10

An Efficient Solving the Travelling Salesman Problem: Global Optimization of Neural Networks by Using Hybrid Method

Yong-Hyun Cho
A School of Computer and Information Comm. Eng., Catholic University of Daegu
330, Kunrakri, Hayangup, Gyeongsan, Gyeongbuk, 712-702
Korea(South)

1. Introduction

The travelling salesman problem (TSP) is a problem in combinatorial optimization studied in operations research and theoretical computer science. Given a list of cities and their pair-wise distances, the task is to find a shortest possible tour that visits each city exactly once (Aarts & Laarhoven, 1985; Beale & Jackson, 1990; Bout & Miller, 1988; Cichock & Unbehaun, 1993; Lin, 1965; Zurada, 1992). 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 (Beale & Jackson, 1990; Freeman & Skapura, 1991; Lin, 1965).

The TSP has several applications even in its purest formulation, such as planning, logistics, and the manufacture of microchips. 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 (Beale & Jackson, 1990; Cichock & Unbehaun, 1993; Freeman & Skapura, 1991; Zurada, 1992). 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 TSP belongs to the class of NP-complete problems (Aarts & Laarhoven, 1985; Abe et al., 1992; Burke, 1994; Freeman & Skapura, 1991; Hopfield & Tank, 1985). Thus, it is assumed that there is no efficient algorithm for solving TSPs. In other words, it is likely that the worst case running time for any algorithm for TSP increases exponentially with the number of cities, so even some instances with only hundreds of cities will take many CPU years to solve exactly. The travelling salesman problem is regarded as difficult to solve. If there is a way to break this problem into smaller component problems, the components will be at least as complex as the original one. This is what computer scientists call NP-hard problems (Aarts & Laarhoven, 1985; Abe et al., 1992; Freeman & Skapura, 1991).

Many people have studied this problem. The easiest (and most expensive solution) is to simply try all possibilities. The problem with this is that for *n* cities you have *(n-1)!* possibilities. This means that for only 11 cities there are about 3.5 million combinations to try (Freeman & Skapura, 1991). In recent years, many algorithms for solving the TSP have been proposed (Cichock & Unbehaun, 1993; Dorigo et al., 1991; Goldberg, 1989; Lin & Kernighan, 1971; Mascato, 1989; Szu & Hartley, 1987). However, these algorithms sustain several disadvantages. First, some of these algorithms are not optimal in a way that the solution they obtain may not be the best one. Second, their runtime is not always defined in advance, since for every problem there are certain cases for which the computation time is very long due to unsuccessful attempts for optimization. They will often consistently find good solutions to the problem. These good solutions are typically considered to be good enough simply because they are the best that can be found in a reasonable amount of time. Therefore, optimization often takes the role of finding the best solution possible in a reasonable amount of time. There have been several types of approaches taken to solving the TSP [10-30] of the numerical methods and the neural networks (NNs) (Beale & Jackson, 1990; Cichock & Unbehaun, 1993; Freeman & Skapura, 1991; Goldberg, 1989; Zurada, 1992). Recently, NN is well suited for this type of problems.

An NN, also known as a parallel distributed processing network, is a computing paradigm that is loosely modeled after cortical structures of the brain (Beale & Jackson, 1990; Cichock & Unbehaun, 1993; Freeman & Skapura, 1991; Zurada, 1992). It consists of interconnected processing elements called nodes or neurons (Beale & Jackson, 1990; Zurada, 1992). NN, due to its massive parallelism, has been rigorously studied as an alternative to the conventional numerical approach for fast solving of the combinatorial optimization or the pattern recognition problems. The optimization is to find the neuron that lead to the energy minimum by applying repeatedly the optimization algorithm.

Hopfield model is energy-minimizing network, and is useful as a content addressable memory or an analog computer for solving combinatorial optimization problems (Abe et al., 1992; Abe, 1993, 1996; Aiyer et al., 1990; Andresol et al., 1997; Gall & Zissimopoulos 1999; Hegde et al., 1988; Sharbaro, 1994; Wilson & Pawley, 1988). Generally, Hopfield model may be operated in a discrete-time mode and continuous-time mode, depending on the model adopted for describing the neurons. The discrete-time mode is useful as a content addressable memory, and the continuous-time mode is also useful as an analog computer for solving combinatorial optimization problems. In formulating the energy function for a continuous-time Hopfield model, the neurons are permitted to have self-feedback loops. On the other words, a discrete-time Hopfield model is no self-feedback loops (Beale & Jackson, 1990; Cichock & Unbehaun, 1993; Freeman & Skapura, 1991).

Gradient-type NNs are generalized Hopfield model in which the computational energy decreases continuously in time loops (Beale & Jackson, 1990; Cichock & Unbehaun, 1993; Freeman & Skapura, 1991).. The continuous-time model is called the gradient-type model and converges to one of the stable minima in the state space. The evaluation of model is in the general direction of the negative gradient of energy function. Typically, the energy function is made equivalent to a certain objective function that needs to be minimized. The search for an energy minimum performed by gradient-type model corresponds to the search for a solution of an optimization problem loops (Beale & Jackson, 1990; Cichock & Unbehaun, 1993; Freeman & Skapura, 1991).

The major drawbacks of the continuous-time Hopfield model when it is used to solve some combinatorial problems, for instance, the TSP, are the non feasibility of the obtained solutions and the trial-and-error setting of the model parameters loops (Beale & Jackson, 1990; Bout & Miller, 1988; Cichock & Unbehaun, 1993; Freeman & Skapura, 1991). Most of the researches have been concentrated on the improvement of either the convergence speed or the convergence rate to the global minimum in consideration of the weight parameter of the energy function, etc. But there are few that try to solve both global convergence and speedup by simultaneously setting the initial neuron outputs (Baba, 1989; Biro et al., 1996; Gall & Zissimopoulos 1999; Gee & Prager, 1995; Heung, 2005).

This chapter proposes an efficient method for improving the convergence performances of the NN by applying a global optimization method. The global optimization method is a hybrid of a stochastic approximation (SA) (Styblinski & Tang, 1990) and a gradient descent method. The approximation value inclined toward a global escaping from a local minimum is estimated first by the stochastic approximation, and then the gradient-type update rule of Hopfield model is applied for high-speed convergence. The proposed method has been applied to the *7*- and *10*-city TSPs, respectively. We demonstrate the convergence performance to the conventional Hopfield model with randomized initial neuron outputs setting.

The rest of the chapter is organized as follows. The travelling salesman problem is introduced in section 2. Section 3 presents the Hopfield model for solving the TSP. Section 4 presents the method for estimating an initial value of optimization problems by using stochastic approximation. Section 5 describes how the proposed method can be applied for globally optimizing the neural network. Section 6 describes the experiments with the proposed global optimization method focusing on the TSP. The performance comparison of the experiment results with the Hopfield model is also given. Finally an outlook to future research activities is presented.

2. Travelling salesman problem

Generally, the optimization problems are typically posed in terms of finding the best way to do something, subject to certain constraints. When solving these problems with computers, often the only possible approach is to calculate every possible solution and then choose the best of those as the answer. Unfortunately, some problems have such large solution spaces that this is impossible to do. These are problems where the solution cannot be found in a reasonable time. These problems are referred to as NP-hard or NP-complete problems. In many cases, these problems are described in term of a cost function (Aarts & Laarhoven, 1985; Beale & Jackson, 1990; Cichock & Unbehaun, 1993; Freeman & Skapura, 1991).

One such problem is the TSP. The TSP describes a salesman who must travel between cities. The order in which he does so is unimportant, provided he visits each one during his trip, and finishes in his starting location. Each city is connected to other close by cities, or nodes. Each of those links between the cities has one or more weights (cost) attached. The cost describes how "difficult" it is to traverse this edge on the graph, and may be given, for example, by the cost of an airplane ticket or train ticket, or perhaps by the length of the edge, or time required for completing the traversal (Beale & Jackson, 1990; Bout & Miller, 1988; Cichock & Unbehaun, 1993; Lin, 1965; Zurada, 1992). The salesman wants to keep both the travel costs, as well as the distance he travels as low as possible. That is, the problem is to find the right sequence of cities to visit. The constraints are that all cities are visited, each is

visited only once, and the salesman returns to the starting city at the end of the travel. The cost function to be minimized is the total distance or cost travelled in the course of the travel.

