Iterated Local Search Approach using Genetic Transformation to the Traveling Salesman Problem

Kengo Katayama **Hiroyuki Narihisa**
Department of Information and Computer Engineering, Okayama University of Science.
1 - 1 Ridai-cho, Okayama 700-0005, Japan.
{katayama, narihisa}@ice.ous.ac.jp

Abstract

When giving different approximate solutions that are near the optimal solution for a combinatorial optimization problem, these solutions may share several important or common parts. This empirical conjecture is often employed in developing good algorithms for solving combinatorial optimization problems. In this paper, we propose a new iterated local search (ILS) approach incorporating this conjecture for the symmetric traveling salesman problem. To escape from local optimum found by a local search procedure to another, standard ILS algorithms generally have an useful technique called the *double-bridge* move. However, in our approach we deal with two approximate solutions, which contain many edges which are not shared parts of these solutions. These edges are cleverly reconnected to create a newly escaped solution. From our experimental results, it was observed that our ILS algorithms could find better solution qualities with fewer iterations than standard ILS algorithms for well-known benchmarks of the TSPLIB. In particular, we showed that one algorithm combined with the Lin-Kernighan heuristic was a very high-performance approach.

1 INTRODUCTION

Many combinatorial optimization problems belong to the class NP-hard. It is generally believed that these problems can not be solved to optimality within polynomially bounded computation times. In such a situation, it is important to have approximation algorithms that can find optimal or near-optimal solutions within reasonable times. Almost all of the algorithms have been proposed based on analogies with processes in nature, biological evolution, etc. Recently, such algorithms have been developed for the traveling salesman problem (TSP), one of the most known combinatorial optimization problems.

The objective of the TSP is to find, given a set $\{c_1, c_2, ..., c_n\}$ of *cities* and for each pair $\{c_i, c_j\}$ of distinct cities a *distance* $d(c_i, c_j)$ [10], a permutation π (*Hamiltonian cycle*) of the cities that minimizes the following quantity;

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)}).$$

In this paper, we concentrate on the *symmetric* TSP, where the distances satisfy the condition $d(c_i, c_j) = d(c_j, c_i)$ for $1 \leq i, j \leq n$. The number of the Hamiltonian cycles, i.e. the size of feasible solutions, is $(n-1)!/2$.

For the TSP, many approximation algorithms have been proposed such as the simulated annealing, tabu search, genetic algorithms, ant colony, and neural networks. Most of these algorithms with various parameters to search are combined with a local search procedure to find good solutions. To have a chance of finding good solutions we can start the local search procedure many times with different starting solutions, e.g., random solutions. A more reasonable idea is not to restart with a completely new starting solution but only to perturb the current local optimal solution. In general this approach is called the *iterated local search* (ILS). The ILS algorithm for the TSP includes a typical local search algorithm such as the 2-Opt, 3-Opt, or Lin-Kernighan (LK) [14] heuristics used as a main engine of search. To escape from the current local optimal solution, a *double-bridge* 4-change move is very often used as an "escape technique", i.e., a technique to escape from local optimum found by a local search algorithm. This technique removes only four edges from a solution and replaces them in order to obtain a non-sequential move. The ILS based on the LK heuristic has been recognized as the most effective approach to the TSP since the beginning of 1990s [9, 10, 15].

In this paper, we propose new ILS algorithms that escape from a current local optimal solution using a technique of *genetic transformation* based on a genetic analogy. The genetic transformation (GT) acts as an intermediary which creates new appropriate solutions in order to apply repeatedly a local search procedure. The GT works so that partially succeeded edges shared on only two solutions of the TSP are not broken, and many other edges are reconnected to create the new solution which transforms the current best (elite) solution of the local optimum. Therefore, the GT technique may be interpreted as a crossover that is a connatural technique in the genetic algorithm. From this point of view, we call the ILS a *Genetic Iterated Local Search* (GILS) algorithm. We test performances of each GILS combined with the typical local search heuristics for several benchmarks for which the optimal tour length is known. From our experimental results, we observed that the GILS algorithms could find better solutions with fewer iterations than standard ILS algorithms, and the GILS combined with the LK heuristic obtained high-quality solutions within reasonable computation times.

The paper is organized as follows: Sect. **2** describes standard ILS and reviews impressive other approaches to the TSP. In Sect. **3** we describe our GILS using the genetic transformation and give a fundamental GT form. In Sect. **4** performances of our approach are tested and compared with standard and impressive approaches, and Sect. **5** contains concluding remarks.

