9

A Multi-World Intelligent Genetic Algorithm to Optimize Delivery Problem with Interactive-Time

Yoshitaka Sakurai and Setsuo Tsuruta
Tokyo Denki University,
Japan

1. Introduction

Due to the complicated road network, the efficiency of product distribution remains on a lower level in Japan compared to that of the USA, which disadvantages the productivity of Japanese industries. This inefficiency also causes social problems and economical losses. Namely, we are facing the necessity of urgently reducing the volume of car exhaust gases to meet environmental requirement as well as curtailing transport expenses in Japan.

There are many distribution systems that should be optimized, including the delivery of parcels, letters and products supply/distribution across multiple enterprises. In order to improve the efficiency of these distributions, it is necessary to optimize the delivery routes, or the delivery order of multiple delivery locations (addresses). One round delivery comprises more than several tens or hundreds of different locations. Thus, the optimization of a delivery route can be modelled as such a large-scale of Traveling Salesman Problem (TSP). However, TSP is a combinatorial problem that causes computational explosion due to n! order of combinations for n-city TSP. Therefore, to practically obtain the efficient delivery route of such a distribution system, a near optimal solving method of TSP is indispensable. Yet, the practical use of such a solving method on an actual site needs human confirmation (which is difficult to formulate) of the solution, since social and human conditions are involved. Namely, human users should check to understand that the solution is practical. Users sometimes should correct manually or select the alternative solution.

Therefore, the TSP solving methods are required to ensure the response time necessary for the above human interaction.

By the way, solutions generated by domain experts may have 2~3% of deviation from the mathematical optimal solution, but they never generate worse solutions which may cause practical problems. On the other hand, conventional approximate TSP solving methods (Lawer et al., 1985; Kolen & Pesch, 1994; Yamamoto & Kubo, 1997) may generate even mathematically optimal solutions in some cases but cannot ensure the amount of errors below 2~3%. Such errors possibly discourage user, which makes those conventional methods not practically useful, especially for the above-mentioned applications.

Strict TSP solving methods, such as the branch and cut method (Grotschel & Holland, 1991) and Dynamic Programming (DP) (Bertsekas, 1987) or approximate solving methods using Simulated Annealing (SA) (Kirkpatrick et al., 1983; Ingber, 1993; Miki et al., 2003) and Tabu

Search (TS) (Glover, 1989; 1990; Hooker & Natraj, 1995; Fang et al., 2003), take much time for calculation. Therefore, they cannot guarantee the above-mentioned real-time conditions. The Lin-Kernighan (LK) method and its improved version (Lin & Kernighan, 1972) are also proposed as solving methods of the TSP. However, they cannot constantly guarantee expert-level accuracy (Kubota et al., 1999).

Thus, we developed a method which efficiently solves the TSP, using Genetic Algorithm (GA) (Onoyama et al., 2000). This method enables to guarantee the responsiveness by limiting the number of generations of GA and by improving genetic operations (initial generations, mutation, and crossover). However, in some distribution patterns, this solving method fell into a local minimum and could not achieve expert-level accuracy. Therefore, we needed further improvement of our solving method to guarantee expert-level accuracy for all cases.

The chapter is organized as follows: In the next (second) section, the delivery route optimization problem and its technical problems are described. In the third section, the method for solving the problem is proposed. Then, in the fourth section, experiments to validate its effect and its results are shown. In the fifth section, the effectiveness of the solving method will be proved based on the experiments, and in the sixth section, we will compare it with other methods. And in the last seventh section, the results will be concluded.

2. Problems in delivery route optimization

In this section, firstly, two kinds of actual distribution systems are depicted. And, in 2.2, the optimization problems of these distribution systems are formally and technically described.

2.1 Delivery route optimization problem

A distribution network across multiple manufacturing enterprises is outlined in Fig. 1. Parts for production are delivered from parts makers (suppliers) to factories directly or through depots. Parts are not delivered to a factory or a depot independently by each parts maker, but a truck goes around several parts makers and collects parts. This improves the distribution efficiency, which contributes to the curtailment of distribution expenses and to the reduction of the volume of car exhaust gases.

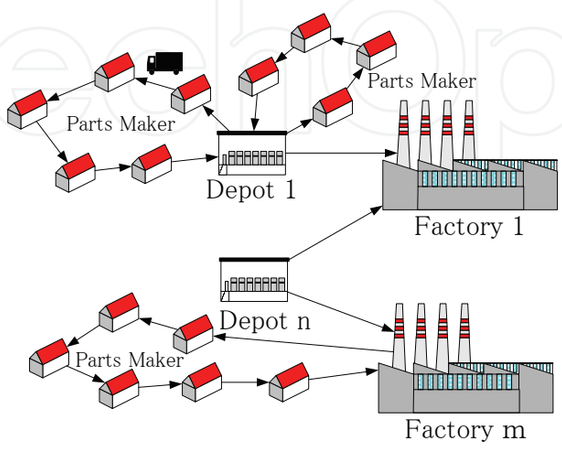


Fig. 1. Large-scale distribution network

In order to optimize the above-mentioned large-scale distribution network, we need to grasp the total cost of distribution under various conditions by repeating the simulation process as shown in Fig. 2. First, the conditions have to be set up manually, concerning locations of more than ten factories (parts integration points for production), locations of dozens of depots (intermediate depositories of parts), and allocation of trucks to transport parts. To calculate distribution cost in each simulation, it is necessary to create delivery routes. However, there are several hundreds of parts makers, dozens of depots and more than ten factories. Therefore, there are about 1000 distributing routes on each of which a truck goes around dozens (max. 40) of parts makers starting from one of the depots or factories. Thus, in each simulation, a delivery route creation is repeated about 1000 times for a set of conditions manually set up, the total delivery cost is calculated, and a person in charge globally decides the network optimality as shown in Fig. 2. To globally evaluate these results, human judgment is indispensable and interactive response time (less than tens of seconds) is required. Thus, the system needs to create about 1000 or several hundreds of distribution routes within at least tens of seconds. Therefore, one route has to be created within tens of milliseconds.

