A Graph-Based Lamarckian–Baldwinian Hybrid for the Sorting Network Problem

Sung-Soon Choi and Byung-Ro Moon

***Abstract*—A hybrid genetic algorithm (GA) is proposed for the optimal sorting network problem. Based on a graph-theoretical viewpoint, we devised a solution repair heuristic which incorporates a strong local optimization. We also propose a new encoding scheme which combines the characteristics of Lamarckian and Baldwinian GAs. Using a single-CPU PC, we obtained results comparable to previous results obtained with supercomputers.**

***Index Terms*—Graph-based approach, hybrid genetic algorithm (GA), Lamarckian–Baldwinian hybrid encoding, sorting network.**

I. Introduction

A SORTING network is a hardware sorting logic in which the comparisons and exchanges of data are carried out in a prescribed order. A sorting network for n inputs is called an n-bus sorting network. A sorting network is composed of a number of homogeneous comparators. We denote a comparator by $\mathrm{c}(a, b)$. In a comparator $\mathrm{c}(a, b)$, the ath bus and bth bus are compared; if their values are in order, no change occurs, otherwise, they are exchanged. Fig. 1 shows a simple sorting network [c(0,1), c(2,3), c(0,2), c(1,3), c(1,2)]. This is a four-bus sorting network with five comparators. The four inputs in the buses 0, 1, 2, and 3 are sorted after passing the five comparators. The intermediate values on the buses represent the values on the buses after passing the comparators; each vertical segment represents a comparator that connects two buses.

We can usually run a number of comparators simultaneously. In Fig. 1, the first and second comparators can run simultaneously since they are independent. Similarly, the third and fourth comparators can also run simultaneously. Thus, the sorting can be completed in just three parallel steps.

The research of sorting networks has two main aims [18]. One is to reduce the number of comparators and the other is to reduce the number of parallel steps in order to minimize run time. Both of these aims have been considered as hard ones for the last 40 years.[1] Among the problems in the former class, the 16-bus one have attracted considerable attention. Many scientists attacked the problem [1], [3], [10], [12], [23], and, in 1969, Green [12] first discovered a 60-comparator sorting network, which is still one of the best known.

Manuscript received March 14, 2003; revised May 5, 2004. This work was supported in part by the Brain Korea 21 Project.

The authors are with the School of Computer Science and Engineering, Seoul National University, Seoul 151-742 Korea (e-mail: sschoi@soar.snu.ac.kr; moon@cs.cnu.ac.kr).

Digital Object Identifier 10.1109/TEVC.2004.841682

[1]The optimal values are known only up to the eight-bus case in the former and up to the ten-bus case in the latter. See [18] for details.

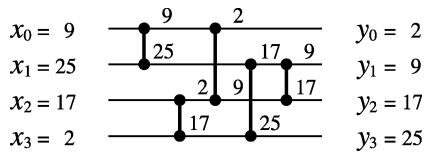

Fig. 1. Four-bus sorting network ($y_0 \leq y_1 \leq y_2 \leq y_3$).

Recently, 16-bus sorting networks have attracted attention again due to the improvements in new stochastic search methods [2], [8], [9], [13], [16], [19], [21], [22]. In 1992, Hillis [13] tackled this problem with a genetic algorithm (GA) with coevolution and found 61-comparator sorting networks. Although he failed to match the quality of the best-known sorting network, his record was encouraging as the first trial with a GA. It is notable that Hillis' research on sorting network was also a highly industry oriented one. He used the sorting network problem as a promising application of the connection machine. Since then, the problem has been implicitly used as a good testbed for effective algorithms and supercomputers [8], [16], and, in 1995, Juillé [16] found 60-comparator sorting networks with a novel stochastic search method.

All these studies used supercomputers due to the huge running time. In this paper, we present a novel approach based on our theoretical view of the problem. The proposed approach has the characteristics of both Lamarckian and Baldwinian GAs. Lamarckian GAs overwrite the chromosome after local optimization. Baldwinian GAs just use the results of local optimization for fitness evaluation. The proposed GA combines both characteristics by representing part of the whole solution in a chromosome. We used a single-CPU Pentium PC and found 60-comparator sorting networks in a few minutes. This was possible by devising effective local optimization heuristics and combining them with genetic operators.

The rest of this paper is organized as follows. In Section II, we provide some preliminary information for the problem. In Section III, we present our theoretical view of the sorting network problem. In Section IV, we describe the proposed local optimization heuristics and in Section V, we provide a genetic framework combined with the local optimization heuristic. We present the experimental results in Section VI. Finally, we list our conclusions in Section VII.

II. Preliminaries

To check the validity of an n-bus network, we have to confirm that the network correctly sorts all the possible input sequences. There are $n!$ sequences and it is costly to check all of them for

large n. Fortunately, it is possible to restrict the inputs to binary sequences by the following proposition [18].

Proposition 1 (Zero-One Principle): If an n-bus sorting network sorts all 2^n sequences of 0's and 1's into nondecreasing order, it sorts any arbitrary sequence of n numbers into nondecreasing order.

The Zero-One Principle says that we can prove the validity of the network by testing just 2^n binary sequences instead of $n!$ sequences. Rabin [18] proved that the *validity* problem is co-NP-complete. This means that this problem is intrinsically difficult in terms of computation theory [11]. Juillé showed that the sorting network problem is a kind of deceptive problem [17] and that the density of the valid solutions in the problem space is very low [16].

There were several computational approaches for the sorting network problem [2], [8], [9], [13], [16], [19], [21], [22] and two of them, described in the following, are particularly remarkable.

In [13], Hillis proposed a GA with a novel feature called coevolution to attack the problem. In his GA, there exist two independent populations: "hosts" (sorting networks) and "parasites" (binary sequences to be tested). The hosts and parasites get improved due to their complementary coevolution. The GA used a huge (of size 65 536) population on a CM-1 supercomputer.

In [16], Juillé devised a stochastic search method called *evolving nondeterminism* (END) which restricts the search to as valid solutions as possible. In the END model, the problem space is represented by a tree. The leaves of the tree represent valid networks and the internal nodes represent invalid networks. That is, the internal nodes are intermediate steps necessary to reach the valid networks. An individual of the END model begins at the root of the tree and progresses level by level in the tree until a leaf is encountered. In the course, it estimates the children of the current node by doing a random sampling and selects a promising node among them. The END model simulates the evolution of a population by performing careful reproduction and elimination of such individuals. It does not allow any backtracking in the course of constructing networks, so a "misled" search cannot be recovered afterwards. It also required a large population (of size 4096) on a Maspar MP-2 supercomputer.

III. Graph-Theoretical View of the Problem

In this section, we present our graph-theoretical view of the sorting network problem.

