1

Population-Based Optimization Algorithms for Solving the Travelling Salesman Problem

Mohammad Reza Bonyadi, Mostafa Rahimi Azghadi
and Hamed Shah-Hosseini
Department of Electrical and Computer Engineering,
Shahid Beheshti University,
Tehran, Iran

1. Introduction

The Travelling Salesman Problem or the TSP is a representative of a large class of problems known as combinatorial optimization problems. In the ordinary form of the TSP, a map of cities is given to the salesman and he has to visit all the cities only once to complete a tour such that the length of the tour is the shortest among all possible tours for this map. The data consist of weights assigned to the edges of a finite complete graph, and the objective is to find a Hamiltonian cycle, a cycle passing through all the vertices, of the graph while having the minimum total weight. In the TSP context, Hamiltonian cycles are commonly called tours. For example, given the map shown in figure l, the lowest cost route would be the one written (A, B, C, E, D, A), with the cost 31.

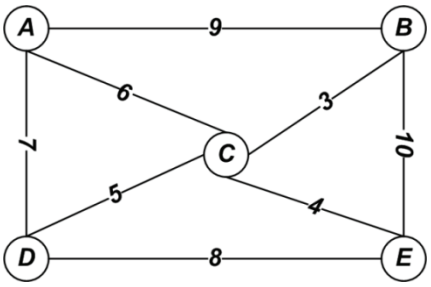

Fig. 1. The tour with A=>B =>C =>E =>D => A is the optimal tour.

In general, the TSP includes two different kinds, the Symmetric TSP and the Asymmetric TSP. In the symmetric form known as STSP there is only one way between two adjacent cities, i.e., the distance between cities A and B is equal to the distance between cities B and A (Fig. 1). But in the ATSP (Asymmetric TSP) there is not such symmetry and it is possible to have two different costs or distances between two cities. Hence, the number of tours in the ATSP and STSP on *n* vertices (cities) is *(n-1)!* and *(n-1)!/2,* respectively. Please note that the graphs which represent these TSPs are complete graphs. In this chapter we mostly consider the STSP. It is known that the TSP is an NP-hard problem (Garey & Johnson, 1979) and is often used for testing the optimization algorithms. Finding Hamiltonian cycles or traveling

Source: Travelling Salesman Problem, Book edited by: Federico Greco, ISBN 978-953-7619-10-7, pp. 202, September 2008,
I-Tech, Vienna, Austria

salesman tours is possible using a simple dynamic program using time and space $O(2^n n^{O(1)})$, that finds Hamiltonian paths with specified endpoints for each induced subgraph of the input graph (Eppstein, 2007). The TSP has many applications in different engineering and optimization problems. The TSP is a useful problem in routing problems e.g. in a transportation system.

There are different approaches for solving the TSP. Solving the TSP was an interesting problem during recent decades. Almost every new approach for solving engineering and optimization problems has been tested on the TSP as a general test bench. First steps in solving the TSP were classical methods. These methods consist of heuristic and exact methods. Heuristic methods like cutting planes and branch and bound (Padberg & Rinaldi, 1987), can only optimally solve small problems whereas the heuristic methods, such as 2-opt (Lin & Kernighan, 1973), 3-opt, Markov chain (Martin et al., 1991), simulated annealing (Kirkpatrick et al., 1983) and tabu search are good for large problems. Besides, some algorithms based on greedy principles such as nearest neighbour, and spanning tree can be introduced as efficient solving methods. Nevertheless, classical methods for solving the TSP usually result in exponential computational complexities. Hence, new methods are required to overcome this shortcoming. These methods include different kinds of optimization techniques, nature based optimization algorithms, population based optimization algorithms and etc. In this chapter we discuss some of these techniques which are algorithms based on population.

Population based optimization algorithms are the techniques which are in the set of the nature based optimization algorithms. The creatures and natural systems which are working and developing in nature are one of the interesting and valuable sources of inspiration for designing and inventing new systems and algorithms in different fields of science and technology. Evolutionary Computation (Eiben & Smith, 2003), Neural Networks (Haykin, 99), Time Adaptive Self-Organizing Maps (Shah-Hosseini, 2006), Ant Systems (Dorigo & Stutzle, 2004), Particle Swarm Optimization (Eberhart & Kennedy, 1995), Simulated Annealing (Kirkpatrik, 1984), Bee Colony Optimization (Teodorovic et al., 2006) and DNA Computing (Adleman, 1994) are among the problem solving techniques inspired from observing nature.

In this chapter population based optimization algorithms have been introduced. Some of these algorithms were mentioned above. Other algorithms are Intelligent Water Drops (IWD) algorithm (Shah-Hosseini, 2007), Artificial Immune Systems (AIS) (Dasgupta, 1999) and Electromagnetism-like Mechanisms (EM) (Birbil & Fang, 2003). In this chapter, every section briefly introduces one of these population based optimization algorithms and applies them for solving the TSP. Also, we try to note the important points of each algorithm and every point we contribute to these algorithms has been stated. Section nine shows experimental results based on the algorithms introduced in previous sections which are implemented to solve different problems of the TSP using well-known datasets.