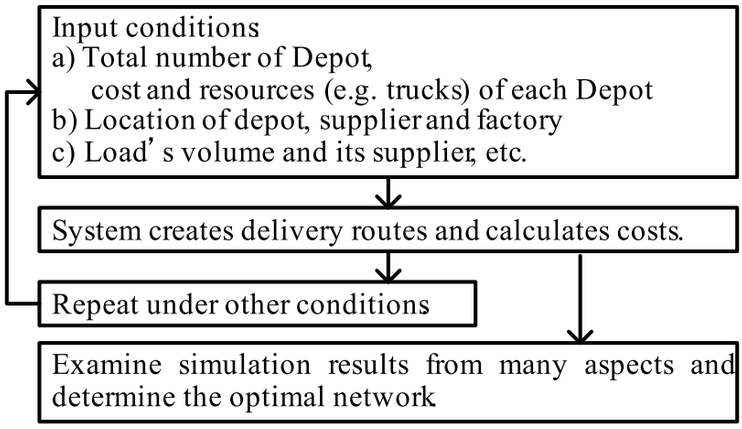


Fig. 2. Simulation process

Meanwhile, as to the delivery route optimization problem for parcels and letters, a round delivery is carried out 1-3 times a day with a small vehicle such as a motorcycle or a small truck.. Delivery zone that is covered by one vehicle is different according to the region. Delivery locations are comparatively overcrowded in the urban area, whereas scattered in the rural area. Therefore, the number of locations (addresses) for delivery differs - over several tens or hundreds - depending on the region and time zone. It is necessary to make and optimize a new delivery route for each round delivery since delivery locations change every day and every time. Though human or social factors should be considered, this is a problem to search the shortest path or route, modelled as a famous "Chinese Postman Problem" or "Traveling Salesman Problem (TSP)". The computer support by near optimal solving method is quite useful to reduce the burden and loss time of workers as well as car exhaust gases in such distribution networks or parcels /letters delivery.

2.2 Technical problems

The delivery route optimization problem of these distribution systems is formulated as follows:

The delivery network is represented by weighted complete graph $G=(V,E,w)$. V is node set. A node v_i (i=1,…,N) represents a location (address) for delivery. N is the number of nodes. E is edge set. A edge e_{ij} represents a route from v_i to v_j. w is edge weight set. A edge weight d_{ij} represents a distance from node v_i to node v_j, $d_{ij} = d_{ji}$. The problem to find the minimal-length Hamilton path in such a graph $G=(V,E,w)$ is called Traveling Salesman Problem (TSP).

Thus, to improve the delivery efficiency of such distribution systems, it is required to obtain an approximate solution of a TSP within an interactive length of time (max. tens of milliseconds). Yet, expert-level accuracy (less than 3% of the deviation from the optimal solution) is always necessary, since domain experts may have such errors in their solutions but never generate worse solutions which may cause practical problems.

We developed an efficient method for solving the TSP by elaborating a random restart method. The developed method enables to guarantee the responsiveness by limiting the number of repetitions and by devising component methods and heuristics (Kubota et al., 1999). However, to meet the required guarantee of expert-level accuracy (below 3% of errors), it took more than 100 milliseconds to solve one TSP, which caused the time to solve one TSP should be significantly decreased.

Therefore, in order to improve the real time behavior, we proposed a GA that uses heuristics for the crossover and the mutation, and yet whose generation number is limited (Onoyama et al., 2000).

However, for some kinds of delivery location patterns, obtained solutions had more than 3% of errors. To overcome these weaknesses of the solving method, other heuristics were applied. Nevertheless, these heuristics were not effective again for some patterns, and the above-mentioned accuracy was still not guaranteed for all kinds of patterns (as is described in detail in section 5.1).

In the next section, an intelligent approximate method to solve above-mentioned problems is proposed.

3. A multi-world intelligent genetic algorithm

As stated in the foregoing sections, the delivery routing problem in the above distribution systems can be formalized as a TSP. Especially a symmetrical (non-directed) Euclidean TSP (Lawer et al., 1985; Yamamoto & Kubo, 1997) is assumed in this chapter.

3.1 Concept of the proposed method

In order to solve problems mentioned above (in section 2), the following multi-world intelligent GA (MIGA) method is proposed. This guarantees both real-time responsiveness and accuracy for various kinds of delivery location patterns. At the initial phase of GA, groups of individuals (population) that become the candidates of the solution are generated. And, based on the population, new individuals (solution candidates) are generated by the crossover and the mutation operator, and individuals are improved by the evaluation and the selection. With our GA, each individual (chromosome) represents the tour, namely the delivery route in TSP. Each gene of the chromosome represents the node number (identification number of the address for delivery). A chromosome is a sequence of nodes whose alignment represents a round order as shown in Fig. 3.

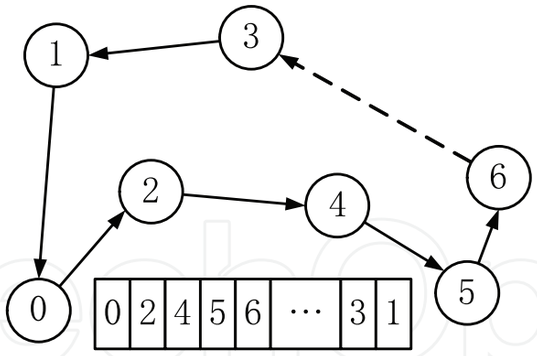


Fig. 3. Chromosome

3.1.1 Multi-world intelligent GA

It is difficult to find an effective search method that always guarantees expert-level optimality as well as the required real-time behavior for various distribution location patterns. Heuristics, that are effective to particular patterns, are not necessarily useful to other patterns. Yet, the application of excessively complicate algorithms or heuristics makes the responsiveness worse. Therefore, a high-speed GA that mainly uses simple general heuristics is combined with an intelligent GA, into which knowledge for handling particular problems is incorporated. In this way, we could avoid local minima for various delivery location patterns.

Concretely speaking, a 2opt-type mutation is used for the high-speed GA. This 2opt-type mutation quickly improves tours. Therefore, good solutions are usually expected to be obtained within a short length of time. However, it also takes risks of falling into a local minimum. Our experiments revealed that this high-speed GA (called 2opt-type GA) computes some inefficient tours for certain delivery location patterns.

Thus, a multi-world intelligent GA method is proposed. In this method, there are two kinds of GA worlds; (1) an intelligent GA world (called block-type GA) holding the knowledge to meet the particularities of problems as well as (2) the high-speed GA world (called 2opt-type GA). Both kinds of worlds are independently executed. Such execution is repeated. The same kind of worlds can be repeated. And they are collaborated through integrating the results.

In the intelligent GA world, the following rather problem-oriented knowledge about the neighborhood conditions or their relaxation is incorporated into operations of the block-type GA so that these operations can be controlled through utilizing the knowledge.

a. Multi-step NI method

This is particular heuristics that constructs the initial tour by using step-by-step the NI (Nearest Insertion) method to globally consider adjacent delivery locations, where the adjacency is defined by problem-oriented knowledge as mentioned later.

b. Block-type mutation

This mutation selects a node randomly out of a tour, and mutates it together with its neighbor nodes in order to avoid local minimum solutions.

