University of Bremen

Master of Science

Thesis

Reordering the Inputs to a Sorting Network to Prove Depth Optimality

Author:
Tobias Haslop

Examiners:
Prof. Dr. Sebastian Maneth
Prof. Dr. Sebastian Siebertz

December 6, 2021

Contents

1 Introduction

Sorting is a common task in the design and implementation of algorithms. That is because sorting enforces an invariant, which can be exploited, typically for performance gains. An obvious example is binary search [1, p. 409-416]. Searching an unordered array of elements has linear worst case time complexity. Binary search improves this to logarithmic time but only works if the array is sorted.

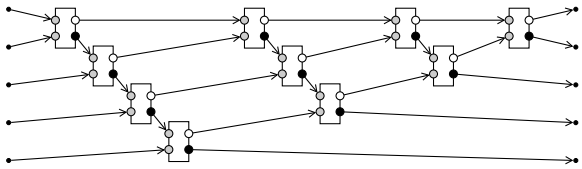
A very common class of sorting algorithms are *comparison sorts*. They only assume a primitive comparison operation that compares two elements and performs a swap if necessary to ensure that eventually the two elements are in the correct order.

It is well known that the best comparison sort runs in $\Omega(n \log n)$ worst case time complexity where n is the number of elements to sort (we will keep that meaning for n throughout the thesis). Quicksort is an example for a comparison sort with this complexity [1, p. 113-122]. Notably sorting algorithms that are not comparison sorts can achieve better, i.e. linear, worst case time complexity for example by exploiting additional assumptions about the data to sort.

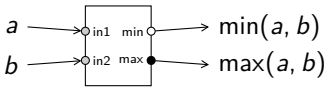
1.1 Sorting Networks

However this thesis is about sorting networks, which represent the subclass of *data-independent* (or data-oblivious) comparison sorts [1, p. 219-221]. A data-independent comparison sort applies the same number of primitive comparison operations on the same data locations in the same order in every run independent of the input data. Unlike most sorting algorithms a sorting network is defined for only one particular number of elements n. Hence a data-independent comparison sort is represented by a family of sorting networks with one sorting network for each value of n.

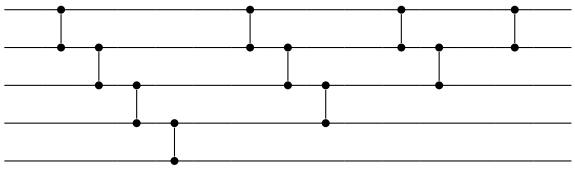
Let us consider the aforementioned Quicksort as well as Bubblesort [1, p. 106-110] to exemplify the notion of a data-independent sort. In the Quicksort algorithm we pick one of the elements as the pivot element p and sort each element x into one of two sets according to whether $x < p$ or $x \geq p$. Then we sort each of the two sets recursively. The pivot element can be picked from an arbitrary data location. At first glance Quicksort may appear to admit a data-independent implementation because in particular we can always pick the pivot element from the same data location. However Quicksort is not data-independent because the sizes of the two sets and thus the number of comparisons in each recursion depend on the value (data) of the original pivot element p.



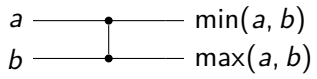
(a) *Bubblesort network*



(b) *Comparison node*



(c) *Bubblesort Knuth diagram*



(d) *Knuth diagram comparator*

Figure 1.1: *Different sorting network depictions of Bubblesort for* $n = 5$*. Note that in both depictions greater elements bubble down instead of up because gates output the greater element in the bottom position.*

Bubblesort works in n iterations. Each iteration performs a chain of primitive comparison operations on adjacent data locations, i.e. location 0 is compared with location 1, 1 with 2, 2 with 3 and so on, such that in the first iteration the greatest value "bubbles" to the last position, after the second iteration the second greatest value is in the second last position and in the end the elements are sorted. Bubblesort is data-independent.

Figure 1.1a shows the sorting network of Bubblesort with $n = 5$. The sorting network has n inputs and n outputs connected by different paths through the network. The primitive comparison operation is denoted by a rectangle with 2 inputs and 2 outputs where one output always contains the minimum and the other the maximum of the two inputs (Fig. 1.1b).

Over the decades a certain type of depiction of sorting networks established itself under the name Knuth diagram (or line representation). In this representation there are n *channels* depicted as straight horizontal lines. The primitive comparison operation is denoted by a vertical line connecting two channels. Figure 1.1c shows the Knuth diagram of Bubblesort corresponding to Figure 1.1a.

We can think of a path through the network as a single memory location that is assigned different values throughout the network run. The advantage of the channel conceptualization of sorting networks over the graph conceptualization is that the n channels unambiguously identify n distinct memory locations indexed by the channels' vertical positions. In the prior representation such an indexing scheme may be mixed up because lines are allowed to cross each other. Actually a comparison between nonadjacent channels necessitates a crossing of the lines in the prior representation. Bubblesort is exceptional in that it only compares adjacent positions, which enables the different representations to look strikingly similar.

Initially this advantage of the channel conceptualization was irrelevant because in their original application sorting networks are hardware circuits [2]. Hence sorting networks were often depicted as a circuit graph like in Figure 1.1a. Due to this we commonly call the primitive comparison operations *comparison gates* (or just *comparator* or *gate*).

1.2 Applications of Sorting Networks

Sorting networks are straightforward to implement as a hardware circuit [1, p. 221]. Additionally any sorting network can be equipped with a routing mechanism to function as a switching network [3].

Due to their data-independence sorting networks always perform the same operations, which ensures that the worst case complexity matches the average case complexity. Besides this performance guarantee data-independence allows an implementation in software without branching instructions. From a security perspective branching may result in different execution times depending on the input data. Sorting networks can prevent this leak of information [4].

Moreover branching instructions are very expensive in pipelined CPUs [5]. Branching instructions are particularly expensive in modern GPU architectures where threads on different branches have to wait for each other to ensure synchronized execution. In fact sorting networks are basic curricular substance in the field of parallel algorithms [6]. For example rank order filters in image processing such as the median filter run a thread per pixel on the GPU and sort the pixel values of its neighborhood [7].

1.3 Optimal Sorting Networks

In parallel complexity theory the two fundamental measures are work and depth complexity [8]. Work complexity counts the number of computation steps, which usually corresponds to the notion of time complexity, whereas depth complexity only counts the computation steps on the longest computation path, which corresponds to the time complexity under idealistic parallelization assumptions. The work complexity of a sorting network is typically referred to as the *size* of the network, that is the number of its comparators. Figure 1.2 shows a parallelized version of the Bubblesort network in Figure 1.1 with depth 7.

While there are many well known practical comparison sorts with $O(n \log n)$ work complexity such as Quicksort, the best known practical data-independent comparison sorts such as the pairwise sorting network family only achieve the slightly worse $O(n \log^2 n)$ work and $O(\log^2 n)$ depth complexity [9]. Actually the AKS family of sorting networks achieves optimal $O(n \log n)$ work and $O(\log n)$ depth complexity, but the large constant factor hiding in the big O notation makes it impractical.

