Volume 7, Issue 3, March 2017 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Optimization Techniques for Solving Travelling Salesman Problem

Mohammed Alhanjouri
Department of Computer Engineering, Islamic University of Gaza,
Palestinian

Abstract*— In the traveling salesman problem (TSP) we wish to find a tour of all nodes in a weighted graph so that the total weight is minimized. The traveling salesman problem is NP-hard but has many real world applications so a good solution would be useful. In this paper, we present several modern optimization techniques to find the shortest tour through all cities (nodes). Genetic Algorithm (GA), Simulated Annealing (SA), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Bacteria Foraging Optimization (BFO), and Bee Colony Optimization (BCO) are applied on several datasets of TSP with different number of cities and different representation: distances between cities, or coordinates of cities. Each optimization technique has unique behaviors which survives it against other techniques. In this paper, the results and comparative study will present for each dataset to calculate the minimum distance and plat the resultant path.

Keywords*— Genetic Algorithm, Simulated Annealing, Particle Swarm Optimization, Ant Colony Optimization, Bacteria Foraging Optimization, Bee Colony Optimization, traveling salesman problem.

I. INTRODUCTION

The idea of the traveling salesman problem (TSP) is to find a tour of a given number of cities, visiting each city exactly once and returning to the starting city where the length of this tour is minimized.

The first instance of the traveling salesman problem was from Euler in 1759 whose problem was to move a knight to every position on a chess board exactly once [1].

The traveling salesman first gained fame in a book written by German salesman BF Voigt in 1832 on how to be a successful traveling salesman [1]. He mentions the TSP, although not by that name, by suggesting that to cover as many locations as possible without visiting any location twice is the most important aspect of the scheduling of a tour. The origins of the TSP in mathematics are not really known all we know for certain is that it happened around 1931.

The standard or symmetric traveling salesman problem can be stated mathematically as follows:

Given a weighted graph G = (V, E) where the weight cij on the edge between nodes i and j is a non-negative value, find the tour of all nodes that has the minimum total cost or distance.

Currently the only known method guaranteed to optimally solve the traveling salesman problem of any size, is by enumerating each possible tour and searching for the tour with smallest cost. Each possible tour is a permutation of n, where n is the number of cities, so therefore the number of tours is (n!). When n gets large, it becomes impossible to find the cost of every tour in polynomial time. Therefore, in this work we will examine some of modern optimization techniques to solve the problem within feasible time..

II. BACKGROUND

Mathematically, A Travelling Salesman Problem (TSP) can be defined on a graph G = (V, E) where V is a vector of cities {1,2,….,n} where n is the number of cities .Then, E = {(i,j): i,j ∈ V, i≠j } is an edge set. A cost matrix C = (cij) is defined on E. In particular, this is the case of planar problems for which the vertices are points Pi = (Xi, Yi) and Pj = (Xj, Yj). So, cij = $\sqrt{(xi - xj)2 + (yi - yj)2}$ is distance between these points. So, the problem could be formalized as: Minimize f(x) = $\sum_{i,j}^{n} cij$, to satisfy our goal to find the minimum total distance of each chromosome was populated, where visiting each city exactly once and returning back to the starting city. That computed f(x) was presented a fitness function in used techniques [2].

Many different methods of optimization have been used to try to solve the TSP. Homaifar [3] states that "one approach which would certainly find the optimal solution of any TSP is the application of exhaustive enumeration and evaluation. The procedure consists of generating all possible tours and evaluating their corresponding tour length. The tour with the smallest length is selected as the best, which is guaranteed to be optimal. If we could identify and evaluate one tour per nanosecond (or one billion tours per second), it would require almost ten million years (number of possible tours = 3.2×1023) to evaluate all of the tours in a 25 city TSP."

Obviously we need to find an algorithm that will give us a solution in a shorter amount of time. As we said before, the traveling salesman problem is NP-hard so there is no known algorithm that will solve it in polynomial time. We will probably have to sacrifice optimality in order to get a good answer in a shorter time. Many algorithms have been tried for the traveling salesman problem.

Greedy Algorithms [4] are a method of finding a feasible solution to the traveling salesman problem. The algorithm creates a list of all edges in the graph and then orders them from smallest cost to largest cost. It then chooses the edges with smallest cost first, providing they do not create a cycle. The greedy algorithm gives feasible solutions however they are not always good.

The Nearest Neighbor [4] algorithm is similar to the greedy algorithm in its simple approach. We arbitrarily choose a starting city and then travel to the city closest to it that does not create cycle. We continue to do this until all cities are in the tour. This algorithm also does not always give good solutions because often the last edge added to the tour (that is, the edge en1 where n is the number of cities) can be quite large.

A minimum spanning tree [4],[5] is a set of n−1 edges (where again n is the number of cities) that connect all cities so that the sum of all the edges used is minimized. Once we have found a minimum spanning tree for our graph we can create a tour by treating the edges in our spanning tree as bidirectional edges. We then start from a city that is only connected to one other city (this is known as a 'leaf' city) and continue following untraversed edges to new cities. If there is no untraversed edge we go back along the previous edge. We continue to do this until we return to the starting city. This will give us an upper bound for the optimal traveling salesman tour. Note, however, that we will visit some cities more than once. We are able to fix this if whenever we need to traverse back to a city we have already been to, we instead go to the next unvisited city. When all cities have been visited we go directly back to the starting city.