2 REVIEW

Several researchers have proposed high-performance heuristic approaches to the TSP. These approaches are based on improving the current single solution (or multiple solutions) by a greedy search and other concepts for the neighborhood of the solution. Here, we review well-known approaches that can find optimal or very good approximate solutions, e.g., iterated local searches and genetic algorithms.

2.1 STANDARD ITERATED LOCAL SEARCH ALGORITHMS

Basically the iterated local search algorithms yield good approximate solutions based on dealing with a single solution rather than many solutions which are mostly used in genetic algorithms.

Martin, Otto, and Felten proposed iterating methods called the "Large-Step Markov Chain" algorithm for the TSP [15]. To escape from local optima found by the local search algorithm, they used a technique that removes four edges from a solution and replaces them in order to obtain a non-sequential move [14] (this technique was called a "double-bridge", see [15] for more detail). To escape from local optima found by typical local search algorithms, which perform only sequential moves, the double-bridge technique is the most natural choice. Johnson investigated an iterating method based on the LK using the technique of the double-bridge move [9]. His iterated local search (so-called "Iterated Lin-Kernighan") algorithm produced successful results with very fast computation times [10]. Moreover, Hong, Kahng, and Moon proposed an improved version of the Large-Step Markov Chain, and investigated the relationship between strength of the perturbation (random k-change moves over the range of values $k \geq 4$) and the Large-Step Markov Chain's performance [8]. Almost all of these powerful iterating methods substantially employ the double-bridge or k-change move (where k is fixed) techniques for the current single solution. Furthermore, Codenotti, Manzini, Margara, and Resta reported the performance of techniques (double-bridge and other moves) to escape from the local optima found by local search procedures of the TSP [4]. They indicated that modified techniques to escape from the local optima produced better solutions for iterating local search approach.

A basic concept of these ILS algorithms is to perform a concentrative search around good solutions, because very good solutions are located around other good solutions [3]. Generally, the local search algorithms find local optimal solutions, i.e., the local search is run until no possibility for decreasing the tour length can be found, and the cost of the solution is generally good. In order to transform other search points, we must escape from the local optima by using an appropriate technique.

A simple illustration of the double-bridge is shown in Fig. 1. A left side of the figure is a state of solution before performing the double-bridge. Alphabets, a, b, ..., h, are city names. Only four edges (a,b), (c,d), (e,f), (g,h) on the solution are changed to other edges (a,f), (b,e), (c,h), (d,g), as shown in a right side. The double-bridge move is randomly performed, and it is only the 4-change move that cannot be obtained by a sequential move such as two or three opt. The general flow of the ILS algorithm is described as follows:

```
**Standard Iterated Local Search**
1 Generate an initial solution $S$ (locally optimized by a local search algorithm).
2 Do the following for a given number of iterations.
   2.1 Perform a double-bridge move on $S$, obtaining $S'$.
   2.2 Run a local search on $S'$, obtaining $S''$.
   2.3 If $Cost(S'') < Cost(S)$, then set $S = S''$.
3 Return $S$.
```

In Step 2.1, the double-bridge move is applied to the current best solution S. In Step 2.2, the solution S'' is found by a local search starting from S' obtained in Step 2.1. In Step 2.3, $Cost()$ denotes an evaluation function for the length of a TSP solution, i.e. the

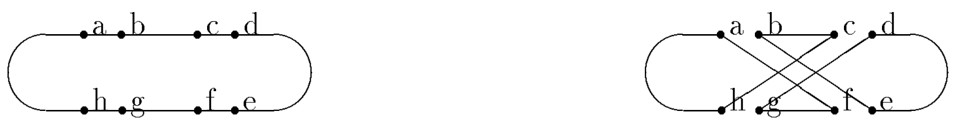

Figure 1: An example of the double-bridge 4-change move.

tour length is obtained by the quantity explained in the Introduction. This process is repeated until the terminating condition in Step 2 is satisfied.

2.2 GENETIC ALGORITHMS

On the other hand, the genetic algorithm is based on principles of natural selection and genetics, and several researchers proposed efficient genetic algorithms for solving the optimization problem. In particular, because the crossover is the most important and connatural operator among several genetic operators, many crossovers that efficiently produce good solutions were developed. Generally, the crossover operator works as follows. By using a pair of solutions (parents) among those selected in a previous generation, new solutions are created using a mechanism that inherits the characters from the parents.

