STUDY AND APPLICATION OF ANT COLONY SYSTEM ON TRAVELLING SALESMAN PROBLEM

Palimpati and Stringfellow

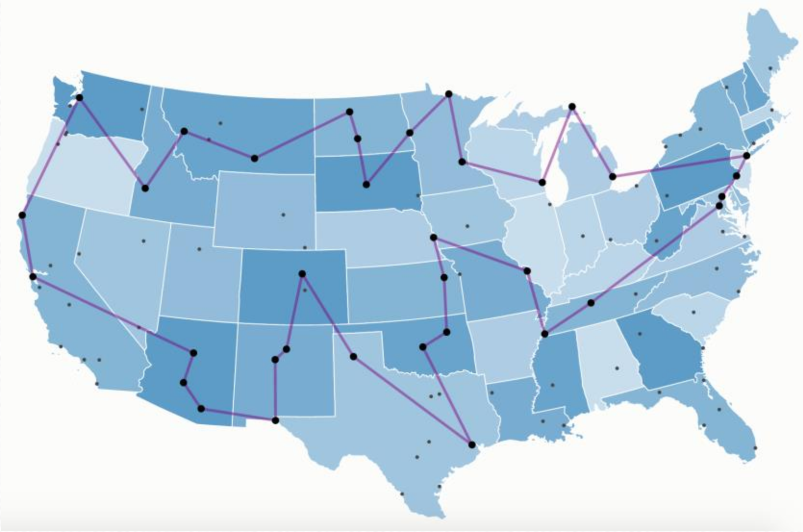
OVERVIEW

- Travelling salesman problem (**TSP**)
- ANT COLONY SYSTEM(**ACS**)
- Description of algorithms
- Rules applied
- Data sets used
- Results
- Summary

TRAVELLING SALESMAN PROBLEM

- Minimum length Hamiltonian circuit.
  - Shortest possible route that visits each city once and returns to the origin city.

- NP-HARD problem.



Graph Theory

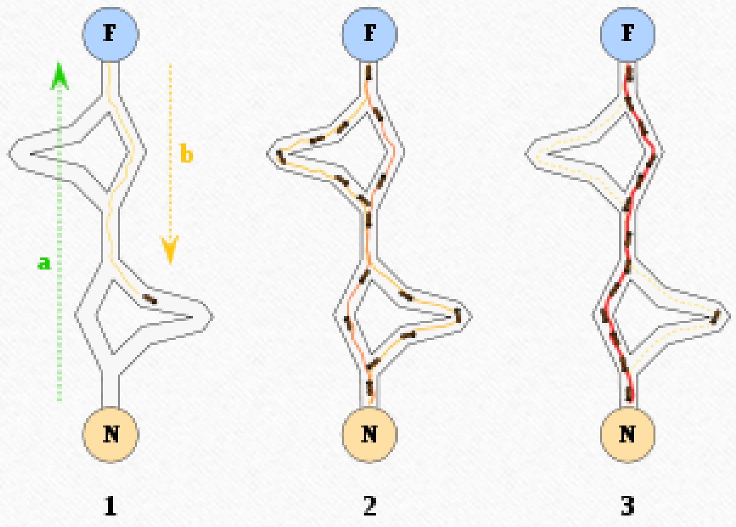
- Each node is referred to as a cities

- A path between two cities is denoted by a sequence of edge pairs *or a sequence of nodes*

- For study purposes: simple and fully connected graphs
- Large TSP data sets available online

ANT COLONY SYSTEM

- Ants deposit pheromones while searching for food.
- A pheromone evaporates with time.
- Ants choose a path depending upon the amount of pheromones.