A. Problem Definition

As mentioned before, the Zero-One Principle guarantees that we can prove the validity of a network by testing 2^n binary sequences. We denote by $\mathcal{T}_n$ the entire set of 2^n binary sequences with which we test and by $\mathcal{S}_n$ the set of sorted sequences from $\mathcal{T}_n (\mathcal{S}_n \subseteq \mathcal{T}_n)$. For example, $\mathcal{T}_4 = \{0000, 0001, 0010, \ldots, 1111\}$ and $\mathcal{S}_4 = \{0000, 0001, 0011, 0111, 1111\}$. We can consider an n-bus sorting network as a function from $\mathcal{T}_n$ to $\mathcal{T}_n$. Let f_χ be the function corresponding to a sorting network χ. If we denote by $|\chi|$ the number of comparators in the sorting network χ, then the n-bus sorting network problem is to find χ^* such that

$$|\chi^*| = \min \{|\chi| \,|\, f_\chi(\mathcal{T}_n) = \mathcal{S}_n\}. \tag{1}$$

Let ϕ be a sequence of comparators. The set of binary sequences not sorted after passing ϕ is $f_\phi(\mathcal{T}_n) - \mathcal{S}_n$. Then, in the case where the comparators of ϕ are fixed in the front part of the networks, the minimum sorting network problem is to find ψ^* such that

$$|\psi^*| = \min \{|\psi| \,|\, f_\psi(f_\phi(\mathcal{T}_n) - \mathcal{S}_n) \subseteq \mathcal{S}_n\}. \tag{2}$$

In the 16-bus problem, if we fix the sequence ϕ_{32} of the first 32 comparators of Green's network as in most studies for the problem [8], [13], [16], we only have to consider the sequences of $f_{\phi_{32}}(\mathcal{T}_{16}) - \mathcal{S}_{16}$ among all the 2^{16} binary sequences of $\mathcal{T}_{16}$. We can easily check that there are only 151 such sequences, so that the problem space is considerably reduced.

B. Graph Representation for Problem Spaces

For an element $t_i \in \mathcal{T}_n$, let s_i be the sequence produced by sorting $t_i (s_i \in \mathcal{S}_n)$. For example, in four-bus networks, if $t_i = 0100$, then $s_i = 0001$. Now, we consider a directed graph G_i, which has t_i as the starting vertex and s_i as the end vertex [see Fig. 2(a)]. Here, c(x, y) represents the comparator that connects input buses x and y. Each vertex represents the state of the binary sequence after sorting t_i by the comparators indicated by the edges from the starting vertex to the vertex. In Fig. 2, the minimum network for sorting t_i is the network composed of only c(1,3), which is the only comparator on the shortest path from the starting vertex to the end vertex.

We now consider $t_j = 1100$ and, consequently, $s_j = 0011$ in a similar way. Then, we have a graph G_j as in Fig. 2(b). In this case, there are four distinct minimum sorting networks with two comparators that sort t_j: [c(1,2), c(0,3)], [c(1,3), c(0,2)], [c(0,2), c(1,3)], and [c(0,3), c(1,2)]. As shown above, it is easy to find a minimum network that sorts an arbitrary sequence. In other words, if the test set were composed of only one sequence, the problem of finding a valid minimum sorting network would be just the shortest-path problem in the graph corresponding to that sequence.

However, it is not trivial when the test set has more than one sequence. For instance, it is not trivial to find a minimum network that sorts both t_i and t_j by individually looking at G_i and G_j in the above example. We combine the two graphs G_i and G_j as follows. Let I_a's and J_b's be the vertices of G_i and G_j, respectively. We first combine I_1 and J_1, the starting vertices of G_i and G_j, so that the combined vertex I_1J_1 becomes the starting vertex of the combined graph. Then, starting from I_1J_1, perform the following process. Given the vertex I_aJ_b combined from I_a and J_b, consider the comparators corresponding to the edges incident to I_a and J_b in G_i and G_j. Assume that I_a and J_b are transited to $I_{a'}$ and $J_{b'}$, respectively, by a comparator c.[2] If there exists the vertex $I_{a'}J_{b'}$ in the combined graph, we just put an edge from I_aJ_b to $I_{a'}J_{b'}$. Otherwise, we add $I_{a'}J_{b'}$ in the graph and put an edge marked by c from I_aJ_b to $I_{a'}J_{b'}$.

[2]It is possibly true that I_a is identical to $I_{a'}$ or J_b is identical to $J_{b'}$, but the two cannot hold simultaneously.

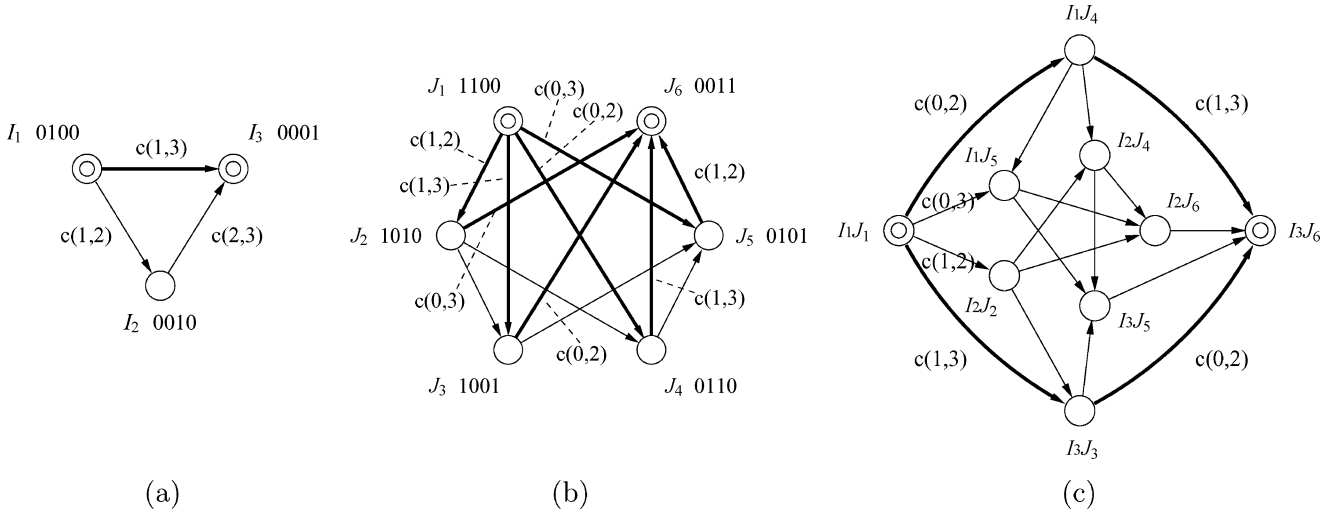

Fig. 2. Graph representations for problem spaces. (a) G_i for $t_i = 0100$. (b) G_j for $t_j = 1100$. (c) Combined graph $G_i \bigotimes G_j$.

Then, we repeat the above process until all vertices in G_i and G_j are considered. We denote by $G_i \bigotimes G_j$ the graph obtained after combining G_i and G_j. Fig. 2(c) shows $G_i \bigotimes G_j$.

Then, from $G_i \bigotimes G_j$, it is easy to find the minimum sorting network for the two test sequences. In this case, there are two distinct minimum networks with two comparators: [c(0,2), c(1,3)] and [c(1,3), c(0,2)]. In this way, we can represent the problem space of an n-bus sorting network as a graph. First, we construct a graph G_i for each $t_i \in \mathcal{T}_n$. Then, we combine all these graphs to obtain the graph representing the problem space of n-bus networks. In the case that a set ϕ of comparators is fixed, we only have to combine the graphs corresponding to the sequences in $f_\phi(\mathcal{T}_n) - \mathcal{S}_n$ instead of $\mathcal{T}_n$. That is, the problem space graph becomes

$$G = \bigotimes_{i=1}^{|f_\phi(\mathcal{T}_n)-\mathcal{S}_n|} G_i \tag{3}$$

which is a subgraph of the graph obtained from all the sequences in $\mathcal{T}_n$.