Nagata and Kobayashi proposed a high-power crossover operator called *edge assembly crossover* (EAX) [19]. The GA using EAX without any TSP local search procedures could obtain optimal solutions with high probability even for over 2000-city instances. Freisleben and Merz developed a high-performance genetic algorithm combined with the LK heuristic and reported impressive results for the TSP [6, 16]. Their algorithm exploits the observation that the solution space of the TSP has a "big-valley" structure [3]. To search hopeful space, *distance preserving crossover*, and the double-bridge mutation are employed. A similar search concept using both the useful crossover operator (modified *maximal preservative crossover* [18]) and the double-bridge mutation, the evolutionary algorithm with a linear distributed population structure, Asparagos96, was developed by Gorges-Schleuter [7]. This algorithm also found high-quality solutions.

3 GENETIC ITERATED LOCAL SEARCH

In the GILS algorithm, the genetic transformation (GT) technique acts as an intermediary which creates an appropriate solution transformed the current best (elite) solution in order to apply repeatedly a local search procedure, i.e., in order to move from local optima found by the local search to other search points. The GT technique for the TSP is motivated by the recent observation of Hong et al [8]. They suggested that the double-bridge move which reconnects only four edges was not optimum, and the best k-change move depended on both the local search procedure and the type of instance. Our GT works by reconnecting many edges which are significantly broken using useful informations from two solutions.

3.1 FRAMEWORK

To implement the GILS, we need multiple solutions because the GT substantially needs at least two solutions. In our GILS for the escape technique, we deal with the best previously found solution and a current solution found by the local search algorithm in each iteration. These solutions will be suitable because they should share many good edges or subtours according to the observation argued by Mühlenbein [18]. A simple framework of the GILS is outlined as follows:

```
Genetic Iterated Local Search
1 Generate two different solutions $S$ and $T$ (locally optimized by a local search algorithm).
2 Do the following for a given number of iterations.
  2.1 Perform $GT(S,T)$, obtaining $T'$.
  2.2 Run a local search on $T'$, obtaining $T''$.
  2.3 If $Cost(T'') < Cost(T)$, then set $T = T''$, and set $S = T''$.
3 Return $T$.
```

In Step 1, the first approximate solutions, S and T, are obtained by a local search starting from random solutions[1]. In Step 2.1, we perform the function GT(a current solution S, the best solution T) to create a new solution T', and in Step 2.2 the local search procedure finds a local optimal solution. In Step 2.3, T'' found in Step 2.2 must be copied to the solution S for using our escape technique GT in the next iteration. Therefore, a state of the new solution returned from the GT is different in every iterations, and is locally optimized by the local search. This process is repeated until the terminal condition in Step 2 is satisfied.

3.2 GENETIC TRANSFORMATION ESCAPE TECHNIQUE

We first describe a fundamental form of the GT technique. We define two solutions, a good solution S_a and another solution S_b. S_b is a local optimal solution, and a cost of S_b is better than that of S_a. The form consists of following conjectures:

- Two solutions, S_a and S_b, contain "common informations" each other.

[1]It is also possible to use better solutions created by tour construction heuristics instead of random solutions.

- The common informations between S_a and S_b are similar to sub-informations of an unknown optimum state.
- It needs that new solutions, which consist of the common informations and other parts between the solutions (S_a and S_b), are created by basing on "peculiar concepts" for solving a particular optimization problem. But new different solutions are required.

To design the GT for various optimization problems, it is necessary to fitly determine "common informations" between the solutions and "peculiar concepts" for the problem, with an intention to move to hopeful regions of the solution space. It depends on a problem and a coding of the solution for the problem.

For the TSP, the GT is realized as follows. Each of two solutions (a good solution S_a and another (better) solution S_b) on n-city TSP is the set of cities 1, 2,..., n. S_b is always a local optimum, and the cost of S_b is always better than $Cost(S_a)$, i.e., we can define that S_b is the best previously found solution during our approach process. Assuming that S_b may share similar information with the exact optimal solution, we can assume that S_a may also share similar information. Therefore, both S_b and S_a may be similar except for several edges. If edges between S_b and S_a differ by only k edges, almost all other edges $(n-k)$, called *shared edges*, should be worth preserving for a transformation, because the shared edges may contain dominant characters that will consist of subsets of the optimal solution. In other words, to intensively explore hopeful regions of the solution space, only unshared k edges should be changed to other appropriate edges by an useful operation based on an important concept for solving the TSP. For more detail, the number of shared edges between S_b and the exact optimal solution may be intuitively larger than the case of S_a. Whenever transforming from the best local optimum S_b by our GT, the shared edges should not be broken, and only k edges should be reconnected to create a new solution that is not a local optimum in order to search by a tour improvement local search heuristic.

