Isomorphism, Normalization, and a Genetic Algorithm for Sorting Network Optimization

Sung-Soon Choi and Byung-Ro Moon
School of Computer Science and Engineering,
Seoul National University,
Seoul, 151-742 Korea
{irranum,moon}@soar.snu.ac.kr

Abstract

In this paper, we define sorting network isomorphism and examine its relationship to graph-theoretic problems. We devise the normalization technique that exploits the functional similarities of sorting networks, which in turn helps genetic algorithms avoid too much perturbation. The sorting network isomorphism provides the basis for the normalization. In addition, we developed an effective local search heuristic for the problem. Combining the local heuristic with a genetic algorithm, we found 60-comparator sorting networks in the 16-bus problem without fixing any comparators on a single-CPU PC. This result is significantly faster and more stable than the previous study conducted with a supercomputer.

1 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 is composed of buses and a number of homogeneous comparators, where each comparator $c(a,b)$ performs the elementary operation that compares the ath and b^{th} buses; if the values are in order, ignore them, otherwise exchange them. We call a sorting network for n inputs an n-bus sorting network. For any input sequence of an n-bus sorting network, the output sequence $y_0, y_1, \ldots, y_{n-1}$ is monotonically nondecreasing ($y_0 \leq y_1 \leq \ldots \leq y_{n-1}$). Figure 1 shows a 4-bus sorting network [$c(0,1)$, $c(2,3)$, $c(0,2)$, $c(1,3)$, $c(1,2)$].

Historically many scientists had made studies of 16-bus sorting networks [3] [11] [1] [21] [13]. In 1969, Green [13] first discovered a 60-comparator sorting network
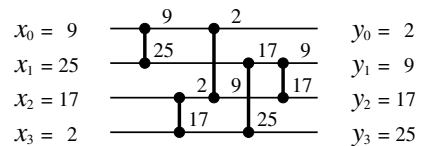

Figure 1: A 4-bus sorting network

which is still one of the best known. Recently, 16-bus sorting networks have attracted attention again due to the improvements in new stochastic search methods [14] [2] [8] [16] [10] [6] [7].

Most of these studies initialized the networks with the first 32 comparators of Green's network to reduce the size of the problem space [14] [8]. To date Juillé [16] is the only one who attacked the problem without fixing the first 32 comparators. He found 60-comparator sorting networks with a stochastic search method called END (Evolving Non-Determinism) on a Maspar MP-2 supercomputer. Even most other studies that fixed the first 32 comparators used supercomputers [14] [8].

In [6] and [7], the authors first found 60-comparator sorting networks on a single-CPU PC with the first 32 comparators fixed. In this paper, we attack the problem without fixing any comparators. Under the single CPU environment again, we found 60-comparator sorting networks. This is thought to be possible by the following: First, we devised a process to maintain the consistency between two parent networks, which in turn helps the genetic algorithm undergo effective exploitation. Next, we designed an effective local optimization heuristic customized to the problem. Finally, we incorporated the local optimization heuristic into the genetic framework to lead a strong synergy.

The rest of this paper is organized as follows. In Section 2, we present some underlying theory related to the sorting network and our theoretical view of it. In
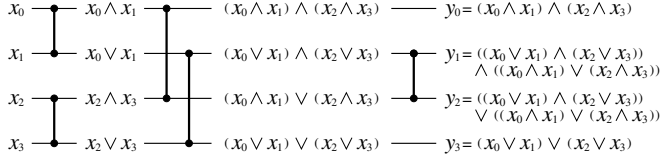
Figure 2: Formulae for the contents of buses

Section 3, we define the sorting network isomorphism and describe normalization as its application to the genetic algorithm. In Section 4, we propose our local optimization heuristic, and in Section 5 we provide a genetic algorithm combined with the local optimization heuristic. We give the experimental results in Section 6. Finally, we make our conclusions in Section 7.

