Ant Colony versus Genetic Algorithm based on Travelling Salesman Problem

Article *in* International Journal of Computer Applications in Technology · June 2011

2 authors, including:

Mohammed Ahmed Alhanjouri
Islamic University of Gaza
31 PUBLICATIONS **89** CITATIONS

SEE PROFILE

ISSN:2229-6093

Ant Colony versus Genetic Algorithm based on Travelling Salesman Problem

Mohammed Alhanjouri
Islamic University of Gaza
mhanjouri@iugaza.edu.ps

Belal Alfarra
Islamic University of Gaza
fbelal@zim.iugaza.edu

Abstract

The travelling salesman problem (TSP) is a nondeterministic Polynomial hard problem in combinatorial optimization studied in operations research and theoretical computer science. And to solve this problem we used two popular meta-heuristics techniques that used for optimization tasks; the first one is Ant Colony Optimization (ACO), and the second is Genetic Algorithm (GA). In this work, we try to apply both techniques to solve TSP by using the same dataset and compare between them to determine the best one for travelling salesman problem. for Ant Colony Optimization, we studied the effect of some parameters on the produced results, these parameters as: number of used Ants, evaporation, and number of iterations. On the other hand, we studied the chromosome population, crossover probability, and mutation probability parameters that effect on the Genetic Algorithm results. The comparison between Genetic Algorithm and Ant Colony Optimization is accomplished to state the better one for travelling salesman problem.

1. Introduction

Optimization is one of the most important tasks of engineers, which the engineer asked to design new, better, more efficient and less expensive systems as well as to devise plans and procedures for the improved operation of existing systems in both industrial and the scientific world.

The travelling salesman problem (TSP) is a nondeterministic Polynomial hard problem in combinatorial optimization studied in operations research and theoretical computer science.

The problem was described as: there are cities and given distances between the, a travelling salesman has to visit all of them, but he does not want to spend much time on travelling, therefore we need to find the sequence of cities to minimize the traveled distance.

The problem was first formulated as a mathematical problem in 1930 and is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods. Even though the problem is computationally difficult, a large number of heuristics and exact methods are known, so that some instances with tens of thousands of cities can be solved.

The TSP is represented in numerous transportation and logistics applications such as:

- arranging routes for school buses to pick up children in a school district,
- delivering meals to home-bound people,
- scheduling stacker cranes in a warehouse,
- Planning truck routes to pick up parcel post and many others.
- Planning, logistics, and the manufacture of microchips.
- A classic example of the TSP is the scheduling of a machine to drill holes in a circuit board.

Slightly modified, it appears as a sub-problem in many areas, such as DNA sequencing. In these applications, the concept *city* represents, for example, customers, soldering points, or DNA fragments, and the concept *distance* represents travelling times or cost, or a similarity measure between DNA fragments. In many applications, additional constraints such as limited resources or time windows make the problem considerably harder.

In the theory of computational complexity, the decision version of the TSP (where, given a length L, the task is to decide whether any tour is shorter than L) belongs to the class of NP-complete problems. Thus, it is likely that the worst case running time for any algorithm for the TSP increases exponentially with the number of cities.

2. Ant Colony Optimization

Ant colony optimization (ACO) [6] is one of the most popular meta-heuristics used for combinatorial optimization (CO) in which an optimal solution is sought over a discrete search space. The well-known CO's example is the traveling salesman problem (TSP) [1] where the search-space of candidate solutions grows more than exponentially as the size of the problem increase, which makes an exhaustive search for optimal solution infeasible.

ISSN:2229-6093

The first ACO algorithm –Ant System (AS)- has been introduced by Marco Dorigo in the early 1990's [2,3,4], and since then several improvement of the AS have been devised (Gambardella & Dorigo, 1995[5]; Stützle & Hoos, 1997[6]). The ACO algorithm is based on a computational paradigm inspired by real ant colonies and the way they function. The underlying idea was to use several constructive computational agents (simulating real ants) [7].

