A Hybrid Genetic Algorithm for the Hexagonal Tortoise Problem

Heemahn Choe, Sung-Soon Choi, and Byung-Ro Moon

School of Computer Science and Engineering,
Seoul National University,
Seoul, 141-742 Korea
{hchoe,irranum,moon}@soar.snu.ac.kr

Abstract. We propose a hybrid genetic algorithm for the hexagonal tortoise problem. We combined the genetic algorithm with an efficient local heuristic and aging mechanism. Another search heuristic which focuses on the space around existing solutions is also incorporated into the genetic algorithm. With the proposed algorithm, we could find the optimal solutions of up to a fairly large problem.

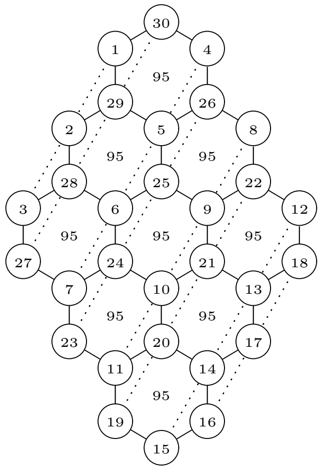
1 Introduction

Hexagonal tortoise problem (HTP), also known as Jisuguimundo, is a numerical puzzle which was invented by medieval Korean scholar and minister Suk-Jung Choi (1646-1715) [1]. The problem is to assign the consecutive numbers 1 through n to the vertices in a graph, which is composed of hexagons like beehives, to make the sum of the numbers of each hexagon the same. Figure 1 shows two example hexagonal tortoises. Choi showed a 30-node hexagonal tortoise in his book along with other various kinds of magic squares.

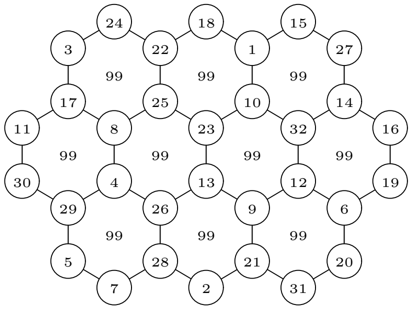
The HTP is somewhat similar to the magic squares. As for the magic squares, there is no known method that generates solutions for an arbitrary HTP, either. A special kind of HTPs, diamond HTPs, have a particular solution using their symmetry [2]. Figure 1(a) shows the particular solution of a 30-node HTP.

We consider a generalized problem to minimize the variance of hexagonal sums. In this paper, we propose an efficient local optimization heuristic for the HTP using problem-specific knowledge. We also combined a hybrid genetic algorithm with aging mechanism and another search heuristic which focuses on the space around existing solutions.

The rest of this paper is organized as follows. In Section 2, we describe some properties of the hexagonal tortoise problem. In Section 3, we describe our local optimization heuristic. In Section 4, we describe the proposed hybrid genetic algorithm in detail. In Section 5, we show the experimental results. Finally, the conclusion is given in Section 6.



(a) A 30-node 3 × 3 diamond hexagonal tortoise using the known particular solution. Dotted lines show numbering sequence.



(b) A 32-node hexagonal tortoise

Fig. 1. Example hexagonal tortoises

2 Hexagonal Tortoise Problem

Hexagonal tortoise problem is to assign a sequence of numbers to vertices in a graph with a fixed topology. The graph is composed of a number of overlapping hexagons. The numbers are typically consecutive integers from 1 to n, where n is the number of vertices in the graph. The term "tortoise" comes from the fact that the overall shape of the graph resembles the theca (or shell) of a turtle. Each number has to be assigned to a vertex exactly once. A hexagonal sum is defined to be the sum of the six numbers of a hexagon. The objective is to find an assignment of the numbers to the vertices so that the standard deviation (or variation) of the hexagonal sums is zero. In this paper, we consider a special type of hexagonal tortoise, a $k \times k$ diamond hexagonal tortoise problem, which arrangement of hexagons is like diamond shape [3]. Figure 1(a) shows a solution of 3 × 3 diamond hexagonal tortoise.

A typical HTP has many solutions with different hexagonal sums, which means that the fitness landscape of the problem is multimodal. For example, in the 16-node HTP, we found 140,915 different optimum solutions among 2^{32} random permutations; from this, we presume there are more than 6×10^8 optimum solutions among total 16! possible assignments for the 16-node HTP. The difficulty of multimodal space search with respect to genetic algorithms was pointed out in the literature [4] [5] [6].

