An Implementation of Ant Colony Optimization for the Traveling Salesperson Problem

İbrahim Erdem KALKAN *

June 25, 2020

Abstract

Ant Colony Optimization is one of the meta-heuristic approaches using an iterative stochastic solution construction process. It has been developed different versions and applications for generic NP-Hard problems since its first introduction. One of the famous one these problems is Traveling Salesperson Problem (TSP). In this paper an implementation of Ant System, the first algorithm introduced as Ant Colony Optimization, built with Python Numpy library for TSP is introduced and tested some small sized examples. With relatively small sized data, some experiments show that the algorithm is capable of finding optimal solution, but towards NP-Hard it is obtained only nearer solutions.

Keywords. Ant Colony, Algorithm, Optimization, Traveling Salesperson

1 Introduction

Ant Colony Optimization (ACO), which is introduced by Dorigo [5] [6] is a recent meta-heuristic approach for solving hard combinatorial optimization (CO) problems. The inspiring source of ACO is the pheromone trail laying and following behavior of real ants which use pheromones as a communication medium [7]. This behaviour enables them to find short paths between their nest and food sources. This characteristic of real ant colonies is exploited in ACO algorithms in order to solve, for example, discrete optimization problems [1].

Seen from the operations research (OR) perspective, ACO algorithms belong to the class of meta-heuristics [1]. Meta-heuristics for optimization problems may be described summarily as a "walk through neighbourhoods", a search trajectory through the solution domain of the problem at hand. Similar to classical heuristics, these are iterative procedures that move from a given solution to another solution in its neighbourhood [2].

Artificial ants used in ACO are stochastic solution construction procedures that probabilistically build a solution by iteratively adding solution components to partial solutions by taking into account (i) heuristic information on the problem instance being

*Cukurova University, Institute of Natural and Applied Sciences, Industrial Engineering, 2020911063

solved, if available, and (ii) (artificial) pheromone trails which change dynamically at run-time to reflect the agents' acquired search experience [7].

The first ACO algorithm proposed was Ant System (AS). AS was applied to some rather small instances of the traveling salesperson problem (TSP) [6]. A brief introduction about AS algorithm and TSP is given with following sections, and finally AS is implemented for TSP.

2 Ant System

AS was the first example of an ACO algorithm to be proposed in the literature whose computational results were promising but not competitive with other more established approaches [7].

Each meta-heuristic has its own behaviour and characteristics. All, however, share a number of fundamental components and perform operations that fall within a limited number of categories. To facilitate the comparison of parallelization strategies for various meta-heuristic classes, it is convenient to define these common elements [2]:

- Initialization
- Neighbourhoods
- A neighbourhood selection criterion
- Candidate selection
- Acceptance criterion
- Stopping criteria

Similar to meta-heuristics generic the ACO approach attempts to solve an optimization problem by iterating the following two steps [1]:

- candidate solutions are constructed using a pheromone model, that is, a parameterized probability distribution over the solution space;
- the candidate solutions are used to modify the pheromone values in a way that is deemed to bias future sampling toward high quality solutions.

According to Blum [1], Ant Colony meta-heuristic algorithm pseudo-code is introduced as follows:

Algorithm 1: Ant Colony Optimization (ACO)

while *termination_conditions_not_met* **do**
```
    AntBasedSolutionConstruction()
    PheromoneUpdate()
    DeamonActions()
```
end

Algorithm 2: The Procedure Ant Based Solution Construction

$s = \langle\rangle$
Determine $N(s)$
while $N(s) \neq \varnothing$ **do**
  $c \leftarrow$ `ChooseFrom(`$N(s)$`)`
  $s \leftarrow$ extend s by appending solution component c
  Determine $N(s)$
end

3 Traveling Salesperson Problem

The Traveling Salesperson Problem (TSP) is one of the most widely studied combinatorial optimization problems. Its statement is deceptively simple, and yet it remains one of the most challenging problems in OR [3]. An ACO application for TSP is introduced by Dorigo and Di Caro [6] It is mentioned by them, since it is the first problem to be attacked by ACO methods, TSP plays a central role in ACO.

