initial state. This is different from the original work on the Ant Algorithm, and has a great impact on the results. I have chosen a set of 13 randomly chosen numbers, so that the set contains all thirteen numbers, and every ant receives a number (the town to visit), which is different from their name. For example, ant 0, which started at town 0, receives a town number between 1 and 12, and ant 7 at town 7 receives a number from 0 to 12, but not 7. This randomising does not exist in the original program flow from Colorni, Dorgi, and Maniezzo. In the first step of the first cycle, they decided to visit the next town according to the transition probability, which depends on the length. Therefore, the ant would choose to arrive at the nearest town in the first step of the program. This has the same problem as seen in the Nearest Neighbour Algorithm, and therefore has the same disadvantages, especially the inability to find the minimum tour length. To prevent this, I have chosen another approach, which is more similar to nature, because no hormones should be found in a network which no ants have visited. It is comfortable and easier to start with an initial concentration of pheromones, as seen in the original work, so no division by zero occurs in equation **(17)**. I chose to stick to nature, and as there are no pheromones in the network, and the ant has no knowledge of the lengths between the towns, their first move is determined randomly. This is much more difficult to program, but this allows the possibility that the decision to visit the second town is not determined by the distance between the first and second towns, and hence does not show the disadvantage seen in the Nearest Neighbour Algorithm. For every ant, there is also a tabu list. This contains the numbers of towns which were already visited. A second list which contains the choosable towns also exists. For example, ant 5 starts its journey at town 5, and receives town 7 as its next destination. In the tabu list, 5 and 7 are noted. The list of towns which were not visited now contains 0, 1, 2, 3, 4, 6, 8, 10, 11, and 12. In the next step, pheromones have been laid on the path between towns 5 and 7. In the initial block of the program, the visibility is calculated for all possible paths and is stored in an array, since the visibilities are constant.

In my case, $Q_1$ was chosen as 10000 and was divided by the distance between towns 5 and 7, to calculate $\tau_{5,7}(1)$. With this, the transition probability is calculated according to formula **(27)**. Ant 5, which is now at town 7, looks up its non-visited list, determines the transition probabilities $\{p_{7,0}, p_{7,1}, p_{7,2}, p_{7,3}, p_{7,4}, p_{7,6}, p_{7,8}, ...\}$, and chooses the largest value. The largest transition probability indicates the next town to visit. This is finished when ant 5 has visited all towns. This is also different from the original work, because we store the total length of ant 5's route.

This procedure occurs with the twelve other ants simultaneously. After all ants have visited all towns, the next cycle begins. The pheromones have evaporated, according to equation **(17)**, the tabu list is emptied, and the unvisited list is filled up with all towns. Ant 0 is placed at town 0, ant 1 is placed at town 1, and so on, but in this cycle, the next town to be visited is not randomly chosen. The town to be visited is chosen based on its transition probability. The transition probabilities are calculated for every step, and the cycle is ended when every ant has visited every town. Again, the best ant is chosen with the shortest tour length. This is compared to the shortest tour in the previous cycle, and if the length is shorter, then the length and its tabu list are stored. The user can choose how many times this cycle is repeated. In the original text, several hundred tours were chosen, but my program shows that even after a few cycles the optimum tour, which we have previously determined through brute force, can be found. This is a very important difference compared to the original program flow. In the original program, there is no output and no comparison between the tours whilst the program is running. The only output is seen after all cycles have been completed. As I was writing the program, I ensured that the output of every step was displayed.

Examples of the output can be seen in Figures **(5)** and **(6)**. There seems to be two types of results: the first type (Figure **(5)**) simply starts from a longer tour length, leading to a shorter tour, which does not change anymore, until all cycles have been completed. The second type (Figure **(6)**) is much more interesting: first, there was a tour length, which led to a minimum tour length, but then, the algorithm tended to a less optimal solution, which was maintained, until all cycles were completed.

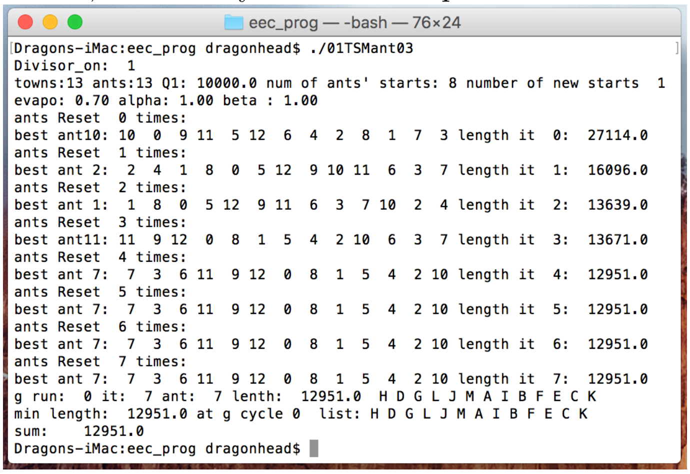



**Figure 5:** Output 1

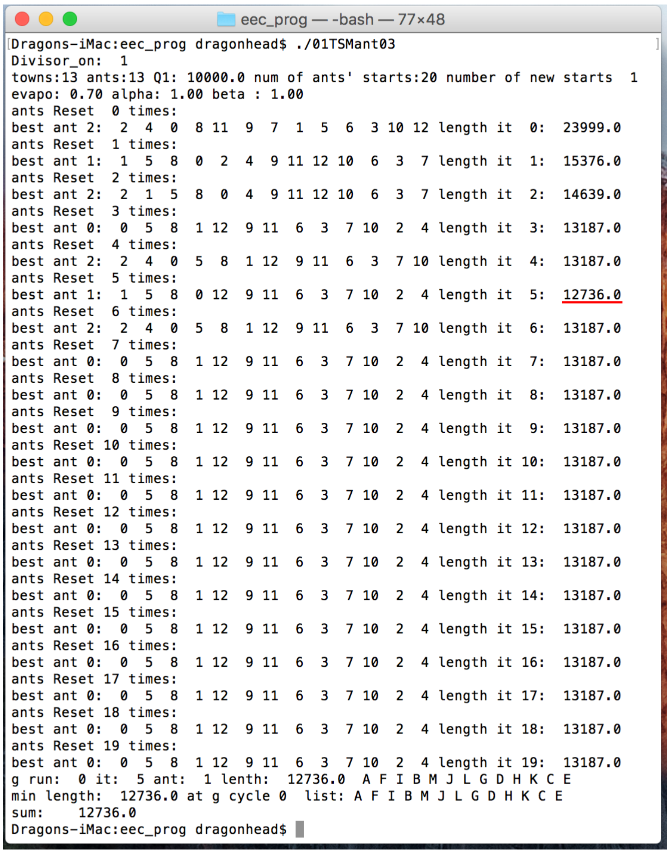
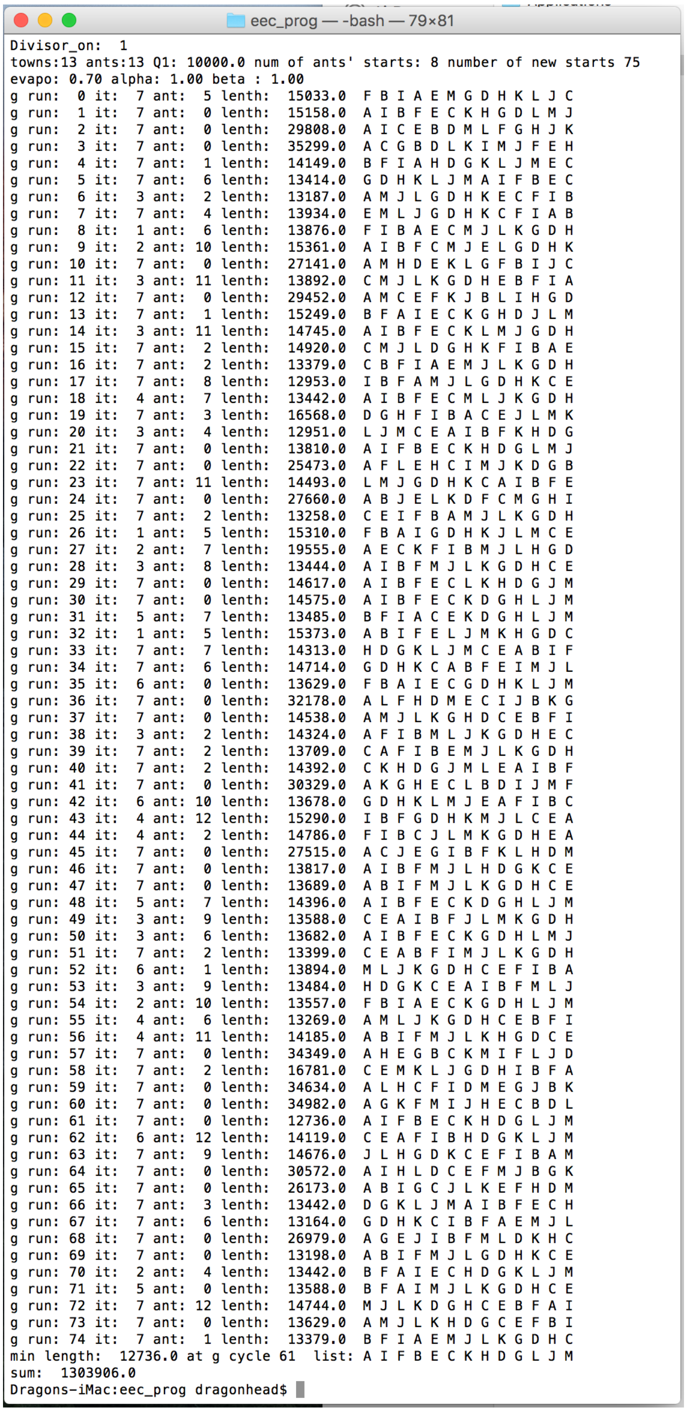



**Figure 6:** Output 2

The algorithm obviously finds the shortest possible tour length, but does not display it, if the original Ant algorithm is used. Therefore, I decided to change the original program, so that every tour length is checked and compared to all others, regardless of whether a cycle was completed, or all cycles were completed. Instead of several hundred cycles, as seen in the original code, my program needs less than ten cycles, but the program is written to start a series of cycles for less than 100 times, and almost always results in the minimum tour length calculated by brute force. I found out that the number of cycles a series consists of is not important, but rather how often the series is restarted, and the second step, which is determined randomly. The probability that this tour is found is $1 \cdot 10^{-6}$. In Appendix **A4** a program flow can be found, and the program is under Appendix **A3**.

