#### Koneru Lakshmaiah Education Foundation

**(Deemed to be University)**

#### Bachelor of Technology in Department of Computer Science Engineering

**A Project Based Report On**

#### ANT COLONY OPTIMIZATION AND

**GOMORY’S CUT PLANE METHOS**

**SUBMITTED BY:**

**2100031406 S .SUJIT KUMAR SEC 2**

**UNDER THE GUIDANCE OF**

**DR.Akhilesh Dubey**



**KL UNIVERSITY**

Green fields, Vaddeswaram – 522 502 Guntur Dt., AP, India.

# CONTENTS

OBJECTIVE 3

ADVANTAGES 5

DISADVANTAGES 6

PROGRAMME 7

RESULTS 9

CONCLUSION 13

### OBJECTIVE

#### ANT COLONY OPTIMIZATION:

In [computer science](https://en.wikipedia.org/wiki/Computer_science) and [operations research](https://en.wikipedia.org/wiki/Operations_research), the **ant colony**

**optimization** [algorithm](https://en.wikipedia.org/wiki/Algorithm) (**ACO**) is a [probabilistic](https://en.wikipedia.org/wiki/Probability) technique for solving computational problems which can be reduced to finding good paths through [graphs](https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)). Artificial ants stand for [multi-agent](https://en.wikipedia.org/wiki/Multi-agent) methods inspired by the behavior of real [ants](https://en.wikipedia.org/wiki/Ant). The pheromone-based communication of biological ants is often the predominant paradigm used. Combinations of artificial ants and [local search](https://en.wikipedia.org/wiki/Local_search_(optimization)) algorithms have become a method of choice for numerous optimization tasks involving some sort

of [graph](https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)), e.g., [vehicle routing](https://en.wikipedia.org/wiki/Vehicle_routing_problem) and internet [routing](https://en.wikipedia.org/wiki/Routing).

This algorithm is a member of the ant colony algorithms family, in [swarm](https://en.wikipedia.org/wiki/Swarm_intelligence) intelligence methods, and it constitutes some [metaheuristic](https://en.wikipedia.org/wiki/Metaheuristic) optimizations. Initially proposed by [Marco Dorigo](https://en.wikipedia.org/wiki/Marco_Dorigo) in 1992 in his PhD thesis,the first algorithm was aiming to search for an optimal path in a graph, based on the behavior of [ants](https://en.wikipedia.org/wiki/Ants) seeking a path between their [colony](https://en.wikipedia.org/wiki/Ant_colony) and a source of food. The original idea has since diversified to solve a wider class of numerical problems, and as a result, several problems have emerged, drawing on various aspects of the behavior of ants. From a broader perspective, ACO performs a model-based search and shares some similarities with [estimation of distribution algorithms](https://en.wikipedia.org/wiki/Estimation_of_distribution_algorithm).

Different optimization techniques have thus evolved based on such evolutionary algorithms and thereby opened up the domain of metaheuristics. Metaheuristic has been derived from two Greek words, namely, Meta meaning one level

above and heuriskein meaning to find. Algorithms such as the Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO) are examples of swarm intelligence and metaheuristics. The goal of swarm intelligence is to design intelligent multi-agent systems by taking inspiration from the collective behaviour of social insects such as ants, termites, bees, wasps, and other animal societies such as flocks of birds or schools of fish.

Stage 1: All ants are in their nest. There is no pheromone content in the environment. (For algorithmic design, residual pheromone amount can be considered without interfering with the probability)

Stage 2: Ants begin their search with equal (0.5 each) probability along each path. Clearly, the curved path is the longer and hence the time taken by ants to reach

food source is greater than the other.

Stage 3: The ants through the shorter path reaches food source earlier. Now, evidently they face with a similar selection dilemma, but this time due to pheromone trail along the shorter path already available, probability of selection is higher.

Stage 4: More ants return via the shorter path and subsequently the pheromone concentrations also increase. Moreover, due to evaporation, the pheromone concentration in the longer path reduces, decreasing the probability of selection of this path in further stages. Therefore, the whole colony gradually uses the shorter path in higher probabilities. So, path optimization is attained.

**Gomory cut plane method :**

Branch-&-Cut algorithms by linking application specific routines to the generic algorithm included in the solver engine. We start providing an introduction to cutting planes and cut separation routines in the next section, following with a section describing how these routines can be embedded in the Branch-&-Cut solver engine using the generic cut callbacks of Python-MIP.

In many applications there are [strong formulations](https://www.researchgate.net/publication/227062257_Strong_formulations_for_mixed_integer_programming_A_survey) that may include an exponential number of constraints. These formulations cannot be direct handled by the MIP Solver: entering all these constraints at once is usually not practical, except for very small instances. In the [Cutting Planes](https://en.wikipedia.org/wiki/Cutting-plane_method) method the LP relaxation is solved and only constraints which are *violated* are inserted. The model is re-optimized and at each iteration a stronger formulation is obtained until no more violated inequalities are found. The problem of discovering which are the missing violated constraints is also an optimization problem (finding *the most* violated inequality) and it is called

the *Separation Problem*.