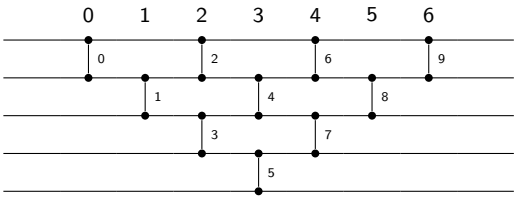


Figure 1.2: *Bubblesort network with size* 10 *and depth* 7

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
s_n	1	3	5	9	12	16	19	25	29	35	39	45 43	51 47	56 51	60 55	71 60	77 65	85 70	91 75
d_n	1	3	3	5	5	6	6	7	7	8	8	9	9	9	9	10	11 10	11 10	11 10

n	21	22	23	24	25	26	27	28	29	30	31	32
s_n	100 80	107 85	115 90	120 95	132 100	139 105	150 110	155 115	165 120	172 125	180 130	185 135
d_n	12 10	12 10	12 10	12 10	13 10	13 10	14 10	14 10	14 10	14 10	14 10	14 10

Table 1.1: *The minimum size (s_n) and minimum depth (d_n) of sortings networks with n channels. For unknown values the cell contains the known lower and upper bound instead. [10]*

It turns out that finding sorting networks of minimal size or depth for very small values of n is already very hard (see Table 1.1). In this thesis we look at the problem of finding sorting networks of optimal depth. Although there are applications for small networks such as as the base case of Quicksort in standard libraries [5], our enterprise is mostly theoretically motivated. Specifically to show optimality we have to prove no sorting network with a smaller depth for some n exists.

This thesis is structured as follows. In Chapter 2 we formally define and relate the graph and channel conceptualizations of sorting networks. Chapter 3 discusses isomorphisms between Sorting Networks, also known as network symmetries. In Chapter 4 we discuss previous work on optimal sorting networks. We look at the approach that yielded the latest depth optimality results, which reduces the problem of the existence of a channel network in Boolean SAT. We consider various minor variations of the state of the art approach in Chapter 5 namely another objective function for prefix optimization, banning "obviously" redundant comparators and utilizing at-most-once cardinality constraints from the literature. Chapter 6 addresses the title of this thesis. We give a SAT encoding of the isomorphic channel network with its inputs in reverse order and discuss the costs and benefits of this encoding. In Chapter 7 we give an alternative SAT encoding of the two networks with reversed inputs. In particular we relate the two encoded channel networks by additionally encoding their graph representation. Further we outline how breaking symmetries introduced by this graph encoding enables us to break symmetries in the encoding of the channel network(s). In Chapter 8 we summarize the results.

2 Formal Models of Comparison Networks

In prior work on small optimal sorting networks the channel conceptualization was used as the main formal model of sorting networks [11]. This is despite sorting networks originally being graphs, as the name network suggests. Here we first formalize sorting networks as graphs before we relate it to a formalization of the channel conceptualization. This necessary because later we extend the Boolean SAT encoding of sorting networks from previous work, which uses channels, by an encoding of the corresponding sorting network graphs.

Unfortunately there is a problem that makes the formalization of sorting networks as graphs slightly awkward: An outgoing arc on a comparison node carries either the minimum or the maximum of the comparison but, it is not clear which of the two it carries. To overcome this problem we define a sorting network as a graph whose nodes have *ports*.

Definition 1 (Port Graph)
A directed port graph is a tuple $(V, P, \text{inports}, \text{outports}, E)$ with

- a set of nodes V,
- a set of ports P,
- functions $\text{inports}, \text{outports} : V \to 2^P$, which provide the set of available input or output ports on a particular node,
- a set of arcs $E \subseteq \{((v_i, p_i), (v_j, p_j)) \mid v_i, v_j \in V, p_i \in \text{inports}(v_i), p_j \in \text{outports}(v_j)\}$. An arc $((v_1, p_1), (v_2, p_2))$ connects the output port p_1 on node v_1 to input port p_2 on node v_2.

We may write $v{:}p$ to refer to port p on node n, e.g. the arc $((v_1, p_1), (v_2, p_2))$ can be written as $(v_1{:}p_1, v_2{:}p_2)$.

Definition 2 (Graph Conceptualization of a Comparison Network)
A comparison network G of size s for n elements is a tuple (S, T, C, E) that defines a directed acyclic port graph $(V, P, \text{inports}, \text{outports}, E)$ where E satisfies the condition that each port has exactly one arc attached and we have

- nodes $V = S \cup T \cup C$
- ports $P = \{\text{in}, \text{out}, \text{in1}, \text{in2}, \text{min}, \text{max}\}$,

- n network-input nodes $S = \{s_0, .., s_{n-1}\}$ with in-degree 0 and out-degree 1, specifically inports$(s_i) = \emptyset$ and outports$(s_i) = \{\text{out}\}$,
- n network-output nodes $T = \{t_0, .., t_{n-1}\}$ with in-degree 1 and out-degree 0, specifically inports$(t_i) = \{\text{in}\}$ and outports$(t_i) = \emptyset$,
- s comparison-gate nodes $C = \{c_0, .., c_{s-1}\}$ with in-degree and out-degree 2, specifically inports$(c_i) = \{\text{in1}, \text{in2}\}$ and outports $= \{\min, \max\}$.

The number of comparison nodes on the longest path in a comparison network is its *depth*. In this thesis we use the letters s and d as the variable names for a network's size and depth respectively.

In Section 3.0.1 we discuss how this definition differs from definitions of comparison networks as graphs in previous work.

A comparison network G permutes an input sequence $x = x_0, .., x_{n-1}$ to the network returning an output sequence $G(x)$ as follows. Each element x_i enters the network at input node s_i, is transported along arcs and finally exits the network at some output node t_j as the element y_j of the network output $G(x) = y = y_0, .., y_{n-1}$. Comparison nodes route the elements arriving at the ports in1 and in2 such that the smaller element always exits the comparison node at port min and the greater element at port max.

Optionally one can use the notion of swaps to define the routing behavior of a comparison node. This has the advantage that the trajectories of each element through the network is unambiguous even if a comparison node compares two equal element. We discuss swaps in 3.0.1.

A sorting network G is a comparison network that sorts all input sequence. A network G sorts an input sequence x if the output $G(x) = y = y_0, .., y_{n-1}$ satisfies $y_i \leq y_{i+1}$ for all i.

2.1 Optimal Sorting Networks

Two comparison nodes with no directed path between them are *parallel*. This parallelity is related to the depth. We may partition the comparison nodes into sets such that all comparison nodes in the same set are parallel to each other. We call a set of parallel comparison nodes a *layer*.

We call a partition of all comparison nodes into layers minimal if no partition of all comparison nodes into fewer layers exists. Recall that the depth is the number of comparison nodes on the longest path through the network. While there may be several different minimal partitions, the number of layers in a minimal partition always is the network's depth. Figure 2.1 shows two different minimal partitions of the same comparison network.