The TSP is computationally intensive if an exhaustive search is to be performed comparing all possible routes to find the best one (Freeman & Skapura, 1991). For an n-city trip, there are n! possible paths. Due to degeneracy, the number of distinct solutions is less than n!. The term distinct in this case refers to trips with different total distances. For a given trip, it does not matter which of the n cities is the starting location, in terms of the total distance traveled. This degeneracy reduces the number of distinct tours by a factor of n. Similarly, for a given trip, it does not also matter which of two directions the salesman travels. This fact further reduces the number of distinct trips by a factor of two. Thus, for n-city trip, there are n!/2n distinct tours to consider.

For a 5-city trip, there would be 120!/10=12 distinct trips-hardly a problem worthy of solution by a computer! A 10-city trip, however, has 3,628,800/20=181,440 distinct trips; a 30-city trip has over $4x10^{30}$ possibilities. Adding a single city to a trip results in an increase in the number of distinct trips by a factor of n. Thus, a 31-city trip requires that we examine 31 times as many trips as we must for a 30-city trip. The amount of computation time required by a digital computer to solve this problem grows exponentially with the number of cities.

There have been many approaches to solving the Traveling Salesman Problem. These approaches range from a simple heuristic algorithm to algorithms based on the physical workings of the human mind to those based on ant colonies (Andresol et al., 1997; Dorigo et al., 1991; Dorigo & Gambardella, 1997; Lin & Kernighan, 1971; Mascato, 1989; Szu & Hartley, 1987). These algorithms all have the same ultimate goal: in a graph with weighted edges, find the shortest Hamiltonian path (the path through all nodes with the smallest sum of edge weights). Unfortunately, this goal is very hard to achieve. The algorithms therefore settle for trying to accomplish two smaller goals: (1) to more quickly find a good solution and (2) to find a better good solution. A good solution is one that is close to being optimal and the best of these good solutions is, of course, the optimal solution itself. There have been several types of approaches taken to solving the TSP. They include heuristic approaches, memetic algorithms, ant colony algorithms, simulated annealing, genetic algorithms, neural networks, and various other methods for more specific variations of the TSP (Abe, 1993; Andresol et al., 1997; Dorigo et al., 1991; Dorigo & Gambardella, 1997; Lin & Kernighan, 1971; Mascato, 1989; Szu & Hartley, 1987; Xavier & Suykens, 2006; Mühlenbein, 1992).

These approaches do not always find the true optimal solution. Instead, they will often consistently find good solutions to the problem. These good solutions are typically considered to be good enough simply because they are the best that can be found in a reasonable amount of time. Therefore, optimization often takes the role of finding the best solution possible in a reasonable amount of time.

2.1 Heuristic algorithms

The heuristic means that a rule of thumb, simplification or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood (Lin & Kernighan, 1971). Unlike algorithms, heuristics do not guarantee optimal, or even feasible, solutions and are often used with no theoretical guarantee. In contrast, an algorithm is defined as "a precise rule (or set of rules) specifying how to solve some problem" (Andresol

et al., 1997; Lin & Kernighan, 1971). To combine these together into a heuristic algorithm, we would have something like "a set of rules specifying how to solve some problem by applying a simplification that reduces the amount of solutions checked". In other words, the algorithm is the instructions for choosing the correct solution to the problem while the heuristic is the idea of how to shrink the list of possible solutions down to a reasonable size.
An example of a heuristic approach to the TSP might be to remove the most weighted edge from each node to reduce the size of the problem (Lin & Kernighan, 1971). The programmer in this situation may assume that the best solution would not have the most weighted edge. Upon close inspection, this heuristic may not actually give the best solution, maybe not even a feasible solution (if all of the most weighted edges from each node are connected with the same node) but it may be a calculated risk that the programmer takes (Lin & Kernighan, 1971).
The main idea of a heuristic approach to a problem is that, although there is exponential growth in the number of possible solutions to the problem, evaluating how good a solution is can be done in polynomial time.
In dealing with the TSP, the most common uses of heuristic ideas work with a local search. Similarly to the above example, the heuristic does not try to encompass every possibility of the problem at hand; instead it attempts to apply common sense to shrink the problem to a manageable size.
Perhaps the most-used local search heuristic that is applied to the TSP is the *n*-opt method developed by Lin and Kernighan (Andresol et al., 1997; Lin & Kernighan, 1971). It simply takes a random path and replaces *n* edges in it until it finds the best of those paths. This is typically done where n is set to *2* or *3* (Lin & Kernighan, 1971). These methods were applied to several different problems. Notably, they were able to find the optimal solutions for a *42*-city problem *4* out of *10* times and the optimal solution to a *48*-city problem *2* out of *10* times (Lin & Kernighan, 1971) (the *10* times in these were running concurrently so the optimum solution was found in each run of the program).

2.2 Simulated annealing

Simulated annealing is a method that is based on the cooling of a physical system (Kawabe et al., 2002; Szu & Hartley, 1987; Xavier et al., 2006). The general idea is that there is a temperature (T) and a cost function (H). In our case, the cost function is the sum of the weights of the edges in our circuit. In the beginning, there is a random solution to the problem. At each iteration, a change is proposed to this solution and that change is evaluated based on the cost function and the temperature. If the cost function decreases then the change is accepted. If the cost function does not decrease then the change is accepted or rejected based on the temperature. The higher the temperature, the better the chance that the change will be accepted. As time progresses, the temperature decreases and eventually there is no possibility for a change to occur without the cost function decreasing. Using this method, researchers were able to get to within two units of the optimal cost for problems up to a size of *100* (Xavier et al., 2006).

2.3 Neural networks

A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use (Beale & Jackson, 1990; Bout & Miller, 1988; Cichock & Unbehaun, 1993; Freeman & Skapura, 1991; Lin, 1965; Zurada, 1992). It resembles the brain in two respects. One is that knowledge is acquired by

the network through a learning process. Another is that interneuron connection strengths known as synaptic weights are used to store the knowledge. Basically, a neural network is made up of many independent units (neurons) and connections between them. The connections are given various weights based on a "learning process". Based on the sum of the products of adjoining neurons and the weights of the connecting edges, each neuron finds a value. Additionally, if the value of one neuron changes then the values of all the adjoining neurons also change. This creates a ripple effect that can change the values of every neuron (although it could also change none of them).
An NN can be applied to a TSP with n cities (Beale & Jackson, 1990; Bout & Miller, 1988; Cichock & Unbehaun, 1993; Freeman & Skapura, 1991; Lin, 1965; Zurada, 1992). This is done by creating n^2 neurons. The output of each neuron ($V_{x,i}$) represents whether city x is visited as the i-th city in the sequence. It is a *1* if this is true or a *0* if it is not. Additionally, the amount d_{xy} is applied to the calculations as the distance between cities x and y.

2.4 Genetic algorithm

A genetic algorithm (GA) is based on the same idea as the theory of evolution (Goldberg, 1989; Mühlenbein, 1992). Basically, several random sets of parameters are applied to an algorithm and a fitness value is returned for each. Based on these fitness values, the best sets are mixed together and new sets are again applied to the algorithm until an optimal set of parameters is obtained. This effect is usually obtained by breaking the genetic algorithm into a few small parts. The main parts are the fitness function and the evolution function (Goldberg, 1989).
The evolution function produces a string of inputs (often a string of bits that are encodings of the input parameters) then asks the fitness function for a fitness value for that string. When several strings have been assigned a fitness value, the evolution function takes the best strings, mixes them together, sometimes throws in a "mutation" to the strings and then sends the results back as new input strings. The biological analogy is to a human's genes. In fact, an input string is often called a chromosome and the bits in the string are referred to as genes.
The fitness function of a genetic algorithm takes in a string of inputs and runs them through the process that is being evaluated (Mühlenbein, 1992). Based on the performance of the inputs, the function returns a fitness value. In the case of the TSP, the fitness function returned the total length or weight of the path found. A GA has two main parts, an evolution function and a fitness function. In the case of the TSP, the parameters produced by the evolution function might be the order of the nodes through which the path will go. The fitness function in that same case would return the total length of the path found. The GA would then compare fitness values for each input string and assign priority to the ones that returns lower path lengths. Genetic algorithms and their applications to the TSP are described by Goldberg (Goldberg, 1989).