TSP is a minimization problem; we consider f(x) as a fitness function, where f(x) calculates cost (or value) of the tour between cities (nodes).

III. METHODOLOGY

In this section, we expose the different optimization techniques that will be used to solve TSP.

A. Genetic Algorithm (GA)

Genetic programming [6-8] is an automated method for solving problems. Specifically, genetic programming progressively breeds a population of computer programs over a series of generations. Genetic programming is a probabilistic algorithm that searches the space of compositions of the available functions and terminals under the guidance of a fitness measure. Genetic programming starts with a primordial ooze of thousands of randomly created computer programs and uses the Darwinian principle of natural selection, recombination (crossover), mutation, gene duplication, gene deletion, and certain mechanisms of developmental biology to breed an improved population over a series of many generations.

Genetic programming breeds computer programs to solve problems by executing the following three steps:

1) Generate an initial population of compositions of the functions and terminals of the problem.
2) Iteratively perform the following substeps (referred to herein as a generation) on the population of programs until the termination criterion has been satisfied:
   a) Execute each program in the population and assign a fitness value using the fitness measure.
   b) Create a new population of programs by applying the following operations. The operations are applied to program selected from the population with a probability based on fitness (with reselection allowed).

Reproduction: Copy the selected program to the new population. The reproduction process can be subdi-vided into two subprocesses: Fitness Evaluation and Selection. The fitness function is what drives the evolutionary process and its purpose is to determine how well a string (individual) solves the problem, al-lowing for the assessment of the relative performance of each population member.

Crossover: Create a new offspring program for the new population by recombining randomly chosen parts of two selected programs. Reproduction may proceed in three steps as follows: 1) two newly re-produced strings are randomly selected from a Mating Pool; 2) a number of crossover positions along each string are uniformly selected at random and 3) two new strings are created and copied to the next generation by swapping string characters between the crossover positions defined before.

Mutation: Create one new offspring program for the new population by randomly mutating a randomly chosen part of the selected program.

Architecture-altering operations: Select an architecture-altering operation from the available repertoire of such operations and create one new offspring pro-gram for the new population by applying the selected architecture-altering operation to the selected program.

3) Designate the individual program that is identified by result designation (e.g., the best-so far individual) as the result of the run of genetic programming. This result may be a solution (or an approximate solution) to the problem. The specification of the designed GA technique depends on used database; population size will selected from 100 and increasing monotonically while find solution, crossover rate around 0.5 – 0.7, Mutation rate in range 0.05 – 0.09, Chromosome Length depends on number of cities in database.

Figure 1 shows the flowchart of the parameter optimizing procedure using GA. For details of genetic operators and each block in the flowchart, one may consult literature [9].

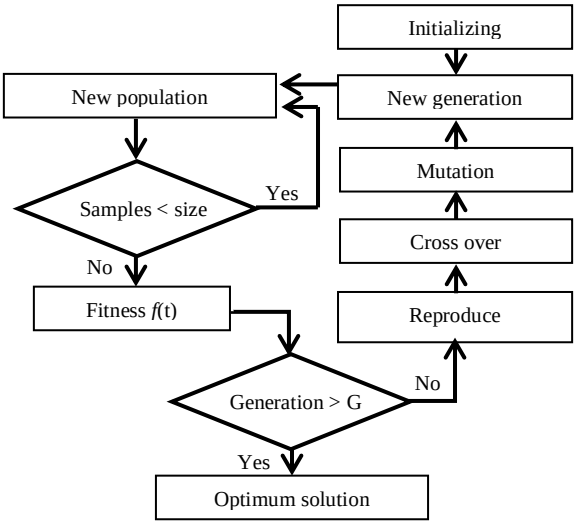


Fig.1. The optimization flowchart of GA technique.

B. Ant Colony Optimization

Ant colony optimization (ACO) 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) [10] 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.

The first ACO algorithm, Ant System (AS), has been introduced by Marco Dorigo in the early 1990's [11-13], and since then several improvement of the AS have been devised (Gambardella & Dorigo, 1995[14]; Stützle & Hoos, 1997[15]). 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) [16]. 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 2), 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.