2 Preliminaries

2.1 Boolean Characteristics of Sorting Networks

A valid sorting network guarantees to sort any input sequence in order. It is, however, possible to restrict the inputs to binary sequences by the following [17]:

Theorem 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 will sort any arbitrary sequence of n numbers into nondecreasing order.*

Then we are able to write down two outputs, min(a,b) and max(a,b), of a comparator c(a,b) as follows (note that a and b are binary numbers): $\min(a,b) = a \wedge b$ and $\max(a,b) = a \vee b$. Generalizing this, for a given sorting network, the contents of any points on a bus can be expressed in disjunctive normal form (DNF) with inputs as variables. We denote the input and output on the kth bus by x_k and y_k ($0 \leq k \leq n-1$) in an n-bus network. Figure 2 illustrates an example. In the figure, the output on 0th bus, $y_0 = (x_0 \wedge x_1) \wedge (x_2 \wedge x_3)$, can be reduced to $x_0 \wedge x_1 \wedge x_2 \wedge x_3$. Similarly, y_1, y_2, and y_3 can be reduced to $(x_0 \wedge x_1 \wedge x_2) \vee (x_0 \wedge x_1 \wedge x_3) \vee \cdots \vee (x_1 \wedge x_2 \wedge x_3)$, $(x_0 \wedge x_1) \vee (x_0 \wedge x_2) \vee \cdots \vee (x_2 \wedge x_3)$, and $x_0 \vee x_1 \vee x_2 \vee x_3$, respectively, by axioms of boolean algebra.

Let σ_k be the kth smallest element in the set of inputs $\{x_0, x_1, \ldots, x_{n-1}\}$ ($0 \leq k \leq n-1$). Then, the following holds in general [17]:

$$\sigma_k = \bigvee \{x_{i_0} \wedge x_{i_1} \wedge \cdots \wedge x_{i_{n-k-1}} \mid$$
$$0 \leq i_0 < i_1 < \cdots < i_{n-k-1} \leq n-1\}.$$

In a valid n-bus sorting network, σ_k corresponds to the kth output, i.e., $y_k = \sigma_k \; \forall \; k$.

2.2 Transforming to Valid Networks

In this section, we briefly review our previous works in [6] and [7] to transform invalid networks to valid ones.

2.2.1 O/N-pairs and Parallel Layers

The Zero-One Principle [17] says that we can prove the validity of an n-bus sorting network by testing just 2^n binary sequences instead of $n!$ 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 $\mathcal{T}_n$. ($\mathcal{S}_n \subseteq \mathcal{T}_n$) We can consider an n-bus network as a function from $\mathcal{T}_n$ to $\mathcal{T}_n$. Then we denote by f_χ the function corresponding to a sorting network χ.

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. 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. We call such an input bus pair (x,y) a *non-ordered pair* (*n-pair*) with respect to the network χ. We denote by $OP(\chi)$ the entire set of o-pairs for a network χ and by $NP(\chi)$ the entire set of n-pairs for it. It is clear that $|OP(\chi)| + |NP(\chi)| = \binom{n}{2}$.

In a sorting network, a number of consecutively located independent comparators are allowed to be shuffled. We handle these interchangeable comparators as a group, since the sequence of these comparators does not affect the function of the network. We call such a group of comparators a *parallel layer*. For example, the network in Figure 1 is composed of three parallel layers.

A parallel layer strongly affects the subsequent search direction. If a considerable number of leading parallel layers have been determined, the rest may be easily constructed without redundancy by the repair heuristic of the next section. From the perspective of parallel layers, the sorting network problem can be considered to be the problem of finding a considerable number of leading parallel layers. For this reason, we evolved only a fixed number of leading parallel layers; this significantly reduced the computational load in [7].

2.2.2 Edit and Repair Heuristics

In [6] and [7], we devised two heuristics, edit and repair, to enhance the GAs' fine-tuning around local optima. The edit removes redundant comparators in a network. If the input bus pair of a comparator is an o-pair with respect to its previous comparators, the comparator is redundant. The repair modifies an invalid network to a valid one. In [6] and [7], the number of n-pairs of a network was used as a measure evaluating the quality of the network. The repair builds a valid network by adding a set of comparators that reduce the largest number of n-pairs. The repair consists of two operations: appending and insertion. The appending adds comparators in the rear part of a network; the insertion adds comparators in the middle of a network. See [7] for details.