To get more insight into the fitness landscape of the problem, we examined the fitness-distance correlation (FDC) [7], which is a popular measure for predicting the difficulty of a problem. We compared it with that of the traveling

Table 1. FDC values of the HTP and the TSP instances

(a) HTP instances

Instance	FDC value
HTP30	0.1197
HTP48	0.1748
HTP70	0.0877
HTP96	0.0816
HTP126	0.0777

(b) TSP instances

Instance	FDC value
lin318	0.6853
pcb442	0.4999
att532	0.5617
rat783	0.6692
dsj1000	0.2187

```
do {
    changed ← False;
    for ($l_0 \leftarrow 0$; $l_0 < n-1$; $l_0$++)
        for ($l_1 \leftarrow l_0 + 1$; $l_1 < n$; $l_1$++)
            if (gain($l_0$, $l_1$) > 0) {
                exchange the gene at $l_0$ with the gene at $l_1$;
                changed ← True;
            }
} while (changed);
```

Fig. 2. A simple 2-Opt exchange

salesman problem (TSP). Table 1 compares the FDC values of the HTP and TSP for a number of instances. In the table, "HTPn" represents the n-node HTP. The table shows that the HTPs with just dozens of nodes have far more rugged landscapes than the TSPs with hundreds of nodes. For example, the HTP with 30 nodes shows a lower FDC value than that of the TSP with 1,000 nodes. Although FDC is not a definite measure for the problem difficulty, this observation says at least the extreme ruggedness of HTP.

3 Local Optimization Heuristic

3.1 Consecutive Exchange

A simple 2-Opt heuristic can be used as a local optimization method for most combinatorial optimization problems [8]. Figure 2 shows the outline of the 2-Opt heuristic for the HTP. It examines all possible $\binom{n}{2}$ pairs of genes. When it finds a pair with positive *gain*, it exchanges them greedily. The gain is defined as gain(i, j) = (fitness after exchanging the genes at i and j) - (fitness before the exchange).

We used a variant of 2-Opt in the following reasons. When we investigated the gene pairs whose exchanges led to fitness improvements, we found that the gene pairs with small gene-value differences were more probable. Figure 3 shows the frequencies of the gene-value differences for the pairs that led to fitness

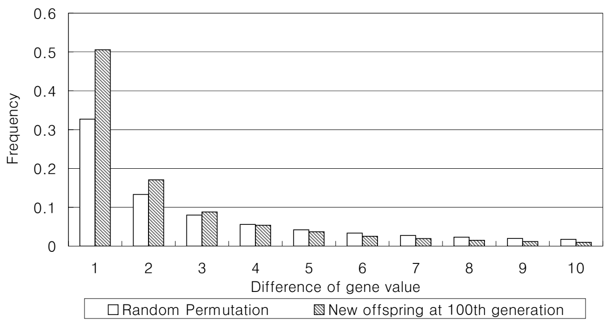


Fig. 3. Frequencies of the gene value of the pairs differences that led to fitness improvements (HTP160)

```
do {
    $changed$ ← False;
    for ($v$ ← 1; $v < n$; $v$++) {
        $l_0$ ← locationOf($v$);
        $l_1$ ← locationOf($v + 1$);
        if (gain($l_0$, $l_1$) > 0) {
            exchange the gene at $l_0$ with the gene at $l_1$;
            $changed$ ← True;
        }
    }
} while ($changed$);
```

Fig. 4. The outline of consecutive exchange

improvement in a typical run of our genetic algorithm for the 160-node HTP. For initial chromosomes, the number of the pairs whose difference in value is only one was more than 30% of the total number of pairs. And the number grew further to more than 50% at the 100^{th} generation. This means that the 2-Opt heuristic would waste most time in vainly examining changes of big differences. This is more serious when high-quality solutions are locally optimized.