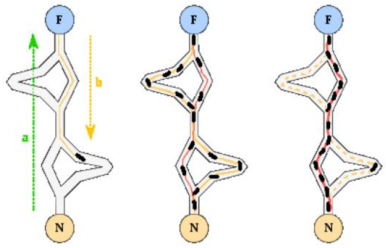


Fig. 2 Ants use pheromone as indirect communication to build best tour

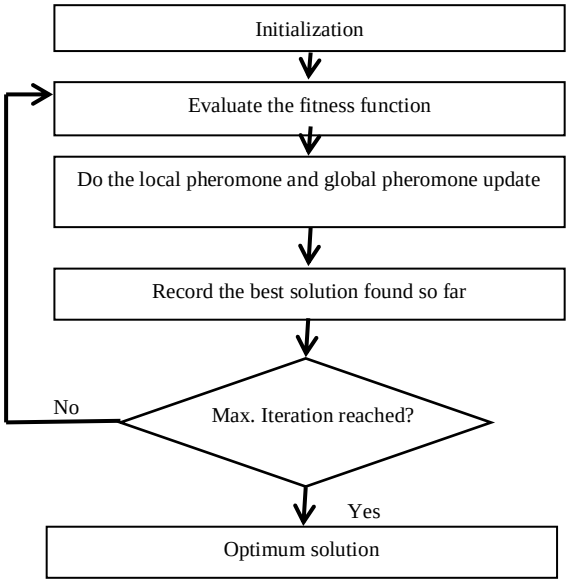
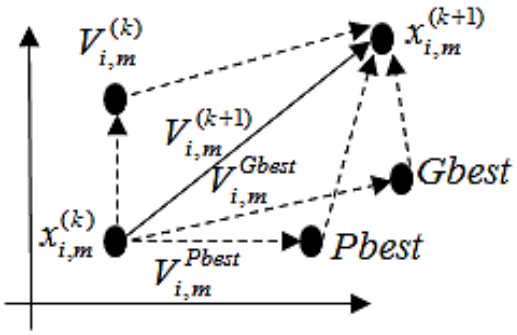


Fig. 3. The optimization flowchart of ACO

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 [17] that the indirect communication between the ants via pheromone trails enables them to find shortest paths between their nest and food sources. The flowchart of ACO is shown in figure (3)

C. Particle Swarm Optimization

PSO is a population based optimization method first proposed by Eberhart and Colleagues [18-20]. Some of the attractive features of PSO include the ease of implementation and the fact that no gradient information is required. It can be used to solve a wide array of different optimization problems. Like evolutionary algorithms, PSO technique conducts search using a population of particles, corresponding to individuals. Each particle represents a candidate solution to the problem at hand. In a PSO system, particles change their positions by flying around in a multidimensional search space until computational limitations are exceeded. Concept of modification of a searching point by PSO is shown in Figure 4.



N = Number of particles in the group, d = dimension, $x_{i,m}^{(k)}$ Current position, $x_{i,m}^{(k+1)}$ Modified position
$V_{i,m}^{(k)}$ Current velocity, $V_{i,m}^{(k+1)}$ Modified velocity, $V_{i,m}^{Pbest}$ Velocity based on Pbesti, $V_{i,m}^{Gbest}$ Velocity based on Gbesti
w = Inertia weight factor, c1,c2 = Acceleration constant, rand() Random number between 0 and 1.
Pbesti = Best previous position of the ith particle, gbest = Best particle among all the particles in the population.

Fig. 4 Concept of modification of a searching point by PSO

The PSO technique is an evolutionary computation technique, but it differs from other well-known evolutionary computation algorithms such as the genetic algorithms. Although a population is used for searching the search space, there are no operators inspired by the human DNA procedures applied on the population. Instead, in PSO, the population dynamics simulates a 'bird flock's' behaviour, where social sharing of information takes place and individuals can profit from the discoveries and previous experience of all the other companions during the search for food. Thus, each companion, called particle, in the population, which is called swarm, is assumed to 'fly' over the search space in order to find promising regions of the landscape. For example, in the minimization case, such regions possess lower function values than other, visited previously. In this context, each particle is treated as a point in a d-dimensional space, which adjusts its own 'flying' according to its flying experience as well as the flying experience of other particles (companions). In PSO, a particle is defined as a moving point in hyperspace. For each particle, at the current time step, a record is kept of the position, velocity, and the best position found in the search space so far.

The assumption is a basic concept of PSO. In the PSO algorithm, instead of using evolutionary operators such as mutation and crossover, to manipulate algorithms, for a variable optimization problem, a flock of particles are put into the d-dimensional search space with randomly chosen velocities and positions knowing their best values so far (Pbest) and the position in the d-dimensional space.

The velocity of each particle, adjusted according to its own flying experience and the other particle's flying experience. For example, the i-th particle is represented as xi = (xi,1 ,xi,2 ,…, xi,d) in the d-dimensional space. The best previous position of the i-th particle is recorded and represented as:

$$\text{Pbesti} = (\text{Pbesti},1, \text{Pbesti},2 ,..., \text{Pbest i},d) \qquad (1)$$

The index of best particle among all of the particles in the group is gbestd. The velocity for particle i is represented as vi = (vi,1 ,vi,2 ,…, vi,d). The modified velocity and position of each particle can be calculated using the current velocity and the distance from Pbesti, d to gbestd as shown in the following formulas [16-18]:

$$V_{i,m}^{(i+1)} = W.V_{i,m}^{(i)} + C_1 * \text{rand}(\ ) * \left(\text{Pbest}_{i,m} - X_{i,m}^{(i)}\right) + C_2 * \text{rand}(\ ) * \left(\text{Gbest}_{i,m} - X_{i,m}^{(i)}\right) \qquad (2)$$

$$X_{i,m}^{(i+1)} = X_{i,m}^{(i)} + V_{i,m}^{(i+1)} \quad i = 1,2,\dots n \quad m = 1,2,\dots d \qquad (3)$$

D. Bacterial Foraging Algorithm (BFA)

Recently, bacterial foraging algorithm (BFA) has emerged as a powerful technique for the solving optimization problems. BFA mimics the foraging strategy of E. coli bacteria which try to maximize the energy intake per unit time.

