A Fast TSP Solver Using GA on JAVA

Hiroaki SENGOKU
Systems Development Laboratory,
Hitachi, Ltd.
Kawasaki, 215 JAPAN

Ikuo YOSHIHARA
Graduate School of Engineering,
Tohoku University
Sendai, 980-77 JAPAN

Abstract

A hybrid algorithm using GA and heuristics for rapid solution of Travelling Salesperson Problems (TSP) is presented. We developed a JAVA program based on this algorithm. Visiting our web pages, everyone can easily try our TSP solver. Since JAVA programs are executable on many platforms, any one who design a new TSP algorithm can compare his own solver and ours on the same machine. Using our program as the criterion, TSP researchers can evaluate their algorithms objectively.

1 Introduction

The Genetic Algorithm (GA)[1] is an optimizing algorithm that is modeled after the evolution of organisms. Some of the GA's merits are that it can be easily developed because it does not require detailed knowledge about the problem, it can search globally, and it can adapt to the changing conditions in the problem.

Despite of these merits, GA is often slower than conventional methods such as heuristic searches. This is because GA does not utilize explicitly the knowledge of how to search for the solutions. Therefore, hybrid methods that combine GA with other techniques have been attempted.

The TSP solver we presented[2] is one of the hybrid methods. It uses GA and the 2opt method (section 2.1). Owing to the 2opt, it is much faster than other TSP solvers based on GA alone.

Generally, it is difficult to compare two approximate algorithms objectively. Some algorithms have the advantage of high speed, but have the disadvantage of a low success rate in finding the optimum solution.

The best way to compare two methods is by running them both on the same machine. But there are few such programs in public domain[4], and we have not found a program that can solve the problems listed in TSPLIB[5].

Consequently, we wrote a JAVA[1] program based on our proposed method, and made it public on our page on the World Wide Web (WWW). JAVA programs are executable on many platforms, so people who have developed a TSP solver can compare their programs and ours by running both programs on the same machine under equal conditions.

We also developed a graphical user interface. People can freely situate towns by using a mouse, and solve the problem they constructed. It's easy and fun. It's helpful for students studying the basics of computer algorithms.

In section 2, we explain our method. In section 3, we show the WWW page in which we present our TSP program.

[1] JAVA is a trademark of Sun Microsystems, Inc.

2 A TSP solver: 2optGA

We present a hybrid method[2] that uses GA and the 2opt method. In our GA, the 2opt method provides mutation, while the crossover operator provides the capability of jumping out from the local minima, where the solution often falls where only 2opt is used.

The algorithm consists of the following steps.

Initialization: Generation of M individuals randomly.

Natural Selection: Eliminate p_e% individuals. The population decreases by $M \times p_e/100$.

Multiplication: Choose $M \times p_e/100$ pairs of individuals randomly and produce an offspring from each pair of individuals. The population reverts to the initial population M.

Mutation by 2opt: Choose p_i% of individuals randomly and improve them by the 2opt method. The elite individual, or the individual that has the best fitness value in the population, is always chosen. If the individual is already improved, do nothing, because it cannot be further improved by 2opt.

We now describe the detail of each step.

2.1 Mutation by 2opt

The 2opt method is one of the most well-known local search algorithms among TSP solving algorithms. It improves the tour edge by edge and reverses the order of the subtour. For example, imagine a tour as shown in the upper part of Fig. 1. Remove the two edges $\overline{ab}$ and $\overline{cd}$, and reverse the order of the subtour (from b to c), and add the two edges $\overline{ac}$ and $\overline{bd}$. This gives us a tour as shown in the lower part of Fig. 1. The lower tour is shorter than the upper one because $\overline{ab} + \overline{cd} > \overline{ac} + \overline{bd}$.

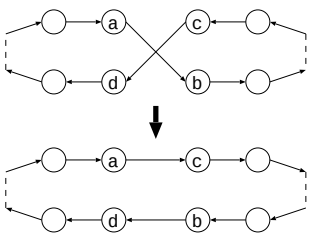


Figure 1: The 2opt method

We check every pair of edges, for example, $\overline{ab}$ and $\overline{cd}$. If $\overline{ab} + \overline{cd} > \overline{ac} + \overline{bd}$ holds, we improve them in the same way as shown in Fig. 1. Actually, if both $\overline{ac} > \overline{ab}$ and $\overline{bd} > \overline{cd}$ hold, then it is not necessary to check the edges. Therefore we can skip the pairs whose edges are far away from each other.

We repeat the procedures described above until no further improvement can be made.