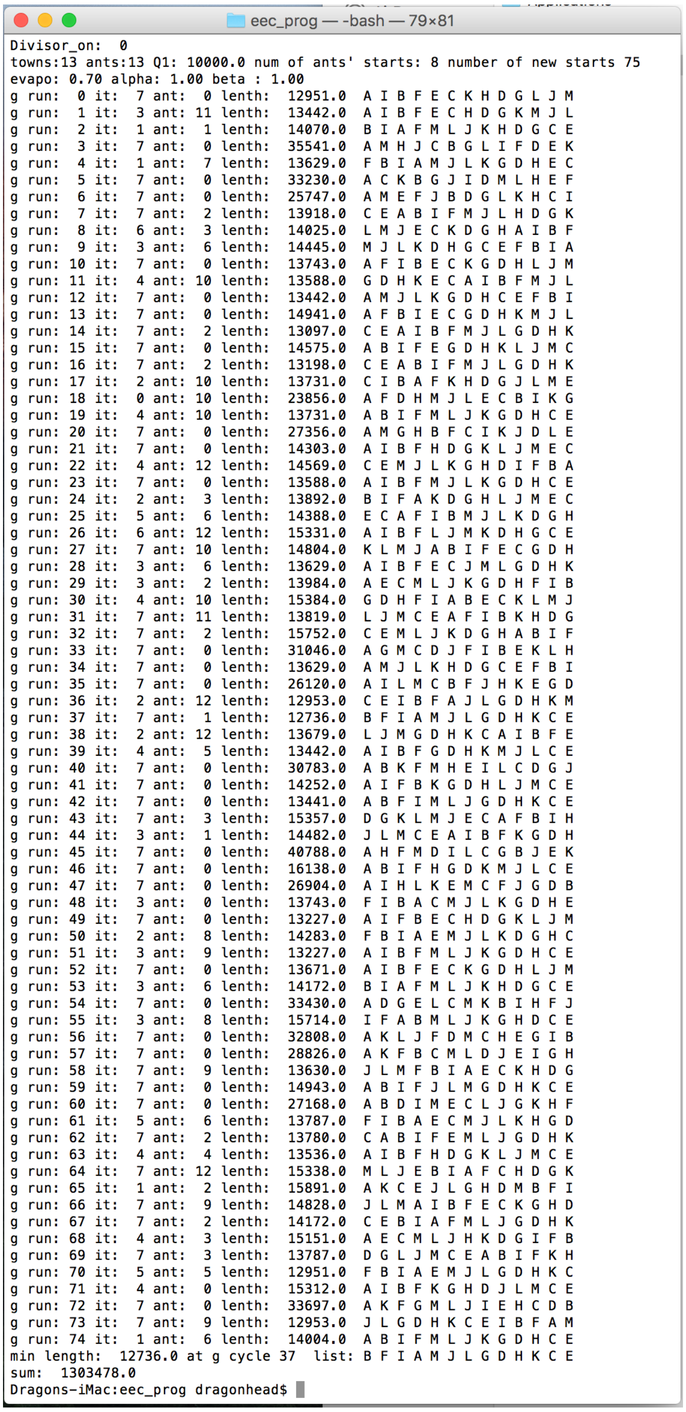
**Discussion of the output from the new Ant System Program**

In Figure **(5)** and Figure **(6)** there are outputs from the intermediate results from my new Ant System program, which demonstrates that it is possible to find the shortest solution (12736 km) after the fifth iteration. It also shows the ants' ability to learn. At iteration 0, for the 13 ants at the 13 towns, the next towns visited are determined by random numbers generated by the computer. For example, our best ant at this iteration is ant 2, who is placed at town 2, and is forced to visit town 4 through random number generation, because no pheromones were previously distributed. Afterwards, ant 2 visits (as a result of the transition probabilities) town 0, 8, 11, 9, and so on, until the tabu list is filled. With the ants' tabu lists, the distances of the tours are calculated and compared to each other. The length of the shortest tour and its tabu list are stored for further comparison. Now, all tabu lists are cleared, and all ants are placed at their original towns. The output shows this in the line 'ants Reset 1 times:' This time, and the following 19 times, from the beginning, the ants choose the next paths through transition probability, as pheromones have been laid. From iteration 0 to iteration 5, the shortest length decreases to the minimum of 12736 km, which was determined by brute force. After this, at iteration 6, the shortest tour length increases to 13187 km. This shows that even if the ants have already found the minimum tour length, the program enables them to choose a less optimal tour. In the normal Ant System, this solution would not be stored and compared - it would be lost. The user would only see the output from the last iteration, iteration 19, which is 13187 km.



```
Divisor_on:  1
towns:13 ants:13 Q1: 10000.0 num of ants' starts: 8 number of new starts 75
evapo: 0.70 alpha: 1.00 beta : 1.00
g run:  0 it:  7 ant:  5 lenth:  15033.0  F B I A E M G D H K L J C
g run:  1 it:  7 ant:  0 lenth:  15158.0  A I B F E C K H G D L M J
g run:  2 it:  7 ant:  0 lenth:  29808.0  A I C E B D M L F G H J K
g run:  3 it:  7 ant:  0 lenth:  35299.0  A C G B D L K I M J F E H
g run:  4 it:  7 ant:  1 lenth:  14149.0  B F I A H D G K L J M E C
g run:  5 it:  7 ant:  6 lenth:  13414.0  G D H K L J M A I F B E C
g run:  6 it:  3 ant:  2 lenth:  13187.0  A M J L G D H K E C F I B
g run:  7 it:  7 ant:  4 lenth:  13934.0  E M L J G D H K C F I A B
g run:  8 it:  1 ant:  6 lenth:  13876.0  F I B A E C M J L K G D H
g run:  9 it:  2 ant: 10 lenth:  15361.0  A I B F C M J E L G D H K
g run: 10 it:  7 ant:  0 lenth:  27141.0  A M H D E K L G F B I J C
g run: 11 it:  3 ant: 11 lenth:  13892.0  C M J L K G D H E B F I A
g run: 12 it:  7 ant:  0 lenth:  29452.0  A M C E F K J B L I H G D
g run: 13 it:  7 ant:  1 lenth:  15249.0  B F A I E C K G H D J L M
g run: 14 it:  3 ant: 11 lenth:  14745.0  A I B F E C K L M J G D H
g run: 15 it:  7 ant:  2 lenth:  14920.0  C M J L D G H K F I B A E
g run: 16 it:  7 ant:  2 lenth:  13379.0  C B F I A E M J L K G D H
g run: 17 it:  7 ant:  8 lenth:  12953.0  I B F A M J L G D H K C E
g run: 18 it:  4 ant:  7 lenth:  13442.0  A I B F E C M L J K G D H
g run: 19 it:  7 ant:  3 lenth:  16568.0  D G H F I B A C E J L M K
g run: 20 it:  3 ant:  4 lenth:  12951.0  L J M C E A I B F K H D G
g run: 21 it:  7 ant:  0 lenth:  13810.0  A I F B E C K H D G L M J
g run: 22 it:  7 ant:  0 lenth:  25473.0  A F L E H C I M J K D G B
g run: 23 it:  7 ant: 11 lenth:  14493.0  L M J G D H K C A I B F E
g run: 24 it:  7 ant:  0 lenth:  27660.0  A B J E L K D F C M G H I
g run: 25 it:  7 ant:  2 lenth:  13258.0  C E I F B A M J L K G D H
g run: 26 it:  1 ant:  5 lenth:  15310.0  F B A I G D H K J L M C E
g run: 27 it:  2 ant:  7 lenth:  19555.0  A E C K F I B M J L H G D
g run: 28 it:  3 ant:  8 lenth:  13444.0  A I B F M J L K G D H C E
g run: 29 it:  7 ant:  0 lenth:  14617.0  A I B F E C L K H D G J M
g run: 30 it:  7 ant:  0 lenth:  14575.0  A I B F E C K D G H L J M
g run: 31 it:  5 ant:  7 lenth:  13485.0  B F I A C E K D G H L J M
g run: 32 it:  1 ant:  5 lenth:  15373.0  A B I F E L J M K H G D C
g run: 33 it:  7 ant:  7 lenth:  14313.0  H D G K L J M C E A B I F
g run: 34 it:  7 ant:  6 lenth:  14714.0  G D H K C A B F E I M J L
g run: 35 it:  6 ant:  0 lenth:  13629.0  F B A I E C G D H K L J M
g run: 36 it:  7 ant:  0 lenth:  32178.0  A L F H D M E C I J B K G
g run: 37 it:  7 ant:  0 lenth:  14538.0  A M J L K G H D C E B F I
g run: 38 it:  3 ant:  2 lenth:  14324.0  A F I B M L J K G D H E C
g run: 39 it:  7 ant:  2 lenth:  13709.0  C A F I B E M J L K G D H
g run: 40 it:  7 ant:  2 lenth:  14392.0  C K H D G J M L E A I B F
g run: 41 it:  7 ant:  0 lenth:  30329.0  A K G H E C L B D I J M F
g run: 42 it:  6 ant: 10 lenth:  13678.0  G D H K L M J E A F I B C
g run: 43 it:  4 ant: 12 lenth:  15290.0  I B F G D H K M J L C E A
g run: 44 it:  4 ant:  2 lenth:  14786.0  F I B C J L M K G D H E A
g run: 45 it:  7 ant:  0 lenth:  27515.0  A C J E G I B F K L H D M
g run: 46 it:  7 ant:  0 lenth:  13817.0  A I B F M J L H D G K C E
g run: 47 it:  7 ant:  0 lenth:  13689.0  A B I F M J L K G D H C E
g run: 48 it:  5 ant:  7 lenth:  14396.0  A I B F E C K D G H L J M
g run: 49 it:  3 ant:  9 lenth:  13588.0  C E A I B F J L M K G D H
g run: 50 it:  3 ant:  6 lenth:  13682.0  A I B F E C K G D H L M J
g run: 51 it:  7 ant:  2 lenth:  13399.0  C E A B F I M J L K G D H
g run: 52 it:  6 ant:  1 lenth:  13894.0  M L J K G D H C E F I B A
g run: 53 it:  3 ant:  9 lenth:  13484.0  H D G K C E A I B F M L J
g run: 54 it:  2 ant: 10 lenth:  13557.0  F B I A E C K G D H L J M
g run: 55 it:  4 ant:  6 lenth:  13269.0  A M L J K G D H C E B F I
g run: 56 it:  4 ant: 11 lenth:  14185.0  A B I F M J L K H G D C E
g run: 57 it:  7 ant:  0 lenth:  34349.0  A H E G B C K M I F L J D
g run: 58 it:  7 ant:  2 lenth:  16781.0  C E M K L J G D H I B F A
g run: 59 it:  7 ant:  0 lenth:  34634.0  A L H C F I D M E G J B K
g run: 60 it:  7 ant:  0 lenth:  34982.0  A G K F M I J H E C B D L
g run: 61 it:  7 ant:  0 lenth:  12736.0  A I F B E C K H D G L J M
g run: 62 it:  6 ant: 12 lenth:  14119.0  C E A F I B H D G K L J M
g run: 63 it:  7 ant:  9 lenth:  14676.0  J L H G D K C E F I B A M
g run: 64 it:  7 ant:  0 lenth:  30572.0  A I H L D C E F M J B G K
g run: 65 it:  7 ant:  0 lenth:  26173.0  A B I G C J L K E F H D M
g run: 66 it:  7 ant:  3 lenth:  13442.0  D G K L J M A I B F E C H
g run: 67 it:  7 ant:  6 lenth:  13164.0  G D H K C I B F A E M J L
g run: 68 it:  7 ant:  0 lenth:  26979.0  A G E J I B F M L D K H C
g run: 69 it:  7 ant:  0 lenth:  13198.0  A B I F M J L G D H K C E
g run: 70 it:  2 ant:  4 lenth:  13442.0  B F A I E C H D G K L J M
g run: 71 it:  5 ant:  0 lenth:  13588.0  B F A I M J L K G D H C E
g run: 72 it:  7 ant: 12 lenth:  14744.0  M J L K D G H C E B F A I
g run: 73 it:  7 ant:  0 lenth:  13629.0  A M J L K H D G C E F B I
g run: 74 it:  7 ant:  1 lenth:  13379.0  B F I A E M J L K G D H C
min length:  12736.0 at g cycle 61  list: A I F B E C K H D G L J M
sum:  1303906.0
Dragons-iMac:eec_prog dragonhead$
```

