New Bounds on Optimal Sorting Networks

2 authors, including:

Thorsten Ehlers
German Aerospace Center (DLR)
24 PUBLICATIONS   **118** CITATIONS

Some of the authors of this publication are also working on these related projects:

Computing Treewidth in Practice View project

New Bounds on Optimal Sorting Networks

Thorsten Ehlers and Mike Müller

Institut für Informatik, Christian-Albrechts-Universität zu Kiel
D-24098 Kiel, Germany.
{the,mimu}@informatik.uni-kiel.de

Abstract. We present new parallel sorting networks for 17 to 20 inputs. For $17, 19$, and 20 inputs these new networks are faster (i.e., they require less computation steps) than the previously known best networks. Therefore, we improve upon the known upper bounds for minimal depth sorting networks on $17, 19$, and 20 channels. Furthermore, we show that our sorting network for 17 inputs is optimal in the sense that no sorting network using less layers exists. This solves the main open problem of [D. Bundala & J. Závodný. Optimal sorting networks, Proc. LATA 2014].

1 Introduction

Comparator networks are hardwired circuits consisting of simple gates that sort their inputs. If the output of such a network is sorted for all possible inputs, it is called a *sorting network*. Sorting networks are an old area of interest, and results concerning their size date back at least to the 50's of the last century.

The size of a comparator network in general can be measured by two different quantities: the total number of comparators involved in the network, or the number of layers the networks consists of. In both cases, finding optimal sorting networks (i.e., of minimal size) is a challenging task even when restricted to few inputs, which was attacked using different methods.

For instance, Valsalam and Miikkulainen [11] employed evolutionary algorithms to generate sorting networks with few comparators. Minimal depth sorting networks for up to 16 inputs were constructed by Shapiro and Van Voorhis in the 60's and 70's, and by Schwiebert in 2001, who also made use of evolutionary techniques. For a presentation of these networks see Knuth [8, Fig.51]. However, the optimality of the known networks for 11 to 16 channels was only shown recently by Bundala and Závodný [4], who partitioned the set of first two layers into equivalence classes and reduced the search to extensions of one representative of each class. They then expressed the existence of a sorting network with less layers extending these representatives in propositional logic and used a SAT solver to show that the resulting formulae are unsatisfiable. Codish, Cruz-Filipe, and Schneider-Kamp [5] simplified the generation of the representatives and independently verified Bundala and Závodný's result.

For more than 16 channels, not much is known about the minimal depths of sorting networks. Al-Haj Baddar and Batcher [2] exhibit a network sorting 18 inputs using 11 layers, which also provides the best known upper bound on the

minimal depth of a sorting network for 17 inputs. The lowest upper bound on the size of minimal depth sorting networks on 19 to 22 channels also stems from a network presented by Al-Haj Baddar and Batcher [1]. For 23 and more inputs, the best upper bounds to date are established by merging the outputs of smaller sorting networks with Batcher's odd-even merge [3], which needs $\lceil \log n \rceil$ layers for this merging step.

The known lower bounds are due to Parberry [10] and Bundala and Závodný. A new insight by Codish, Cruz-Filipe, and Schneider-Kamp [6] into the structure of the last layers of sorting networks lead to a significant further reduction of the search space. Despite all this recently resparked interest in sorting networks, the newly gained insights were insufficient to establish a tight lower bound on the depth of sorting networks for 17 inputs.

We use the SAT approach by Bundala and Závodný to synthesize new sorting networks of small depths, and thus provide better upper bounds for $17, 19$, and 20 inputs. Furthermore, our improvements upon their method allow us to raise the lower bound for 17 inputs. Therefore, for the first time after the works of Shapiro, Van Voorhis, and Schwiebert, we present here a new optimal depth sorting network.

An overview of the old and new upper and lower bounds for the minimal depth of sorting networks for up to 20 inputs is presented in Table 1.

Table 1. Bounds on the minimal depth of sorting networks for up to 20 inputs.