2.2 Crossover in Multiplication

When we apply the 2opt method to a solution, the solution often falls into a local minimum. Then it cannot be improved further by the 2opt method. Consider two solutions that have fallen into different local minima. Potentially, each solution may have the best subtour for a different part of the tour. Then, we can make a better solution if we combine those best subtours appropriately. We cannot say, of course, which of the subtours are good, but after many trials, we can expect offspring solutions to be located in the valley of the global minimum as shown in Fig. 2.

We propose a new crossover operator that acquires the longest possible sequence of parents' subtours. We named it 'Greedy Subtour Crossover (GSX)'. We showed[3] by experiments that using the GSX, the solution can pop up from local minima more effectively than by using simulated annealing (SA) methods.

In the GSX, we use the path representation for a genetic coding. For example, the chromosome

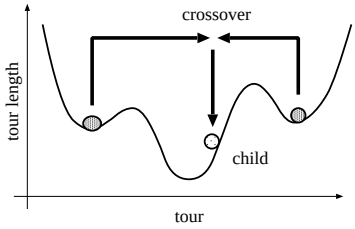


Figure 2: Pop-up from local minima.

$g = (D, H, B, A, C, F, G, E)$ means that the salesperson visits towns D, H, B, A, ..., E, successively, and returns to town D.

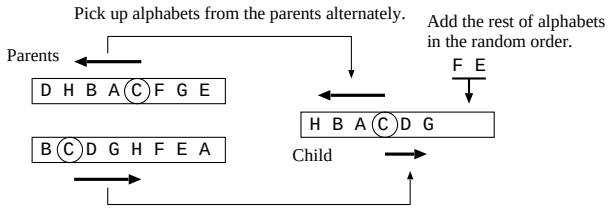


Figure 3: Greedy Subtour Crossover

Algorithm: Greedy Subtour Crossover
Inputs: Chromosomes $g_a = (a_0, a_1, \ldots, a_{n-1})$ and $g_b = (b_0, b_1, \ldots, b_{n-1})$.
Outputs: The offspring chromosome g.

procedure crossover(g_a,g_b) {
  $f_a \leftarrow$ **true**
  $f_b \leftarrow$ **true**
  choose town t randomly
  choose x, where $a_x = t$
  choose y, where $b_y = t$
  $g \leftarrow t$
  do {
    $x \leftarrow x - 1 \pmod n$,
    $y \leftarrow y + 1 \pmod n$.
    if $f_a =$ **true then** {
      if $a_x \notin g$ **then** {
        $g \leftarrow a_x \cdot g$,
      } **else** {
        $f_a \leftarrow$ **false**.
      }
    }
    if $f_b =$ **true then** {
      if $b_y \notin g$ **then** {
        $g \leftarrow g \cdot b_y$,
      } **else** {

$f_b \leftarrow$ **false**.
}
}
} **while** $f_a =$ **true or** $f_b =$ **true**
if $|g| < |g_a|$ **then** {
add the rest of towns
to g in the random order
}
return g
}

Note that "$\cdot$" in "$g \leftarrow a_x \cdot g$" is the concatenation operator, and that sentence means to add a_x before the chromosome g.

An example is shown in Fig. 3. Suppose that chromosomes of parents are $g_a = (D,H,B,A,C,F,G,E)$ and $g_b = (B,C,D,G,H,F,E,A)$. First, choose one town at random. In this example, town C is chosen. Then, $x = 4$ and $y = 1$ because $a_4 = C$ and $b_1 = C$ respectively. Now the child g is (C).

Next, pick up towns from the parents alternately. Begin with a_3 (town A) because $x \leftarrow 4 - 1 = 3$, and next is b_2 (town D) because $y \leftarrow 1 + 1 = 2$. The child becomes $g = (A,C,D)$.

In the same way, add a_2 (town B), b_3 (town G), a_1 (town H), and the child becomes $g = (H,B,A,C,D,G)$. Now the next town is $b_4 = H$ and town H has already appeared in the child (remember the salesperson may not visit the same town twice), so we can't add any more towns from parent g_b.

Therefore we add towns from parent g_a. The next town is $a_0 = D$, but D is already used. Thus we can't add towns from parent g_a, either.

Then, we add the rest of the towns, i.e., E and F, to the child in the random order. Finally the child is $g = (H,B,A,C,D,G,F,E)$.

2.3 Natural Selection

Eliminate $R = M \times p_e/100$ individuals. In the so-called simple GA, the possibility of survival is proportional to the fitness value, but we pay more attention to the diversity of the population. We eliminate similar individuals to maintain the diversity in order to avoid the immature convergence that is one of the well-known problems in GA.