2.5 Memetic algorithms

A memetic algorithm (MA) is really a combination of several different techniques (Mascato, 1989). Generally, an MA can be thought of as an algorithm that combines local search heuristics with a crossover operator (the same type of mixing and matching that happens with a GA's evolution function). Despite this, the difference between an MA and a GA is very distinct. As opposed to the fitness functions of GAs, MAs use a local search heuristic to determine how the parameter definitions will be modified each iteration. For example, an

MA might use simulated annealing to find a solution with some parameters and return that value to the crossover operator just like a GA would return a value from a fitness function. For this reason there are many other terms used to refer to MAs including hybrid genetic algorithms, parallel genetic algorithms, and genetic local search algorithms. The research in MAs was most notably conducted by Mascato (Mascato, 1989). Researchers such as Mühlenbein have shown MAs to be near-optimal with sizes at least as large as a *200*-city problem.

2.6 Ant colony algorithms

Ant-based algorithms are based on studies of ant colonies in nature (Dorigo et al., 1991; Dorigo & Gambardella, 1997). The main idea in these algorithms is that the behavior of each individual ant produces an emergent behavior in the colony. When applied to the TSP, individual agents ("ants") traverse the graph of the problem, leaving a chemical (pheromone) trail behind them. At each node it comes to, an ant must decide which edge to take to the next node. This is done by checking each edge for pheromone concentration and applying a probability function to the decision of which edge to choose. The higher the concentration of pheromone, the more likely the ant is to choose that edge. Also, to avoid stagnation in travel, the pheromone is given an evaporation rate so that in each iteration the pheromone loses a certain percentage on each edge. This method was researched originally by Dorigo, et al. (Dorigo et al., 1991). This method has been shown to do better than other algorithms on random *50*-city problems as well as finding the optimum solutions for problems with up to *100* cities (Dorigo & Gambardella, 1997).

3. Hopfield model for solving TSP information

In the most general case, NNs consist of a (often very high) number of neurons, each of which has a number of inputs, which are mapped via a relatively simple function to its output (Beale & Jackson, 1990; Bout & Miller, 1988; Cichock & Unbehaun, 1993; Freeman & Skapura, 1991; Lin, 1965; Zurada, 1992). Networks differ in the way their neurons are interconnected (topology), in the way the output of a neuron determined out of its inputs(propagation function) and in their temporal behavior(synchronous, asynchronous or continuous).

Ever since Hopfield and Tank (Hopfield & Tank, 1985) showed that the feedback neural network could be possibly used to solve combinatorial optimization problems such as the TSP, great efforts have been made to improve the performance. Most of the early work focused on ways to find valid solutions because of the disappointing results reported (Freeman & Skapura, 1991). While some researchers tried to find more appropriate parameters in the energy function (Aiyer et al., 1990; Baba, 1989; Biro et al., 1996; Burke, 1994; Gall & Zisssimopoulos, 1999; Gee & Prager, 1995; Hegde et al., 1988; Hopfield & Tank, 1985; Huang, 2005; Sharbaro, 1994; Wilson & Pawley, 1988), others hoped to get better energy functions (Baba, 1989). To date, research work has been extended to every aspect of the Hopfield model (Aarts & Laarhoven, 1985; Abe et al., 1992; Aiyer et al., 1990; Baba, 1989; Biro et al., 1996; Burke, 1994; Hegde et al., 1988; Hopfield & Tank, 1985; Sharbaro, 1994; Wilson & Pawley, 1988), and it is now clear how to correctly map problems onto the network so that invalid solutions never emerge. As for the quality of obtained solutions, while there are indications that the Hopfield model is solely suitable for solving Euclidean

TSPs of small size (Wilson & Pawley, 1988), some researchers argue it is unreasonable to take the TSP as the benchmark to measure the optimization ability of the Hopfield model (Sharbaro, 1994). According to that, the applicability of the Hopfield model to solve other optimization problems should not be ignored. By now, the Hopfield model has been successfully applied to many fields.

A key issue in the application of the Hopfield model is the choice of the weights1 in the energy function. Previous work addressing this problem is only concerned with the occasion for solving the TSP. The most successful and earliest work was conducted by Aiyer et al. (Aiyer et al., 1990). Using the eigenvalue analysis, they obtained values for the weights which make the network converge to very good solutions. Other works concerning on this problem include the technique of suppressing spurious states (Abe, 1993).

3.1 Hopfield model

Hopfield described a new way of modeling a system of neurons capable of performing computational tasks (Cichock & Unbehaun, 1993; Zurada, 1992). The Hopfield model emerged, initially as a means of exhibiting a content addressable memory (CAM). A general CAM must be capable of retrieving a complete item from the system's memory when presented with only sufficient partial information. Hopfield showed that his model was not only capable of correctly yielding an entire memory from any portion of sufficient size, but also included some capacity for generalization, familiarity recognition, categorization, error correction, and time-sequence retention (Hopfield & Tank, 1985).

The Hopfield model, as described in (Beale & Jackson, 1990; Zurada, 1992), comprises a fully interconnected system of n computational elements or neurons. Fig. 1 is a model of artificial neural network.

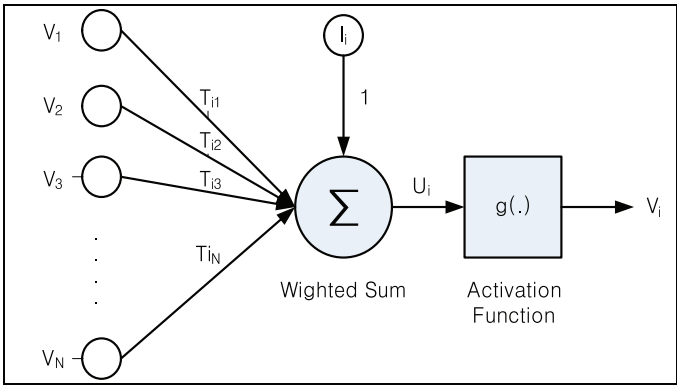


Fig. 1. Model of artificial neural network

In Fig. 1, Hopfield's original notation has been altered where necessary for consistency. The strength of the connection, or weight, between neuron i and neuron j is determined by T_{ij}, which may be positive or negative depending on whether the neurons act in an excitatory or inhibitory manner (Freeman & Skapura, 1991). The internal state of each neuron U_i is equivalent to the weighted sum of the external states of all connecting neurons. The external state of neuron i is given by Vi, with $0 \leq V_i \leq 1$. An external input, Ii, to each neuron i is also incorporated. The relationship between the internal state of a neuron and its output level in

this continuous Hopfield model is determined by an activation function $g_i(U_i)$, which is bounded below by 0 and above by 1.
Then the dynamics of each neuron can be given by a system of the differential eq. (1).

$$\frac{dU_i}{dt} = \sum_j T_{ij} V_j - \frac{U_i}{\tau} + I_i, \quad U_i = g_i^{-1}(V_i) \tag{1}$$

Where is a time constant, *Ii* is the external input(bias) of neuron *i*, and V_i and U_i are the output and input of neuron *i*. The relation g_i between the input U_i and the output V_i is characterized by a monotonically increasing function such as a sigmoid, or a piecewise linear function.
Hopfield model is a dynamic network (Beale & Jackson, 1990; Zurada, 1992), which iterates to converge from an arbitrary input state. The Hopfield model works as minimizing an energy function. The Hopfield model is single layer network which are fully interconnected. It is a weighted network where the output of the network is fed back and there are weights to each of this link. The fully connected Hopfield model is shown in following Fig. 2.