**Figure 7:** Output 3



```
Divisor_on:  0
towns:13 ants:13 Q1: 10000.0 num of ants' starts: 8 number of new starts 75
evapo: 0.70 alpha: 1.00 beta : 1.00
g run:  0 it:  7 ant:  0 lenth:  12951.0  A I B F E C K H D G L J M
g run:  1 it:  3 ant: 11 lenth:  13442.0  A I B F E C H D G K M J L
g run:  2 it:  1 ant:  1 lenth:  14070.0  B I A F M L J K H D G C E
g run:  3 it:  7 ant:  0 lenth:  35541.0  A M H J C B G L I F D E K
g run:  4 it:  1 ant:  7 lenth:  13629.0  F B I A M J L K G D H E C
g run:  5 it:  7 ant:  0 lenth:  33230.0  A C K B G J I D M L H E F
g run:  6 it:  7 ant:  0 lenth:  25747.0  A M E F J B D G L K H C I
g run:  7 it:  7 ant:  2 lenth:  13918.0  C E A B I F M J L H D G K
g run:  8 it:  6 ant:  3 lenth:  14025.0  L M J E C K D G H A I B F
g run:  9 it:  3 ant:  6 lenth:  14445.0  M J L K D H G C E F B I A
g run: 10 it:  7 ant:  0 lenth:  13743.0  A F I B E C K G D H L J M
g run: 11 it:  4 ant: 10 lenth:  13588.0  G D H K E C A I B F M J L
g run: 12 it:  7 ant:  0 lenth:  13442.0  A M J L K G D H C E F B I
g run: 13 it:  7 ant:  0 lenth:  14941.0  A F B I E C G D H K M J L
g run: 14 it:  7 ant:  2 lenth:  13097.0  C E A I B F M J L G D H K
g run: 15 it:  7 ant:  0 lenth:  14575.0  A B I F E G D H K L J M C
g run: 16 it:  7 ant:  2 lenth:  13198.0  C E A B I F M J L G D H K
g run: 17 it:  2 ant: 10 lenth:  13731.0  C I B A F K H D G J L M E
g run: 18 it:  0 ant: 10 lenth:  23856.0  A F D H M J L E C B I K G
g run: 19 it:  4 ant: 10 lenth:  13731.0  A B I F M L J K G D H C E
g run: 20 it:  7 ant:  0 lenth:  27356.0  A M G H B F C I K J D L E
g run: 21 it:  7 ant:  0 lenth:  14303.0  A I B F H D G K L J M E C
g run: 22 it:  4 ant: 12 lenth:  14569.0  C E M J L K G H D I F B A
g run: 23 it:  7 ant:  0 lenth:  13588.0  A I B F M J L K G D H C E
g run: 24 it:  2 ant:  3 lenth:  13892.0  B I F A K D G H L J M E C
g run: 25 it:  5 ant:  6 lenth:  14388.0  E C A F I B M J L K D G H
g run: 26 it:  6 ant: 12 lenth:  15331.0  A I B F L J M K D H G C E
g run: 27 it:  7 ant: 10 lenth:  14804.0  K L M J A B I F E C G D H
g run: 28 it:  3 ant:  6 lenth:  13629.0  A I B F E C J M L G D H K
g run: 29 it:  3 ant:  2 lenth:  13984.0  A E C M L J K G D H F I B
g run: 30 it:  4 ant: 10 lenth:  15384.0  G D H F I A B E C K L M J
g run: 31 it:  7 ant: 11 lenth:  13819.0  L J M C E A F I B K H D G
g run: 32 it:  7 ant:  2 lenth:  15752.0  C E M L J K D G H A B I F
g run: 33 it:  7 ant:  0 lenth:  31046.0  A G M C D J F I B E K L H
g run: 34 it:  7 ant:  0 lenth:  13629.0  A M J L K H D G C E F B I
g run: 35 it:  7 ant:  0 lenth:  26120.0  A I L M C B F J H K E G D
g run: 36 it:  2 ant: 12 lenth:  12953.0  C E I B F A J L G D H K M
g run: 37 it:  7 ant:  1 lenth:  12736.0  B F I A M J L G D H K C E
g run: 38 it:  2 ant: 12 lenth:  13679.0  L J M G D H K C A I B F E
g run: 39 it:  4 ant:  5 lenth:  13442.0  A I B F G D H K M J L C E
g run: 40 it:  7 ant:  0 lenth:  30783.0  A B K F M H E I L C D G J
g run: 41 it:  7 ant:  0 lenth:  14252.0  A I F B K G D H L J M C E
g run: 42 it:  7 ant:  0 lenth:  13441.0  A B F I M L J G D H K C E
g run: 43 it:  7 ant:  3 lenth:  15357.0  D G K L M J E C A F B I H
g run: 44 it:  3 ant:  1 lenth:  14482.0  J L M C E A I B F K G D H
g run: 45 it:  7 ant:  0 lenth:  40788.0  A H F M D I L C G B J E K
g run: 46 it:  7 ant:  0 lenth:  16138.0  A B I F H G D K M J L C E
g run: 47 it:  7 ant:  0 lenth:  26904.0  A I H L K E M C F J G D B
g run: 48 it:  3 ant:  0 lenth:  13743.0  F I B A C M J L K G D H E
g run: 49 it:  7 ant:  0 lenth:  13227.0  A I F B E C H D G K L J M
g run: 50 it:  2 ant:  8 lenth:  14283.0  F B I A E M J L K D G H C
g run: 51 it:  3 ant:  9 lenth:  13227.0  A I B F M L J K G D H C E
g run: 52 it:  7 ant:  0 lenth:  13671.0  A I B F E C K G D H L J M
g run: 53 it:  3 ant:  6 lenth:  14172.0  B I A F M L J K H D G C E
g run: 54 it:  7 ant:  0 lenth:  33430.0  A D G E L C M K B I H F J
g run: 55 it:  3 ant:  8 lenth:  15714.0  I F A B M L J K G H D C E
g run: 56 it:  7 ant:  0 lenth:  32808.0  A K L J F D M C H E G I B
g run: 57 it:  7 ant:  0 lenth:  28826.0  A K F B C M L D J E I G H
g run: 58 it:  7 ant:  9 lenth:  13630.0  J L M F B I A E C K H D G
g run: 59 it:  7 ant:  0 lenth:  14943.0  A B I F J L M G D H K C E
g run: 60 it:  7 ant:  0 lenth:  27168.0  A B D I M E C L J G K H F
g run: 61 it:  5 ant:  6 lenth:  13787.0  F I B A E C M J L K H G D
g run: 62 it:  7 ant:  2 lenth:  13780.0  C A B I F E M L J G D H K
g run: 63 it:  4 ant:  4 lenth:  13536.0  A I B F H D G K L J M C E
g run: 64 it:  7 ant: 12 lenth:  15338.0  M L J E B I A F C H D G K
g run: 65 it:  1 ant:  2 lenth:  15891.0  A K C E J L G H D M B F I
g run: 66 it:  7 ant:  9 lenth:  14828.0  J L M A I B F E C K G H D
g run: 67 it:  7 ant:  2 lenth:  14172.0  C E B I A F M L J G D H K
g run: 68 it:  4 ant:  3 lenth:  15151.0  A E C M L J H K D G I F B
g run: 69 it:  7 ant:  3 lenth:  13787.0  D G L J M C E A B I F K H
g run: 70 it:  5 ant:  5 lenth:  12951.0  F B I A E M J L G D H K C
g run: 71 it:  4 ant:  0 lenth:  15312.0  A I B F K G H D J L M C E
g run: 72 it:  7 ant:  0 lenth:  33697.0  A K F G M L J I E H C D B
g run: 73 it:  7 ant:  9 lenth:  12953.0  J L G D H K C E I B F A M
g run: 74 it:  1 ant:  6 lenth:  14004.0  A B I F M L J K G D H C E
min length:  12736.0 at g cycle 37  list: B F I A M J L G D H K C E
sum:  1303478.0
Dragons-iMac:eec_prog dragonhead$
```

**Figure 8:** Output 4

However, my program compares all tour lengths and finds the minimum tour length of 12736 km after 5 iterations. This means that at a maximum of $6 \cdot 20 = 120$ ant runs, the minimum tour length was found, and the chance of this tour length being found is $\frac{2}{12!} = \frac{1}{239500800}$. My result could not be randomly generated - the result was generated after learning. The great advantage of my program is that this solution is not lost as every tour length is compared, unlike in other programs, where this is not the case. This is most likely the reason why others needs hundreds of iterations with thousands of ant runs. These results have shown that it is a waste of runtime to let the ants run for several hundred times, since an optimal tour will be lost, as it is not stored. In Figure **(5)** and Figure **(6)**, there are outputs where this program block with the second random town occurs only once (the text with the yellow underline states, "number of new starts 1"). In Figure **(6)**, the ants were reseted 19 times (the text with the green underline states, "num of ants' starts 20"). Since the total weight does not change from the sixth iteration onwards, the program block should be stopped after 8 iterations.

However, in Figure **(5)**, the ants were reseted 7 times (the text with the blue underline states, "num of ants' starts 20"). The minimum weight occurs at the fourth iteration, and from here on, the total weight does not change. From this, it occurred to me that the recommended number of iterations should be 8 times.
After this was found out, I wanted to investigate the influence of the denominator. I chose 8 as the number of ant starts, and repeated the block 75 times (the text with the magenta underline states "number of new starts 75"). This means that the second town is chosen randomly 75 times. Additionally, I decided to add up all weights, in order to compare the quality of the results. The outputs are given in Figures **(7)** and **(8)**. From Figure **(7)**, where the divisor is calculated according to equation **(19)**, at the $61^{st}$ cycle, the minimum weight was found. From Figure **(8)**, where the divisor is switched off, according to equation **(27)**, the minimum weight was found at the $37^{th}$ cycle. It is surprising that the minimum is found, when the divisor is switched off. However, it is outstanding that from the comparison of the sum of the weights, it can be seen that the divisor has no impact on the quality of the results. In Figure **(7)**, the sum of all weights found is 1303478, but in Figure **(8)**, the sum of all weights found is 1303906.

## Final Discussion and Conclusion

I showed, using my own program, that the Nearest Neighbour Algorithm results in a very sub-optimal solution $\pi_{NNA} = (C, E, A, I, B, F, M, J, L, K, G, D, H, C)$, which has a weight of $w_{\pi_{NNA}} = 13588$, and the explanation lies in its greedy nature. This was found by comparing this result with the output of a brute force program I wrote, which found a solution of $\pi_{BFmin} = (A, I, F, B, E, C, K, H, D, G, L, J, M, A)$ and $w_{\pi_{BFmin}} = 12736$. It was also shown that parts of both tours were the same. Furthermore, after investigating the effects of the different variables of the transition probability, I showed that the same problem occurs in the original Ant cycle algorithm, and I showed that this problem was covered up through the hundreds of ant runs. In my new algorithm, I avoided these problems, by randomly choosing the first town to be visited by the ants. Additionally, I showed the runtime-consuming division is unnecessary, through the usage of a program, which allowed me to switch the divisor on and off. The two outputs of the program showed that the total sum of the weights is unaffected, and that the shortest tour can be easily found, after around 30 iterations, when the exponents $\alpha = \beta = 1$. This also saves runtime. This resulted in a program based on a new algorithm which successfully found the solution 12736 km after a few iterations. I also displayed these results as a Hamiltonian cycle in a map.
**Applicability** My program can be applied for Travelling Salesman Problems with not only 13 towns, but many more, which could not be determined through brute force due to the large runtime through the enormous number of permutations to check.

## Bibliography


Jameson, S. (2010). Edexcel AS and A Level Modular Mathematics Decision Mathematics 2 D2. Heinemann, pp.62-79.

Nöhring, F. (2007). Cite a Website - Cite This For Me. [online] Thi.uni-hannover.de. Available at: https://www.thi.uni-hannover.de/fileadmin/forschung/arbeiten/noehring-ba.pdf [Accessed 17 Dec. 2017].

Theobald, T. (2007). Skriptum Diskrete Mathematik. [online] Math.uni-frankfurt.de. Available at: http://www.math.uni-frankfurt.de/đmst/teaching/lecture_notes/diskmathAlt/skriptTheobald WS07.pdf [Accessed 9 Dec. 2017].