2. Evolutionary algorithms

2.1 Introduction

Evolutionary Algorithms (EAs) imitates the process of biological evolution in nature. These are search methods which take their inspiration from natural selection and survival of the fittest as exist in the biological world. EA conducts a search using a population of solutions. Each iteration of an EA involves a competitive selection among all solutions in the

population which results in survival of the fittest and deletion of the poor solutions from the population. By swapping parts of a solution with another one, recombination is performed and forms the new solution that it may be better than the previous ones. Also, a solution can be mutated by manipulating a part of it. Recombination and mutation are used to evolve the population towards regions of the space which good solutions may reside.
Four major evolutionary algorithm paradigms have been introduced during the last 50 years: genetic algorithm is a computational method, mainly proposed by Holland (Holland, 1975). Evolutionary strategies developed by Rechenberg (Rechenberg, 1965) and Schwefel (Schwefel, 1981). Evolutionary programming introduced by Fogel (Fogel et al., 1966), and finally we can mention genetic programming which proposed by Koza (Koza, 1992). Here we introduce the GA (Genetic Algorithm) for solving the TSP. At the first, we prepare a brief background on the GA.

2.2 Genetic algorithms

Genetic Algorithms focus on optimizing general combinatorial problems. GAs have long been studied as problem solving tools for many search and optimization problems, specifically those that are inherent in NP-Complete problems. Various candidate solutions are considered during the search procedure in the system, and the population evolves until a candidate solution satisfies the predefined criteria. In most GAs, a candidate solution, called an individual, is represented by a binary string (Goldberg, 1989) i.e. a string of 0 or 1 elements. Each solution (individual) is represented as a sequence (chromosome) of elements (genes) and is assigned a fitness value based on the value given by an evaluation function. The fitness value measures how close the individual is to the optimum solution. A set of individuals constitutes a population that evolves from one generation to the next through the creation of new individuals and deletion of some old ones. The process starts with an initial population created in some way, e.g. through a random process. Evolution can take two forms:
Crossover:
Two selected chromosomes can be combined by a crossover operator, the result of which will replace the lowest fitness chromosome in the population. Selection of each chromosome is performed by an algorithm to ensure that the selection probability is proportional to the fitness of the chromosome. A new chromosome has the chance to be better than the replaced one. The process is oriented towards the sub-regions of the search space, where an optimal solution is supposed to exist (Goldberg, 1989).
Mutation:
In mutation process, a gene from a selected chromosome is randomly changed. This provides additional chances of entering unexplored sub-regions. Finally, the evolution is stopped when either the goal is reached or a maximum CPU time has been spent (Goldberg, 1989).
In the following the GA operation pseudo code has been written:

1. Start
2. Population initialization
3. Repeat until (satisfying termination criteria)
    - Selection
    - Cross over
    - Mutation

- Making new population with the fittest solutions
- Evaluation
- Checking the termination criterion

4. Take the best solution as output
5. End

2.3 Solving the TSP using GA

As mentioned earlier, the TSP is known as a classical NP-complete problem, which has extremely large search spaces and is very difficult to solve (Louis & Gong, 2000). Hence, classical methods for solving TSP usually result in exponential computational complexities. These methods consist of heuristic and exact methods. Heuristic methods like cutting planes and branch and bound (Padherg & Rinaldi, 1987), can only optimally solve small problems while the heuristic methods, such as 2-opt (Lin & Kernighan, 1973), 3-opt, Markov chain (Martin et al., 1991), simulated annealing (Kirkpatrick et al., 1983) and tabu search are good for large problems. Besides, some algorithms based on greedy principles such as nearest neighbour, and spanning tree can be used as efficient solving methods. Nevertheless, because of the tremendous number of possible solutions and large search spaces, GAs seem to be wise approaches for solving the TSP especially when they are accompanied with carefully designed genetic operators (Jiao & Wang, 2000). GAs search the large space of solutions toward best answer and the operators can help the search process become faster and also they prepare the ability to avoid being trapped in local optima.

In recent years, solving the TSP using evolutionary algorithms and specially GAs has attracted a lot of attention. Many studies have been performed and researchers try to contribute to different parts of solving process. Some of researchers pose different forms of GA operators (Yan et al., 2005) in comparison to the former ones and others attempt to combine GA with other possible approaches like ACO (Lee, 2004), PSO and etc. In addition, some authors implement a new evolutionary idea or combine some previous algorithms and idea to create a new method (Bonyadi et al., 2007). Here we investigate some of these works and compare their results. Due to the spread of related works we can not mention all of them here. But The reader is referred to the prepared references for further information.

In all of the performed works, two instances are mentionable. First: all of the proposed algorithms work toward finding the nearest answer to the best solution. Second: solving the TSP in a more little time is a key point in this problem because of its special application which require, finding the best feasible answer fast.

In (Bonyadi et al., 2007), the authors made some changes to two previous local search algorithms i.e. the Shuffled Frog Leaping (SFL) and the Civilization and Society (CS) and combined these two algorithms with the GA idea. In this study, as it is common in a conventional GA, at first the elements of the population perform mutation or crossover in random order. Then for every element of this population, a local search algorithm, which is a mix of both SFL and CS, is performed. The results demonstrate significant improvements in terms of time complexity and reaching better solutions in comparison to the GAs which apply only SFL or CS in their usual forms. Hence, the main contribution in this work is combining two previous search methods and using them with the GA, simultaneously. The evaluation results of the proposed algorithm have been prepared in section nine.