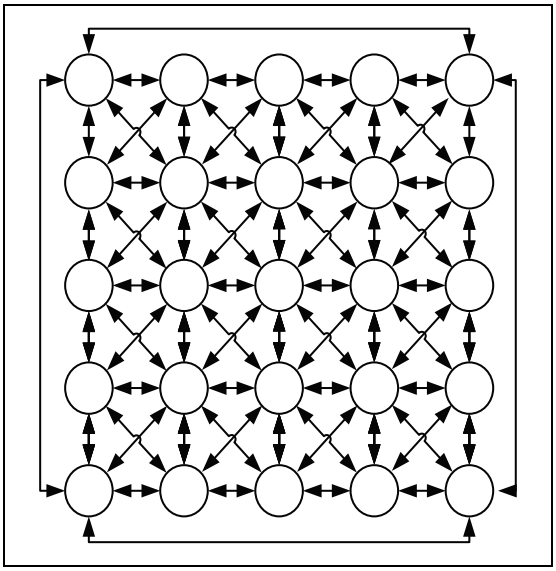


Fig. 2. Fully connected Hopfield model for 5-city TSP

As long as the neuron has a sufficiently high gain, the first term in (1) can be neglected. In that case, the Hopfield model has the Lyapunov energy function of eq. (2)

$$E = -\frac{1}{2}\sum_i \sum_j V_i T_{ij} V_j - \sum_i I_i V_i \tag{2}$$

And moreover we may note the following relations hold:

$$\frac{du_i}{dt} = -\frac{\partial E}{\partial V_i} \; and \; \frac{dE}{dt} \le 0 \tag{3}$$

This means that the energy function monotonically decreases with the evolution of the network's state, and when the network reaches the final stable state, the energy function falls into a local minimum. The general method of applying the Hopfield model to solve optimization problems is to map the objectives and constraints involved in the problem into an energy function, and then obtain the neuron's dynamic equation by means of eq. (3).

3.2 Hopfield model to solve the TSP

The TSP is concerned with how to find a shortest closed path that travels each of n cities exactly once. In terms of the geometric structure of the distribution of the cities and the symmetry of distances between a pair of cities, the TSP can be classified into several categories (Aiyer et al., 1990; Baba, 1989; Biro et al., 1996; Burke, 1994; Gee & Prager, 1995; Hegde et al., 1988; Sharbaro, 1994; Wilson & Pawley, 1988).

The Hopfield model for the TSP is built of $n * n$ neurons. The network consists of n rows, containing n neurons according to Fig. 3.

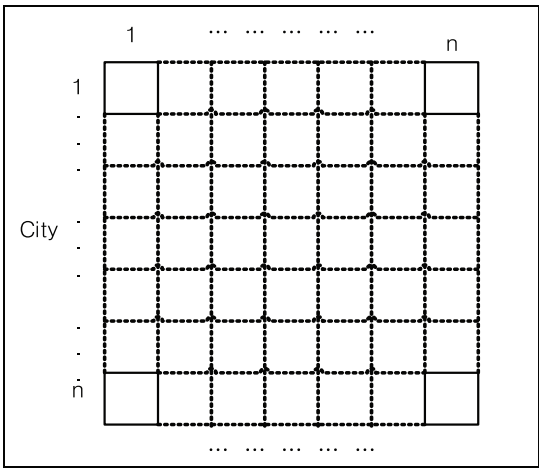


Fig. 3. The division of the network

All neurons have two subscripts. The first one defines the city number and the second one the position of the city in the tour. If a neuron in the stable state of the network, has the output signal $V_{x,i} = 1$, then it means that the city x should be visited in the stage i of the tour (Beale & Jackson, 1990; Zurada, 1992).

The energy function for mapping the TSP proposed by Hopfield is described by eq. (4)

$$E = \frac{A}{2}\sum_x\sum_i\sum_{j\neq i} V_{x,i}V_{x,j} + \frac{B}{2}\sum_i\sum_x\sum_{y\neq x} V_{x,i}V_{x,i} + \frac{C}{2}\left(\sum_x\sum_i V_{x,i} - n\right)^2 + \frac{D}{2}\sum_x\sum_{y\neq x}\sum_i d_{x,y}V_{x,i}(V_{y,i-1} + V_{y,i+1}) \tag{4}$$

Where $d_{x,y}$is the distance from city x to city y, and the scaling parameters A, B, C, D are positive constants. The first and second term represents the constraint that at most one neuron of the array V is on fire at each row and column, respectively. The third term

represents the constraint that the total number of neurons on fire is exactly n. The fourth term measures the tour length corresponding to a given tour, where the two terms inside the parenthesis stand for two neighboring visiting cities of $V_{x,i}$, implying the tour length is calculated twice. The energy function reaches a local minimum when the network is at a valid tour state.

With this formulation, the Hopfield model has the connection strengths and external input given as eq. (5) and eq. (6)

$$T_{xi,yi} = -\left\{ A\delta_{x,y}(1-\delta_{i.j}) + B(1-\delta_{x,y})\delta_{i,j} + C + D(\delta_{i,j-1} + \delta_{i,j+1})d_{x,y} \right\} \tag{5}$$

$$I_{x,i} = Cn \tag{6}$$

Where $\delta_{i,j}$ is equal to *1* ($i \neq j$) or *0* (otherwise).

It is known that the Hopfield model formulation does not work well for the TSP since the network often converges to infeasible solutions. It has been widely recognized that the formulation is not ideal, even for problems other than the TSP. The nature of the energy function that the method utilizes causes infeasible solutions to occur most of the time. A number of penalty parameters which are an initial values of weights and neurons and the activation function, need to be fixed before each simulation of the network, yet the values of these parameters that will enable the network to generate valid solutions are unknown. The problem of optimally selecting these parameters is not trivial, and much work has been done to try to facilitate this process (Abe, 1993, 1996; Hegde et al., 1988; Lai & Coghill, 1994). Many other researchers believed that the Hopfield model's energy function needed to be modified before any progress would be made, and considerable effort has also been spent in this area.

4. Initial value estimation by stochastic approximation

We consider the following problem of global unconstrained optimization: minimize the multiextremal function $f(x) \in \boldsymbol{R}^{\mathrm{i}}$, $\mathrm{x} \in \boldsymbol{R}^{\mathrm{n}}$, i.e.

$$\min_{x \in R^n} f(x) \tag{7}$$

A multiextremal function can be represented as a superposition of uniextremal function(i.e., having just one minimum) and other multiextremal function that add some noise to the uniextremal function. The objective of smoothing can be visualized as filtering out the noise and performing minimization on the smoothed uniextremal function, in order to reach the global minimum. In general, since the minimum of the smoothed uniextremal function does not coincide with the global function minimum, a sequence of minimization runs is required to zero in the neighborhood of global minimum (Styblinski & Tang, 1990). The smoothing process is performed by averaging $f(x)$ over some region of the parameter space $\boldsymbol{R}^n$ using a proper weighting (or smoothing) function $h\hat{}(x)$.

Let us introduce a vector of random perturbations $\eta \in \boldsymbol{R}^n$, and add η to x. The convolution function $f\tilde{}(x, \beta)$ is created as follows (Styblinski & Tang, 1990).

$$\tilde{f}(x,\beta) = \int_{R^n} h\hat{}(\eta,\beta) f(x-\eta) d\eta \tag{8}$$

Hence:

$$\tilde{f}(x,\beta) = E_\eta[f(x-\eta)] \tag{9}$$

Where $\tilde{f}(x, \beta)$ is the smoothed approximation to original function $f(x)$, and the kernel function $h\hat{}(\eta, \beta)$ is the probability density function(*pdf*) used to sample η. β controls the dispersion of $h\hat{}(\eta, \beta)$, i.e. the degree of $f(x)$, smoothing (Styblinski & Tang, 1990).
Note that $\tilde{f}(x, \beta)$ can be regarded as an averaged version of $f(x)$, weighted by $h\hat{}(\eta, \beta)$. $E_\eta[f(x-\eta)]$ is the expectation with respect to the random variable η.
Therefore an unbiased estimator $\tilde{f}(x, \beta)$ is the average:

$$\tilde{f}(x,\beta) = \frac{1}{N}\sum_{i=1} E_\eta[f(x-\eta^i)] \tag{10}$$

Where η is sampled with the *pdf* $h\hat{}(\eta, \beta)$.
The kernel function $h\hat{}(\eta, \beta)$ should have the following properties (Styblinski & Tang, 1990):

$$h\hat{}(\eta,\beta) = \left(\frac{1}{\beta^n}\right) \tag{11}$$

is piecewise differentiable with respect to η.