It can be found a definition of TSP in Laporte [3]: Let $G = (V, A)$ be a graph where V is a set of n vertices. A is a set of arcs or edges, and let $C = (C_{ij})$ be a distance (or cost) matrix associated with A. The TSP consists of determining a minimum distance circuit passing through each vertex once and only once. According to Blum [1], the nodes V of this graph represent the cities, and the edge weights represent the distances between the cities. The goal is to find a closed path, called a tour, in G that contains each node exactly once and whose length is minimal. Thus, the search space consists of all tours in G. The objective function value of a tour is defined as the sum of the edge-weights of the edges that are in that tour. The TSP can be modelled in many different ways as a discrete optimization problem. The most common model consists of a binary decision variable X_e for each edge in G. If in a solution $Xe = 1$, then edge e is part of the tour that is defined by the solution.

4 Implementation

A generic symmetric TSP model is coded with "Python" language with "Numpy" library according to the algorithm mentioned above. Numpy is one of the suitable libraries for probabilistic selection. It is determined an iteration number to be used for termination condition of algorithm. The source code is available at https://github.com/ibrahimerdem/ModernHeuristicsFinal/blob/master/AntSystem.py (For repository: https://github.com/ibrahimerdem/ModernHeuristicsFinal)

[1]:
```
def VeryBadTour(costs, number_of_cities):
    t = list(range(number_of_cities))
    return SolutionValue(costs, t)
```

According to Luke [9], for the TSP, the ACO folks often set the initial value of pheromone is $popsize * (1/Cost(D))$, where *popsize* is number of cities constructing a trail, D is some costly, absurd tour like the Nearest Neighbor Tour. *VeryBadTour(costs, number_of_cities)* is used to calculate an absurd D value with using just an ordered sequence, t, where *costs* represents input data and $number\_of\_cities$ represents number of input cities.

[2]:
```
def PheromoneUpdate(v, s):
    global pheromones
    global evaporation_constant
    for ix, i in np.ndenumerate(pheromones):
        pheromones[ix[0]][ix[1]] = (1-evaporation_constant) * i
    for j in range(len(s)-1):
        pheromones[s[j]][s[j+1]] += (1 / v)
        pheromones[s[j+1]][s[j]] += (1 / v)
```

Pheromone evaporation and pheromone deposit operations are performed by *PheromoneUpdate(v, s)*. It takes the solution value and solution sequence as input. The procedure also uses the global variables of pheromones matrix and *evaporation_constant*. For symmetric TSPs, the distances between the cities i, and j are independent of the direction of traversing the edges [7]. Since the distances between i to j and between j to i are the same, this edge is represented twice in the input data and the pheromones matrix, thus, the pheromone is deposited twice.

[3]:
```
def SolutionValue(costs, s):
    v = 0
    for i in range(len(s)):
        if i == len(s)-1:
            v = v + costs[s[i]][s[0]]
        else:
            v = v + costs[s[i]][s[i+1]]
    return v
```

SolutionValue(costs, s) is the procedure that calculate the current solution's function value. It takes costs data and solution sequence as input. It returns the value, v.

[4]:
```
def SolutionConstruction(l = None):
    if type(l) == int:
        s = [l]
    else:
        s = []
    N = Determine(s)
    while len(N) > 0:
        c = ChooseFrom(N, s)
        s.append(c)
```

```
        N = Determine(s)
    return s

def Determine(s):
    global number_of_cities
    N = list(range(number_of_cities))
    if len(s) > 0:
        for i in s:
            N.remove(i)
    return N

def ChooseFrom(N, s):
    global pheromones
    global alpha
    global beta
    probabilities = []
    total_factor = 0
    if len(s) > 0:
        current_loc = s[-1]
        for n in N:
            phe = pheromones[current_loc][n]
            dis = costs[current_loc][n]
            total_factor += (np.power(phe, alpha) * np.power(1/dis, beta))
        for n in N:
            phe = pheromones[current_loc][n]
            dis = costs[current_loc][n]
            decision_probability = (np.power(phe, alpha) * np.power(1/
 ↪dis, beta)) / total_factor #pheromone model
            probabilities.append(decision_probability)
        c = np.random.choice(N, p=probabilities, size=1)[0]␣
 ↪#probabilistic decision for next state
    else:
        c = np.random.choice(N) #random starting location if it is not set
    return c
```

AntSystem() uses *SolutionConstruction(l = starting_location)* procedure to construct a candidate solution. The procedure determines the list of available states with the procedure *Determine(s)*, and chooses the next state from the list of available states with the procedure of *ChooseFrom()*. *SolutionConstruction()* can take an optional argument of starting location if it is preferred the determined starting position information as an input. *SolutionConstruction()* returns with the new candidate solution sequence of s. $total\_factor$ represents the denominator of the composition of the pheromone trails and heuristic values seen below [6]:

$$a_{ij} = \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in \mathcal{N}_i}[\tau_{il}(t)]^{\alpha}[\eta_{il}]^{\beta}} \quad \forall j \in \mathcal{N}_i$$

The heuristic values are the distance between cities i and j seen below. In other words, the shorter the distance between two cities i and j, the higher the heuristic value [6].

$$\eta_{ij} = 1/J_{c_i c_j}$$

To construct candidate solutions *ChooseFrom(N, s)* procedure takes two argument, *N* is a list of possible cities in order to produce a feasible solution, and *s* is represents current sequence so far. Before choosing, it must be run *Determine()* procedure which takes existing sequence of trail so far as an argument.