From the very early days it has drawn attention of researchers due to its effectiveness in the optimization domain. So as to improve its performance, a large number of modifications have already been undertaken. The bacterial foraging system consists of four principal mechanisms, namely chemotaxis, swarming, reproduction and elimination-dispersal. A brief description of each of these processes along with the pseudo-code of the complete algorithm is described below.

Chemotaxis: This process simulates the movement of an E.coli cell through swimming and tumbling via flagella. Biologically an E.coli bacterium can move in two different ways. It can swim for a period of time in the same direction or it may tumble, and alternate between these two modes of operation for the entire lifetime. Suppose $\theta^i(j,k,l)$ represents ith bacterium at jth chemotactic, kth reproductive and lth elimination-dispersal step. C(i) is the size of the step taken in the random direction specified by the tumble (run length unit). Then in computational chemotaxis the movement of the bacterium may be represented by.

$$\theta^j(j+1,k,l) = \theta^i(j,k,l) + c(i)\frac{\Delta(i)}{\sqrt{\Delta^t(i)\Delta(i)}} \qquad (3)$$

Where Δ indicates a vector in the random direction whose elements lie in [-1, 1].

Swarming: An interesting group behavior has been observed where a group of E.coli cells arrange themselves in a traveling ring by moving up the nutrient gradient when placed amidst a semisolid matrix with a single nutrient chemoeffecter. The cells, when stimulated by a high level of succinate, release an attractant aspertate, which helps them to aggregate into groups and thus move as concentric patterns of swarms with high bacterial density. The cell-to-cell signaling in E. coli swarm may be represented by the following function.

$$J_{cc} = \sum_{i=1}^{S}\left[-d_{attrac\,tan\,t}{}^{e^{\left(-w_{attrac\,tan\,t}\sum_{m=1}^{P}(\theta_m-\theta_m{}^i)^2\right)}}\right] + \sum_{i=1}^{S}\left[-h_{repellant}{}^{e^{\left(-w_{repellant}\sum_{m=1}^{P}(\theta_m-\theta_m{}^i)^2\right)}}\right]$$

$$J_{cc}(\theta,P(j,k,l) = \sum_{i=1}^{S} J_{cc}(\theta,\theta^i(j,k,l) \qquad (4)$$

where $J_{cc}(\theta,P(j,k,l)$ is the objective function value to be added to the actual objective function (to be minimized) to present a time varying objective function, S is the total number of bacteria, p is the number of variables to be optimized, which are present in each bacterium and $\theta = [\theta_1,\theta_2,\ldots,\theta_p$ is a point in the p dimensional search domain.

Reproduction: The least healthy bacteria eventually die while each of the healthier bacteria (those yielding lower value of the objective function) asexually split into two bacteria, which are then placed in the same location. This keeps the swarm size constant.

Elimination and Dispersal: Gradual or sudden changes in the local environment where a bacterium population lives may occur due to various reasons e.g. a significant local rise of temperature may kill a group of bacteria that are currently in a region with a high concentration of nutrient gradients. Events can take place in such a fashion that all the bacteria in a region are killed or a group is dispersed into a new location.

Size of population 'S': Increasing S can significantly increase the computational complexity of the algorithm. However, for larger values of S, it is more likely at least some bacteria near an optimum point should be started, and over time, it is then more likely that many bacterium will be in that region, due to either chemotaxis or reproduction.

Length of chemotactic step 'C(i)': If C(i) are too large, then if the optimum value lies in a valley with steep edges, the search will tend to jump out of the valley, or it may simply miss possible local minima by swimming through them without stopping. On the other hand, if C(i) are too small, convergence can be slow, but if the search finds a local minimum it will typically not deviate too far from it. c(i) is a sort of a "step size" for the algorithm.

Chemotactic step 'Nc': If the size of Nc is chosen to be too short, the algorithm will generally rely more on luck and reproduction, and in some cases, it could more easily get trapped in a local minimum (premature convergence). Ns creates a bias in the random walk (which would not occur if Ns = 0), with large values tending to bias the walk more in the direction of climbing down the hill.

Reproduction number 'Nre': If Nre is too small, the algorithm may converge prematurely; however, larger values of Nre clearly increase computational complexity.

Elimination and dispersal number 'Ned': A low value for Ned dictates that the algorithm will not rely on random elimination-dispersal events to try to find favorable regions. A high value increases computational complexity but allows the bacteria to look in more regions to find good nutrient concentrations. Clearly, if ped is large, the algorithm can degrade to random exhaustive search. If, however, it is chosen appropriately, it can help the algorithm jump out of local optima and into a global optimum.