Ant's behavior is governed by the goal of colony survival rather than being focused on the survival of individuals. The behavior that provided the inspiration for ACO is the ants' foraging behavior (see figure 1), and in particular, how ants can find shortest paths between food sources and their nest. When searching for food, ants initially explore the area surrounding their nest in a random manner. While moving, ants leave a chemical pheromone trail on the ground. Ants can smell pheromone.

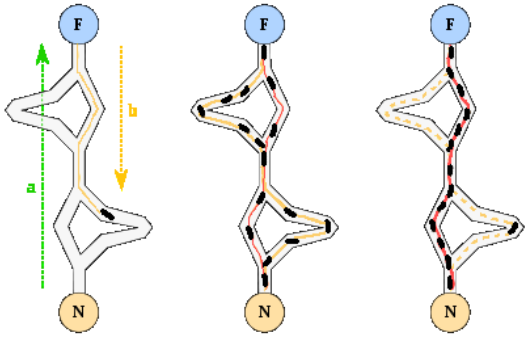


Figure 1: Ants use pheromone as indirect communication to build best tour

When choosing their way, they tend to choose, in probability, paths marked by strong pheromone concentrations. As soon as an ant finds a food source, it evaluates the quantity and the quality of the food and carries some of it back to the nest. During the return trip, the quantity of pheromone that an ant leaves on the ground may depend on the quantity and quality of the food. The pheromone trails will guide other ants to the food source. It has been shown in [8] that the indirect communication between the ants via pheromone trails enables them to find shortest paths between their nest and food sources. In this paper I'll view the relations between ACO parameters and how the number of iterations is increased as the number of ants decreased or as the evaporation coefficient increased.

2.1. Ant System (AS)

The first ACO algorithm was called the Ant system [5], the objective of AS is to solve the travelling salesman problem (TSP). A travelling salesman is required to pass through a number of cities, each city is visited once and he needs to find the shortest closed path – tour- that link all cities. Thus, we have undirected graph consists of V nodes -or cities- linked by undirected E edges G = (V, E) the edge weights represent the distances between the cities.

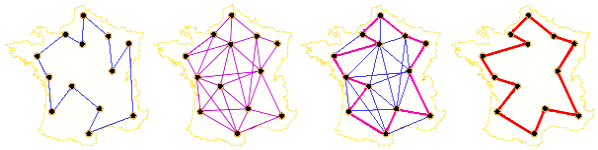


Figure 2: Undirected graph show nodes and the edges between the nods, the figure show the four stages of ACO to reach shortest closed path.

As shown in figure 2, the search space S consists of all tours in G. The objective function value f(s) of a tour s ∈ S is defined as the sum of the edge-weights of the edges that are in s. The TSP can be modeled in many different ways as a discrete optimization problem. Concerning the AS approach, the edges of the given TSP graph can be considered solution components, ant introduce a pheromone value $\tau_{i,j}$ for the edge ei,j. The general algorithm is based on a set of ants, each making one of the possible tours or round-trips along the cities. Each tour considered as one solution s of search space S, and the sum of the edges-weights is the objective function f(s). Now we search for the best tour s at which we have smallest f(s). The following steps describe how each ant constructs a solution s:

1- Each ant chose randomly one city as start node.
2- The ant starts building the tour by moving from one city to another unvisited city.
3- The traversed edge is chose by probability P(ei,j).

$$\boldsymbol{p}(e_{i,j}) = \frac{\tau_{i,j}}{\sum_{\{k\in\{1,\dots,|V|\}|v_k\notin T\}}\tau_{i,k}},\ \forall j \in \{1,\dots,|V|\}, v_i \notin \boldsymbol{T} \quad (1)$$

4- The traversed edge is added to the solution being constructed.
5- When all cites are visited the ant move to the start node.
6- Having completed its journey, the ant deposits more pheromones on all edges it traversed. Deposited pheromones is:

$$\tau_{i,j} \leftarrow \tau_{i,j} + \frac{Q}{f(s)} \quad (2)$$

7- After each iteration, trails of pheromones evaporate done as follows:

$$\boldsymbol{\tau_{i,j}} \leftarrow (\mathbf{1}-\boldsymbol{\rho})\boldsymbol{\tau_{i,j}}, \quad \forall \boldsymbol{\tau_{i,j}} \in \mathrm{T} \quad (3)$$

The previous steps is used to construct one tour, these steps can be repeated more and more to obtain the optimum solution. In each tour the more intense the pheromone trail laid out on an edge between two cities, the greater the probability that that edge will be chosen.