Number of inputs	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Old upper bound	0	1	3	3	5	5	6	6	7	7	8	8	9	9	9	9	11	11	12	12
New upper bound	0	1	3	3	5	5	6	6	7	7	8	8	9	9	9	9	**10**	11	**11**	**11**
Old lower bound	0	1	3	3	5	5	6	6	7	7	8	8	9	9	9	9	9	9	9	9
New lower bound	0	1	3	3	5	5	6	6	7	7	8	8	9	9	9	9	**10**	**10**	**10**	**10**

2 Preliminaries

A *comparator* is a gate with two inputs in_1 and in_2 and two outputs $out_{\min}$ and $out_{\max}$, that compares, and if necessary rearranges its inputs such that $out_{\min} = \min\{in_1, in_2\}$ and $out_{\max} = \max\{in_1, in_2\}$. Combining zero or more comparators in a network yields a *comparator network*. Comparator networks are usually visualized in a graphical manner in a so-called *Knuth diagram* as depicted in **??**. Here a comparator connecting two channels is drawn as ⌶ , where by convention the upper output is $out_{\min}$ and the lower output is $out_{\max}$. A maximal set of comparators with respect to inclusion that can perform parallel comparisons in a comparator network is called a *layer*. The number of layers of a network is called the *depth* of the network. A useful tool to verify that a network is a sorting network is the "0-1-principle" [8], which states that a comparator network is a sorting network, if and only if it sorts all binary inputs.

For more details about sorting networks we refer to Knuth [8, Sect. 5.3.4].
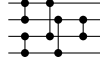
Fig. 1. A comparator network of depth 3 with 5 comparators

3 Improved Techniques

In this section we introduce the new techniques and improvements on existing techniques we used to gain our results. We will stick to the formulation by Bundala and Závodný [4], and introduce new variables if necessary. Furthermore, we will extend a technique introduced in their paper, called *subnetwork optimization*. It is based on the fact that a sorting network must sort all its inputs, but in order to prove non-existence of sorting networks of a certain depth, it is often sufficient to consider only a subset of all possible inputs, which are not all sorted by any network of this restricted depth. Bundala and Závodný chose subsets of the form $T^r = \left\{ 0^a x 1^b \mid |x| = r \text{ and } a + b + |x| = n \right\}$ for $r < n$, which are inputs having a *window* of size r. For an input $0^a x 1^b$ from this set the values on the first a channels at any point in the network will always be 0, and those on the last b channels will always be 1, which significantly reduces the encoding size for these inputs if a and b are sufficiently large.

3.1 Prefix optimization

It is a well-known fact that permuting the channels of a sorting network, followed by a repair-procedure called untangling yields another feasible sorting network [10]. In fact, Parberry [10] used a first layer with comparators of the form $(2i - 1, 2i), 1 \leq i \leq \lfloor \frac{n}{2} \rfloor$, which we will call *Pb-style* whereas Bundala and Závodný used comparators $(i, n + 1 - i), 1 \leq i \leq \lfloor \frac{n}{2} \rfloor$ in the first layer, which we call *BZ-style*; see **??**. Both versions are equivalent in the sense that if there exists a

Fig. 2. Pb-style first layer (left), and BZ-style (right) for 6 channels

sorting network C_n^d, then there exist sorting networks of the same depth with both of these prefixes. Nevertheless, for creating networks obeying a certain prefix as well as proving their non-existence, a given prefix may be hard-coded into the SAT formula. Given a prefix P of depth $|P|$, the remaining SAT formula encodes the proposition *"There is a comparator network on n channels of depth $d - |P|$ which sorts all outputs of P"*. Interestingly, the outputs of different prefix styles are not equally handy for the SAT encoding. A Pb-style first layer performs compare-and-swap operations between adjacent channels, thus the presorting performed here is more local than the one done by BZ-style first layers. Let

$out(P)$ denote the set of outputs of a prefix P on n layers. Then, the number of channels that actually must be considered in the SAT formula is given by

$$\sum_{x \in out(P)} \left(n - \max\{a \mid x = 0^a x'\} - \max\{b \mid x = x'1^b\} \right),$$

i.e., the sum of window-sizes of all outputs of P.

Table 2 shows the impact of these previous deliberations when using a 1-layer-prefix for $2 \leq n \leq 17$ channels.

Table 2. Number of channels to consider in the encoding after the first layer

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Pb-style	0	5	12	44	84	233	408	1016	1704	4013	6564	14948	24060	53585	85296	186992
BZ-style	0	4	10	36	72	196	358	876	1524	3532	5962	13380	22128	48628	79246	171612

Table 3. Impact of prefix style when proving that no sorting network for 16 channels with at most 8 layers exists.