In addition, it is important to define the shared edges of $(n-k)$. We define these as *complete subtours* [11]. The complete subtour consists of the several shared edges, and a set of cities includes the same cities in the exact same order rather than a different order. In genetic algorithms, the complete subtours have been used as the inherited characters to create offspring in the crossover operators, e.g., crossovers using complete subtours were proposed by Freisleben et al. [6] and [11]. Therefore, the shared edges of $(n-k)$ consist of multiple complete subtours. One of the complete subtours contains a set of adjacent edges, which consist of a subset of the shared edges on two solutions. In the TSP graph, we can implicitly interpret as if each complete subtour had only two nodes on the graph, i.e., broadly speaking, we can ignore the inner part of the adjacent edges in each complete subtour when performing the GT technique. A function GT is described below.

```
Function $GT(S,T)$
1 Perform a random 4-change move on $S$, obtaining $S'$.
2 Enumerate all complete subtours on two solutions, $S'$ and $T$.
3 Choose a starting city $i$ randomly from the city on either side of each subtour or other cities not contained in these subtours.
4 Do the following until a new different solution $T'$ is created.
  4.1 Make a candidate-list except for used cities and subtours.
  4.2 Find the nearest candidate city $j$ to $i$.
  4.3 Connect city $j$ to $i$, and set $i = j$. (if city $j$ is from either side of a subtour, connect the subtour to $i$, and set $i =$ the city on the other side of the subtour.)
5 Return $T'$.
```

In our GT, we use the best solution T and a good solution S as described above. Only a single solution is returned from the GT function. In Step 1 of the function, a new solution S' is obtained by performing a random 4-change move on S. The reason for Step 1 is described later. In Step 2, we perform an $O(n)$ time enumeration algorithm [11] to enumerate all common subtours on two solutions. If any subtours do not exist on two solutions, there is no substantial gain to perform a crossover which uses the subtours for creating a new solution [11]. The same point was also described simply in [6]. Therefore, we also take into account this point because the GT uses the same type subtours. To certainly count several complete subtours and help for creating the new solution, we perform the random 4-change move, which removes four edges on the current solution S and reconnects four other edges randomly selected, before enumerating the subtours in Step 2. Although we use the random 4-change, it is possible to perform a random k-change move also ($k > 4$, but it should not be very large). In other words, it can be interpreted as a special case such as the mutation operation to the solution used in the GT. Good subtours of S may be slightly broken by this mutation. However, this operation contributes to a progress of our approach and a break-through of exploratory limitations that are likely when solutions obtained by the search are located extremely near the optimum state.

Step 3 of the GT function randomly chooses a city from first or last cities on each of the subtours or other cities, which are not enumerated as subtours. In Step 4.1, a candidate-list contains the information of the candidate cities except for previously used cities and invalid cities, which have a very long distance. Step 4.2 finds the nearest candidate city on the list by the useful operation such as the nearest neighbor heuristic, and then the candidate city is connected to be a valid solution while performing Step 4. Therefore, all the unshared edges are not always reconnected to other new edges from edges previously appeared on the solutions. However, several new edges, which are not appeared on the solutions, are often linked by the nearest neighbor operation. This advantage must be important as observed in [19] of Nagata et al.

By using this GT technique, the new escaped solution does not go too far from the best solution, and the distance of a new solution can be mostly between distances of the solutions used in the GT. That is, the GT is biased so as to obtain the new solution slightly away from the best solution rather than an improved solution in most cases. In our observations, the shared edges (or complete subtours) ranging from roughly 90 to 99% are transmitted to the new solution, and the number of enumerated complete subtours depends on the size of instances and the state of two solutions. By increasing the size of instance, the number of subtours also increases substantially, and the number of unshared edges also increases. Therefore, when applying to very large instances, the GT technique and local search procedures may spend more computation times according to the number of the unshared edges than the case of the double-bridge technique, which uses only four edges. A similar point was described in Sect. 6.3 of [10]. It is important to consider a trade-off on that point, i.e., between obtained qualities and computation times by each ILS.

From these, this GT is interpreted as an adaptive k-change move technique (the number of k in the standard technique is always fixed as described in Sect. **2**) because the number of k is changeable according to the states between the best solution and another one via the random 4-change move for good current solutions in each iteration. The GT such as the adaptive k-change move is incorporated into our approach, and the GILS is achieved.

3.3 TSP LOCAL SEARCH ALGORITHMS

We can combine with any TSP local search heuristics for the GILS approach. In this paper, we employ the 2-Opt, 3-Opt, and Lin-Kernighan (variable r-Opt) heuristic algorithms as main search engines. In particular, the LK algorithm has been known to be the best improvement heuristic for the TSP.

