Team Members:
Guntakanti Sai Koushik(180050035), Mekala Anmol Reddy (180050059), Potta Nayan Akarsh (180050074),
Sourav Kumar (180050105)

Travelling Salesman Problem description:

"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?"

Design of the program:

Aspects

We are adopting the ACO algorithm to find the solution. We create artificial "ants" which go around different routes and update information about route quality along the route itself by depositing pheromone of fixed quantity along the path. This information is used by the next ants and finally these routes converge to the best solution i.e; the shortest route.

Input is selected by the user on a GUI and output will be displayed there itself along with the steps involved in arriving at the answer. The GUI will take the input and display the ant movements . The best loop found so far is displayed at every instant of time .And also the length of the best path is displayed. The solution is visible as it happens by a feature called "trace" which highlights a particular ant so that its motion can be followed so as to clearly see what's happening and also the best loop so far, which is highlighted.

We are also providing a way for solving the problem by the basic brute force method in order to show it's inefficiency. This solutions can also be tracked as it happens as the best-loop so far is highlighted. We have created a way to keep switching from different methods for the same set of points. And while in the brute force method, since we can't check all the solutions we will provide a probability on the screen about how sure we are that the best so far is the global best.

We have also put the option of solving this on a selected map so as to demonstrate its real-world-applicability.
This map case too has the option of solving by brute-force or the ACO optimization technique and also to switch between methods.

Code description

All the major aspects; like ants , points, edges (the "roads" which join 2 points) and loops (full cycles) are defined as objects in the beginning.
There are two files main.rkt and image.rkt,images are taken from outside and are processed in images.rkt.

Then we have some general definitions and global variable initializations. Then there is the graphics part of the code where we did a big bang and took the input from the GUI followed by the "main" part in which we solved the problem. In the end, we had to integrate the abstract and graphics part so that any change in the abstract part is immediately reflected in the graphics part by continuous updation as and when state change occurs.

Limitations and bugs

The major limitation is in the brute force method being that it can't check all possibilities for even cases where the number of cities is greater than 8. So we have tried only a limited number of times and displayed the probability that this is correct.

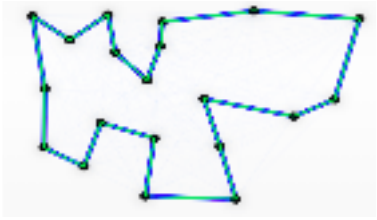
When solving on a map,you should select the points from 1 to 4 first and then click on the method.

While selecting the points on the screen, we should not drag the mouse to the right end of the screen, otherwise the program will crash and you will have to run the program once again. Anyways this can be avoided by properly following only the clicks that are needed at each stage..

Another 'limitation" is that, since ACO is an optimization technique, it helps reaching the correct answer very fast but as we've still not tried all we can't say that that is correct and we may end up reaching a local minima instead of a global minima. But we have partly counteracted that by drying the pheromone at time intervals during program execution so that over-weightage may not be given to an initially achieved local minima causing the unnecessary convergence. Only the "fittest" solutions survive the evaporation. Even this is not 100% perfect and can cause problems when a local minima is very close to the global it seems to work almost perfectly every time. Another thing that can be done is to increase the number of iterations when we do for larger numbers in the code.

Sample Input

Sample Output


(there's some animation too which can't be shown here)

Points of interest

- The program is heavily **object and structure oriented** and all the abstract objects are integrated with their corresponding graphics structs.
- **State abstraction** is the most fundamental part of the code as state is used severely in graphics - where the frame type to be decided, the buttons to be available, the variables stored so far from the GUI are all included in the graphics big-bang of state.

- State is also a major part of the object-oriented code segment as each ant must remember all the places it travelled to and the place where it started from-(because it must return there) and also the best vector found is put in state.
- We used abstraction by defining a higher order function for scaling the images.