In another work (Yan et al., 2005) a new algorithm based on Inver-over operator, for combinatorial optimization problems has been proposed. Inver-over is based on simple

inversion; however, knowledge taken from other individuals in the population influences its action. In this algorithm some new strategies including selection operator, replace operator and some new control strategy have been applied. The results prove that these changes are very efficient to accelerate the convergence. A consequence, it is inferred that, one of the points for contribution is operators. Suitable changes in the conventional form of operators might lead to major differences in the search and optimization procedure.
Through the experiments, GAs are global search algorithms appropriate for problems with huge search spaces. In addition, heuristic methods can be applied for search in local areas. Hence, combination of these two search algorithms can result in producing high quality solutions. Cooperation between Speediness of local search methods in regional search and robustness of evolutionary methods in global search can be very useful to obtain the global optimum. Recently, (Nguyen et al., 2007) proposed a hybrid GA to find high-quality solutions for the TSP. The main contribution of this study is to show the suitable combination of a GA as a global search with a heuristic local search which are very promising for the TSP. In addition, the considerable improvements in the achieved results prove that the effectiveness and efficiency of the local search in the performance of hybrid GAs. Through these results, one of other points where it can be kept in mind is the design of the GA in a case that it balances between local and global search. Moreover, many other studies have been performed that all of them combine the local and global search mechanisms for solving the TSP.
As mentioned earlier, one of the points that solving the TSP can contribute is recombination operators i.e. mutation and crossover. Based on (Takahashi, 2005) there are two kinds of crossover operators for solving the TSP. Conventional encoding of the TSP which is an array representation of chromosomes where every element of this array is a gene that in the TSP shows a city. The first kind of crossover operator corresponds to this chromosome structure. In this operator two parents are selected and with exchanging of some parts in parents the children are reproduced. The second type performs crossover operation with mentioning epistasis. In this method it is tried to retain useful information about links of parent's edges which leads to convergence. Also, in (Tsai et al., 2004) another work on genetic operators has been performed which resulted in good achievements.

3. Ant colony optimization (ACO)

3.1 Introduction

The ACO (Ant Colony Optimization) heuristic is inspired by the real ant behaviour (figure 2) in finding the shortest path between the nest and the food (Beckers et al., 1992). This is achieved by a substance called pheromone that shows the trace of an ant. In its searching the ant uses heuristic information which is its own knowledge of where the smell of the food comes from and the other ants' decision of the path toward the food by pheromone information (Holldobler & Wilson, 1990).

Fig. 2. Real ant behaviour in finding the shortest path between the nest and the food

In fact the algorithm uses a set of artificial ants (individuals) which cooperate to the solution of a problem by exchanging information via pheromone deposited on graph edges. The ACO algorithm is employed to imitate the behaviour of real ants and is as follows:

Initialize
Loop
    Each ant is positioned on a starting node
    Loop
        Each ant applies a state transition rule to incrementally build a solution and a local pheromone updating rule
    Until all ants have built a complete solution
    A global pheromone updating rule is applied
Until end condition

3.2 State transition

Consider n is the city amount; m is the quantity of the ants in an ACO problem; dij is the length of the path between adjacent cities i and j; $\tau_{ij}(t)$ is the intensity of trail on edge (i, j) at time t . At the beginning of the algorithm, an initialization algorithm determines the ants positions on different cities and initial value $\tau_{ij}(0)$, a small positive constant c for trail intensity are set on edges. The first element of each ant's tabu list is set to its starting city.

The state transition is given by equation 1, which ant k in city i chooses to move to city j :

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{(\tau_{ij}(t))^{\alpha}(\eta_{ij}(t))^{\beta}}{\sum_{k \notin allowed_k}(\tau_{ik}(t))^{\alpha}(\eta_{ik}(t))^{\beta}}, & \text{if } j \notin allowed_k \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where $allowed_k = \{N\text{-}tabu_k\}$, which is the set of cities that remain to be visited by ant k positioned on city i (to make the solution feasible) α and β are parameters that determine the relative importance of trail versus visibility, and $\eta = 1/d$ is the visibility of edge (i, j) .

3.3 Trial updating

In order to improve future solutions, the pheromone trails of the ants must be updated to reflect the ant's performance and the quality of the solutions found. The global updating rule is implemented as follows. Once all ants have built their tours, pheromone is updated on all edges according to the following formula (equations 2 to 4):

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \sum_{k=1}^{m}\Delta\tau_{ij}^{k} \tag{2}$$

where

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{L_k}, & \text{if edge } (i, j) \text{ is visited by the kth ant at current cycle} \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \quad (4)$$

ρ ($0 < \rho < 1$) is trail persistence, L_k is the length of the tour found by kth ant , Q is a constant related to the quantity of trail laid by ants. In fact, pheromone placed on the edges plays the role of a distributed long-term memory (Dorigo & Gambardella, 1997). The algorithm iterates in a predefined number of iterations and the best solutions are saved as the results.

3.4 Solving the TSP using ACO

As it is mentioned, the ACO algorithm has good potential for problem solving and recently has attracted a lot of attentions specifically for solving NP-Hard set of problems. One of the earliest best works for solving the TSP uses the ACS (Ant Colony System) is presented in (Dorigo & Gambardella, 1997). They use the ACS algorithm for solving the TSP and they claim that the ACS outperforms other nature-inspired algorithms such as simulated annealing and evolutionary computation. In addition, they compared ACS-3-opt, a version of the ACS improved with a local search procedure, to some of the best performing algorithms for symmetric and asymmetric TSPs.