This graph is similar to the state-space tree of the END model, in that a path in the graph matches the network with the comparators corresponding to the edges on the paths. However, they are different in representing the states that are transited from a state by different-shaped networks with the same functionality. They are treated differently in the END model; but they are merged into a single vertex in our model.

In the proposed graph model, the problem of finding a minimum sorting network is equivalent to solving the shortest path problem from the starting vertex to the end vertex in G. When the input size is small, it is not difficult to construct such a graph G. However, the size of the graph sharply increases with the growth of input size. Even when we fix the set ϕ_{32} of the first 32 comparators in the 16-bus problem, construction of the problem space graph is infeasible. Fig. 3 shows the number of vertices of the problem space graph in terms of the number of sequences that are sequentially chosen from $f_{\phi_{32}}(\mathcal{T}_{16}) - \mathcal{S}_{16}$.

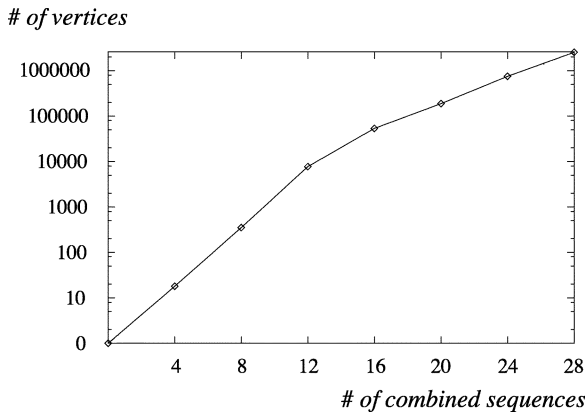

Fig. 3. Number of vertices in the problem space graph with respect to the number of the sequences combined (plotted on a log scale).

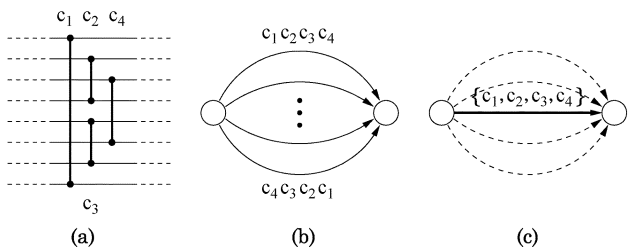

Fig. 4. Example of parallel layers. (a) Parallel layer. (b) Different paths. (c) Conceptual path.

C. Parallel Layers

In a sorting network, consecutively arranged independent comparators can be shuffled. Fig. 4(a) shows an example. In this figure, the four comparators can be arranged in any order. The permutations of such interchangeable comparators correspond to the different paths between two vertices in the problem space graph [Fig. 4(b)]. We handle these interchangeable comparators as a set, so that we can represent such different paths as a single conceptual path [Fig. 4(c)]. We call such a set of comparators a *parallel layer*. The network in Fig. 1 has three parallel layers.

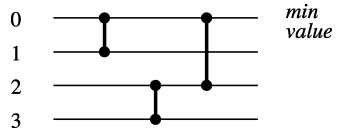

Fig. 5. Invalid four-bus network.

A parallel layer strongly affects the subsequent search direction. In the 16-bus problem, the four parallel layers, corresponding to the set ϕ_{32} of the first 32 comparators, are generally fixed in the front part of a network. As mentioned before, fixing these four parallel layers makes the search space significantly smaller.

If a considerable number of leading parallel layers have been determined, the rest may be straightforward. For example, if the first eight layers of a 60-comparator network have been determined, it is fairly easy to find the remaining comparators by the local optimization heuristic described in the next section. That is, the sorting network problem can be considered to be the problem of finding a considerable number of leading parallel layers. For this reason, we evolve only a fixed number of parallel layers from the front. The details are described in Section IV and Section V.

D. O-Pairs and N-Pairs

Let $t(i)$ be the ith bit value of a binary sequence t. For two input-bus indices x and y such that $x < y$ $(x, y = 0, 1, 2, \ldots, n-1)$, if $(f_\chi(t))(x) \leq (f_\chi(t))(y)$ for all $t \in \mathcal{T}_n$, then the sorting is acceptable with the sorting network χ as far as the two input buses are concerned. In this case, we call such an input bus pair (x, y) an *ordered pair* (o-pair) with respect to the network χ. On the other hand, if there exists a $t \in \mathcal{T}_n$ such that $(f_\chi(t))(x) > (f_\chi(t))(y)$, then the sorting is not guaranteed for the two buses. In this case, we call such an input bus pair (x, y) a *nonordered pair* (n-pair) with respect to the network χ.

Fig. 5 shows an invalid four-bus network. Whatever sequences we provide for this network, the input bus pairs (0,1), (0,2), and (0,3) are eventually in order since the 0th bus contains the minimum value of the four buses. The input bus pairs (0,1), (0,2), and (0,3) are, thus, o-pairs for the network. On the other hand, for the input bus pair (1,2), there exist unordered input sequences such as $t_i = 0100$. Thus (1,2) is an n-pair. The pairs (1,3) and (2,3) are also n-pairs.

We denote by $\mathrm{OP}(\chi)$ the entire set of o-pairs for a network χ and by $\mathrm{NP}(\chi)$ the entire set of n-pairs for it. Since the path from the starting vertex to a vertex v in the problem space graph corresponds to a network χ, we use $\mathrm{OP}(v)$ [or $\mathrm{NP}(v)$] and $\mathrm{OP}(\chi)$ [or $\mathrm{NP}(\chi)$] interchangeably. It is clear that $|\mathrm{OP}(\chi)| + |\mathrm{NP}(\chi)| = |\mathrm{OP}(v)| + |\mathrm{NP}(v)| = \binom{n}{2}$.

$\mathrm{NP}(v)$ [equivalently, $\mathrm{OP}(v)$] presents an upper bound for the distance from the vertex v to the end vertex in the problem space graph. This bound helps effective search in the problem space graph. We describe the details in Section IV.

IV. Solution Repair and Local Optimization

The local optimization consists of two sequential processes: edit and repair. The edit removes redundant comparators in a network, and the repair transforms an invalid network to a valid one by adding new comparators. These are performed after crossover and mutation in our GA.

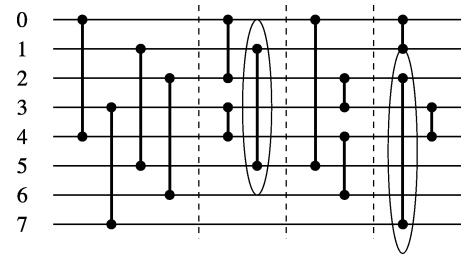

Fig. 6. Example of redundant comparators.

A. Edit

The edit removes redundant comparators in a network. Consider the network in Fig. 6. The input bus pair (1,5) is an o-pair with respect to the comparators in the first parallel layer; the marked comparator in the second layer is redundant. Similarly, the input bus pair (2,7) is an o-pair with respect to the comparators in the first three parallel layers; the marked comparator in the last layer is, thus, redundant.

As shown in the above example, if an input bus pair corresponding to a comparator is an o-pair with respect to its previous comparators, the comparator is redundant. We can find all the redundant comparators as follows. First, we provide all the test sequences to the network and perform sorting. In this process, we check the comparators that do not exchange any input pair. These comparators are those that do not contribute to sorting with respect to the test sequences. All these redundant comparators are removed in the edit process. After the edit process, there may be some parallel layer that has no comparator. In this case, we shift left the parallel layers following the empty layer.