Colorni, A., Dorigo, M. and Maniezzo, V. (1992). Distributed Optimization by Ant Colonies. [ebook] Milano. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.421.7129&rep=rep1&type=pdf [Accessed 4 Dec. 2017].

Kopp, S., Leßmann, N. and Kranstedt, A. (2003). Ant Colony Optimization Ausarbeitung zum Vortrag im Seminar "Intelligente Algorithmen". [ebook] Available at: https://www.techfak.uni-bielefeld.de/ags/wbski/lehre/digiSA/WS0304/IntAlg/Ausarbeitungen/AntColony.pdf [Accessed 31 Nov. 2017].

Rouse, M. (2006). What is brute force cracking? - Definition from WhatIs.com. [online] SearchSecurity. Available at: http://searchsecurity.techtarget.com/definition/brute-force-cracking [Accessed 3 Jan. 2018].

**Appendix**

```
#include <sys/types.h>
#include <sys/uio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <math.h>
#include <stdio.h>
int main (){
    int a,b,placemin,placeminStart,col,z,mintour,d;
    char memb[13];
    char set[14];
    char minset[14];
    int numbermemb[13];
    char letter[1];
    long length[14][14];//AB,AC,AD,AE...
    long lengthWork[14][14];
    long min=1000000;
    long totallength,mintotallength;
    memb[0]='A';
    memb[1]='B';
    memb[2]='C';
    memb[3]='D';
    memb[4]='E';
    memb[5]='F';
    memb[6]='G';
    memb[7]='H';
    memb[8]='I';
    memb[9]='J';
    memb[10]='K';
    memb[11]='L';
    memb[12]='M';
    numbermemb[0]=0;
    numbermemb[1]=1;
    numbermemb[2]=2;
    numbermemb[3]=3;
    numbermemb[4]=4;
    numbermemb[5]=5;
    numbermemb[6]=6;
    numbermemb[7]=7;
    numbermemb[8]=8;
    numbermemb[9]=9;
    numbermemb[10]=10;
    numbermemb[11]=11;
    numbermemb[12]=12;
    length[0][0]=0;/*AA*/
    length[0][1]=360;/*AB*/
    length[0][2]=1117;/*AC*/
```

```
length[0][3]=4570;/*AD*/
length[0][4]=834;/*AE*/
length[0][5]=701;/*AF*/
length[0][6]=4254;/*AG*/
length[0][7]=4432;/*AH*/
length[0][8]=222;/*AI*/
length[0][9]=2253;/*AJ*/
length[0][10]=2682;/*AK*/
length[0][11]=2437;/*AL*/
length[0][12]=1738;/*AM*/
length[1][0]=360;/*BA*/
length[1][1]=0;/*BB*/
length[1][2]=1262;/*BC*/
length[1][3]=4650;/*BD*/
length[1][4]=1022;/*BE*/
length[1][5]=347;/*BF*/
length[1][6]=4443;/*G*/
length[1][7]=4544;/*H*/
length[1][8]=155;/*I*/
length[1][9]=2605;/*J*/
length[1][10]=2845;/*K*/
length[1][11]=2789;/*L*/
length[1][12]=2090;/*M*/
length[2][0]=1117;/*CA*/
length[2][1]=1262;/*CB*/
length[2][2]=0    ;/*CC*/
length[2][3]=3410;/*CD*/
length[2][4]=474; /*CE*/
length[2][5]=1574;/*CF*/
length[2][6]=3022;/*G*/
length[2][7]=3302;/*H*/
length[2][8]=1214;/*I*/
length[2][9]=1715;/*J*/
length[2][10]=1605;/*K*/
length[2][11]=1792;/*L*/
length[2][12]=1482;/*M*/
length[3][0]=4570;/*DA*/
length[3][1]=4650;/*DB*/
length[3][2]=3410;/*DC*/
length[3][3]=    0;/*DD*/
length[3][4]=3814;/*DE*/
length[3][5]=4958;/*DF*/
length[3][6]=613;/*G*/
length[3][7]=1291;/*H*/
length[3][8]=4600;/*I*/
length[3][9]=3085;/*J*/
length[3][10]=2005;/*K*/
length[3][11]=2813;/*L*/
length[3][12]=3640;/*M*/
length[4][0]=834;/*EA*/
length[4][1]=1022;/*EB*/
```

```
length[4][2]=474;  /*EC*/
length[4][3]=3814;/*ED*/
length[4][4]=0    ;/*EE*/
length[4][5]=1392;/*EF*/
length[4][6]=3480;/*G*/
length[4][7]=3725;/*H*/
length[4][8]=915;/*I*/
length[4][9]=1678;/*J*/
length[4][10]=1907 ;/*K*/
length[4][11]=1806;/*L*/
length[4][12]=1290;/*M*/
length[5][0]=701;/*FA*/
length[5][1]=347;/*FB*/
length[5][2]=1574;/*FC*/
length[5][3]=4958;/*FD*/
length[5][4]=1392;/*FE*/
length[5][5]=0    ;/*FE*/
length[5][6]=4773;/*G*/
length[5][7]=4886;/*H*/
length[5][8]=310;/*I*/
length[5][9]=2957;/*J*/
length[5][10]=3154;/*K*/
length[5][11]=3141;/*L*/
length[5][12]=2442;/*M*/
length[6][0]=4254;/*Ga*/
length[6][1]=4443;/*Gb*/
length[6][2]=3022;/*Gc*/
length[6][3]=613;/*Gd*/
length[6][4]=3480;/*Ge*/
length[6][5]=4773;/*Gf*/
length[6][6]=0;/*Gg*/
length[6][7]=1816;/*Gh*/
length[6][8]=4342;/*Gi*/
length[6][9]=2477;/*Gj*/
length[6][10]=1626;/*Gk*/
length[6][11]=2205;/*Gl*/
length[6][12]=3174;/*Gm*/
length[7][0]=4432;/*Ha*/
length[7][1]=4544;/*Hb*/
length[7][2]=3302;/*Hc*/
length[7][3]=1291;/*Hd*/
length[7][4]=3725;/*He*/
length[7][5]=4886;/*Hf*/
length[7][6]=1816;/*Hg*/
length[7][7]=0;/*Hh*/
length[7][8]=4514;/*Hi*/
length[7][9]=3710;/*Hj*/
length[7][10]=2093;/*Hk*/
length[7][11]=3392;/*Hl*/
length[7][12]=4138;/*Hm*/
length[8][0]=222;/*Ia*/
```

```
length[8][1]=155;/*Ib*/
length[8][2]=1214;/*Ic*/
length[8][3]=4600;/*Id*/
length[8][4]=915;/*Ie*/
length[8][5]=310;/*If*/
length[8][6]=4342;/*Ig*/
length[8][7]=4514;/*Ih*/
length[8][8]=0;/*Ii*/
length[8][9]=2475;/*Ij*/
length[8][10]=2765;/*Ik*/
length[8][11]=2659;/*Il*/
length[8][12]=1960;/*Im*/
length[9][0]=2253;/*Ja*/
length[9][1]=2605;/*Jb*/
length[9][2]=1715;/*Jc*/
length[9][3]=3085;/*Jd*/
length[9][4]=1678;/*Je*/
length[9][5]=2957;/*Jf*/
length[9][6]=2477;/*Jg*/
length[9][7]=3710;/*Jh*/
length[9][8]=2475;/*Ji*/
length[9][9]=0;/*Jj*/
length[9][10]=1646;/*Jk*/
length[9][11]=259;/*Jl*/
length[9][12]=557;/*Jm*/
length[10][0]=2682;/*Ka*/
length[10][1]=2845;/*Kb*/
length[10][2]=1605;/*Kc*/
length[10][3]=2005;/*Kd*/
length[10][4]=1907;/*Ke*/
length[10][5]=3154;/*Kf*/
length[10][6]=1626;/*Kg*/
length[10][7]=2093;/*Kh*/
length[10][8]=2765;/*Ki*/
length[10][9]=1646;/*Kj*/
length[10][10]=0;/*Kk*/
length[10][11]=1466;/*Kl*/
length[10][12]=2077;/*Km*/
length[11][0]=2437;/*La*/
length[11][1]=2789;/*Lb*/
length[11][2]=1792;/*Lc*/
length[11][3]=2813;/*Ld*/
length[11][4]=1806;/*Le*/
length[11][5]=3141;/*Lf*/
length[11][6]=2205;/*Lg*/
length[11][7]=3392;/*Lh*/
length[11][8]=2659;/*Li*/
length[11][9]=259;/*Lj*/
length[11][10]=1466;/*Lk*/
length[11][11]=0;/*Ll*/
length[11][12]=818;/*Lm*/
```

```
length[12][0]=1738;/*Ma*/
length[12][1]=2090;/*Mb*/
length[12][2]=1482;/*Mc*/
length[12][3]=3640;/*Md*/
length[12][4]=1290;/*Me*/
length[12][5]=2442;/*Mf*/
length[12][6]=3174;/*Mg*/
length[12][7]=4138;/*Mh*/
length[12][8]=1960;/*Mi*/
length[12][9]=557;/*Mj*/
length[12][10]=2077;/*Mk*/
length[12][11]=818;/*Ml*/
length[12][12]=0;/*Mm*/
mintotallength=1000000;
for (z=0; z<13;z++){/*Start at all towns for (z=0; z<13;z++){*/
    for (a=0; a<13;a++){
        for (b=0; b<13;b++){lengthWork[a][b]=length[a][b];}
    }
    placemin=z;
    placeminStart=placemin;
    letter[0] = memb[placemin];
    set[0]=memb[placemin];
    totallength=0;
    for (b=0; b<12;b++){/*for (b=0; b<12;b++){*/
        col=placemin;
        min=1000000;
        lengthWork[col][col]=10000;
        for (a=0; a<13;a++){
            if (lengthWork[col][a]<min){
                min=lengthWork[col][a];placemin=a;
            }
        }
        for (a=0; a<13;a++){lengthWork[a][col]=1000000;}
        letter[0] = memb[placemin];
        set[b+1]= memb[placemin];
        totallength=totallength+min;
    }/*for (b=0; b<12;b++){*/
    letter[0] = memb[placeminStart];
    set[b+1]= memb[placeminStart];
    totallength=totallength+length[placemin][placeminStart];
    printf("length: %ld ",totallength);
    for(d=0;d<14;d++){printf("%c ",set[d]);}
    printf("\n");
    if(totallength<mintotallength){
        mintotallength=totallength;mintour=z;
        for(d=0;d<14;d++){minset[d]=set[d];}
    }
}/*for (z=0; z<13;z++){*/
printf("min length: %ld ",mintotallength);
for(d=0;d<14;d++){printf("%c ",minset[d]);}
printf("\n");
```

```
        return 0;
}
```

