Sorting Networks: to the End and Back Again

Michael Codish[a], Luís Cruz-Filipe[b,*], Thorsten Ehlers[c], Mike Müller[c], Peter Schneider-Kamp[b]

[a] *Department of Computer Science, Ben-Gurion University of the Negev, Israel*
[b] *Department of Mathematics and Computer Science, University of Southern Denmark*
[c] *Institut für Informatik, Christian-Albrechts-Universität zu Kiel, Germany*

Abstract

New properties of the front and back ends of sorting networks are studied, illustrating their utility when searching for bounds on optimal networks. Search focuses first on the "out-sides" of the network and then on the inner part. Previous works focused on properties of the front end to break symmetries in the search. The new, out-side-in, properties shed understanding on how sorting networks sort, and facilitate the computation of new bounds on optimality. We present new, faster, parallel sorting networks for 17–20 inputs. For 17 inputs, we show that no sorting network using less layers exists.

1. Introduction

Sorting networks are a classical model for sorting algorithms on fixed-length lists. The elements to sort are placed on the input channels of a network of compare-and-exchange units connecting pairs of channels. The sequence of these units, called *comparators*, uniquely determines the algorithm. Consecutive comparators can be viewed as a "parallel layer" if no two touch on the same channel. Ever since sorting networks were introduced, there has been a quest to find optimal sorting networks for particular small numbers of inputs: networks of optimal depth (in the number of parallel layers), as well as of optimal size (in the number of comparators).

In their celebrated result, Ajtai, Komlós and Szemerédi [1] give a construction for sorting networks with $O(n \log n)$ comparators in $O(\log n)$ parallel levels. These AKS sorting networks are a classical example of an algorithm optimal in theory, but highly inefficient in practice. Although they attain the theoretically optimal $O(n \log n)$ number of comparisons and $O(\log n)$ depth, the AKS networks are infamous for the large constants hidden in the big-O notation. On the other hand, already in 1968, Batcher [5] gave a simple recursive construction that, even though it creates networks with $O(n \log^2 n)$ comparators and depth $O(\log^2 n)$, is superior to AKS networks for all practical values of n. For an overview on sorting networks see, for example, Knuth [24] or Parberry [28].

Sorting networks are data-oblivious sorting algorithms, i.e., the sequence of comparisons they perform is independent of the input list. Thus, they are well suited to implementation as hardware circuits, where their size determines the number of gates used and their depth the delay of the circuit. Recent work [13, 20] has also shown their performance potential in software implementations of sorting as base cases of general recursive sorting algorithms.

The formal simplicity of the model, the ubiquitousness of sorting, and the computationally challenging nature of the optimization problems even for very small instances is a potent mixture, which has intrigued computer scientists since the middle 1950s. In the late 1960s, Donald E. Knuth and Robert W. Floyd authored a series of articles on the topic, essentially settling all optimality questions for networks with

*Corresponding author; tel. +45 6550 2387, fax +45 6550 2373
Email addresses: mcodish@cs.bgu.ac.il (Michael Codish), lcf@imada.sdu.dk (Luís Cruz-Filipe), the@informatik.uni-kiel.de (Thorsten Ehlers), mimu@informatik.uni-kiel.de (Mike Müller), petersk@imada.sdu.dk (Peter Schneider-Kamp)

up to 8 channels. The results of this effort are presented in the chapter dedicated to sorting networks in the famous monograph by Knuth [24]. Nevertheless, no further breakthroughs were made before Ian Parberry in 1989 [29, 30] settled the depth-optimality for 9 and 10 channels, using a then state-of-the-art Cray supercomputer. Only very recently, depth optimality for 11 to 16 [8] and size optimality for 9 and 10 channels [12] were finally settled.

The new results are due to both an exponential increase in computational power, vastly improved search methods, and new theoretical results on symmetries. To illustrate the first point, the program from [29, 30] for 9 channels runs in just 12.3 seconds on a single thread of Parberry's current desktop computer instead of 200 hours on a supercomputer [32]. Processing the case of 11 channels requires more than 2 months, and that of 13 channels is infeasible. In contrast, using SAT solvers and stronger notions of symmetry, the results for 11 and 13 channels were obtained in approx. 15 minutes and 11 hours, respectively [7].

Initial interest in sorting networks derived from the potential to provide fast parallel implementations of sorters in hardware, see e.g. [5]. Over the years, sorting networks have become central also in other types of applications. For example, median networks are circuits that identify the median value from a set of inputs. Median circuits can be implemented using sorting networks, and are applied in the design of median filters, which play an important role in video and image processing [6, 9, 10, 22]. Another modern application relates to selection networks: networks that select the k largest elements from n inputs. While selection networks are often implemented using sorting networks, Knuth [24] shows a simple construction of a selection network with $O(n \log^2 k)$ size, whereas the corresponding sorting network is of size $O(n \log^2 n)$. Selection networks are used when encoding cardinality constraints to CNF in several SAT-based constraint solvers, see e.g. [2, 3, 16].

Another line of research on sorting networks, which started in the early 1990s, focuses on the application of genetic algorithms to find small (depth and size) sorting networks. Noteworthy results include the work of Koza *et al.* [25], of Juillé [23], who found a 13-input network with 45 comparators in 10 parallel layers in 1995, and of Choi and Moon [11], who combined techniques involving network isomorphism and genetic algorithms to find 60-comparator sorting networks for 16 inputs (a network of this size was first discovered by Green in 1969, as reported in [21]). More recently, Valsalam and Miikkulainen [34] applied notions of symmetry and evolutionary search to find networks with smaller numbers of comparators than those known before for 17, 18, 19, 20, 21 and 22 inputs. Many other works apply genetic and evolutionary algorithms to search for small sorting networks. Their results provide upper bounds, but do not address the question of optimality.

This paper reveals new properties of sorting networks that deepen our understanding of how they work and facilitate our search for new results on depth optimality for sorting networks with more than 16 channels. All recent results on depth optimality have been made by identifying and breaking symmetries in the first layer(s) of the sorting networks, usually based on the property of sorting networks being closed under a particular variant of permutation. For example, Parberry's results on depth optimality [29, 30] derive from the observation that the first layer can be fixed, while the results of Bundala and Závodný [8] derive by fixing the first two layers and using an improved notion of symmetry.

All of the recent results on optimality of sorting networks are obtained as encodings to Boolean satisfiability, where a SAT solver is applied to determine the existence of a sorting network of a given size or depth. In this approach, the size of the SAT encodings is polynomial in the set of all inputs to the sorting network, and hence exponential in the number of elements to sort. This complexity seems inherent. Fixing some layers of the network leads to significant reduction in the size of the encoding, and is key to these new results.

The first contribution of this paper is a study of the *end* layers of a sorting network. We show that the comparators in the last layer of a sorting network are of a very particular form, and that those in penultimate layer are also of limited structure. We show that these results translate to constraints on the search for optimal sorting networks and lead to significant performance improvements when searching for networks of a given depth using SAT encodings. We also obtain theoretical results that, while not directly impacting the SAT solving times, establish a duality between the symmetries in the first layers and in the last layers of a sorting network.

The second contribution is an improvement of the SAT encoding used by [8], again reducing the size of

the SAT problems and improving SAT solving performance. Together with the first contribution, we are able to find a sorting network on 17 channels with depth 10 very efficiently, by fixing the first 3 layers as in a Green filter [17]. This result improves the previous upper bound of 11 to 10.

While the upper bound can simply be improved by finding a smaller sorting network, improving the lower bound from 9 to 10 requires showing that there does not exist a sorting network of depth 9 on 17 channels. In other words, this requires reasoning over the full space of 9-layer sorting networks. Fortunately, extending on the idea of [8], we do not need to consider all 2^n inputs: if there is no 9-layer network that sorts a subset of the inputs, then there is no 9-layer sorting network.

The third contribution of this paper is the observation that the performance of the SAT-based search for extensions of a prefix using the above idea heavily depends on the prefix chosen. We show how to refine prefixes by permutation in such a way that the size of the SAT problems is significantly reduced, and that this translates to real-world performance gains.

By combining all three contributions, the process of SAT solving for unsatisfiable instances experiences a speed-up of several orders of magnitude. This allows us to show that there is indeed no depth-9 sorting network on 17 channels, i.e., that the new depth-10 sorting network we found is not only faster, but that it is also optimal.

This paper is structured as follows. In Section 2 we summarize relevant concepts, theory, and results on sorting networks, and present an improved SAT encoding. Section 3 presents the new theoretical results about the structure of the last layers and exploits them to improve the SAT encoding. Section 4 then focuses on permuting prefixes in order to optimize the SAT encoding. As a result, we present in Section 5 more efficient sorting networks than previously known, and in Section 6 we prove a new lower bound for 17 channels. We summarize our results in Section 7.

This article combines and significantly extends preliminary work presented in [15] and in [19], building upon the original study of properties of the first two layers of sorting networks from [7].

2. An Overview of Sorting Networks

This section first introduces background material on sorting networks, together with an overview of previous results that are used in the remainder of the paper. At its end, we present an improved version of the SAT encoding from [8] for finding sorting networks.

2.1. Sorting Network Classics

A *comparator network* on n channels is a sequence $C = L_1, \ldots, L_d$ of *layers*, where each layer L_k is a non-empty set of pairs (i, j), with $1 \leq i < j \leq n$, such that: if $(i, j), (i', j') \in L_k$, then $\{i, j\} \cap \{i', j'\} = \emptyset$. The pairs (i, j) are called *comparators*; the *depth* of C is d, and the *size* of C is the total number of comparators in all its layers, $\sum_{k=1}^{d} |L_k|$.

An n-channel comparator network is viewed as a function with n inputs, which propagate through the network to give n outputs. Each comparator potentially changes the order between values on its two channels. Let $\bar{x}$ be a sequence of inputs from some totally ordered set and denote by $\bar{x}_\ell$ its ℓ-th element. Sequence $\bar{x}$ propagates through C as follows: at each layer L_k, if $(i, j) \in L_k$ and $\bar{x}_i > \bar{x}_j$, then $\bar{x}_i$ and $\bar{x}_j$ are interchanged. More formally, we define a sequence $\bar{x}^0, \ldots, \bar{x}^d$ as follows:

$$\bar{x}^0 = \bar{x}$$
$$\bar{x}_i^k = \begin{cases} \bar{x}_j^k & (i,j) \in L_k, \bar{x}_i^{k-1} > \bar{x}_j^{k-1} \\ \bar{x}_j^k & (j,i) \in L_k, \bar{x}_j^{k-1} > \bar{x}_i^{k-1} \\ \bar{x}_i^k & \text{otherwise} \end{cases} \tag{1}$$

The *output* of C on $\bar{x}$, which we denote by $C(\bar{x})$, is $\bar{x}^d$. The set of all outputs of C is $\mathsf{outputs}(C)$, and C is said to be a *sorting network* (on n channels) if all these sequences are sorted. The zero-one principle (see e.g. [24]) states that a comparator network C is a sorting network if and only if C sorts all Boolean inputs. Hence, in the remainder of the paper, we consider only comparator networks with Boolean inputs.