3.1.2 Limiting the generation number of GA

In this method, the computation time necessary for processing the GA is calculated as follows.

Let

- n be the the population size,
- l be the length per individual,
- $T(X)$ be the computation time of X,
- $Prob(X)$ the probability of X, and
- $Ave(X)$ the average of X.

So the computation time can be estimated as
computation time =
*T(initialize) + number of generations * T(one generation of GA) + margin constant C*
with

- $T(initialize) = n * C_{ini} * f_{ini}(l)$
- *T(one generation of GA) =*
  T(crossover stage) + T(mutation stage) + T(evaluation) + T(selection)
- *T(crossover stage) = n * Prob(crossover) * Ave(T(crossover))*
- $Ave(T(crossover)) = C_{cros} * f_{cros}(l)$
- *T(mutation stage) = n * Prob(mutation) * Ave(T(mutation))*
- $Ave(T(mutation)) = C_{mut} * f_{mut}(l)$
- $T(evaluation) = C_{fit} * f_{fit}(l, n)$, $T(selection) = C_{sel} * f_{sel}(n)$

As the parameters of GA, n, l, *Prob(crossover)* and *Prob(mutation)* are given. $f_{ini}(l)$, $f_{cros}(l)$, $f_{mut}(l)$, $f_{fit}(l, n)$ and $f_{sel}(n)$ are respectively computational complexity of each operation (initialization, crossover, mutation, fitness evaluation, and selection) that basically does not depend on hardware details such as the CPU architecture. These are derived through analyzing the algorithm. For example, since "quick sort" is used in the selection operator, $f_{sel}(n)$ *is* calculated as follows:

$$f_{sel}(n) = n^*(\log n) + C_1^* n + C_0 \tag{1}$$

Here, C_1 and C_0 are the coefficients for the minor dimension to calculate the complexity of the quick sort algorithm. C_{ini}, C_{cros}, C_{mut}, C_{fit}, C_{sel}, are coefficients to obtain computation time from the complexity of each operation mentioned above. These coefficients are hardware dependant but can be identified using the result of experiments. More precisely, these can be calculated using the number of steps of each program, and identified/adjusted using the result of experiments to take detailed factors such as the CPU architecture into account. The computation time for one generation of GA changes stochastically.
However, the estimation error can be suppressed within allowable ranges, through

1. dividing the computation time into that of fundamental components and
2. calculating the computation time using each component's computational complexity and the experimental results to determine the coefficient.

Furthermore, to absorb this estimation error, and to guarantee the interactive real-time responsiveness, a *margin constant C* can be set by the user as a safety margin.
Using this computation time, the number of generations repeatable within the required response time can be calculated.

3.2 Components of the proposed method

In this method, a gene represents a (traveling) node, and an individual represents a tour.

3.2.1 Method for generating initial individuals

In order to obtain a highly optimized solution by avoiding the convergence into a local minimum, the randomness of the initial individuals is important. However, the speed of convergence slows down, if totally random initial solutions are generated as is done by a random method. Thus, the other methods are devised as shown below.

a. Random method

Construct a tour by putting nodes in a random order.

b. Random NI method

Put nodes in a random order and insert them into a tour by using the NI method according to the randomized order.

c. Multi-step NI method

In case experts generate a tour (a traveling route), they usually determine the order of delivery locations, globally considering the whole route, so that the nearest location from the present one can always be the next location to deliver. On the model of such global consideration of experts, a multi-step NI method is proposed which enables to generate a tour similar to the tour generated by experts.

In detail, this method constructs a tour through the following steps:

1. *Que(nodes)* is a queue of nodes with their *check count* of each node initialized as zero. $tour_{gen}$ is the tour generated. w is a real type variable that meets the requirement $1 <= w$. Its initial value is decided by problem-oriented knowledge. For example, w is decided based on the position of the entire node as follows:

$$w = (Dist_{ave} + d * \sigma) \ / \ Dist_{ave} \tag{2}$$

   $Dist_{ave}$ is an average of the distance between each node and a depot. σ is a standard deviation of the distance among nodes. The initial value of d is 1.0.
2. Enqueue all nodes to *Que(nodes)* at random order.
3. Dequeue a node ($node_{add}$) from *Que(nodes)*.
4. Temporally add a $node_{add}$ to $tour_{gen}$ by the NI method.
5. Evaluate L_{pre} and L_{after}. L_{pre} is the length of $tour_{gen}$ before its addition. L_{after} is the length of $tour_{gen}$ after its addition. … (*)
6. If $L_{after} < (L_{pre} * w)$, then $node_{add}$ is inserted (actually added) into $tour_{gen}$ and w is returned to an initial value. Else enqueue $node_{add}$ to *Que(nodes)*, with *check count* of $node_{add}$ incremented.
7. f the *check count* of the top node of *Que(nodes)* is not zero, then w is increased, and the *check count* of every node in *Que(nodes)* is initialized zero. Here, the quantity of this increase is also decided by problem-oriented knowledge, for example, $d_{new} = d_{old} + 0.5$ of the equation (2).
8. If *Que(nodes)* is empty, then it ends. Else it returns to step (3) and repeat.

* Distances between two points are calculated for all their combinations by the Dijkstra method beforehand.

3.2.2 Method for crossover (NI-combined crossover)

To inherit good features of parents by crossover and to realize the quick convergence in GA, a crossover operation using the NI method is proposed. This crossover operation called NI-combined crossover comprises the following steps (Fig. 4):

1. $tour_{par1} = \{x_1, x_2, \ldots, x_n\}$ and $tour_{par2} = \{y_1, y_2, \ldots, y_n\}$ are parent tours and $tour_{chi}$ is a child tour.
2. Determine a crossover point x_i from $tour_{par1}$.
3. Copy a sub-tour $\{x_1, x_2, \ldots, x_i\}$, representing a group of nodes located before the crossover point in $tour_{par1}$, to $tour_{chi}$.
4. Change the order of remaining nodes $\{x_{i+1}, \ldots, x_n\}$, according to the order of nodes in $tour_{par2}$.
5. Insert the remaining nodes into $tour_{chi}$, using the NI method in the order that is changed in (4). When all nodes of $tour_{par1}$ are inserted to $tour_{chi}$, the crossover processing ends.

In this way, the generated tour is represented as a new child. Through applying this NI-combined crossover, the order of nodes contained in parents is inherited to their children to increase the convergence speed.