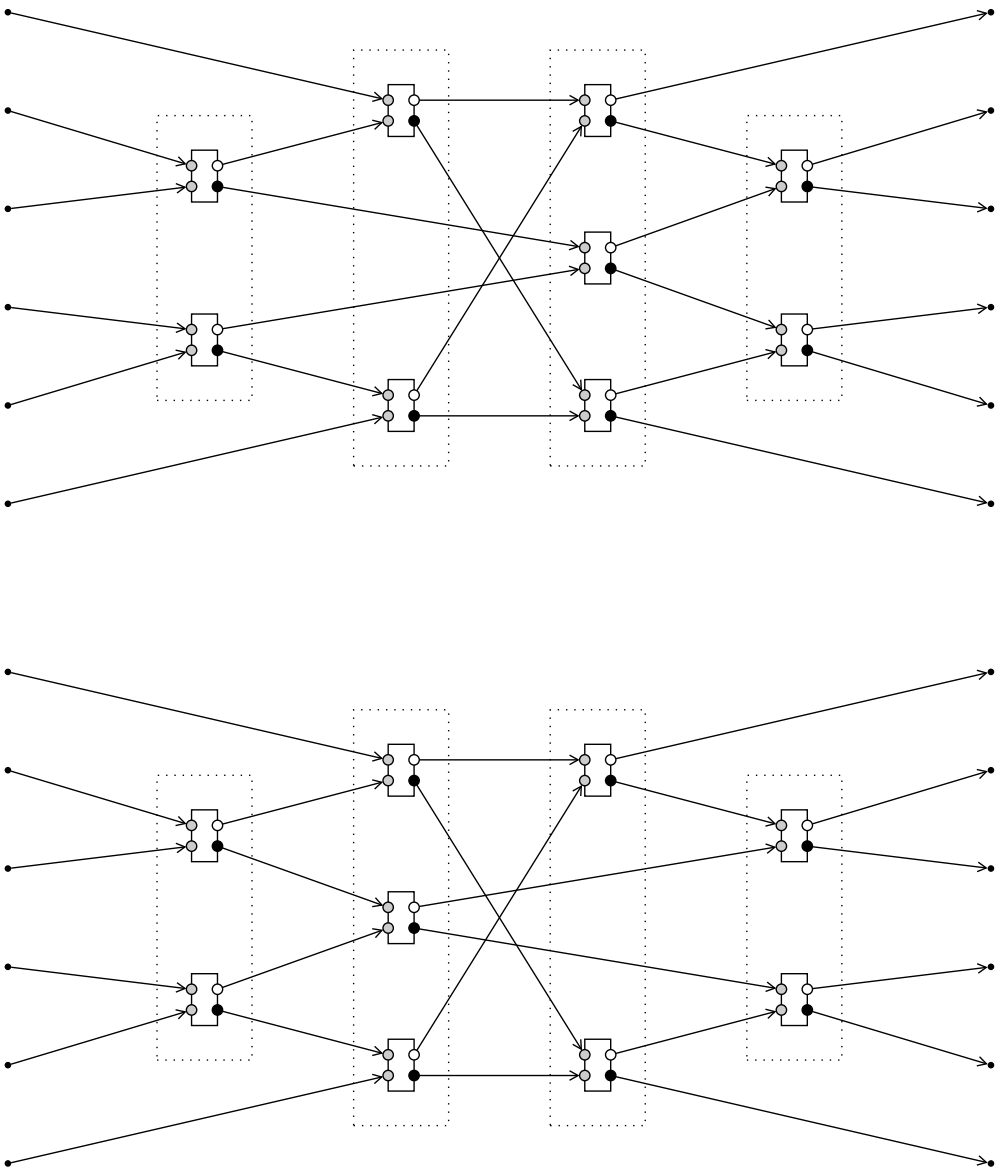


Figure 2.1: *Two different ways to partition the comparison nodes of the same network into layers*

This thesis addresses the depth optimization problem that asks for the minimum depth of a sorting network on n elements. The related size optimization problem asks for the minimum size of a sorting network on n elements. We develop techniques to augment depth optimality proofs for small n. The closest open problems are $18 \leq n \leq 32$, see Table 1.1. Section 4 gives an overview on prior results.

To classify the complexity of sorting network optimization we turn to the related decision problem variants. The optimal depth decision problem asks whether a sorting network on n elements with depth d exists. A well-known suspected boundary between the easily computable and the hardly computable is the one between NP-complete problems and problems in P. If we could verify in polynomial time that a comparison network with certain dimensions sorts, that would locate the decision problems in NP. However Parberry showed that sorting network verification is coNP-complete [12]. With access to a coNP oracle we locate our decision problems in Σ_2^p, which is in the second level of the polynomial hierarchy [13][p. 91-100]. I conjecture sorting network optimization to be Σ_2^p-complete, that is one level above the well known NP-complete problems.

2.2 Channel Conceptualization of Sorting Networks

As already mentioned in Section 1.1 there is another popular formalization of comparison networks, which Bilardi calls the line representation of a comparison network (graph) [14]. Another common name is Knuth diagram, as Knuth's comprehensive introduction to sorting networks belongs to the "early" literature that systematically used this representation [1]. In particular I believe Knuth's work popularized the visual notation for a comparison gate as a vertical line with dots at its endpoints as shown in Figure 1.1d.

In the channel conceptualization of comparison networks there are n channels depicted as horizontal lines that carry elements from left to right through the network. A gate is visualized by a vertical line connecting two channels. The elements carried by channels i and j on the left are sorted such that that on the right channel i carries the smaller element and channel j carries the greater element.

We can think of the horizontal axis in a Knuth diagram as the time. For any point in time there is an intermediate sequence z where z_i is the element carried by channel i at that moment. In that sense channel i corresponds to the ith position of the sequence that is being sorted. In other words a channel represents one data-location.

An in-place sorting algorithm does not require more memory space than already allocated for the input sequence (except for negligible constant overhead, e.g. any temporary swap variables). Obviously at any point in time during a comparison network run there are exactly n data-locations in use. Hence any Knuth diagram can be realized as an in-place algorithm.

Definition 3 (Channel Conceptualization of a Comparison Network)
A comparison network N on n elements with m layers is a sequence of layers $N = L_0, .., L_{m-1}$. Each layer L_k is a set of parallel gates. A gate (i, j) with $i \neq j$ sorts the minimum element to channel i and the maximum element to channel j. Gates are parallel if they work on different channels.

Previous work, primarily that conceptualizing networks with channels, frequently defines the depth of a network as the number of its layers [15, 16]. This differs from our definition of the depth as the length of the longest path. Specifically a comparison network with a non-minimal partition has a greater number of layers than its depth. However we are concerned with comparison networks of minimal depth and hence only consider partitions that are minimal. Therefore the difference is neglectable for our purposes.

2.3 Relating the Graph and Channel Conceptualizations of Networks

Because of the nodes' in- and out-degrees it is always possible to partition the arcs into n arc-disjoint paths such that each input node s_i is connected to some output node t_j by a path [14]. We can think of such a path through a comparison network graph as a single data location that is assigned different values throughout the network run. Equivalently we can think of each such path as a channel, which enables us to relate the graph conceptualization to the channel conceptualization of comparison networks.

Definition 4 (π-Channel Labeling)
Let (S, T, C, E) be a comparison network for n elements. Let π be a permutation of the channel numbers. The function $chan : E \to \{0, .., n-1\}$ is a proper π-channel labeling if there is a path $P = a_0, a_1, .., a_k$ from the network input node $s_{\pi^{-1}(i)}$ to the network output node t_i with $chan(a_0) = chan(a_1) = .. = chan(a_k) = i$ for every channel number $0 \leq i < n$.

We can alternatively/additionally label the ports (instead of the arcs) on each path with the same channel number such that we have $chan(c{:}p) = chan(a) = chan(d{:}q)$ for any arc $a = (c{:}p, d{:}q)$.

Note that for output nodes we have $chan(t_i{:}\text{in}) = i$, whereas for input nodes we have $chan(s_i{:}\text{out}) = \pi(i)$.