One of the other recent approaches for solving the TSP is proposed in (Song et al., 2006). In particular, the option that an ant hunts for the next step, the use of a combination of two kinds of pheromone evaluation models, the change of size of population in the ant colony during the run of the algorithm, and the mutation of pheromone have been studied. One of the most powerful attitudes in their paper was choosing the appropriate ACO model that proposed by M. Dorigo which were called ant-cycle, ant-quantity and ant-density models. These three models differ in the way the pheromone trail is updated. In ant-cycle algorithm, the trail is updated after all the ants finish their tours. In contrast, in the last two models, each ant lays its pheromone at each step without waiting for the end of the tour (Song et al., 2006). Furthermore they claim that in early stage of iterations, the convergence speed is faster using ant-density model in comparison with the other two models. Thus, at the beginning, the ant-density model is applied. Because the Ant-cycle system has the advantage of utilizing the global information, it is used at the other times. A mutation mechanism same as in genetic algorithm has been added to the improved ACO algorithm to assist the algorithm to jumping out from local optima's. In their proposed improved ACO, a population sizing method is used which changes the number of individuals (ants).

4. Particle swarm optimization (PSO)

4.1 Introduction

Particle Swarm Optimization (PSO) uses swarming behaviours observed in flocks of birds, schools of fish, or swarms of bees (figure 3), and even human social behaviour, from which intelligence emerges (Kennedy & Eberhart, 2001).

The standard PSO model consists of a swarm of particles. They move iteratively through the *feasible* problem space to find the new solutions. Each particle has a position represented by a position-vector $\vec{x}_i$ (i is the index of the particle), and a velocity represented by a velocity-vector $\vec{v}_i$. Each particle remembers its own best position so far in a vector $\vec{x}_i^{\#}$ and its j-th

dimensional value is $x_{ij}^{\#}$. The best position-vector among the swarm heretofore is then stored in a vector x^* and its j-th dimension value is x^*_j.The PSO procedure is as follows:


Fig. 3. Birds or fish exhibit such a coordinated collective behaviour

Algorithm 1 Particle Swarm Algorithm
01. Begin
02. Parameter settings and swarm initialization
03. Evaluation
04. $g = 1$
05. While (the stopping criterion is not met) do
06. For each particle
07. Update velocity
08. Update position and local best position
09. Evaluation
10. EndFor
11. Update leader (global best particle)
12. $g++$
15. End While
14. End

The PSO algorithm has several phases consist of Initialization, Evaluation, Update Velocity and Update Position. These phases are described in more details (See figure 5).

4.2 Initialization

The initialization phase is used to determine the position of the m particles in the first iteration. The random initialization is one of the most popular methods for this job. There is no guarantee that a randomly generated particle be a good answer and this will make the initialization more attractive. A good initialization algorithm make the optimization algorithm more efficient and reliable. For initialization, some known prior knowledge of the problem can help the algorithm to converge in less iterations. As an example, in 0-1 knapsack problem, there is a greedy algorithm which can generate good candidate answers but not optimal one. This greedy algorithm can be used for initializing the population and the optimization algorithm will continue the optimization from this good point.

4.3 Update velocity and position

In each iteration, each particle updates its velocity and position according to its heretofore best position, its current velocity and some information of its neighbours. Equation 5 is used for updating the velocity:

$$\overrightarrow{v_i(t)} = \underbrace{w\overrightarrow{v_i(t-1)}}_{inertia} + \underbrace{c_1 r_1 \left(x_i^{\#}(t-1) - \overrightarrow{x_i(t-1)} \right)}_{Personalinfluence} + \underbrace{c_2 r_2 \left(\overrightarrow{x^*(t-1)} - \overrightarrow{x_i(t-1)} \right)}_{Socialinfluence} \quad (5)$$

Where $\overrightarrow{x_i(t)}$ is the position-vector in iteration t (i is the index of the particle), $\overrightarrow{v_i(t)}$ is the velocity-vector in iteration t. $x_1^{\#}(t)$ is the best position so far of particle i in iteration t and its j-th dimensional value is $x^{\#}_{ij}(t)$. The best position-vector among the swarm heretofore is then stored in a vector $x^*(t)$ and its j-th dimension value is $x^*_j(t)$. $r1$ and $r2$ are the random numbers in the interval [0,1]. $c1$ is a positive constant, called as coefficient of the self-recognition component, $c2$ is a positive constant, called as coefficient of the social component. The variable **w** is called as the inertia factor, which value is typically setup to vary linearly from 1 to near 0 during the iterated processing. In fact, a large inertia weight facilitates global exploration (searching new areas), while a small one tends to facilitate local exploration. Consequently a reduction on the number of iterations required to locate the optimum solution (Yuhui & Eberhart, 1998). Figure 4 illustrates this reduction. The algorithm invokes the equation 6 for updating the positions:

$$\overrightarrow{x_i(t)} = \overrightarrow{x_i(t-1)} + \overrightarrow{v_i(t)} \quad (6)$$

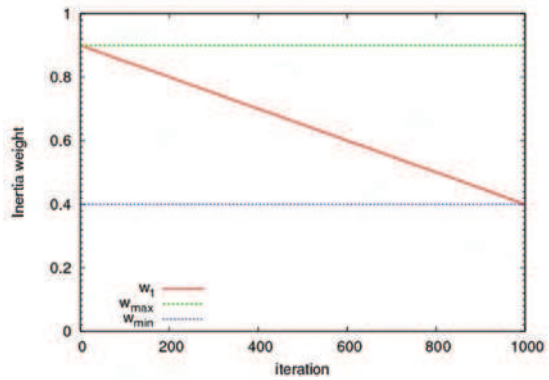

Fig. 4. The value of the inertia weight is decreased during a run

4.4 Solving the TSP using PSO

As it is described before, Particle Swarm Optimization (PSO) has a good potential for problem solving. The susceptibilities and charms of this nature based algorithm convinced researchers to use the PSO to solve NP-Hard problems such as TSP and Job-Scheduling. Here, we investigate some of these proposed approaches for solving the TSP.

