More Effective Genetic Search for the Sorting Network Problem

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 [8], the authors proposed a hybrid genetic algorithm for the optimal sorting network problem. In this paper, we propose an approach to further improve this work. For this, we designed a novel data structure to efficiently process the dominating operation in solving the problem. We also developed an effective local search heuristic based on a fast approximate measure. Combining it with genetic operators, we obtained the most stable results so far.

1 Introduction

A sorting network is composed of buses and a number of homogeneous comparators. Each comparator c(a,b) performs the elementary operation that compares the a^{th} and b^{th} buses; if their values are in order, they pass the comparator straight, otherwise they are exchanged. We call a sorting network with n inputs an n-bus sorting network. For any input sequence of an n-bus sorting network, the output sequence is monotonically non-decreasing ($y_0 \leq y_1 \leq \ldots \leq y_{n-1}$). Figure 1 shows a 4-bus sorting network with five comparators: [c(0,1), c(2,3), c(0,2), c(1,3), c(1,2)].

Usually, there are some comparators that can run simultaneously. In Figure 1, the first and second comparators can run simultaneously since they are independent. Likewise, the third and fourth comparators also can run simultaneously. Thus, the sorting can be completed in just three parallel steps.

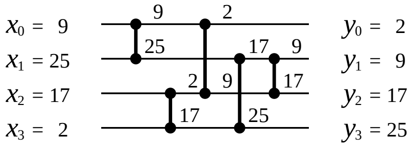


Figure 1: A 4-bus sorting network

There have been many studies of sorting networks because of their applications and rich underlying theory [17] [20] [18]. The research has two main aims [17]: to reduce the number of comparators or to reduce the number of parallel steps. We focus on minimizing the number of comparators following the convention [13] [9] [15].

This paper pursues improvement upon the authors' work on the sorting network problem [8]. We focus on the 16-bus sorting network problem. And we fix the first 32 comparators as in [8]. Historically most studies included 16-bus sorting networks [4] [11] [2] [21] [12]. In 1969, Green [12] first discovered a 60-comparator 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 [13] [3] [9] [15] [10] [8].

Most practical studies of 16-bus sorting networks used supercomputers because of the huge computational needs [13] [9] [15]. In [7] and [8], the authors first found 60-comparator sorting networks on a single-CPU PC in a few minutes. In this study, we propose an approach to further improve this result. For this, we first designed a novel data structure in order to efficiently process the time-dominating operation in solving the problem. We also devised a fast approximate measure that evaluates the potential quality of a given network. Then we developed an effective local search heuristic based on this measure. Finally, we combined it with genetic operators to lead a strong synergy.

The rest of this paper is organized as follows. In Section 2, we present our previous works related to the sorting network problem. In Section 3, we describe the novel data structure and our local optimization heuristic, and in Section 4 we provide a genetic algorithm combined with the local optimization heuristic. We give the experimental results in Section 5. Finally, we summarize the study in Section 6.

2 Preliminaries

In this section, we briefly review our previous works in [7] and [8] for further discussions.

2.1 Zero-One Principle and Networks as Functions

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 sorts any arbitrary sequence of n numbers into nondecreasing order.

The Zero-One Principle 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 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.

2.2 O/N-pairs and Parallel Layers

Let $t(i)$ be the i^{th} bit value of a binary sequence t. For two input bus indices x and y such that $x < y$ $(x, y = 0, 1, 2, ..., n-1)$, 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 has 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 to the optimality 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 [8].

2.3 Edit and Repair Heuristics

In [7] and [8], 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 [7] and [8], the number of the 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 [8] for details.

3 The Proposed Approach

3.1 Performance Improvement Using Function Value Tables

Rabin showed that the validity check problem of a given sorting network is co-NP-complete; it is considered that the problem is intrinsically difficult [17]. If we know the number of the n-pairs in a network, we can decide whether the network is valid or not. So it is also intrinsically difficult to find the number of the n-pairs.

Thus, it occupied most of the running time, in [8], for the edit and repair since they use the number of n-pairs as a measure of evaluating a network. They examined the output sequences obtained by feeding binary sequences into a network in order to get the number of n-pairs simultaneously with o/n-pairs, which are used in the edit and repair heuristics. Thus the comparison operation for the comparators dominated the edit and repair processes.