[5]:
```
import numpy as np

evaporation_constant = 0.5
number_of_cities = 2
pheromones = np.ones((number_of_cities, number_of_cities))
alpha = 1
beta = 2

def Initialization(costs, e, a, b):
    global evaporation_constant
    evaporation_constant = e
    global number_of_cities
    number_of_cities = len(costs)
    global alpha
    alpha = a
    global beta
    beta = b
    global pheromones
    D = VeryBadTour(costs, number_of_cities)
    initial_pheromone_value = number_of_cities / D
    initial_pheromones = np.ones((number_of_cities, number_of_cities))
    initial_pheromones.fill(initial_pheromone_value)
    pheromones = initial_pheromones

def AntSystem(costs, e = 0.5, n = 10, alpha = 1, beta = 5,␣
 ↪starting_location = None):
    Initialization(costs, e, alpha, beta)
    best_solution = []
```

```
    best_value = VeryBadTour(costs, number_of_cities)
    while n > 0:
        s = SolutionConstruction(l = starting_location)
        v = SolutionValue(costs, s)
        if v < best_value or len(best_solution) == 0:
            s.append(s[0])
            best_solution = s
            best_value = v
        PheromoneUpdate(v, s)
        n = n - 1
    print(f'{best_solution} is the best tour and {best_value} is the best␣
↪value')
```

The main block of code is *AntSystem(costs, e = 0.5, n = 100, alpha = 1, beta = 1, starting_location = None)*, where inputs:

- *costs* is a matrix (n, n) where *n* is number of cities, representing all combination of distances between cities,
- *e* is evaporation rate with default value of .5,
- *n* is number of iteration of algorithm with default value of 100,
- *alpha* and *beta* are the parameters to control the relative weight of pheromone trail and heuristic value [6] and their default values are 1 and 5 respectively [5],
- *starting_location* can be take an integer value which can be used in order to start the tour with specific location, but the default value of it is *None*.

Procedure evaluates different solution values to find the least and return a sequence *best_solution* representing the cities where the salesperson going through and finished a tour, and a value of objective function *best_value*.

Initialization(costs, e, alpha, beta) sets the initial values of global variables and *pheromones* matrix.

5 Tests and Results

For testing the implementation it is chosen a problem example from Winston's Operations Research book [4]. One can be found in the book a small sized TSP, and a linear programming construction of it.

Example. Joe State lives in Gary, Indiana. He owns insurance agencies in Gary, Fort Wayne, Evansville, Terre Haute, and South Bend. Each December, he visits each of his insurance agencies. The distance between each agency (in miles) is shown in table below. What order of visiting his agencies will minimize the total distance traveled ?

	Gary	Fort Wayne	Evansville	Terre Haute	South Bend
Gary	0	132	217	164	58

	Gary	Fort Wayne	Evansville	Terre Haute	South Bend
Fort Wayne	132	0	290	201	79
Evansville	217	290	0	113	303
Terre Haute	164	201	113	0	196
South Bend	58	79	303	196	0

For solution, let numbers $[0, 4]$ are represent the cities respectively in order to construct cost matrix. According to Winstons's example the starting location salesperson corresponding to number 0, thus, the *starting_location* is set to be 0. With the inputs of cost matrix seen below, 100 of iteration, $alpha = 1$, $beta = 5$, and .5 of evaporation constant, the outputs of all trials is $\langle 0-4-1-3-2-0 \rangle$ sequence with 668 miles cost value. According to Winston, it is obtained that algorithm results with an optimum solution.

One can observe all input data and experiment results at : https://github.com/ibrahimerdem/ModernHeuristicsFinal/blob/master/AntSystemNoMarkdown.ipynb. One can also download the Jupyter Notebook file to execute the implementation.