One of the attractive works for solving the TSP was cited in (Yuan et al.., 2007). They propose a novel hybrid algorithm which invokes the sufficiency of both PSO and COA (Chaotic Optimization Algorithm) (Zhang et al., 2001). In fact, they exert the COA to restrain the particles from getting stock on local optima's in rudimentary iterations. In other word, they claim that the COA could considerably useful to keep particle's global searching ability.

One of the other exciting algorithms based on PSO for solving TSP is introduced in (Pang et al., 2004). In this paper they propose an algorithm based on PSO which uses the fuzzy matrices for velocity and position vectors. In addition, they use the fuzzy multiplication and addition operators for velocity and position updating formulas (equations (5) and (6)). The mentioned PSO algorithm in previous sections modified to an algorithm which works based on fuzzy means such as fuzzification and defuzzification. In each iteration, the position of each generated solution has been defuzzified to determine the cost of the individual. This cost will be used for updating the local best position.

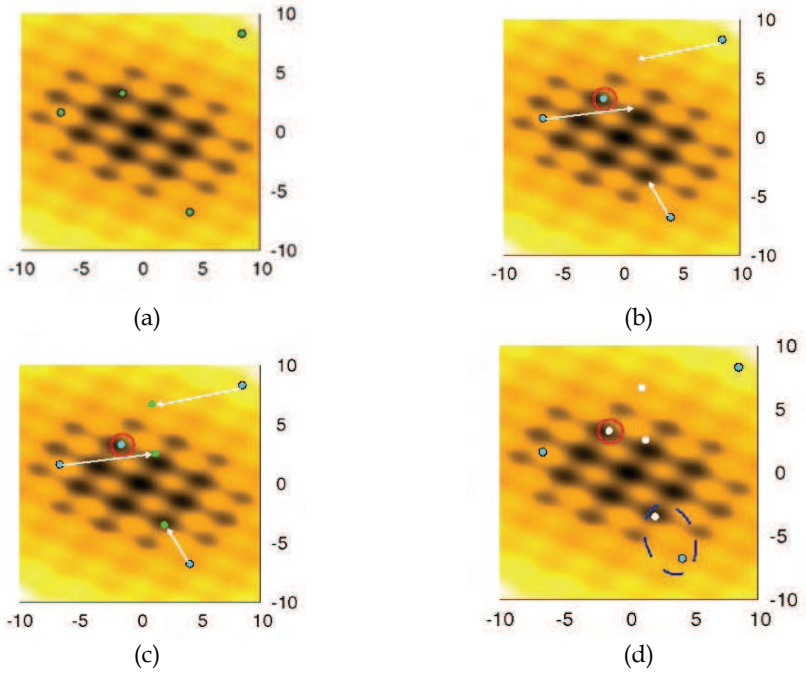

Fig. 5. (a) Create a 'population' of agents (called particles) uniformly distributed over X (feasible region) and Evaluate each particle's position according to the objective function, (b) Update particles' velocities according to equation (5), (c) Move particles to their new positions according to equation (6), (d) If a particle's current position is better than its previous best position, update it.

5. Intelligent water drops

5.1 Introduction

The last work on the population based optimization algorithms inspired by nature is a novel problem solving method proposed by Hamed Shah-hosseini (Shah-hosseini, 2007). This method is called "Intelligent Water Drops" or IWD algorithm which is based on the processes that happen in the natural river systems and the actions and reactions that take place between water drops in the river and the changes that happen in the environment that river is flowing. Here we prepare a complete description on this new and interesting

method. To start with, the inspiration of IWD, natural water drops, will be stated. After that the IWD system has been introduced. And finally these ideas are embedded into the proposed algorithm for solving the Traveling Salesman Problem or the TSP.

5.2 Natural water drops

In nature, we often see water drops moving in rivers, lakes, and seas. As water drops move, they change their environment in which they are flowing. Moreover, the environment itself has substantial effects on the paths that the water drops follow. Consider a hypothetical river in which water is flowing and moving from high terrain to lower terrain and finally joins a lake or sea. The paths that the river follows, based on our observation in nature, are often full of twists and turns. We also know that the water drops have no visible eyes to be able to find the destination (lake or river). If we put ourselves in place of a water drop of the river, we feel that some force pulls us toward itself (gravity). This gravitational force as we know from physics is straight toward the center of the earth. Therefore with no obstacles and barriers, the water drops would follow a straight path toward the destination, which is the shortest path from the source to the destination. However, due to different kinds of obstacles in the way of this ideal path, the real path will have to be different from the ideal path and we often see lots of twists and turns in a river path. In contrast, the water drops always try to change the real path to make it a better path in order to approach the ideal path. This continuous effort changes the path of the river as time passes by. One feature of a water drop is the velocity that it flows which enables the water drop to transfer an amount of soil from one place to another place in the front. This soil is usually transferred from fast parts of the path to the slow parts. As the fast parts get deeper by being removed from soil, they can hold more volume of water and thus may attract more water. The removed soils which are carried in the water drops are unloaded in slower beds of the river. There are other mechanisms which are involved in the river system which we don't intend to consider them all here.

In summary, a water drop in a river has a non-zero velocity. It often carries an amount of soil. It can load some soil from an area of the river bed, often from fast flowing areas and unload them in slower areas of the river bed. Obviously, a water drop prefers an easier path to a harder path when it has to choose between several branches that exist in the path from the source to the destination. Now we can introduce the intelligent water drops.