We designed a local optimization heuristic which examines only the gene pairs whose values are differed by one. We call it *consecutive exchange*. Figure 4 shows the outline of the consecutive exchange. For the problem of size n, while the time complexity of the inner loop of the original 2-Opt exchange in Figure 2 is $\Theta(n^2)$, that of the consecutive exchange is $\Theta(n)$. Comparisons of performances of the 2-Opt and consecutive exchanges are presented in Table 2 which are described in the next section.

```
$T$ = { original solution };
do {
    $changed \leftarrow$ False;
    for each pair $l_0$, $l_1$ by the exchanging policy
        if (gain($l_0$, $l_1$) $> 0$) {
            exchange the gene at $l_0$ with the gene at $l_1$;
            $T \leftarrow$ { current solution };
            $changed \leftarrow$ True;
        } else if (gain($l_0$, $l_1$) $= 0 \wedge$ current solution $\notin T$) {
            exchange the gene at $l_0$ with the gene at $l_1$;
            $T \leftarrow T \cup$ { current solution };
            $changed \leftarrow$ True;
        }
} while ($changed$);
```

Fig. 5. Local optimization with tabu search

3.2 Further Optimization Using Tabu Search

In the process of local optimization, there are gene pairs whose exchanges do not affect fitness. Exchanging them may help improvement in other gene positions since it changes the distribution of the hexagonal sums. Usually, there are a great number of equi-fitness solutions near a solution. We adopted tabu search to search these solutions efficiently; we kept a list of the visited equi-fitness solutions and prevented revisiting them.

Tabu search is a meta-heuristic for optimization problems invented by Glover [9]. The tabu search keeps one or more lists of recently visited solutions to prevent the search from revisiting the visited solutions before.

Figure 5 shows the outline of our local optimization incorporated with tabu search. The performances of the 2-Opt heuristic and the consecutive exchange with and without tabu search are summarized at Table 2. We tested them with a traditional genetic algorithm of population size 1,000. We used the number of fitness calculation as the index of processor time. Table 2 shows the numbers of fitness calculations of the heuristics for the initial chromosomes and the chromosomes in 100^{th} generation, respectively. In both cases, the 2-Opt heuristic had slightly lower standard deviation of hexagonal sums (higher fitness) than that of the consecutive exchange. But, the 2-Opt was much slower than the consecutive exchange, especially with tabu search. Table 2 indicates that the consecutive exchanges is more effective in later generations.

4 The Hybrid Genetic Algorithm for the Hexagonal Tortoise Problem

A notable feature of our hybrid genetic algorithm is that it is combined with another search heuristic to search the space around the solutions in the current

Table 2. Performances of the local heuristics. Problem size is 160. SD means standard deviation

(a) The initial chromosomes in a typical GA run

Version	Avg. SD of hex. sums	Avg. fitness gain	Avg. # of fitness calculation	Avg. gain per average calculation
2-Opt without tabu search	1.1648	12364.90	143127.4	0.086391
2-Opt with tabu search	0.7343	12365.75	831483.0	0.014872
Consecutive without tabu search	1.1748	12364.87	8018.8	1.541989
Consecutive with tabu search	0.7516	12365.72	8939.1	1.383337

(b) The chromosomes in the 100th generation in a typical GA run

Version	Avg. SD of hex. sums	Avg. fitness gain	Avg. # of fitness calculation	Avg. gain per average calculation
2-Opt without tabu search	0.9425	401.68	104929.3	0.003828
2-Opt with tabu search	0.6452	402.19	844419.2	0.000476
Consecutive without tabu search	0.9866	401.59	3073.6	0.130660
Consecutive with tabu search	0.6840	402.13	3871.9	0.103858

population. We call it *nearby search*. This scheme can be regarded as a three level optimization; the GA is used for large-scale search, the nearby search is used for medium-scale search, and the local optimization is used for small-scale search.

The overall structure of the proposed hybrid genetic algorithm is described in Figure 6. In the main loop, offsprings are generated and local-optimized as in a typical hybrid GA. Then, the nearby search is applied to all the chromosomes in the population.

The details of the algorithm are as follows.

- Problem Encoding: Each locus is numbered top to bottom in zigzag order. Figure 7 shows indices of vertices in the 30-node HTP. The zigzag order was chosen to reflect more geographic localities of genes in the chromosome.
- Fitness and Selection: The fitness of a chromosome is defined as