When using cut callbacks be sure that cuts are used only to *improve* the LP relaxation but not to *define* feasible solutions, which need to be defined by the initial formulation. In other words, the initial model without cuts may be *weak* but needs to be *complete* [1](https://python-mip.readthedocs.io/en/latest/custom.html). In the case of TSP, we can include the weak sub-tour elimination constraints presented in tsp-label in the initial model and then add the stronger sub-tour elimination constraints presented in the previous section as cuts.

# ADVANTAGES

ADVANTAGES OF **ANT COLONY OPTIMIZATION**:

* Can be used in dynamic applications
* Positive Feedback leads to rapid discovery of good solutions
* Distributed computation avoids premature convergence

ADVANTAGES OF **Gomory's Cutting Plane Method**:

* By using the constraints present in the LP from previous lower bound generations, large savings are made in terms of time and memory allocation, as constraints do not need to be stored or LP tableaus regenerated.
* Any valid cutting planes present in the LP from previous lower bound generations may be used to initialize the constraint set when generating the current lower bound. This means that the enumeration process does not need to be started from scratch each time.
* Further, when solving a node, the description of the solution polytope is being improved for all nodes of the search tree at the same time.

## DISADVANTAGES

DISADVANTAGES OF **ANT COLONY OPTIMIZATION**:

* Convergence is guaranteed, but time to convergence uncertain
* Coding is not straightforward

DISADVANTAGES OF **Gomory's Cutting Plane Method**:

* For large problems, the number of variables in the LP relaxations may become sufficiently large, so as to make the LP solution time prohibitive, and thus, reduce the effectiveness of the branch and cut procedure.
* Consequently, reducing the number of 0 - 1 variables through preprocessing is often required. Sparse graph techniques as used by Grotschel and Holland [ll] may also reduce the computing times.
* As would be expected, if violations are detected more often, then the lower bounds usually improve.
* Thus, the method can be updated as further research advances are made. This update may not be possible if another relaxation is used, where typically one characteristic of the underlying structure is exploited to get a lower bound.

### PROGRAME FOR ANT COLONY OPTIMIZATION:

import numpy as np from numpy import inf

d = np.array([[0,10,12,11,14]

,[10,0,13,15,8]

,[12,13,0,9,14]

,[11,15,9,0,16]

,[14,8,14,16,0]])

iteration = 100

n\_ants = 5

n\_citys = 5 m = n\_ants n = n\_citys e = .5

alpha = 1

beta = 2 visibility = 1/d

visibility[visibility == inf ] = 0 pheromne = .1\*np.ones((m,n)) rute = np.ones((m,n+1))

for ite in range(iteration): rute[:,0] = 1

for i in range(m):

temp\_visibility = np.array(visibility) for j in range(n-1):

combine\_feature = np.zeros(5) cum\_prob = np.zeros(5) cur\_loc = int(rute[i,j]-1) temp\_visibility[:,cur\_loc] = 0

p\_feature = np.power(pheromne[cur\_loc,:],beta) v\_feature = np.power(temp\_visibility[cur\_loc,:],alpha)

p\_feature = p\_feature[:,np.newaxis] v\_feature = v\_feature[:,np.newaxis]

combine\_feature = np.multiply(p\_feature,v\_feature) total = np.sum(combine\_feature)

probs = combine\_feature/total cum\_prob = np.cumsum(probs) r = np.random.random\_sample()

city = np.nonzero(cum\_prob>r)[0][0]+1 rute[i,j+1] = city

left = list(set([i for i in range(1,n+1)])-set(rute[i,:-2]))[0] rute[i,-2] = left

rute\_opt = np.array(rute) dist\_cost = np.zeros((m,1)) for i in range(m):

s = 0

for j in range(n-1):

s = s + d[int(rute\_opt[i,j])-1,int(rute\_opt[i,j+1])-1] dist\_cost[i]=s

dist\_min\_loc = np.argmin(dist\_cost) dist\_min\_cost = dist\_cost[dist\_min\_loc] best\_route = rute[dist\_min\_loc,:] pheromne = (1-e)\*pheromne

for i in range(m):

for j in range(n-1): dt = 1/dist\_cost[i]

pheromne[int(rute\_opt[i,j])-1,int(rute\_opt[i,j+1])-1] = pheromne[int(rute\_opt[i,j])-1,int(rute\_opt[i,j+1])-1] + dt

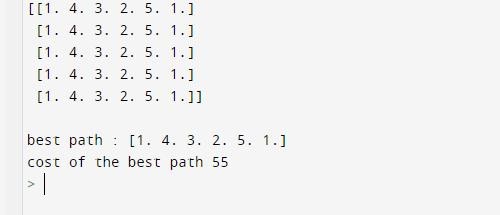
print('route of all the ants at the end :') print(rute\_opt)

print()

print('best path :',best\_route)

print('cost of the best path',int(dist\_min\_cost[0]) + d[int(best\_route[-2])-1,0])