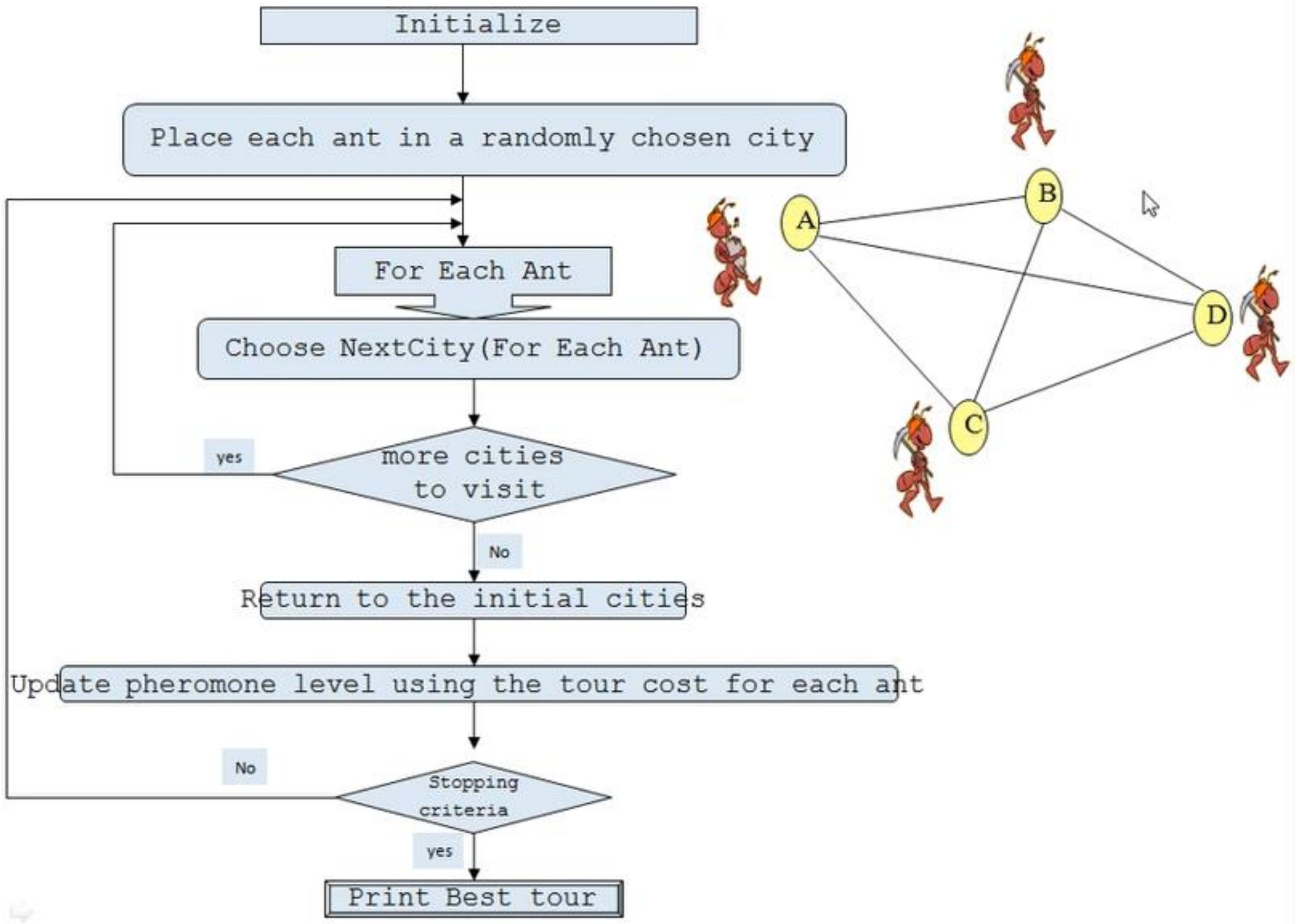
TOUR CONSTRUCTION IN ACS

1. An ant starts at a some city.

2. Use pheromone and heuristic (educated or trial and error) values to probabilistically construct a tour by iteratively adding cities that the ant has not visited yet, until all cities have been visited.

3. Go back to the initial city.

Ant Systems Algorithm for TSP



PHEROMONE AS A HEURISTIC IN ACS

- Initialize edge weights
- Update edge weights (add the pheromone to traversed edges)
- Decrement edge weights (evaporate pheromone)

INITIALIZATION OF PHEROMONE IN ACS

- Set equal amount of pheromone for all edges
  - value > expected amount of pheromone deposited by an ant during an iteration.
- If set too low, the search is quickly biased by the first tours generated by the ants.
- If set too high, many iterations are lost waiting until pheromone evaporates enough.

SIGNIFICANT FACTORS IN ACS ALGORITHM

- Pheromone τ – remaining information on the path the ants pass through
  - $\tau_{ij} = \text{weight} of\ the\ edge\ between\ nodes\ i\ and\ j$

- Visibility η – reciprocal distance between nodes
  - $\eta_{ij} = \frac{1}{d_{ij}},\ d\ is\ distance\ between\ nodes\ i\ and\ j$

SIGNIFICANT FORMULA IN ACS ALGORITHM

- Probability P – At each construction step of the shortest path, ant_k applies a probabilistic rule to decide which city to visit next.