2.2. Methodology

There are several extensions and improvements of the original AS algorithm were introduced. All of which covered by the definition of the ACO meta-heuristic. But in general the following algorithm is used:

2.2.1. ACO Algorithm

Algorithm 1: Ant colony optimization (ACO)

init pheromone $\tau i = const$ *for each component* ci;
while termination conditions not met do
  for all ants i: $construct\_solution(i)$;
    for all ants i: $global\_pheromone\_update(i)$;
    for all pheromones i: $evaporate$: $\tau i = (1-\rho)\cdot \tau i$;
end loop;

Algorithm 2. construct_solution(i);

init $s = \{\}$;
while s *is not a solution*:
    choose cj *with probability* $= p(ci \mid s)$
    expand s *by* cj;
end loop;

In most ACO algorithms the respective probabilities—also called the transition probabilities—are defined as follows:

$$\boldsymbol{p}(C_i \mid S) = \frac{[\boldsymbol{\tau_i}]^{\boldsymbol{\alpha}}.[\boldsymbol{\eta}(\boldsymbol{C_i})]^{\boldsymbol{\beta}}}{\sum_{C_j \in \boldsymbol{N}(\boldsymbol{s})} [\boldsymbol{\tau_j}]^{\boldsymbol{\alpha}}.[\boldsymbol{\eta}(\boldsymbol{C_j})]^{\boldsymbol{\beta}}}, \quad \forall C_i \in \boldsymbol{N}(\boldsymbol{s}) \qquad (4)$$

Where η is an optional weighting function commonly called the heuristic information, that it sometimes depends on the current sequence s, the exponents α and β are positive parameters whose values determine the relation between pheromone information and heuristic. N(s) is the set of all feasible solution component. For TSP example, we chose not to use any weighting function η, and we have set α to 1.

Algorithm 3. global_pheromone_update(i);
for all cj *in the solution* s:
    increase pheromone: $\tau j = \tau j + const / f(s)$;

2.3. ACO Parameters

As mentioned before ACO algorithm is meta-heuristic that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. In general meta-heuristic doesn't guarantee an optimal solution is ever found. The most asked question is: what is the best result can we obtain in less iteration with minimum cost and time? Or when to terminate?. This require to have good estimation for parameters used with ACO algorithm, like α, β, the number of ants (M), the maximum number of iterations, and the most important parameters: The pheromone trail decay coefficient (p) and pheromone amount ($\Delta\boldsymbol{\tau}(\mathbf{t})$) which have an important impact on the performance of ACO. The selection of the parameters depends on the problem needed to be optimized. In this paper the deposited pheromone ($\Delta\boldsymbol{\tau}(\mathbf{t})$) calculated using Eq. (2) with Q=1. And p which simulates the pheromone evaporation becomes more important for more complex problems.

In this work we implement the algorithm and, by trial and error, try to determine the best number of iteration required to reach the optimal tour with different value of M, and p. And repeat this method for different data sets.

3. Introduction to genetic Algorithm

Genetic algorithms are a part of evolutionary computing technique, which is a rapidly growing area of artificial intelligence.

Genetic algorithms are inspired by Darwin's theory about evolution. Simply said, solution to a problem solved by genetic algorithms is evolved.