3 Isomorphism and Normalization

3.1 Definition of Sorting Network Isomorphism

If we denote by $|\chi|$ the number of comparators in χ, a sorting network χ is generally represented by a sequence of comparators $c_0, c_1, \ldots, c_{|\chi|-1}$. And each comparator c_i is represented by a pair of input bus indices (a_i, b_i). Here, the input bus indices a_i and b_i indicate only the absolute locations of the input buses of c_i. We cannot know the relationship between c_i and the input bus index system of χ, from the values of a_i and b_i.

In the previous section, we saw that an intermediate value of any point on a bus can be expressed in DNF with inputs as variables. For a comparator c_i in a network $\chi = c_0, c_1, \ldots, c_{|\chi|-1}$ $(0 \leq i \leq |\chi| - 1)$, we denote by $DI(c_i)$ the set of the DNFs of c_i's inputs. $(|DI(c_i)| = 2)$ The sequence of $DI(c_i)$'s reflects the construction process of χ under a given input bus index system.

Let S_n be the set of all the permutations of input set $\{x_0, x_1, \ldots, x_{n-1}\}$. For a DNF $d = \bigvee_i \bigwedge_j x_{ij}$ $(x_{ij} \in \{x_0, x_1, \ldots, x_{n-1}\})$ and a permutation $p \in S_n$, we define $p(d)$ as

$$p(d) = \bigvee_i \bigwedge_j p(x_{ij}) .$$

Now, we define the *sorting network isomorphism* as follows:

Definition 1 *For two n-bus sorting networks* $\chi = c_0, c_1, \ldots, c_{|\chi|-1}$ *and* $\chi' = c'_0, c'_1, \ldots, c'_{|\chi'|-1}$,
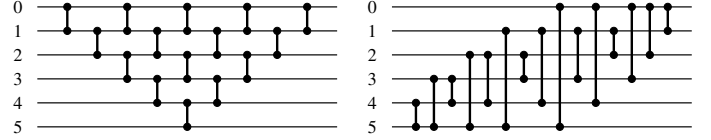 χ *is* isomorphic to χ' $(\chi \simeq \chi')$

Figure 3: Two isomorphic networks

$$\iff |\chi| = |\chi'| \ and \ \exists p \in S_n \ such \ that$$
$$p(DI(c_i)) = DI(c'_i) \ \forall i = 0, 1, \ldots, |\chi| - 1.$$

From the definition, two isomorphic networks are constructed essentially in the same way, and their input bus indices may be different. In other words, a network can be transformed into another isomorphic network by appropriate permutation of the bus indices and comparator reconnection. Here, we should note that the comparator reconnection is more than the simple transposition of comparators' end points according to the permutation.

Figure 3 shows an example of two isomorphic networks. Between the two networks, there are a permutation $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 5 & 4 & 3 & 2 & 1 & 0 \end{pmatrix}$ and a nontrivial reconnection of comparators. Although they are quite different in appearance, they are the same networks in viewpoint of the construction scheme.

3.2 Validity Preserving Property Under Isomorphism

From the definition of sorting network isomorphism, given an n-bus network χ and a permutation $p \in S_n$, we can construct the network χ' isomorphic to χ. Suppose that $\chi = c_0, c_1, \ldots, c_{|\chi|-1}$ and $\chi' = c'_0, c'_1, \ldots, c'_{|\chi|-1}$. We choose the bus pair, as the input bus pair of c'_i, whose DNF set coincides with $p(DI(c_i))$. Figure 4 shows the construction algorithm for an isomorphic network. Note that, in the algorithm, the permutation keeps changing according to the history of construction.

Generally, the validity is preserved among isomorphic networks. This property is utilized in the process of normalization in Section 3.4.