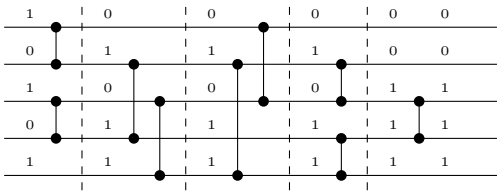


Figure 1: The sorting network $\{(1,2),(3,4)\},\{(2,4),(3,5)\},\{(1,3),(2,5)\},\{(2,3),(4,5)\},\{(3,4)\}$, on 5 channels, operating on the input 10101.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
s_n	0	1	3	5	9	12	16	19	25	29	35 33	39 37	45 41	51 45	56 49	60 53	71 58	78 63	86 68	92 73
t_n	0	1	3	3	5	5	6	6	7	7	8	8	9	9	9	9	**10**	11 **10**	**11** **10**	**11** **10**

Table 1: Best known values and bounds on optimal size (s_n) and depth (t_n) of sorting networks on n inputs, for $n \leq 20$. The contributions of this paper are shown in boldface.

It is common practice to represent comparator networks graphically as a Knuth diagram, as showed in Figure 1. Channels are depicted as horizontal lines, with values traveling from left to right, and comparators as vertical lines connecting two channels. Layers are sometimes made explicit and then separated by dashed vertical lines.

For sorting networks, two natural optimization questions arise.

- The *optimal size problem*: what is the minimum number of comparators, s_n, in a sorting network on n channels?
- The *optimal depth problem*: what is the minimum number of layers, t_n, in a sorting network on n channels?

The sorting network in Figure 1, constructed in 5 layers with a total of 9 comparators, is both size-optimal and depth-optimal. In general, there is not always a single network that is optimal for both criteria. Table 1 presents the currently known values of s_n and t_n. In case the precise value is unknown, the table presents the best known lower and upper bounds. For instance, the smallest open problem for the optimal size problem is for 11 channels. The minimum number of comparators in a sorting network on 11 channels is known to be between 33 and 35, but the precise number is not known. Values which are derived as contributions of this paper are set in boldface. The values of s_1–s_8 and t_1–t_8 were already known in the 1960s and are included in [24], as well as all upper bounds up to s_{16} and t_{16}. The remaining upper bounds for s_n can be found in [34], and the upper bounds $t_{17}, t_{18} \leq 11$ and $t_{19}, t_{20} \leq 12$ in [4]. A computer-assisted proof of the exact values of s_9 and s_{10} was first presented in 2014 [12], and the remaining lower bounds for s_n follow from these by the result of van Voorhis [35] that $s_n \geq s_{n-1} + \lceil \log_2 n \rceil$. The exact values of t_9 and t_{10} were first shown by Parberry in 1989 [29, 30], and those of t_{11}–t_{16} by Bundala and Závodný in 2014 [7, 8]. The latter results imply that $t_n \geq 9$ for every $n \geq 17$. Among the contributions of this paper are the proofs that $t_{17} = 10$ and $t_{20} \leq 11$. As a consequence, we obtain that $10 \leq t_n \leq 11$ for $18 \leq n \leq 20$.

For $\bar{x}, \bar{y} \in \{0,1\}^n$ we write $\bar{x} \leq \bar{y}$ to denote that every bit of $\bar{x}$ is less than or equal to the corresponding bit of $\bar{y}$, and $\bar{x} < \bar{y}$ for $\bar{x} \leq \bar{y}$ and $\bar{x} \neq \bar{y}$. The following two observations will be be instrumental for proofs in later sections.

Lemma 1. *Let C be a comparator network with d layers, $\bar{x}^0 \in \{0,1\}^n$, and $\bar{x}^0, \ldots, \bar{x}^d$ the sequence defined in Equation (1). If $\bar{x}^0$ has r leading zeroes and s trailing ones ($r + s \leq n$), then each sequence $\bar{x}^i$, with $1 \leq i \leq d$, also has r leading zeroes and s trailing ones. In particular, if $\bar{x}^0$ is sorted, then the sequence $\bar{x}^0, \ldots, \bar{x}^d$ is constant.*

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$\lvert G_n\rvert$	10^0	10^1	10^1	10^1	10^2	10^2	10^3	10^3	10^4	10^5	10^5	10^6	10^7	10^7	10^8	10^8	10^9	10^{10}
$\lvert R_n\rvert$	1	2	4	5	8	12	22	21	48	50	117	94	262	211	609	411	1,367	894

Table 2: Order of magnitude of $|G_n|$, the number of possible second layers on n channels, and value of $|R_n|$, a complete set of two-layer filters, for $n \leq 20$.

To the best of our knowledge, Lemma 1 has never been stated explicitly, but it is used implicitly in [8]. The observation that $\bar{x}^0 = \bar{x}^1 = \cdots = \bar{x}^d$ when $\bar{x}^0$ is sorted was already made in [24].

Lemma 2 (Theorem 4.1 in [4])**.** *Let C be a comparator network and $\bar{x}, \bar{y} \in \{0,1\}^n$ be such that $\bar{x} \leq \bar{y}$. Then $C(\bar{x}) \leq C(\bar{y})$.*

2.2. On Fixing the First Two Layers

Throughout this paper we focus mostly on the optimal-depth problem. We aim to find new results to strengthen and extend the type of techniques applied in previous works that helped establish the values of t_9–t_{16}.

In order to establish that $t_n > k$, it is necessary to show that no comparator network on n channels with depth k can sort all inputs. Except in very simple cases, all known techniques require the analysis of the entire search space of depth-k comparator networks. However, this space grows very rapidly: the number of all possible layers on n channels coincides with the number of matchings in a complete graph on n vertices, which is exponential in n, so exhaustively generating it only works for small values of n.

Previous work reduces the search space by considering symmetries in the first two layers of a sorting network. The following terminology is useful. Given two comparator networks $C_1 = L_1, \ldots, L_d$ and $C_2 = L'_1, \ldots, L'_{d'}$, their concatenation is the network $C = C_1; C_2 = L_1, \ldots, L_d, L'_1, \ldots, L'_{d'}$, and we say that C_1 is a *d-layer prefix* (or simply a *prefix*) of C. A layer is *maximal* if it contains exactly $\lfloor \frac{n}{2} \rfloor$ comparators. We will call a set $\mathcal{F}$ of comparator networks of depth k a *complete set of filters* (of depth k) if there is an optimal-depth sorting network on n channels of the form $F; C$ with $F \in \mathcal{F}$. Parberry proved [29, 30] that $\{L\}$ is a complete set of filters of depth 1 for any maximal layer L on n channels, and illustrated results using the particular first layer filter that we denote

$$L_1^P = \left\{ \; (2i-1, 2i) \;\middle|\; 1 \leq i \leq \left\lfloor \tfrac{n}{2} \right\rfloor \; \right\} \tag{2}$$

Parberry also considered restrictions on the second layer of a sorting network, noting that one need not consider prefixes that are identical modulo permutations of channels that leave the first layer fixed.

Parberry's ideas were later extended by Bundala and Závodný [8], who pruned the search space by further restricting the possibilities for the second layer of a sorting network. These results were strengthened in [14], which presents an efficient algorithm to generate complete sets of filters R_n of depth 2 for any n. The sizes of these sets for $n \leq 20$ are given in Table 2; for comparison, we also detail the order of magnitude of the total number $|G_n|$ of possible layers. The construction of the sets R_n is described in [8, 14] and uses two different ideas. The first is the notion of *saturation*: a two-layer network $L_1; L_2$ is saturated if L_1 is maximal and it is not possible to find $L'_2 \supsetneq L_2$ such that $\mathsf{outputs}(L_1; L'_2) \subsetneq \mathsf{outputs}(L_1; L_2)$. It follows that the set of all saturated networks is a complete set of two-layer filters. The second notion involves permutations. Given two comparator networks C and C' on n channels, we say that C *subsumes* C', denoted $C \preceq C'$, if $\pi(\mathsf{outputs}(C)) \subseteq \mathsf{outputs}(C')$ for some permutation π of $\{1, \ldots, n\}$. In this situation, if C' can be extended to a sorting network, then so can C (within the same size or depth), which allows C' to be removed from the search space. In fact, the sets R_n are not unique, as replacing any network in a complete set of filters by one of its permutations yields another complete set of filters. In the following, we denote by $\mathcal{F}_P$ a complete set of filters that all have the layer L_1^P of Equation (2) as first layer.

Given a complete set $\mathcal{F}$ of (two-layer) filters on n channels, one may search for a depth-d sorting network considering (in parallel) the independent search problems of extending each $F \in \mathcal{F}$ to a sorting network

of depth d. It is sufficient to solve any one of these separate search problems. Similarly, to prove that no depth-d sorting network on n channels exists, it is sufficient to show (in parallel) that none of the filters $F \in \mathcal{F}$ extend to a depth-d sorting network.

2.3. Prefixes with More than Two Layers

Over the years, when searching for smaller sorting networks (improved upper bounds) it has become common practice to fix the prefix of the network, typically with more than just two layers. For example, in 1969, Green found the smallest network on 16 channels known until today (with 60 comparators) applying what is now called a *Green filter* [24]. A Green filter on n channels (where n is a power of 2) consists of $\log_2 n$ maximal layers, structured as exemplified for 16 channels in Figure 2. It is also interesting to note that Parberry's construction for Pairwise sorting networks introduced in [31] includes a Green filter (by construction), although it is not presented as such. Codish and Zazon showed in [16] that Parberry's Pairwise network is also a simple reordering of the layers in Batcher's Odd-Even network [5].

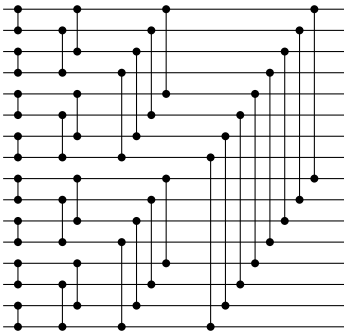


Figure 2: A Green filter on 16 channels.

When searching for improved upper bounds on the depth (or size) of a sorting network, a common criterion to select a particular filter is to consider the number of its (unsorted) outputs. For example, Coles [17] suggests to evaluate filters in this way, introducing the notion of Square filters, and observing that, for 16-channel networks, these produce fewer outputs than Green filters. Later, Baddar and Batcher [4] developed the `sortnet` program to facilitate the specification of a filter and its evaluation with respect to its number of outputs.