5.3 Intelligent water drops

Based on the observation on the behavior of water drops, we develop an artificial water drop which possesses some of the remarkable properties of the natural water drop. This Intelligent Water Drop, IWD for short, has two important properties:

1. The amount of the soil it carries now, Soil (IWD).
2. The velocity that it is moving now, Velocity (IWD).

flows in its environment. This environment depends on the problem at hand. In an environment, there are usually lots of paths from a given source to a desired destination, which the position of the destination may be known or unknown. If we know the position of the destination, the goal is to find the best (often the shortest) path from the source to the destination. In some cases, in which the destination is unknown, the goal is to find the optimum destination in terms of cost or any suitable measure for the problem.

We consider an IWD moving in discrete finite-length steps. From its current location to its next location, the IWD velocity is increased by the amount nonlinearly proportional to the inverse of the soil between the two locations. Moreover, the IWD's soil is increased by removing some soil of the path joining the two locations. The amount of soil added to the IWD is inversely (and nonlinearly) proportional to the time needed for the IWD to pass from its current location to the next location. This duration of time is calculated by the simple laws of physics for linear motion. Thus, the time taken is proportional to the velocity of the IWD and inversely proportional to the distance between the two locations.
Another mechanism that exists in the behavior of an IWD is that it prefers the paths with low soils on its beds to the paths with higher soils on its beds. To implement this behavior of path choosing, we use a uniform random distribution among the soils of the available paths such that the probability of the next path to choose is inversely proportional to the soils of the available paths. The lower the soil of the path, the more chance it has for being selected by the IWD.
In this part, we specifically express the steps for solving the TSP. The first step is how to represent the TSP in a suitable way for the IWD. For the TSP, the cities are often modeled by nodes of a graph, and the links in the graph represent the paths joining each two cities. Each link or path has an amount of soil. An IWD can travel between cities through these links and can change the amount of their soils. Therefore, each city in the TSP is denoted by a node in the graph which holds the physical position of each city in terms of its two dimensional coordinates while the links of the graph denote the paths between cities. To implement the constraint that each IWD never visits a city twice, we consider a visited city list for the IWD which this list includes the cities visited so far by the IWD. So, the possible cities for an IWD to choose in its next step must not be from the cities in the visited list.

5.4 Solving the TSP using IWD

In the following, we present the proposed Intelligent Water Drop (IWD) algorithm for the TSP:

1. Initialization of static parameters: set the number of water drops N_{IWD}, the number of cities N_C, and the Cartesian coordinates of each city i such that $\mathbf{c}(i) = [x_i, y_i]^T$ to their chosen constant values. The number of cities and their coordinates depend on the problem at hand while the N_{IWD} is set by the user. Here, we choose N_{IWD} to be equal to the number of cities. For velocity updating, we use parameters $a_v = 1000$, $b_v = .01$ and $c_v = 1$. For soil updating, we use parameters $a_s = 1000$, $b_s = .01$ and $c_s = 1$. Moreover, the initial soil on each link is denoted by the constant $InitSoil$ such that the soil of the link between every two cities i and j is set by $soil(i, j) = InitSoil$. The initial velocity of IWDs is denoted by the constant $InitVel$. Both parameters $InitSoil$ and $InitVel$ are also user selected. In this paper, we choose $InitSoil = 1000$ and $InitVel = 100$. The best tour is denoted by T_B which is still unknown and its length is initially set to infinity: $Len(T_B) = \infty$. Moreover, we should specify the maximum number of iterations that the algorithm should be repeated or some other terminating condition suitable for the problem.

2. Initialization of dynamic parameters: For every IWD, we create a visited city list $V_c(IWD) = \{\ \}$ set to the empty list. The velocity of each IWD is set to $InitVel$ whereas the initial soil of each IWD is set to zero.
3. For every IWD, randomly select a city and place that IWD on the city.
4. Update the visited city lists of all IWDs to include the cities just visited.
5. For each IWD, choose the next city j to be visited by the IWD when it is in city i with the following probability (equation 7):

$$p_i^{IWD}(j) = \frac{f\left(soil(i,j)\right)}{\sum_{k \notin vc(IWD)} f\left(soil(i,k)\right)} \tag{7}$$

such that $f(soil(i,j)) = \dfrac{1}{\varepsilon_s + g(soil(i,j))}$ and

$g(soil(i,j)) = \begin{cases} soil(i,j) & if \min_{l \notin \text{vc(IWD)}}(soil(i,l)) \geq 0 \\ soil(i,j) - \min_{l \notin vc(IWD)}(soil(i,l)) & else \end{cases}$. Here ε_s is a small positive number to prevent a possible division by zero in the function $f(.)$. Here, we use $\varepsilon_s = 0.01$. The function $\min(.)$ returns the minimum value among all available values for its argument. Moreover, $vc(IWD)$ is the visited city list of the IWD.

6. For each IWD moving from city i to city j, update its velocity based on equation 8.

$$vel^{IWD}(t+1) = vel^{IWD}(t) + \frac{a_v}{b_v + c_v . soil(i,j)} \tag{8}$$

such that $vel^{IWD}(t+1)$ is the updated velocity of the IWD. $soil(i,j)$ is the soil on the path (link) joining the current city i and the new city j. With formula (8), the velocity of the IWD increases less if the amount of the soil is high and the velocity would increase more if the soil is low on the path.