Proposition 1 *For a valid sorting network* χ, *every network isomorphic to* χ *is valid.*

Proof: Omitted by space limitation. ■

In Figure 3, the left network is valid; accordingly, the right network is also valid.

```
ConstructIsomorphicNetwork(χ, p) (const-network)
// χ : a given sorting network
//      c_0, c_1, ..., c_{|χ|−1}, c_i = (a_i, b_i)
// χ′ : the isomorphic network returned
//      c′_0, c′_1, ..., c′_{|χ|−1}, c′_i = (a′_i, b′_i)
// p : a permutation ∈ S_n
{
    for i ← 0 to |χ| − 1 {
        a′_i ← min(p(a_i),p(b_i));
        b′_i ← max(p(a_i),p(b_i));
        p(a_i) ← a′_i;
        p(b_i) ← b′_i;
    }
    return χ′;
}
```

Figure 4: The construction algorithm

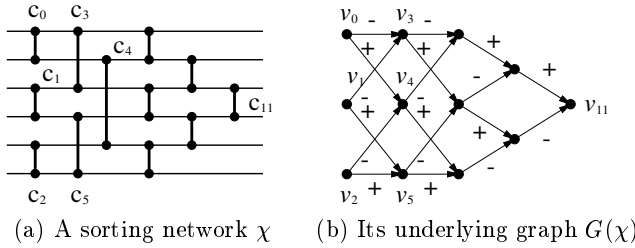(a) A sorting network χ (b) Its underlying graph $G(\chi)$

Figure 5: An example of an underlying graph

3.3 Underlying Graphs and Isomorphism

Given two networks χ and χ', the sorting-network isomorphism problem asks whether they are isomorphic or not. Considering the definition of sorting network isomorphism, it seems that the problem itself is rather complicated. However, we intend to look at the problem from another angle.

Sorting networks have been generally represented in such a way as in Figure 5(a). Since it draws buses across comparators in order to focus on the flows of data on buses, it is not appropriate in representing the relationship among comparators. We devised a model to better represent the relationship among comparators.

Set a vertex for each comparator. If an output of comparator c_i is fed into comparator c_j as an input, connect vertex v_i and vertex v_j by an arc e_{ij} from v_i to v_j. At this time, mark e_{ij} with "+" if it represents the maximum output of c_i. Otherwise, mark e_{ij} with "−". We have a directed graph $G = (V, E)$ corresponding to a given sorting network where V is the set of vertices and E is the set of arcs. (e.g., Figure 5(b))

For a given network χ, we call such a directed graph obtained from χ an *underlying graph* of χ and de-

note by $G(\chi)$. It is obvious that those graphs are acyclic and the degrees of their vertices are bounded by 4. $G(\chi)$ represents the relationships among the comparators of χ; consequently $G(\chi)$ represents the construction scheme of χ. There is strong relationship between the sorting network isomorphism and underlying graphs.

For two directed graphs $G = (V, E)$ and $G' = (V', E')$, G and G' are isomorphic if and only if there exists a bijection $f : V \mapsto V'$ such that $w(x, y) = w(f(x), f(y))$ for every ordered pair (x,y) of vertices in V, where $w(i, j)$ indicates the mark of the arc from vertex i to vertex j [18]. At this time, we denote by $G \simeq G'$. It holds, in general, that if two networks are isomorphic, then their underlying graphs are also isomorphic, and vice versa.

Proposition 2 *For two networks χ and χ',*

$$\chi \simeq \chi' \iff G(\chi) \simeq G(\chi') .$$

Proof: Omitted by space limitation. ∎

Since sorting networks can be efficiently transformed to their underlying graphs and the degree of any vertex in the underlying graphs are bounded by 4, Proposition 2 implies that the sorting network isomorphism problem is polynomially reduced to the bounded valence graph isomorphism problem.

It is an open problem whether the general graph isomorphism problem is NP-complete or not [12] [18]. However, it is known that the isomorphism of graphs of bounded valence can be tested in polynomial time [20]. So the sorting network isomorphism problem can be efficiently solved.