First, sort the individuals in fitness-value order. Compare the fitness value of adjoining individuals. If the difference is less than ε (a small positive real number), eliminate preceding individual while the number of eliminated individuals is less than R. Let r be the number of eliminated individuals.

Next, if $r < R$, eliminate $R - r$ individuals in the order of lowest fitness value.

3 JAVA applet

3.1 Applet on WWW

We developed a JAVA applet based on the proposed method, and put it on our WWW page:

```
http://www.hitachi.co.jp
/Div/sdl/e-naiyo/e-seika22/TSP.html
```

It is easy to use. First, using a mouse, put "towns" in the rectangular field. Then, push the "start" button. The applet solves the problem you've just made and displays the tour and its length (Fig. 4). The parameters of the 2optGA method, i.e., "Population" (M), "Selection" (p_e), "2opt" (p_i), are adjustable by filling out each field on the applet.

We announced the web page on the Net News in Japanese (fj.* newsgroups) only once in July 1996. Since then, the page has been accessed more than 200 times a month.

One person who sent us comments had mistakenly thought that GA was a tool only for experiments, and that GA would be so slow that it could not be applied to practical problems. But after accessing our web pages, he could understand why GA is usable.

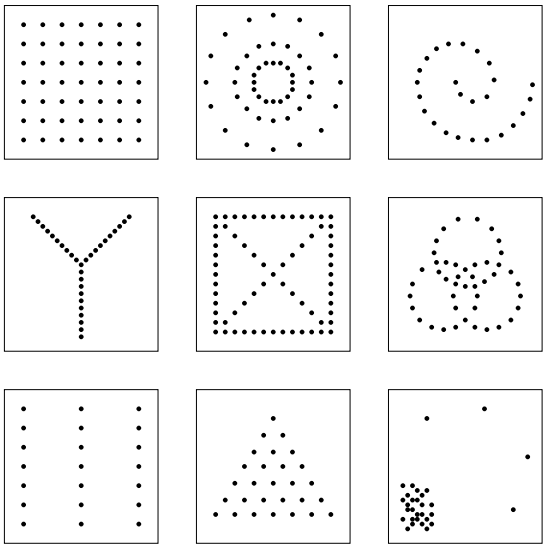
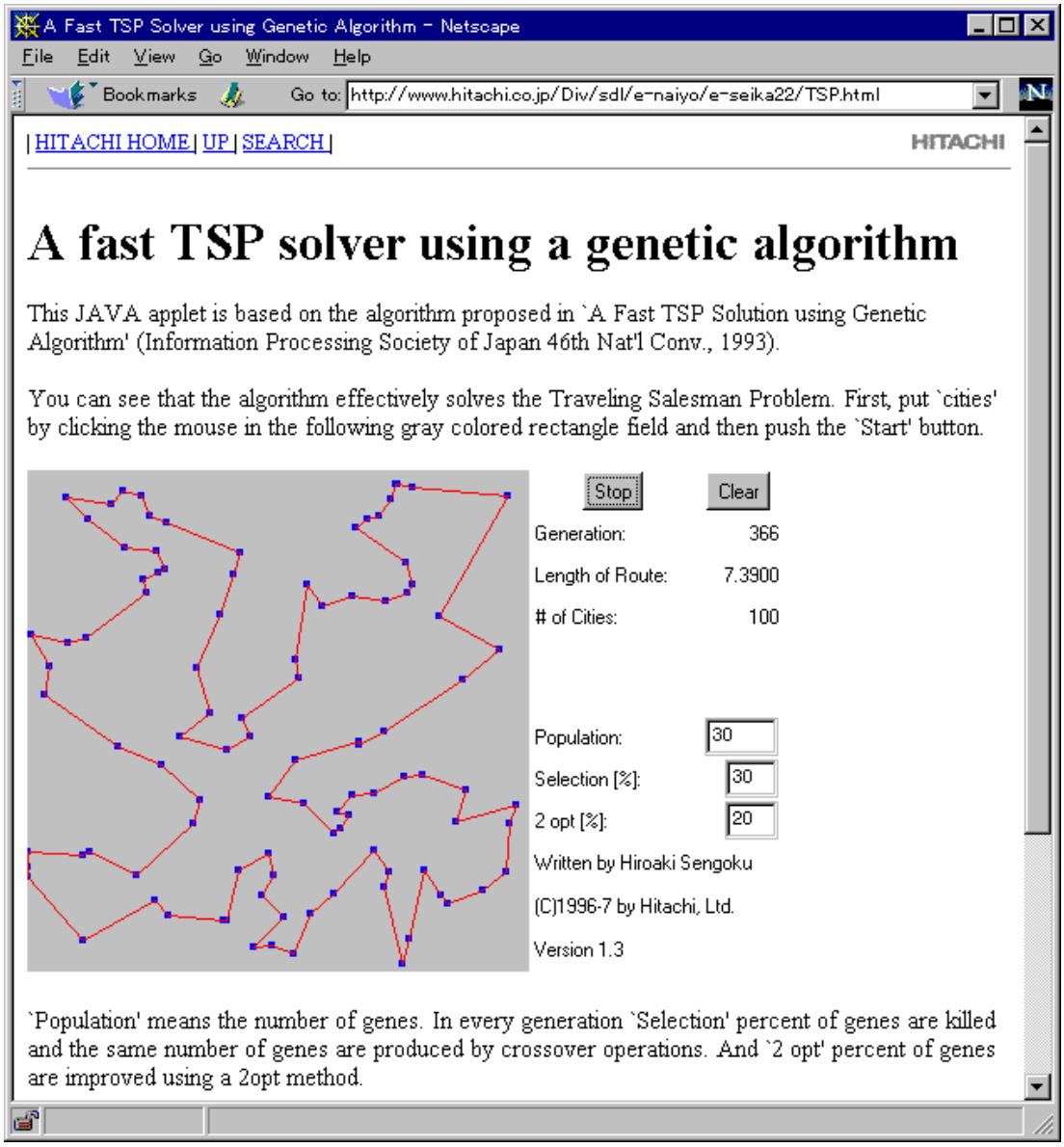