Definition 5 (π-Line Representation and π-Graph Representation)
Let $G = (S, T, C, E)$ be a network graph and let N be a channel network. Let there be a partition of C into layers and let $chan$ be a π-channel labeling of G for some π.

We say that N is a line representation of G under that π-channel labeling and partition into layers if the following statement holds: There is a comparison node c in layer k in

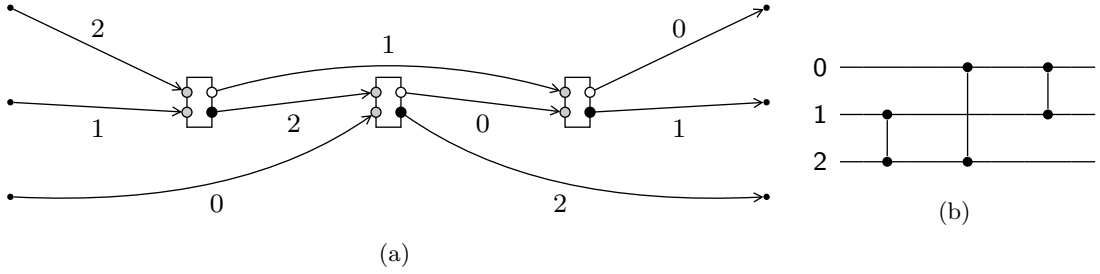


Figure 2.2

G iff there is a gate (i, j) in layer k where $chan(c{:}\min) = i$ and $chan(c{:}\max) = j$. Note that parallel comparison nodes are labeled by distinct channel numbers, which is why we can consider *chan* to be invertible on a per layer basis.

Keep in mind that a graph network may admit several distinct partitions into layers and several distinct π-channel labelings for the same π. But given a graph network, a π-channel labeling and a partition into layers the conditions of the line representation uniquely determine a channel network as *the* line representation. Conversely given a channel network N and a permutation π on the element numbers the conditions determine a graph network G with a partition of its comparison nodes into layers and a π channel-labeling. For this reverse direction we call G the π-graph representation of N.

Figure 2.2 shows a network graph and a channel network. The network graph in Figure 2.2a is the $\left(\begin{smallmatrix}0&1&2\\2&1&0\end{smallmatrix}\right)$-graph representation of the channel network in Figure 2.2b. Likewise the channel network in Figure 2.2b is the $\left(\begin{smallmatrix}0&1&2\\2&1&0\end{smallmatrix}\right)$-line representation of the network graph in Figure 2.2a.

Notably the graph conceptualization of comparison networks is strictly more expressive than the channel conceptualization. For example a network graph without comparison nodes can express any permutation on the elements by connecting the input nodes to the output nodes accordingly. In contrast the channel conceptualization of a comparison network necessarily performs the identity permutation id in the absence of gates.

For the next paragraph we define that a permutation $\pi = \left(\begin{smallmatrix}0&1&\dots&n-1\\ \pi_0&\pi_1&\dots&\pi_{n-1}\end{smallmatrix}\right)$ on the channel numbers may also act on a sequence $x = x_0, x_1, .., x_{n-1}$, i.e. $\pi(x) = x_{\pi_0}, x_{\pi_1}, .., x_{\pi_{n-1}}$.

Let G be a comparison network and let there be a partition into layers and a proper π-channel labeling that yield the π-line representation N. In general N and G are not functionally equivalent, i.e. we may have $N(x) \neq G(x)$ for some input sequence x. However the π-line representation relates the functional behavior of N with that of G through π. Specifically we have $N(x) = G(\pi(x))$ and rearranged $G(x) = N(\pi^{-1}(x))$ for any input sequence x. Recall that for any π-channel labeling *chan* we have $chan(s_i{:}\mathrm{out}) = \pi(i)$.

This establishes that N and G are functionally equivalent if G admits a proper id-channel labeling that (together with the partition into layers) yields N. Remarkably there are

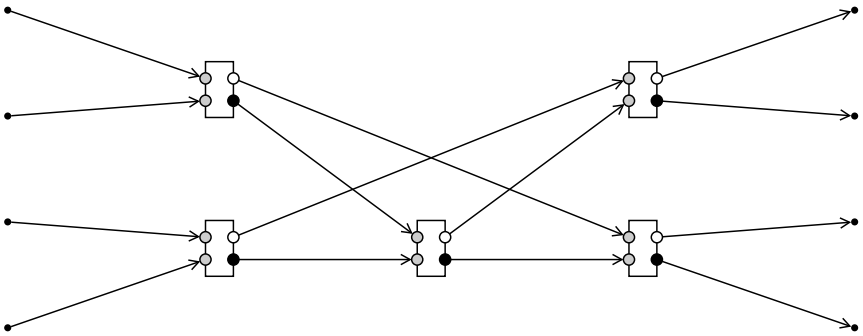


Figure 2.3: *A network that does not admit a proper* id*-channel labeling*

networks that do not admit a proper id-channel labeling even though every input node has a path to every output node (see Fig. 2.3).

Actually the graph of a sorting network necessarily admits a π-channel labeling for any π in particular it admits an id-channel labeling. To understand this let $e = 0, 1, .., n-1$ be the sequence of element numbers in ascending order. Consider the network run $G(\pi(e)) = e$ with the input sequence $\pi(e)$. Intuitively the elements are shuffled by π and then sorted by G. The trajectories of the elements through the network in that run correspond to a π-channel labeling where the element numbers are used as channel numbers. Because the channel numbers are distinct their trajectories are unambiguous.

There are many line representations of the same comparison network and vice versa. In the Sections 3.1.2 and 3.1.3 we discuss the conditions for two channel networks to be π and ρ-line representations of the same network graph. For that we for now ignore the ambiguity of different admissable partitions of the comparison nodes into layers. We resolve this ambiguity for our purposes in Section 4.2.4.

2.4 Standard and Generalized Line Representations

In the channel conceptualization of comparison networks we distinguish between *min-max* gates and *max-min* gates. A gate (i, j) is min-max if $i < j$, otherwise if $j > i$ it is max-min. A network where all gates are min-max is called *standard*. A generalized network may contain both kinds of gates. Likewise a generalized gate is either min-max or max-min and a standard gate is min-max. Figure 2.4 shows depictions of a min-max and a max-min gate.

(a) *Min-max gate* (b) *Max-min gate*

Figure 2.4

Any generalized sorting network of size s on n elements can be transformed into a standard sorting network of the same dimensions [1, p. 238,667-668][6]. Hence it suffices to consider standard networks when looking for sorting networks of minimal depth (or size).

The standardization procedure iterates through the gates from first layer to last. If the gate (i, j) is max-min it switches channel i and j in the current layer as well as all succeeding layers. Any parallel gates from the same layer and any gates from preceding layers remain min-max such that finally the network is standard. If the original generalized network sorts, then the resulting standard network sorts too. Figure 2.5 shows the standardization of a generalized network.

We may transfer the notions of standard and generalized channel networks to apply to channel labelings instead. A comparison node c is standard under some channel-labeling *chan* iff we have $chan(c{:}\min) < chan(c{:}\max)$. In the previous Section 2.3 we observed that the graph of a sorting network admits a π-channel labeling for any π because the sorting of the channel numbers shuffled by π implies a proper π-channel labeling. Clearly sorting the channel numbers ensures that $chan(c{:}\min) < chan(c{:}\max)$ for every comparison node c. Hence the aforementioned π-channel labelings are standard.