RESULT:



#### PROGRAME FOR Gomory's Cutting Plane Method:

from typing import List, Tuple from random import seed, randint from itertools import product from math import sqrt

import networkx as nx

from mip import Model, xsum, BINARY, minimize, ConstrsGenerator, CutPool

class SubTourCutGenerator(ConstrsGenerator): """Class to generate cutting planes for the TSP""" def init (self, Fl: List[Tuple[int, int]], x\_, V\_):

self.F, self.x, self.V = Fl, x\_, V\_

def generate\_constrs(self, model: Model, depth: int = 0, npass: int = 0): xf, V\_, cp, G = model.translate(self.x), self.V, CutPool(), nx.DiGraph() for (u, v) in [(k, l) for (k, l) in product(V\_, V\_) if k != l and xf[k][l]]:

G.add\_edge(u, v, capacity=xf[u][v].x) for (u, v) in F:

val, (S, NS) = nx.minimum\_cut(G, u, v) if val <= 0.99:

aInS = [(xf[i][j], xf[i][j].x)

for (i, j) in product(V\_, V\_) if i != j and xf[i][j] and i in S and j in S] if sum(f for v, f in aInS) >= (len(S)-1)+1e-4:

cut = xsum(1.0\*v for v, fm in aInS) <= len(S)-1 cp.add(cut)

if len(cp.cuts) > 256: for cut in cp.cuts:

model += cut return

for cut in cp.cuts: model += cut

n = 30

V = set(range(n)) seed(0)

p = [(randint(1, 100), randint(1, 100)) for i in V] Arcs = [(i, j) for (i, j) in product(V, V) if i != j]

c = [[round(sqrt((p[i][0]-p[j][0])\*\*2 + (p[i][1]-p[j][1])\*\*2)) for j in V] for i in V]

model = Model()

x = [[model.add\_var(var\_type=BINARY) for j in V] for i in V]

y = [model.add\_var() for i in V]

model.objective = minimize(xsum(c[i][j]\*x[i][j] for (i, j) in Arcs)) for i in V:

model += xsum(x[i][j] for j in V - {i}) == 1 for i in V:

model += xsum(x[j][i] for j in V - {i}) == 1 for (i, j) in product(V - {0}, V - {0}):

if i != j:

model += y[i] - (n+1)\*x[i][j] >= y[j]-n for (i, j) in Arcs:

model += x[i][j] + x[j][i] <= 1

F, G = [], nx.DiGraph() for (i, j) in Arcs:

G.add\_edge(i, j, weight=c[i][j]) for i in V:

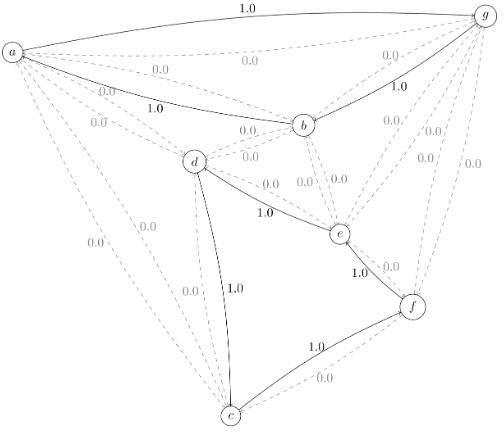
P, D = nx.dijkstra\_predecessor\_and\_distance(G, source=i) DS = list(D.items())

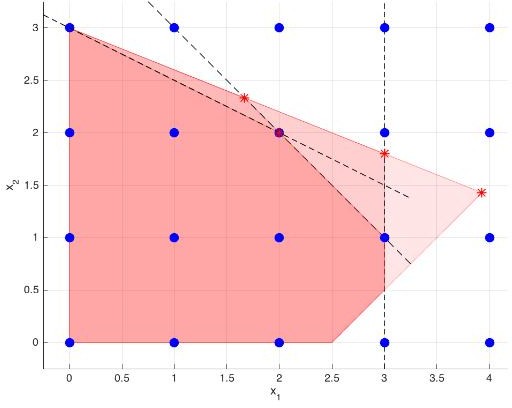
DS.sort(key=lambda x: x[1]) F.append((i, DS[-1][0]))

model.cuts\_generator = SubTourCutGenerator(F, x, V) model.optimize()

print('status: %s route length %g' % (model.status, model.objective\_value))

**RESULT:**





## CONCLUSION

FROM ANT COLONY OPTIMAIZATION WE HAVE LEARNT THAT WE CAN FIND THE OPTIMIUM PATH FROM ONE POINT TO ANY POINT AND WE HAVE ALSO LEARNT THAT WE CAN FIND THE BEST PATH USING THIS METHOD .

FROM GOMORY’S CUT PLANE METHOD WE HAVE LEARNT THAT TO FIND THE THE OPTIMAL SOLUTION BY USING SIMPLEX METHOD WE CAN FIND THE OPTIMAL SOLUTION .IN THIS WE WILL BE FINDING ALL THE POSTIVE VALUES .