Figure 5: Examples of the problems the students made.

We utilized the web page for education. We introduced the page to the students majoring in computer science at the University of Tokyo and to the audience at a Hitachi employee seminar. They enjoyed mak-



Netscape is a trademark of Netscape Communications Corporation.

Figure 4: applet on WWW

ing new problems one after another and executing the applet. Some of those problems are shown in Fig. 5.

Some students added and removed a few towns in their problems and watched the transformation of the solution. Some tried to alter the parameters of the 2optGA and observe what would happen. Through many trials, they got the feeling of the difficulty of the TSP, and gained a concrete understanding of the optimization problems.

A summary of comments from the students follows:

- Good Points
  - It's easy and fun.
  - I can see the progress of solving the problem simultaneously.
  - I had thought that web pages were only for reading, so I was surprised to see your page for solving the TSP.

- Bad Points
  - No grid. The coordinates of the mouse are not displayed, so I cannot put towns accurately into the places I want.
  - I can't try pre-defined problems. I can't save problems. (Note: the *current* version of the applet *can* read the TSP file that contains the coordinates of towns.)
  - I want to see the convergence graph. I wish to check not only the elite but also the other individuals in the population. It would be convenient if the CPU time consumed to solve the problem was displayed.

We also present the following sample problems in our WWW pages:

- randomly located 100 towns
- double circle 192 towns ("C"-type)
- double circle 192 towns ("O"-type)

Double circle problems were used as benchmarks by Yamamura et al.[6] The minimum solutions of the problems are known and they are either "C"-type or "O"-type.

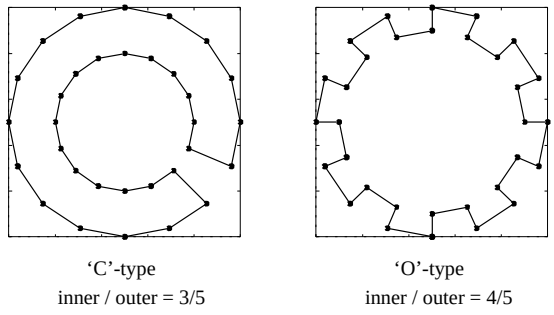


Figure 6: the minimum solutions of the double circle problems

As shown in Fig. 6, the type of solution that will yield the minimum, depends on the number of towns and the ratio of the radii of the inner and outer circles.

3.2 Stand-alone Application

The applet can also be used as a stand-alone application. First, download class files listed in Table 1 and sample problems listed in Table 2 from the URL:

```
http://www.hitachi.co.jp
/Div/sdl/e-naiyo/e-seika22/
```

Table 1: Class files to download.

File name	Size
Cities.class	3k
City.class	1k
GA.class	3k
Gene.class	3k
Sort.class	1k
Sortable.class	1k
TSP.class	8k
TspRoute.class	4k

Table 2: Sample problems.