3.4 Normalization of Networks

If we perform crossover with two near-isomorphic solutions of significantly different shapes, the offspring will be significantly different from both parents. For example, consider Figure 6(a) and Figure 6(b). Although the subnetworks with the first eight comparators in (a) and (b) are totally different, they are isomorphic; they have the same construction schemes. If we crossover the two networks as they stand in appearance, these common attributes of the two networks are prone to be broken in the process of crossover. In a parent's point of view, the crossover is like too strong a mutation. We minimize this "visual inconsistency" before crossover.

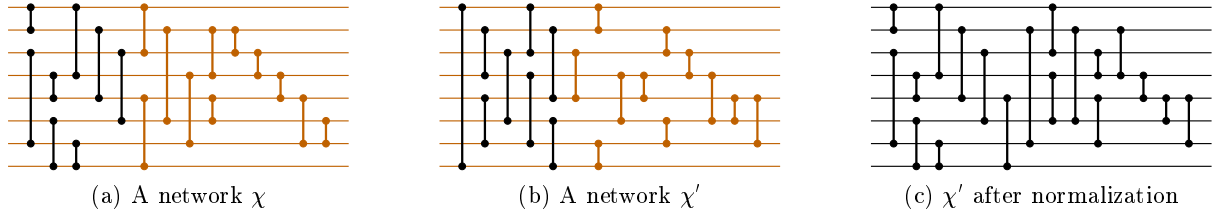We define the *distance* of two networks as follows:

(a) A network χ (b) A network χ' (c) χ' after normalization

Figure 6: An illustration of normalization process

Definition 2 *For two networks* $\chi = c_0, c_1, \ldots, c_{|\chi|-1}$ *and* $\chi' = c'_0, c'_1, \ldots, c'_{|\chi'|-1}$, *let* $s = max\{t | c_i = c'_i, 0 \le i \le t - 1\}$. *The* distance *between the two networks* χ *and* χ' *is defined as*

$$d(\chi, \chi') = max\{|\chi|, |\chi'|\} - s .$$

It is known that a network can be reconstructed so that the distance between the network and its isomorphic one is 0.

We transform one of the parents as closely as possible to the other parent. We call such a transformation *normalization*. Although much simpler, normalization had been performed for genetic multiway graph partitioning to help maintain consistency between two parents [19].

Given a mapping of buses to minimize the distance between networks, normalization can be achieved by the construction algorithm in the previous section. In order to find the mapping, we proceed layer by layer to the right and check whether the underlying graphs, corresponding to the subnetworks (to the layer) of the two parents, are isomorphic or not. If the two graphs are not isomorphic at the kth layer for the first time, we have found an isomorphism between the underlying graphs corresponding to the two subnetworks from the first layer to the $k - 1^{th}$ layer. Then we match the bus indices of the corresponding source vertices (whose in-degrees are zero) of the graphs from the isomorphism, so as to generate the mapping of buses.[1] Figure 7 shows the outline of normalization.

Figure 6(c) shows the network after the network in Figure 6(b) is normalized with respect to the network in Figure 6(a), where the mapping is $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 7 & 1 & 2 & 5 & 4 & 3 & 6 \end{pmatrix}$. The actual crossover is conducted between the networks (a) and (c).

---

[1] [20] guarantees that such normalization can be efficiently processed in viewpoint of computation theory. Since we do not need the precise algorithm, we perform the normalization process using a type of labeling heuristic that is fast but allows errors to some degree.

```
Normalize(χ,χ')
// l, l' : numbers of parallel layers of χ and χ'
// χ_i : the i^th parallel layer of χ (0 ≤ i ≤ l − 1)
// χ'_i : the i^th parallel layer of χ' (0 ≤ i ≤ l' − 1)
{
    k ← min{l − 1, l' − 1};
    for i ← 0 to min{l − 1, l' − 1} {
        if (G(χ_0 ··· χ_i) ≇ G(χ'_0 ··· χ'_i)) {
            k ← i;
            break;
        }
    }
    if (k > 0) {
        generate the mapping p
            from G(χ_0 ··· χ_{k−1}) and G(χ'_0 ··· χ'_{k−1});
        return const-network(χ,p);
    } else {
        return χ';
    }
}
```