**A1:** Listing to Nearest Neigbourh Algorithmus Program in C

```
#include <sys/types.h>
#include <sys/uio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <math.h>
#include <stdio.h>

int main (){
        int a,b,c,d,e,f,g,h,i,j,k,l,z;
        long counter=0;
        char memb[13];
        char set[13];
        char minRoute[13];
        char maxRoute[13];
        int numberminRoute[13];
        int numbermaxRoute[13];
        int numbermemb[13];
        int numberset[13];
        long tourlength;
        long totallength;
        long length[14][14]; //AB,AC,AD,AE...
        long placemin=0;
        long placemax=0;
        long min=1000000;
        long max=0;
        memb[0]='A';
        memb[1]='B';
        memb[2]='C';
        memb[3]='D';
        memb[4]='E';
        memb[5]='F';
        memb[6]='G';
        memb[7]='H';
        memb[8]='I';
        memb[9]='J';
        memb[10]='K';
        memb[11]='L';
        memb[12]='M';
        numbermemb[0]=0;
        numbermemb[1]=1;
        numbermemb[2]=2;
        numbermemb[3]=3;
        numbermemb[4]=4;
```

```
    numbermemb[5]=5;
    numbermemb[6]=6;
    numbermemb[7]=7;
    numbermemb[8]=8;
    numbermemb[9]=9;
    numbermemb[10]=10;
    numbermemb[11]=11;
    numbermemb[12]=12;
    length[0][0]=0;/*AA*/
    length[0][1]=360;/*AB*/
    length[0][2]=1117;/*AC*/
    length[0][3]=4570;/*AD*/
    length[0][4]=834;/*AE*/
    length[0][5]=701;/*AF*/
    length[0][6]=4254;/*AG*/
    length[0][7]=4432;/*AH*/
    length[0][8]=222;/*AI*/
    length[0][9]=2253;/*AJ*/
    length[0][10]=2682;/*AK*/
    length[0][11]=2437;/*AL*/
    length[0][12]=1738;/*AM*/
    length[1][0]=360;/*BA*/
    length[1][1]=0;/*BB*/
    length[1][2]=1262;/*BC*/
    length[1][3]=4650;/*BD*/
    length[1][4]=1022;/*BE*/
    length[1][5]=347;/*BF*/
    length[1][6]=4443;/*G*/
    length[1][7]=4544;/*H*/
    length[1][8]=155;/*I*/
    length[1][9]=2605;/*J*/
    length[1][10]=2845;/*K*/
    length[1][11]=2789;/*L*/
    length[1][12]=2090;/*M*/
    length[2][0]=1117;/*CA*/
    length[2][1]=1262;/*CB*/
    length[2][2]=0    ;/*CC*/
    length[2][3]=3410;/*CD*/
    length[2][4]=474; /*CE*/
    length[2][5]=1574;/*CF*/
    length[2][6]=3022;/*G*/
    length[2][7]=3302;/*H*/
    length[2][8]=1214;/*I*/
    length[2][9]=1715;/*J*/
    length[2][10]=1605;/*K*/
    length[2][11]=1792;/*L*/
    length[2][12]=1482;/*M*/
    length[3][0]=4570;/*DA*/
    length[3][1]=4650;/*DB*/
    length[3][2]=3410;/*DC*/
    length[3][3]=    0;/*DD*/
```

```
length[3][4]=3814;/*DE*/
length[3][5]=4958;/*DF*/
length[3][6]=613;/*G*/
length[3][7]=1291;/*H*/
length[3][8]=4600;/*I*/
length[3][9]=3085;/*J*/
length[3][10]=2005;/*K*/
length[3][11]=2813;/*L*/
length[3][12]=3640;/*M*/
length[4][0]=834;/*EA*/
length[4][1]=1022;/*EB*/
length[4][2]=474;  /*EC*/
length[4][3]=3814;/*ED*/
length[4][4]=0    ;/*EE*/
length[4][5]=1392;/*EF*/
length[4][6]=3480;/*G*/
length[4][7]=3725;/*H*/
length[4][8]=915;/*I*/
length[4][9]=1678;/*J*/
length[4][10]=1907  ;/*K*/
length[4][11]=1806;/*L*/
length[4][12]=1290;/*M*/
length[5][0]=701;/*FA*/
length[5][1]=347;/*FB*/
length[5][2]=1574;/*FC*/
length[5][3]=4958;/*FD*/
length[5][4]=1392;/*FE*/
length[5][5]=0    ;/*FE*/
length[5][6]=4773;/*G*/
length[5][7]=4886;/*H*/
length[5][8]=310;/*I*/
length[5][9]=2957;/*J*/
length[5][10]=3154;/*K*/
length[5][11]=3141;/*L*/
length[5][12]=2442;/*M*/
length[6][0]=4254;/*Ga*/
length[6][1]=4443;/*Gb*/
length[6][2]=3022;/*Gc*/
length[6][3]=613;/*Gd*/
length[6][4]=3480;/*Ge*/
length[6][5]=4773;/*Gf*/
length[6][6]=0;/*Gg*/
length[6][7]=1816;/*Gh*/
length[6][8]=4342;/*Gi*/
length[6][9]=2477;/*Gj*/
length[6][10]=1626;/*Gk*/
length[6][11]=2205;/*Gl*/
length[6][12]=3174;/*Gm*/
length[7][0]=4432;/*Ha*/
length[7][1]=4544;/*Hb*/
length[7][2]=3302;/*Hc*/
```

```
length[7][3]=1291;/*Hd*/
length[7][4]=3725;/*He*/
length[7][5]=4886;/*Hf*/
length[7][6]=1816;/*Hg*/
length[7][7]=0;/*Hh*/
length[7][8]=4514;/*Hi*/
length[7][9]=3710;/*Hj*/
length[7][10]=2093;/*Hk*/
length[7][11]=3392;/*Hl*/
length[7][12]=4138;/*Hm*/
length[8][0]=222;/*Ia*/
length[8][1]=155;/*Ib*/
length[8][2]=1214;/*Ic*/
length[8][3]=4600;/*Id*/
length[8][4]=915;/*Ie*/
length[8][5]=310;/*If*/
length[8][6]=4342;/*Ig*/
length[8][7]=4514;/*Ih*/
length[8][8]=0;/*Ii*/
length[8][9]=2475;/*Ij*/
length[8][10]=2765;/*Ik*/
length[8][11]=2659;/*Il*/
length[8][12]=1960;/*Im*/
length[9][0]=2253;/*Ja*/
length[9][1]=2605;/*Jb*/
length[9][2]=1715;/*Jc*/
length[9][3]=3085;/*Jd*/
length[9][4]=1678;/*Je*/
length[9][5]=2957;/*Jf*/
length[9][6]=2477;/*Jg*/
length[9][7]=3710;/*Jh*/
length[9][8]=2475;/*Ji*/
length[9][9]=0;/*Jj*/
length[9][10]=1646;/*Jk*/
length[9][11]=259;/*Jl*/
length[9][12]=557;/*Jm*/
length[10][0]=2682;/*Ka*/
length[10][1]=2845;/*Kb*/
length[10][2]=1605;/*Kc*/
length[10][3]=2005;/*Kd*/
length[10][4]=1907;/*Ke*/
length[10][5]=3154;/*Kf*/
length[10][6]=1626;/*Kg*/
length[10][7]=2093;/*Kh*/
length[10][8]=2765;/*Ki*/
length[10][9]=1646;/*Kj*/
length[10][10]=0;/*Kk*/
length[10][11]=1466;/*Kl*/
length[10][12]=2077;/*Km*/
length[11][0]=2437;/*La*/
length[11][1]=2789;/*Lb*/
```

```
length[11][2]=1792;/*Lc*/
length[11][3]=2813;/*Ld*/
length[11][4]=1806;/*Le*/
length[11][5]=3141;/*Lf*/
length[11][6]=2205;/*Lg*/
length[11][7]=3392;/*Lh*/
length[11][8]=2659;/*Li*/
length[11][9]=259;/*Lj*/
length[11][10]=1466;/*Lk*/
length[11][11]=0;/*Ll*/
length[11][12]=818;/*Lm*/
length[12][0]=1738;/*Ma*/
length[12][1]=2090;/*Mb*/
length[12][2]=1482;/*Mc*/
length[12][3]=3640;/*Md*/
length[12][4]=1290;/*Me*/
length[12][5]=2442;/*Mf*/
length[12][6]=3174;/*Mg*/
length[12][7]=4138;/*Mh*/
length[12][8]=1960;/*Mi*/
length[12][9]=557;/*Mj*/
length[12][10]=2077;/*Mk*/
length[12][11]=818;/*Ml*/
length[12][12]=0;/*Mm*/
set[0]='A';
numberset[0]=numbermemb[0];
for (a=1; a<13;a++){/*for (a=1; a<6;a++){*/
set[1]=memb[a];
numberset[1]=numbermemb[a];
for (b=1; b<13;b++){/*for (b=1; b<6;b++){*/
if(memb[b]!=set[1]){/*if(memb[b]!=set[1]){*/
set[2]=memb[b];
numberset[2]=numbermemb[b];
for (c=1; c<13;c++){/*for (c=1; c<6;c++){*/
if(memb[c]!=set[1]){/*if(memb[c]!=set[c]){*/
if(memb[c]!=set[2]){/*if(memb[c]!=set[c]){*/
set[3]=memb[c];
numberset[3]=numbermemb[c];
for (d=1; d<13;d++){/*for (d=1; d<6;d++){*/
if(memb[d]!=set[1]){/*if(memb[d]!=set[1]){*/
if(memb[d]!=set[2]){/*if(memb[d]!=set[2]){*/
if(memb[d]!=set[3]){/*if(memb[d]!=set[3]){*/
set[4]=memb[d];
numberset[4]=numbermemb[d];
for (e=1; e<13;e++){/*for (e=1; e<6;e++){*/
if(memb[e]!=set[1]){/*if(memb[e]!=set[1]){*/
if(memb[e]!=set[2]){/*if(memb[e]!=set[1]){*/
if(memb[e]!=set[3]){/*if(memb[e]!=set[1]){*/
if(memb[e]!=set[4]){/*if(memb[e]!=set[1]){*/
set[5]=memb[e];
numberset[5]=numbermemb[e];
```

```
for (f=1; f<13;f++){
if(memb[f]!=set[1]){
if(memb[f]!=set[2]){
if(memb[f]!=set[3]){
if(memb[f]!=set[4]){
if(memb[f]!=set[5]){
set[6]=memb[f];
numberset[6]=numbermemb[f];
for (g=1; g<13;g++){
if(memb[g]!=set[1]){
if(memb[g]!=set[2]){
if(memb[g]!=set[3]){
if(memb[g]!=set[4]){
if(memb[g]!=set[5]){
if(memb[g]!=set[6]){
set[7]=memb[g];
numberset[7]=numbermemb[g];
for (h=1; h<13;h++){
if(memb[h]!=set[1]){
if(memb[h]!=set[2]){
if(memb[h]!=set[3]){
if(memb[h]!=set[4]){
if(memb[h]!=set[5]){
if(memb[h]!=set[6]){
if(memb[h]!=set[7]){
set[8]=memb[h];
numberset[8]=numbermemb[h];
for (i=1; i<13;i++){
if(memb[i]!=set[1]){
if(memb[i]!=set[2]){
if(memb[i]!=set[3]){
if(memb[i]!=set[4]){
if(memb[i]!=set[5]){
if(memb[i]!=set[6]){
if(memb[i]!=set[7]){
if(memb[i]!=set[8]){
set[9]=memb[i];
numberset[9]=numbermemb[i];
for (j=1; j<13;j++){
if(memb[j]!=set[1]){
if(memb[j]!=set[2]){
if(memb[j]!=set[3]){
if(memb[j]!=set[4]){
if(memb[j]!=set[5]){
if(memb[j]!=set[6]){
if(memb[j]!=set[7]){
if(memb[j]!=set[8]){
if(memb[j]!=set[9]){
set[10]=memb[j];
numberset[10]=numbermemb[j];
for (k=1; k<13;k++){
```

```
if(memb[k]!=set[1]){
if(memb[k]!=set[2]){
if(memb[k]!=set[3]){
if(memb[k]!=set[4]){
if(memb[k]!=set[5]){
if(memb[k]!=set[6]){
if(memb[k]!=set[7]){
if(memb[k]!=set[8]){
if(memb[k]!=set[9]){
if(memb[k]!=set[10]){
set[11]=memb[k];
numberset[11]=numbermemb[k];
for (l=1; l<13;l++){
if(memb[l]!=set[1]){
if(memb[l]!=set[2]){
if(memb[l]!=set[3]){
if(memb[l]!=set[4]){
if(memb[l]!=set[5]){
if(memb[l]!=set[6]){
if(memb[l]!=set[7]){
if(memb[l]!=set[8]){
if(memb[l]!=set[9]){
if(memb[l]!=set[10]){
if(memb[l]!=set[11]){
set[12]=memb[l];
numberset[12]=numbermemb[l];
counter++;
tourlength=0;
totallength=0;
tourlength=length[numberset[0]][numberset[1]];
totallength=tourlength;
tourlength=length[numberset[1]][numberset[2]];
totallength=totallength+tourlength;
tourlength=length[numberset[2]][numberset[3]];
totallength=totallength+tourlength;
tourlength=length[numberset[3]][numberset[4]];
totallength=totallength+tourlength;
tourlength=length[numberset[4]][numberset[5]];
totallength=totallength+tourlength;
tourlength=length[numberset[5]][numberset[6]];
totallength=totallength+tourlength;
tourlength=length[numberset[6]][numberset[7]];
totallength=totallength+tourlength;
tourlength=length[numberset[7]][numberset[8]];
totallength=totallength+tourlength;
tourlength=length[numberset[8]][numberset[9]];
totallength=totallength+tourlength;
tourlength=length[numberset[9]][numberset[10]];
totallength=totallength+tourlength;
tourlength=length[numberset[10]][numberset[11]];
totallength=totallength+tourlength;
```

```
    tourlength=length[numberset[11]][numberset[12]];
    totallength=totallength+tourlength;
    tourlength=length[numberset[12]][numberset[0]];
    totallength=totallength+tourlength;
    if(totallength<min){min=totallength;placemin = counter-1;
        for (z=0; z<13;z++){minRoute[z] = set[z];numberminRoute[z] =
            numberset[z];}
    }
    if(max<totallength){max=totallength;placemax = counter-1;
        for (z=0; z<13;z++){maxRoute[z] = set[z];numbermaxRoute[z] =
            numberset[z];}
    }
    }}}}}}}}}}
    }/*for (l=1; l<6;l++)*/
    }}}}}}}}}
    }/*for (k=1; k<6;k++)*/
    }}}}}}}}
    }/*for (j=1; j<6;i++)*/
    }}}}}}}
    }/*for (i=1; i<6;i++)*/
    }}}}}}
    }/*for (h=1; h<6;h++)*/
    }}}}}
    }/*for (g=1; g<6;g++)*/
    }}}}
    }/*for (f=1; f<6;f++)*/
    }
    }
    }
    }
    }/*for (e=1; e<6;e++){*/
    }/*if(memb[d]!=set[3]){*/
    }/*if(memb[d]!=set[2]){*/
    }/*if(memb[d]!=set[c]){*/
    }/*for (d=1; d<6;d++){*/
    }/*if(memb[c]!=set[c]){*/
    }/*if(memb[c]!=set[c]){*/
    }/*for (c=1; c<6;c++){*/
    }/*if(memb[b]!=set[1]){*/
    }/*for (b=1; b<6;b++){*/
    }/*for (a=1; a<6;a++){*/
    printf("Number of all routes: %ld ",counter);
    printf("\n");
    printf("minimum : %ld km route: ",min);
    for (z=0; z<13;z++){printf("%c ",minRoute[z]);}
    printf("found at: %ld ",placemin);
    printf("\n");
    printf("minimum : %ld km route: ",max);
    for (z=0; z<13;z++){printf("%c ",maxRoute[z]);}
    printf("found at: %ld ",placemax);
    printf("\n");
```

```c

    return 0;

}
```