Prefix style	Overall time (s)	Maximum time (s)
Pb	22,241	326
BZ	10,927	150
Opt	5,492	36

Table 3 shows running times when proving that no sorting network on 16 channels with 8 layers exists. In this case, we used 2-layer-prefixes according to [5], and proved unfeasibility for each of the 211 distinct prefixes. In the first case, they were permuted and untangled such that the first layer is in Pb-style, whereas the second case has BZ-style first layers. In the third case, we used an evolutionary algorithm to find a prefix such that the number of channels to consider when using 800 distinct outputs of the respective prefix was minimized. As we have $d - |P|$ Boolean variables for each channel which cannot be hard-coded, this procedure minimizes the number of variables in the resulting SAT formula.

This technique reduced the overall running time by factors of 4.05 and 1.99, and the maximum running times by 9.0 and 4.1, respectively.

3.2 Iterative encoding

As mentioned above, it is usually not necessary to use all 2^n input vectors to prove lower bounds. In order to take advantage of this fact, we implemented an iterative approach. We start with a formula which describes a feasible comparator

network and a (potentially empty) set of initial inputs, and iteratively add inputs until either a feasible sorting network has been found, or no network can be found which is able to sort the given set of inputs, as depicted in figure 3. During the iterative process, counter-examples (i.e., inputs that are not sorted by the network created so far) of minimal window-size are chosen.
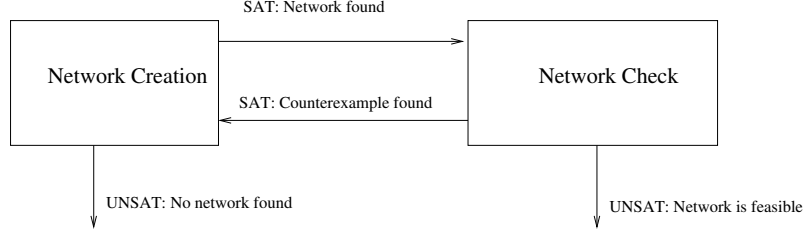

Fig. 3. Iterative generation of new inputs

Using this technique, we tested the impact of different prefix style on both running time, and the number of inputs required. Table 4 shows results for one

Table 4. Impact of prefix style on running time and number of iterations

Initial Inputs		0	100	200	300	400	500	600	700	800	900
Pb	Time	157	139	128	86	61	56	45	52	54	59
	Iterations	264	174	72	4	1	1	1	1	1	1
BZ	Time	88	75	64	47	28	14	14	13	13	19
	Iterations	358	259	165	66	10	1	1	1	1	1

2-layer-prefix for 16 channels, used to prove that this cannot be extended to a sorting network with 8 layers. Here, more inputs are required to prove that a BZ-style prefix cannot be extended when compared to a Pb-style prefix. Nevertheless, BZ-style prefixes are superior in terms of running time. Interestingly, the process becomes faster when more inputs than actually required were chosen, this is, using slightly more inputs is beneficial.

Next, we turn to improve the SAT encoding.

3.3 Improved SAT encoding

We modified the SAT encoding of Bundala and Závodný [4], significantly reducing the number of clauses. A variable $g_{i,j}^k$, with $i < j$, encodes the fact that there is a comparator comparing channels i and j in layer k in the network. Furthermore, the variable v_i^k stores the value on channel i after layer k. For completeness' sake

we list the original encoding completely:

$$once_i^k(C_n^d) = \bigwedge_{1 \leq i \neq j \neq l \leq n} \left(\neg g_{\min(i,j),\max(i,j)}^k \vee \neg g_{\min(i,l),\max(i,l)}^k \right)$$

$$valid(C_n^d) = \bigwedge_{1 \leq k \leq d, 1 \leq 1 \leq n} once_i^k(C_n^d)$$

$$used_i^k(C_n^d) = \bigvee_{j < i} g_{j,i}^k \vee \bigvee_{i < j} g_{i,j}^k$$

$$update_i^k(C_n^d) = \left(\neg used_i^k(C_n^d) \to (v_i^k \leftrightarrow v_i^{k-1}) \right) \wedge$$
$$\bigwedge_{1 \leq j < i} \left(g_{j,i}^k \to \left(v_i^k \leftrightarrow (v_j^{k-1} \vee v_i^{k-1}) \right) \right) \wedge$$
$$\bigwedge_{i < j \leq n} \left(g_{i,j}^k \to \left(v_i^k \leftrightarrow (v_j^{k-1} \wedge v_i^{k-1}) \right) \right)$$