- $lim_{\beta\to 0}\, h\hat{}(\eta, \beta) = \delta(\eta)$ (Dirac delta function)
- $h\hat{}(\eta, \beta)$ is a pdf.

Under above the conditions, $lim_{\beta\to 0}\, \tilde{f}(x, \beta) = \int_{R^n} \delta(\eta)\, f(x - \eta)\, d\eta = f(x - 0) = f(x)$. Several *pdf*s fulfill above conditions, such as Gaussian, uniform, and Cauchy *pdf*s.
Smoothing is able to eliminate the local minima of $\tilde{f}(x, \beta)$, if β is sufficiently large. If $\beta \to 0$, then $\tilde{f}(x, \beta) \to f(x)$. This should actually happen at the end of optimization to provide convergence to the true function minimum (Styblinski & Tang, 1990). Formally, the optimization problem can be written as:

$$\min_{x\in R^n} \tilde{f}(x,\beta) \tag{12}$$

with $\beta \to 0$ as $x \to x^*$. Where x^* is the global minimum of original function $f(x)$. One class of methods to solve the modified problem Eq. (12), to be called large sample(LS) stochastic methods, can be characterized as follows: for each new point x, a large number of points sampled with the *pdf* $h\hat{}(\eta, \beta)$ (Eq. (11)) is used to estimate $\tilde{f}(x, \beta)$ and its gradient $\nabla_x \tilde{f}(x, \beta)$. The number of samples used should be sufficiently large to give small errors of the relevant estimators. Optimization and averaging are separated in LS methods. This is very inefficient (Styblinski & Tang, 1990).
Optimization and averaging can be combined into one iterative process, leading to much more efficient small-sample (SS) methods of stochastic programming. A large class of SS methods, called stochastic approximation, is applied to the function minimization or maximization (Styblinski & Tang, 1990). Their basic principle of operation is that only a small number of samples are used in each iteration to find the necessary estimators, but all the information is averaged over many steps.
In function minimization, SA methods create stochastic equivalent to the gradient methods of nonlinear programming. The advanced algorithms are proposed to estimate the gradient

$\nabla_x \tilde{f}(x, \beta)$. As the algorithm progresses, $\beta \to 0$, reducing the smoothing degree of the $f(x)$, and providing convergence to the true minimum. The SA algorithm implements a well-defined approximation to the conjugate gradient. The value x based on the smoothed function $\tilde{f}(x, \beta)$ is updated, as following,

$$\begin{aligned} \xi_k &= \nabla_x \tilde{f}(x_k, \beta) \\ S_k &= STEP \;/\; ||\xi_k|| \\ d_k &= (1-\rho_k) d_{k-1} + \rho_k \xi_k \quad (0 \le \rho_k \le 1) \\ \rho_k &= (1-\rho_{k-1}) \;/\; (1+\rho_{k-1} - R) \\ x_{k+1} &= x_k - S_k d_k \end{aligned} \tag{13}$$

Where $k(1,2,\ldots,MAXITER)$ is the number of iterations, ξ is the gradient, S is a step size, d is the search direction, ρ is the gradient averaging coefficient of $\tilde{f}(x, \beta)$, and $R(0<R<1)$ is a constant controlling the rate of change of ρ_k. Therefore, we can find the global minimum of original function by iteratively performing one cycle of the SA optimization as $\beta \to 0$. This is called the stochastic approximation with smoothing (SAS) (Styblinski & Tang, 1990).
Fig. 4 is the flow chart of SA algorithm. In this Fig. 4, each new value x is performed in the direction $S_k d_k$, where d_k is a convex combination of the previous direction d_{k-1} and a new gradient ξ_k. Especially R is responsible for the rate of change of ρ_k, that is, it modifies the search direction d_k and provides a suitable amount of inertia of gradient direction.

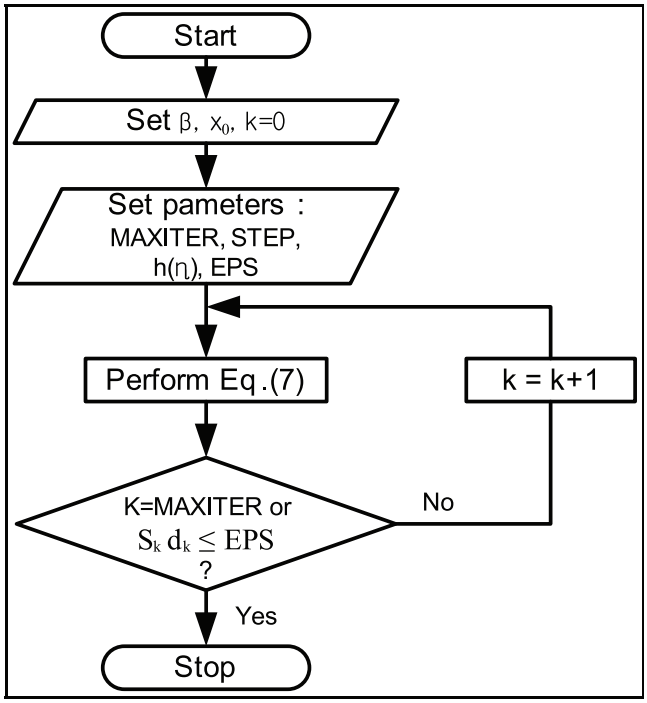


Fig. 4. Flowchart of stochastic approximation

Fig. 5 is the flow charts of SAS algorithm that repeatedly performs the SA algorithm according to a sequence: $\{\beta_0, \beta_1, ...\} \rightarrow 0$. We can get the global minimum by using the SAS algorithm based on specific sequences $\{\beta\}$ and $\{NMAX\}$. It turned out that the final solutions were not very sensitive to a specific choice of theses sequences based on rough heuristic criteria such as: low problem dimensionality requires a smaller number of function evaluations, β should be large at the beginning of optimization (to determine the approximate position of the global minimum), and small at the end of optimization (for precision) (Styblinski & Tang, 1990).

We consider the function $f(x) = x^4 - 16x^2 + 5x$ as an example (Styblinski & Tang, 1990). This function is continuous and differentiable, and it has two distinct minima as shown in Fig. 6. The smoothed $f^-(x, \beta)$ is plotted to different values of $\beta \rightarrow 0(\{5, 4, 3, 2, 1, 0.001, 0.0\})$ and $MAXITER=100$ for uniform *pdf*. We can show that minimize the smoothed function $f^-(x, \beta)$ with $\beta \rightarrow 0$ as $x \rightarrow x^*$.

As shown in Fig. 6, the smoothed functional $f^-(x, \beta)$ is an uniextremal function having one minimum x_1 from $\beta = 5$ to $\beta = 3$. That is, smoothing is able to eliminate the local minima of $f^-(x, \beta)$, if β is sufficiently large. If $\beta \rightarrow 0$, then $f^-(x, \beta) = f(x)$. We can also find out that the minimum x_1 of uniextremal function inclines toward the global minimum x^* of the original function $f(x)$ in Fig. 6.