The local optimization algorithm described in the next section evaluates a network on the basis of the length of the edited network and the number of n-pairs.

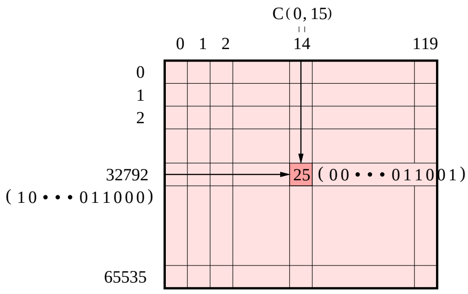


Figure 2: A function value table

The comparison operation still dominates the running time. Hence, the efficiency of the suggested approach essentially depends on how efficiently we process the comparison operation.

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 $\mathcal{T}_n$. We denote by $|\chi|$ the number of comparators in a sorting network χ. Suppose that an n-bus network χ consists of the comparators $c_0, c_1, \ldots, c_{|\chi|-1}$. If we denote by f_χ the function corresponding to χ and by f_{c_i} the function corresponding to c_i $(0 \le i \le |\chi| - 1)$, then, for any $t \in \mathcal{T}_n$, it holds that

$$\begin{aligned} f_\chi(t) &= (f_{c_{|\chi|-1}} \circ f_{c_{|\chi|-2}} \circ \cdots \circ f_{c_1} \circ f_{c_0})(t) \\ &= f_{c_{|\chi|-1}}(\, f_{c_{|\chi|-2}}\, (\cdots f_{c_1}(f_{c_0}(t)) \cdots))\ . \end{aligned}$$

From the above equation, if we know the images of all binary sequences under f_{c_i} for all 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 j^{th} comparator to the i^{th} binary sequence. In the case of the 16-bus network problem, the memory space required to create such a table is $\binom{16}{2} \times 2^{16} \times 2 = 120 \times 65536 \times 2$ bytes $<$ 16 Mbytes. Figure 2 shows the matrix for the 16-bus problem. In addition to this, we generate a table that, for each $t \in \mathcal{T}_n$, stores the pairs in t that are not in order. By using these tables, we could considerably reduce the computational time for the comparison operations.[1]

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

3.2 A Local Search Heuristic

3.2.1 An Approximate Measure for Network Quality

We mentioned before that we had devised edit and repair heuristics to enhance the GA's fine-tuning around local optima in [7] and [8]. 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. In general, the perturbation by crossover and mutation considerably lowers the qualities of offspring networks. In this case, it is difficult to recover the qualities of networks only by the edit and the *greedy* and *constructive* repair heuristic. In this study, we devised a local search heuristic to alleviate this problem.

Let χ' be a neighbor of a network χ in the problem space. The eventual quality of χ' can be the quality of the network obtained by performing the edit and repair to χ'. Due to the tablization of the previous section, we can considerably reduce the time for edit and repair. Nevertheless, the repair is still a heavy-computing operation.

For a sorting network χ, let $E(\chi)$ and $R(\chi)$ be the networks obtained by performing edit and repair heuristics to χ, respectively. We get the following fact from Fact 2 in [7]:

Fact 1 *For a sorting network* χ,

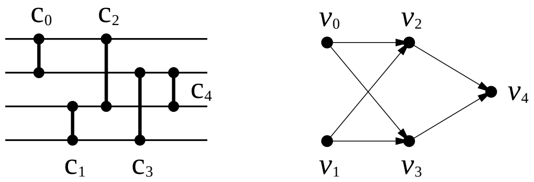
$$|R(E(\chi))| \le |E(\chi)| + |NP(\chi)|\ .$$

Proof: Omitted. ■

We denote by $\widehat{|\chi|}$ the value of $|E(\chi)| + |NP(\chi)|$ and call $\widehat{|\chi|}$ as the *potential-length upper bound* of χ. From the above fact, $\widehat{|\chi|}$ is an upper bound of the length of the network obtained by performing the edit and repair heuristics to χ. In other words, the quality of χ is bounded by $\widehat{|\chi|}$.

For a neighbor network χ' of a valid network χ, it is clear that $\widehat{|\chi'|} \le |\chi|$ implies $|R(E(\chi'))| \le |\chi|$. This means that, if the potential-length upper bound of χ' is not greater than the length of χ, the potential quality of χ' is not worse than the quality of χ. Therefore, we can replace χ with χ' and improve the network quality.