Conversely if a network graph admits a standard π-channel labeling, then it sorts any sorted sequence of distinct element shuffled beforehand by π. That is because in a standard labeling we have $chan(c{:}\min) < chan(c{:}\max)$ for each comparison node c. Likewise the value at port $c{:}\min$ in a sorting run is less than the value at port $c{:}\max$. Thus the π-channel labeling models a network run where the input sequence is shuffled beforehand by π.

This means that a standard π-channel labeling corresponds one-to-one to a network run that sorts an input sequence shuffled by π. Scaling this up the $n!$ standard π-channel labelings correspond one-to-one to the $n!$ sorting runs on n distinct elements. Note that if a network sorts all permutations of distinct elements, then it also sorts any sequence, i.e. it is a sorting network. In short a network sorts iff it admits a standard π-channel labeling for any π. .

Moreover we may interpret the standardization of a channel network N as sorting the π-channel labeling at each comparison node in the π-graph representation of N for some π if such a representation exists and N sorts a sequence of distinct elements shuffled by π. Note that any channel network satisfies the first condition for $\pi = \mathsf{id}$. However if the second condition is not satisfied the final channel-labeling *chan* will not be proper as it will violate $chan(t_i) = i$. Sorting networks satisfy both conditions for any π.

In this chapter we defined conceptualizations of comparison networks as graphs or with channels. We related the two conceptualizations of comparison networks by defining a channel labeling on a graph's arcs, a line representation and a graph representation. We considered that comparison nodes may be parallel and discussed the related notions of layers, minimal partitions and depth. We also assessed the complexity of finding a sorting network with optimal depth.

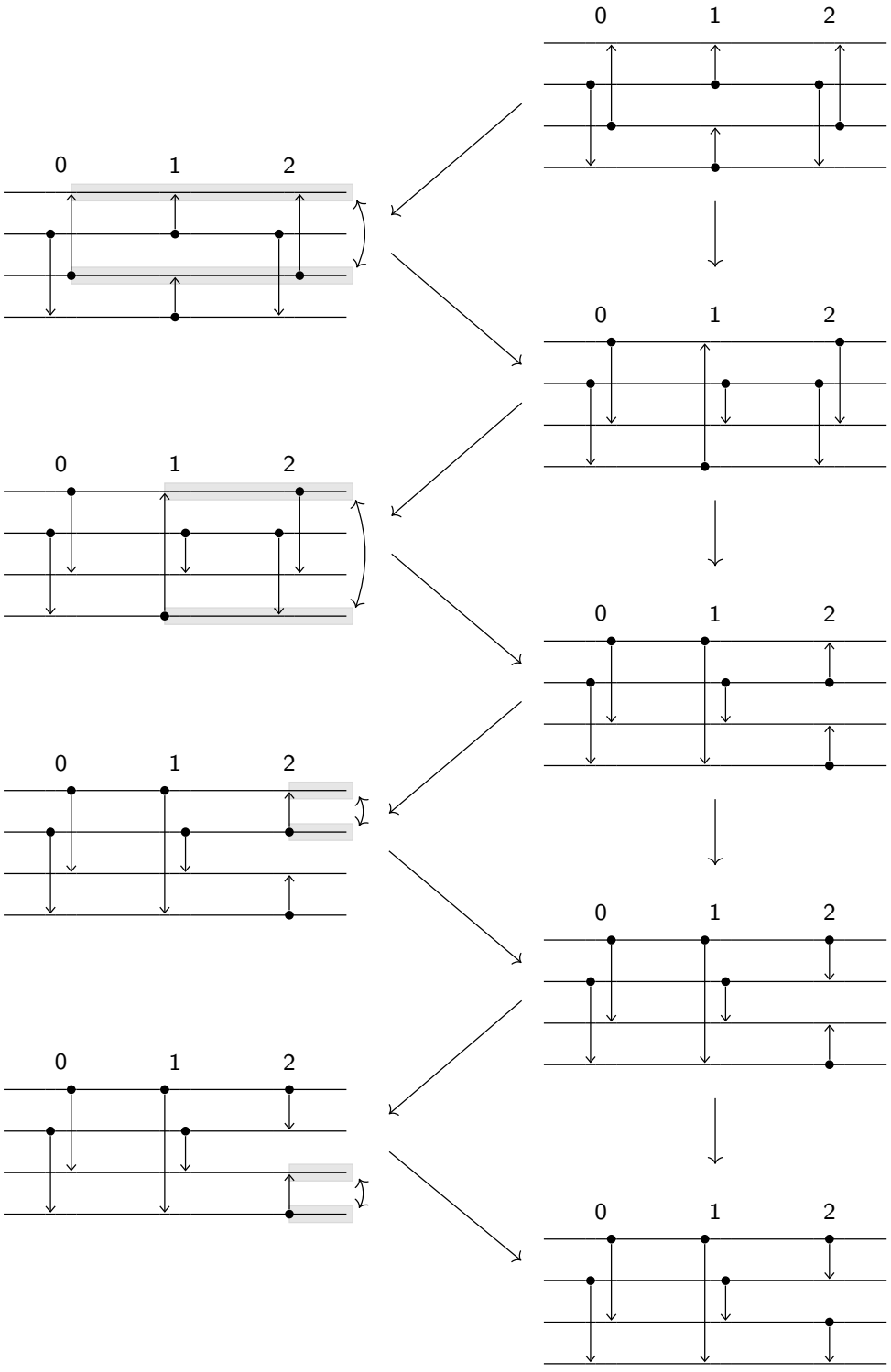


Figure 2.5: *Standardization of a generalized network*

Furthermore we saw that it suffices to consider the class of standard networks, as any generalized network can be standardized to a network with the same dimensions. We learned that we may interpret a transformation on a channel network, e.g. standardization, as a transformation on the channel labeling of the network's graph representation. Finally we observed that the standard channel labelings correspond one-to-one to the sorting runs on n distinct elements.

3 Comparison Network Isomorphism

Definition 6 (Comparison Network Isomorphism)
A comparison network isomorphism between two comparison networks $G = (S, T, C, E)$ and $G' = (S', T', C', E')$ is a triple of node bijections $(p, q, r) : (S \to S', T \to T', C \to C')$ that preserves arcs (with a possibly different input port). To be exact the triple induces a bijection f with those properties.

$$f(v) = \begin{cases} p(v) & v \in S \\ q(v) & v \in T \\ r(v) & v \in C \end{cases}$$

$$\begin{array}{lrcl} \forall v, a, w, b. & (v{:}a, w{:}b) \in E & \to & (\exists! c.(f(v){:}a, f(w){:}c) \in E') \\ \forall v, a, w, c. & (\exists! b.(v{:}a, w{:}b) \in E) & \leftarrow & (f(v){:}a, f(w){:}c) \in E' \end{array}$$

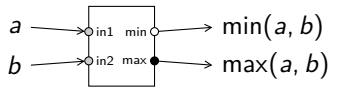
3.0.1 Alternative Definitions of Comparison Networks as Graphs

Note that the definition of comparison networks can easily be changed such that comparison nodes only have a single input port but with exactly two arcs attached. Figure 3.1b illustrates this. Then we could require f to be a proper port graph isomorphism that preserves all components of an arc, even the input port.