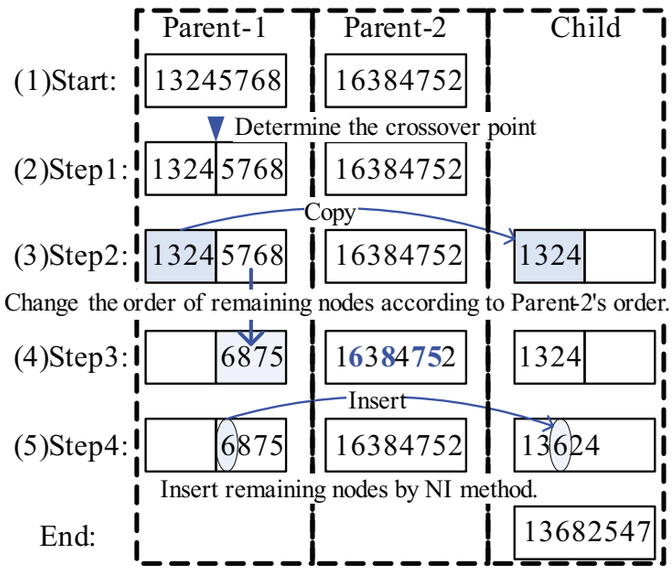


Fig. 4. NI-combined crossover

3.2.3 Method for mutation

Mutation of GAs often did not have much impact on the convergence of solutions without combining local search methods or without embedding problem-oriented knowledge. Thus, the following two mutation methods are proposed.

a. 2opt-type mutation

This method enables to improve the convergence speed by combining a 2opt-like simple local search heuristic method with GA's mutation operation. This consists of the following steps :

1. $tour_{par} = \{x_1, x_2, \ldots, x_n\}$ is a parent tour and $tour_{chi}$ is a child tour.
2. Copy the contents of $tour_{par}$ to $tour_{chi}$.
3. Select a node x_i randomly from $tour_{chi}$.
4. Select another node x_j randomly from $tour_{chi}$ except $\{x_i, x_{i+1}\}$.
5. Generate $tour_{gen}$ $\{x_1,\ldots, x_i,x_j, \ldots, x_{i+1},x_{j+1},\ldots, x_n\}$ by reversing sub-tour $\{x_{i+1}, \ldots, x_j\}$ of $tour_{chi}$ $\{x_1,\ldots, x_i,x_{i+1}, \ldots, x_j,x_{j+1},\ldots, x_n\}$

6. If $L_{chi} < L_{gen}$ (tour length is not improved), then it ends. Else copy the contents of $tour_{gen}$ to $tour_{chi}$. Until such link exchanges are all checked, return to step (4) and repeat. L_{gen} is the length of $tour_{gen}$. L_{chi} is the length of $tour_{chi}$.

b. Block-type mutation

2opt-type mutation easily improves tours, and good solutions are expected to be obtained within a short length of time. However, it also takes risks of failing into a local minimum. To obtain a solution closer to the optimum, it is desirable to escape from a local minimum by destroying a block of a tour at a time. For this purpose, the following block-type mutation is proposed. This consists of the following steps:

1. $tour_{par} = \{x_1, x_2, \ldots, x_n\}$ is a parent tour. $tour_{chi}$ is a child tour.
2. Select a node x_i randomly from $tour_{par}$.
3. Move the nodes, except $\{x_{i-r}, \ldots, x_{i+r}\}$ namely except x_i and its neighbor nodes of $tour_{par}$, to $tour_{chi}$. The size of neighborhood r is specified as problem-oriented knowledge, for instance, a random number from *0* to *B * (the distance to the node farthest from a depot)*. *B* is a constant number specified as problem-oriented knowledge.
4. Insert $\{x_{i-r}, \ldots, x_{i+r}\}$ into $tour_{chi}$ using the NI method. When all nodes have been inserted to $tour_{chi}$, the mutation processing ends.

3.2.4 Method for selection

In order to get highly optimized solutions and realize quick convergence in GAs, individuals are selected out of the population including both parents' and children's. And, 10% of individuals in a new generation are selected randomly from the above populations to give the chance of reproduction to even inferior individuals. Furthermore, to enhance the evolution efficiency, only one individual is selected when the same individuals are generated.

3.3 Proposed solving method

Through integrating above components, the following three kinds of GA methods are proposed to ensure both real-time responsiveness and accuracy for various kinds of delivery location patterns.

3.3.1 2opt-type GA (high-speed GA)

This method is shown in Fig. 5. This method makes it possible to guarantee quick convergence of solutions through improving initial solutions due to the random NI method and through applying the NI-combined crossover and the 2opt-type mutation.

The computational complexity of the NI-combined crossover is $O(n^2)$. On the other hand, the computational complexity of the 2opt-type mutation is much smaller. Indeed, the computational complexity of the 2opt-type mutation is $O(n^2)$ in worst cases, but it hardly occurs for highly optimized individuals generated in the initial population phase by the random NI method and improved in later phases by the NI-combined crossover.

In more detail, the probability of improvement by link exchanges of the 2opt method is small, since the NI method inserts each node so that the difference, between

1. the sum of the length of both links with both sides' neighbors and
2. the length of the link among both neighbors before its insertion,

can be minimized. Thus, the computation time of the 2opt-type mutation is expected to be much smaller than that of operations using the NI method such as the NI-combined crossover.

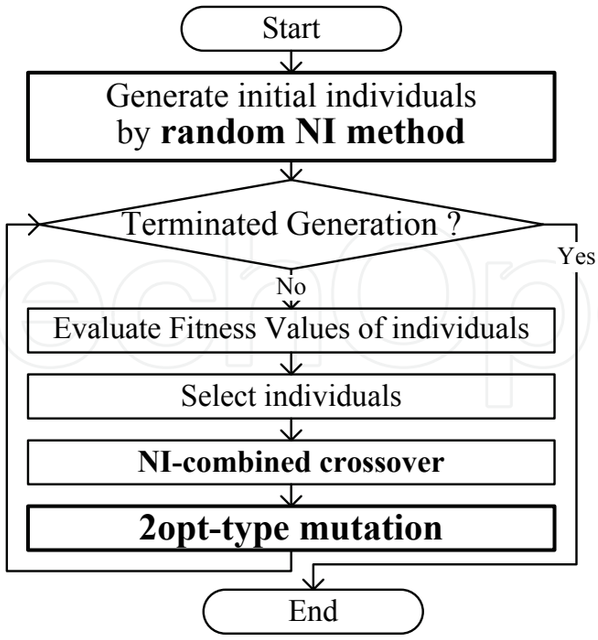


Fig. 5. 2opt-type GA

3.3.2 Block-type GA (intelligent GA)

This method is shown in Fig. 6. This method is expected to obtain highly optimized solutions through avoiding local minima. This can be attained through (1) constructing highly optimized initial solutions by means of the multi-step NI method, and (2) reconstructing a large part of a locally optimized tour by means of block-type mutation to avoid falling into a local minimum.