File name	Problem
gr96.data	Africa-Subproblem of 666-city TSP
gr202.data	Europe-Subproblem of 666-city TSP
test100.dat	Randomly located 100 towns
c192-4682.dat	Double circle 192 towns ("C"-type)
c192-4684.dat	Double circle 192 towns ("O"-type)

"Africa-Subproblem of 666-city TSP" and "Europe-Subproblem of 666-city TSP" are contained in TSPLIB[5].

"TSP.class" contains the *main* method, so execute that file as a JAVA application with arguments 'filename' and 'generation'. The first argument, 'filename', is the name of the TSP file that contains the location of towns of the TSP to be solved. The second arguments, 'generation', is the maximum generation to be evolved. Optional flags listed in Table 3 can be used.

Each line in the TSP file represents the coordinates of a town and consists of two fields separated by a comma. The first field means the X coordinate of the town and the second field means the Y coordinate. The cost between two towns is measured as the Euclidean distance between the coordinates of each town.

If the TSP file contains the line

```
EDGE_WEIGHT_TYPE: GEO
```

the cost is defined by the geographical distance of two towns, i.e., the distance on the earth, instead of

Table 3: Optional flags of TSP.class

Option	Meaning	Default
-p M	The population	100
-e p_e	Eliminate p_e% individuals	30
-R p_i	Improve p_i% individuals by 2opt	20
-v	Increment verbosity level	

the Euclidean distance. The X and Y coordinates described above mean the latitude and the longitude respectively.

If the TSP file contains the line

`Min:` n

where n is the positive real number, the program will stop when the elite, whose tour length is n, is found.

On Windows[2]95 with the Java Development Kit (JDK)[7], for example, type the following command in the "MS-DOS[2] Prompt" window.

```
java TSP -v -p 200 gr96.data 300
```

That command means to solve the TSP stored in the file 'gr96.data' (Africa-Subproblem of 666-city TSP in TSPLIB[5]), where the population is 200, and the maximum generation is 300.

The output from the command follows (partially omitted):

```
C:\>java TSP -v -p 200 gr96.data 300
File:  gr96.data
Population:  200
Selection:  30 %
2 opt:  20 %
1:3[93 94 92 91 76 ..... 27 64 95]57284
2:253[94 93 95 64 65 ..... 76 91 92]55840
3:119[37 34 33 32 10 ..... 31 35 36]55481
19:1251[49 51 52 54 50 ..... 47 46 45]55332
44:2646[93 94 92 91 76 ..... 65 64 95]55322
120:7132[41 40 39 38 78 ..... 49 48 42]55291
165:9780[49 45 46 47 53 ..... 54 52 51]55259
180:10692[52 51 49 45 46 ..... 48 50 54]55209
```

The program outputs the elite, when a new elite is found, in the following format:

g : i [*tour*] ℓ

where g, i, *tour*, and ℓ mean the generation, the ID of the individual, the tour (a list of towns in the visiting order) that corresponds to the individual, and the length of the tour.

The last line, for example, says that a new elite is found at 180th generation, it is the 10692nd individual, and the length of the tour is 55209. According to TSPLIB, it is the minimum solution.

4 Conclusion

We presented an algorithm for rapid solution of the TSP that combines the GA and the 2opt method. We published a program based on the algorithm on our web pages. Because the program is written in JAVA language and can be executed on many platforms, anyone can verify the efficiency of our algorithm. Anyone who designed a new algorithm can compare his own TSP solver with ours by running both programs on the same machine. Our TSP solver is useful as a criterion for evaluating the performance of TSP solvers.

References

[1] Holland, J.H.: *Adaptation in Natural and Artificial Systems,* Univ. of Michigan Press (1975)

[2] Sengoku, H., Yoshihara, I.: *A Fast TSP Solution using Genetic Algorithm (Japanese),* Information Processing Society of Japan 46th Nat'l Conv. (1993)

[3] Sengoku, H., Yoshihara, I.: *An Evaluation of Optimizing Capability of Genetic Algorithm — GA vs SA — (Japanese),* Information Processing Society of Japan 47th Nat'l Conv. (1993)

[4] Moscato, P.: *TSPBIB,* http://www.ing.unlp.edu.ar/cetad/mos/TSPBIB_home.html

[5] Reinelt, G.: *TSPLIB,* ftp://softlib.rice.edu/pub/tsplib/tsplib.tar

[6] Yamamura, M., Ono, T., Kobayashi, S.: *Character-Preserving Genetic Algorithms for Traveling Salesman Problem (Japanese),* Journal of Japanese Society for Artificial Intelligence Vol.7, No.6 (1992)

[7] Cornell, G., Horstmann, C.S.: *core JAVA,* SunSoft Press (1996)

[2]MS-DOS and Windows are registered trademarks of Microsoft Corporation