- The shortest travelled path so far, will have a high τ and a high η making P (probability) stronger for selection purposes.

RANDOM PROPORTIONAL RULE

- The rule is given as follows:

$$p_{\mathrm{ij}}^{k} = \frac{\tau_{ij}^{\alpha}\ \eta_{ij}^{\beta}}{\sum_{z\in i} \tau_{iz}^{\alpha}\ \eta_{iz}^{\beta}},\quad \mathit{0\ if\ no\ next\ city}$$

k is an ant.

$\boldsymbol{p_{ij}^{k}}$ is probability of choosing the next city j when an ant is at i.

τ_{ij} is pheromone quantity between the edge between x and y.

$\eta_{ij} = \frac{1}{d_{ij}}$, where d is the distance between i and j.

α is a parameter to control the influence of pheromone.

β is a parameter to control the influence of distance.

ACS TOUR CONSTRUCTION

- With a probability of P the ant makes the best possible move.
- With a probability of (1-P), the ant performs a biased exploration of edges.
- Tuning P allows modulation of the degree of exploration and the choice to explore other tours.

- The ant which has the best tour will now add pheromone after each iteration according to Global Pheromone Trail Update.

Global Pheromone Trail Update

In pheromone trail update, both evaporation and new pheromone deposit only apply to the arcs that belong to the best so far solution.

- $\Delta\tau_{ij} = Q/Lk$ is quantity of pheromone laid on the path (i,j) by ant k, where Q is a constant value of total pheromone and L_k is the tour length of the kth ant.
- Update $\tau_{ij} = (1 - \rho)\,\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}$, where ρ is the constant evaporation factor and m is the number of ants in the colony

Improvement in Efficiency

- The pheromone update at each iteration is reduced from $O(n^2)$ to $O(n)$.

Local Pheromone Trail Update

- Ants use a local pheromone update rule that they apply immediately after having crossed an arc (I ,j) during their tour.

$$\tau_{ij} = (1 - \rho)\, \tau_{ij} + \rho\, \tau_0$$

- Each time an ant uses an arc its pheromone level is reduced, so that the arc becomes less desirable for the following ants.

PYTHON CODE IMPLEMENTATION

- The SKO package consists of optimization algorithms and sample code.

```
from sko.ACA import ACA_TSP
```

- In the following example code, `size_pop` is set to 3 ants to find the shortest path among 12 cities (points).
- Function `cal_total_distance()` refers to a path length computation and is the objective function.
- Function `ACA_TSP()` finds the best path given the data.