On the other hand, the simulated annealing is often explained in terms of the energy that particle has at any given temperature (Kwabe et al., 2002; Szu & Hartley, 1987; Xavier et al., 2006). A similar explanation can be given to the smoothed approximation approach discussed. Perturbing x can be viewed as adding some random energy to a particle which x

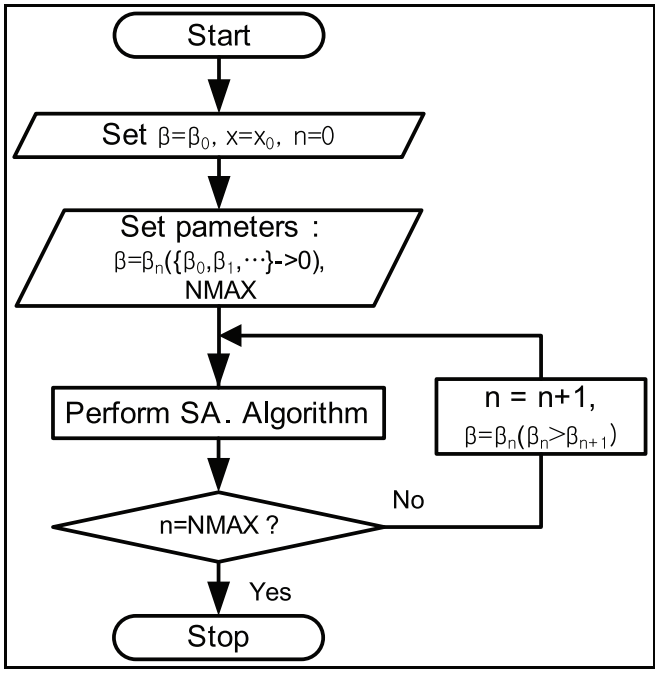


Fig. 5. Flowchart of stochastic approximation with smoothing

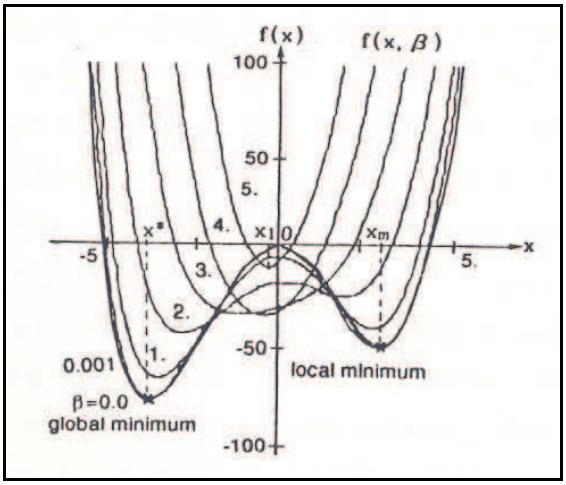


Fig. 6. Smoothed function $f^{-}(x, \beta)$ to β values

represents. The larger the β, the larger the energy (i.e., the larger the temperature in the simulated annealing), and also the broader the range of x changes. Reducing β for the smoothed approximation corresponds to temperature reduction in the simulated annealing.
Although the global minimum can be found by repeatedly applying SA according to a sequence: $\{\beta_0, \beta_1, ...\} \rightarrow 0$, there are a few problems as follows: a specific sequences and a parameters should be determined heuristically in each iterations, and, due to its stochastic process, its convergence speed is rather slower than that of the deterministic algorithm and sometimes results in approximate solution.
For this reason, SAS is the stochastic algorithm as the simulated annealing. The stochastic algorithms guarantee that converges to the global minimum, but their convergence speed is lower than that of the deterministic algorithms. In order to solve the limitation of convergence speed, we present a new optimization method that combines advantages of both the stochastic algorithm and the deterministic algorithm. That is, we propose a hybrid method of SA algorithm and gradient descent algorithm. SA algorithm is previously applied to estimate an initial value leading to the global minimum, and the gradient descent algorithm is also applied for high-speed convergence. In Fig. 6, if we utilize the minimum x_1 as an initial value of gradient descent algorithm, the global minimum of original function can be quickly and correctly looked for rather than that find by repeatedly applying the SA according to a sequences $\{\beta\}$.
Fig. 7 is the flow chart of proposed method. If the other minima exist between x_f (minimum of original function by using the gradient descent algorithm) and global minimum x^*, x^* can be find out by repeatedly applying the proposed method.

5. Neural network optimization by the proposed method

The basic idea in this paper is that, in applying the SA, if we initially choose a large β, we can get an uniextremal smoothed function $f^{-}(x, \beta)$, the minimum value x_1 of which can approximately point out the hill side value of the global minimum well. Comparing optimization of functions with NNs, minimization of the function $f(x)$ to variable x are much the same as the minimization of energy function $E(\boldsymbol{V})$ to neuron outputs $\boldsymbol{V}$.

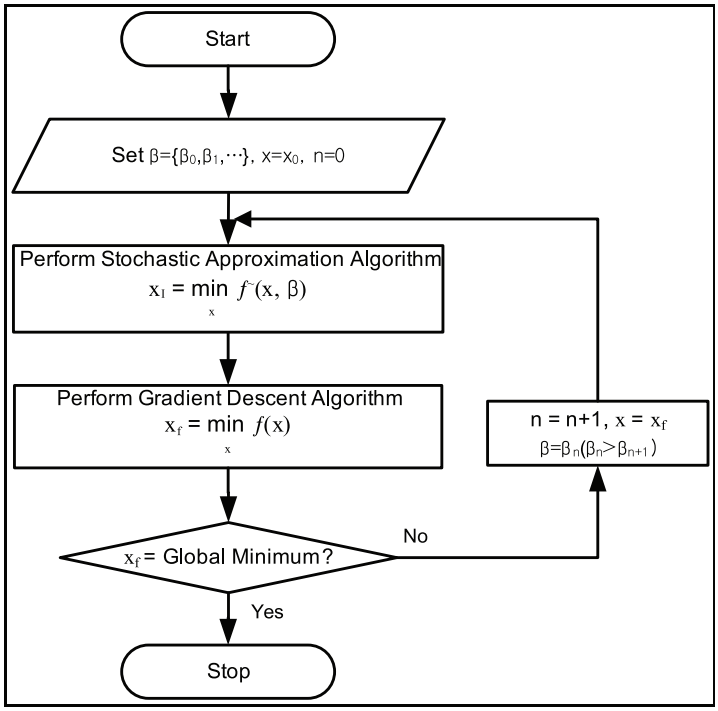


Fig. 7. Flowchart of the proposed method

Accordingly, we apply the proposed method to optimize the neural network. The update rule of Hopfield model is used in optimization as a gradient descent algorithm and operated in batch mode. SA algorithm is previously applied to estimate an initial neuron outputs leading to the global minimum, and then the update rule of Hopfield model is also applied for high-speed convergence. The neural network will be quickly optimized and clearly guaranteed that converges to a global minimum in state space if we go about it like this.
Therefore, the proposed hybrid algorithm using the SA and the update rule of Hopfield model can be detailed as follows:

Step 1. Define the energy function $E(\boldsymbol{V})$ for the given problems.

$$E(\mathbf{V}) = -(1/2)\Sigma_i \Sigma_j T_{ij} V_i V_j - \Sigma_i I_i V_I \tag{14}$$

Where T_{ij} denotes the weight value connecting the output of the j neuron with the input of the i neuron, I_i is the bias of i neuron, V_i and V_j are the outputs of i and j neurons, respectively.

Step 2. Calculate the smoothed gradient $\nabla_V \tilde{E}(\boldsymbol{V}, \beta)$ over the $E(\boldsymbol{V})$.

$$\begin{aligned}\nabla_V \tilde{E}(V,\beta) &= (1/2)(\beta/\eta)\left[E(V+\beta\eta) - E(V-\beta\eta)\right] = -(1/2)\eta^2 \textstyle\sum_i \sum_j T_{ij}(V_i + V_j) - \eta^2 \sum_i I_i \\ &= -(1/2)\eta^2 \left[(\textstyle\sum_i \sum_j T_{ij} V_i + \sum_i I_i) + (\sum_i \sum_j T_{ij} V_j) + \sum_i I_i) \right]\end{aligned} \tag{15}$$

Step 3. Set the randomized initial neuron outputs $\boldsymbol{V}_0$.

Step 4. Estimate the neuron outputs by performing SA with a large β according to the gradient $\nabla_V E^-(\boldsymbol{V}, \beta)$.

Step 5. Perform the conventional update rule of Hopfield model using the neuron outputs estimated in Step 4.

Step 6. If the energy function $E(\boldsymbol{V})$ value by the step 5 is less than a tolerance limit EPV, then stop. Otherwise go to step 4.

6. Experimental results and discussions

The proposed method has been applied to the *7*- and *10*-city TSPs. TSP is one of the combinatorial optimization problems. For an n-city tour, there are $n!/2n$ distinct circuits to consider (Freeman & Skaoura, 1991).