In this paper we show that, using the first three layers of the Green filter depicted in Figure 2 together with our other contributions, we can find a 10-layer sorting network on 17 channels. This improves on the previously smallest-known depth of a sorting network on 17 channels (11 layers).

2.4. SAT Encoding for Depth-Restricted Sorting Networks

A first approach to encode sorting networks as formulae in propositional logic was suggested by Morgenstern and Schneider [27]. However, their encoding to SAT did not prove sufficient to find new results concerning optimal-depth networks, as it did not scale for $n > 10$. Bundala and Závodný [8] continued this approach, and introduced a better SAT encoding that was able both to find sorting networks of optimal depth with up to 13 channels and to prove their optimality. As this lower bound matches the upper bound for sorting networks on 16 channels, this implies the optimal depth of the best known networks with up to 16 channels, cf. Table 1. This was the first SAT formulation that led to new results on optimal-depth sorting networks. Bundala *et al.* [7] further improve this encoding to establish optimal depth for networks with $n \leq 16$ channels directly. In this paper, we obtain results applying further extensions and optimizations of the SAT encoding presented in [7], hence for sake of completeness, we recall it here.

A comparator network of depth d on n channels is represented by a set of Boolean variables $C_n^d = \left\{ g_{i,j}^k \;\middle|\; 1 \leq i < j \leq n, 1 \leq k \leq d \right\}$, the value of $g_{i,j}^k$ indicating whether there is a comparator on channels i and j in layer k in the network or not. Furthermore, denoting the inputs into the network by the variables v_i^0,

with $1 \leq i \leq n$, the variables v_i^k, with $1 \leq i \leq n$ and $1 \leq k \leq d$, store the value on channel i in the network after layer k. The existence of a sorting network of depth d on n channels is then encoded in terms of the following propositional constraints:

$$
\begin{aligned}
\mathit{once}_i^k(C_n^d) &= \bigwedge_{1 \leq i \neq j \neq \ell \leq n} \left(\neg g^k_{\min(i,j),\max(i,j)} \vee \neg g^k_{\min(i,\ell),\max(i,\ell)} \right) \\
\mathit{valid}(C_n^d) &= \bigwedge_{1 \leq k \leq d, 1 \leq i \leq n} \mathit{once}_i^k(C_n^d) \\
\mathit{used}_i^k(C_n^d) &= \bigvee_{j<i} g^k_{j,i} \vee \bigvee_{i<j} g^k_{i,j} \\
\mathit{update}_i^k(C_n^d, \bar{v}, w) &= \left(\neg \mathit{used}_i^k(C_n^d) \rightarrow (w \leftrightarrow v_i) \right) \wedge \\
&\quad \bigwedge_{1 \leq j < i} \left(g^k_{j,i} \rightarrow (w \leftrightarrow (v_j \vee v_i)) \right) \wedge \\
&\quad \bigwedge_{i < j \leq n} \left(g^k_{i,j} \rightarrow (w \leftrightarrow (v_j \wedge v_i)) \right)
\end{aligned}
$$

Here, *once* encodes the fact that each channel may be used only once in one layer, and *valid* enforces this constraint for each channel and each layer. The *update* constraint specifies the value w in channel i after layer k, given that the values before that layer are those in $\bar{v} = (v_1, \ldots, v_n)$. In the next constraint, $\bar{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ and $\bar{y} = (y_1, \ldots, y_n)$ is the sequence obtained by sorting $\bar{x}$, while $\bar{v}^k = (v_1^k, \ldots, v_n^k)$ are fresh variables for $k = 0, \ldots, d$.

$$
\mathit{sorts}(C_n^d, \bar{x}) = \bigwedge_{1 \leq i \leq n} (v_i^0 \leftrightarrow x_i) \wedge \bigwedge_{1 \leq k \leq d, 1 \leq i \leq n} \mathit{update}_i^k(C_n^d, \bar{v}^{k-1}, v_i^k) \wedge \bigwedge_{1 \leq i \leq n} (v_i^d \leftrightarrow y_i) \tag{3}
$$

The constraint *sorts* encodes whether a certain input, $\bar{x}$, 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 $\bar{y}$. A sorting network for n channels on d layers exists if and only if the following constraint is satisfiable.

$$
\varphi(n,d) = \mathit{valid}(C_n^d) \wedge \bigwedge_{\bar{x} \in \{0,1\}^n} \mathit{sorts}(C_n^d, \bar{x}) \tag{4}
$$

When seeking a sorting network of depth d that extends a given prefix P, we consider instead the proposition *"There is a comparator network on n channels of depth $d - |P|$ that sorts* $\mathsf{outputs}(P)$*"*, or, formally:

$$
\varphi_P(n,d) = \mathit{valid}(C_n^{d-|P|}) \wedge \bigwedge_{\bar{x} \in \mathsf{outputs}(P)} \mathit{sorts}(C_n^{d-|P|}, \bar{x}) \tag{5}
$$

The encoding that we propose is of size exponential in the number of channels, n. This is also the case for the encoding presented in [27]. We can state our problem symbolically as the following formula.

$$
\exists C_n^d . \mathit{valid}(C_n^d) \wedge \forall \bar{x} \in \{0,1\}^n . \mathit{sorts}(C_n^d, \bar{x}) \tag{6}
$$

This formula is of the form $\exists\forall\varphi$ (there *exists* a network that sorts *all* of its inputs), and is easily shown to be in Σ_2^P. We expect that, similar to the problem of circuit minimization [33], it is also complete in Σ_2^P, although we have not succeeded to prove this. We do not expect that there exists a polynomial size encoding to SAT. We have also experimented with a QBF solver, but results for small instances indicate that this approach is not competitive with the results obtained using SAT solvers.

2.5. Our Improved SAT Formulation

In this paper we us a further optimized version of the SAT encoding detailed in Section 2.4. First, we use Lemma 1 to hardwire leading zeros, and trailings ones, as done already in [8]. Although this can also be detected by Failed Literal Branching [26] on the SAT formula, it is beneficial to do this at encoding time.

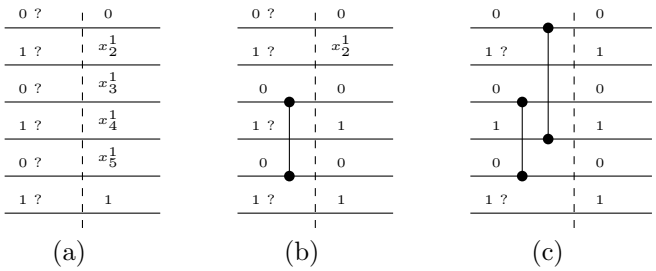


Figure 3: Propagations for the first layer of a sorting network on 6 channels determined from the input sequence 010101 (see Example 1).

The main optimization is about adding additional clauses and variables that help propagate information in the SAT solving phase. We demonstrate this (and also the first optimization) by means of an example.

Example 1. *Consider a sorting network on $n = 6$ channels, an input $\bar{x}^0 = (0, 1, 0, 1, 0, 1)$, and the output of the first layer, $\bar{x}^1 = (x_1^1, x_2^1, x_3^1, x_4^1, x_5^1, x_6^1)$. Figure 3(a) illustrates this setting, where a "?" on an input value indicates that we do not know whether a comparator will be placed somewhere on the corresponding channel. By Lemma 1, $x_1^1 = 0$ and $x_6^1 = 1$, so $\bar{x}^1 = (0, x_2^1, x_3^1, x_4^1, x_5^1, 1)$, as indicated in the second layer of Figure 3(a). Now consider the value of x_4^1. Clearly, the only first level comparator that will change the input value $x_4^0 = 1$ is $(4, 5)$. Therefore, adding any other comparator on channel 5 determines that $x_4^1 = 1$ and one could specifically add propagation clauses of the form $g_{i,5}^1 \to x_4^1$ for $1 \leq i < 4$.*

Figure 3(b) illustrates the situation where comparator $(3, 5)$ is placed in layer 1. Channels 3 and 5 are now in use, hence the "?" is removed from the corresponding input values. The values of x_3^1 and x_5^1 are determined by the comparator. Moreover, as argued above, the value of x_4^1 is set to 1.

Figure 3(c) illustrates the situation if a second comparator, $(1, 4)$, is added to layer 1. The value $x_2^1 = 1$ is determined by an argument similar to the one that determined $x_4^1 = 1$.

Example 1 demonstrates that positioning of single comparators has additional implications that can be considered in a SAT encoding. These implications derive from specific input sequences, but the information required to use them is "global". Thus, for every layer k and every pair of channels (i, j) we introduce propositional variables $oneDown_{i,j}^k$ and $oneUp_{i,j}^k$, which indicate whether there is a comparator $g_{\ell,j}^k$ for some $i \leq \ell < j$ or $g_{i,\ell}^k$ for some $i < \ell \leq j$, respectively.

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

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

To make use of these new propositional variables, given an input $\bar{x} = (0, 0, \ldots, 0, x_t, x_{t+1}, \ldots, x_{t+r-1}, 1, 1, \ldots, 1)$, for all $t \leq i \leq t + r - 1$ and at each layer k, we add the following constraints to the definition of *sorts* given in Equation (3).

$$\bigwedge_{1\leq k\leq d} v_i^{k-1} \wedge noneDown_{i,t+r-1}^k \to v_i^k$$

$$\bigwedge_{1\leq k\leq d} \neg v_i^{k-1} \wedge noneUp_{t,i}^k \to \neg v_i^k$$

With these constraints added, we remove some of the clauses of the *update* constraint: the clauses that describe that the value on one channel remains unchanged independent of the value on another channel are covered by the new constraints, and may therefore be removed. This encoding reduces the overall encoding size. Table 7 shows the impact of this improved encoding. Furthermore – as depicted in the example – the

improved encoding allows for more propagations during search. Thus, a SAT solver can recognize infeasible branches of the search tree earlier, which is beneficial for its performance.

3. The End Game

Recent results on depth-optimal sorting networks stemmed from the restriction of the search space of all comparator networks based on complete sets of filters. In this section, we focus on the dual problem: what do the *last* layers of a sorting network look like? We obtain several interesting theoretical results that shed some insight on the semantics of sorting networks, and also help to solve open instances of the optimal-depth problem.

To the best of our knowledge, the only previous attempt to address this issue was Parberry's heuristic for deciding whether a given prefix could be extended to a sorting network [29, 30], which used necessary conditions on the last two layers. However, those conditions are not expressible without knowing what all but the last two layers are. In contrast, we look for generic conditions about all sorting networks.