Parameters defining cell-to-cell attractant functions 'Jcc': If the attractant width is high and very deep, the cells will have a strong tendency to swarm (they may even avoid going after nutrients and favor swarming). On the other hand, if the attractant width is small and the depth shallow, there will be little tendency to swarm and each cell will search on its own. Social versus independent foraging is then dictated by the balance between the strengths of the cell-to-cell attractant signals and nutrient concentrations. [21]

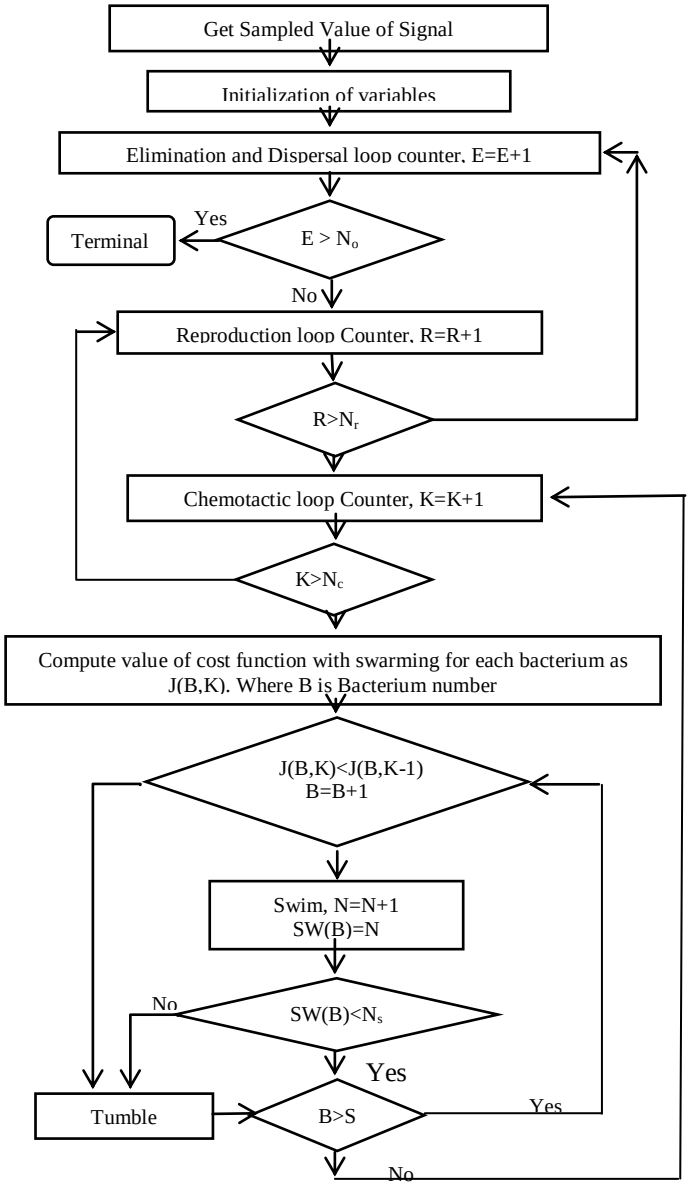


Fig. 5: BFA flow chart

E. Artificial Bee Colony Algorithm

The artificial bee colony (ABC) consists of three groups of bees: employed bees, onlooker bees and scout bees and three actions: searching food source, recruiting bees for the food source and abandoning the food source. Employed bees exploit food source and share the information about the food source with onlookers. Onlooker bees wait in the hive for the information employed bees provided. Scout bees search for the new food source. Employed bees share information about food sources by dancing in the dance area and the nature of dance is proportional to the nectar content of food source just exploited by the dancing bees. Onlooker bees watch the dance and choose a food source according to the probability proportional to the quality of that food source. Therefore, good food sources attract more onlooker bees. Whenever a bee, whether it is scout or onlooker, finds a food source it becomes employed. Whenever a food source is exploited fully, all the employed bees associated with it abandon it, and may become scouts or onlookers. In the ABC algorithm, a food source position represents a possible solution of the problem to be optimized which is represented by a d-dimension real-valued vector. The nectar amount of a food source corresponds to the quality (fitness) of the associated solution. The number of employed bees or the onlookers is equal to the number of the food sources (solutions) in the population. In other words, for every food source, there is only one employed bee [22].

Algorithm:

At the first step, the artificial bees colony generates a randomly distributed initial population P(G = 0) of pn solutions (food source positions), where pn denotes the size of population. Each solution xi (i = 1, 2, ..., pn) is a d-dimensional vector. Here, d is the number of optimization parameters. After initialization, the population of the positions (solutions) is subjected to repeated cycles, C = 1, 2, ...,MCN (maximum cycle number), of the search processes of the employed bees, the onlooker bees and scout bees. An employed bee produces a modification on the position (solution) in her memory depending on the local information (visual information) and tests the nectar amount (fitness value) of the new source (new solution). Provided that the nectar amount of the new one is higher than that of the previous one, the

bee memorizes the new position and forgets the old one. Otherwise she keeps the position of the previous one in her memory. After all employed bees complete the search process; they share the nectar information of the food sources and their position information with the onlooker bees on the dance area. An onlooker bee evaluates the nectar information taken from all employed bees and chooses a food source with a probability related to its nectar amount. As in the case of the employed bee, she produces a modification on the position in her memory and checks the nectar amount of the candidate source. Providing that its nectar is higher than that of the previous one, the bee memorizes the new position and forgets the old on. An artificial onlooker bee chooses a food source depending on the probability value associated with that food source, pi calculated by the equation:

$$p_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n} \quad (5)$$

where fiti is the fitness value of the solution i which is proportional to the nectar amount of the food source in the position i and pn is the number of food sources which is equal to the number of employed bees (pn).