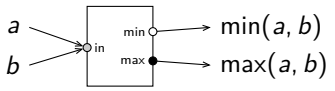
$$\forall v, a, w, b. \qquad (v{:}a, w{:}b) \in E \quad \leftrightarrow \quad (f(v){:}a, f(w){:}b) \in E'$$

On the other hand the first definition supports the model of a swap that may or may not be performed. We say a comparison node performs a swap if the content at port in1 matches max's and in2's content matches min's.

I am only aware of previous work that treats comparison networks as graphs with network input and network output nodes and also distinguishes the input ports [14, 17]. There



(a) *Comparison node with two input ports*



(b) *Comparison node with a single input port*

Figure 3.1

is work that does not distinguish the input ports but it also omits network input and network output nodes [18, 19]. We discuss the implications of this omission in Section 3.1.2 and argue against it in Section 3.1.3. To be clear with the exception of swaps nothing prevents us from not distinguishing the input port without omitting the network input and network output nodes.

In Chapter 4 we encode the existence of a comparison network (graph) as a Boolean SAT problem. In our main encoding we distinguish the input ports. Thus we continue with the clearly more awkward definition that distinguishes the input ports. In Section 7.2.2 we even consider a variant that encodes the swap as a propositional variable. Yet we also explore a variant that does not distinguish between input ports in Section 7.2.1.

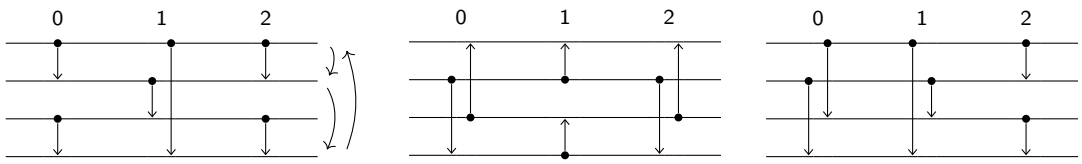
3.1 Isomorphism Refinement

In later chapters we utilize the fact that some networks are isomorphic to prune the search space in order to speed up the search for sorting networks with a certain depth. In this section we introduce and contrast different concepts of isomorphism between networks. Specifically we first introduce concepts from previous work before we settle on the notion used in the later chapters of this thesis.

3.1.1 Isomorphisms between standard sorting networks

There is the notion of a π-sorting networks whose output elements are always in sorted order after permuting them by the same permutation π [6, 19]. We say they sort under permuted outputs.

We may permute the channels of a standard sorting network by any π to obtain a generalized π-sorting network. Standardizing this π-sorting network yields an id-sorting network, i.e. a normal sorting network. This transformation is called permuting and untangling [16, 19]. Figure 3.2 shows an example of permuting and untangling. Permuting the channels of the original standard sorting network in Figure 3.2a by $\left(\begin{smallmatrix}0&1&2&3\\1&3&2&0\end{smallmatrix}\right)$ yields the generalized $\left(\begin{smallmatrix}0&1&2&3\\1&3&2&0\end{smallmatrix}\right)$-sorting network in Figure 3.2b. The untangling of this network is shown in Figure 2.5 and yields the permuted and untangled standard sorting network in Figure 3.2c.



(a) *Original* (b) *Permuted* (c) *Permuted and untangled*

Figure 3.2: *Permuting and untangling a standard sorting network*

Recall from Section 2.4 that we may interpret a channel network transformation such as standardizing as a transformation on the channel labeling of one of the graph representations. In this interpretation permuting and untangling converts between the $n!$ standard channel labelings of a sorting network. Specifically permuting by π and untangling transforms e.g. the ρ-channel labeling of a sorting network's ρ-graph representation into a $(\rho \circ \pi)$-channel labeling.

3.1.2 Isomorphisms between generalized networks

Recall that the standardization procedure permutes the channel numbers of a gate c in its current layer as well as all succeeding layers, but the preceding layers stay the same. I refer to this as twisting the outputs of gate c. Note that outputs are not necessarily twisted to standardize a network, since we may as well twist the outputs of a standard gate to turn it into a max-min gate.

In my bachelor thesis I prove the following result:

" The transformation that permutes channels, twists outputs of comparators and reorders parallel comparators corresponds to an isomorphism between the underlying graphs. " [19]

Basically this statement also applies to the formalization of comparison networks in our thesis, however there is a caveat. Reordering parallel gates refers to the ambiguity that there are different ways to partition the comparison nodes into layers such that some gates may be moved into different layers. We discuss and resolve this ambiguity in Section 4.2.4. Thus we assume that parallel comparators are not reordered.

Another caveat is the following: The formalization of the underlying graphs in that statement actually neither contain any input nodes nor any output nodes and hence also lack any arcs to those nodes. As a consequence the underlying graph representation defined in my bachelor thesis does not coincide with the π-graph representation defined in Section 2.3 in this thesis.

Recall that in a π-channel labeling we have $chan(t_i{:}\text{in}) = i$ for output nodes and $chan(s_i{:}\text{out}) = \pi(i)$ for input nodes. In any example such as the previous we could choose the permutation π in one of the two channel labelings such that the arcs connecting the network input nodes are isomorphic. However this cannot be done for the network output nodes. This way

We could have defined a π-channel labeling differently such that we have $chan(t_i{:}\text{in}) = \pi^{-1}(i)$ for output nodes and $chan(s_i{:}\text{out}) = i$ for input nodes. In this case a network graph G relates to a channel network N by $N(x) = \pi(G(x))$ instead of $N(x) = G(\pi(x))$. In fact there is prior work that that relates network graphs to channel networks this way [17].

We even could have generalized this to a π-ρ-channel labeling. For such a labeling we have $chan(t_i{:}\text{in}) = \rho^{-1}(i)$ for output nodes and $chan(s_i{:}\text{out}) = \pi(i)$ for input nodes such that $N(x) = \rho(G(\pi(x)))$. This solves our discrepancy of the representations as the omission of the network input and output nodes corresponds to leaving π and ρ unspecified. We integrate the statement into our formal framework as follows:

Proposition 7
The transformation between two networks that permutes channels, twists outputs of comparators and reorders parallel comparators exists iff an isomorphism exists between the π-ρ graph representation of one network and the τ-σ-graph representation of the other network for some π, ρ, σ and τ.

Recall that we may interpret the standardization of a generalized network N as sorting the π-channel labeling at each comparison node in the π-graph representation of N for some π, but we have to take care that such a representation exists and the modified channel labeling *chan* satisfies $chan(t_i) = i$, otherwise it is not proper. With the notion of a π-ρ-channel labeling there are always π and ρ for any (intermediate) stage of the modification such that a π-ρ-graph representation exists and its π-ρ-channel labeling is proper.

The result from my bachelor thesis shows exactly which specific transformations preserve the graph structure and thus can be interpreted as a transformation on a graph's π-ρ channel labeling.

3.1.3 Isomorphisms between generalized sorting networks

While a network isomorphism maps between two network graphs preserving the structure, an automorphism is an isomorphism that maps a network to itself. The set of all automorphisms of a network graph is a group under composition. Some refer to automorphisms as symmetries, because like geometrical symmetries automorphisms preserves the structure of the object.

Previous work on finding optimal sorting networks is largely about symmetry breaking [15]. That means that we only consider one network from an isomorphism class of symmetric networks. Specifically we only check whether a single network sorts an input sequence as opposed to checking this for a large class of networks. This makes the search more efficient. We expand on that in Section 4.2.5.