Figure 7: The outline of normalization

4   Local Optimization

We mentioned before that we had devised edit and repair heuristics to enhance the GA's fine-tuning around local optima in [6] and [7]. The repair heuristic is a type of greedy and constructive algorithm that modifies an invalid network to a valid one with the number of n-pairs as a measure. Although they were somewhat powerful in the version that fixes the first 32 comparators, we find that they are not enough for the version of the problem that does not fix any comparators. The problem space is incomparably huge and more powerful local optimization is desired. We designed a local search heuristic to search the problem space around a valid network after repair.

From the viewpoint of underlying graphs, the optimal sorting network problem can be considered to be the problem of finding the edges that optimally connect the vertices corresponding to given comparators — the problem of finding the optimal network topology. We consider the networks, obtained by exchanging every two input bus indices of comparators in each layer of

```
    create initial population of a fixed size;
    do {
        choose parent1 and parent2 from population;
        parent2 ← normalization(parent1, parent2);
        offspring ← crossover(parent1, parent2);
        mutation(offspring);
        edit(offspring);
        repair(offspring);
        local-optimization(offspring);
        replace(population, offspring);
    } until (stopping condition);
    return the best individual;
```

Figure 8: The outline of the hybrid genetic algorithm

χ proceeding from left to right, as neighbor networks of χ. This exchanging process is equivalent to swapping two input-output pairs of the two vertices corresponding to two comparators in the same layer in the underlying graph $G(\chi)$. Such a strategy considerably reduces the computational load of local search for the following reason.

Suppose that the number of parallel layers in χ is l, and let the parallel layers of χ be $\chi_0, \chi_1, \ldots, \chi_{l-1}$; let the functions corresponding to these parallel layers be $f_{\chi_0}, f_{\chi_1}, \ldots, f_{\chi_{l-1}}$. In general, if χ does not have any redundant comparator, the number of unsorted binary sequences steeply decreases as the parallel layers are added one by one. In other words,

$$|f_{\chi_0}(\mathcal{T}_n) - \mathcal{S}_n| \gg |f_{\chi_1}(f_{\chi_0}(\mathcal{T}_n)) - \mathcal{S}_n| \gg \cdots$$
$$\gg |f_{\chi_{l-1}}(f_{\chi_{l-2}} \cdots f_{\chi_0}(\mathcal{T}_n) \cdots) - \mathcal{S}_n|.$$

Therefore, we can considerably reduce the time of evaluating the neighbors by considering only the unsorted sequences instead of all the sequences in $\mathcal{T}_n$. Such a strategy, which successively improves layers left to right, is natural in that a parallel layer strongly affects the subsequent layers [7].

5 GA Framework

We used a typical hybrid steady-state genetic algorithm. Figure 8 shows the outline of the hybrid genetic algorithm. We describe the details of our genetic algorithm in the following.

- **Encoding:** Each sorting network is represented by a chromosome. A chromosome is composed of a fixed number of parallel layers and one supplemental layer. Each gene is represented by a pair of input buses and corresponds to a comparator. Each parallel layer consists of a bounded number of independent comparators and the supplemental

layer consists of an unlimited number of comparators.

In our GA, only the genes in the parallel layers are used for crossover. On the other hand, the genes in the supplemental layer are used only for the evaluation of fitness; genes in the supplemental layer are appended by repair and local optimization. This is a variant of Baldwinian hybrid GAs [22] [15] [23].

- **Initialization:** We set the population size to be 100. 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 chromosome valid.

- **Parent Selection:** The fitness value F_i of chromosome i is calculated as follows:

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

where

L_w : the length of the worst (longest),
L_b : the length of the best (shortest), and
L_i : the length of chromosome i.