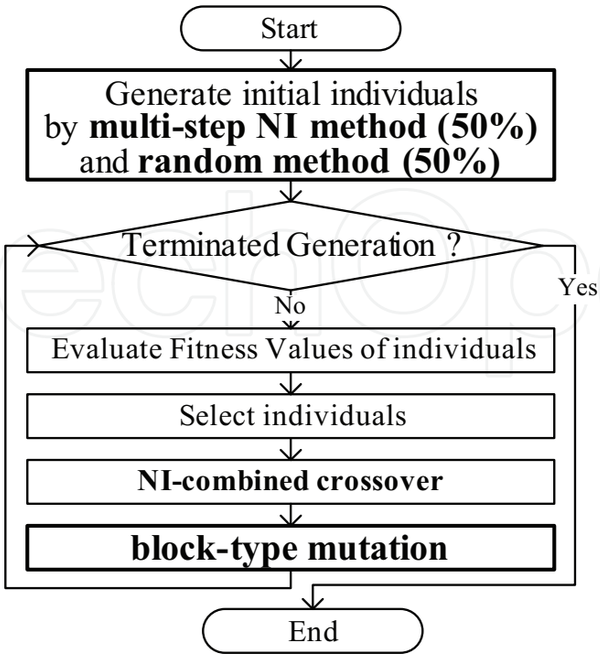


Fig. 6. Block-type GA

3.3.3 Multi-world intelligent GA

The finally proposed method is called "Multi-world intelligent GA (MIGA)". This comprises the 2opt-type GA method (world) and the block type GA method (world) that follows the former. MIGA selects the better one out of the solutions obtained in these two GA worlds. This raises the probability to have highly accurate solutions for various types of delivery location patterns within an interactive real-time context, because of the following reasons:

1. As is explained in 3.3.1, though the computation time of the NI-combined crossover is $O(n^2)$, the computation time of the 2opt-type mutation in the 2opt-type GA method (world) is much smaller than the former. Furthermore, the NI method checks just links among neighbors but all links among neighbors in the tour to be inserted. Meanwhile, though not all links, the 2opt operation in the 2opt-type mutation checks links between nodes that are not neighbors. Thus the 2opt-type mutation in the 2opt-type GA world but not being in the block type GA world can have the possibility to search other optimal solutions than the NI method, namely the block type GA method where only NI method is used effectively as heuristics.
2. On the other hands, the block type GA world can have the possibility to search other optimal solutions than 2opt-type GA, owing to the Multi step NI method and the block-type mutation, both of which exploits the power of the NI method enforced by problem oriented knowledge mentioned previously in (c) of 3.2.1 and in (b) of 3.2.3.

Yet, to guarantee real-time responsiveness, both of these two GAs finish their processing within the limited length of time through offline calculation of the number of generations repeatable within the time limit (e.g. 15 milliseconds for each GA).

4. Experiment and results

4.1 Experiment

In this section, the experiment to evaluate the proposed method is explained. The program codes are written by C language. A computer equipped with Intel Pentium II (450MHz) processor and 256MB memory is used for this experiment.

As explained by the footnote in the introduction, 40 cities TSPs were used for this experiment. Yet, various combinations of 40 delivery locations are possible. Thus, randomly selected 20000 different patterns of 40 delivery locations were prepared. Then, to evaluate three kinds of GA methods described in 3.3, each solving method solved 20000 test patterns, 100 times per pattern, and the probability to obtain solutions within 3% of errors was calculated.

In this experiment, the population size is 100 and each crossover rate and mutation rate are 10% respectively. These parameters were determined by the way of comparative experiments with many sets of parameters.

4.2 Results

To guarantee the real-time responsiveness, the time necessary for processing one generation is calculated, and based on this value, the number of generations of the GA is determined. Table 1 shows an example of the number of generations to respond within 30 milliseconds when the population size is 100. Then, the tests were repeated 100 times per pattern for three kinds of GAs.

Each test used 20000 kinds of delivery location patterns. The probability to obtain solutions within 3% of errors compared to the optimal solutions was checked. Furthermore, the

probability to obtain the optimal solutions within 30 milliseconds was also checked. These results are shown in Table 2.

#	Method	Generating Initial Individuals	Mutation	Number of Generations
1	2opt-type	Random NI	2opt-type	24
2	Block-type	Random + Multi step NI	Block-type	20

Table 1. The number of generations of each method repeatable within 30 milliseconds

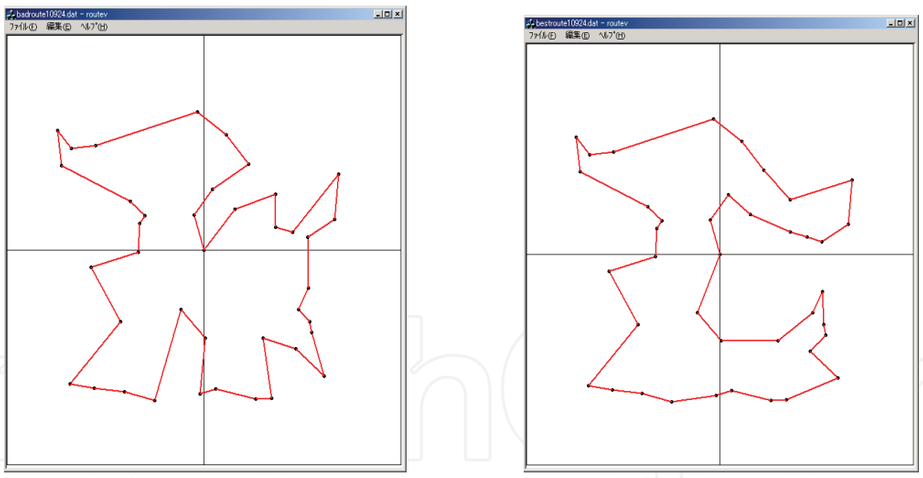
#	Method	Optimal	Below 3%
1	2opt-type GA	84.45%	99.885%
2	Block-type GA	83.75%	99.785%
3	MIGA	92.05%	100.0%

Table 2. The solution optimality

5. Evaluation

5.1 Effect of the proposed TSP solving method

Table 2 shows the usefulness of the proposed MIGA quite convincingly. Only MIGA method could solve a 40 cities TSP with less than 3% of errors with 100% of probability within 30 milliseconds.