01TSMbrute.c

**A2:** Listing to Brute Force Program in C

```c
#include <sys/types.h>
#include <sys/uio.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <unistd.h>
#include <math.h>
#include <stdio.h>
#include <time.h>          /* time */
//path cd /Users/dragonhead/Documents/eec_prog
//compile gcc -Wall -o 01TSMant01 01TSMant01.c
int main (){
    int num_of_new_ants_starts=8;/*number how often ants are placed at 13 towns after pheromone 0.01 and random*/
    int town=13;
    double length[town][town];
    int ant=13;
    int a,b,j,stop,r,ant_no_move,t,num_of_towns_ant_has_vis;
    int c,go_to,e,best_ant,it,from,to,best_gamma_run, best_it;
    int best_best_ant;
    int rl,counter_rl;
    int random[ant];
    int tabu_best[town];
    int tabu_ant[num_of_new_ants_starts][ant][town];
    double transition_probability_new[town][town];
    double tau_t_ij_old[town][town];
    double tau_t_ij_new[town][town];
    double delta_tau_ij_k_ant;
    double evap=0.7;
    double alpha,beta;
    double visibility_ij_pow_beta[town][town];
    double nominator_new[town][town];
    double denominator_new;
    double visibility_ij;
    int not_visited_list[ant][town];
    double transition_probability_max,tour_lenth[ant];
    double tour_lenth_min[num_of_new_ants_starts];
    int alpha_run,beta_run,gamma_run;
    int alpha_run_max=1;
    int beta_run_max=1;
    int gamma_run_max=75;/*number of new start with new random and new 0.01 pheromone*/
```

```
double lenth_min_const_alpha_beta[alpha_run_max][beta_run_max];
double q=10000;
double sum_of_tour_lenth_min,sum_lenth_min_const_alpha_beta[alpha_run_max][bet
int output=0;
int divisor_on=1;
int min_length_at_END;
double minlgthofallgams_conalbelgth[alpha_run_max][beta_run_max];
int tabu_list_of_best_ant_of_const_alpha_beta[alpha_run_max][beta_run_max]
time_t tim;
length[0][0]=0;/*AA*/
length[0][1]=360;/*AB*/
length[0][2]=1117;/*AC*/
length[0][3]=4570;/*AD*/
length[0][4]=834;/*AE*/
length[0][5]=701;/*AF*/
length[0][6]=4254;/*AG*/
length[0][7]=4432;/*AH*/
length[0][8]=222;/*AI*/
length[0][9]=2253;/*AJ*/
length[0][10]=2682;/*AK*/
length[0][11]=2437;/*AL*/
length[0][12]=1738;/*AM*/
length[1][0]=360;/*BA*/
length[1][1]=0;/*BB*/
length[1][2]=1262;/*BC*/
length[1][3]=4650;/*BD*/
length[1][4]=1022;/*BE*/
length[1][5]=347;/*BF*/
length[1][6]=4443;/*G*/
length[1][7]=4544;/*H*/
length[1][8]=155;/*I*/
length[1][9]=2605;/*J*/
length[1][10]=2845;/*K*/
length[1][11]=2789;/*L*/
length[1][12]=2090;/*M*/
length[2][0]=1117;/*CA*/
length[2][1]=1262;/*CB*/
length[2][2]=0    ;/*CC*/
length[2][3]=3410;/*CD*/
length[2][4]=474; /*CE*/
length[2][5]=1574;/*CF*/
length[2][6]=3022;/*G*/
length[2][7]=3302;/*H*/
length[2][8]=1214;/*I*/
length[2][9]=1715;/*J*/
length[2][10]=1605;/*K*/
length[2][11]=1792;/*L*/
length[2][12]=1482;/*M*/
length[3][0]=4570;/*DA*/
```

```
length[3][1]=4650;/*DB*/
length[3][2]=3410;/*DC*/
length[3][3]=    0;/*DD*/
length[3][4]=3814;/*DE*/
length[3][5]=4958;/*DF*/
length[3][6]=613;/*G*/
length[3][7]=1291;/*H*/
length[3][8]=4600;/*I*/
length[3][9]=3085;/*J*/
length[3][10]=2005;/*K*/
length[3][11]=2813;/*L*/
length[3][12]=3640;/*M*/
length[4][0]=834;/*EA*/
length[4][1]=1022;/*EB*/
length[4][2]=474;  /*EC*/
length[4][3]=3814;/*ED*/
length[4][4]=0     ;/*EE*/
length[4][5]=1392;/*EF*/
length[4][6]=3480;/*G*/
length[4][7]=3725;/*H*/
length[4][8]=915;/*I*/
length[4][9]=1678;/*J*/
length[4][10]=1907  ;/*K*/
length[4][11]=1806;/*L*/
length[4][12]=1290;/*M*/
length[5][0]=701;/*FA*/
length[5][1]=347;/*FB*/
length[5][2]=1574;/*FC*/
length[5][3]=4958;/*FD*/
length[5][4]=1392;/*FE*/
length[5][5]=0     ;/*FE*/
length[5][6]=4773;/*G*/
length[5][7]=4886;/*H*/
length[5][8]=310;/*I*/
length[5][9]=2957;/*J*/
length[5][10]=3154;/*K*/
length[5][11]=3141;/*L*/
length[5][12]=2442;/*M*/
length[6][0]=4254;/*Ga*/
length[6][1]=4443;/*Gb*/
length[6][2]=3022;/*Gc*/
length[6][3]=613;/*Gd*/
length[6][4]=3480;/*Ge*/
length[6][5]=4773;/*Gf*/
length[6][6]=0;/*Gg*/
length[6][7]=1816;/*Gh*/
length[6][8]=4342;/*Gi*/
length[6][9]=2477;/*Gj*/
length[6][10]=1626;/*Gk*/
length[6][11]=2205;/*Gl*/
length[6][12]=3174;/*Gm*/
```

```
length[7][0]=4432;/*Ha*/
length[7][1]=4544;/*Hb*/
length[7][2]=3302;/*Hc*/
length[7][3]=1291;/*Hd*/
length[7][4]=3725;/*He*/
length[7][5]=4886;/*Hf*/
length[7][6]=1816;/*Hg*/
length[7][7]=0;/*Hh*/
length[7][8]=4514;/*Hi*/
length[7][9]=3710;/*Hj*/
length[7][10]=2093;/*Hk*/
length[7][11]=3392;/*Hl*/
length[7][12]=4138;/*Hm*/
length[8][0]=222;/*Ia*/
length[8][1]=155;/*Ib*/
length[8][2]=1214;/*Ic*/
length[8][3]=4600;/*Id*/
length[8][4]=915;/*Ie*/
length[8][5]=310;/*If*/
length[8][6]=4342;/*Ig*/
length[8][7]=4514;/*Ih*/
length[8][8]=0;/*Ii*/
length[8][9]=2475;/*Ij*/
length[8][10]=2765;/*Ik*/
length[8][11]=2659;/*Il*/
length[8][12]=1960;/*Im*/
length[9][0]=2253;/*Ja*/
length[9][1]=2605;/*Jb*/
length[9][2]=1715;/*Jc*/
length[9][3]=3085;/*Jd*/
length[9][4]=1678;/*Je*/
length[9][5]=2957;/*Jf*/
length[9][6]=2477;/*Jg*/
length[9][7]=3710;/*Jh*/
length[9][8]=2475;/*Ji*/
length[9][9]=0;/*Jj*/
length[9][10]=1646;/*Jk*/
length[9][11]=259;/*Jl*/
length[9][12]=557;/*Jm*/
length[10][0]=2682;/*Ka*/
length[10][1]=2845;/*Kb*/
length[10][2]=1605;/*Kc*/
length[10][3]=2005;/*Kd*/
length[10][4]=1907;/*Ke*/
length[10][5]=3154;/*Kf*/
length[10][6]=1626;/*Kg*/
length[10][7]=2093;/*Kh*/
length[10][8]=2765;/*Ki*/
length[10][9]=1646;/*Kj*/
length[10][10]=0;/*Kk*/
length[10][11]=1466;/*Kl*/
```

```
    length[10][12]=2077;/*Km*/
    length[11][0]=2437;/*La*/
    length[11][1]=2789;/*Lb*/
    length[11][2]=1792;/*Lc*/
    length[11][3]=2813;/*Ld*/
    length[11][4]=1806;/*Le*/
    length[11][5]=3141;/*Lf*/
    length[11][6]=2205;/*Lg*/
    length[11][7]=3392;/*Lh*/
    length[11][8]=2659;/*Li*/
    length[11][9]=259;/*Lj*/
    length[11][10]=1466;/*Lk*/
    length[11][11]=0;/*Ll*/
    length[11][12]=818;/*Lm*/

    length[12][0]=1738;/*Ma*/
    length[12][1]=2090;/*Mb*/
    length[12][2]=1482;/*Mc*/
    length[12][3]=3640;/*Md*/
    length[12][4]=1290;/*Me*/
    length[12][5]=2442;/*Mf*/
    length[12][6]=3174;/*Mg*/
    length[12][7]=4138;/*Mh*/
    length[12][8]=1960;/*Mi*/
    length[12][9]=557;/*Mj*/
    length[12][10]=2077;/*Mk*/
    length[12][11]=818;/*Ml*/
    length[12][12]=0;/*Mm*/
    srand((unsigned) time(&tim));
    printf("Divisor_on: %2d \n", divisor_on);
    printf("towns:%2d ants:%2d Q1:%8.1f num of ants' starts:%2d number
       of new starts %2d \n",town,ant,
       q,num_of_new_ants_starts,gamma_run_max);
    beta=0.9;
    for (beta_run=0; beta_run<beta_run_max; beta_run++) {/*b run*/
        beta=beta+0.1;
        for (a=0; a<town; a++) {/*for (t=0; t<town; t++) {*/
            for (b=0; b<town; b++) {
                if(a!=b){
                visibility_ij=1/length[a][b];
                visibility_ij_pow_beta[a][b]=pow(visibility_ij,beta);
                }
                else{
                    visibility_ij_pow_beta[a][b]=0;
                }
            }
        }/*for (t=0; t<town; t++) {*/
        alpha=0.9;
        for (alpha_run=0; alpha_run<alpha_run_max; alpha_run++) {/*a
           run*/
            alpha=alpha+0.1;
```

```
            printf("evapo: %4.2f alpha: %4.2f beta : %4.2f \n",evap,alpha,beta);
            minlgthofallgams_conalbelgth[alpha_run][beta_run]=1000000;
            best_gamma_run=0;
            best_best_ant=100;
            sum_lenth_min_const_alpha_beta[alpha_run][beta_run]=0;;
for (gamma_run=0; gamma_run<gamma_run_max; gamma_run++) {/*for (gamma_run=0; gamma_run<numb_of_new_exp; gamma_run++) {*/
    sum_of_tour_lenth_min=0;
            //init for cost alpha and beta
            lenth_min_const_alpha_beta[alpha_run][beta_run]=1000000;
            for (a=0; a<town; a++) {/*for (t=0; t<town; t++) {*/
                for (b=0; b<town; b++) {
                    tau_t_ij_new[a][b]=0.01;/*not needed*/
                }
            }/*for (t=0; t<town; t++) {*/
            it=0;
            num_of_towns_ant_has_vis=0;
            if(output==1){printf("ncycle%2d: \n",num_of_towns_ant_has_vis);}
            //place ant on town and first next town is random
            for (a=0; a<ant; a++) {/*for (a=0; a<ant; a++) {*/
                tabu_ant[it][a][num_of_towns_ant_has_vis]=a;/*ant a at town t */
                for (j=1; j<town; j++) {tabu_ant[it][a][j]=a;}//tabulist filled with initial town
            }/*for (i=0; i<ant; i++) {*/
            //ant1 is at town1,ant2 is at town2......ant1 is at town1,ant2 is at town2......ant1 is at town1,ant2 is at town2......ant1 is at town1,ant2 is at town2......

            ant_no_move=0;
            while(ant_no_move==0){/*while(ant_no_move==0){*/
                ant_no_move=1;
                stop=0;
                while (stop==0) {/*while (stop==0)*/
                    for (r=0; r<town; r++) {/*or (a=0; a<13; a++) {*/
                        rl=rand()%town;
                        random[r]=rl;

                    }/*or (a=0; a<13; a++) {*/
                    stop=1;
                    for (a=0; a<town; a++) {/*or (a=0; a<13; a++) {*/
                        counter_rl=0;
                        for (b=0; b<town; b++) {/*for (b=0; b<13; b++) {*/
                            if(random[a]==random[b]){counter_rl++;}
                        }/*for (b=0; b<13; b++) {*/
                        if(2<=counter_rl){/*2<=counter_rl no number is double*/
                            stop=0;
```

```
                }/*2<=counter_rl no number is double*/
            }/*or (a=0; a<town; a++) {*/
        }/*while (stop==0)*/

        for (a=0; a<town; a++) {/*or (a=0; a<13; a++) {*/
            if(tabu_ant[it][a][0]==random[a]){
                ant_no_move=0;
            }
            else{tabu_ant[it][a][1]=random[a];}
        }/*or (a=0; a<13; a++) {*/
    }/*while(ant_no_move==0){*/
//ant1 is at town random1.....ant2 is at town random2.....ant3 is at town random3.....

    num_of_towns_ant_has_vis=1;
    if(output==1){printf("mcycle%2d:\n",num_of_towns_ant_has_vis);}
    //transition_probability adjust

    //evaporation for all town
    for (a=0; a<town; a++) {
        for (b=0; b<town; b++){
            tau_t_ij_old[a][b]=tau_t_ij_new[a][b];
            tau_t_ij_new[a][b]=evap*tau_t_ij_old[a][b];
        }
    }
    for (a=0; a<ant; a++) {/*or (a=0; a<13; a++) {*/
        from=tabu_ant[it][a][num_of_towns_ant_has_vis-1];
        to=  tabu_ant[it][a][num_of_towns_ant_has_vis];
        delta_tau_ij_k_ant = q/length[from][to];
        tau_t_ij_new[from][to]=tau_t_ij_new[from][to]+delta_tau_ij_k_
    }/*or (a=0; a<13; a++) {*/
    denominator_new=0;
    for (a=0; a<town; a++) {/*for (t=0; t<town; t++) {*/
        for (b=0; b<town; b++) {
                nominator_new[a][b]=pow(tau_t_ij_new[a][b],alpha)*visi
                denominator_new=denominator_new+nominator_new[a][b];
        }

    }
    for (a=0; a<town; a++) {/*for (t=0; t<town; t++) {*/
        for (b=0; b<town; b++) {
            if(divisor_on==1){transition_probability_new[a][b]=nomina
            else{transition_probability_new[a][b]=nominator_new[a][b]

        }

    }
    for (num_of_towns_ant_has_vis=2; num_of_towns_ant_has_vis<town; num_of_towns_ant_has_vis++){/*for
```

```
(num_of_towns_ant_has_vis=2;*/
        for (a=0; a<ant; a++){/*for (a=0; a<ant; a++){*/
            for (c=0; c<town; c++){not_visited_list[a][c]=c;}
            for (b=0; b<num_of_towns_ant_has_vis; b++){/*for (b=0; b<town; b++){*/
                for (t=0; t<town-b; t++){/*or (t=0; t<town; t++){*/
                    if(not_visited_list[a][t]==tabu_ant[it][a][b]){/*
                        not_visited_list[a][t]=100;
                        for (e=t; e<town-1; e++){not_visited_list[a][e]=not_visited_li
                    }/*if(t==tabu_ant[it][a][b]){*/
                }/*or (t=0; t<town; t++){*/
            }/*for (b=0; b<town; b++){*/
        }/*for (a=0; a<ant; a++){*/
        for (a=0; a<ant; a++){/*for (a=0; a<ant; a++){*/
