Solving the Traveling Sales Man Problem

Branch & Bound, Deluge, Ants

Helge Janicke, Niels-Peter de Witt, Karsten Wolke

June 9, 2002

1 Creating graphs

There are two different ways to create graphs. You can choose to draw the knots and edges or you can specify the distances between knots in a matrix. Feel free to use them both!
All graphs are asymetric, so you have to specify two edges if the connection is bidirectional.
A convienient way to create symetric test-graphs is to draw knots and choose `Menu - Edges - create Edges - Knot 2 Knot graphical distance`.
By the way, if you don't like the colors, press the right mouse button inside the drawing area...

1.1 Drawing graphs

The first tab in the main menu allows you to draw graphs. The information on the drawn knots and edges is shown below the drawing area. Here you can later change attributes or select knots/ edges. On the left side of the drawing region you find a panel where you can select between different drawmodes.

1.1.1 Draw modes

There are six modes, where you can select from:

Draw a knot
Choose this mode to draw knots. The mouse cursor will change to a cross when moving over the draw area. Draw a knot by clicking the left mousebutton. The name of the knot is taken from the textfield *name of the knot* below the draw region.

Select or move a knot
Choose this mode to select a knot. You can select a knot by clicking its' bullet with the left mouse button. When you keep the mouse button pressed and move the mouse, the knot will change its' position to the mouse position where you released the mouse button. The selected knot will be highlighted.

Removes a knot
Choose this mode to remove knots. The mouse cursor will change to a hand when moving over the draw area. Remove a knot by clicking on its' bullet.

Draws an edge
Choose this mode to draw edges. The mouse cursor will change to a cross when moving over

the draw area. To draw an edge, click on the knots' bullet, where the edge should start. Release the mouse button. You will see a line leading from this knot to your mousecursor. Click onto the knots' bullet which is to be the end of the edge, to create the edge. The length (cost) of this edge is taken from the *length (cost) of the edge* textfield below the draw region.

Select an edge
Choose this mode to select an edge. You can select an edge by clicking close to its' line. When there are more than one edge at this position you'll be asked to select the one you mean. The selected edge will be highlighted.

Removes an edge
Choose this mode to remove an edge. You can choose the edge by clicking close to its' line. When there are more than one possibilities you'll be asked to select the one you mean.

1.1.2 Information on knots and edges

Below the drawing region you find lists showing all knots and edges of the graph. Here you can select the knot/edge and change attributes for the selection.

Change the name of a knot
Select the knot you want to change the name of, using the select knot mode or by clicking its old name in the list of all knots. Its' name is copied to the textfield *name of the knot*. Edit the name and press *return* or click the *accept* button next to it.

Change the length (cost) of an edge
Select the edge you want to change the length of, using the select edge mode or by selecting it in the list of all edges. Its' current length is copied to the textfield *length of the edge*. Edit the length and press *return* or click the *accept* button next to it.
Note: The value you enter must be an valid double value or `infinity`.

1.2 Distance matrix

This is another way to create a graph. The N x N matrix is labled with the N cities' names (knots). Read it in this way:

It takes LENGTH (a value in a cell) *to go from city* READ LEFT CITIES NAME *to* READ CITY NAME ABOVE.

Add new cities using the *Add new city* button at the bottom.
You will be prompted for a *name* of the city (knots' name) and a position. The position (pixel) is used to draw the knot in the *Draw graph* tab. You can change the position any time by moving the knot see here 1.1.1.
Press OK to create the new city.

Change city changes the attributes for the knot.
You will be prompted for a new city name and new cooridiantes (pixel). Press OK to accept the changes.

Remove city removes a knot.
You will be propted for the city to remove. Select the city in the combobox and press OK to remove it.

Creating Edges / Changing length of edges
Since you have defined at least two knots, you see the matrix representing the shortest connections (edges) from one knot to another one; The value is the length (cost) of (for) the traversial. The diagonal elements are not accessible, because in most cases it doesn't make sense to specify an edge from one knot to itself.

Click on an cell to edit its' value. All double values are accepted. Leaving the cell blank will remove the edge. Anything not a number is treated as `infinity`.

2 Branch & Bound

Branch & Bound is an algorithm that finds an optimal solution. But the price to pay is high – even if the algorithm doesn't check all valid solutions, the complexity is exponential. This means you shouldn't try TSPs with more than 8-10 knots, unless you have plenty of time.