For the n-bus problem, we assume that the number of comparators in a near-optimal network is $O(n \log n)$ according to [18]. Then, the time complexity of the edit is $O(|\mathcal{T}| n \log n)$, where $\mathcal{T}$ is the set of test sequences.

B. Repair

The repair process makes an invalid network valid by adding new comparators.

1) Appending: As seen in the previous section, the problem space can be represented by a graph G, where each path from the starting vertex to a nonend vertex stands for an invalid network. So every vertex except the end vertex matches an invalid network. The process of appending a new comparator to the network in the repair process corresponds to a movement from the vertex to one of its adjacent vertices in the graph. A shortest path from the vertex to the end vertex means a minimal use of comparators. Once a wrong direction is taken in the search process, the path to the end vertex can never be a shortest path.

We call the adjacent vertices of a vertex v in a graph G the explicit neighbors of v. We denote by $\mathcal{EN}(v)$ the set of the explicit neighbors of v. The best way to select a promising vertex from $\mathcal{EN}(v)$ is to choose a vertex whose distance to the end vertex is shortest. Let $d_G(v)$ be the shortest distance from vertex v to the end vertex. In general, it is impossible to find the exact value

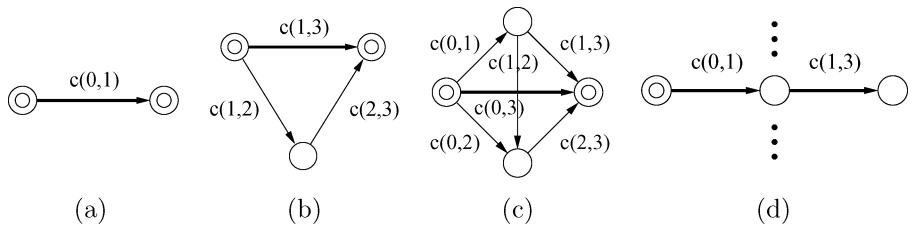

Fig. 7. Example of strongly related sequences. (a) $t_0 = 1011$. (b) $t_1 = 0100$. (c) $t_2 = 1000$. (d) Combined graph.

TABLE I
NUMBER OF N-PAIRS FOR EACH $n_v(i,j)$

$n_v(i,j)$	$n_v(0,1)$	$n_v(0,2)$	$n_v(0,3)$	$n_v(1,2)$	$n_v(1,3)$
1011	0111	1011	1011	1011	1011
0100	0100	0100	0100	0010	0001
1000	0100	0010	0001	1000	1000
$\lvert NP(n_v(i,j))\rvert$	2	4	3	4	3

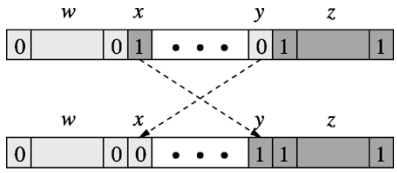

Fig. 8. Binary sequence t in T_k.

of $d_G(v)$ in tractable time. We have a clear (but rough) pair of bounds for $d_G(v)$ in the following fact.

Fact 1: For $G = \bigotimes_i G_i$

$$\max\{d_{G_i}(v)\} \le d_G(v) \le \sum_i d_{G_i}(v). \tag{4}$$

Although it is not difficult to obtain the upper bound of $d_G(v)$, it is too loose and not very helpful in selecting promising neighbors.

The following heuristic selects promising neighbors. First, in each graph G_i, we find the comparators that transit from v to the neighbors on the shortest paths from v to the end vertex of the graph. Among these comparators, we select a comparator that appeared the most often through all the G_i's. We select the vertex as a promising neighbor in G, to which vertex v is transited by this comparator.

However, since this heuristic is greedy, it may not be useful when the graphs are strongly related to one another. More precisely, if a sequence t_k is sorted as a result of sorting other sequences, it would not be desirable to consider G_k at this time. Fig. 7 shows an example. For a four-bus invalid network ϕ, assume that there are three sequences t_0, t_1, and t_2 in $f_\phi(\mathcal{T}_4) - \mathcal{S}_4$. Let $n_v(i,j)$ be the explicit neighbor vertex in G to which v is transited by comparator $\mathrm{c}(i,j)$ among the elements of $\mathcal{EN}(v)$ ($n_v(i,j) \in \mathcal{EN}(v)$). If we sort t_0 and t_1, then t_2 is sorted by itself. However, the above heuristic assigns the three neighbors $n_v(0,1)$, $n_v(1,3)$, and $n_v(0,3)$ equal importance.

Such cases motivate another heuristic which considers all G_i's together. Experiments indicate that the number of n-pairs is fairly effective in selecting a promising neighbor. In the example, we select the neighbor $n_v(0,1)$ by this measure. Table I shows the number of n-pairs remaining after transition by each possible comparator. We can get a tighter bound for $d_G(v)$ using the number of n-pairs.

Fact 2: For $G = \bigotimes_i G_i$

$$\max\{d_{G_i}(v)\} \le d_G(v) \le \lvert \mathrm{NP}(v)\rvert. \tag{5}$$

Proof: The lower bound is trivially true. We only prove the correctness of the upper bound. Let $v_0 = v$. For v_k such that $\mathrm{NP}(v_k) \neq \emptyset$ ($k = 0, 1, 2, \ldots$), we denote by p_k one of the n-pairs in $\mathrm{NP}(v_k)$ that contain the smallest input bus index among the n-pairs that contain the largest input bus index. We denote by v_{k+1} the vertex to which v_k is transited by the comparator corresponding to p_k.

Let x and $y (x < y)$ be the two input bus indices of p_k. We denote by χ_k the network corresponding to v_k, and by T_k the set of the sequences in $f_{\chi_k}(\mathcal{T}_n) - \mathcal{S}_n$ such that the xth bit value is 1 and the yth bit value is 0. Since $p_k = (x, y)$ is an n-pair in $\mathrm{NP}(v_k)$ that contains the smallest input bus index among the n-pairs that contain the largest input bus index, for any $t \in T_k$, $t(w) = 0\,\forall\, w < x$, and $t(z) = 1\,\forall\, z > y$. (Suppose that $t(w) = 1$ for some $w < x$). Then, (w, y) becomes an n-pair, which contradicts the fact that x is the smallest input bus index such that (x, y) is an n-pair for the fixed y. We can prove that $t(z) = 1\,\forall\, z > y$ in a similar way.) The comparator corresponding to p_k renders the pair of the xth and yth bit values of t in order and does not make any new nonordered bit pairs of t. On the other hand, it does not change any sequences in $f_{\chi_k}(\mathcal{T}_n) - \mathcal{S}_n$, except those in T_k (Fig. 8).

Therefore, the transition by the comparator corresponding to p_k changes p_k to an o-pair and does not make any new n-pairs, so that $\mathrm{NP}(v_{k+1}) \subseteq \mathrm{NP}(v_k) - \{p_k\}$. Hence, $\lvert\mathrm{NP}(v_{k+1})\rvert \le \lvert\mathrm{NP}(v_k)\rvert - 1$. In this way, each transition by the comparator corresponding to p_k ($k = 0, 1, 2, \ldots$) decreases the number of n-pairs by at least one and, consequently, we can reach the end point of G from v by at most $\lvert\mathrm{NP}(v)\rvert$ moves. This means that $d_G(v) \le \lvert\mathrm{NP}(v)\rvert$. □

2) Insertion: Not only do the comparators that are "appended" to a network transit the vertex v to promising neighbors, the comparators "inserted" in the middle of the network also have the potential to transit v to promising neighbors. When we consider an invalid network ϕ as a sequence of comparators, there are $\lvert\phi\rvert + 1$ candidate positions to which we can add a new comparator in the repair process.