These TSP local search heuristics improve the current tour by changing the appropriate shorter edges from longer ones until no possibility for decreasing the tour length can be found in the neighborhood $\mathcal{N}$, i.e., set of all possible "neighbor" tours. Given a tour $x = \{(c_1, c_2), (c_2, c_3), ..., (c_{n-1}, c_n), (c_n, c_1)\}$, where (c_i, c_{i+1}) denotes the edge connecting the ith point c_i and the $(i+1)$th point c_{i+1} in the tour, the r-Opt ($r \geq 2$) neighborhood is defined by $\mathcal{N}_{r-opt}(x) = \{x^{'}|x^{'}$ is a tour obtained from x by removing r edges and adding the same number of other appropriate edges$\}$. See [10] and [14] for implementation details of the heuristics. In the naïve implementations, the r-Opt algorithms generally take $O(n^r)$ time, i.e., the size of the neighborhood grows polynomially with r. However, it is possible to reduce dramatically the enormous amount of computation time of these algorithms by using appropriate data structures and useful programming techniques. To implement fast algorithms, we use the *neighbor-list* implementation of size 40 for the each node that are initialized by nearest neighbor queries on the K-d tree, and incorporate the *don't look bits* technique into our local search algorithms [12], see [1, 2, 5, 10, 16] for details on these techniques.

4 EXPERIMENTAL RESULTS

In the experiment, we demonstrate performances of three GILS algorithms (2-Opt GILS, 3-Opt GILS, and LK GILS) for well-known TSP instances in the TSPLIB95 [21], so that closeness to optimality can be judged. Since the algorithms presented in this paper attempt only to find near-optimal solutions, it is important to evaluate the relation between the average quality of the solution and the computation time required by the algorithms. All computations of the algorithms written in C are performed on a Fujitsu S-4/5 workstation (microSPARCII 110MHz). We carry out ten runs for each instance. All initial solutions are generated randomly, and improved by each local search before performing each iterating search process.

4.1 2-OPT AND 3-OPT VERSIONS

Table 1 summarizes the best (best tour length) and average (average tour length) qualities obtained by standard 2-Opt ILS and 3-Opt ILS using the double-bridge move for the well-known 532-city instance [20], `att532` (the optimal tour length is 27686). In the table, time(s) denotes the average computation times in seconds for each iteration shown, and the results performed by the algorithms are after $n/10$, $n/10^{0.5}$, n, and $2n$ iterations, where n denotes a number of cities. These iteration numbers are derived from those of Johnson and McGeoch [10]. The results of 2-Opt and 3-Opt versions of our GILS approach for `att532` are shown in Table 2.

From these tables, we observed that the GILS algorithms obtained better solutions than standard ILS algorithms. Similar observations of the 2-Opt version were reported in [13]. The ability of optimization of the GILS algorithms was performed faster than standard algorithms when measuring fixed same iterations. For example, the average quality of 3-Opt GILS at n iterations was clearly better than 3-Opt ILS at $2n$ iter-

Table 1: Best and average percentage excess over the optimal tour length for 2-Opt ILS and 3-Opt ILS after $n/10$, $n/10^{0.5}$, n, and $2n$ iterations. ($n = 532$)

att532	2-Opt ILS			3-Opt ILS		
iterations	best	average	time(s)	best	average	time(s)
$n/10$	4.302 (28877)	6.074 (29367.7)	2	1.719 (28162)	2.788 (28458.0)	4
$n/10^{0.5}$	2.853 (28476)	4.651 (28973.6)	7	0.921 (27941)	1.915 (28216.2)	11
n	2.384 (28346)	3.276 (28593.0)	19	0.686 (27876)	1.357 (28061.7)	33
$2n$	1.853 (28199)	2.239 (28305.8)	37	0.560 (27841)	1.094 (27988.9)	64

Table 2: Best and average percentage excess over the optimal tour length for 2-Opt GILS and 3-Opt GILS after $n/10$, $n/10^{0.5}$, n, and $2n$ iterations. ($n = 532$)

att532	2-Opt GILS			3-Opt GILS		
iterations	best	average	time(s)	best	average	time(s)
$n/10$	1.882 (28207)	3.534 (28664.3)	3	1.481 (28096)	2.082 (28262.6)	5
$n/10^{0.5}$	1.665 (28147)	2.532 (28387.1)	10	0.842 (27919)	1.231 (28026.9)	15
n	1.055 (27978)	1.901 (28212.4)	28	0.368 (27788)	0.781 (27902.1)	41
$2n$	0.708 (27882)	1.534 (28110.7)	53	0.238 (27752)	0.665 (27870.1)	80

ations. The average computation time of 3-Opt GILS at n iterations was faster than 3-Opt ILS at $2n$ iterations, although the time of $2n$ iterations of the GILS was slightly slower than the time of the ILS in the same iteration. In addition, we tested each of four algorithms for $10n$ iterations. The 2-Opt and 3-Opt ILSs obtained average qualities 1.57% and 0.55%, respectively. The GILSs found 0.98% and 0.36%, respectively. From these tests, we confirmed that each algorithm intrinsically could improve solution qualities, and our GILS approach could obtain even better ones with longer iterations. Even though 2-Opt and 3-Opt GILS algorithms demonstrated good prospects in comparison to the standard algorithms, they may not be sufficient to achieve much better solutions.