2.1 Starting Branch & Bound

To start B&B simply press the *SOLVE* button. The result is emphasised in both, the graph and the matrix. (If the graph doesn't highlight a path, you may have turned of *display - bestpath* in the rightclick context menu). The total length of the solution is shown in the *best path* textbox. The *time* textbox shows the time used for the evaluation.

You have the option for a stepwise evaluation. Press the *SINGLE STEP SOLVE* button to keep track of each expand made by the algorithm. Both graph and matrix will display the active path. The length of the active path is shown in the *active path* textfield.
After you've found the optimal solution, you may use *SINGLE STEP SOLVE* to find sub-optimal solutions.
The *time* textbox shows the time used for the evaluation step.

Press the *RESET* button to reset the algorithm.

2.2 The Branch & Bound Algorithm

Saying the algorithm doesn't check all valid solutions requieres an explanation of the algorithm.
We use a kind of treeprunning to cut off solution paths, that cannot lead to an better solution. Branch and Bound keeps track of all analysed paths using a sorted history list. The shortest paths can be found at the beginning.
In each step of the algorithm the first path (shortest) is expanded. This means that all edges that lead further and are not cyclic, or if cyclic are a solution, are sorted into the history.
If the first path already is a solution the algorithm ends because we assume that the length of each edge is positive.

For the more mathematical inclined:

Let:

Knot $s \in K$ be the start of the round trip, where K is the set of all knots.

$E(s) \subset E$ be a set of all edges leading away from s, where E is a set of all edges.

An edge is an triple $(k1, l, k2)$ where $k1$ is the start knot, $l \geq 0$ the length, $k2$ the end knot.

P be the actual processed path.

H a sorted list of all visited paths, shortest path first.

A path is a list of edges $P(0)$ to $P(j)$ satisfying the following equations:

$$P(0\ldots j)\epsilon E \tag{1}$$

$$\forall(0 \leq i < j)\ (P(i) \rightarrow k2 = P(i+1) \rightarrow k1) \tag{2}$$

A path is cyclic if:

$$\exists\, i\ P(i)\!\rightarrow\! k2\ \in\ \{P(n)\!\rightarrow\! k1 \setminus 0 \leq n \leq j\} \tag{3}$$

The length of a path is:

$$\sum_{i=0}^{j} P(i) \rightarrow l \tag{4}$$

Our implementation is very straight forward:

1. set start Knot s, $H = \{[e]\ |e \in E(s)\}$
2. first element x in H is a solution $\Rightarrow$ best solution

   $$(x(0)\!\rightarrow\! k1 = x(j)\!\rightarrow\! k2) \wedge \{x(i)\!\rightarrow\! k2\ |\forall i\} = K \tag{5}$$

   else set $P = x$
3. expand P

   $$M = \{[P, e]\ |e \in E(s)\} \tag{6}$$

4. check cycles

   $$C = \{x \in M\ |x \text{ is cyclic}\} \tag{7}$$

   $$\overline{C} = M \setminus C \tag{8}$$

5. check solution

   $$R = \{x \in C\ |x \text{ is solution (see 5)}\} \tag{9}$$

6. sort in

   $$H = H \cup \overline{C} \cup R \setminus \{P\} \tag{10}$$

7. $\Rightarrow$ 2.

3 The Great Deluge (GD)

The great deluge algorithm is founded on the assumption, that a person can walk through a hilly country and a tyde is slowly rising. She can not see much forward, only her nearest neighborhood. So she tries to change her way a little for not getting wet boots on the way. So she walks up and down the hills until she is trapped in the flooding, that surround her.

The great deluge algorithm can not promise that the solution, that it finds, is the global optimum. It can be a local optimum. That depends on the rising of the tyde and where the person and the tyde started. But the solution that can be found is sometimes almost as good as the global optimum and much faster than finding it by an algorithm like branch and bound (BB). It also has a solution after each step. This is advantageously because the algorithm can be stopped and can return the best solution it has found until now.

In our case we are searching the minimal length of a way through an amount of cities which are allowed to be visited only once. Only the startpoint is allowed to be visited again at the end of the path.
This is a little different than described above, where we were looking for a maximum. Now we are looking for a minimum, so the tyde is falling. The person is now like a fish in the water and looks for the deepest valley to survive while the tyde is falling.

We are using a graph to describe the problem. A graph exists on knots and edges. Knots are our cities and the edges are the ways from one city to another. A solution is a path through all cities without visiting one city twice, except the city we are starting with.