I.Rechenberg introduced the idea of evolutionary computing in the 1960s in his work "Evolution strategies" (Evolutions strategy in original). Other researchers then developed his idea. Genetic Algorithms (Gas) wre invented by John Holland and developed by him and his students and colleagues.[1]

In 1992 john koza has used genetic algoritm to evolve programs to perform certain tasks. He called his method "genetic programming" (GP) [5], LISP programs were used; because programs in this language can be expressed in the form of a "parse tree", which is the object the GA works on.

3.1. Basic Description of GA

Genetic algorithm is started with a set of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one.

Solutions which are selected to form new solutions (offspring) are selected according to their fitness; the more suitable they are the more chances they have to reproduce. This is repeated until some condition (for example number of populations or improvement of the best solution) is satisfied.
It is well known that problem solving can be often expressed as looking for the extreme of a function. This is exactly the case with the following problem: some function is given and GA tries to find the minimum of the function.

3.2. Outline of the basic Genetic Algorithm

1- [Start] Generate random population of n chromosomes (suitable solutions for the problem).
2- [Fitness] Evaluate the fitness f(x) of each chromosome x in the population.
3- [New population] Create a new population by repeating the following steps until the new population is complete.
   a. [Selection] Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
   b. [Crossover] With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
   c. [Mutation] With a mutation probability mutate new offspring at each locus (position in chromosome)
   d. [Accepting] Place new offspring in a new population.
4- [Replace] use new generated population for a further run of algorithm
5- [Test] If the end condition is satisfied, stop, and return the best solution in current population
6- [Loop] Go to step 2

The above outline of GA is very general. There are many things that can be implemented differently in various problems.
First question to be answered is how to create chromosomes and what types of encoding to choose. The next question is how to select parents (in hope that the better parents will produce better offspring), that making a new population only by new offspring can cause loss of the best chromosome from the last population. This is true, so a method called Elitism is often used. This means that at least one best solution is copied without changes to a new population, so the best solution found can survive to the end of the run.

As we can see from the genetic algorithm outline, the crossover and mutation are the most important part of the genetic algorithm. The performance is influenced mainly by these two operators. Then basic parameters of GA are: Crossover Probability, Mutation Probability and Population size.
Crossover probability says how often crossover will be performed. If there is no crossover, offspring is an exact copy of parents.
If there is a crossover, offspring is made from parts of parent's chromosome. If crossover probability is 100% then all offspring is made by crossover. If it is 0 %, whole new generation is made from exact copies of chromosomes from old population (but this does not mean that the new generation is the same).
Crossover is made in hope that new chromosomes will have good parts than old chromosomes and they may be better. However it is good to leave some parts of the population to survive to the next generation.
Mutation probability shows how often parts of chromosome will be mutated. If there is no mutation, offspring is taken after crossover without any change. If mutation is performed, part of chromosome is changed. If mutation probability is 100%, whole chromosome is changed, if it is 0%, nothing is changed. Mutation is made to prevent falling GA into local extreme, but it should not occur very often, because in this case GA will in fact change to random search.
Population size says how many chromosomes are in population (in one generation). If there are too few chromosomes, GA has a few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if there are too many chromosomes, GA slows down. Research shows that after some limit (which depends mainly on encoding and the problem) it is not useful to increase population size, because it does not make solving the problem faster.
As we already know from the GA outline, chromosomes are selected from the population to be parents to crossover. The problem is how to select these chromosomes. According to Darwin's evolution theory the best ones should survive and create new offspring. There are many methods how to select the best chromosomes; for example, there is Roulette wheel selection, Boltzman selection, Tournament selection, Rank selection, Steady state selection, or some others.
Encoding of chromosomes is one of the problems, when we are starting to solve problem with GA. Encoding depends greatly on the problem; there are many encoding style like as: Binary encoding,

Permutation encoding, Value encoding, or Tree encoding.

First, we need to decide how to represent a route of the salesman. The most natural way of representing a route is the *path representation.* Each city is given an alphabetic or numerical name, the route through the cities is represented as a chromosome, and appropriate genetic operators are used to create new routes. For travelling salesman problem, we use Permutation encoding to ordering the problem. Every chromosome is a string of numbers, which represents the numbers in a sequence.