When we insert (as opposed to "append") a comparator in the middle of ϕ, the comparator does not necessarily transit v to an adjacent vertex of v in the graph G. We call such a vertex, which is not adjacent to v, but can be transited to by a comparator insertion, an implicit neighbor of v. We denote by $\mathcal{IN}(v)$ the set of the implicit neighbors of v (Fig. 9).

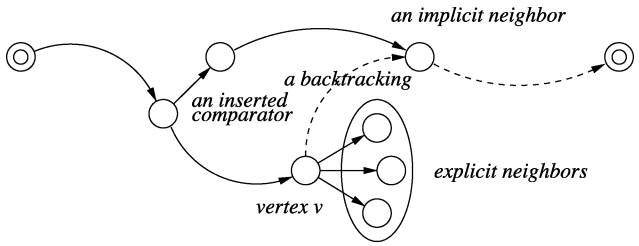

Fig. 9. Symbolic representation of an implicit neighbor.

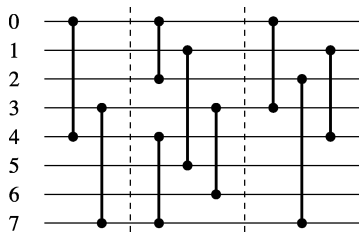

Fig. 10. Example of eight-bus network.

In selecting a promising neighbor of v, we are able to search the problem space more extensively by considering $\mathcal{IN}(v)$, as well as $\mathcal{EN}(v)$. However, it is too expensive to consider all the elements of $\mathcal{IN}(v)$. We restrict the search by protecting the boundaries of parallel layers.

We consider only the comparators that do not break the structure of the parallel layers in the repair process. Fig. 10 shows an example of eight-bus network. In the figure, the buses 0, 3, 4, and 7 have already been used in the first parallel layer. So, we do not consider comparators that connect at least one of these buses in the first layer. Instead, we consider only the $\binom{4}{2} = 6$ comparators that are obtained by connecting two buses among the four unused buses. In the second layer, we do not consider any comparators since all the buses in the layer have been used. In the third layer, there are two unused buses and we, thus, consider only the comparator c(5,6).

In this manner, we can search the elements of $\mathcal{IN}(v)$, where reasonably maintaining the structure of parallel layers. We call the vertices, which are not adjacent to v but to which v is transited in this way, the restricted implicit neighbors of v. We denote by $\mathcal{RN}(v)$ the set of the restricted implicit neighbors of v. It is clear that $\mathcal{RN}(v) \subseteq \mathcal{IN}(v)$.

We consider the elements of $\mathcal{RN}(v)$ with respect to the fixed number of parallel layers and the elements of $\mathcal{EN}(v)$ in the repair process. It seems desirable to select a comparator that reduces the number of n-pairs as much as possible among the comparators that transit v to $\mathcal{RN}(v) \cup \mathcal{EN}(v)$. Here, an inserted comparator can produce new redundant comparators after it, unlike the appended comparators; the length of the network can even be smaller after insertion. Thus, we also consider the edited lengths of the networks when we evaluate the elements of $\mathcal{RN}(v)$.

We select the promising neighbor of v according to the following order of priority.

1) A vertex in $\mathcal{RN}(v)$ such that its corresponding network has fewer comparators than the network of v and the number of its n-pairs is also smaller than $|\mathrm{NP}(v)|$.
2) A vertex in $\mathcal{RN}(v)$ such that its corresponding network has fewer comparators than the network of v and the number of its n-pairs is the same as $|\mathrm{NP}(v)|$.
3) A vertex in $\mathcal{RN}(v)$ such that its corresponding network has the same length as that of the network corresponding to v and the number of its n-pairs is smaller than $|\mathrm{NP}(v)|$.
4) A vertex in $\mathcal{RN}(v) \cup \mathcal{EN}(v)$ that has the smallest number of n-pairs.

When a tie occurs in 4), we give preference to the comparators that are inserted in the front side.[3]

As before, we assume that the number of comparators in a near-optimal network is $O(n \log n)$ for the n-bus problem. Then, the time complexity of the whole repair process is $O(|\mathcal{T}| n^3 \log n)$, where $\mathcal{T}$ is the set of test sequences.

The END model does not allow any backtracking process, so that a wrong direction taken in the search process cannot be recovered. On the other hand, the search for the implicit neighbors in the insertion is a backtracking process as shown in Fig. 9. This enhances the repair process's search.

C. Performance Improvement Using Function Value Tables

If we know the number of n-pairs in a network, we can decide whether the network is valid or not. Of course, it is intrinsically difficult to find the number of n-pairs. Since the edit and repair heuristics use the number of n-pairs as a measure of evaluating a network, counting the number of n-pairs consumes most of the running time. They count the number of n-pairs by feeding binary sequences into a network. Thus, the comparison operation for the comparators dominates the edit and repair processes.

As mentioned above, we consider an n-bus sorting network as a function from $\mathcal{T}_n$ to $\mathcal{T}_n$. Similarly, we can consider each comparator as a function from $\mathcal{T}_n$ to $\mathcal{T}_n$. Suppose that an n-bus network χ consists of the comparators $\mathrm{c}_0, \mathrm{c}_1, \ldots, \mathrm{c}_{|\chi|-1}$. If we denote by f_χ the function corresponding to χ and by f_{c_i} the function corresponding to $\mathrm{c}_i (0 \le i \le |\chi| - 1)$, then, for any $t \in \mathcal{T}_n$, it holds that

$$\begin{aligned} f_\chi(t) &= \left(f_{\mathrm{c}_{|\chi|-1}} \circ f_{\mathrm{c}_{|\chi|-2}} \circ \cdots \circ f_{\mathrm{c}_1} \circ f_{\mathrm{c}_0}\right)(t) \\ &= f_{\mathrm{c}_{|\chi|-1}}\left(f_{\mathrm{c}_{|\chi|-2}}\left(\cdots f_{\mathrm{c}_1}\left(f_{\mathrm{c}_0}(t)\right)\cdots\right)\right). \end{aligned} \tag{6}$$

If we know the images of all binary sequences under f_{c_i} for all $\mathrm{c}_i\ (0 \le i \le |\chi|-1)$, we could get the value $f_\chi(t)$ for any $t \in \mathcal{T}_n$ in just $|\chi|$ table lookups, instead of comparing a sequence of pairs.

We prepare a matrix in advance. In the matrix, each row represents an integer in $[0, 2^n - 1]$; each integer corresponds to a binary sequence in $\mathcal{T}_n$. Each column represents an integer in $[0, \binom{n}{2} - 1]$ and corresponds to a comparator. The element m_{ij} in the matrix represents the binary sequence after applying the jth comparator to the ith binary sequence. In the case of the 16-bus network problem, the memory space required to create such a table is $\binom{16}{2} \times 2^{16} \times 2 < 16$ Mbytes. Fig. 11 shows the matrix for the 16-bus problem. In addition to this, we generate a table that stores the pairs not in order for each $t \in \mathcal{T}_n$. By using

[3]Let $\mathrm{rand}[a, b]$ be a random integer in $[a, b]$. When the number of ties is t, we select the k^{th} comparator from the front in the order of inserted positions such that $k = \mathrm{rand}[1, \mathrm{rand}[1, t]]$.