7. For each IWD, compute the amount of the soil, $\Delta soil(i,j)$, that the current water drop IWD loads from its the current path between two cities i and j using equation 9.

$$\Delta soil(i,j) = \frac{a_s}{b_s + c_s . time\left(i, j; vel^{IWD}\right)} \tag{9}$$

such that $time\left(i, j; vel^{IWD}\right) = \dfrac{\left\| \mathbf{c}(i) - \mathbf{c}(j) \right\|}{\max\left(\varepsilon_v, vel^{IWD}\right)}$ which computes the time taken to travel from city i to city j with the velocity vel^{IWD}. Here, the function $\mathbf{c}(.)$ represents the two

dimensional positional vector for the city. The function $\max(.,.)$ returns the maximum value among its arguments, which is used here to threshold the negative velocities to a very small positive number $\varepsilon_v = 0.0001$.

8. For each IWD, update the soil of the path traversed by that IWD using equation 10.

$$\begin{aligned} soil(i,j) &= (1-\rho).soil(i,j) - \rho.\Delta soil(i,j) \\ soil^{IWD} &= soil^{IWD} + \Delta soil(i,j) \end{aligned} \tag{10}$$

where $soil^{IWD}$ represents the soil that the IWD carries. The IWD goes from city i to city j. The parameter ρ is a small positive number less than one. Here we use $\rho = 0.9$.

9. For each IWD, complete its tour by using steps 4 to 8 repeatedly. Then, calculate the length of the tour traversed by the IWD, and find the tour with the minimum length among all IWD tours in this iteration. We denote this minimum tour by T_M.
10. Update the soils of paths included in the current minimum tour of the IWD, denoted by T_M which is computed based on equation 11.

$$soil(i,j) = (1-\rho).soil(i,j) - \rho.\frac{2.soil^{IWD}}{N_C\left(N_C - 1\right)} \quad \forall (i,j) \in T_M \tag{11}$$

11. 11. If the minimum tour T_M is shorter than the best tour found so far denoted by T_B, then we update the best tour by applying equation 12.

$$T_B = T_M \quad and \quad Len(T_B) = Len(T_M) \tag{12}$$

12. Go to step 2 unless the maximum number of iterations is reached or the defined termination condition is satisfied.
13. The algorithm stops here such that the best tour is kept in T_B and its length is $Len(T_B)$.

It is reminded that it is also possible to use only T_M and remove step 11 of the IWD algorithm. However, it is safer to keep the best tour T_B of all iterations than to count on only the minimum tour T_M of the last iteration. The IWD algorithm is experimented by artificial and some benchmark TSP environments. The proposed algorithm converges fast to optimum solutions and finds good and promising results. This research (Shah-Hosseini, 2007) is the beginning of using water drops ideas to solve engineering problems. So, there is much space to improve and develop the IWD algorithm.

6. Artificial immune systems

6.1 Introduction

Recently, there was an increasing interest in the area of Artificial Immune System (AIS) and its application for solving various problems specifically for the TSP (Zeng & Gu, 2007), (Lu

et al., 2007). AIS is inspired by natural immune mechanism and uses immunology idea in order to develop systems capable of performing different tasks in various areas of research such as pattern recognition, fault detection, diagnosis and a number of other fields including optimization. Here we want to know the AIS completely. To start with, it might be useful to become more familiar with natural immune system.
Natural immune systems consist of the structures and processes in the living body that provide a defence system against invaders and also altered internal cells which lead to disease. In a glance, immune system's main tasks can be divided into three parts; recognition, categorization and defence. As recognition part, the immune system firstly has to recognize the invader and foreign antigens e.g. bacteria, viruses and etc. After recognition, classification must be performed by immune systems, this is the second part. And appropriate form of defence must to be applied for every category of foreign aggressive phenomenon as the third part. The most significant aspect of the immune systems in mammals is learning capability. Namely, the immune systems can grow during the life time and is capable of using learning, memory and associative retrieval in order to solve mentioned recognition and classification tasks. In addition, the studies show that the natural immune systems are useful phenomena in information processing and can be helpful in inspiration for problem solving and various optimization problems (Keko et al., 2003).

6.2 Artificial immune system

Like the natural immune systems the AIS is a set of techniques, which try to algorithmically mimic natural immune systems' behaviour (Dasgupta, 99). As mentioned earlier, the immune system is susceptible to all of the invaders, also the outer influences, like vaccines which are artificial ways of raising individual's immunity. Vaccines are other factors that can stimulate the immune system's susceptibility. This feature is the key point of the AIS structure. The vaccines in the AIS are abstracted forms of the preceding information. Vaccination modifies genes based on the useful knowledge of the problem to achieve higher fitness in comparison to the fitness that obtained from a random process when for example a classical GA is applied. Once again it is necessary to point out that, vaccines contain some important information about the problem and in consequence the vaccination process employed in a right manner can be very useful in the performance of the algorithm. Like classical GA and based on its structure the AIS can work. The GA operators (crossover and mutation) search the problem space randomly and hence they don't have enough capability of meeting the actual problem at the local level. GAs are known as incapable of search fine local tuning because they are global search algorithms. Immune method through vaccination tries to overcome such blindness of crossover and mutation (Keko et al., 2003). After vaccination, the immune method might leads to deterioration. This case happens when vaccination leads to smaller fitness values than previous ones. Hence, another important part of immune algorithm is prevention of deterioration when inserting vaccine. In short, immune operators perform in four steps: firstly, an individual is selected, randomly. Now as the second step, the vaccine is inserted at the individual's randomly chosen place. Vaccine insertion might leads to deterioration, the third step is checking for deterioration. And finally the forth step is discarding every individual that shows degeneration right after vaccine. This way of checking could be dangerous for diversity and could result in algorithm's inability to avoid local optima, especially when combined with small populations. The studies show that the use of immune systems resulted in faster

convergence when population is large enough and diversity is secured. The combination of immune algorithm and GA, form the immune genetic algorithm (IGA). Many of previous works that are performed on the TSP used IGA. Now, we first investigate the IGA and its structure in detail and after that we have a look at some previous works around the TSP.
In summery, the IGA consists of these steps:

1. Creation of initial population in some way, e.g. through a random process
2. Abstract vaccines according to the former information
3. Checking the termination criterion (if it is satisfied go to step 10 and else go to next step)
4. Crossover on the randomly selected individuals
5. Mutation on the produced children
6. Vaccination on the former step outcome
7. Deterioration checking
8. Discarding every individual that shows degeneration right after vaccine
9. Go to step 3
10. End

As it is obvious from the ten steps which have been mentioned above, the IGA is very similar to the conventional GA, but they are different in operators. The IGA has vaccine operator to overcome the universality problem of the conventional GA. For more information on IGA you can refer to many cited papers which are prepared at the end of this chapter.

6.3 Solving the TSP using AIS and IGA

The first work in investigating potential application of the immune system in solving numerical optimization problems was the study by Bersini and Varela (Bersini & Varela, 90), who proposed immune employment mechanism. After that, many studies have been performed that focus on the AIS and IGA. Also, the IGA and AIS have been applied for solving the TSP in many cases. In (Jiao & Wang, 2000) the IGA and its parts have been introduced in detail and the IGA has been shown as an algorithm that accomplished in two steps: 1) a vaccination and 2) an immune selection. These phases are completely similar to that we mentioned about IGA and AIS in this section. In the mentioned paper, it is proved that the IGA theoretically converges with probability one. Besides, strategies and methods of selecting vaccines and constructing an immune operator are also given. Also, the IGA has been applied to the TSP and the results which are presented in this study illustrate that IGA is able to restrain the degenerate phenomenon effectively during the evolutionary process and can improve the searching ability, adaptability and greatly increase the converging speed. Recently, some works have been performed on the TSP which employ IGA. In (Zeng & Gu, 2007), a novel genetic algorithm based on immunity and growth for the TSP is presented. In this paper at first, a reversal exchange crossover and mutation operator is proposed which lead to preservation of the good sub-tours and making individuals various. At the next part, a new immune operator is proposed to restrain individuals' degeneracy. In addition, a novel growth operator is proposed to obtain the optimal solution with more chances. Results and investigations that performed in this study show that the algorithm is feasible and effective as it is claimed. In addition, in another study (Lu et al., 2007), a modified immune genetic algorithm is applied to solve the Travelling Salesman Problem. This method called an improved IGA by its authors. In this paper, at first, a new selection strategy is incorporated into the conventional genetic algorithm to improve the performance

of genetic algorithm. Besides the authors changed the selection strategy and in a new form it includes three computational procedures: evaluating the diversity of genes, calculating the percentage of genes, and computing the selection probability of genes. Based on the prepared results it is inferred that, by incorporating inoculating genes into conventional procedures of genetic algorithm, the number of evolutional iterations to reach an optimal solution can be significantly reduced and in consequence it results in faster answer in comparison to conventional IGA.

In addition to the mentioned works, the biological immune idea can be combined with other population based optimization algorithms which all of them are prepared in this chapter. As an instance, the paper (Qin et al., 2006) proposes a new diversity guaranteed ant colony algorithm by adopting the method of immune strategy to ant colony algorithm and simulating the behaviour of biological immune system. This method has been applied to the TSP benchmarks and results show that the presented algorithm has strong capability of optimization; it has diversified solutions, high convergence speed and succeeds in avoiding the stagnation and premature phenomena.

Based on the performed studies some points can be inferred as mentioned in the following (Keko et al., 2003):

The simulation results show that the variation in population size has the same effect on the GA and IGA. In both of the mentioned techniques, large population sizes require more generation to achieve higher fitness, resulting in relatively slow rate of convergence. Hence new ideas are required for faster convergence. Some of these new ideas had been presented in some works as you see in some investigated papers.

Also, based on the simulation results, the running time of the IGA and the regular GA do not have large differences, since in the IGA all the vaccines are determined before the algorithm starts and when they are required they can be loaded from a look up table.

Combining immune operator with another local improving operator can be an additional idea for getting better answers from the IGA.

One of the advantages of the IGA over the plain GA is that it is less susceptible to changing control parameters such as crossover or mutation probability. The simulation results demonstrate that changing these parameters has slight influence to the overall performance.

It is worth mentioning that more studies and attentions in the AIS and IGA are employing other AIS features like adaptive vaccine selection.

7. Bee colony optimization

7.1 Introduction

Similar to other natural inspired and collective intelligence based algorithms such as PSO which is taken from the bird's life and ACO based on the ant colony social life, another kind of artificial intelligence systems that can be useful in solving many engineering, management, control and computational problems, is an algorithm inspired from Bee colonies in nature. The Bee Colony Optimization (BCO) algorithms are interesting metaheuristic algorithms that represent another direction in the field of swarm intelligence. Here, firstly we introduce the bee system and bee colony optimization briefly and then some recent works on the TSP which have used bee systems are investigated.

7.2 Bee colony optimization

The bee colony's function according to nature is as follows. At first, each bee belonging to a colony looks for the feed individually. When a bee finds the feed, it informs other bees by