Permutation encoding is only useful for ordering problems. Even for these problems for some types of crossover and mutation corrections must be made to leave the chromosome constant (i.e. have real sequence in it). The travelling salesman problem (TSP) would be a good example of permutation encoding. Chromosomes says order of cities, in which salesman will visit them.

The crossover and mutation operators depend on type of encoding and also on the problem. For our encoding and problem, we use single point crossover; as shown in Figure (3), the permutation is copied from the first parent until we reach this point, then the second parent is scanned and if the number is not yet in the offspring it is added. There are more ways to produce the rest after crossover point (*)

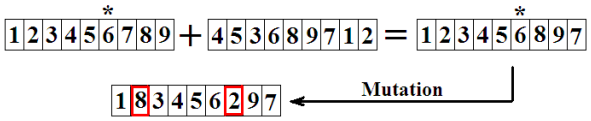


Figure 3. Crossover and mutation for permutation encoding

4. Experimental results of Ant Colony Optimization

We perform several experiments on one synthetic datasets of 14 dimension -14 nodes- and on three real-life datasets [9] with different dimensions (16, 22 and 29 dimension), dataset's characteristics are illustrated in Table 1. All our experiments use MATLAB version 7.6 R2008a on windows 7 on Intel core(TM) 2 Duo CPU T8300 2.40 GHz computer with 4 gigabytes of memory.

Table 1: Datasets characteristics

NAME	TYPE	COMMENT	DIMENSION	Best f(s)
ulysses16	TSP	Odyssey of Ulysses (Groetschel/Padberg)	16	73.9876
ulysses22	TSP	Odyssey of Ulysses (Groetschel/Padberg)	22	56.2237
bays29	TSP	COMMENT: 29 cities in Bavaria, street distances (Groetschel,Juenger,Reinelt)	29	____
Node 14	synthetic	8 2, 0 4, -1 6, 2 -1, 4 -2, 6 0.5, 3 0, 10 3.7, 2.5, 1.8, -5 1, 7 0, 9 4, 11 3, 13 2	14	45.562

Figures 4, 5, 6 and 7 shows the shortest path can be obtained when we apply the algorithm 1 described in section 3.1 on the ulysses16, ulysses22, bays29, and node14 datasets respectively.

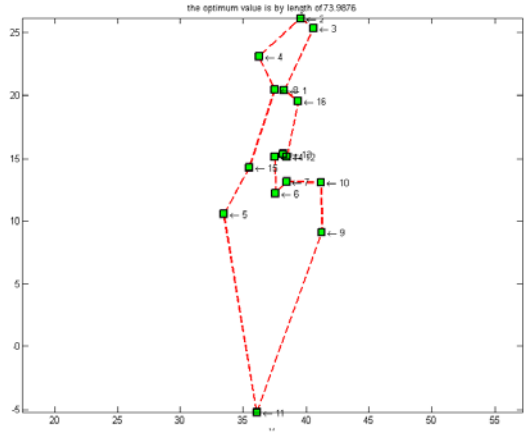


Figure 4. The best tour for ulysses16.tsp obtained by ACO algorithm.

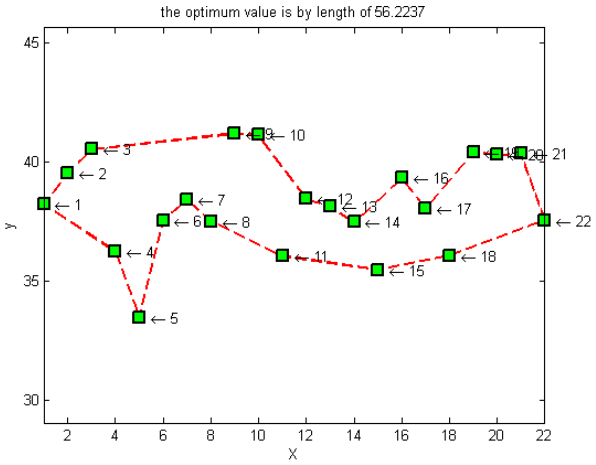


Figure 5. The best tour for ulysses22.tsp obtained by ACO algorithm.