4.2 LK VERSION

Results for the LK GILS on the TSPLIB instances of 101 – 3795 cities are shown in Table 3 (a result of other instance d2103 tested is described in the end of this section). In this table, Opt/10runs denotes the number of optimal tours obtained by the LK GILS for ten runs. Optimal tours were always observed up to 2392-city instance, however the LK GILS obtained very high-quality solutions even for the largest 3795-city instance in this experiment.

In the early study of the Iterated Lin-Kernighan (ILK) algorithm using the double-bridge move, Johnson reported that the ILK found optimal tours of 318, 532, 2392-city instances, etc [9]. Here, we compare our results with those of the ILK recently presented in Table 16 of [10][2]. Our LK GILS clearly found better solutions with fewer iterations than the results of the ILK for several instances shown in [10]. This behavior is similar to previous results of 2- and 3-Opt versions. For example, the ILK for the instance of 532-city showed average qualities, 1.20%, 1.08%, and 1.03%, with corresponding numbers of iterations, $n/10$, $n/10^{0.5}$, and n, respectively. The average quality (of att532), here shown in Table 3, was 0.223% at only $n/10$ (i.e., 53) iterations within 13(sec) although the total computation time of the LK GILS was slightly slower than the ILK for att532 (roughly a factor of 4) when comparing at the final n iteration (Johnson et al. used 150 MHz MIPS processor).

[2]Table 16 of [10] is the result that intentionally sacrificed accuracy for ILK search. However, the ILK algorithm must be able to find good (or several optimal) solutions with larger iterations and computation times.

Merz and Freisleben reported impressive results of the genetic algorithm that is similar to one combined both the ILK and our approach [16]. Their algorithm obtained the average tour length 27698.4 for att532 within 290(sec) on 233 MHz DEC Alphastation. For the 783-city instance (optimal length is 8806), the algorithm obtained a better average length (8806.2) within 424(sec) than our result in Table 3. When giving $4n$ iterations (3132 iterations) for rat783, the LK GILS obtained an average tour length of 8806.2 for ten runs within an average computation time of 400(sec) in our additional test (optimal tours were found 9 times in 10 runs). Using another high-power GA, Nagata and Kobayashi reported a 0.034% average quality for att532 within 781(sec), 0.000% for rat783 within 3013(sec), and 0.006% for pr2392 within 33285(sec) on 200 MHz Pentium processor [19]. Our result of pr2392 was the same value with the high-power GA. Again we performed the additional test for att532. In the case of $4n$ iterations (2128 iterations), the LK GILS obtained an average tour length 27694.1 (0.029%) for ten runs within an average computation time of 440(sec) (optimal tours were found 5 times in 10 runs). Therefore, our approach will find even better solutions by increasing the iteration number within the limits of

Table 3: Best and average percentage excess over the optimal tour length for LK GILS after $n/10$, $n/10^{0.5}$, and n iterations.

`eil101`	n = 101	Opt/10runs: 10	
iterations	best	average	time(s)
$n/10$	0.000 (629)	0.302 (630.9)	<1
$n/10^{0.5}$	0.000 (629)	0.000 (629.0)	<1
n	0.000 (629)	0.000 (629.0)	2

`kroA200`	n = 200	Opt/10runs: 10	
iterations	best	average	time(s)
$n/10$	0.000 (29368)	0.164 (29416.2)	1
$n/10^{0.5}$	0.000 (29368)	0.000 (29368.0)	4
n	0.000 (29368)	0.000 (29368.0)	10

`lin318`	n = 318	Opt/10runs: 8	
iterations	best	average	time(s)
$n/10$	0.128 (42083)	0.410 (42201.4)	6
$n/10^{0.5}$	0.000 (42029)	0.162 (42096.9)	17
n	0.000 (42029)	0.057 (42053.0)	47

`pcb442`	n = 442	Opt/10runs: 9	
iterations	best	average	time(s)
$n/10$	0.014 (50785)	0.065 (50811.0)	8
$n/10^{0.5}$	0.000 (50778)	0.016 (50785.9)	23
n	0.000 (50778)	0.003 (50779.7)	65