Here, *once* encodes the fact that each channel may be used only once in one layer, and *valid* encodes this constraint for each channel and each layer. The *update*-formula describes the impact of comparators on the values stored on each channel after every layer.

$$sorts(C_n^d, x) = \bigwedge_{1 \leq i \leq n} (v_i^0 \leftrightarrow x_i) \wedge \bigwedge_{1 \leq k \leq d, 1 \leq i \leq n} update_i^k(C_n^d) \wedge \bigwedge_{1 \leq i \leq n} (v_i^d \leftrightarrow y_i)$$

The constraint *sorts* encodes if a certain input is sorted by the network C_n^d. For this purpose, the values after layer d (i.e., the outputs of the network) are compared to the vector y, which is a sorted copy of the input x. A sorting network for n channels on d layers exists iff $valid(C_n^d) \wedge \bigwedge_{x \in \{0,1\}^n} sorts(C_n^d, x)$ is satisfiable.

Consider an input $x = 0^a x' 1^b$, and a comparator $g_{i,j}^k$ with $i \leq a$. This is, we have $v_i^k \leftrightarrow 0 \wedge v_j^{k-1} \equiv 0$, and $v_j^k \leftrightarrow 0 \vee v_j^{k-1} \equiv v_j^{k-1}$. As the same holds for $j > n - b$, we have that comparators "leaving" a subnetwork need not be considered for sorting the respective inputs. Furthermore, if $v_i^{k-1} \leftrightarrow 1$ for some k and i, using any comparator $g_{j,i}^k$ will cause $v_i^k \leftrightarrow 1$. Thus, for every channel i we introduce $oneDown_{i,j}^k$- and $oneUp_{i,j}^k$-variables which indicate whether there is a comparator $g_{l,j}^k$ for some $i \leq l < j$ or $g_{i,l}^k$ for some $i < l \leq j$, respectively.

$$oneDown_{i,j}^k \leftrightarrow \bigvee_{i < l \leq j} g_{i,l}^k \qquad\qquad noneDown_{i,j}^k \leftrightarrow \neg oneDown_{i,j}^k$$

$$oneUp_{i,j}^k \leftrightarrow \bigvee_{i \leq l < j} g_{l,j}^k \qquad\qquad noneUp_{i,j}^k \leftrightarrow \neg oneUp_{i,j}^k$$

To make use of these new predicates, given an input $x = 0^a x' 1^b$, for all $a < i \leq n - b$ we add

$$v_i^{k-1} \wedge noneDown_{i,n-b}^k \to v_i^k$$
$$\neg v_i^{k-1} \wedge noneUp_{a+1,i}^k \to \neg v_i^k,$$

to the formula and remove all update-constraints that are covered by these constraints. This offers several advantages: Firstly, it reduces the size of the resulting formula in terms of both the number of clauses, and the overall number of literals. Secondly, this encoding allows for more propagations, thus, conflicts can be found earlier. Thirdly, it offers a more general perspective on the reasons of a conflict. Table 5 shows the impact of both the new encoding and permuting the prefix, which results in an average speed-up of 8.2, and a speed-up of 17.1 for the hardest prefixes.

Table 5. Results for different settings when proving the non-existence of 8-layer sorting-networks for 16 channels

Encoding	Prefix-Style	Overall time (s)	Max. time (s)	Variables	Clauses	Literals
Old	Pb	22,241	326	108,802	4,467,201	13,977,393
	BZ	10,927	150	99,850	3,996,902	12,442,522
	Opt	5,492	36	84,028	3,183,363	9,879,588
New	Pb	11,766	196	110,398	2,443,186	8,108,501
	BZ	4,359	54	101,404	2,049,744	6,799,486
	Opt	2,702	19	85,652	1,504,177	4,981,882

4   Obtaining new lower bounds

In a first test, we tried to prove that there is no sorting network on 17 channels using at most 9 layers by using a SAT encoding which was almost identical to the one introduced in [4], enriched by constraints on the last layers [6]. Before we broke up this experiment after 48 days, we were able to prove that 381 out of 609 prefixes cannot be extended to sorting networks of depth 9. Showing unsatisfiability of these formulae took $353 \cdot 10^6$ seconds of CPU time, with a maximum of $3 \cdot 10^6$ seconds.