Table 2 shows the best number of iterations required to have the optimal solution at which we have minimum value of objective function. the table uses '_' to mean that the algorithm converged to suboptimal solutions, the maximum iteration is 1400 iteration, this means that, for each set of parameters the algorithm is run in the range of 1 to 1400 iterations. If the algorithm converged to the optimal value with N iterations, we repeat the algorithm more and more with iterations N to be ensured that N is the best number of iterations at parameters p and M to have minimum f(s). Three different values were used (0.1, 0.5, and 0.9) for evaporation parameter p with three different values (100, 250, and 500) for M which represent the number of ants "agents". With these parameters values and by using and apply the trial-and-error (by matlab code) we obtain the results shown in table. It is clear that, if the evaporation parameter p is small then the optimal solution is obtained with the smallest number of iteration. Also, the number of iterations proportional with number of ants that used to solve the problem.

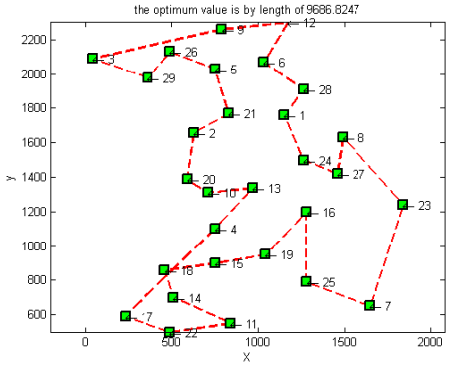


Figure 6. The best tour for bays29.tsp obtained by ACO algorithm.

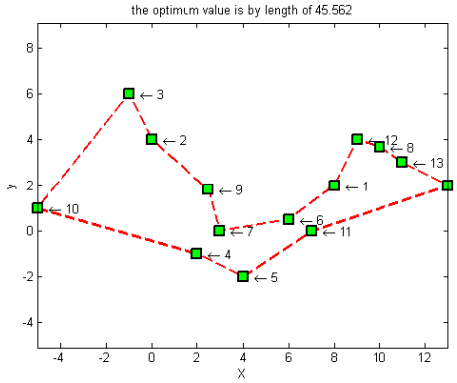


Figure 7. The best tour for node14.tsp obtained by ACO algorithm.

Table 2: Best number of iteration required by the algorithm to converge optimal f(s),

Dataset	P=.9	P=.5	P=.1	P=.9	P=.5	P=.1	P=.9	P=.5	P=.1
	M=100			M=250			M=500		
ulysses 16.tsp	_	950	130	_	_	500	_	_	200
ulysses 22.tsp	_	_	600	_	_	800	_	_	1100
bays29 .tsp	_	_	1000	_	_	_	_	_	_
Node1 4.tsp	1400	820	100	600	250	35	450	200	20

An '_' means with maximum iteration 1400 the algorithm converge to suboptimal value, M: number of ants, p: evaporation coefficient.

By using large number of iterations (up to 3000 iterations) the table is competed but with extreme spent time (in some cases up to 40 min.), and we forced some problems to determine the suitable values of parameters P, M, and No. of iterations, that effects on the results.

5. Experimental results of Genetic Algorithms

By use MATLAB version 7.6 R2008a on windows 7 on Intel core(TM) 2 Duo CPU T8300 2.40 GHz computer with 4 gigabytes of memory, we perform several experiment on GA that applied on synthetic datasets for cities locations.

Figure 8 shows the shortest path of 20 cities by GA with 300 of chromosome population, Crossover

probability is 0.8, and mutation probability is 0.1. And 90 iterations are needed to achieve the best tour.