`att532`	n = 532	Opt/10runs: 2	
iterations	best	average	time(s)
$n/10$	0.072 (27706)	0.223 (27747.6)	13
$n/10^{0.5}$	0.025 (27693)	0.092 (27711.6)	40
n	0.000 (27686)	0.056 (27701.5)	113

`rat783`	n = 783	Opt/10runs: 3	
iterations	best	average	time(s)
$n/10$	0.045 (8810)	0.153 (8819.5)	13
$n/10^{0.5}$	0.023 (8808)	0.073 (8812.4)	37
n	0.000 (8806)	0.022 (8807.9)	103

`pcb1173`	n = 1173	Opt/10runs: 1	
iterations	best	average	time(s)
$n/10$	0.047 (56919)	0.204 (57008.1)	43
$n/10^{0.5}$	0.002 (56893)	0.074 (56934.0)	118
n	0.000 (56892)	0.014 (56900.1)	319

`rl1323`	n = 1323	Opt/10runs: 1	
iterations	best	average	time(s)
$n/10$	0.020 (270254)	0.195 (270725.5)	113
$n/10^{0.5}$	0.010 (270226)	0.095 (270456.0)	355
n	0.000 (270199)	0.039 (270303.2)	1022

`pr2392`	n = 2392	Opt/10runs: 3	
iterations	best	average	time(s)
$n/10$	0.011 (378074)	0.051 (378226.0)	177
$n/10^{0.5}$	0.000 (378033)	0.016 (378093.9)	551
n	0.000 (378032)	0.006 (378054.1)	1635

`fl3795`	n = 3795	Opt/10runs: 0	
iterations	best	average	time(s)
$n/10$	0.129 (28809)	0.341 (28870.1)	2658
$n/10^{0.5}$	0.028 (28780)	0.189 (28826.4)	8933
n	0.021 (28778)	0.091 (28798.2)	26958

the time than the results shown in Table 3.

Recently, a very high-performance algorithm called the "Thermal Cycling" was developed by Möbius, Neklioudov, Díaz-Sánchez, Hoffmann, Fachat, and Schreiber [17]. This algorithm has roughly the same performance in comparison to ours for 442, 532, and 783-city instances. In particular, results of instances of `fl` type tested in [17] were extremely impressive. For example, the instance of `fl3795`, which is pathologically clustered, have been considered that it is very hard to optimize for neighbor search approaches. Therefore, we relatively need large computation times as presented in [10, 16]. From our result for `fl3795` on Table 3, we observed better approximate solutions (0.091%) on the average than results of Gorges-Schleuter (0.167%) [7] and Merz et al. (0.335%) [16], although our result was not as good as the result of Möbius et al [17].

Finally, we tested the LK GILS for the `d2103` instance, which has not been solved exactly yet [21]. The known lower and upper bounds for the optimum of the instance are [80330,80450]. The best tour length found by our LK GILS was 80450, which corresponded with the upper bound. In this case, we used 40 nearest cities for the neighbor-list implementation. The average computation time for ten runs was 3700(sec), and the average tour length of 80480.4 was found within 2103 iterations.

Our approach is a half-breed of the iterated local search and the genetic algorithm without a complicated setting of several parameters. In particular, one or two solutions rather than many solutions are used to find high-quality solutions. The genetic transformation technique combined both the crossover and mutation operators is performed to intensively explore the hopeful regions of the search space where better solutions may be found by the local search so that a diversity of the elite solution is obtained. From our results which very good average quality solutions were found from early iterations, we confirmed that the GILS was one of the high-performance approaches to the TSP.

5 CONCLUSION

In recent years, many high-performance heuristics such as the iterated local search and genetic algorithms have been examined for the traveling salesman problem. By combining these algorithms, the performance in obtaining good approximate solutions can be increased. From this point of view, we have presented a new approximation approach "Genetic Iterated Local Search" based on the iterated local search.

From our experimental results, it would seem that an escape technique, which changes only four edges such as a standard one, is not always suitable. The number of edges reconnected by the genetic transformation technique in our GILS algorithms depends on states between the best previously found solution and another good solution. In this case, the number of edges is larger than the four edges usually changed by the double-bridge move, and the new solutions produced by our adaptive k-change move technique can be located onto promising regions of the search space where better solutions are likely to be found by the local search procedure. Consequently, the effectiveness of the GILS has been demonstrated.

There are several areas for future research, such as investigating the effects of a parallel implementation of the GILS, analyzing a detailed relation between the region of the search space and local search, and evaluating the performance of our approach for very large TSP instances.