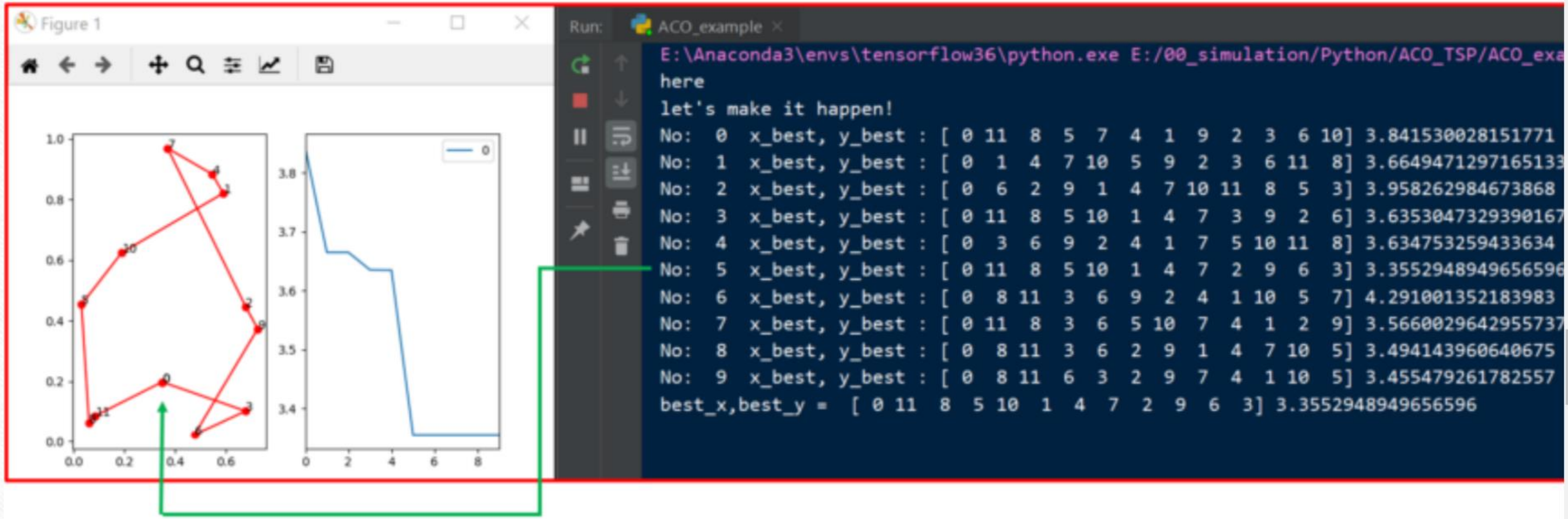
```python
from __future__ import division
import numpy as np
from scipy import spatial
import pandas as pd
import matplotlib.pyplot as plt
from sko.ACA import ACA_TSP

# generate coordinate points
num_points = 12
points_coordinate = np.random.rand(num_points, 2)
distance_matrix = spatial.distance.cdist(points_coordinate, points_coordinate, \
                       metric='euclidean')


def cal_total_distance(routine):
    num_points, = routine.shape
    return sum([distance_matrix[routine[i % num_points], \
                 routine[(i + 1) % num_points]] \
                      for i in range(num_points)])
```

```
def main():
    #find shortest path
    aca = ACA_TSP(func=cal_total_distance, n_dim=num_points,
                  size_pop=3, max_iter=10,
                  distance_matrix=distance_matrix)
    best_x, best_y = aca.run()

    # Plot the result
    fig, ax = plt.subplots(1, 2)
    best_points_ = np.concatenate([best_x, [best_x[0]]])
    best_points_coordinate = points_coordinate[best_points_, :]
    for index in range(0, len(best_points_)):
        ax[0].annotate(best_points_[index], (best_points_coordinate[index, 0], \
                                             best_points_coordinate[index, 1]))
    ax[0].plot(best_points_coordinate[:, 0], best_points_coordinate[:, 1], 'o-r')
    pd.DataFrame(aca.y_best_history).cummin().plot(ax=ax[1])
    plt.show()
```



ACO-TSP finds the best travel-sequence in the No. 5

INSIGHT INTO THE ACO_TSP RUN( ) FUNCTION

ACO_TSP problem (1997)

- TSP : travel sales problem.

Find a shortest route of visiting all red-dot points

$$\eta_{ij} = \frac{1}{d_{ij}} \text{ , } d \text{ is distance}$$

$$P_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{z \in i} \tau_{iz}^{\alpha}\eta_{iz}^{\beta}} \\ 0 \text{ , otherwise} \end{cases}$$

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k} \\ 0 \text{ , otherwise} \end{cases}$$

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

```python
def run(self, max_iter=None):
    self.max_iter = max_iter or self.max_iter
    for i in range(self.max_iter):  # 对每次迭代
        prob_matrix = (self.Tau ** self.alpha) * ((self.prob_matrix_distance) ** self.beta)  # 转移概率，无须归一化。
        for j in range(self.size_pop):  # 对每个蚂蚁
            self.Table[j, 0] = 0  # start point，其实可以随机，但没什么区别
            for k in range(self.n_dim - 1):  # 蚂蚁到达的每个节点
                taboo_set = set(self.Table[j, :k + 1])  # 已经经过的点和当前点，不能再次经过
                allow_list = list(set(range(self.n_dim)) - taboo_set)  # 在这些点中做选择
                prob = prob_matrix[self.Table[j, k], allow_list]
                prob = prob / prob.sum()  # 概率归一化
                next_point = np.random.choice(allow_list, size=1, p=prob)[0]
                self.Table[j, k + 1] = next_point
        # 计算距离
        y = np.array([self.func(i) for i in self.Table])
        # 顺便记录历史最好情况
        index_best = y.argmin()
        x_best, y_best = self.Table[index_best, :].copy(), y[index_best].copy()
        self.x_best_history.append(x_best)
        self.y_best_history.append(y_best)
        # 计算需要新涂抹的信息素
        delta_tau = np.zeros((self.n_dim, self.n_dim))
        for j in range(self.size_pop):  # 每个蚂蚁
            for k in range(self.n_dim - 1):  # 每个节点
                n1, n2 = self.Table[j, k], self.Table[j, k + 1]  # 蚂蚁从n1节点爬到n2节点
                delta_tau[n1, n2] += 1 / y[j]  # 涂抹的信息素
            n1, n2 = self.Table[j, self.n_dim - 1], self.Table[j, 0]  # 蚂蚁从最后一个节点爬回到第一个节点
            delta_tau[n1, n2] += 1 / y[j]  # 涂抹信息素
        # 信息素飘散+信息素涂抹
        self.Tau = (1 - self.rho) * self.Tau + delta_tau
    best_generation = np.array(self.y_best_history).argmin()
    self.best_x = self.x_best_history[best_generation]
    self.best_y = self.y_best_history[best_generation]

    return self.best_x, self.best_y
```