            transition_probability_max=0;go_to=100;
            for (t=0; t<town-num_of_towns_ant_has_vis; t++){/*for (t=0; t<town-b-1; t++){*/
                if(transition_probability_max<transition_probability_r
                    not_visited_list[a][t] ]){
                    transition_probability_max=transition_probability_
                        not_visited_list[a][t] ];
                    //write_to_cons_sh(transition_probability[tabu_an
                        not_visited_list[a][t] ]);
                    go_to = not_visited_list[a][t] ;
                }

            }/*for (t=0; t<town-b-1; t++){*/
            if(go_to==100){
                //happens in pairs when antx goes to y and anty goes to x
                transition_probability_max=0;go_to=100;
                rl=rand()%(town-num_of_towns_ant_has_vis);

                go_to=not_visited_list[a][rl];

            }
            tabu_ant[it][a][num_of_towns_ant_has_vis]=go_to;

        }/*for (a=0; a<ant; a++){*/




        for (a=0; a<town; a++) {
            for (b=0; b<town; b++){
                tau_t_ij_old[a][b]=tau_t_ij_new[a][b];
                tau_t_ij_new[a][b]=evap*tau_t_ij_old[a][b];
            }
        }
```

```
            for (a=0; a<ant; a++) {/*or (a=0; a<13; a++) {*/
                from=tabu_ant[it][a][num_of_towns_ant_has_vis-1];
                to=  tabu_ant[it][a][num_of_towns_ant_has_vis];
                delta_tau_ij_k_ant = q/length[from][to];
                tau_t_ij_new[from][to]=tau_t_ij_new[from][to]+delta_tau_ij
            }/*or (a=0; a<13; a++) {*/
            denominator_new=0;
            for (a=0; a<town; a++) {/*for (t=0; t<town; t++) {*/
                for (b=0; b<town; b++) {
                    nominator_new[a][b]=pow(tau_t_ij_new[a][b],alpha)*visi
                    denominator_new=denominator_new+nominator_new[a][b];
                }

            }
            for (a=0; a<town; a++) {/*for (t=0; t<town; t++) {*/
                for (b=0; b<town; b++) {
                    if(divisor_on==1){transition_probability_new[a][b]=no
                    else{transition_probability_new[a][b]=nominator_new[a
                }

            }
        }/*for (num_of_towns_ant_has_vis=2; num_of_towns_ant_has_vis<5; num_of_towns_ant_has_vis++){*/
        if(output==1){printf("ants reset %2d times: \n",it);}
        tour_lenth_min[it]=1000000;
        for (a=0; a<ant; a++) {/*or (a=0; a<13; a++) {*/

            tour_lenth[a]=0;
            for (b=0; b<town-1; b++) {
                tour_lenth[a]=tour_lenth[a]+length[tabu_ant[it][a][b]][tal
            }
            tour_lenth[a]=tour_lenth[a]+length[tabu_ant[it][a][b]][tabu_a

            if(tour_lenth[a]<tour_lenth_min[it]){
                tour_lenth_min[it]=tour_lenth[a]; best_ant=a;
            }
        }/*or (a=0; a<13; a++) {*/
        if(output==1){printf("best ant%2d: ",best_ant);}
            for (b=0; b<town; b++) {
                tabu_best[b]=tabu_ant[it][best_ant][b];
                if(output==1){printf("%2d ",tabu_best[b]);}
            }
        if(output==1){printf("length it: %8.1f \n",tour_lenth_min[it]);}
        if(tour_lenth_min[it]<=lenth_min_const_alpha_beta[alpha_run][beta_
        sum_of_tour_lenth_min=sum_of_tour_lenth_min+tour_lenth_min[it];
        best_it=0;
        for (it=1; it<num_of_new_ants_starts; it++){/*for (it=0; it<num_of_new_ants_starts; it++)*/
            min_length_at_END=0;
            num_of_towns_ant_has_vis=0;
```

```
            for (a=0; a<ant; a++) {/*for (a=0; a<ant; a++) {*/
                for (j=0; j<town; j++)
                    {tabu_ant[it][a][j]=a;}//tabulist filled with
                    initial town
            }/*for (i=0; i<ant; i++) {*/

            for (num_of_towns_ant_has_vis=1;
               num_of_towns_ant_has_vis<13;
               num_of_towns_ant_has_vis++){/*for
               (num_of_towns_ant_has_vis=2; */
                for (a=0; a<ant; a++){/*for (a=0; a<ant; a++){*/
                    for (c=0; c<town; c++){not_visited_list[a][c]=c;}
                    for (b=0; b<num_of_towns_ant_has_vis; b++){/*for
                       (b=0; b<town; b++){*/
                        for (t=0; t<town-b; t++){/*or (t=0; t<town;
                           t++){*/
                            if(not_visited_list[a][t]==tabu_ant[it][a][b]
                                not_visited_list[a][t]=100;
                                for (e=t; e<town-1;
                                   e++){not_visited_list[a][e]=not_visite
                            }/*if(t==tabu_ant[it][a][b]){*/
                        }/*or (t=0; t<town; t++){*/
                    }/*for (b=0; b<town; b++){*/
                }/*for (a=0; a<ant; a++){*/


                for (a=0; a<ant; a++){/*for (a=0; a<ant; a++){*/
                    transition_probability_max=0;go_to=100;
                    for (t=0; t<town-num_of_towns_ant_has_vis;
                       t++){/*for (t=0; t<town-b-1; t++){*/
                        if(transition_probability_max<transition_probabili
                           not_visited_list[a][t]  ]){
                            transition_probability_max=transition_probabil
                                not_visited_list[a][t]  ];
                            //write_to_cons_sh(transition_probability[tab
                                not_visited_list[a][t]  ]);
                            go_to = not_visited_list[a][t]  ;
                        }

                    }/*for (t=0; t<town-b-1; t++){*/
                    if(go_to==100){
                        //happens in pairs when antx goes to y and
                           anty goes to x
                        printf("second prob");
                        transition_probability_max=0;go_to=100;
                        rl=rand()%(town-num_of_towns_ant_has_vis);

                        go_to=not_visited_list[a][rl];

                    }
                    tabu_ant[it][a][num_of_towns_ant_has_vis]=go_to;
```

```

                }/*for (a=0; a<ant; a++){*/
                for (a=0; a<town; a++) {
                    for (b=0; b<town; b++){
                        tau_t_ij_old[a][b]=tau_t_ij_new[a][b];
                        tau_t_ij_new[a][b]=evap*tau_t_ij_old[a][b];
                    }
                }
                for (a=0; a<ant; a++) {/*or (a=0; a<13; a++) {*/
                    from=tabu_ant[it][a][num_of_towns_ant_has_vis-1];
                    to=  tabu_ant[it][a][num_of_towns_ant_has_vis];
                    delta_tau_ij_k_ant = q/length[from][to];
                    tau_t_ij_new[from][to]=tau_t_ij_new[from][to]+delta_ta
                }/*or (a=0; a<13; a++) {*/
                denominator_new=0;
                for (a=0; a<town; a++) {/*for (t=0; t<town; t++) {*/
                    for (b=0; b<town; b++) {
                        nominator_new[a][b]=pow(tau_t_ij_new[a][b],alpha)
                        denominator_new=denominator_new+nominator_new[a][
                    }

                }
                for (a=0; a<town; a++) {/*for (t=0; t<town; t++) {*/
                    for (b=0; b<town; b++) {
                        if(divisor_on==1){transition_probability_new[a][b
                        else{transition_probability_new[a][b]=nominator_ne
                    }
                }
            }/*for (num_of_towns_ant_has_vis=1;
              num_of_towns_ant_has_vis<5;
              num_of_towns_ant_has_vis++){*/
            if(output==1){printf("ants reset %2d times: \n",it);}
            tour_lenth_min[it]=1000000;
            for (a=0; a<ant; a++) {/*or (a=0; a<13; a++) {*/
                tour_lenth[a]=0;
                for (b=0; b<town-1; b++) {
                    tour_lenth[a]=tour_lenth[a]+length[tabu_ant[it][a][b]
                }
                tour_lenth[a]=tour_lenth[a]+length[tabu_ant[it][a][b]][tal
                if(tour_lenth[a]<tour_lenth_min[it]){
                    tour_lenth_min[it]=tour_lenth[a]; best_ant=a;
                }
            }/*or (a=0; a<13; a++) {*/
output=0;
            if(output==1){printf("best ant%2d: ",best_ant);}

            for (b=0; b<town; b++) {/*for (b=0; b<town; b++) {*/
                tabu_best[b]=tabu_ant[it][best_ant][b];
                if(output==1){printf("%2d ",tabu_best[b]);}
            }/*for (b=0; b<town; b++) {*/
            if(output==1){printf("length it %2d: %8.1f
```

```
\n",it ,tour_lenth_min[it]);}
                 output=0;
                tour_lenth[a]=0;
                if(output==1){printf(" length: %8.1f \n",tour_lenth[a]);}
                for (b=0; b<town-1; b++) {
                    tour_lenth[a]=tour_lenth[a]+length[tabu_ant[it][best_ant]
                    if(output==1){printf(" length: %8.1f
                      \n",tour_lenth[a]);}
                }
                tour_lenth[a]=tour_lenth[a]+length[tabu_ant[it][best_ant][b]]
                if(output==1){printf(" length: %8.1f \n",tour_lenth[a]);}
                sum_of_tour_lenth_min=sum_of_tour_lenth_min+tour_lenth_min[it
                if(tour_lenth_min[it]<=lenth_min_const_alpha_beta[alpha_run][
                    lenth_min_const_alpha_beta[alpha_run][beta_run]=tour_lenth
                    best_it=it;
                    best_best_ant=best_ant;

                }
            }/*for (it=0; it<num_of_new_ants_starts; it++)*/

output=1;
    if(output==1){/*if(output==1){*/
            printf("g run: %2d it: %2d ant: %2d lenth: %8.1f
              ",gamma_run,best_it ,best_best_ant ,lenth_min_const_alpha_beta[a
            for (b=0; b<town; b++) {
                //printf("%2d ",tabu_best[b]);
            printf("%c ",tabu_best[b]+65);

            }
            printf("\n");
    }/*if(output==1){*/
output=0;
    if(lenth_min_const_alpha_beta[alpha_run][beta_run]<minlgthofallgams_conal
        minlgthofallgams_conalbelgth[alpha_run][beta_run]=lenth_min_const_alp
        for (b=0; b<town; b++)
          {tabu_list_of_best_ant_of_const_alpha_beta[alpha_run][beta_run][b]=
        best_gamma_run=gamma_run;
    }
    sum_lenth_min_const_alpha_beta[alpha_run][beta_run]=sum_lenth_min_const_a
            //loop ant restart

            //write to disc for every alpha beta
//              printf("alpha: %6.2f",alpha);printf("beta:
    %6.2f",beta);printf("beta: %6.2f
    \n",lenth_min_const_alpha_beta[alpha_run][beta_run]);
//              fds = open ("res02.txt",O_APPEND |O_CREAT | O_WRONLY,0644);
//              if(fds > 0){/*if(fd > 0){if(fd > 0){if(fd > 0){*/
//
    res=alpha*1000;write_to_disc(res,fds);res=beta*1000;write_to_disc(res,fds)
//                  for (b=0; b<town; b++)
    {write_to_disc_sh(tabu_best[b],fds);}
```

```
//                      write(fds,"\n",2);
//                  }/*if(fd > 0){if(fd > 0){if(fd > 0){*/
//                  close(fds);
                 //write to disc for every alpha beta

}/*for (gamma_run=0; gamma_run<numb_of_new_exp; gamma_run++) {*/

//printf("minlgthofallgams_conalbelgth: %8.1f ",minlgthofallgams_conalbelgth[alpha_run][beta_run]);
//                  printf("sum of all min length: %12.1f ",sum_of_tour_lenth_min);


                 printf("min length: %8.1f at g cycle %d ",minlgthofallgams_conalbelgth[alpha_run][beta_run],best_gamma_
                 printf("list: ");
//for (b=0; b<town; b++) {printf("%2d ",tabu_list_of_best_ant_of_const_alpha_beta[alpha_run][beta_run][b]);}
//                  printf("or ");


//   letter=tabu_list_of_best_ant_of_const_alpha_beta[alpha_run][beta_run][b]+6
                 for (b=0; b<town; b++) {printf("%c ",tabu_list_of_best_ant_of_const_alpha_beta[alpha_run][beta_rur
                         //printf("%c ",tabu_list_of_best_ant_of_const_alpha_beta[alpha_run]

                                   printf("\n");
                 printf("sum: %10.1f \n",sum_lenth_min_const_alpha_beta[alpha_run][beta_run]);

             }/*a run*/
     }/*b run*/

     return 0;
}
```