3.1 Getting started

There must be an initial solution. It can be a random one, but should be not too bad.
We generate our first solution by using a spanning tree. We take our first knot and expand the edges from this knot. After that we traverse the first edge. If there is no edge, or the way would be to long (infinity) we go back, remove the edge and take the next one. This is done recursivly until there is one solution. These solution is for the moment our best solution we have and will be saved. If the tree was fully expanded and there was no solution found we can stop our algorithm at this point and can say that there is no possible solution. But if there was a solution we can now look in our nearest neighborhood for a new path that is maybe better and not above the tyde (remember we are a fish and breath under water).

3.2 Finding a new path

Now we are looking for a new path. To get a new path we need two edges that have no common knots. If there are no such edges, it means that the current path has only three or less knots, the only chance to find a better solution is to reverse the path if this is possible. In most cases the problems have more than three knots, so that we can find these two edges by choosing them randomly from our path using the condition that they have no common knots. The path will be splitted into two pieces by removing these edges. First we try to build a new path connecting the startknot of the first chosen edge with the startknot of the second one. Then the path that starts with the endknot of our newly added edge must be reverseable and the end of this path must have a possible connection to its startknot (see figure below).

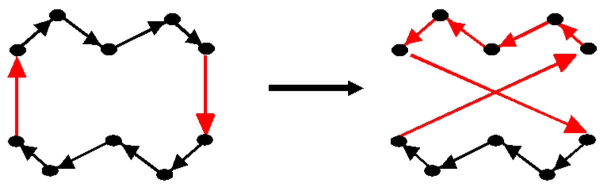

Figure 1: First try to change path [swapEdge2.png]

If this fails we try it the other way round, swapping the edges.

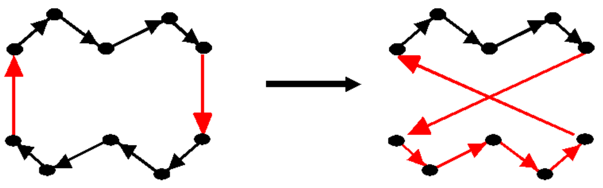

Figure 2: Second try to change path [swapEdge2.png]

If this fails, too, the last chance is to reverse our path (see figure below). If this also fails, these two edges lead to no solution and after decreasing the tyde there will be the possibility that in the next turn two other edges will be taken.

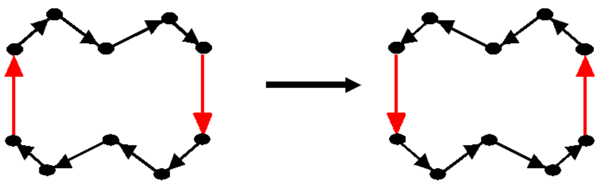

Figure 3: Try to reverse the path [reversePath.png]

But if one of these procedures above succeed and generate a new path this path must be checked, if it is a new solution. If the length of the path is too long, so it is above the tyde, the path will be canceled. If it is beneath the tyde it will be our new current solution. Is it even better than our best solution it will become our best solution. Now the tyde will decrease.

3.3 Decreasing the tyde

The tyde decreases using this formular:

$Tide = Tide - X * (Tide - Y * (length\ of\ current\ path))$

With X and Y the falling of the tyde can be affected. These factors can vary for each case found in ”finding a new path”. If a new path was found that was beneath the tyde it uses ALPHA for X and BETA for Y. If a path was found, but it leads to the surface the factor DELTA is used for X and GAMMA for Y and if there was no path possible PHI is used for X and TETA for Y. Values used for X should be near 0.0095 and for Y near 1. These are experienced values that are confirmed by us.

3.4 The end of the algorithm

After the tyde has decreased it is checked if the current path is at the surface. If it is, the algorithm is at its’ end and the best solution can be used. The current path must not be the best solution, because we can have once visited the deepest valley and are now stuck in a local valley somewhere else.
But if the current path is beneath the surface we go back to ”finding a new path” and repeat this until there is no water to breath or the user stopped the algorithm.

4 Ant Algorithm

A strong interest in recent years appeared on procedures for the solution of NP-hard combinatorial optimization problems which are inspired by natural courses of events. One of the most recent one of these nature analogous procedures is the optimization with ant colonies (ACO, Ant Colony Optimization). This is inspired by behavior of real ant colonies during forage. It represents the experiment as in the case of the genetic algorithms to solve optimization problems heuristically by adaptation of natural behavior. But all procedures, which use an heuristic basis can return a statement about the solution quality.