Each chromosome is selected as a parent with a probability proportional to its fitness value. This is a typical proportional selection scheme.

- **Normalization:** As mentioned in Section 3, we normalize one of the parents with respect to the other before crossover.

- **Crossover:** As mentioned, we consider only the parallel layers of the two parents in crossover. We generate five cut points. Since the lengths of the two parents are often different, we first generate five logical points and translate them into "relatively" the same positions in the parents. Few parallel layers of offsprings made in this way are usually valid.[2] 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=0.03) in each layer and change one of the input buses at random. If

---

[2]Here, a "valid" layer means a layer that consists of independent comparators. This usage is different from the other parts, e.g., Section 2.1, of this paper.
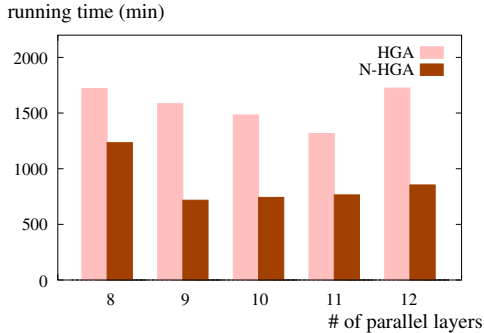
running time (min)

Figure 9: The average running times according to the number of parallel layers

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

- **Edit, Repair and Local Optimization:** As mentioned in Section 2, we perform the edit and repair processes to an offspring after mutation. Then we perform the local search heuristic in Section 4 to it.

- **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 [24], and showed successful results in [4].

6   Experimental Results

We evolved the population of networks using a genetic algorithm on an Intel Pentium III 866 MHz. We used a population size of 100, which is significantly smaller than Juillé's [16].

We denote by N-HGA the proposed hybrid genetic algorithm with normalization and by HGA the algorithm without the normalization. When we used 8 to 12 parallel layers, HGA found 60-comparator sorting networks in all of the 50 trials and N-HGA found 60-comparator sorting networks in all of the 100 trials. Figure 9 shows the average running times of the GAs according to the number of parallel layers. N-HGA outperformed HGA.

With 9 parallel layers, N-HGA showed the most stable performance. We conducted more experiments on the N-HGA with 9 parallel layers. Table 1 summarizes the experimental results and environments of Juillé's

Table 1: Comparison of experimental results and environments

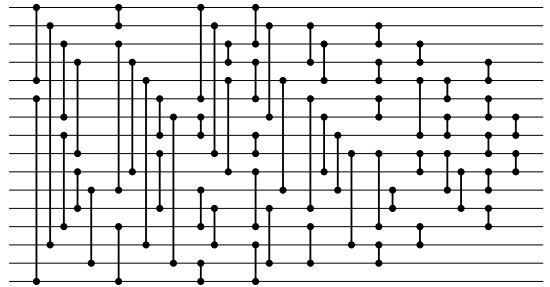|	END [16]	N-HGA
Population size	65,536	100
Machine	Maspar MP-2 (17,000 Mips)	Pentium III 866 MHz
# of processors	4,096	1
Results	60 comparators, 2 for 3 runs	60 comparators, 50 for 50 runs
Execution time	2880 to 4320 min	346 to 1334 min (average 743 min)

Figure 10: A 60-comparator sorting network that we found

END model [16] and the N-HGA. Juillé [16] reported that they found 60-comparator networks in two of the three runs. We found 60-comparator networks in all the 50 runs. They consumed 2 to 3 days with the Maspar MP-2 supercomputer; we consumed around half a day with a single-CPU PC. Overall, N-HGA was significantly faster and more stable than END. We present in Figure 10 one of the 60-comparator sorting networks that we found.

7   Conclusion

We defined the concept of sorting network isomorphism and examined the characteristics that isomorphic networks share. We took a graph-theoretical approach for the sorting network isomorphism by investigating its relationship with underlying graphs. We also devised the normalization technique on the basis of sorting network isomorphism and used it in the genetic algorithm, so that we could avoid more-than-necessary perturbation by crossover.