(a) 2opt-type GA (Tour length:259)

(b) Block-type GA (Tour length:249)

Fig. 7. Difference of tour shape in a special location pattern

5.1.1 Effect of block-type GA (intelligent GA)

Tour shapes were examined as to solutions generated by the 2opt-type GA and leaving more than 3% errors. As a result, most of these shapes were like gear wheels as shown in Fig. 7 (a). Experts usually generate more straight routes as shown in Fig. 7 (b). If experts find inefficient routes such as shown in Fig. 7 (a), they reject to use the system since they consider it as an unreliable one.

In case of using a block-type GA (intelligent GA), tours similar to Fig. 7(b) were generated even for such delivery location patterns. The reason is why the intelligent GA has the block-type mutation in order to avoid falling into a local minimum.

5.1.2 Effect of multi-world intelligent GA

According to our experiment, in case of the 2opt-type GA, 43 cases out of 20000 tests had more than 3% errors. In case of the block-type GA method, 23 cases had more than 3% errors.

However, the multi-world intelligent GA (MIGA), namely, the 2opt-type GA subsequently followed by the block-type GA could generate solutions below 3% of error within 30 milliseconds, for every case in 20000 tests. The reason is that, coping with various delivery location patterns, either the 2opt-type GA or the intelligent GA can avoid falling into a local minimum (over 3% errors). Thus, MIGA method could guarantee the responsiveness as well as the expert-level accuracy, namely, below 3% errors.

5.2 Applicability of the proposed solving method

To evaluate the applicability of our proposed solving method, we applied it to a simulation of a parts supply logistics network which consists of one (assembly) factory, 7 depots, and 30 part makers (suppliers). This simulation needs to optimize the distribution area allocation of each truck, as well as to optimize each truck route. This simultaneous optimization of each truck area allocation and each truck route is classified as the Vehicle Routing Problem (VRP) (Laporte, 1992).

To apply our proposed methods to the VRP, we modified the chromosome structure and the NI method which is used in GA operations (initial construction, mutation, and crossover). Namely, the chromosome structure is extended to represent multiple trucks' routes instead of one truck's route, and the extended NI method puts a new node into the best position out of all truck routes instead of only one truck route.

We simulated the above-mentioned logistics network in 3 cases, which have the same simulation conditions except the amount of loads, as shown in table 3. The resultant number of trucks in table 3 is verified as optimal. Moreover, the total tour length deviation from the optimal solution is less than 3 %.

Consequently, our proposed GA methods such as the multi-step NI method and the block-type mutation are applicable not only to the TSP but also to more general problems such as the VRP. That is to say, these methods are effective in solving the problems defined over metric space such as the TSP and the VRP. Moreover, the concept of the block type mutation is applicable to the problems defined over topological space which does not have metric system but only neighborhood system.

#	Condition (number of loads)	Number of trucks	Total tour length (Deviation)
1	700	29	1015km (2.6%)
2	900	41	1345km (2.8%)
3	1000	48	1612km (2.8%)

Table 3. VRP simulation result

6. Comparisons

A lot of methods to solve TSP are proposed for practical applications. In this section, our methods are compared with other methods.

Three types of the proposed methods (multi-world intelligent GAs) are tested, each of which has different stopping condition. MIGA1 (multi-world intelligent GA type 1) stops searching when it stops improving the solution. The computation time of MIGA2 is one tenth of that of MIGA1. The computation time of MIGA3 is 30 milliseconds. Experiments were conducted under the following computation environment. Namely, CPU is AMD Athlon 64 X2 3800+ 2GHz processor. It is almost the same performance as Athlon 64 3200+ 2GHz because of its execution on the single core mode with 1GB memory. The programs were written in C language, compiled by Microsoft Visual C++ .NET 2003 ver. 7.1.3091 with /O2 option (directing the execution speed preference), and executed on Windows XP Professional.

Yet, since other solution methods to be compared are executed on machines with different performance specification, it is necessary to take the difference into account. Therefore, referring to the statistical results of tests using RC5-72 benchmark (JLUG, 2008) for measuring the arithmetic processing speed, we obtained the spec difference correction coefficient (SDCC). This can be obtained by dividing the resultant value of the benchmark test executed on the experimental environments of other solution methods, by the resultant value of the benchmark test on our experimental environment. Through multiplying SDCC to the execution time of other solution methods, we calculated an assumed execution time on the same specification machine as ours.

As for the strict optimization method, Branch-and-cut and Dynamic Programming (DP) are proposed. These methods require long computation time though they can obtain optimal solutions. Some algorithms using DP can search very near-optimal solutions for the Euclidean TSP in polynomial time (Arora, 1998). However, even these algorithms take too long time for practical applications such as ours, and it seems too hard for ordinary system developers to modify or adjust them for coping with various particular requirements.

As for the approximate solution technique, various techniques are proposed. LK is famous as the heuristics search technique for TSP. However, LK and its improving methods (Lin & Kernighan, 1972; Yamamoto & Kubo, 1997) also take a long computation time though the optimality of obtained solutions is high and these methods are often incorporated with the meta-heuristics search such as SA, TA and GA.

Simulated Annealing (SA) and Tabu Search (TS) are known as the meta-heuristics search technique. Theoretically, SA (Kirkpatrick et al., 1983; Ingber, 1993; Miki et al., 2003) is said to be able to search very near-optimal solutions by decreasing the risk of falling into a local minimum. But practically, it is very difficult to adjust SA's parameters such as cooling speed for coping with various location patterns. Furthermore, SA usually takes a long computation time to get above-mentioned theoretical near-optimal solutions.

TS (Glover, 1989; 1990; Hooker & Natraj, 1995; Fang et al., 2003) usually needs a long computation time to get practically optimal solutions. Worse still, TS is said to be weak in maintaining solution diversity though it has strong capability for local search. However, TS is improved in these weaknesses by Kanazawa & Yasuda, 2004.

So-called random restart methods (Yanagiura & Ibaraki, 2000), which apply local search such as 2-opt for improving random initial solutions, can obtain near-optimal solutions. These include GRASP (Feo et al., 1994) or the elaborated random restart method (Kubota et al., 1999) that can guarantee responsiveness by limiting the number of repetitions. However,

according to our experiments, the above-mentioned elaborated random restart method needed about 100 milliseconds to solve 40 cities TSP and to guarantee less than 3% errors (Kubota et al., 1999).