Furthermore, we approximate the quality of any network χ obtained in the process of local search by $\widehat{|\chi|}$. It is, of course, not true that $\widehat{|\chi|} \le \widehat{|\chi'|}$ always implies $|R(E(\chi))| \le |R(E(\chi'))|$. However, experiments showed that there is a strong correlation between $\widehat{|\chi|}$



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

Figure 3: An example sorting network and its underlying graph

and $|R(E(\chi))|$. When a tie occurs, we give favor to the networks with shorter edited lengths.

3.2.2 Underlying Graphs of Sorting Networks

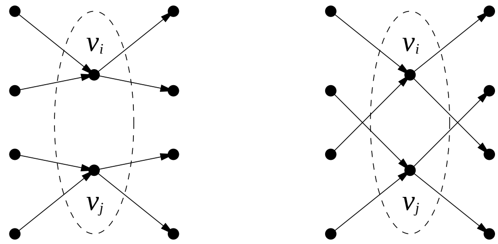
Sorting networks have been generally represented in such a way as in Figure 1. Since it draws buses across comparators in order to focus on the data flows on buses, it is not appropriate for representing a physical situation involving discrete comparators and the relationship among them. We devised a more intuitive model to represent the relationship among comparators.

We set a vertex for each comparator. If an output of comparator c_i is fed into comparator c_j, we put an arc e_{ij} from the vertex v_i to the vertex v_j. We have a graph $G = (V, E)$ corresponding to a sorting network where V is the set of vertices and E is the set of directed edges. For a given network χ, we call such a graph an *underlying graph* and denote by $G(\chi)$. Figure 3 shows an example sorting network and its underlying graph.

3.2.3 Search Strategy

From the viewpoint of underlying graphs, the optimal sorting network problem can be considered to be the problem of finding the edges that optimally connects the vertices corresponding to given comparators — the problem of finding the optimal network topology. At first glance, it seems to be reasonable to do local search by creating and deleting arcs between two random vertices. However, for a network χ' obtained from a network χ in this manner, we have to consider all the binary sequences in the test set in order to get the value of $\widehat{|\chi'|}$ in most cases. This makes the local search intractable.

For this reason, we consider the networks, obtained by exchanging every two input bus indices of comparators in each layer of 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 $G(\chi)$. Figure 4 shows an example. Such a strategy considerably reduces the computational load of local search for the following reason.



(a) A sorting network χ (b) A neighbor of χ

Figure 4: An illustration of exchanging process

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,

$$|f_{\chi_0}(f_{\phi_{32}}(\mathcal{T}_n)) - \mathcal{S}_n| \gg |f_{\chi_1}(f_{\chi_0}(f_{\phi_{32}}(\mathcal{T}_n))) - \mathcal{S}_n| \gg \cdots \gg |f_{\chi_{l-1}}(f_{\chi_{l-2}} \cdots f_{\chi_0}(f_{\phi_{32}}(\mathcal{T}_n)) \cdots) - \mathcal{S}_n| \,.$$

Therefore, in evaluating the qualities of neighbors, when we exchange two input bus indices for the comparators in the i^{th} layer of χ, we can considerably reduce the time of editing the neighbors and getting the numbers of n-pairs by considering only the sequences in $f_{\chi_{i-1}}(f_{\chi_{i-2}} \cdots f_{\chi_0}(f_{\phi_{32}}(\mathcal{T}_n)) \cdots) - \mathcal{S}_n$ instead of all the sequences in $f_{\phi_{32}}(\mathcal{T}_n) - \mathcal{S}_n$. Such a strategy, which successively improves layers left to right, is natural in that a parallel layer strongly affects the subsequent layers [8].

In each layer, we improve the layer by exchanging input bus indices of comparators in the manner of sequential 2-opt [16] [19] [1]. Let n be the number of input bus indices and l the number of parallel layers of a network χ. And let χ_i be the i^{th} layer of χ $(0 \le i \le l-1)$. We denote by $exchange(\phi, a, b)$ the layer obtained by exchanging the a^{th} and b^{th} input bus indices of comparators in a parallel layer ϕ and by χ_{ϕ/χ_i} the network obtained by replacing a layer χ_i in χ with another layer ϕ. Figure 5 describes our local optimization algorithm.