Output,
50_ants by 25 cities matrix distance data

output

OTHER VIDEOS

For a trace of the algorithm:

https://www.youtube.com/watch?v=783ZtAF4j5g

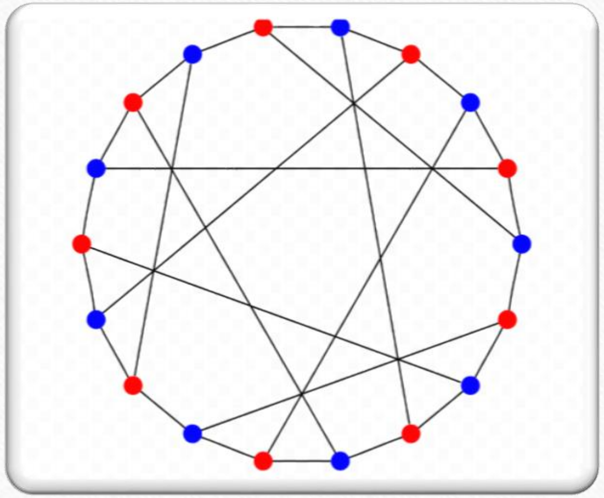
NEAREST NEIGHBOR (NN) ALGORITHM

1. Initialize all vertices as unvisited.
2. Select an arbitrary vertex, set it as the current vertex U. Mark U as visited.
3. Find the shortest edge from U to one of the unvisited vertex V.
4. Mark V as the current vertex and mark it as visited.
5. If all the vertices are visited, then terminate.
6. Otherwise go to step 3.

DATA SETS

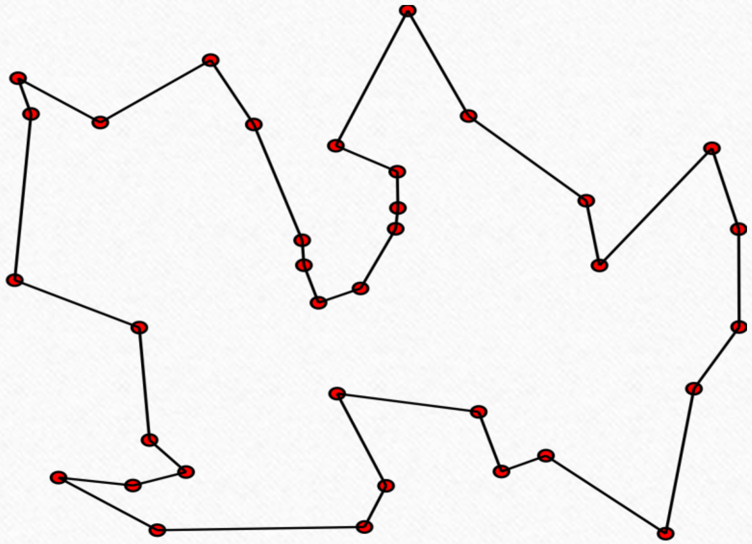
<u>All points on a circle</u>

ACS and Nearest Neighbour algorithm will work *perfectly* as the nearest node will always be on the circle.



Data sets

Random Nodes: As the number of Nodes increase the ACS performance decreases as the number of solutions increase.



Some NN vs ACS Results

Relative errors obtained using NN and ACS on Travelling Sales Man Problem (compared to known optimal solution).

NO OF NODES	NN RELATIVE ERROR	ACS RELATIVE ERROR
6	10.37	2.37
17	20.47	4.05
26	18.67	4.85
48	121.07	88.96

RESULTS


140
120
100
80
60
40
20
0
-20
RELATIVE ERROR
0
10
20
30
40
50
60
NUMBER OF CITIES
NN ERROR
ACS ERROR

SUMMARY

- Tough ACS does not yield an optimal solution it gives much better results than many other algorithms.
- More number of paths are explored in ACS as the individual ants take individual routes while constructing tour.
- Tuning parameters will improve the results obtained by ACS.