3.1. The Last Layers of a Sorting Network

We begin by recalling the notion of redundant comparator, introduced in Exercise 5.3.4.51 of [24] with credit to R.L. Graham.

Definition 1. *Let $C;(i,j);C'$ be a comparator network. The comparator (i,j) is* redundant *if $\bar{x}_i \leq \bar{x}_j$ for all $\bar{x} \in \mathsf{outputs}(C)$.*

A sorting network without redundant comparators is called *non-redundant*.

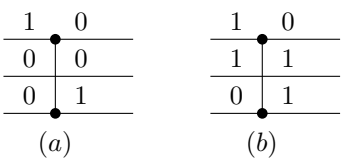
Lemma 3. *Let D be a comparator network on n channels. Construct D' by removing every redundant comparator from D. Then D' is a sorting network iff D is a sorting network.*

Proof. From the definition of redundant comparator, it follows that $D(\bar{x}) = D'(\bar{x})$ for every $\bar{x} \in \{0,1\}^n$. □

Lemma 3 was already explored in the proof of optimality of the 25-comparator sorting network on 9 channels [12]. In general, it is not easily applicable to proofs of optimal depth based on SAT-encodings, because redundancy is a semantic property that is not easy to encode syntactically. In order to use it in this context, we need syntactic criteria for redundancy.

Lemma 4. *Let C be a non-redundant sorting network on n channels. Then all comparators in the last layer of C are of the form $(i, i+1)$.*

Proof. Let C be a non-redundant sorting network with a comparator $c = (i, i+2)$ in the last layer. Since c is not redundant, there is an input $\bar{x}$ such that channels i to $i+2$ before applying c look like (a) or (b) on the right.



(a) (b)

Suppose $\bar{x}$ is an input yielding case (a), and let $\bar{y}$ be any input obtained by replacing one 0 in $\bar{x}$ by a 1. Since C is a sorting network, $C(\bar{y})$ is sorted, but since $\bar{x} < \bar{y}$ the value in channel i before applying c must be a 1 (Lemma 2), hence $\bar{y}$ yields situation (b). Dually, given $\bar{y}$ yielding (b), we know that any $\bar{z}$ obtained by replacing one 1 in $\bar{y}$ by a 0 will yield (a).

Thus all inputs with the same number of zeroes as $\bar{x}$ or $\bar{y}$ must yield either (a) or (b), in particular sorted inputs, contradicting Lemma 1. The same reasoning generalizes to show that c cannot have the form $(i, i+k)$ with $k \geq 2$, thus it has to be of the form $(i, i+1)$. □

Corollary 1. *Suppose that C is a non-redundant sorting network that contains a comparator (i,j) at layer d, with $j > i+1$. Then at least one of channels i and j is used in a layer d' with $d' > d$.*

Proof. If neither i nor j are used after layer d, then comparator (i, j) can be moved to the last layer without changing the function computed by C. By the previous lemma, C cannot be a non-redundant sorting network. □

Lemma 4 restricts the number of possible comparators in the last layer in a sorting network on n channels to $n-1$, instead of $n(n-1)/2$ in the general case.

Theorem 1. *The number of possible last layers in an n-channel sorting network with no redundancy is $L_n = F_{n+1} - 1$, where F_n denotes the Fibonacci sequence.*

Proof. Denote by L_n^+ the number of possible last layers on n channels, where the last layer is allowed to be empty (so $L_n = L_n^+ - 1$). There is exactly one possible last layer on 1 channel, and there are two possible last layers on 2 channels (no comparators or one comparator), so $L_1^+ = F_2$ and $L_2^+ = F_3$.

Given a layer on n channels, there are two possibilities. Either the first channel is unused, and there are L_{n-1}^+ possibilities for the remaining $n-1$ channels; or it is connected to the second channel, and there are L_{n-2}^+ possibilities for the remaining $n-2$ channels. So $L_n^+ = L_{n-1}^+ + L_{n-2}^+$, whence $L_n^+ = F_{n+1}$. □

Even though L_n grows quickly, it grows much slower than the number $|G_n|$ of possible layers in general (Table 2). In particular, $L_{17} = 2583$, whereas $G_{17} = 211{,}799{,}312$.

To move backwards from the last layer, we introduce an auxiliary notion.

Definition 2. *Let C be a depth d sorting network without redundant comparators, and let $k < d$. A k-*block *of C is a maximal set of channels B such that: if $i, j \in B$, then there is a sequence of channels $i = x_0, \ldots, x_\ell = j$ where (x_i, x_{i+1}) or (x_{i+1}, x_i) is a comparator in a layer $k' > k$ of C.*

For each k the set of k-blocks of C is a partition of the set of channels of C. Given a k-block B, we will abuse terminology and refer to "the comparators in B" to denote the comparators connecting channels in B after layer k.

Given a comparator network of depth d, we will call its $(d-1)$-blocks simply *blocks* – so Lemma 4 states that a block in a sorting network C contains either (a) a channel unused at the last layer of C or (b) two adjacent channels connected by a comparator at the last layer of C.

Example 2. *Recall the sorting network shown in Figure 1, and reproduced in Figure 4. Its 4-blocks, or simply blocks, are $\{1\}$, $\{2\}$, $\{3, 4\}$ and $\{5\}$; its 3-blocks are $\{1\}$, $\{2, 3, 4, 5\}$; and for $k < 3$ there is only the trivial k-block $\{1, 2, 3, 4, 5\}$. The 3-block $\{2, 3, 4, 5\}$ contains the comparators $(2, 3)$, $(4, 5)$ and $(3, 4)$.*

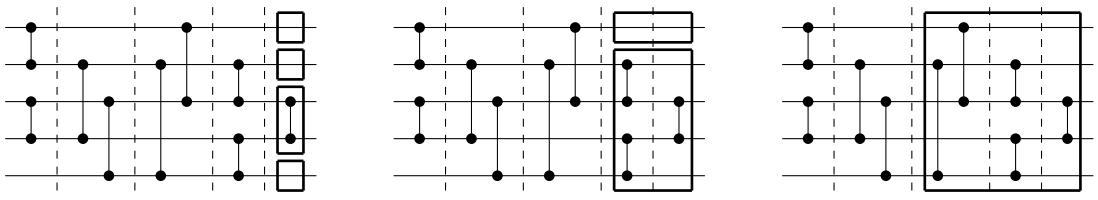


Figure 4: The 4-, 3- and 2-blocks of the sorting network $\{(1,2),(3,4)\}, \{(2,4),(3,5)\}, \{(1,3),(2,5)\}, \{(2,3),(4,5)\}, \{(3,4)\}$.

The notion of block helps understand the semantics of a sorting network: after layer k, all k-blocks are sorted except for at most one.

Lemma 5. *Let C be a sorting network on n channels with depth d, and $k < d$. For each input $\bar{x} \in \{0,1\}^n$, there is at most one k-block that receives both 0s and 1s as input.*

Proof. From the definition of k-block, it follows that values cannot move between two different k-blocks. Therefore, for every input, if there is a k-block that receives both 0s and 1s as inputs, then all k-blocks above it must receive only 0s and those below it must receive only 1s in order for the output to be sorted. □

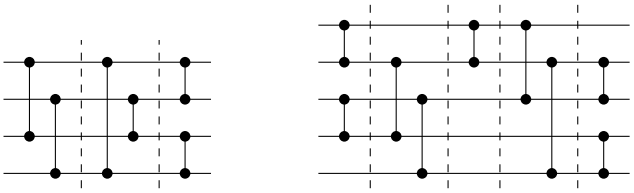


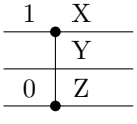
Figure 5: Sorting networks containing a comparator $(i, i+3)$ in their penultimate layer.

For example, in Figure 4 (left), if the 4-block $\{3,4\}$ receives one 0 and one 1, then the 4-blocks $\{1\}$ and $\{2\}$ must receive a 0 and the 4-block $\{5\}$ must receive a 1.

Lemma 6. *Let C be a non-redundant sorting network on n channels with depth d. Then all comparators in layer $d-1$ connect adjacent blocks of C.*

Proof. The proof is similar to that of Lemma 4, but now considering blocks instead of channels. Let c be a comparator in layer $d-1$ of C that does not connect adjacent blocks of C. Observe that c cannot connect two channels in the same block, as then there would be two copies of c in the last two layers, and the second one would be redundant.

Since c is not redundant, there must be some input $\bar{x}$ that provides c with input 1 on its top channel and 0 on its bottom channel. The situation is depicted below, where X and Z are blocks, and Y is the set of channels in between. According to Lemma 5, there are five possible cases for X, Y and Z, depending on the number of 0s in $\bar{x}$.



X	all 0s	all 0s	all 0s	all 0s	mixed
Y	all 0s	all 0s	mixed	all 1s	all 1s
Z	mixed	all 1s	all 1s	all 1s	all 1s
	(a)	(b)	(c)	(d)	(e)

Suppose that input $\bar{x}$ leads to case (a). By changing the appropriate number of 0s in $\bar{x}$ to 1s, we can find an input $\bar{y}$ that leads to case (b), since again by monotonicity of C $\bar{y}$ cannot place a 0 to the top input of c. Likewise, we can reduce (e) to (d). But now we can move between (b), (c) and (d) by changing one bit of the word at a time. By Lemma 2, this must keep either the top 1 input of c or the lower 0, while the other value is kept by the fact that C is a sorting network. As in Lemma 4, this proves that this configuration occurs for all words with the same number of 0s, which is absurd since it cannot happen for sorted inputs. □

Combining this result with Lemma 4 we obtain the explicit configurations that can occur in a sorting network.

Corollary 2. *Let C be a non-redundant sorting network on n channels with depth d. Then every comparator (i,j) in layer $d-1$ of C satisfies $j-i \leq 3$. Furthermore, if $j = i+2$, then either $(i, i+1)$ or $(i+1, i+2)$ occurs in the last layer; and if $j = i+3$, then both $(i, i+1)$ and $(i+2, i+3)$ occur in the last layer.*

The sorting networks in Figure 5 show that the bound $j - i \leq 3$ is tight.

Theorem 2. *If C is a non-redundant sorting network on n channels and (i,j) is a comparator at layer k of C, then i and j are either in the same k-block or in adjacent k-blocks of C.*

Proof. As for Lemma 6, considering k-blocks instead of blocks. □

As a consequence of Theorem 2, all k-blocks in a non-redundant sorting network are sets of consecutive channels.