In order to produce a candidate food position from the old one in memory, the artificial BA uses the following expression in eq. (6).

$$v_{ij} = x_{ij} + \phi_{ij}(x_{ij} - x_{kj}) \quad (6)$$

where k ∈ { 1, 2,..., pn} and j ∈ { 1, 2,...,d} are randomly chosen indexes. Although k is determined randomly, it has to be different from i. Øij is a random number between [-1, 1]. It controls the production of neighbor food sources around xij and represents the comparison of two food positions visually by a bee. As can be seen from eq. (6), as the difference between the parameters of the x_{ij} and x_{kj} decreases, the perturbation on the position xij gets decrease, too. Thus, as the search approaches to the optimum solution in the search space, the step length is adaptively reduced. If a parameter value produced by this operation exceeds its predetermined limit, the parameter can be set to an acceptable value. In this work, the value of the parameter exceeding its limit is set to its limit value. The food source of which the nectar is abandoned by the bees is replaced with a new food source by the scouts. In ABC, this is simulated by producing a position randomly and replacing it with the abandoned one. Hence that providing that a position cannot be improved further through a predetermined number of cycles, then that food source is assumed to be abandoned. The value of predetermined number of cycles is an important control parameter of the artificial BA, which is called "limit" for abandonment. Assume that the abandoned source is x_{ij} and j ∈ {1, 2,..., d}, then the scout discovers a new food source to be replaced with xi .This operation can be defined as in eq. (7).

$$x^i_j = x^i_{\min} + rand(0,1)(x^i_j - x^i_{\min}) \quad (7)$$

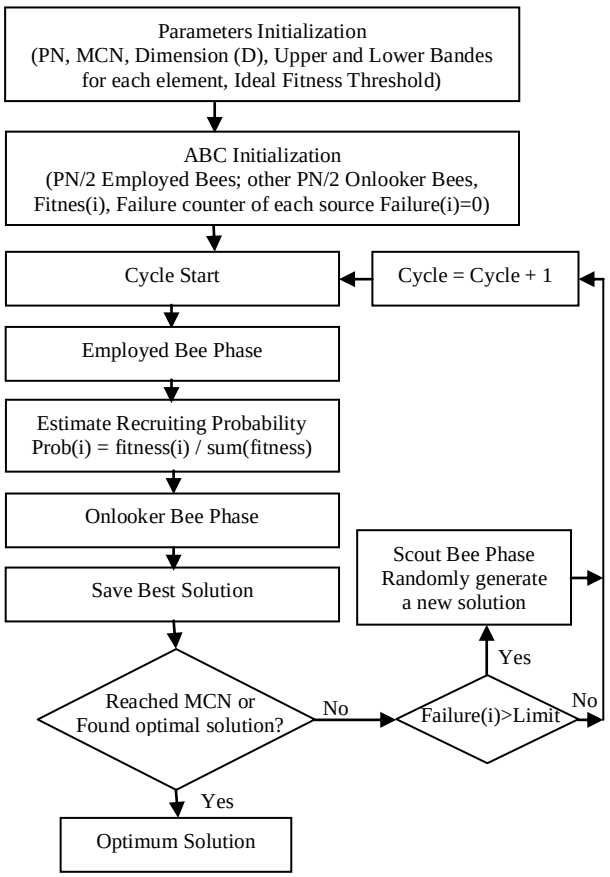


Fig. 6 Artificial Bee Colony flow chart

After each candidate source position vij is produced and then evaluated by the artificial bee, its performance is compared with that of its old one. If the new food has equal or better nectar than the old source, it is replaced with the old one in the memory. Otherwise, the old one is retained in the memory. In other words, a greedy selection mechanism is employed as the selection operation between the old and the candidate one. It is clear from the above explanation that there are four control parameters used in the ABC: The number of food sources (pn) which is equal to the number of employed and onlooker bees, the value of limit, the maximum cycle number (mcn).

F. Simulated Annealing (SA)

Simulated Annealing is a generic probabilistic meta-algorithm used to find an approximate solution to global optimization problems. It is inspired by annealing in metallurgy which is a technique of controlled cooling of material to reduce defects [23][24][25]. The Simulated Annealing algorithm starts with a random solution. A random nearby solution is formed by every iteration. If this solution is a better solution, it will replace the current solution. If it is a worse solution, it may be chosen to replace the current solution with a probability that depends on the temperature parameter. As the algorithm progresses, the temperature parameter decreases, giving worse solutions a lesser chance of replacing the current solution.

We allow worst solutions at the beginning to avoid solution converging to a local minimum rather than the global minimum. We will use the simulated annealing algorithm to solve Random Travelling Salesman Problem. The salesman must visit each city only once and return to the same city in which he began. The constraints that affect the outcome of the algorithm are the initial temperature, the rate at which the temperature decreases and the stopping condition of the algorithm. By adjusting these values, we run the SA and to see that how algorithm responds.
The SA algorithm is described as in flow chart in figure 7.