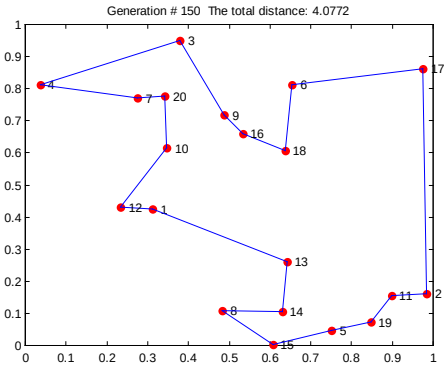
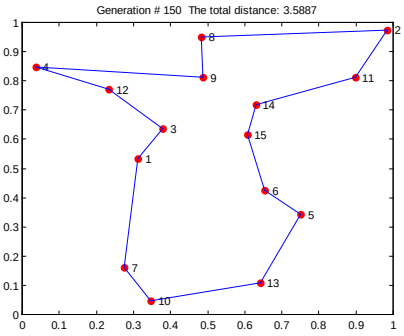
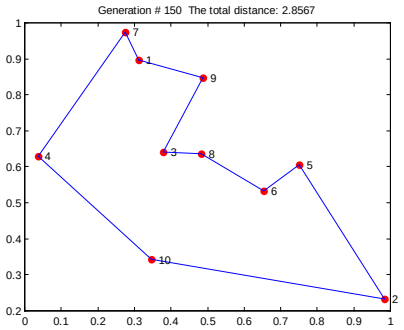


Figure 8 the best tour between 20 cities by GA

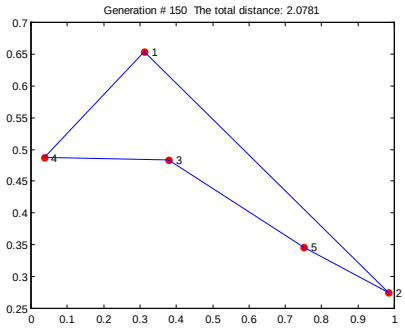
As shown in Figure 9, the best tour between the cities in different cases: (a) for 15 cities, (b) for 10 cities, and (c) for 5 cities. Experimentally the results obtained after 70, 60, and 48 iterations respectively, at the same crossover, and mutation probabilities.



(a)



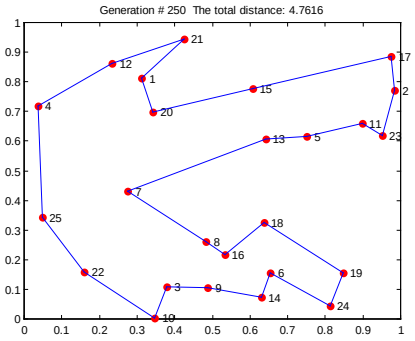
(b)



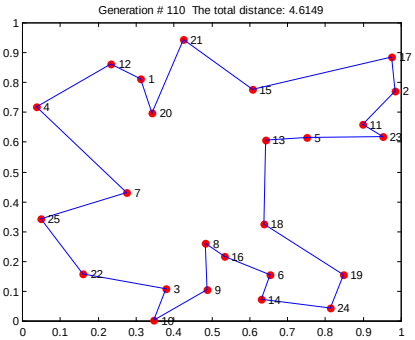
(c)

Figure 9 the best tour for different case number of cities

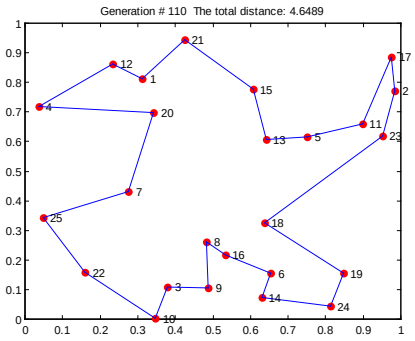
In case of 25 cities or more, we have challenges to determine GA parameters because the results become very sensitive for any variation with it. Figure 10 illustrate the results for different cases of Mutation probability as example, in (a, b, and c) the best tour by GA with mutation probability of 0.01, 0.009, and 0.008 respectively.