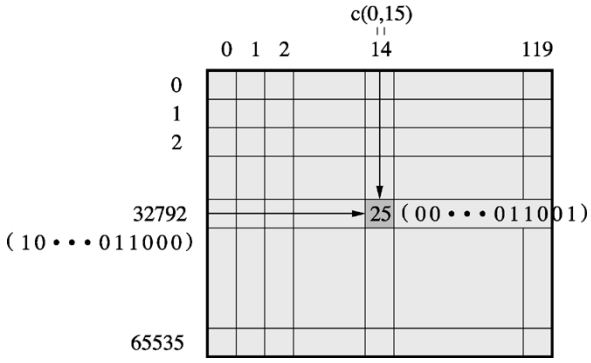

Fig. 11. Function value table.

```
create initial population of a fixed size;
do {
    choose parent1 and parent2 from population;
    offspring = Crossover(parent1, parent2);
    Mutation(offspring);
    Edit(offspring);
    GraphBasedRepair(offspring);
    Replace(population, offspring);
} until (stopping condition);
return the best individual;
```

Fig. 12. Outline of the hybrid GA.

these tables, we reduce considerably the computational time for the comparison operations.[4]

V. Genetic Algorithms (GAs)

We used a typical hybrid steady-state GA. Fig. 12 shows the outline of the hybrid GA. To evolve the parallel layers efficiently, we used a new encoding scheme with a Lamarckian–Baldwinian hybrid, and devised an appropriate crossover and mutation strategy. In this section, we first present the proposed encoding scheme and then describe the details of the rest of our GA.

A. Lamarckian–Baldwinian Hybrid Encoding Scheme

Each sorting network is represented by a chromosome. Fig. 13 shows the structure of a chromosome. A chromosome is composed of a fixed number of parallel layers and one supplemental layer. Each gene represents a pair of input buses and corresponds to a comparator. Each parallel layer consists of a bounded number (at most half the number of input buses) of independent comparators, and the supplemental layer consists of an unlimited number of comparators. There is no limit to the number of genes in a chromosome.

The hybrid GAs that combine local search and genetic search are generally classified into two groups: Lamarckian and Baldwinian [14], [24], [25]. Given a current solution, if the improved solution by local search replaces the current one, we refer to this as a Lamarckian GA. If the improved solution is merely used to evaluate the fitness of the current one, it is referred to as a Baldwinian GA.

In our GA, only the genes in the parallel layers are inherited to offspring in the process of crossover. On the other hand, the genes in the supplemental layer are used only for the evaluation of fitness. In other words, the genes in the parallel layers are updated as a result of local optimization (a form of Lamarckian evolution), while those in the supplemental layer are appended for fitness evaluation and then ignored in the genetic process (a kind of the Baldwin effect). This is a combination of Lamarckian and Baldwinian GAs.

Such a scheme is based on the observation that, when the chromosomal size of the Lamarckian side is properly determined, the Baldwinian side is easily optimized to a solution of best-known quality. This scheme alleviates the computational load of the Lamarckian side by ignoring some of the last comparators in the evolution. It reduces the problem search space and helps the GA to conduct an efficient search. The experiments in Section VI support this.

B. GA Framework

- **Initialization**: We set the population size to be 50. This is significantly smaller than those of Hillis' (65 536) [13] and Juillé's (4096) [16]. For each parallel layer in a chromosome, we randomly generate independent comparators. The number of independent comparators is chosen to be between a quarter and half the number of input buses. We then perform edit and repair processes to make the chromosomes valid.
- **Parent Selection**: Since all the chromosomes in the population are valid, we consider only the lengths of the chromosomes in selection; shorter chromosomes are rewarded. More formally, the fitness value F_i of chromosome i is calculated as

$$F_i = \left(\frac{1}{L_i} - \frac{1}{L_w}\right) + \frac{\left(\frac{1}{L_b} - \frac{1}{L_w}\right)}{3} \tag{7}$$

  where L_i, L_w, and L_b are the lengths of chromosome i, the worst (longest), and the best (shortest) in the population, respectively. Each chromosome is selected as a parent with a probability proportional to its fitness value. Thus, the probability that the best chromosome is chosen is four times as high as the probability that the worst is chosen. This is a typical proportional selection scheme.
- **Crossover**: As mentioned, we consider only the parallel layers of the two parents in crossover. We generate k cut points. (Cut-points occur only between the comparators.) Since the lengths of the two parents are often different, we first generate k logical points and translate them into "relatively" the same positions in the parents. The comparators in the parallel layers made in this way are usually not independent of one another. We convert each layer to a valid one by removing comparators until there is no comparator that shares the same bus with another comparator in the layer.
- **Mutation**: We randomly select each comparator with a low probability p_m and change one of the input buses at

[4]Experiments showed that the computational time was reduced to less than one third by using the tables.

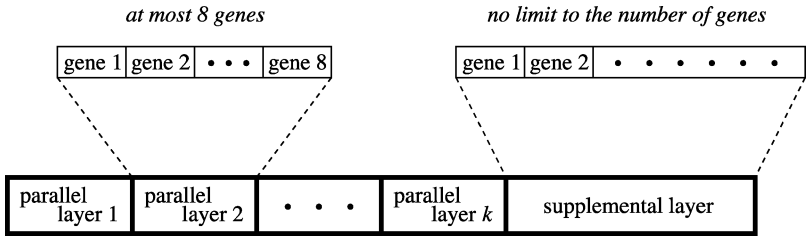

Fig. 13. Structure of a chromosome in 16-bus case.

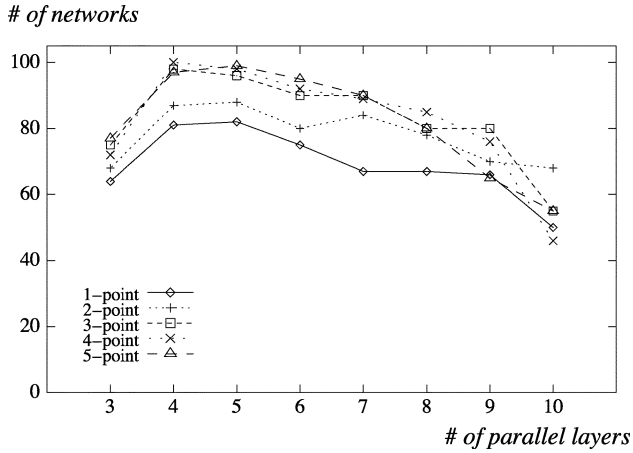

Fig. 14. Number of 60-comparator networks that GA found in 100 trials.

random. If there exists another comparator, say c, in the same layer that occupies the changed bus, we connect the comparator c to the (necessarily) absent bus after change.

- **Local Optimization**: Section IV described the edit and repair processes that are applied to each offspring after mutation.
- **Replacement**: We replace the inferior of the two parents if the offspring is not worse than both parents. Otherwise, we replace the worst member of the population. This scheme is a compromise between preselection [5] and GENITOR-style replacement [26], and showed successful results in [4].

VI. Experimental Results

Experiments were performed on an Intel Pentium III 866 MHz. We initialized each network with the first 32 comparators of Green's network as in most studies for the 16-bus problem [8], [13], [16] and evolved the population of networks using the GA.