As for the Genetic Algorithms (GA) to efficiently solve TSP, various techniques are proposed. GA applied solving methods using the edges assembly crossover (EAX) (Nagata & Kobayashi, 1999) and the distance-preserving crossover (DPX) (Whiteley & Starkweather, 1989) could get highly optimized solutions in case of very-large-scale TSPs (with 1000-10000 cities) (Tamaki et al., 1994; Baraglia et al., 2001; Tsai et al., 2004; Nguyen et al., 2007). These crossover methods examine characteristics of parent's tour edge to strictly inherit to children. However, since these crossover operations take long computation time for analyzing edges, using it for not-very -large-scale TSP is often inefficient.

In reference (Baraglia et al., 2001), two kinds of methods are compared in many cases. It shows that Cga-LK is advantageous to 300-10000 cities TSP, but Random-LK is advantageous to 198 cities TSP. Therefore, the solution that can efficiently solve TSP of 1000 cities or more can not necessarily efficiently solve TSP of about 100 cities. As to TSP of our intended scale (with 10s to 100 cities), in reference (Baraglia et al., 2001), a TSP lin105 is solved with 1.77% average error rate in 231seconds. The performance specification of this experimental environment is 200-MHz PentiumPro PC running Linux 2.2.12. Since this SDCC is 0.048, the solving time on our experimental environment is 11.088 seconds. Moreover, in reference (Cheng et al., 2002), the performance comparison experiments were conducted using various crossover operators. Even when the best crossover operator is used, average error rate is 3.1% and computation time is 750 seconds by SUN SPARC Ultra-5 10 machine. Since this SDCC is 0.065, the solving time on our environment is 48.75 seconds. Meanwhile, our MIGA1 obtains the optimal solution within 1.11 seconds, and MIGA2 obtains a solution with average error rate 0.31% in 0.15 seconds.

A GA method with the same purpose as ours (aiming to obtain high quality approximate solution as fast as possible for 10s - 100s cities TSPs) is proposed by Yan et al., 2007. To compare the proposed methods with GA by Yan and TS by Kanazawa, the proposed methods are tested on nine benchmark problems in TSPLIB whose number of cities ranges from 70 to 280. Each problem is solved 100 times.

Table 4 presents the SDCC of each method. And Table 5 presents the experimental results obtained by applying MIGA to the above nine benchmark problems and results corrected by using SDCC. The mark "-" on the Table 5 indicates no data. The digits (e.g. 70) contained in the name（e.g. st70） of TSP indicate the number of cities.

Results of GA by Yan are compared with those of MIGA. Results for the problem st70 indicate MIGA can obtain the solution having almost the same accuracy as GA by Yan, while the computation time is 20%. Results of problem eil76 indicate MIGA can obtain always optimal solution though the average error rate of GA by Yan is 1.184% within the same computation time. Specific results of problem a280 indicate MIGA can obtain solutions whose average error rate is 4% which is lower than that of GA by Yan and the computation time is 7% compared with that of GA by Yan.

Next, results of TS by Kanazawa and MIGA are compared. Results indicate MIGA obtained almost the same accuracy solutions as TS by Kanazawa, while the computation time is 19% for the problem pr107, 63% for pr144, 45% for pr152, and 5% for pr226.

Overall results show that MIGA is more effective than GA by Yan and TS by Kanazawa in solving the above mentioned nine TSP benchmark problems whose number of cities ranges from 70 to 280.

Results of MIGA3 show the average error rate is around 1% and even the worst case error is under 3.5%, to solve, within 30 milliseconds, eight TSP benchmark problems whose number of cities ranges from 70 to 226.

	GA by Yan	TS by Kanazawa
Spec of computing machines	CPU：Pentium4 2.4GHz, メモリ：256MB	CPU：Pentium4 2.55GHz, メモリ：1GB (DDR266)
Spec Difference Correction Coefficient (SDCC)	0.519	0.590

Table 4. Spec Difference Correction Coefficient (SDCC)

Name of TSP	Average error rate from optimal solution [%] (Average execution time [sec])					
	GA by Yan	TS by Kanazawa	MIGA1	MIGA2	MIGA3	
					Average	Worst
st70	0.312 (0.348)	-	0 (0.750)	0.370 (0.074)	0.400 (0.030)	1.333
eil76	1.184 (0.602)	-	0 (0.844)	1.914 (0.080)	2.193 (0.030)	2.974
kroA100	0.016 (0.877)	-	0 (0.937)	0.176 (0.131)	0.873 (0.030)	1.588
pr 107	-	0.290 (0.826)	0.086 (0.985)	0.256 (0.156)	0.449 (0.030)	0.899
pr 136	0 (3.690)	0.190 (4.378)	0.624 (2.016)	1.067 (0.233)	2.460 (0.030)	3.481
pr 144	0 (4.136)	0.019 (4.685)	0 (2.625)	0.149 (0.284)	1.665 (0.030)	2.896
pr 152	-	0.120 (7.558)	0.153 (3.375)	0.551 (0.338)	1.175 (0.030)	2.77
pr 226	-	0.510 (12.685)	0 (3.125)	0.515 (0.652)	1.854 (0.030)	2.566
a280	10.770 (17.371)	-	3.180 (10.453)	6.572 (1.100)	-	-

Table 5. Results compared with related works on TSPLIB

7. Conclusion

In this chapter, an intelligent GA method for solving the TSP was proposed and evaluated. This is applicable to the optimization of various distribution systems such as the parcel and letter delivery as well as large-scale distribution networks that requires repetitive interactive simulations. This kind of application requires responsiveness as well as optimality, for example, solving a TSP with expert-level accuracy within 30 milliseconds.

In order to guarantee expert-level solutions for various kinds of delivery location patterns, the high-speed GA world was combined with the intelligent GA world. The high-speed GA world comprises the random NI method and the 2opt-type mutation. And this high-speed GA mainly uses simple general heuristics. The intelligent GA world includes the random method, the multi-step NI method, and the block-type mutation. And particular knowledge was incorporated in this intelligent GA to overcome the weakness of the high-speed GA. Namely, to cope with delivery location patterns for which the high-speed GA cannot guarantee expert-level solutions, this intelligent GA has rather problem-oriented knowledge.

According to our experiment, in case of using the former high-speed GA, 23 test cases out of 20000 test cases had more than 3% of errors compared to the optimal solution. However, our proposed multi-world intelligent GA method (which comprises the high-speed GA world and the intelligent GA world) could solve each of all 20000 test cases within 30 milliseconds at expert-level accuracy (less than 3% errors).