Next we consider an isomorphism that generalizes the symmetry of permuting and untangling on sorting networks. In particular it is more precise than the symmetry of permuting and twisting outputs on generalized comparison networks.

Note that permuting the channels by π relates a sorting network to a π-sorting network [19]. Similarly twisting the outputs of a gate at the end of a network may change a sorting network into one that only sorts under permuted outputs. However ultimately

this thesis is about sorting networks. This is due to our focus on networks in standard form, whose output is already in ascending order if they sort. Specifically Section 4.2.1 discusses a highly effective optimization, which only applies to standard networks.

To this end my bachelor thesis provides another result that relates permuting channels and output twists with input twists. Input twists are dual to output twists. While output twists swap the channel labels of the two channel path segments starting at the output ports of a comparison node, input twists swap the channel labels of the two channel path segments ending at the input ports of a comparison node. In my thesis I prove that the symmetry by twisting inputs is finer than the symmetry by permuting channels and twisting outputs and and conclude:

> A symmetry between two networks is correct under permuted outputs if it preserves to be π-sorting for some π. In constrast a strongly correct symmetry preserves to be π-sorting for the same π. Permuting and twisting outputs are both correct under permuted outputs, while both twisting inputs and permuting parallel comparators are strongly correct. The highlight of the analysis shows that twisting inputs is in fact strongly complete relative to permuting channels and twisting outputs. [That means any two generalized π- and ρ-sorting networks with $\pi = \rho$ that are related by permuting channels and twisting outputs are also related by input twists.] Hence the symmetry by input twists is suited as a more precise substitute for the symmetry by permuting and untangling. [19]

For example the two id-sorting networks in the Figures 3.2a and 3.2c that are related by permuting and untanling are also related by twisting inputs. Specifically twisting the inputs of comparator $(1, 2)$ in layer 1 converts between the two standard networks.

We conclude with a final remark on the difficulty to work with input twists. Given π one can permute the channels of a standard sorting network N by π and standardize the result to obtain another standard sorting network N'. In contrast given π we do not know on the first try how to twist the inputs of each gate to obtain the same network, even though we know it is possible. This is because twisting the outputs of a gate toggles its state between min-max (standard) and max-min. However a gate remains min-max or max-min if its inputs are twisted such that we cannot derive a dual standardization algorithm with input twists instead of output twists.

4 SAT Solver Assisted Optimality Proofs

Table 1.1 in Section 1.3 shows sorting network optimality results from previous work. For each number of elements n there is an upper bound and a lower bound. An upper bound is a sorting network with a certain size or depth. In contrast a lower bound is a proof that there cannot be a sorting network with that size or depth.

Note that both optimal depth and optimal size are monotonically increasing. That is because any sorting network on $n+1$ channels can be transformed to a sorting network on n channels with smaller size and same depth by removing a path from some input node to either the first or last output node together with all comparison nodes on that path [17, 20]. Specifically we have the inequality $s_{n+1} \geq s_n + \lceil \log n \rceil$ since 1972 where s_n is the optimal size of a sorting network network on n elements [20].

We know that the optimal size does not always admit the optimal depth and vice versa, which is first the case for $n = 11$ [21]. Perhaps surprisingly the results so far suggest that size optimization is an even harder problem than depth optimality (see Table 1.1). A brief discussion by Codish et al. points out that the optimal size problem has a much larger search space than the optimal depth problem [22].

In 1969 Knuth compiled a list of known sorting networks, which provided size and depth upper bounds for $n \leq 16$ [1, p. 227, 229]. Their optimality proofs for $n \leq 8$ followed in 1974 [23].

From here on the term proof refers to a computer assisted proof, since the problem is likely located in the second level of the polynomial hierarchy. In particular for lower bounds computers are used to perform an exhaustive search over all comparison networks for a certain number of elements and a certain size or depth to conclude that none of these networks sort.

In 1989, twenty years later Knuth's networks were proven depth optimal for $n = 9$ and $n = 10$ [24] on a supercomputer. As far as I know there was a break of another two decades before work on lower bounds resumed in 2011 [25]. In 2013 Bundala and Závodný proved the sorting networks from 1969 depth-optimal for $11 \leq n \leq 16$ [15].

In 2014 Ehlers and Müller gave sorting networks for 17, 19 and 20 elements with depth 10, 11 and 11 respectively [26]. They proved the depth 10 optimal for $n = 17$ shortly after in 2015 [27]. Sorting networks for 18, 21 and 22 elements with depth 11, 11 and 12 respectively had been given by Baddar and Batcher in 2009 [28]. In 2017 Ehlers gave sorting networks for 23 and 24 elements both with depth 12 [29].

This thesis aims to improve computer assisted proofs of depth lower bounds. Specifically we aim to improve Ehlers' latest work on depth lower bounds [11], which in turn is an improvement of Bundala's and Závodný's work. In their work they reduce the optimal depth decision problem to Boolean SAT and solved it with a Boolean satisfiability solver. I want to point out that SAT solving is not the only exhaustive search technique and that there have been computer assisted depth lower bound proofs without SAT solvers albeit only up to $n \leq 12$ [30, 31].

The size optimality proof of Knuth's networks for $n = 9$ and $n = 10$ from 2014 also builds on Bundala's and Závodný's SAT solver approach [22]. However while the most recent optimality proof of the size of Knuth's networks for $n = 11$ and $n = 12$ similarly draws on insights from previous work, it discards the SAT solver approach in favor of a custom dynamic algorithm [17]. This indicates that the SAT solver approach is not necessarily optimal. Though the SAT solver approach may over time profit from advances in the field of SAT solving whereas a custom algorithm only improves by being run on better hardware. The main insights for improving lower bounds are the different kinds of comparison network symmetries that allow to prune the search space to speed up the search (see Section 4.2.5).

Unlike lower bounds, which require an exhaustive search, the approach to finding upper bounds is quite different. Non-exhaustive search techniques such as evolutionary algorithms are a popular approach for size upper bounds in particular [18, 32, 33, 10]. Further since no exhaustive search is required, it is common practice to guess large parts of the network [29]. There are even interactive design approaches that can be described as handcrafting [34].

4.1 SAT Solver Preliminaries

In Section 4.2 we describe the techniques that yield the latest results on depth lower bounds. Since these techniques employ a SAT solver, this section gives a brief introduction to Boolean Satisfiability Solving.

The Boolean Satisfiability problem (SAT) asks whether there is a satisfying assignment for a propositional logic formula. In the context of Satisfiability such formulas are converted into conjunctive normal form $\bigwedge_i \bigvee_j l_{i,j}$ where $\bigvee_j l_{i,j}$ are called *clauses* and $l_{i,j}$ are *literals*. A literal is a variable that may or may not be negated. Let x be a variable, then we call x a positive literal and $\neg x$ a negative literal.

An *assignment* maps every variable to a Boolean truth value (true or false). A *satisfying* assignment of a formula φ is a mapping such that φ with the variables substituted evaluates to true. A *partial* assignment is a mapping of some but not all variables. Note that e.g. the partial assignment $x_1 \mapsto$ true satisfies the formula $x_1 \vee x_2$ but cannot satisfy $\neg x_1 \wedge x_2$ invariant of the value of x_2.