In a new attempt, we used a modified encoding as described in the previous section. For every equivalence class of 2-layer-prefixes, we chose a representative which minimizes the number of variables in the SAT encoding when using $2,000$ distinct inputs. This time, we were able to prove that none of the prefixes can be extend to a sorting network using 9 layers. The overall CPU time for all 609 equivalence classes was $27.63 \cdot 10^6$ seconds with a maximum running time of $97,112$ seconds. This is a speed up of at least 42.7 concerning the maximum running time, and 20.4 for the average running time. Since the result for all 2-layer-prefixes was unsat, we conclude:

Theorem 1 *Any sorting network for $n \geq 17$ channels has at least* 10 *layers.*

5 Finding faster networks

Even though SAT encodings for sorting networks as well as SAT solvers themselves have become much better within the last years, generating new, large sorting networks from scratch is still out of their scope. Hence, we extended ideas by Al-Haj Baddar and Batcher [2].

5.1 Using hand-crafted prefixes

A well-known technique for the creation of sorting networks is the generation of partially ordered sets for parts of the input in the first layers. **??** shows comparator networks which create partially ordered sets for 2, 4 and 8 input bits. In the case of $n = 2$, the output will always be sorted. For $n = 4$ bits, the set of
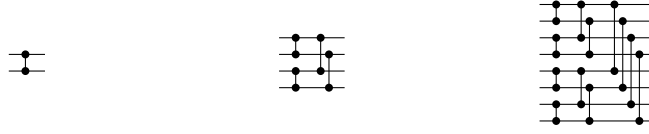

Fig. 4. Generating partially ordered sets for $n \in \{2, 4, 8\}$ inputs.

possible output vectors is given by

$$\left\{ \begin{pmatrix} 0\ 0\ 0\ 0 \end{pmatrix}^T, \begin{pmatrix} 0\ 0\ 0\ 1 \end{pmatrix}^T, \begin{pmatrix} 0\ 0\ 1\ 1 \end{pmatrix}^T, \begin{pmatrix} 0\ 1\ 0\ 1 \end{pmatrix}^T, \begin{pmatrix} 0\ 1\ 1\ 1 \end{pmatrix}^T, \begin{pmatrix} 1\ 1\ 1\ 1 \end{pmatrix}^T \right\},$$

i.e., there are 6 possible outputs. Furthermore, the first output bit will equal zero unless all input bits are set to one, and the last output bit will always be set to one unless all input bits equal zero. Similarly, the network for $n = 8$ inputs allows for 20 different output vectors. Prefixes of sorting networks which consist of such snippets are referred to as *Green filters* [7].

5.2 Results

We present two sorting networks improving upon the known upper bounds on the minimal depth of sorting networks. The networks presented in **??** are sorting networks for 17 channels using only 10 layers, which outperform the currently best known network due to Al-Haj Baddar and Batcher [2]. The first three layers of the network on the left are a Green filter on the first 16 channels, the remainder of this network was created by a SAT solver. To create the network on the right, we applied the prefix optimization procedure described earlier to the Green filter prefix. The first version of our solver required 29921 seconds to find this network, whereas our current solver can find these networks in 282 seconds, when using the Green filter, and 60 seconds when using the optimized prefix.

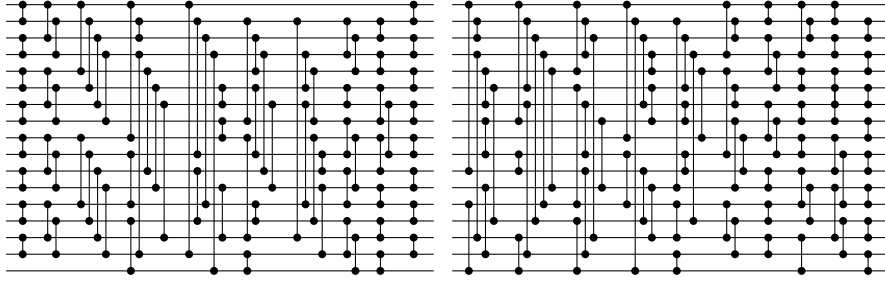Thus, we can summarize our results in the following theorem:

Fig. 5. Sorting networks for 17 channels of depth 10.