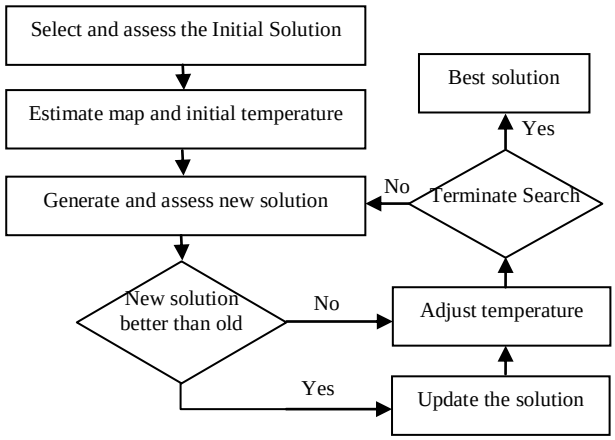


Fig. 7 Simulated Annealing flow chart

IV. IMPLEMENTATION

The experimental results are produced and tested by MatLab software. To encode the inputs (coordinates of cities, or distance between cities), each dataset stored in excel file and read it by MatLab as a matrix.
In this paper we use six datasets of cities with different sizes:

1. Arabic24.tsp : 24 capital cities in Arabic world
2. berlin52.tsp: 52 locations in Berlin
3. ch150.tsp: 150 cities (churritz)
4. lin318.tsp: 318 cities (Lin/Kernighan)
5. fl1400.tsp: 1400 Drilling (Reinelt)
6. brd14051.tsp : 14051 BR Deutschland in den Grenzen von 1989 (Bachem/Wottawa)

The Machine Specification that used to implement the experimental results is:-

HP-Z420 Workstation Desktop
Available Processors : 4
Available Cores : i7
Available Memory : 4 G
Available Windows : WIN-7, 64-bit

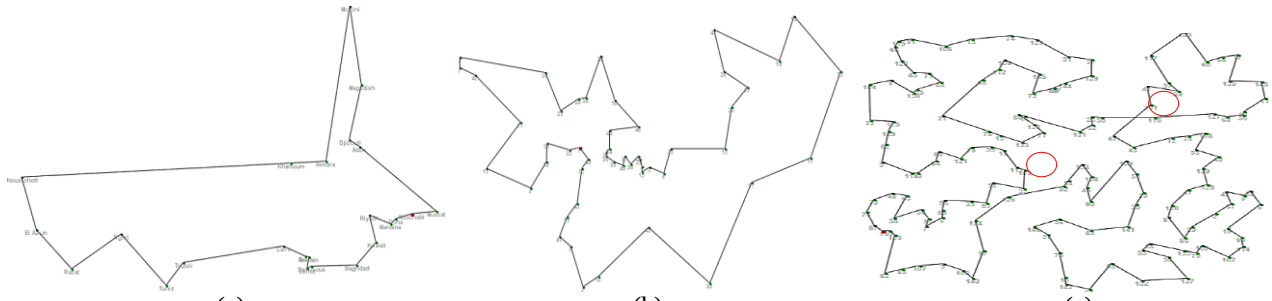
As shown table 1, all techniques work well with suitable running time for dataset with small size up to tens of cities. But for datasets with hundreds cities some techniques converge to local minimum solution, is not the best one. On other hand, for thousands cities all techniques suffer to converge the solution and we cannot guarantee the best solution. Also the techniques show different results within different long running time.

For dataset with 14050 cities, GA, ABC and BFO cannot complete for different reasons: GA needs more memory, ABC faced cycling problem, while BFO do not converge at all running times.On the other side, three techniques ACO, PSO, and SA complete all dataset although they spent a lot of time (around 24 hours) to converge solution that is not guaranteed to be the best.

Table I Results Of All Techniques To Measure Best Value (Path), No. Of Iteration, And Running Time

Dataset		GA	ACO	PSO	BFO	BCO	SA
Arabic24	Best Value	229	229	229	229	229	229
	# of Iterations	131	155	104	98	110	97
	Run Time	1 s.	0 s.	0 s.	0 s.	0 s.	0 s.
berlin52	Best Value	7544	7544	7544	7544	7544	7544
	# of Iterations	129	128	116	91	114	101
	Run Time	6 s.	4	3	5	4	3
ch150	Best Value	6873	6873	6874	6877	6874	6872
	# of Iterations	248	192	207	237	209	176
	Run Time	70 s.	27	20	22	25	19
lin318	Best Value	47,940	47,941	47,941	47,946	47,943	47,936
	# of Iterations	615	582	568	602	593	571
	Run Time	848 s.	562	517	648	681	449
fl1400	Best Value	57,761	57,754	57,749	57,772	57,765	57,754
	# of Iterations	20,898	21,826	19,517	22,004	22,109	22,038
	Run Time	32,863 s.	26,352 s.	24,798	29,513	28,976	20,997
brd14051	Best Value	Out of time	8,183,942	7,970,284	Out of time	Out of time	7,963,861
	# of Iterations		152,729	128,097			133,410
	Run Time		87,519	85,725			81,839

We can determine if the solution is wrong or may be the best solution from graph of tour. As in figures of datasets solutions, if there intersecting lines on path of solution, we can conclude that solution is not optimal (best), the best solution should be without intersecting lines, figure (8 a) shows the best tour for dataset (Arabic24), and figure (8 b) also gives the best tour for database (berlin52), while figure (8 c) illustrates some intersecting lines that indicate that the solution is not the best tour.