Nobody know precisely why the ants are ecologically so successful. Since 135 million years, this species hardly underwent changes in the evolution. The key could be the swarm intelligence how it also can be found at other insect states.

An ant can put pheromons in any thickness and mark its' trail this way. While isolated ants almost move by chance, an ant meeting a pheromone lane can observe these and follows it. The ant itself can put pheromons simultaneously again, strengthening the trace. In such a way, a positive reaction bow results: the more ants follow a path, the more attractive it becomes for the entire colony.

Pheromones evaporate with time similar to other perfumes. Therefore, unused paths loose their attraction potential as time passes by.

4.1 Transfer to computer science

The algorithms of the Ant Colony Optimization family make use of the pheromon lanes used by real ants for the communication. These represent partial solutions of several individuals which compositely represent a solution with each other.

The ant becomes an agent, all agents represents the system, the colony. An agent is comparable with the ant: in relation to the overall system, she is small and simple. Since her actions have only small influence on the overall system, her loss is negligible.

The cooperation between the agents is like the ants: every agent reads locally available markings. The Ant choose a way by probability, based on the length and the pheromons on the way. Accordingly, the amount of virtual pheromones gives information about the quality of the solution.

The evaporation can be modeled, that adaptive behavior becomes achieved - the presumably biggest feature of the ant algorithms. It is possible without reinitialization to make changes to the problem during the run.

4.2 Transfer to an Algorithm

We want now present the ant algorithm by using the TSP problem. Doing this, we need some information.

- Heuristic values
- Ant-Routing Tables
- History lists

The edges between the individual knots are chosen as heuristic values.

The Ant-Routing tables contain the marking strengths of the edges.

The History list contains knowledge. To go to every town only once every agent has a backup memory in which the attended cities are entered as a list. With this memory you can calculate the quality of the way.

An evaluation occurs after the end of all agent tours. Every agent then amplifies, depending on the quality of her tour solution, the density of pheromons on her way.

4.2.1 How does the algorithm work

1. During initialization, the agents are distributed, every town gets one agent. The start city is entered into the respective History list and the Iterator initialized.

2. The agents now move from city to city, where the next city is chosen with a specific probability, depending on the length and marker thickness. For every decision, the attended city is entered into the history list. This is repeated until all agents have ended their round trips.

3. Before the evaluation, the agents are reset onto their start city. The overall length of every agent tour based on the history lists (the lenght of the round trip) will then be computed. If this is shorter than the up to now shortest one, this one will be observed. For every edge, the length dependent evaluations of all agents are summed up.

4. The old markers evaporate (normally through a factor less than 1), then the new evaluations will be added. The iteration counter will be incremented.

5. If the iteration counter has the maximum iteration number the Algorithm stops and the up to now shortest tour will be the result. Otherwise all history lists are cleared and the algorithm is executed once more from the 2nd point.

4.2.2 Choice of the way continuation with weighting parameters

The choice of the way continuation happens to ants algorithm based on the marking strengths and the distances between the knots. These are provided with weighting parameters α and β which must be chosen reasonably. We define the length between a city i and a city j as l_{ij}.
$allowedK$ is the set of all non attendet cities. The probability p that a agent walk from city i to city j can compute as follows:

$$\begin{aligned} p_{ij} &= \frac{\tau_{ij}^{\alpha} * (\frac{1}{l_{ij}})^{\beta}}{N} \\ N &= \sum_{k \,\epsilon\, allowedK} \tau_{ik}^{\alpha} * (\frac{1}{l_{ik}})^{\beta} \end{aligned}$$

4.2.3 Choice of the weighting parameters

A random attitude of the weighting parameters forbids itself because you do not get sensible result.

If one chooses weighting for markers α too small, the cooperation ability between the ants is limited much. You'll receive a small search room per agent and the procedure resembles a `GREEDY` Algorithm.

If one chooses weighting for markers α too large, it leads to the over interpretation of the marker thicknesses of other ants: all ants follow the same tour and it comes to stagnation.

If one chooses weighting for markers β too small (less than 1),the algorithm works independent of the distances between the knots, but the TS problem and/or the algorithm depend on it!
The weight of the inverted length β should be between 1 and 5, in dependence of it weighting for markers α between 0.5 and 1. With these settings we were able to find the best (or close to it) solution of several TS problems. Moreover, the evaporation factor can still be chosen between [0,1]. Where 1 means no evaporation, and 0 means total evaporation. We were able to make good experiences with values between 0.5 and 1.