Corollary 3. *Let C be a non-redundant sorting network on n channels and B be a k-block of C containing m comparators. Then B consists of at most $m+1$ channels.*

Proof. By induction on m. The case $m = 1$ is simply Corollary 1. If $m > 1$, let c be a comparator at layer $k+1$ of C and consider the sorting network C' obtained by placing c into an individual layer. By Theorem 2, either (i) c connects channels within a $(k+1)$-block B' of C' or (ii) c connects channels in adjacent $(k+1)$-blocks B_1 and B_2 of C'. In case (i), block B' contains $m-1$ comparators, hence by induction hypothesis it consists of at most $m-2$ channels; since c does not add any new channels, the thesis also holds for B. In case (ii), blocks B_1 and B_2 contain m_1 and m_2 comparators each, with $m_1 + m_2 = m - 1$, and by induction hypothesis they consist of at most $m_1 + 1$ and $m_2 + 1$ channels, respectively. Since B consists of the union of these sets, it has at most $(m_1 + 1) + (m_2 + 1) = m + 1$ elements. □

3.2. Co-saturation

Using the results from Section 3.1, we can reduce the search space of possible sorting networks of a given depth simply by restricting to comparator networks satisfying the necessary conditions presented, namely Lemma 4 and Corollary 2. However, the successful strategies in [8, 12, 29, 30] all focus on also imposing *sufficient* conditions on those networks. This is expressed by results of the form "if there is a sorting network, then there is one satisfying this property".

We now follow this approach pursuing the idea of saturation from [7]: how many (redundant) comparators can we safely add to the last layers of a sorting network? We will show how to do this in a structured way that actually reduces the number of possibilities for the last two layers. Again, we capitalize on the observation that redundant comparators do not change the function represented by a comparator network and can, thus, be removed or added at will.

Lemma 7. *Let C be a sorting network on n channels. There is a sorting network N of the same depth as C whose last layer: (i) only contains comparators between adjacent channels; and (ii) does not contain two adjacent unused channels.*

Proof. Let C be a sorting network on n channels. By Lemma 3, we can eliminate all redundant comparators from C to obtain a sorting network S. By Lemma 4, all comparators in the last layer of S are of the form $(i, i+1)$. Let j be such that j and $j+1$ are unused in the last layer of S; since S is a sorting network, this means that comparator $(j, j+1)$ is redundant, and we can add it to the last layer of S. Repeating this process for $j = 1, \ldots, n$ we obtain a sorting network N that satisfies both desired properties. □

We say that a sorting network satisfying the conditions of Lemma 7 is in *last layer normal form* (llnf).

Theorem 3. *The number of possible last layers in llnf on n channels is $K_n = P_{n+5}$, where P_n denotes the Padovan sequence, defined as $P_0 = 1$, $P_1 = P_2 = 0$ and $P_{n+3} = P_n + P_{n+1}$.*

Proof. Let K_n^+ be the number of layers in llnf that begin with the comparator $(1, 2)$, and K_n^- the number of those where channel 1 is free. Then $K_n = K_n^+ + K_n^-$. Let $n > 3$. If a layer in llnf begins with a comparator, then there are K_{n-2} possibilities for the remaining channels; if it begins with a free channel, then there are K_{n-1}^+ possibilities for the remaining channels. Therefore $K_n = K_n^+ + K_n^- = K_{n-2} + K_{n-1}^+ = K_{n-2} + K_{n-3}$. There exist one last layer on 1 channel (with no comparator), one on 2 channels (with one comparator between them) and two on 3 channels (one comparator between either the top two or the bottom two channels), so $K_1 = P_6$, $K_2 = P_7$ and $K_3 = P_8$. From the recurrence it follows that $K_n = P_{n+5}$. □

Sequence K_n grows much slower than the total number L_n of non-redundant last layers identified in Theorem 1. For example, $K_{17} = 86$ whereas $L_{17} = 2583$.

Given that the last layer is required to be in llnf, we can also study the previous layer. By Lemma 6, we know that every block can only be connected to the adjacent ones; again we can *add* redundant comparators to reduce the number of possibilities for the last two layers.

Lemma 8. *Let C be a sorting network of depth d in llnf. Let $i < j$ be two channels that are unused in layer $d-1$ and that belong to different blocks. Then adding the comparator (i, j) to layer $d-1$ of C still yields a sorting network.*

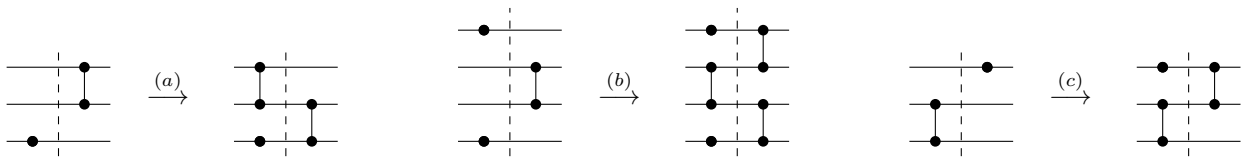


Figure 6: Transformations in the proof of Lemma 9.

Proof. Suppose there is an input $\bar{x}$ such that channel i carries a 1 at layer $d-1$ and channel j carries a 0 at that same layer. Since neither channel is used, their corresponding blocks will receive these values. But then $C(\bar{x})$ has a 1 in a channel in the block containing i and a 0 in the block containing j, and since $i < j$ this sequence is not sorted by C, contradicting the assumption that C is a sorting network. Therefore the comparator (i, j) at layer $d-1$ of C is redundant, and can be added to this network. □

Incidentally, this lemma provides a partial answer to a problem also posed in Exercise 5.3.4.21 [24]: when can we add comparators to a sorting network while keeping it sorting? It was already known that this can always be done in the first and last layers, while a result by N.G. de Bruijn [18] states that this can always be done in networks that only contain comparators of the form $(i, i+1)$. Lemma 8 gives a general sufficient condition, and it implies in particular that comparators can also always be added between any two free channels in the penultimate layer of any sorting network.

Lemma 9. *Let C be a sorting network of depth d in llnf. Suppose that there is a comparator $(i, i+1)$ in the last layer of C, that channel $i+2$ is used in layer $d-1$ but not in layer d, and that channels i and $i+1$ are both unused in layer $d-1$ of C (see Figure 6 (a)). Then there is a sorting network C' of depth d in llnf such that channels $i+1$ and $i+2$ are both used in layers $d-1$ and d.*

Proof. Since channels i and $i+1$ are unused in layer $d-1$, comparator $(i, i+1)$ can be moved to that layer without changing the behavior of C; then the redundant comparator $(i+1, i+2)$ can be added to layer d, yielding the sorting network C' (Figure 6 (a)). If $i > 1$ and channel $i-1$ is not used in the last layer of C, then C' must also contain a comparator $(i-1, i)$ in its last layer (Figure 6 (b)). □

A construction similar to that in Lemma 9 can also be applied if channel $i-1$ (instead of $i+2$) is used at layer $d-1$ and unused in layer d (Figure 6 (c)).

Definition 3. *A sorting network of depth d is* co-saturated *if: (i) its last layer is in llnf, (ii) no two consecutive blocks at layer $d-1$ have unused channels, and (iii) if $(i, i+1)$ is a comparator in layer d and channels i and $i+1$ are unused in layer $d-1$, then channels $i-1$ and $i+2$ (if they exist) are used in layer d.*

Theorem 4. *If C is a sorting network on n channels with depth d, then there is a co-saturated sorting network N on n channels with depth d.*

Proof. Assume C is given. Apply Theorem 3 to find a sorting network S in llnf, containing no redundant comparators except possibly in the last layer.

Let $B_1, \ldots, B_k$ be the $(d-1)$-blocks in S. For $i = 1, \ldots, k-1$, if blocks B_i and B_{i+1} have a free channel, add a comparator between them. (It may be possible to add *two* comparators between these blocks, namely if they both have two channels and none is used in layer $d-1$.) Let N be the resulting network. By Lemma 8, all the comparators added from S to N are redundant, so N is a sorting network; by construction, N satisfies (ii).

If N does not satisfy (iii), then applying Lemma 9 transforms it into another sorting network N' that does. □

In order to get a quantitative measure of the reduction of the search space obtained by adding these constraints, we wrote a simple computer program to generate all co-saturated two-layer suffixes on n channels. The values are given in Table 3.

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16
#	4	4	12	26	44	86	180	376	700	1,440	2,892	5,676	11,488	22,848

n	17	18	19	20
#	45,664	90,976	182,112	363,896

Table 3: Number of distinct co-saturated two-layer suffixes on n channels, for $n \leq 20$.

3.3. Co-subsumption

We now explore a stronger restriction of the search space, based on a different dualization of the ideas in [7]. Given a comparator network C, define $\mathsf{sorts}(C) = \{\bar{x} \in \{0,1\}^n \mid C(\bar{x}) \text{ is sorted}\}$. We say that C *co-subsumes* C', $C \sqsubseteq C'$, if there exists a permutation π such that $\pi(\mathsf{sorts}(C)) \subseteq \mathsf{sorts}(C')$. When we want to make the permutation π explicit, we will write $C \sqsubseteq_\pi C'$. A *generalized comparator* is a comparator (i, j) where $i > j$, and a *generalized comparator network* is a comparator network that may contain generalized comparators. A *generalized sorting network* is a generalized comparator network that sorts all binary inputs. When relevant, we will refer to a *standard* comparator network to emphasize that we are referring to a comparator network with no generalized comparators.

Lemma 10. *If $C \sqsubseteq C'$ and there is a sorting network of the form $N; C$, then there is a generalized sorting network of the form $N'; C'$ of the same size and depth.*

Proof. Construct N' from N by renaming the channels according to π. Given an input $\bar{x}$, $N(\bar{x}) \in \mathsf{sorts}(C)$, and therefore $\pi(N)(\pi(\bar{x})) \in \mathsf{sorts}(C')$. □

In particular, the following corollary states that we can fix the last layer, as we could fix the first one.

Corollary 4. *There is an optimal-depth generalized sorting network whose last layer is L_P^1 (Equation (2)).*

Proof. Let C be the last layer of an optimal-depth (generalized) sorting network. Then $C \sqsubseteq L_P^1$: if $C = \{(i_1, j_1), \ldots, (i_\ell, j_\ell)\}$, take π to be the permutation mapping i_k to $2k-1$ and j_k to $2k$. □

Note that this result cannot be combined with e.g. Parberry's result that there always exists a sorting network whose first layer is L_P^1: by fixing the first layer we dictate what the last layer must be, and conversely.

In general, the networks constructed in these proofs can have reversed comparators, and the usual transformation described in Exercise 5.3.4.16 of [24] for removing these cannot be applied, as it changes the last layers. However, we can also define a *dual* standardization procedure that can be applied in these cases.