First, we evaluated the performance of the GA according to the number of parallel layers and the number of cut points in crossover. Fig. 14 shows the number of 60-comparator sorting networks (of quality equivalent to the best known) that GA found in 100 trials.[5] When we used less than three parallel layers, we could not find any 60-comparator sorting network from 100 trials. The versions with four and five layers showed most desirable performances. This indicates the importance of appropriate number of parallel layers and the usefulness of Lamarckian–Baldwinian combination. Since the four-layer GA spent less time than the five-layer GA, we selected the four-layer GA with four-point crossover as the official version for comparisons with other approaches. (The mutation probability p_m was set to be 0.05.) The GA found 60-comparator sorting networks in all of the 100 trials in 40–190 s.

We conducted experiments with other meta-heuristics. First, we designed the multistart heuristic (MS) that tries multiple runs of our local heuristic on different initial random networks and produces the best as its final solution. Next, we combined our local heuristic with large-step Markov chains (LSMC), which is one of the meta-heuristics known to be efficient for large-sized combinatorial optimization problems [15], [20]. Starting from an initial solution, it performs a chain of "perturbation + local optimization." Fig. 15 shows the number of 60-comparator networks that MS and LSMC found in 100 trials according to the number of parallel layers and the mutation (perturbation) probabilities p_m.[6] MS could hardly find 60-comparator networks from 100 trials. This indicates that the local heuristic alone is not enough to solve the problem. LSMC showed the best performance with a rather high mutation $(p_m = 0.16)$. LSMC outperformed MS. However, the performance of LSMC did not reach that of GA. This shows the strong synergy between the local heuristic and the GA.

Table II summarizes the experimental results and the environments in the works of Hillis [13], Juillé (END, the version fixing the first 32 comparators) [16], and our study (graph-based Lamarckian–Baldwinian hybrid, GLBH). GLBH used a population size of 50, significantly smaller than the others, and showed the most stable performance in significantly less time even after considering the difference in CPU power.

When we examined the 100 networks with 60 comparators that we found using four parallel layers, there were 81 distinct sorting networks (of course ignoring the sequences in the same parallel step). Table III classifies the 81 sorting networks according to the number of parallel steps. We present in Fig. 16 one of the 60-comparator sorting networks (with ten parallel steps) that we found. Fig. 17 shows the best and average quality in GLBH over the number of generations in a typical run.

Finally, Table IV shows the experimental results of GLBH for the 8-, 10-, and 12-bus problems. For the 12-bus problem, we initialized each network with the 18 comparators to reduce the search space as in the 16-bus problem. (At this time, there remain 171 binary sequences to test among total $2^{12} = 4096$ sequences.) GLBH found the networks of best-known qualities for those instances in every run.

[5]For each number of parallel layers, the mutation probability p_m was optimized among a few values (0.01, 0.03, 0.05, 0.07, and 0.09).

[6]For MS and LSMC, we used the same amount of CPU time as that of GA.

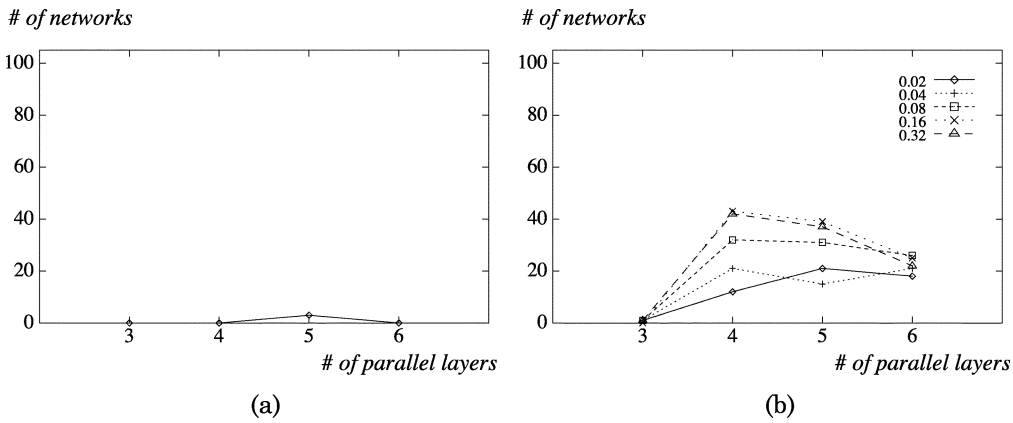

Fig. 15. Number of 60-comparator networks that MS and LSMC found in 100 trials. (a) Multistart. (b) LSMC.

TABLE II
COMPARISON OF EXPERIMENTAL RESULTS AND ENVIRONMENTS

	Hillis [13]	END [16]	GLBH
Population size	65,536	4,096	50
Machine	CM-1	Maspar MP-2 (17,000 MIPS)	Pentium III 866 MHz (2,340 MIPS†)
# of processors	65,536	4,096	1
Results	61 comparators	60 comparators, almost 100 %	60 comparators, 100 % for 100 runs
Execution time	5 to 50 min	5 to 10 min	40 to 190 sec (average 92.1 sec)

† The value is from the Sandra benchmark test (http://www.sisoftware.net).

TABLE III
NUMBER OF DISTINCT SORTING NETWORKS ACCORDING TO THE NUMBER OF PARALLEL STEPS

# of parallel steps	10	11	12	Total
# of sorting networks	29	19	33	81

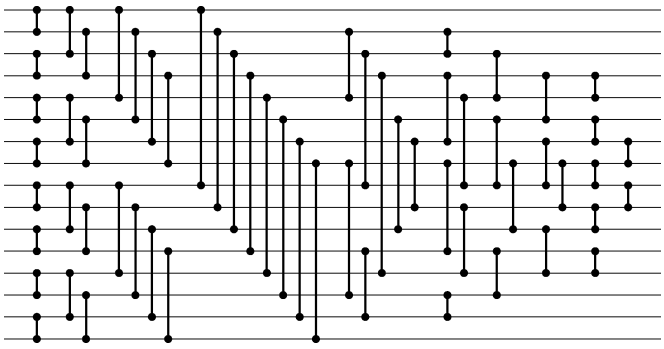

Fig. 16. A 60-comparator sorting network that we found.

VII. CONCLUSION

We considered the sorting network problem space as a graph. This viewpoint provided us with a new insight into the problem, which led us to a more formal approach. By devising the concept of o-pairs and n-pairs, we could efficiently find the redundant comparators and could select promising comparators to repair an invalid network. We hope that this type of graph transformation of problem spaces will provide a new pathway to the solution of other combinatorial optimization problems.

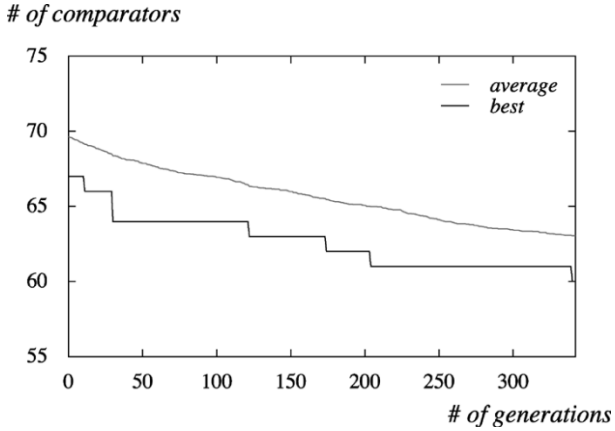

Fig. 17. Best and average lengths according to the number of the generations in GLBH.