We developed an effective local search heuristic. The local search heuristic is another factor in the improvement of the performance. When the local optimization heuristic and the genetic algorithm were combined, they showed strong synergy. We found solutions with the best known quality in the 16-bus problem with a fairly small time budget. To the best of our knowledge, this is the first result that found 60-comparator sort-

ing networks without fixing any comparators under a (single-CPU) PC environment.

Although the suggested method performed impressively, we consider that there remains room for further improvement. We are currently working on the improvements for both of the local and genetic searches. We are also considering the investigation of general relationship between the isomorphism of phenotypes and the exploitation of genetic algorithms.

Acknowledgements

References

[1] K. E. Batcher. A new internal sorting method. *Goodyear Aerospace Report GER-11759*, 1964.

[2] R. K. Belew and T. Kammayer. Evolving aesthetic sorting networks using developmental grammars. In *Fifth International Conference on Genetic Algorithms*, page 629. Morgan Kauffmann, 1993.

[3] R. C. Bose and R. J. Nelson. A sorting problem. *Journal of ACM 9*, pages 282–296, 1962.

[4] T. N. Bui and B. R. Moon. Genetic algorithm and graph partitioning. *IEEE Trans. on Computers*, 45(7):841–855, 1996.

[5] D. Cavicchio. *Adaptive Search Using Simulated Evolution*. PhD thesis, University of Michigan, Ann Arbor, MI, 1970. Unpublished.

[6] S. S. Choi and B. R. Moon. A graph-based approach to the sorting network problem. In *Congress on Evolutionary Computation*, pages 457–464, 2001.

[7] S. S. Choi and B. R. Moon. A hybrid genetic search for the sorting network problem with evolving parallel layers. In *Genetic and Evolutionary Computation Conference*, pages 258–265, 2001.

[8] G. L. Drescher. Evolution of 16-number sorting networks revisited. Unpublished manuscript, 1994.

[9] C. Ebeling and O. Zaijicek. Validating VLSI circuit layout by wirelist comparison. In *Proc. of International Conference on Computer-Aided Design*, 1983.

[10] J. R. Koza et al. Evolving sorting networks using genetic programming and the rapidly reconfigurable Xilinx 6216 field-programmable gate array. In *31st Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 404–410, 1998.

[11] R. W. Floyd and D. E. Knuth. Improved constructions for the Bose-Nelson sorting problem. *Notices of the Amer. Math. Soc. 14*, page 283, 1967.

[12] M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.

[13] M. W. Green. Some improvements in nonadaptive sorting algorithms. Technical report, Stanford Research Institute, Menlo Park, California, 1969.

[14] W. D. Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. In C. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II*. Addison Wesley, 1995.

[15] G. E. Hinton and S. J. Nowlan. How learning can guide evolution. *Complex Systems*, 1(3):495–502, 1987.

[16] H. Juillé. Evolution of non-deterministic incremental algorithms as a new approach for search in state spaces. In *Sixth International Conference on Genetic Algorithms*, pages 351–358, 1995.

[17] D. E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison Wesley, 1973.

[18] J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity*. Birkhäuser, 1993.

[19] G. Laszewski. Intelligent structural operators for the k-way graph partitioning problem. In *Fourth International Conference on Genetic Algorithms*, pages 45–52, 1991.

[20] E. M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25:42–65, 1982.

[21] G. Shapiro. In D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley, 1973.

[22] G. G. Simpson. The Baldwin effect. *Evolution*, 7:110–117, 1953.

[23] D. Whitley, V. Gordon, and K. Mathias. Lamarckian evolution, the Baldwin effect and function optimization. In *International Conference on Evolutionary Computation*, pages 6–15, 1994.

[24] D. Whitley and J. Kauth. Genitor: A different genetic algorithm. In *Rocky Mountain Conf. on Artificial Intelligence*, pages 118–130, 1988.