Our experimental results showed that the proposed methods enable to solve TSPs with responsiveness and optimality necessary for a large-scale distribution network's interactive simulation.

8. References

Arora, S. (1998). Polynomial Time Approximation Schemes for Euclidean TSP and Other Geometric Problems, *Journal of the ACM*, Vol. 45, No. 5, pp. 753-782

Baraglia, R.; Hidalgo, J.I. & Perego, R. (2001). A hybrid heuristic for the traveling salesman problem, *IEEE Transactions on Evolutionary Computation*, Vol. 5, Issue 6, pp. 613-622

Bertsekas, D. P. (1987). *Dynamic Programming: Deterministic and Stochastic Models*, Englewood Cliffs, NJ: Prentice-Hall

Cheng, C.; Lee, W. & Wong, K. (2002). A genetic algorithm-based clustering approach for database partitioning, *IEEE Transactions on Systems, Man and Cybernetics*, Part C, Vol. 32, Issue 3, pp. 215-230

Fang, Y.; Liu, G.; He, Y. & Qiu, Y. (2003). Tabu search algorithm based on insertion method, *Proc. of the 2003 International Conference on Neural Networks and Signal Processing*, pp. 420-423

Feo, T.A.; Recende M.G.C. & Smith S.H. (1994). A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set, *Operations Research*, Vol. 42, No. 5, pp. 860-878

Glover, F. (1989). Tabu Search - Part I, *ORSA Journal on Computing*, Vol. 1, pp. 190-206

Glover, F. (1990). Tabu Search - Part II, *ORSA Journal on Computing*, Vol. 2, pp. 4-32

Grotschel, M. & Holland, O. (1991). Solution of large-scale symmetric traveling salesman problems, *Mathematical Programming*, Vol. 51, pp. 141-202

Hooker, J.H. & Natraj, N.R. (1995). Solving a General Routing and Scheduling Problem by Chain Decomposition and Tabu Search, *Transportation Science*, Vol. 29, No. 1, pp. 30-44

Ingber, L. (1993). Simulated Annealing Practice versus Theory, *Journal of Mathl. Comput. and Modelling*, Vol. 18, No. 11, pp. 29-57

JLUG (2008). Benchmark - JLUG RC5-72 Cracking, *http://www.orange.co.jp/~masaki/rc572/*

Kanazawa, T. & Yasuda, K. (2004). Proximate Optimality Principle Based Tabu Search, *IEEJ Transactions on Electronics, Information and Systems*, Vol. 124, No. 3, pp. 912-920

Kirkpatrick, S.; Gelatt, C. D. & Vecchi, M. P. (1983). Optimization by Simulated Annealing, *Science*, Vol. 220, Num. 4598, pp. 671-680

Kolen, A. & Pesch, E. (1994). Genetic Local Search in Combinatorial Optimization, *Discrete Applied Mathematics*, Vol. 48, pp. 273-284

Kubota S.; Onoyama T.; Onayagi K. & Tsuruta S. (1999). Traveling Salesman Problem Solving Method fit for Interactive Repetitive Simulation of Large-scale Distribution Network, *Proc. of IEEE SMC'99*, pp. 533-538

Laporte, G. (1992). The Vehicle Routing Problem: An overview of exact and approximate algorithms, *European Journal of Operational Research*, Vol. 59, pp. 345-358

Lawer, E.; Lenstra, J.; Rinnony, K. & Shmoys, D. (1985). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley & Sons, ISBN 0471904137

Lin, S. & Kernighan, B.W. (1972). An effective heuristic algorithm for the traveling salesman problem, *Operations Research*, Vol. 21, No. 2, pp. 498-516

Miki, M.; Hiroyasu, T. & Jitta, T. (2003). Adaptive Simulated Annealing for maximum temperature, *Proc. of IEEE International Conference on Systems, Man and Cybernetics 2003*, Vol. 1, pp. 20-25

Nagata, Y. & Kobayashi, S. (1999). The Proposal and Evaluation of a Crossover for Traveling Salesman Problems: Edge Assembly Crossover, *Journal of Japan Society for AI*, Vol. 14, No. 5, pp. 848-859

Nguyen, H.D.; Yoshihara, I.; Yamamori, K. & Yasunaga, M. (2007). Implementation of an Effective Hybrid GA for Large-Scale Traveling Salesman Problems, *IEEE Transactions on Systems, Man and Cybernetics*, Part. B, Vol. 37, Issue. 1, pp. 92-99

Onoyama, T.; Kubota, S.; Oyanagi, K. & Tsuruta, S. (2000). A Method for Solving Nested Combinatorial Optimization Problems, *Proc. of IEEE SMC2000*, pp. 340-345

Tamaki, H.; et al. (1994). A comparison study of genetic codings for the traveling salesman problem, *Proc. of the 1st IEEE Conference on Evolutionary Computation*, pp. 1-6

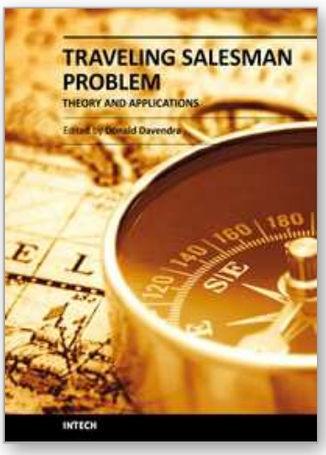
Tsai, H.; Yang, J.; Tsai, Y. & Kao, C. (2004). An evolutionary algorithm for large traveling salesman problems, *IEEE Transactions on Systems, Man and Cybernetics*, Part. B, Vol. 34, Issue. 4, pp. 1718-1729

Whiteley, D. & Starkweather, T. (1989). Scheduling Problem and Traveling Salesman: The Genetic Edge Recombination Operator, *Proc. of ICGA'89*, pp. 133-140

Yamamoto, Y. & Kubo, M. (1997). *Invitation to Traveling Salesman Problem*, Asakura Syoten, Tokyo

Yan, X.; Liu, H.; Yan, J. & Wu, Q. (2007). A Fast Evolutionary Algorithm for Traveling Salesman Problem, *Proc. of Third International Conference on Natural Computation (ICNC 2007)*, Vol. 4, pp. 85-90

Yanagiura, M. & Ibaraki, T. (2000). On Metaheuristic Algorithms for Combinatorial Optimization Problems, *Transactions of the IEICE*, Vol. J85-D-II, No. 8, pp. 3-25



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
open science | open minds

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

IntechOpen