Pentium *IV-2.8G* CPU has been used in experiments. The initial values of neuron outputs are randomly chosen in [*-0.5* ~ *+0.5*] by using the random seeds, the output function in response to the net input of neuron is sigmoid function, and then the gain is chosen in *0.5*. The stopping rule is used in each experiment so as to terminate the calculation if all the outputs do not change any more or the energy function $E(\boldsymbol{V})$ becomes less than the tolerance limit $EPV=0.0001$. The initial dispersion control parameter $\beta_0=3.0$ and the smoothing function $h(\eta)$ with uniform *pdf* are chosen, respectively.

The experimental results for each example are shown in Table 1 and 4, where N_{HM}, and N_{SA} are the number of iterations of Hopfield model and SA algorithm, respectively. E_t is the final energy value in termination. t_{HM} and t_{PM} are the CPU time in [sec] of Hopfield model and proposed algorithm, respectively. In Table 2 and 4, $\tilde{x}$ and σ are mean and standard deviation.

Fig. 8 shows the *7*- and *10*-city coordinates that are randomly generated, respectively. A (x, y) coordinates are the *2*-dimensional city positions.

Table 1 shows the experimental results of *7*-city TSP to *10* random seeds. In case of Hopfield model, the constraints are satisfied at random seeds *5, 20*, and *30*, but the stopping rule is only satisfied in random seeds *5* and 20. The proposed method satisfies the stopping rule to

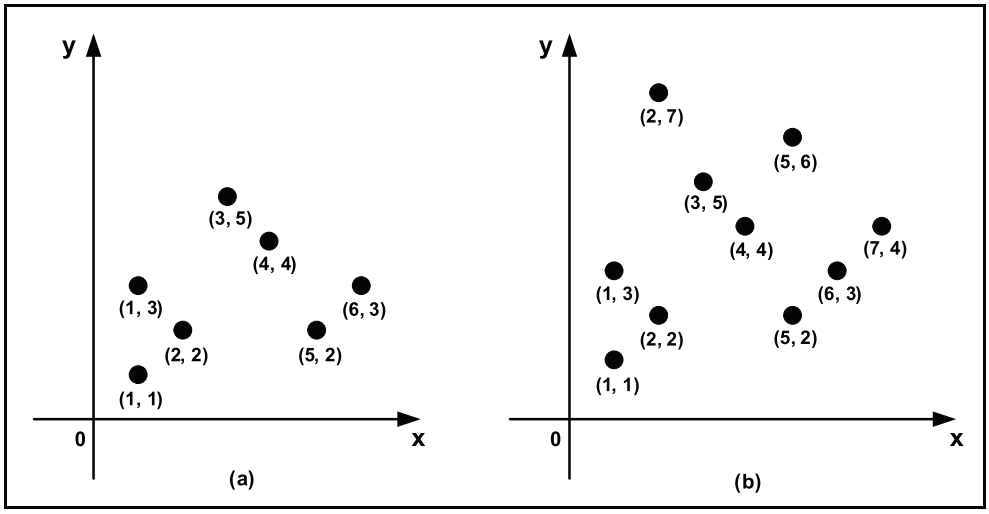


Fig. 8. *7*-city (a) and *10*-city (b) coordinates

Random seeds	Hopfield model			Proposed method		
	N_{HM}	E_t	t_{HM}	N_{SA}, N_{HM}	E_t	t_{PM}
0	241	0.0025 #	0.08	1, 287	0.0001	0.05
3	266	0.0021 #	0.07	1, 287	0.0001	0.05
5	298	0.0001	0.04	1, 287	0.0001	0.05
8	153	0.0023 #	0.04	1, 287	0.0001	0.05
10	325	0.0022 #	0.11	1, 287	0.0001	0.05
15	212	0.0018 #	0.06	1, 287	0.0001	0.05
20	337	0.0001	0.06	1, 287	0.0001	0.05
30	315	0.0014 #	0.05	1, 287	0.0001	0.05
50	182	0.0024 #	0.03	1, 287	0.0001	0.05
100	248	0.0019 #	0.08	1, 287	0.0001	0.05

Table 1. Experimental results of *7*-city TSP(# : non-convergence)

all *10* trials. The Hopfield model shows the faster convergence than the proposed method in case of the random seed *5*. This result shows that the deterministic rule of steepest descent may converge fast if the initial point is happen to be set near the global minimum. But there are few systematic methods that guarantee this initial point setting. The convergence rate by proposed method is *5* times and its convergence speed (time) is some higher than that of Hopfield model in case of successful convergence. We can also know that one cycle of SA takes more time than that of Hopfield model. Compared with the update rule of Hopfield model, SA is by reason of stochastic algorithm. But the SA algorithm is executed by a number of iteration in the proposed method.

Table 2 represents the experimental results of *7*-city TSP to *100* trials. Especially, Table 2 shows the experimental results that satisfy the stopping rule. *N, t,* and P_r are the number of iterations, the CPU time, and convergence ratio. As seen, the convergence rate by the proposed method is about *2.3* times and its convergence speed (time) is about *1.2* times higher than that of Hopfield model, respectively. The experimental results show that the convergence performances of proposed method are superior to that of Hopfield model with randomized initial neuron outputs setting. The standard deviation of proposed method is lower than that of Hopfield model. It means that the proposed method is less affected by the initial outputs setting than Hopfield model.

Table 3 shows the experimental results of *10*-city TSP to *10* random seeds. In case of Hopfield model, the constraints are satisfied at random seeds *3, 5*, and *30*, but the stopping rule is only satisfied in random seed *3*. The proposed method satisfies the stopping rule to all *10* trials as seen Table 1. The convergence rate by proposed algorithm is *10* times, and its convergence speed (time) is about *1.9* times higher than that of Hopfield model in case of successful convergence. We can also know that one cycle of SA algorithm takes more time than that of Hopfield model in this Table 3.

Table 4 also represents the experimental results of 10-city TSP to 100 trials. Table 4 also shows the experimental results that satisfy the stopping rule. As seen, the convergence rate Table 1 shows the experimental results of *7*-city TSP to *10* random seeds. In case of Hopfield model, the constraints are satisfied at random seeds *5, 20*, and *30*, but the stopping rule is only satisfied in random seeds *5* and 20. The proposed method satisfies the stopping rule to

N, t	Hopfield model		Proposed method	
	$\tilde{x}$	σ	$\tilde{x}$	σ
N	338.4	112.5	288	0
t	0.06	0.02	0.05	0
P_r(%)	43		100	

Table 2. Experimental results of *7*-city TSP to *100* trials

Random seeds	Hopfield model			Proposed method		
	N_{HM}	E_t	t_{HM}	N_{SA}, N_{HM}	E_t	t_{PM}
0	889	0.0043 #	1.16	2, 640	0.0001	0.44
3	793	0.0001	0.82	2, 640	0.0001	0.44
5	1470	0.0023 #	0.95	2, 640	0.0001	0.44
8	488	0.0174 #	0.61	2, 640	0.0001	0.44
10	1108	0.0128 #	1.50	2, 640	0.0001	0.44
15	697	0.0205 #	0.78	2, 640	0.0001	0.44
20	738	0.0223 #	0.52	2, 640	0.0001	0.44
30	1377	0.0016 #	0.86	2, 640	0.0001	0.44
50	1136	0.0102 #	0.73	2, 640	0.0001	0.44
100	508	0.0097 #	0.65	2, 640	0.0001	0.44

Table 3. Experimental results of 10-city TSP(# : non-convergence)

N, t	Hopfield model		Proposed method	
	$\tilde{x}$	σ	$\tilde{x}$	σ
N	1052.42	364.5	642	0
t	0.73	0.25	0.43	0
P_r(%)	19		100	

Table 4. Experimental results of 10-city TSP to 100 trials

Table 1 shows the experimental results of *7*-city TSP to *10* random seeds. In case of Hopfield model, the constraints are satisfied at random seeds *5, 20*, and *30*, but the stopping rule is only satisfied in random seeds *5* and 20. The proposed method satisfies the stopping rule to Compared Table 2(*7*-city) with Table 4(*10*-city), the Hopfield model is more difficult to set initial outputs for proving a good convergence as the problem size becomes larger. But the proposed method is less affected by the initial outputs setting and so gives relatively better results than the Hopfield model. Consequently, the convergence rate and speed by proposed method is higher than that of Hopfield model with randomized initial weights setting.