$$\text{Fitness} = -\sigma^2$$

  where σ^2 is the variance of hexagonal sums. Note that the fitness values are negative. Selection is done based on the rank of the *effective fitness* that is slightly modified from the above fitness. The effective fitness is described in Section 4.2.
- Population: We set the population size to be 512.
- Local optimization: Local optimization used in our algorithm was explained in Section 3.

```
create initial population $\{c_0, c_1, \ldots, c_{N-1}\}$;
for ($i \leftarrow 0$; $i < N$; $i$++)
    $c_i \leftarrow$ localOptimize($c_i$);

while (stop condition is not satisfied) {
    for ($i \leftarrow 0$; $i < \frac{N}{2}$; $i$++) {
        choose two parents $p_1$, $p_2$;
        $o_i \leftarrow$ crossover($p_1$, $p_2$);
        $o_i \leftarrow$ mutation($o_i$);
        $o_i \leftarrow$ localOptimize($o_i$);
    }
    replace $\frac{N}{2}$ chromosomes in population with $o_0, o_1, \ldots, o_{\frac{N}{2}-1}$;

    for ($i \leftarrow 0$; $i < N$; $i$++) {
        $n_i \leftarrow$ modification($c_i$);
        $n_i \leftarrow$ localOptimize($n_i$);
        if (fitness($n_i$) $\geq$ fitness($c_i$)) $c_i \leftarrow n_i$;
    }
}
return the best solution;
```

Fig. 6. Overall structure of proposed hybrid GA

– Replacement: At each generation, the worst half of the population is replaced.
– Crossover: Two-point crossover is used.
– Mutation: Each gene value is increased by one with probability $\frac{1}{3}$ or decreased by one with probability $\frac{1}{3}$.
– Modification: This is used in the nearby search and described in Section 4.1.
– Repair: Since crossover or mutation can generate infeasible solutions (i.e., one number is assigned to more than one node), repair is needed. In repair process, gene values are replaced with their sorted orders. If there are genes with the same value, they are randomly reordered. Figure 8 shows an example of the process.
– Stop Condition: The algorithm stops when an optimum is found or the number of generations reaches a threshold.

4.1 Nearby Search

Since the HTPs have significantly rugged landscapes as mentioned before, traditional GAs do not solve well the HTPs with dozens or more nodes, even if they are combined with some local optimization heuristics. We adopted another search heuristic which focuses on the space around the solutions in the current population. This heuristic is similar to the idea of iterated local search [10]. But our approach is based on population and the search is applied to each chromosome once at each generation. From the viewpoint of high-quality solutions that

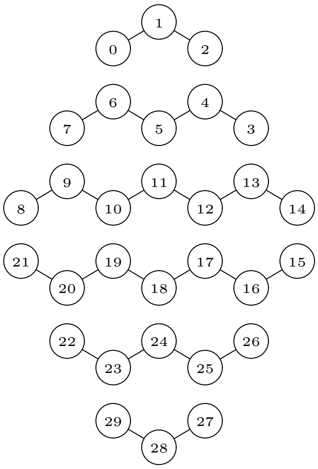


Fig. 7. Problem encoding

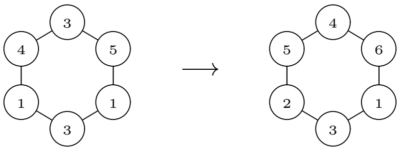


Fig. 8. An example repair process

survive over a number of generations, the heuristic is similar to iterated local search or large-step Markov chain [11].

Nearby search is applied to every chromosome in the population at the end of each generation. Each chromosome is modified and the local optimization heuristic is applied to it. If the fitness of the newly generated chromosome is not worse than the original one, the original chromosome is replaced by the new one.

We devised a guided modification which uses problem-specific knowledge. We define the *error* for a gene k as follows:

$$\varepsilon(k) = \sum_{h \in H_k} [s(h) - \mu],$$

where H_k is the set of hexagons that include k, $s(h)$ is the hexagonal sum of hexagon h, and μ is the mean of hexagonal sums.

The modification decreases the value of the gene whose error is the largest to positive and increases the value of the gene whose error is the largest to negative. It compulsorily alleviates the worst defect caused by those genes and lets the local optimization heuristic to cope with the change in hexagonal sums afterward.

Since the perturbation of the applied modification is weak, many of the modified solutions return to the original ones by the local optimization heuristic. In our algorithm, we look up the tabu list of the original solutions as well as that of the new ones so that the new chromosomes are different from the original ones.

Experimental results for the nearby search are presented in Section 4.3.