(a) (b) (c)

Fig. 8 Solution of some datasets: (a) Optimal solution of (Arabic 24), (b) optimal solution of (berlin52)**, (**c) Solution of (ch150) (not optimal)

V. CONCLUSION

This paper presents some optimization techniques for solving the Travelling Salesman Problem. All techniques produced good results for small dataset but may be not optimal if the size will increase. The running time and path distance increases with increasing number of cities. The quality of the results of any dataset increases with increasing population size but when increases the population size, the running time was increased. So we must to do the best to balance between runtime and the solution quality.

Finally, we noted that GA is good technique but needs more space memory to present its power for solving the problems, while SA does not need additional space because it saves the best solution as so far and ignore others.

In this work, SA, PSO, and then ACO proved their capability to solve TSP.

REFERENCES

[1] V. Cerny, Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm ,Journal of optimization theory and applications: vol. 45, no. l, january i985

[2] Gilbert Laporte, The Traveling Salesman Problem: An overview of exact and approximate algorithms, European Journal of Operational Research 59 (1992) 231-247 North-Holland.

[3] Abdollah Homaifar, Shanguchuan Guan, and Gunar E. Liepins. Schema analysis of the traveling salesman problem using genetic algorithms. Complex Systems, 6(2):183–217, 1992.

[4] Gerard Reinelt. The Traveling Salesman: Computational Solutions for TSP Applications. Springer-Verlag, 1994.

[5] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. The Traveling Salesman. John Wiley and Sons, 1986.

[6] L. Hao, C. L. Ma and F. Li, "Study of Adaptive PID Controller Based on Single Neuron and Genetic Optimization," The 8th International Conference on Electronic Measurement and Instruments ICEMI, Vol. 1, 2007, pp. 240-243.

[7] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu and G. Lanza, "Genetic Programming IV. Routine Human-Competitive Machine Intelligence," Kluwer Academic Publishers, Dordrecht, 2003.

[8] C. R. Reeves, "Using Genetic Algorithms with Small Populations," Proceedings of the 5th International Conference on Genetic Algorithms, 1993.

[9] [Housam Binous, "Dynamic and control of tank's height using genetic algorithm toolbox and fminsearch", Updated 24 September 2007, this toolbox is available at: (http://www.mathworks.com/matlabcentral/fileexchange/16523-dynamic-and-control-of-tanks-height-using-genetic-algorithm-toolbox-and-fminsearch).

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

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

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

[13] 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.

[14] 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.

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

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

[17] 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.

[18] Kennedy J., Eberhart R., Particle swarm optimization, Proc. IEEE Int. Conf. on Neural Network, Vol. 4, p. 1942 – 1948, 1995.

[19] Shi Y., Eberhart R., A modified particle swarm optimizer, Proc. 1998 Int. Conf. on Evolutionary Computation, The IEEE World Congress on Computational Intelligence, Anchorage, May 1998, p. 69 – 73.

[20] Clerc M., Kennedy J., The particle swarm-explosion, stability, and convergence in a multidimensional complex space, IEEE Trans. Evolutionary Computation, 2002, Vol. 6, p. 58 –73.

[21] Ahmed Bensenouci, "PID Controllers Design for a Power Plant Using Bacteria Foraging Algorithm", IEEE 2011.

[22] D.T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri , S. Rahim , M. Zaidi, " The Bees Algorithm – A Novel Tool for Complex Optimization Problems", Manufacturing Engineering Centre, Cardiff University, Cardiff CF24 3AA, UK.

[23] http://www.btluke.com/simann1.html

[24] Emile Aarts, Jan Korst, and Wil Michiels. Simulated Annealing, chapter 7. Springer, 2005. Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques

[25] David Bookstaber, ―Simulated Annealing for Traveling Salesman Problem‖, Spring, 1997

[26] Mohd. Junedul Haque and Khalid. W. Magld, Improving the Solution of Traveling Salesman Problem Using Genetic, Memetic Algorithm and Edge assembly Crossover,2012.

[27] Rajesh Matai, Surya Prakash Singh and Murari Lal Mittal, Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches.

[28] [28] fozia hanif khan, nasiruddin khan, syed inayatullah, and shaikh tajuddin nizami, solving tsp problem by using genetic algorithm.

[29] Adewole Philip, Akinwale Adio Taofiki, Otunbanowo Kehinde, A Genetic Algorithm for Solving Travelling Salesman Problem, January2011.

[30] Naef Taher Al Rahedi and Jalal Atoum, Solving TSP problem using New Operator in Genetic Algorithms, American Journal of Applied Sciences 6(8)

[31] B. Freisleben and P. Merz, "A Genetic local search Algorithm for Solving Symmetric and Asymmetric Travelling Salesman Problems," International Conference On Evolutionary Computation, 1996.

[32] Genetic Algorithms and the Traveling Salesman Problem by Kylie Bryant December 2000.