With the same inputs, without *starting_location*, which means with random starting location, it is obtained twice $\langle 1-4-0-3-2-1 \rangle$ route with 704 miles, once $\langle 3-2-1-4-0-3 \rangle$ route with 704 miles, twice $\langle 3-2-0-4-1-3 \rangle$ route with 668 miles, once $\langle 2-3-1-4-0-2 \rangle$ route with 668 miles, once $\langle 3-1-4-0-2-3 \rangle$ route with 668 miles, and 3 times $\langle 0-4-1-3-2-0 \rangle$ with 668 miles outputs of 10 trials.

For further analysis the with same data, it is tested a drawback of nearest-neighbor heuristic (NNH) mentioned by Winston. With NNH, and *starting_location* value 2, it is obtained $\langle 2-3-0-4-1-2 \rangle$ route with 704 miles corresponding to a non-optimal solution. However, with ACO, it is obtained 5 times $\langle 2-3-1-4-0-2 \rangle$ route with 668 miles, and 5 times $\langle 2-3-0-4-1-2 \rangle$ route with 704 miles outputs of 10 trials. In addition, with 1000 of iterations, it is obtained 5 times $\langle 2-3-1-4-0-2 \rangle$ route with 668 miles, 4 times $\langle 2-3-0-4-1-2 \rangle$ route with 704 miles output of 10 trials.

Another example is gathered from Google OR-Tools and tested with the data of 12 locations (Available at: https://developers.google.com/optimization/routing/tsp). Cities are represented by integers as: 0 New York, 1 Los Angeles, 2 Chicago, 3 Minneapolis, 4 Denver, 5 Dallas, 6 Seattle, 7 Boston, 8 San Francisco, 9 St. Louis, 10 Houston, 11 Phoenix, 12 Salt Lake City. As a results of 20 experiments with starting point of 0, $\langle 0-7-2-9-5-10-11-1-8-6-12-4-3-0 \rangle$ route with the value 7343 miles is obtained. However, optimal solution value can be seen as 7293 miles.

Different size of data can be gathered from TSPLIB. *Bays*29 represents the data of 29 cities in Bavaria, street distances (Groetschel,Juenger,Reinelt)(Available at: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/bays29.tsp). According to 20 experiments with this data and starting location of 0, $\langle 0-27-5-11-8-4-25-28-2-1-19-9-3-14-17-13-21-16-10-18-15-24-6-22-26-23-12-20-7-0 \rangle$ sequence is obtained once with the value of 2288 miles. On the other hand the optimal

solution value of *Bays*29 reported as 2020 miles.(Optimal solutions are available at: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/stsp-sol.html)

6 Conclusion

Experiments show that ACO with the small size of data as in the example, capable of constructing optimal solution with a non-trivial probability. Furthermore, with relatively high number of iterations, one can notice that the probability of optimal solution can be increased. On the other hand with a bit more sizable data results can only approach the optimum solution. Since the obtaining probability of optimal solution decreases, the capability of avoiding local extremum obstacles of ACO decreases with given inputs. However, it should be tested same data with different values of parameters *alpha* and *beta*.

As it is mentioned by Blum [1], even though the original AS algorithm achieved encouraging results for the TSP problem, it is found to be inferior to state-of-the-art algorithms for the TSP as well as for other CO problems. Therefore, several extensions and improvements of the original AS algorithm were introduced over the years. As an extension a "hill-climbing" effort can be assembled to the original algorithm as proposed by Luke [9].

As a fairly young research field, it is expected a meta-heuristic some desirable properties. In terms of desirable properties [8], ACO meets most of desires with its simplicity, precision, coherence and effectiveness, but with this initial version, AS, it can be considered as efficiency and robustness are not completed. However, one can be overcome these drawbacks with extensions and improved versions, in terms of innovation.

References

[1] Blum C. Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews*, 2(4):353–373, 2005.

[2] Crainic T. G. and Toulouse M. Parallel strategies for meta-heuristics. *Handbook of Metaheuristics*, 2003.

[3] Laporte G. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.

[4] Winston W. L. *Operations Research Applications and Algorithms*. Thomson, fourth edition, 2004.

[5] Dorigo M. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992.

[6] Dorigo M. and Di Caro G. Ant colony optimization: A new meta-heuristic. volume 2, page 1477 Vol. 2, 02 1999.

[7] Dorigo M. and Stützle T. The ant colony optimization metaheuristic: algorithms, applications, and advances. *Handbook of Metaheuristics*, 2003.

[8] Hansen P. and Mladenovic N. Variable neighborhood search. *Handbook of Metaheuristics*, 2003.

[9] Luke S. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/$\sim$sean/book/metaheuristics/.