01TSMant02.c

**A3:** Listing to new ant Program in C

1.0 Initialize:
1.1 Set $t := 0$, $\rho := 0.7$, $\alpha := 1.0$, $\beta := 1.0$ and $num\_of\_new\_ants\_starts := 8$
1.2 Set the values for $length_{(i,j)}$ between $town_i$ and $town_j$
1.3 Calculate all $visibility_{(i,j)}$ values
1.4 Set the initial value $\tau_{ij}(0) = 0$ for every $path(i, j)$
1.5 Place $b_{(a)}$ ants on every $town_{(a)}$
1.6 Insert $town_{(a)}$ in $ant_{(a)}$'s tabulist for all ants
1.7 Fill all $ant_{(k)}$'s $not\_visited\_list_{(k)}$ with values 0 to 12
1.8 Set $\Delta\tau^k_{ij}(0, 1) = 0$

2.0 For $gamma\_run$:= 0 to $gamma\_run\_max$ do
2.1 Repeat until every ant visits a different town by:
2.2 Generating a set $random[13]$ of 13 random numbers so that every number ocures only once
2.3 Insert the first element of this set $random[0]$ with $ant_{(0)}$'s tabu list and so on
2.4 Remove value associated with $random[0]$ from $ant_{(0)}$'s $not\_visited\_list_{(0)}$
2.5 Calculate transition probability $p_{ij}(t)$ for every $path(i, j)$

3.0 For $number\_of\_towns\_ant\_has\_visited$:= 2 to town do
3.1 Calculate $\rho \cdot \tau_{ij}(number\_of\_towns\_ant\_has\_visited)$
3.2 For $a := 0$ to $number\_of\_ants$
3.3 Choose the next $town_{(j)}$ for $ant_{(a)}$ at $town_{(i)}$ to visit by finding the biggest value for the possible $p_{ij}(number\_of\_towns\_ant\_has\_visited)$
3.4 Update $ant_{(k)}$'s $not\_visited\_list_{(k)}$ and tabulist
3.5 Calculate transition probability $p_{ij}(t)$ for every $path(i, j)$

4.0 determine the shortest tour-length of all ants and store this value and the tabulist

5.0 For $it$:= 1 to $num\_of\_new\_ants\_starts$ do
5.1 Place $b_{(a)}$ ants on ever $town(a)$
5.2 Insert town a in $ant_{(a)}$'s tabulist for all ants
5.3 Fill all $ant_{(k)}$'s $not\_visited\_list_{(k)}$ with value 0 to 12
5.4 For $number\_of\_towns\_ant\_has\_visited$:= 1 to town do
5.5 Calculate $\rho \cdot \tau_{ij}(number\_of\_towns\_ant\_has\_visited)$
5.6 For $a := 0$ to $number\_of\_ants$
5.7 Choose the next $town_{(j)}$ for $ant_{(a)}$ at $town_{(i)}$ to visit by finding the biggest value for the possible $p_{ij}(number\_of\_towns\_ant\_has\_visited)$
5.8 Update $ant_{(k)}$'s $not\_visited\_list_{(k)}$ and tabulist
5.9 Calculate transition probability $p_{ij}(t)$ for every $path(i, j)$

6.0 Determine the shortest tour-length of all ants and *it* and store this value and the tabulist

7.0 Display the shortest tour-length and its tabulist of all ants and *it* t

**A4:** Program flow to find shortest tour length