4.2 Aging

Since the HTPs have very rugged landscapes, most of the population is often occupied by local optima that contribute little to find the global optima. We adopted a kind of aging mechanism [12] [13] in which the fitness of a chromosome decreases over the generations of the genetic algorithm. The age of a newly generated solution is set to zero. At each generation, the age increases by one. The effective fitness considering the aging is defined as follows:

$$\text{Effective Fitness} = \text{Fitness} - \frac{1}{10H} \cdot \text{Age}$$

where H is the number of hexagons in a graph. The coefficient $\frac{1}{10H}$ is chosen by the following reasons. Since low-quality solutions are easily replaced by better solutions even without aging, we focused on high-quality solutions. High-quality solutions that can be found in a mature population usually have the property that most of its hexagonal sums have the same value and only a few of the sums are larger or smaller by one. For a chromosome having i hexagons with hexagonal sum $S+1$, j hexagons with hexagonal sum $S-1$ and $H-i-j$ hexagons with hexagonal sum S, the variance of its hexagonal sums is

$$\sigma^2 = \frac{i+j}{H} - \frac{(i-j)^2}{H^2}.$$

For small i and j, $\sigma^2 \approx \frac{i+j}{H}$. So the variances of hexagonal sums of high-quality solutions are discrete and the interval between them is about $\frac{1}{H}$. (The property holds even for the hexagons with hexagonal sum $S\pm2$.) Since the fitness is defined to be $-\sigma^2$, we chose the coefficient $\frac{1}{10H}$ so that the fitness of a solution decreases at intervals of $\frac{1}{H}$ every ten generations. Thus, if the fitness of a chromosome does not improve by the nearby search, it is eventually replaced by a new chromosome.

4.3 Performance Improvement by Nearby Search and Aging

Table 3 shows the effects of nearby search and aging. The 96-node HTP was used for the test. We set four versions of GA depending on the uses of nearby search and aging. All the versions except version I, which is a standard hybrid GA, found optimal solutions on every run.

When we compare version I and Version II, we can see the usefulness of nearby search. Version II was significantly better and faster than Version I. Version III was also similarly better and faster than Version I. Version IV, which combines nearby search and aging, showed further improvement over version II and III.

Table 3. The effect of the nearby search and aging for the 96-node HTP. The maximal number of generation was set to be 10,000. SD and CV means standard deviation and coefficient of variation, respectively.

Version	Nearby Search	Aging	SD of hex. sums		CPU time (seconds)	
			Best	Avg.	Avg.	CV(%)
I	N	N	0.0000	0.1524	2143.69	18.76
II	Y	N	0.0000	0.0000	170.73	138.53
III	N	Y	0.0000	0.0000	257.96	55.16
IV	Y	Y	0.0000	0.0000	73.67	50.48

Table 4. Experimental results for various instances. Generation 0 means that the optimum is found in initial population. SD and CV mean standard deviation and coefficient of variation, respectively.

Size	SD of hex. sums			Generations		CPU time (seconds)		Time per generation
	Best	Avg.	SD	Avg.	CV(%)	Avg.	CV(%)	
30	0.0000	0.0000	0.0000	0.00	(SD=0.00)	0.12	17.70	‡
48	0.0000	0.0000	0.0000	0.59	107.99	0.35	24.98	‡
70	0.0000	0.0000	0.0000	14.90	80.82	3.61	64.82	0.28990
96	0.0000	0.0000	0.0000	253.41	57.31	79.40	55.01	0.32395
126	0.0000	0.0000	0.0000	1424.19	49.19	679.13	47.71	0.48207
160	0.0000	0.0047	0.0239	17480.10	68.89	13782.66	67.35	0.75851

‡ We could not compute reliable values for these problem sizes since the numbers of generations are too small for these problems.

5 Experimental Results

Our official version is the GA with the nearby search and aging mechanism. The algorithm was implemented in C and run on Pentium4 1.5 GHz. Table 4 shows the experimental results of our algorithm for various instances. The result of the 160-node HTP is from 50 runs and those of the rest are from 100 runs. We found the optimal solutions for up to the 160-node HTP in reasonable time. Also, we could find some optimal solutions of the 198-node HTP in a few days. The average time to find the optimal solutions increases exponentially with respect to the problem size. For the problem of sizes 30, 48, 70, 96, and 126, the algorithm found optimal solutions on every run. Figure 9 shows one of the solutions that we found for the 198-node HTP.