TABLE IV
EXPERIMENTAL RESULTS FOR OTHER INSTANCES

Instance	8-bus	10-bus	12-bus
Population size	50	50	50
Machine	Pentium III 866 MHz	Pentium III 866 MHz	Pentium III 866 MHz
Results	19 comparators 100 % for 100 runs	29 comparators, 100 % for 100 runs	39 comparators, 100 % for 50 runs
Execution time	up to 20 sec	23 to 193 sec (average 91.6 sec)	20 to 204 sec (average 77.2 sec)

The invalid solutions produced after crossover and mutation need a repairing algorithm. We incorporated strong local optimization in the repairing process. The concept of parallel layers plays an important role in the local optimization and the genetic process.

Another notable aspect of the proposed GA is that it evolves only a fixed number of parallel layers; the remaining layers are appended by the local optimization just for the evaluation of the solutions. This is a combination of Lamarckian and Baldwinian GAs. This significantly reduces the problem search space and helps the GA to conduct an efficient search.

When the local optimization heuristic and the GA were combined, they showed strong synergy. We found solutions of the best-known quality for the 16-bus problem with a fairly small

time budget. To the best of our knowledge, this is the first report (with preliminary studies in [6] and [7]) that found 60-comparator sorting networks on a (single-CPU) PC environment.

There are, however, a few remaining problems. We restricted the problem space by fixing the first 32 comparators. Although it is often believed that the global optimum would contain the fixed 32 comparators, no actual proof exists. The global optimum may be in the restricted space, outside of it, or in both. We are currently working on the theoretical justification of the restriction. The study includes experiments using the model without fixing any comparators. We will also consider greater-than-16 bus problems in the future.

ACKNOWLEDGMENT

The authors would like to thank the ICT at Seoul National University who provided research facilities for this study.

REFERENCES

[1] K. E. Batcher, "A new internal sorting method," Goodyear Aerospace, Rep. GER-11 759, 1964.
[2] R. K. Belew and T. Kammayer, "Evolving aesthetic sorting networks using developmental grammars," in *Proc. 5th Int. Conf. Genetic Algorithms*, 1993, p. 629.
[3] R. C. Bose and R. J. Nelson, "A sorting problem," *J.f ACM 9*, pp. 282–296, 1962.
[4] T. N. Bui and B. R. Moon, "Genetic algorithm and graph partitioning," *IEEE Trans. Comput.*, vol. 45, no. 7, pp. 841–855, Jul. 1996.
[5] D. Cavicchio, "Adaptive search using simulated evolution," Ph.D. dissertation, Univ. Michigan, Ann Arbor, MI, 1970.
[6] S. S. Choi and B. R. Moon, "A graph-based approach to the sorting network problem," in *Congr. Evol. Comput.*, 2001, pp. 457–464.
[7] ——, "A hybrid genetic search for the sorting network problem with evolving parallel layers," in *Proc. Genetic Evol. Comput. Conf.*, 2001, pp. 258–265.
[8] G. L. Drescher, "Evolution of 16-number sorting networks revisited," Unpublished manuscript, 1994.
[9] J. R. Koza *et al.*, "Evolving sorting networks using genetic programming and the rapidly reconfigurable Xilinx 6216 field-programmable gate array," in *Proc. 31st Asilomar Conf. Signals, Syst. Comput.*, vol. 1, 1998, pp. 404–410.
[10] R. W. Floyd and D. E. Knuth, "Improved constructions for the Bose–Nelson sorting problem," *Notices Amer. Math. Soc. 14*, p. 283, 1967.
[11] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.
[12] M. W. Green, "Some improvements in nonadaptive sorting algorithms," Stanford Res. Inst., Menlo Park, CA, Tech. Rep., 1969.
[13] W. D. Hillis, "Coevolving parasites improve simulated evolution as an optimization procedure," in *Artificial Life II*, C. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, Eds. Reading, MA: Addison-Wesley, 1992, pp. 313–324.
[14] G. E. Hinton and S. J. Nowlan, "How learning can guide evolution," *Complex Syst.*, vol. 1, no. 3, pp. 495–502, 1987.
[15] I. Hong, A. B. Kahng, and B. R. Moon, "Improved large-step Markov chain variants for the symmetric TSP," *J. Heuristics*, vol. 3, no. 1, pp. 63–81, 1997.
[16] H. Juillé, "Evolution of nondeterministic incremental algorithms as a new approach for search in state spaces," in *Proc. 6th Int. Conf. Genetic Algorithms*, 1995, pp. 351–358.
[17] ——, "Incremental coevolution of organisms: A new approach for optimization and discovery of strategies," in *Proc. 3rd Eur. Conf. Artif. Life*, 1995, pp. 246–260.
[18] D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd ed. Reading, MA: Addison-Wesley, 1998.
[19] S. Levy, *Artificial Life: The Quest for a New Creation*. New York: Pantheon Books, 1992.
[20] O. Martin, S. W. Otto, and E. W. Felten, "Large-step Markov chains for the traveling salesman problem," *Complex Syst.*, vol. 5, pp. 299–326, 1991.
[21] I. Parberry, "A computer-assisted optimal depth lower bound for nine-input sorting networks," *Math. Syst. Theory*, vol. 24, pp. 101–116, 1991.
[22] C. Ryan, "Pygmies and civil servants," *Adv. Genetic Program.*, pp. 243–263, 1994.
[23] G. Shapiro, *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd ed, D. E. Knuth, Ed. Reading, MA: Addison-Wesley, 1998.
[24] G. G. Simpson, "The Baldwin effect," *Evolution*, vol. 7, pp. 110–117, 1953.
[25] D. Whitley, V. Gordon, and K. Mathias, "Lamarckian evolution, the Baldwin effect and function optimization," in *Proc. Int. Conf. Evol. Comput.*, 1994, pp. 6–15.
[26] D. Whitley and J. Kauth, "Genitor: A different genetic algorithm," in *Proc. Rocky Mountain Conf. Artif. Intell.*, 1988, pp. 118–130.

Sung-Soon Choi received the B.S. degree in computer science from Seoul National University, Seoul, Korea, in 2000. He is currently working towards the Ph.D. degree in the School of Computer Science and Engineering, Seoul National University.

His research interests include algorithm design/analysis, combinatorial optimization, and evolutionary computation.

Byung-Ro Moon received the B.S. degree in computer science and statistics from Seoul National University, Seoul, Korea, the M.S. degree in computer science from the Korea Advanced Institute of Science and Technology, Daejeon, Korea, and the Ph.D. degree from the Pennsylvania State University, University Park, in 1985, 1987, and 1994, respectively.

From 1987 to 1991, he was an Associate Research Engineer in the Central Laboratory, LG Electronics Company, Ltd., Seoul, Korea. From November 1994 to 1995, he was a Postdoctoral Scholar in the VLSI CAD Laboratory, University of California, Los Angeles. From 1996 to 1997, he was a Principal Research Staff Member in the DT Research Center, LG Semicon Co., Ltd., Seoul, Korea. Since September 1997, he has been an Associate Professor in the School of Computer Science and Engineering, Seoul National University, Seoul, Korea. His major interest is the theory and application of optimization methodologies (evolutionary computation, algorithm design/analysis, optimization modeling, etc.). The applications of optimization include combinatorics, one-to-one marketing, e-commerce, search, graph partitioning, circuit layout, and scheduling.

Prof. Moon was the recipient of Korean Government Scholarships during the 1991–1994 academic years. He served as a committee member of GP'97, GP'98, ISIAC'98, and GECCO'99 through 2004. He was the Publication Chair of the IEEE CEC'2001.