(a)



(b)



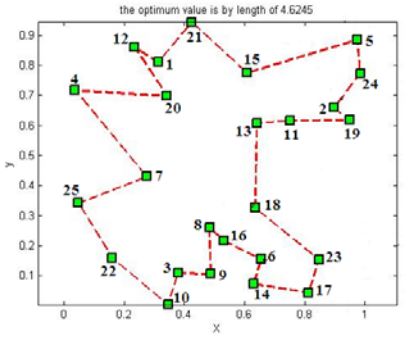
(c)

Figure 10 the best tour for 25 cities by GA with different Mutation Probabilities (0.01, 0.009, and 0.008)

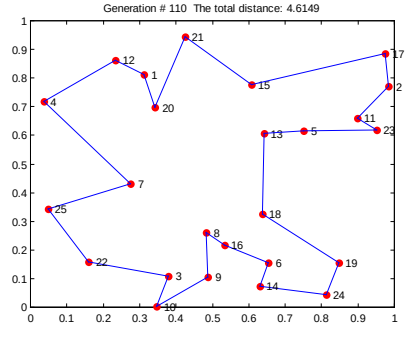
6. Comparison between GA and ACO for TSP

Both techniques (Genetic Algorithm and Ant Colony Optimization) are used to solve travelling salesman problem with high acceptable performance, therefore we here to compare between them and determine when we can use one as better than other.

As in figure 11, we can see the best tour and distance between 25 cities for the same data by using both GA and ACO.



(a)



(b)

Figure 11 the best tour of 25 cities using both (a) ACO and (b) GA.

For ACO, the result obtain by using 2500 iterations, 250 ants for each iteration, and 0.9 as an evaporation coefficient to produce the best distance as 4.6245. While for GA, the best distance is 4.6149 by using crossover and mutation probability is 0.75 and 0.009 respectively, and after 110 iterations with 200 chromosomes. But the first advantage for GA is the small spent time against to the large required time by ACO.

7. Conclusions

As shown by the experiment, it is difficult to select the best parameter for ACO, but we can observe the dependency of the number of iterations on both the evaporation coefficient p and the number of ants M. that if p=0 that have no evaporation, the algorithm does not converge. But when p is large enough (p=0.9), the algorithm often converged to suboptimal solutions for complex problem. This paper is the first step on determining best number iteration for ACO to have the optimal solution. It is necessary to evaluate the relation between costs, alpha, and beta and how these parameters effect on best number of iterations and evaporations coefficient.

Also for GA, we need to select the best value for chromosome population, crossover, and mutation probabilities. But still at this time the GA is better than ACO for TSP.

8. References

[1] Lawler E, Lenstra JK, Rinnooy Kan AHG, Shmoys DB. The travelling Salesman problem. New York: John Wiley & Sons; 1985.

[2] Dorigo M, Optimization, learning and natural algorithms. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.

[3] Dorigo M, Maniezzo V, Colorni A, Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991.

[4] Dorigo M, Maniezzo V, Colorni A. Ant System: Optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybernet Part B 1996;26(1):29–41.

[5] M. Dorigo et L.M. Gambardella, *Ant Colony System : A Cooperative Learning Approach to the Traveling Salesman Problem*, IEEE Transactions on Evolutionary Computation, volume 1, numéro 1, pages 53-66, 1997.

[6] Dorigo M, Stützle T. Ant Colony optimization. Cambridge, MA: MIT Press; 2004.

[7]S. Camazine and J.L. Deneubourg. Self-organization in biological systems. Princeton University Press, Princeton, NJ, 2001.

[8] Deneubourg J-L, Aron S, Goss S, Pasteels J-M. The self-organizing exploratory pattern of the Argentine ant. J Insect Behaviour 1990;3:159–68.

[9] MP-TESTDATA - The TSPLIB Symmetric Traveling Salesman Problem Instances. available at (http://elib.zib.de/pub/mptestdata/tsp/tsplib/tsp/index.html) , Last update: June 1, 1995.