Definition 4. *Let C be a generalized sorting network, and define a standard comparator network as follows: let k be the last layer of C where a reversed comparator (i, j) occurs, and interchange i and j everywhere in layers 1 to k. Iterate this procedure until there are no reverse comparators left; we denote the resulting network by C^{st}.*

Lemma 11. *If C is a generalized sorting network, then C^{st} is a sorting network.*

Proof. At each step, we are applying the construction in Lemma 10 to the suffix of C starting with comparator (i, j). □

Corollary 5. *The sorting networks constructed in Lemma 10 and Corollary 4 can be taken to be standard.*

In particular, there is always a sorting network on n channels whose last layer is L_P^1.

3.4. Practical Impact

We now show how to improve the SAT encoding presented in Section 2.4 using the structure of the last layers of a sorting network. We first consider the impact of the necessary conditions stated in Section 3.1.

Consider Lemma 4, which states that non-redundant comparators in the last layer have to be of the form $(i, i+1)$. The following propositional constraint forbids comparators in the last layer that connect non-adjacent channels. The constraint is expressed in terms of a quadratic number of unit clauses:

$$\varphi_1 = \left\{ \neg g^d_{i,j} \;\middle|\; 1 \leq i, i+1 < j \leq n \right\}$$

Corollary 1 generalizes Lemma 4, stating that whenever a comparator at any layer connects two non-adjacent channels, necessarily one of these channels is used at a later layer. This is captured by the following propositional constraint, which adds a single clause for each of the $(n-1)(n-2)/2$ non-adjacent comparators at any given depth ℓ:

$$\varphi_1(\ell) = \left\{ g^\ell_{i,j} \rightarrow \bigvee_{\ell < k \leq d} \mathit{used}^k_i \vee \mathit{used}^k_j \;\middle|\; 1 \leq i, i+1 < j \leq n \right\}$$

Note that indeed $\varphi_1(d) = \varphi_1$, as there is no depth k with $\ell < k \leq d$.

Consider now the penultimate layer, $d-1$. According to Corollary 2, no comparator at this layer can connect two channels more than 3 channels apart. Similar to Lemma 4, we encode this restriction by adding unit clauses for each of the $(n-3)(n-4)/2$ comparators more than 3 channels apart:

$$\varphi_2 = \left\{ \neg g^{d-1}_{i,j} \;\middle|\; 1 \leq i, i+3 < j \leq n \right\}$$

Corollary 2 also states that the existence of a comparator $(i, i+3)$ on the penultimate layer implies the existence of the two comparators $(i, i+1)$ and $(i+2, i+3)$ on the last layer. This is encoded introducing $2(n-3)$ clauses in the following constraint :

$$\varphi_3 = \left\{ g^{d-1}_{i,i+3} \rightarrow g^d_{i,i+1}) \wedge (g^{d-1}_{i,i+3} \rightarrow g^d_{i+2,i+3} \;\middle|\; 1 \leq i \leq n-3 \right\}$$

Corollary 2 also states that the existence of a comparator $(i, i+2)$ on the penultimate layer implies the existence of either of the comparators $(i, i+1)$ or $(i+1, i+2)$ on the last layer. This can be encoded using $n-2$ clauses:

$$\varphi_4 = \left\{ g^{d-1}_{i,i+2} \rightarrow g^d_{i,i+1} \vee g^d_{i+1,i+2} \;\middle|\; 1 \leq i \leq n-2 \right\}$$

We now consider the sufficient conditions from Section 3.2. According to Lemma 7 (ii), we can break symmetries by requiring that there be no adjacent unused channels in the last layer, i.e., that the network be in llnf.

$$\psi_1 = \left\{ \mathit{used}^d_i \vee \mathit{used}^d_{i+1} \;\middle|\; 1 \leq i < n \right\}$$

Essentially, this forces the SAT solver to add a (redundant) comparator between any two adjacent unused channels on the last layer.

The next symmetry break is based on a consideration of two adjacent blocks. There are four possible cases: two adjacent comparators, a comparator followed by an unused channel, an unused channel followed by a comparator, and two unused channels. The latter is forbidden by the symmetry break ψ_1 (and thus not regarded further).

The case of two adjacent comparators is handled by formula ψ^a_2:

$$\psi^a_2 = \left\{ g^d_{i,i+1} \wedge g^d_{i+2,i+3} \rightarrow \left(\mathit{used}^{d-1}_i \wedge \mathit{used}^{d-1}_{i+1}\right) \vee \left(\mathit{used}^{d-1}_{i+2} \wedge \mathit{used}^{d-1}_{i+3}\right) \;\middle|\; 1 \leq i \leq n-3 \right\}$$

This condition essentially forces the SAT solver to add a (redundant) comparator on layer $d-1$, if both blocks have an unused channel in that layer.

The same idea of having to add a comparator at layer $d-1$ is enforced for the two remaining cases of a comparator followed by an unused channel or its dual by ψ_2^b and ψ_2^c, respectively:

$$\psi_2^b = \left\{ \; g^d_{i,i+1} \wedge \neg used^d_{i+2} \rightarrow \left(used_i^{d-1} \wedge used_{i+1}^{d-1} \right) \vee used_{i+2}^{d-1} \;\middle|\; 1 \leq i \leq n-2 \; \right\}$$
$$\psi_2^c = \left\{ \; \neg used^d_{i} \wedge g^d_{i+1,i+2} \rightarrow used_i^{d-1} \vee \left(used_{i+1}^{d-1} \wedge used_{i+2}^{d-1} \right) \;\middle|\; 1 \leq i \leq n-2 \; \right\}$$

The final symmetry break is based on Lemma 9, i.e., on the idea of moving a comparator from the last layer to the second last layer. We encode that such a situation cannot occur, i.e., that whenever we have a comparator on the last layer d following or followed by an unused channel, one of the channels of the comparator is used on layer $d-1$:

$$\psi_3^a = \left\{ \; g^d_{i,i+1} \wedge \neg used^d_{i+2} \rightarrow used_i^{d-1} \vee used_{i+1}^{d-1} \;\middle|\; 1 \leq i \leq n-2 \; \right\}$$
$$\psi_3^b = \left\{ \; g^d_{i,i+1} \wedge \neg used^d_{i-1} \rightarrow used_i^{d-1} \vee used_{i+1}^{d-1} \;\middle|\; 2 \leq i \leq n-1 \; \right\}$$

Defining $\psi_2 = \psi_2^a \wedge \psi_2^b \wedge \psi_2^c$ and $\psi_3 = \psi_3^a \wedge \psi_3^b$, and denoting

$$last = \bigwedge_{i=1}^{4} \varphi_i \wedge \bigwedge_{i=1}^{3} \psi_i$$

we can enrich the SAT encodings expressed by Equations (4) and (5) to obtain

$$\varphi^{last}(n,d) = valid(C_n^d) \wedge \bigwedge_{\bar{x} \in \{0,1\}^n} sorts(C_n^d, \bar{x}) \wedge last \tag{7}$$

$$\varphi_P^{last}(n,d) = valid(C_n^{d-|P|}) \wedge \bigwedge_{\bar{x} \in \mathsf{outputs}(P)} sorts(C_n^{d-|P|}, \bar{x}) \wedge last \tag{8}$$

The constraints $\varphi_1(\ell)$ do not seem to have a positive impact on the performance of the SAT solvers, so we will not use them hereafter.

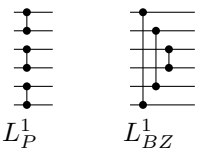
We note that the results from Section 3.3 cannot be applied in this scenario: the proof strategy relies on the fact that there must exist a sorting network whose two first layers have a particular form, and this is incompatible with the construction in Corollary 4.

4. Back to the Front: Permuting Filters

In this section, we come back to Lemma 1 and to the improved SAT encoding described in Section 2.5. When considering the proposition of Equation (5) for a fixed prefix P, the SAT encoding that searches for an extension of P to a sorting network is optimized with respect to the leading zeroes and trailing ones in $\mathsf{outputs}(P)$.

It is a well-known fact that, if P can be extended to a sorting network of depth d, then so can any prefix P' obtained by permuting the channels in P followed by a procedure called *untangling* (which basically turns back comparators that were turned "upside down" in the permutation step – see [30] or Exercise 5.3.4.16 of [24]). Although P' is equally good as P in terms of the number of its (unsorted) outputs and in terms of its potential to extend to a sorting network of depth d, different permutations of P can lead to smaller SAT encodings in light of the optimizations based on leading zeroes and/or trailing ones.

This behavior can already be observed in the first layer: Parberry [30] used the first layer L_P^1 consisting of comparators of the form $(2i-1, 2i)$, for all $1 \leq i \leq \lfloor \frac{n}{2} \rfloor$, whereas Bundala and Závodný [8] employed a first layer L_{BZ}^1, which is a permutation thereof consisting of comparators $(i, n+1-i)$, for all $1 \leq i \leq \lfloor \frac{n}{2} \rfloor$ – see the figure on the right for an example on 6 channels.



We define the *window size* of a vector $\bar{x}$ to be its length minus the number of its leading zeros and trailing ones. Formally, for $\bar{x} = (x_1, x_2, \ldots, x_n)$,

$$ws(\bar{x}) = n - \max\{\ a \mid\ x_1 = x_2 = \cdots = x_a = 0\ \} - \max\{\ b \mid\ x_{n-b+1} = x_{n-b+2} = \cdots = x_n = 1\ \}$$

By the previous considerations, the number of channels for which the output on the channel is not immediately determined by the input to this channel, and which have thus to be explicitly considered in the SAT formula specifying the behavior of the network for an input vector from a set S, is equal to the sum of the window sizes of the vectors in S. Table 4 shows the sum of window sizes of $\mathsf{outputs}(L_P^1)$ and $\mathsf{outputs}(L_{BZ}^1)$ for $3 \leq n \leq 17$ channels: a first layer L_{BZ}^1 always leads to a smaller SAT encoding than the corresponding first layer L_P^1.

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
L_P^1	5	12	44	84	233	408	1,016	1,704	4,013	6,564	14,948	24,060	53,585	85,296	186,992
L_{BZ}^1	4	10	36	72	196	358	876	1,524	3,532	5,962	13,380	22,128	48,628	79,246	171,612

Table 4: Number of channels to consider in the encoding after the first layer.

We have observed that some permutations of a given prefix are better for SAT encodings than others, and that new results on optimal-depth sorting networks derive from fixing 2-layer prefixes. So, we introduce a new criterion on prefixes: select a permutation of the prefix that minimizes the sum of the window sizes of its outputs. This idea is further illustrated in the following example.