Acknowledgements

The authors would like to thank David S. Johnson of AT&T Labs Research and Hiroshi G. Okuno of Kitano Symbiotic Systems Project for valuable discussions and Yasuhiro Senoo of FujiSoft ABC Inc. for useful suggestions on the implementations of GILS.

References

[1] J.L Bentley, "K-d trees for semidynamic point sets," Proc. 6th Annual ACM Symp. on Computational Geometry, pp.187–197, 1990.

[2] J.L Bentley, "Fast algorithms for geometric traveling salesman problems," ORSA Journal on Computing, vol.4, no.4, pp.387–411, 1992.

[3] K.D. Boese, A.B. Kahng, and S. Muddu, "A new adaptive multi-start technique for combinatorial global optimizations," Operations Research Letters, vol.16, pp.101–113, 1994.

[4] B. Codenotti, G. Manzini, L. Margara, and G. Resta, "Perturbation: An efficient technique for the solution of very large instances of the Euclidean TSP," Tech. Rep. TR-93-035, International Computer Science Institute, University of California at Berkeley, 1993.

[5] M.L. Fredman, D.S. Johnson, L.A. McGeoch, and G. Ostheimer, "Data structures for traveling salesmen," Journal of Algorithms, vol.18, pp.432–479, 1995.

[6] B. Freisleben and P. Merz, "New genetic local search operators for the traveling salesman problem," Proc. Parallel Problem Solving from Nature IV, pp.890–899, 1996.

[7] M. Gorges-Schleuter, "Asparagos96 and the traveling salesman problem," Proc. 4th IEEE Int. Conf. on Evolutionary Computation, IEEE Press, pp.171–174, 1997.

[8] I. Hong, A.B. Kahng, and B.-R. Moon, "Improved large-step Markov chain variants for the symmetric TSP," Journal of Heuristics, vol.3, no.1, pp.63–81, 1997.

[9] D.S. Johnson, "Local optimization and the traveling salesman problem," Proc. 17th. Colloquium on Automata, Languages, and Programming, pp.446–461, 1990.

[10] D.S. Johnson and L.A. McGeoch, "The traveling salesman problem: A case study," Local Search in Combinatorial Optimization, (E. Aarts and J.K. Lenstra, eds.), Wiley & Sons, pp.215–310, 1997.

[11] K. Katayama, H. Hirabayashi, and H. Narihisa, "Performance analysis of a new genetic crossover for the traveling salesman problem," IEICE Trans. Fundamentals, vol.E81-A, no.5, pp.738–750, 1998.

[12] K. Katayama and H. Narihisa, "New iterated local search algorithms using genetic transformation for the traveling salesman problem," Tech. Rep. of IEICE, COMP98-80, 1999.

[13] K. Katayama and H. Narihisa, "A new iterated local search algorithm using genetic crossover for the traveling salesman problem," Proc. 1999 ACM Symp. on Applied Computing, San Antonio, USA, Feb.28 – Mar. 2, pp.302–306, 1999.

[14] S. Lin and B.W. Kernighan, "An effective heuristic algorithm for the traveling salesman problem," Operations Research, vol.21, pp.498–516, 1973.

[15] O. Martin, S.W. Otto, and W. Felten, "Large-step Markov chains for the TSP incorporating local search heuristics," Operations Research Letters, vol.11, pp.219–224, 1992.

[16] P. Merz and B. Freisleben, "Genetic local search for the TSP: New results," Proc. 1997 IEEE Int. Conf. on Evolutionary Computation, IEEE Press, pp.159–164, 1997.

[17] A. Möbius, A. Neklioudov, A. Díaz-Sánchez, K.H. Hoffmann, A. Fachat, and M. Schreiber, "Optimization by thermal cycling," Physical Review Letters, vol.79, no.22, pp.4297–4301, 1997.

[18] H. Mühlenbein, "Evolution in time and space – the parallel genetic algorithm," Foundations of Genetic Algorithms, (G. Rawlins, ed.), Morgan-Kaufmann, pp.316–337, 1991.

[19] Y. Nagata and S. Kobayashi, "Edge assembly crossover : A high-power genetic algorithm for the traveling salesman problem," Proc. 7th Int. Conf. on Genetic Algorithm, pp.450–457, 1997.

[20] M. Padberg and G. Rinaldi, "Optimization of a 532-city symmetric traveling salesman problem by branch and cut," Operations Research Letters, vol.6, pp.1–7, 1987.

[21] G. Reinelt, http://www.iwr.uni-heidelberg/iwr/comopt/soft/TSPLIB95