4 GA Framework

We used a hybrid steady-state genetic algorithm. Figure 6 shows the outline of the hybrid genetic algorithm.

LocalOptimize(χ)
{
  for $i \leftarrow 0$ **to** $l-1$ {
    $Q \leftarrow \emptyset$;
    $\chi_i^0 \leftarrow \chi_i$;
    for $j \leftarrow 1$ **to** $n/2$ {
      choose $a, b \in \{0, 1, \ldots, n-1\} - Q$
        such that $\phi_j = exchange(\chi_i^{j-1}, a, b)$
          and $|\chi_{\widehat{\phi_j/\chi_i}}|$ is maximal;
      $Q \leftarrow Q \cup \{a, b\}$;
      $\chi_i^j \leftarrow exchange(\chi_i^{j-1}, a, b)$;
    }
    choose $k \in \{1, 2, \ldots, n/2\}$
      such that $|\chi_{\widehat{\chi_i^k/\chi_i}}|$ is maximal;
    $\chi \leftarrow \chi_{\widehat{\chi_i^k/\chi_i}}$;
  }
  $\chi \leftarrow R(E(\chi))$;
  return χ;
}

Figure 5: The outline of the local optimization

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

Figure 6: The outline of the hybrid genetic algorithm

The details are described in the following.

- **Encoding:** Each sorting network is represented by a chromosome. Figure 7 shows the structure of a chromosome. A chromosome is composed of a fixed number of parallel layers and one supplemental layer. Each gene is represented by an integer corresponding to a comparator as in the function value table. 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] [14] [23].

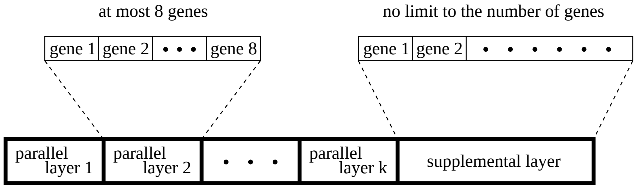


Figure 7: The structure of a chromosome in the 16-bus case

- **Initialization:** We set the population size to be 50. 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 a half the number of input buses. We then perform the edit and repair processes to make the chromosome valid.

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

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

  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.

- **Crossover:** As mentioned, we consider only the parallel layers of the two parents in crossover. We perform one-point crossover with each layer independent of the other layers. Since the numbers of genes in the two parents are usually not the same, the traditional one-point crossover is not possible. Thus, we generate a cut point on the "relatively" similar position 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=0.07) in each layer and change one of the input buses at random. If there exists a comparator, say c, in the same layer

[2]Here, a "valid" layer means a layer that consists of independent comparators. This usage is different from the other parts, e.g., Section 2.3, of this paper.

Table 1: Comparison of Experimental Results and Environments

	Hillis [13]	Drescher [9]	Juillé [15]†	Choi & Moon [8]	This Study
Population size	65,536	524,288	4,096	50	50
Machine	CM-1	CM-5	Maspar MP-2 (17,000 Mips)	Pentium III 866 MHz	Pentium III 866 MHz
# of processors	65,536	64	4,096	1	1
Results	61 comparators	60 comparators, 100 % for 10 runs	60 comparators, almost 100 %	60 comparators, 100 % for 100 runs	60 comparators, 100 % for 100 runs
Execution time	5 to 50 min	5 to 18 min	5 to 10 min	2 to 15 min (average 5 min)	32 to 144 sec (average 72.5 sec)

†The version where the first 32 comparators are fixed

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 of Section 3 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 [6] and GENITOR-style replacement [24], and showed successful results in [5].

The genes in the parallel layers are updated as a result of local optimization while those in the supplemental layer are appended only for fitness evaluation. This is a combination of Lamarckian and Baldwinian GAs. This model can reduce the computational load of the Lamarckian side by ignoring some of the last comparators in the evolution.

5 Experimental Results

With the first 32-comparators of networks fixed, we evolved the population of networks using a genetic algorithm on an Intel Pentium III 866 MHz. We used a population size of 50, which is significantly smaller than other studies [13] [9] [15].