6 Conclusion

We proposed an efficient local optimization heuristic for the hexagonal tortoise problem using some problem-specific knowledge. The hybrid GA was further improved by the addition of nearby search. The proposed GA can be regarded as a three level optimization. Another notable aspect of our algorithm is the

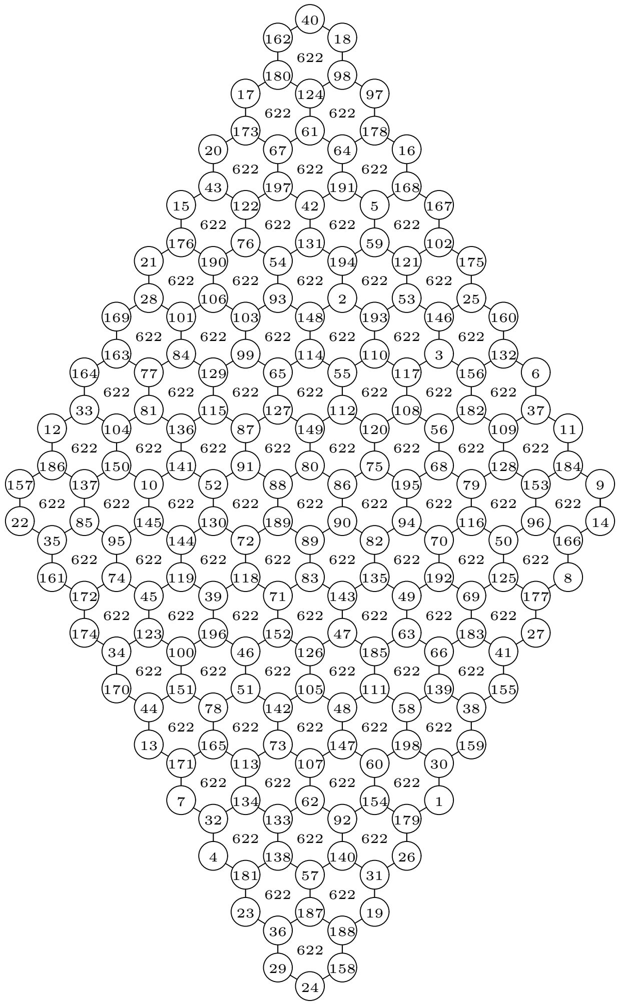


Fig. 9. An optimal solution of the 198-node hexagonal tortoise

aging mechanism. It helps the GA to weed out local optima that contribute little to find global optima.

The proposed GA found optimal solutions for up to the 198-node hexagonal tortoise problem. This seems to be a fairly good achievement considering the extreme ruggedness of the problem. Future studies may include investigating larger size of HTPs, probably with parallel computing, applying the developed ideas to other combinatorial optimization problems, and further refining the GA.

Acknowledgements. This work was partly supported by Optus Inc. and Brain Korea 21 Project. The RIACT at Seoul National University provided research facilities for this study.

References

1. S. J. Choi. *Gusuryak (a reprint).* Shungshin Womens University Press, Seoul, Korea, 1983.
2. Y. H. Jun. Mysteries of mathematics: The order of the universe hidden in numbers. *Dong-A Science*, 14(7):68–77, 1999.
3. S. K. Lee, D. I. Seo, and B. R. Moon. A hybrid genetic algorithm for optimal hexagonal tortoise problem. In *Genetic and Evolutionary Computation Conference*, page 689, 2002.
4. D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley, 1989.
5. S. Forrest and M. Mitchell. Relative building-block fitness and the building-block hypothesis. In *Foundations of Genetic Algorithms*, volume 2, pages 109–126. Morgan Kaufmann, 1993.
6. J. Horn and D. E. Goldberg. Genetic algorithm difficulty and the modality of fitness landscapes. In *Foundations of Genetic Algorithms*, volume 3, pages 243–270. Morgan Kaufmann, 1995.
7. T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Sixth International Conference on Genetic Algorithms*, pages 184–192. Morgan Kaufmann, 1995.
8. E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization.* John Wiley & Sons, 1997.
9. F. Glover. Tabu search: Part I,. *ORSA Journal of Computing*, 3(1):190–206, 1977.
10. H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In F. W. Glover and G. Kochenberger, editors, *Handbook of Metaheuristic*, chapter 11. Kluwer Academic Publisher, 2002.
11. O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5:299–326, 1991.
12. Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolutionary Programs.* Springer, 1992.
13. A. Ghosh, S. Tsutsui, and T. Tanaka. Function optimization in nonstationary environment using steady state genetic algorithms with aging of individuals. In *IEEE International Conference on Evolutionary Computation*, pages 666–671, 1998.