Example 3. *Consider the three representations C_1, C_2, and C_3 of a particular 2-layer filter in R_6, presented in Figure 7. Each of these prefixes can be transformed into the other two by permuting channels and untangling, hence they all represent the same filter. The outputs from the prefix C_1, on the left, give a total window size of* 35*. The outputs from the prefix C_2, in the middle, give a total window size of* 34*. The outputs from the prefix C_3, on the right, give a total window size of* 28*.*

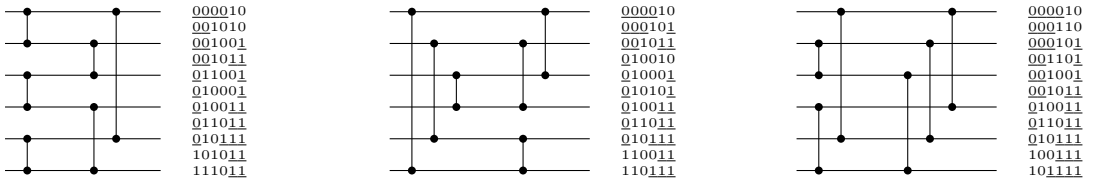


Figure 7: Three permutations of the same 2-layer-prefix on 6 channels. C_1 (left) has L_P^1 as first layer, C_2 (middle) has L_{BZ}^1 as first layer, and C_3 (right) is optimized to minimize total window size in the outputs from the filter. Each filter is accompanied by the set of its non-sorted outputs.

To determine the existence of an n-channel sorting network of depth d we need to consider SAT instances for $|R_n|$ 2-layer prefixes. For each prefix P, we seek a permutation, P', that minimizes the total window sizes of the vectors in $\mathsf{outputs}(P')$. To this end, we apply a very simple evolutionary algorithm, which we briefly describe. Offspring are only created by mutations that swap two randomly chosen channels, followed by untangling. We use the sum of the window sizes of the outputs as the fitness function, and keep the 32 best prefixes in each of 20 iterations. Finally we keep a single prefix with best fitness, which we denote by F_{opt}. The results achieved in this way are not optimal, but provide significant speedups. An implementation of the evolutionary algorithm that we applied to find optimized permutations of the 2-layer prefixes in R_n can be downloaded from `https://github.com/the-kiel/JCSS`.

Table 5 compares the average number of channels to consider for 2-layer prefixes. We denote by $\mathcal{F}_P$ a representation of the 2-layer prefixes from R_n whose first layer is always L_P^1. Note that this representation is not unique, as any given 2-layer prefix with first layer L_P^1 can be permuted to different 2-layer prefixes

with the same first layer. Analogously, we denote by $\mathcal{F}_{BZ}$ a representation of the 2-layer prefixes from R_n whose first layer is always L^1_{BZ}, and by $\mathcal{F}_{opt}$ the representations found using our evolutionary algorithm. The values presented are not unique, as they depend on the specific representations of $\mathcal{F}_P$, $\mathcal{F}_{BZ}$ and $\mathcal{F}_{opt}$ selected.

n	5	6	7	8	9	10	11	12	13	14	15	16	17
$\mathcal{F}_P$	20	43	110	196	456	786	1,651	2,715	5,534	9,094	17,808	28,581	55,314
$\mathcal{F}_{BZ}$	18	40	97	178	402	714	1,480	2,483	5,014	8,406	16,332	26,633	51,221
$\mathcal{F}_{opt}$	17	35	89	156	362	619	1,328	2,168	4,503	7,371	14,711	23,496	46,331

Table 5: Average number of channels (over the complete set of filters) to consider in the encoding after the second layer.

Table 6 shows SAT solving times when proving that no sorting network on 16 channels with 8 layers exists, using the different representations of the 211 prefixes in R_{16}. The evolutionary algorithm required 5 minutes to compute the set $\mathcal{F}_{opt}$. Optimizing the prefixes reduced the overall SAT solving time by factors of circa 4 and 2, and the maximum running times by factors of circa 9 and 4, respectively.

Prefix	Overall time (s)	Maximum time (s)
$\mathcal{F}_P$	22,241	326
$\mathcal{F}_{BZ}$	10,927	150
$\mathcal{F}_{opt}$	5,492	36

Table 6: Impact of permuting the prefix when proving that no sorting network for 16 channels with at most 8 layers exists.

Table 7 exemplifies the impact of the choice of the prefix, together with the improved SAT encoding, on the size of the resulting SAT formula. Here we encoded one permutation of one 2-layer prefix together with 2,000 inputs of minimum window size, which is sufficient to prove unsatisfiability. The new encoding adds a few more variables, and reduces the number of clauses in exchange. Using both an optimized prefix and the new encoding reduces the number of clauses by 67%.

Prefix type	Encoding	# variables	#clauses
$\mathcal{F}_P$	old	115,815	4,861,186
$\mathcal{F}_{BZ}$	old	104,769	4,260,513
$\mathcal{F}_{opt}$	old	89,057	3,438,352
$\mathcal{F}_P$	improved	117,446	2,803,674
$\mathcal{F}_{BZ}$	improved	106,393	2,270,755
$\mathcal{F}_{opt}$	improved	90,513	1,598,509

Table 7: Sizes of the SAT formula depending on choice of prefix and encoding

The idea presented in this chapter – to search for sorting networks that extend a given prefix by first selecting a permutation of the prefix to minimize total window size – plays a central role in the forthcoming chapters.

5. Improved Upper Bounds for 17, 19 and 20-Channel Sorting Networks

In this section, we show how the improvements presented in the previous sections apply to facilitate the search for new upper bounds for sorting networks with 17, 19 and 20 channels. To do this, we exhibit sorting networks for 17 channels and for 20 channels that have one less layer than the previous best known sorting networks for these cases – 10 and 11 layers, respectively. The 11-layer sorting network for 20 channels implies the improved upper bound also for 19 channels, which was previously equal to 12.

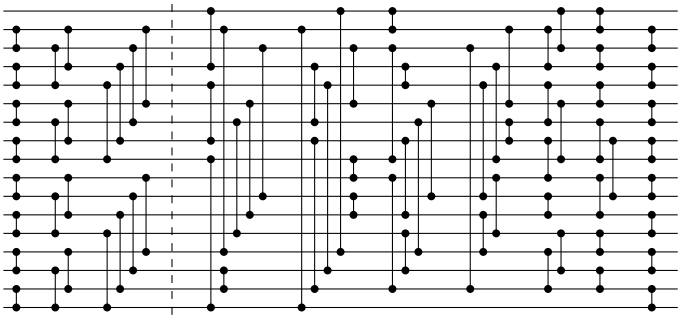


Figure 8: A 10-layer sorting network on 17 channels with a 3-layer Green prefix.

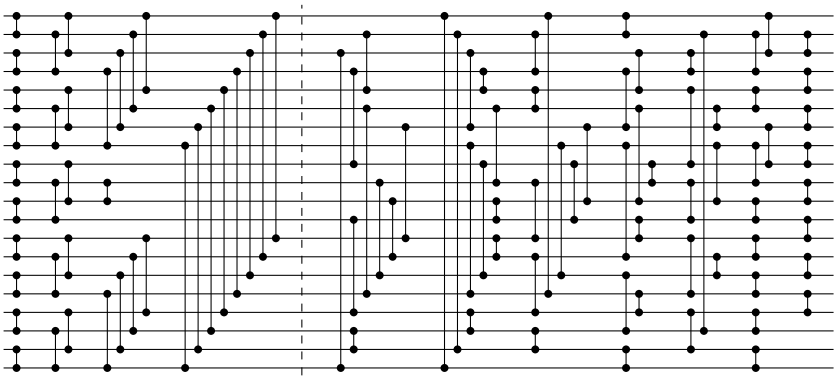


Figure 9: An 11-layer sorting network on 20 channels with the selected 4-layer prefix.

When searching for upper bounds on the depth of sorting networks, it is common practice to search for an extension of a particular chosen prefix of the network. This is in contrast to the quest to prove lower bounds, where we need to show that there is no prefix that extends to a network of the specified depth. For upper bounds, any choice of prefix is fair play. For $n = 17$ channels, we chose a prefix consisting of the first three layers of a Green Filter on 16 channels, leaving one channel unused, and we were able to extend it to the sorting network with 10 layers depicted as Figure 8. The selected prefix, separated by the dashed line, produces 800 outputs, including the 18 sorted ones. This network improves on the best previously known sorting network for 17 channels. For $n = 20$ channels, we chose a 4-layer prefix constructed from a 4-layer Green filter on 16 channels (the top 8 and bottom 8), and where the remaining middle 4 channels are connected with a 3-layer optimal-depth sorting network. We were able to extend this prefix by an additional 7 layers to obtain the 11-layer sorting network depicted as Figure 9, where the selected prefix is again marked by the dashed line. This improves on the best previously known sorting network for 20 channels, which consisted of 12 layers. The selected prefix produces 840 outputs, including 21 sorted ones. As reported in a preliminary paper [19], both of these results can be found using a straightforward SAT encoding.

The techniques presented in this paper massively improve the computational time required to find the depth-10 and depth-11 sorting networks for 17 and 20 channels, respectively. Table 8 presents the impact of the three main contributions: introducing the constraints on the last layers (column one), applying the improved SAT encoding (column two), and optimizing the selected prefix (column three). The bottom row of the table specifies that, without applying the techniques of this paper, the SAT solving times to find the two networks are 9,995 and 45,596 seconds, respectively. The first row of the table specifies that, when applying all three techniques together, the SAT solving times to find the two networks are 17 and 46 seconds, respectively. This constitutes speed-ups by a factor of 587 in the case of 17 channels, and 991, for 20 channels.

Closer examination of the table highlights the impact of each combination of the three techniques. For

example, the third row from the bottom, in comparison to the bottom row, indicates speed-ups in order(s) of magnitude in the search for a depth-10 and depth-11 sorting networks on 17 and 20 channels when introducing only the improved SAT encoding described in Section 2.5. The actual networks found with each configuration of the solver might differ; for example, the networks in Figures 8 and 9, which were found using the improved encoding, do not have an optimized prefix; that in Figure 9 also does not have a co-saturated last layer. If these optimizations are also used, different sorting networks of the same depth will be found.

Symmetry Break (last layers)	Improved Encoding	Opt. Prefix	Time/s	
			$n = 17$	$n = 20$
yes	yes	yes	17	46
yes	yes	no	78	560
yes	no	yes	37	13,199
yes	no	no	265	1,255
no	yes	yes	64	97
no	yes	no	838	412
no	no	yes	3,723	36,159
no	no	no	9,995	45,596

Table 8: Impact of the different optimizations in the time required to find the new sorting networks on 17 and 20 channels.

The software used in all experiments is built on top of MiniSAT 2.20, and can be downloaded at `https://github.com/the-kiel/JCSS`. All experiments were performed on Intel Xeon E5-4640 CPUs clocked at 2.40 GHz.