7. Conclusions

This paper proposes a global optimization of neural network by applying a hybrid method, which combines a stochastic approximation with a gradient descent. The approximation

point inclined toward a global escaping from a local minimum is estimated first by applying the stochastic approximation, and then the update rule of Hopfield model is applied for high-speed convergence.

The proposed method is applied to the *7*- and *10*-city TSPs, respectively. The experimental results show that the proposed method has superior convergence performances to the conventional method that performs the update rule of Hopfield model with randomized initial neuron outputs setting. Especially, the proposed method is less affected by the initial outputs setting and so gives relatively better results than the Hopfield model as the prom size becomes larger.

Our future research is to solve on a large scale combinatorial optimization problems by using neural networks of the proposed method.

8. References

Aarts, E. H. L. & Laarhoven, P. J. M. (1985). Statistical Cooling: A General Approach to Combinatorial Optimization Problems. *Philips Journal of Research 40*, pp.193–226

Abe, S.; Kawami, J. & Hirasawa, K. (1992). Solving Inequality Constrained Combinatorial Optimization Problems by the Hopfield Neural Networks. *Neural Networks,* Vol. 5, pp. 663–670

Abe, S. (1993). Global Convergence and Suppression of Spurious States of the Hopfield Neural Networks. *IEEE Trans. on Circuits Syst*., Vol. 40, No. 4, pp. 246-257

Abe, S. (1996). Convergence Acceleration of the Hopfield Neural Network by Optimizing Integration Step Sizes. *IEEE Trans. on System., Man, and Cybernetics-Part B*., Vol. 26, No. 1, pp. 194-201

Aiyer, S. V.; Niranjan, M. & Fallside, F. (1990). A Theoretical Investigation into the Performance of the Hopfield Model. *IEEE Trans. on Neural Networks*, Vol.1, No.2, pp.204-215

Andresol, R.; Gendreau, M. & Potvin, J. Y. (1997). *A Hopfield-Tank Neural Network Model for the Generalized Traveling Salesman Problem, in Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss et al. (eds.), Kluwer Academic Publishers, Boston

Baba, N. (1989). A New Approach for Finding the Global Minimum of Error Function of Neural Networks. *IEEE Trans. on Neural Networks*, Vol.2, pp.367-373

Beale, R. & Jackson, T. (1990). *Neural Computing: An Introduction*, IOP Publishing Ltd., Bristol, U.K.

Biro, J.; Koronkai, Z. & Tron, T. (1996). A Noise Annealing Neural Network for Global Optimization. *ICNN'1996*, Vol.1, pp. 513-518

Bout, D. E. & Miller, T. K. (1988). A Traveling Salesman Objective Function That Works. *ICNN-88*, pp. 299-303

Burke, L. I. (1994). Adaptive Neural Networks for the Traveling Salesman Problem: Insights from Operations Research. *Neural Networks*, Vol. 7, pp. 681–690

Cichock, A. & Unbehaun, R. (1993). *Neural Networks for Optimization and Signal Processing*, John-Wiley & Sons, New York

Dorigo, M.; Maniezzo, V. & Colorni, A. (1991). The Ant System: An Autocatalytic Optimizing Process. *Technical Report no. 91-016 Revised,* Politecnico di Milano, Italy

Dorigo, M. & Gambardella, L. (1997). Ant Colonies for the Traveling Salesman Problem. *Bio Systems*, Vol. 43, pp. 73-81

Freeman, J. A. & Skapura, D. M. (1991). *Neural Networks : Algorithms, Applications, and Programming Techniques,* Addison-Wesley Pub. Co., New York

Gall, A. L. & Zissimopoulos, V. (1999). Extended Hopfield Models for Combinatorial Optimization. *IEEE Trans. on Neural Networks*, Vol.10, No.1, pp.72-80

Gee, A. H. & Prager, R. W. (1995). Limitations of Neural Networks for Solving Traveling Salesman Problems. *IEEE Trans. on Neural Networks*, Vol. 6, pp. 280–282

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning,* Addison Wesley

Hegde, S. U. ; Sweet, J. L. & Levy, W. B. (1988). Determination of Parameters in a Hopfield/Tank Computational Network. *ICNN-88*, pp.291-298

Hopfield, J. J. & Tank, D. W. (1985). Neural Computation of Decisions in Optimization Problems. *Biological Cybernetics*, Vol. 52, pp.141-152

Huang, X. (2005). A New Kind of Hopfield Networks for Finding Global Optimum. *IJCNN'05*, Foster City, USA, Vol.2, pp. 764-769

Kawabe, T.; Ueta, T. & Nishio, Y. (2002). Iterative Simulated Annealing for Hopfield Neural Network. *International Journal of Modelling and Simulation*, *Marina del Rey, USA*, pp. *353-409*

Lai, W.K. & Coghill, G. G. (1994). Initializing the Continuous Hopfield Net. *Proc. ISCAS,* pp. 4640-4644

Lin, S. (1965). Computer Solutions of the Traveling Salesman Problem. *Bell Syst. Journal*, Vol. 44, pp.2245-2269

Lin, S. & Kernighan, B. W. (1971). An Effective Heuristic Algorithm for the Traveling Salesman Problem. *Oper. Res.* 21, pp. 498-516

Mascato, P. (1989). On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts : Towards Memetic Algorithms. *Caltech Concurrent Computation program*, pp. 158-179

Mühlenbein, H. (1992). Parallel Genetic Algorithms in Combinatorial Optimization. *Computer Science and Operations Research*, pp. 441-456

Sharbaro, D. (1994). A Stability Criterion for Hopfield Networks Based on Popov Theorem," *Proc. IJCNN*, pp. 4567-4570

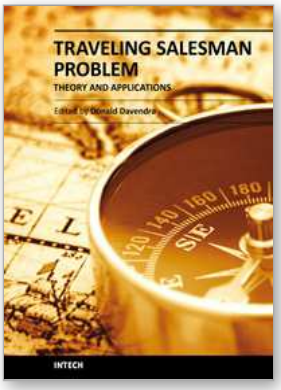
Styblinski, M. A. & Tang, T.S. (1990). Experiments in Nonconvex Optimization: Stochastic Approximation with Function Smoothing and Simulated Annealing. *IEEE Trans. on Neural Networks*, Vol.3, pp.467-483

Szu, H. & Hartley, R. (1987). Fast Simulated Annealing. *Physics Letters*, Vol. 122, pp.157-162

Wilson, G. V. & Pawley, G. S. (1988). On the Stability of the TSP Problem Algorithm of Hopfield and Tank," *Biological Cybernetics*, Vol. 58, pp. 63-70

Xavier, S. S.; Suykens, J. A. K.; Vandewalle, J. & Bolle, D. (2006). Coupled Simulated Annealing for Continuous Global Optimization. *Internal Report 06-07*, ESAT-SISTA, K.U.Leuven (Leuven, Belgium)

Zurada, J. M. (1992). *Introduction to Artificial Neural Systems*, West Pub. Co., New York



Traveling Salesman Problem, Theory and Applications
Edited by Prof. Donald Davendra

ISBN 978-953-307-426-9
Hard cover, 298 pages
Publisher InTech
Published online 30, November, 2010
Published in print edition November, 2010

This book is a collection of current research in the application of evolutionary algorithms and other optimal algorithms to solving the TSP problem. It brings together researchers with applications in Artificial Immune Systems, Genetic Algorithms, Neural Networks and Differential Evolution Algorithm. Hybrid systems, like Fuzzy Maps, Chaotic Maps and Parallelized TSP are also presented. Most importantly, this book presents both theoretical as well as practical applications of TSP, which will be a vital tool for researchers and graduate entry students in the field of applied Mathematics, Computing Science and Engineering.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821