Theorem 2 *The optimum depth for a sorting network on* 17 *channels is* 10.

The network displayed in **??** sorts 20 inputs in 11 parallel steps, which beats the previously fastest network using 12 layers [1]. In the first layer, partially ordered sets of size 2 are created. These are merged to 5 partially ordered sets of size 4 in the second layer. The third layer is used to create partially ordered sets of size 8 for the lowest and highest wires, respectively. These are merged in the fourth layer.
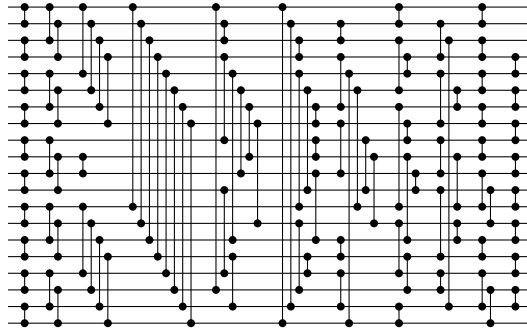

Fig. 6. A sorting network for 20 channels of depth 11.

The wires in the middle of the network are connected in order to totally sort their intermediate output. Using this prefix and the necessary conditions on sorting networks depicted above, we were able to create the remaining layers using our iterative, SAT-based approach. Interestingly, the result was created in 588 iterations, thus 587 different input vectors were sufficient.

6 Tools

Our software is based on the well-known SAT solver MiniSAT 2.20. Before starting a new loop of our network creation process, we used some probing-based preprocessing techniques [9] as they were quite successful on this kind

of SAT formulae. MiniSAT uses activity values for clauses which are used for managing the learnt clause database. Here, we changed the value "cla-decay" to 0.9999, which leads to better control on learnt clauses that were not used for a long time. Our experiments were performed on Intel Xeon E5-4640 CPUs clocked at 2.40GHz. The software used for our experiments can be downloaded at `http://www.informatik.uni-kiel.de/~the/SortingNetworks.html`.

Acknowledgments

References

1. S. W. A. Baddar and K. E. Batcher. A 12-step sorting network for 22 elements. Technical Report 2008-05, Kent State University, Dept. of Computer Science, 2008.
2. S. W. A. Baddar and K. E. Batcher. An 11-step sorting network for 18 elements. *Parallel Processing Letters*, 19(1):97–103, 2009.
3. K. E. Batcher. Sorting networks and their applications. In *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference, Atlantic City, NJ, USA, 30 April - 2 May 1968*, volume 32 of *AFIPS Conference Proceedings*, pages 307–314. Thomson Book Company, Washington D.C., 1968.
4. D. Bundala and J. Závodný. Optimal sorting networks. In *Language and Automata Theory and Applications - 8th International Conference, LATA 2014, Madrid, Spain, March 10-14, 2014. Proceedings*, volume 8370 of *LNCS*, pages 236–247. Springer, 2014.
5. M. Codish, L. Cruz-Filipe, and P. Schneider-Kamp. The quest for optimal sorting networks: Efficient generation of two-layer prefixes. *CoRR*, abs/1404.0948, 2014.
6. M. Codish, L. Cruz-Filipe, and P. Schneider-Kamp. Sorting networks: the end game. *CoRR*, abs/1411.6408, 2014.
7. D. Coles. Efficient filters for the simulated evolution of small sorting networks. In T. Soule and J. H. Moore, editors, *Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012*, pages 593–600. ACM, 2012.
8. D. E. Knuth. *The art of computer programming, volume 3: sorting and searching*. Addison-Wesley Professional, 1998.
9. I. Lynce and J. P. M. Silva. Probing-based preprocessing techniques for propositional satisfiability. In *15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2003), 3-5 November 2003, Sacramento, California, USA*, page 105. IEEE Computer Society, 2003.
10. I. Parberry. A computer-assisted optimal depth lower bound for nine-input sorting networks. *Mathematical Systems Theory*, 24(2):101–116, 1991.
11. V. K. Valsalam and R. Miikkulainen. Using symmetry and evolutionary search to minimize sorting networks. *Journal of Machine Learning Research*, 14:303–331, 2013.