6. Depth 10 is Optimal for Sorting 17 Inputs

After having found a sorting network for 17 inputs using only 10 layers, we seek to prove its optimality. This task can be decomposed into showing that none of the 609 prefixes in R_{17} can be extended to a 9-layer sorting network [8].

In a first attempt, we tried to prove this by using a straightforward SAT encoding, basically identical to the one introduced in [8], enriched by the symmetry-breaking constraints on the last layers described in Section 3. Using 150 CPU cores, we started a SAT solver for each of the first 150 cases. Whenever a filter was shown unfeasible, we started a solver for the next case which had not yet been considered. 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 a total of $353 \cdot 10^6$ seconds of CPU time, with a maximum of $3 \cdot 10^6$ seconds.

In a second attempt, we applied all three of the optimizations described in this paper: introducing the constraints on the last layers, applying the improved SAT encoding, and optimizing the selected prefix. For the latter, for each of the 609 distinct 2-layer prefixes in R_{17} we chose a permutation minimizing the size of the windows in the outputs of the prefix, as explained in Section 4. The 609 2-layer prefixes of R_{17} and their permutations as considered in this experiment can also be obtained from `https://github.com/the-kiel/JCSS`. This time, we were able to prove that none of the 609 prefixes can be extended to a 9-layer sorting network. The overall CPU time for all 609 instances was $27.63 \cdot 10^6$ seconds, with a maximum running time of 97,112 seconds. Optimizing the prefixes took a total of 25 minutes. This is a speed up of factor 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 5. *Every sorting network on $n \geq 17$ channels has at least 10 layers.*

7. Conclusions

The contributions of this paper can be divided in theoretical insights into, improved methods for, and consequent new results on depth-optimal sorting networks.

Based on the first systematic exploration of what happens at the *end* of a sorting network, we have discovered both necessary and sufficient properties of the last layers of depth-optimal sorting networks. To this end we have introduced the general concept of k-blocks, allowing to reason about the sorting behavior of suffixes of sorting networks. Specifically, we have shown that non-redundant comparators on the last layer must connect adjacent channels, and that there is always a depth-minimal sorting networks with L_P^1 as the last layer. We also discovered a sufficient condition for when adding a comparator to a sorting network results in another sorting network.

We have improved SAT-based methods for deciding whether a prefix can be extended to a sorting network of a given depth in three ways. First, based on the novel theoretical insights, we have added symmetry breaking constraints regarding the last layers of the network. Second, we have optimized the encoding for windowed inputs by trading off a slight increase in variables for vast reductions of the number of clauses and literals. Third, we have shown that not all prefixes are created equal and that, indeed, choosing a permutation of the prefix that maximizes leading zeros and trailing ones leads to a significant reduction of the overall size of the SAT problem. Computer experiments have shown that all three improvements separately translate to significant improvements in SAT solving time, yielding several orders of magnitude when used synergistically.

Exploiting the improved SAT-based methods, we have been able to find sorting networks of lesser depth than previously known for 17, 20, and (consequently) 19 channels, lowering the upper bounds for optimal depth to 10, 11, and 11, respectively. In addition, we have proved optimality of our sorting network on 17 channels using substantial computer resources, determining the optimal depth for this case to be 10, and pushing the boundary of our knowledge one step further.

Acknowledgements

This work was supported by the Israel Science Foundation, grant 182/13, by the Danish Council for Independent Research, Natural Sciences, the German Federal Ministry of Education and Research, combined project 01IH15006A, and by a fellowship within the FITweltweit programme of the German Academic Exchange Service (DAAD). The necessary computational resources were provided by an IBM Shared University Award (Ben-Gurion University) and the Dependable Systems Group, University of Kiel.

References

[1] Miklós Ajtai, János Komlós, and Endre Szemerédi. An $O(n \log n)$ sorting network. In *STOC*, pages 1–9, 1983.

[2] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks and their applications. In Oliver Kullmann, editor, *SAT*, volume 5584 of *LNCS*, pages 167–180. Springer, 2009.

[3] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. Cardinality networks: a theoretical and empirical study. *Constraints*, 16(2):195–221, 2011.

[4] Sherenaz W. Al-Haj Baddar and Kenneth E. Batcher. *Designing Sorting Networks: A New Paradigm*. Springer, 2011.

[5] Kenneth E. Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computing Conference*, pages 307–314, 1968.

[6] Luca Breveglieri and Vincenzo Piuri. Digital median filters. *J. VLSI Signal Process. Syst.*, 31(3):191–206, July 2002.

[7] Daniel Bundala, Michael Codish, Luís Cruz-Filipe, Peter Schneider-Kamp, and Jakub Závodný. Optimal-depth sorting networks. *CoRR*, abs/1412.5302, 2014.

[8] Daniel Bundala and Jakub Závodný. Optimal sorting networks. In Adrian Horia Dediu, Carlos Martín-Vide, José Luis Sierra-Rodríguez, and Bianca Truthe, editors, *LATA*, volume 8370 of *LNCS*, pages 236–247. Springer, 2014.

[9] Chaitali Chakrabarti. Sorting network based architectures for median filters. *IEEE Trans. Circuits and Systems II: Express Briefs*, 40(11):723–727, November 1993.

[10] Chaitali Chakrabarti and Li-Yu Wang. Novel sorting network-based architectures for rank order filters. *IEEE Trans. VLSI Syst.*, 2(4):502–507, 1994.

[11] Sung-Soon Choi and Byung Ro Moon. Isomorphism, normalization, and a genetic algorithm for sorting network optimization. In William B. Langdon et al., editor, *GECCO*, pages 327–334. Morgan Kaufmann, 2002.

[12] Michael Codish, Luís Cruz-Filipe, Michael Frank, and Peter Schneider-Kamp. Sorting nine inputs requires twenty-five comparisons. *J. Computer and System Sciences*, 82(3):551–563, 2016.
[13] Michael Codish, Luís Cruz-Filipe, Markus Nebel, and Peter Schneider-Kamp. Applying sorting networks to synthesize optimized sorting libraries. In Moreno Falaschi, editor, *LOPSTR*, volume 9527 of *LNCS*, pages 127–142. Springer, 2015.
[14] Michael Codish, Luís Cruz-Filipe, and Peter Schneider-Kamp. The quest for optimal sorting networks: Efficient generation of two-layer prefixes. In Franz Winkler, Viorel Negru, Tetsuo Ida, Tudor Jebelan, Dana Petcu, Stephen M. Watt, and Daniela Zaharie, editors, *SYNASC*, pages 359–366. IEEE, 2015.
[15] Michael Codish, Luís Cruz-Filipe, and Peter Schneider-Kamp. Sorting networks: the end game. In Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide, and Bianca Truthe, editors, *LATA*, volume 8977 of *LNCS*, pages 664–675. Springer, 2015.
[16] Michael Codish and Moshe Zazon-Ivry. Pairwise cardinality networks. In Edmund M. Clarke and Andrei Voronkov, editors, *LPAR-16*, volume 6355 of *LNCS*, pages 154–172. Springer, 2010.
[17] Drue Coles. Efficient filters for the simulated evolution of small sorting networks. In Terence Soule and Jason H. Moore, editors, *GECCO*, pages 593–600. ACM, 2012.
[18] N.G. de Bruijn. Sorting by means of swappings. *Discrete Mathematics*, 9(4):333–339, October 1974.
[19] Thorsten Ehlers and Mike Müller. New bounds on optimal sorting networks. In Arnold Beckmann, Victor Mitrana, and Mariya Ivanova Soskova, editors, *CiE*, volume 9136 of *LNCS*, pages 167–176. Springer, 2015.
[20] Timothy Furtak, José Nelson Amaral, and Robert Niewiadomski. Using SIMD registers and instructions to enable instruction-level parallelism in sorting algorithms. In *SPAA*, pages 348–357. ACM, 2007.
[21] Milton W. Green. Some improvements in nonadaptive sorting algorithms. Technical report, Stanford Research Institute, Menlo Park, California, 1969.
[22] Víctor Jiménez-Fernández, Carlos Ventura-Arizmendi, Denisse Martínez-Navarrete, and Francisco González Martínez. Digital architecture for a median filter of image based on sorting network. In *CISST*, pages 56–59. World Scientific and Engineering Academy and Society, 2011.
[23] Hugues Juillé. Incremental co-evolution of organisms: A new approach for optimization and discovery of strategies. In Federico Morán, Alvaro Moreno, Juan J. Merelo Guervós, and Pablo Chacón, editors, *Advances in Artificial Life*, volume 929 of *LNCS*, pages 246–260. Springer, 1995.
[24] Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison–Wesley, 1973.
[25] John R. Koza, Forrest H Bennett III, Jeffrey L. Hutchings, Stephen L. Bade, Martin A. Keane, and David Andre. Evolving sorting networks using genetic programming and the rapidly reconfigurable Xilinx 6216 field-programmable gate array. In *Signals, Systems, and Computers*, volume 1, pages 404–410. IEEE Press, 2-5 November 1997.
[26] Inês Lynce and João P. Marques Silva. Probing-based preprocessing techniques for propositional satisfiability. In *ICTAI*, pages 105–110. IEEE Computer Society, 2003.
[27] Andreas Morgenstern and Klaus Schneider. Synthesis of parallel sorting networks using SAT solvers. In Frank Oppenheimer, editor, *MBMV*, pages 71–80. OFFIS-Institut für Informatik, 2011.
[28] Ian Parberry. *Parallel complexity theory*. Research notes in theoretical computer science. Pitman, 1987.
[29] Ian Parberry. A computer assisted optimal depth lower bound for sorting networks with nine inputs. In *Proceedings of the 1989 ACM/IEEE Conference on Supercomputing*, Supercomputing '89, pages 152–161. ACM, 1989.
[30] Ian Parberry. A computer-assisted optimal depth lower bound for nine-input sorting networks. *Mathematical Systems Theory*, 24(2):101–116, 1991.
[31] Ian Parberry. The pairwise sorting network. *Parallel Processing Letters*, 2:205–211, 1992.
[32] Ian Parberry. Personal communication, 2015.
[33] Christopher Umans. The minimum equivalent DNF problem and shortest implicants. In *FOCS*, pages 556–563. IEEE Computer Society, 1998.
[34] Vinod K. Valsalam and Risto Miikkulainen. Using symmetry and evolutionary search to minimize sorting networks. *J. Machine Learning Research*, 14:303–331, 2013.
[35] David C. Van Voorhis. Toward a lower bound for sorting networks. In Raymond E. Miller, James W. Thatcher, and Jean D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 119–129. Springer, 1972.