The chart in Figure 8 shows the performances of GAs according to the number of parallel layers used for the GAs. Each bar indicates the number of 60-comparator sorting networks (of quality equivalent to the best known) that the corresponding GA found in 144 seconds in 100 trials. When we used four parallels, the GA found 60-comparator networks in all of the 100 trials. On the other hand, when we used one parallel layer, we could not find any 60-comparator sorting network. The performance of the GA showed a bitonic distribution with respect to the number of parallel layers.

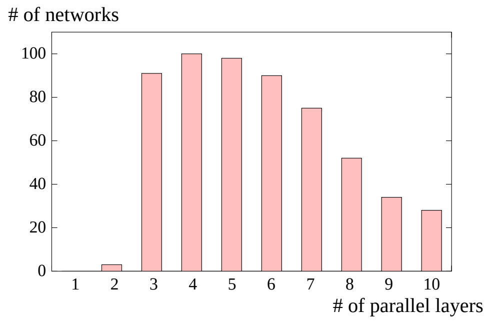


Figure 8: The number of 60-comparator sorting networks that we found in 144 seconds in 100 trials, according to the number of parallel layers

Table 1 summarizes the experimental results and the environments of Hillis [13], Drescher [9], Juillé [15], our previous study [8], and this study. This work showed the most stable performance in significantly less time even after considering the difference of CPU powers.

When we examined the 100 networks with 60 comparators that we found using 4 parallel layers, there were 72 distinct sorting networks.[3] Table 2 classifies the 72 sorting networks according to the number of parallel steps. We present in Figure 9 one of the 60-comparator sorting networks (with 10 parallel steps) that we found.

[3]We ignored the sequences in the same parallel step.

Table 2: The number of distinct sorting networks according to the number of parallel steps

# of parallel steps	10	11	12	Total
# of sorting networks	47	4	21	72

6 Conclusion

We paid attention to the relative importance of the comparison operations in the process of solving the sorting network problem. Experiments showed that these operations have a great effect on the performance. We designed a novel data structure in order to process the operations efficiently, and this led to the remarkable performance improvement.

To the best of our knowledge, no effective local search algorithm for the optimal sorting network problem has been proposed yet. We realized that this was because there was no efficient measure for evaluating the qualities of networks. We devised an approximate measure that not only reflects the qualities of networks sufficiently but also needs small computational time.

We also developed an effective local search heuristic based on this measure. In particular, we maximized the efficiency of the local search heuristic by reflecting the characteristics of the problem. Such an efficient and effective local search heuristic came to be another factor in the improvement of the performance.

A 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 a Lamarckian and a Baldwinian GA. It turned out 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 significantly reduces the problem search space and helps the GA to conduct an efficient search.

When the local optimization heuristic and the genetic algorithm were combined, they showed strong synergy. We found the best known solutions in 16-bus networks with a fairly small time budget. To the best of our knowledge, this is the most efficient result in the discovery of 60-comparator sorting networks.

We restricted the problem space by fixing the first 32 comparators. 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.

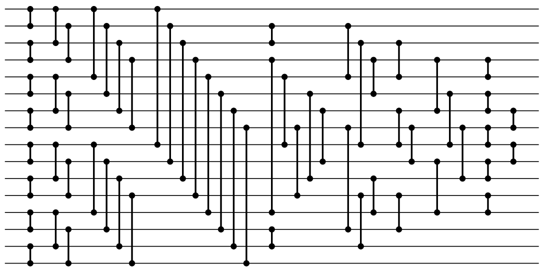


Figure 9: A 60-comparator sorting network that we found

Acknowledgements

This work was supported by KOSEF through the Statistical Research Center for Complex Systems at Seoul National University (SNU) and Brain Korea 21 Project. The RIACT at SNU provided research facilities for this study.

References

[1] E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization.* John Wiley & Sons, 1997.

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

[3] 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.

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

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

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

[7] 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.

[8] 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.

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

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

[13] 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.

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

[15] 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.

[16] B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49:291–307, 1970.

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

[18] W. Li, J. Lin, and R. Unbehauen. On the analysis of sorting networks from the viewpoint of circuit theory. *IEEE Trans. on Circuits and Systems I — Fundamental Theory and Applications*, 45(5):591–593, 1998.

[19] S. Lin and B. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21(4598):498–516, 1973.

[20] I. Pitas and A. N. Venetsanopoulos. A new filter structure for the implementation of certain classes of image processing operations. *IEEE Trans. on Circuits and Systems*, 35(6):636–647, 1988.

[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.