The Cook-Levin theorem stated SAT as the first NP-complete problem. Not least due to this prominence there has been an ongoing effort to develop specialized programs to solve SAT problems. The progress of such SAT solvers is tracked in yearly SAT-competitions [35].

Even modern SAT solvers are based on the Davis-Putnam-Logemann-Loveland algorithm (DPLL) [36] from 1960. The algorithm guesses truth values and retracts the previous guess if the corresponding partial assignment cannot be solved or both true and false have been guessed for the current variable. Additionally the DPLL algorithm identifies a situation where guessing is suboptimal: If a clause is not yet satisfied and only one of its literals has not been mapped to a truth value yet, then DPLL maps the truth value of the literal's variable such that the clause is satisfied. This is called *Unit Propagation*.

Conflict Driven Clause Learning (CDCL) improves DPLL [36]. If a clause cannot be satisfied by a partial assignment, a CDCL solver identifies variables involved in that 'conflict'. We consider an example from an accessible talk on SAT solvers [37].

Let the formula be:

$$
\begin{aligned}
&(x_1 \vee x_2) \\
\wedge\, &(x_1 \vee x_3 \vee x_8) \\
\wedge\, &(\neg x_2 \vee \neg x_3 \vee x_4) \\
\wedge\, &(\neg x_4 \vee x_5 \vee x_7) \\
\wedge\, &(\neg x_4 \vee x_6 \vee x_8) \\
\wedge\, &(\neg x_5 \vee \neg x_6) \\
\wedge\, &(x_7 \vee \neg x_8) \\
\wedge\, &(x_7 \vee \neg x_9 \vee x_{10})
\end{aligned}
$$

A DPLL algorithm may guess or unit propagate (UP) the partial assignment

$x_7 \mapsto \text{false}$	
$x_8 \mapsto \text{false}$	(UP clause $x_7 \vee \neg x_8$)
$x_9 \mapsto \text{true}$	
$x_{10} \mapsto \text{true}$	(UP clause $x_7 \vee \neg x_9 \vee x_{10}$)
$x_1 \mapsto \text{false}$	
$x_2 \mapsto \text{true}$	(UP clause $x_1 \vee x_2$)
$x_3 \mapsto \text{true}$	(UP clause $x_1 \vee x_3 \vee x_8$)
$x_4 \mapsto \text{true}$	(UP clause $\neg x_2 \vee \neg x_3 \vee x_4$)
$x_5 \mapsto \text{true}$	(UP clause $\neg x_4 \vee x_5 \vee x_7$)
$x_6 \mapsto \text{false}$	(UP clause $\neg x_5 \vee \neg x_6$)
$x_6 \mapsto \text{true}$	(UP clause $\neg x_4 \vee x_6 \vee x_8$)

In this case a DPLL algorithm retracts the guess that $x_1 \mapsto \text{false}$, since x_6 cannot be both true and false, and instead guess $x_1 \mapsto \text{false}$. However with CDCL the algorithm analyzes the conflict and observes that e.g. the following partial assignment already leads to this conflict.

$$
\begin{aligned}
x_4 &\mapsto \text{true} \\
x_7 &\mapsto \text{false} \\
x_8 &\mapsto \text{false}
\end{aligned}
$$

By the contrapositive if an assigment causes no conflicts, then it satisfies the clause $\neg x_4 \vee x_7 \vee x_8$. We say that the algorithm *learns* this clause, i.e. we add it to the other clauses that need to be satisfied. The algorithm then jumps back depending on the

learned clause. In our example after we guess that $x_7 \mapsto$ false unit propagation in the learned clause now infers that $x_4 \mapsto$ false. In contrast the plain DPLL algorithm spends a lot of time guessing (and backtracking!) x_9 and x_1 even though they do not cause the conflict.

Too many clauses slow down unit propagation, hence SAT solvers apply different schemes to identify useful clauses to keep and useless clauses to discard. Similarly there are heuristics for the order in that we guess the truth values of the variables. A static order is not the best. One successful heuristic is Variable State Independent Decaying SUM (VSIDS) [38]. Further we may consider learned clauses to be progress of the SAT solver (in addition/in contrast to the DPLL search tree). A common tactic of SAT solvers are restarts that discard the search tree but keep learned clauses.

For an overview of how much speedup each of the aforementioned techniques yields see [39].

Since clauses are added by conjunctions, adding new clauses to a formula never increases its number of satisfying assignments. Therefore we also call clauses *constraints*.

Boolean SAT solving is difficult, because the solver may have to guess and possibly backtrack, which costs a lot of time. In that respect the number of variables are an upperbound for the worst case. Of course techniques like unit propagation save the solver from having to guess. Whether the solver actually has to guess a lot depends on the problem instance. It may be that the solver does not need to guess at all.

Adding additional clauses tends to speed up the search if they cause propagation and save the solver from guessing. Of course the solver could always guess right on the first try in theory. On the other hand the speed of some recurring steps in the algorithm like unit propagation are linked to the number of clauses such that adding unecessarily many clauses slows the solver down. Similarly adding variables may enable the solver to guess wrong and waste time.

Finally note that the variables in an encoding determine which intermediate solving states the solver can be in, since an intermediate solving state is a partial assignment that maps the variables to truth values. Choosing the wrong variables for an encoding may prevent a solver from backtracking to an adequate intermediate state. Conversely choosing the right variables may enable a solver to backtrack adequately.

4.1.1 Symmetry Breaking

As mentioned in Section 3.1.3 sorting networks are rich in symmetries. In this thesis we use the term symmetry to refer to an equivalence relation on comparison networks that divides comparison networks of the same depth and on the same number of elements into equivalence classes or symmetry classes.

We only refer to correct sorting network symmetries as symmetries. A correct sorting network symmetry has the property that in each equivalence class either all or none of the comparison networks are sorting networks. Due to this correctness it suffices to determine whether a single comparison network from a symmetry network sorts, because this implies whether every other network from that class sorts.

In this thesis we reduce the optimal depth decision problem to Boolean SAT such that a satisfying assignment of a formula φ corresponds to a comparison network, specifically a sorting network.

We do not want to explore the SAT solver to explore all assignment (comparison networks) from a symmetry class. Instead we only want the solver to only consider a single representative network from each symmetry class. To this end we may add a symmetry breaking constraint ψ such that $\varphi \wedge \psi$ has fewer satisfying assignments than φ.

- A symmetry breaking constraint ψ is *correct* if there is at least one assignment from each symmetry class that satisfies ψ.
- A symmetry breaking constraint ψ is *complete* if there is at most one assignment from each symmetry class that satisfies ψ.
- A correct symmetry constraint that is not complete is called a *partial* symmetry breaking constraint.

The idea is that the symmetry breaking constraint causes the SAT solver to backtrack earlier if the assignment is not the representative assignment. This reduces the redundant exploration of other networks in the same symmetry class, which is particularly effective for solver runs that prove unsatisfiability. Keep in mind that a symmetry breaking constraint should not have too many clauses, as this may impact the solving time negatively.

4.1.2 Cardinality Constraints

For the next sections we introduce the constraints eo, amo and alo whose satisfying assignments satisfy exactly one, at most one or at least one of a given set of literals $\{x_0, .., x_m\}$ respectively.

$$\text{eo}(\{x_0, .., x_m\}) = \text{alo}(\{x_0, .., x_m\}) \wedge \text{amo}(\{x_0, .., x_m\}) \tag{4.1}$$

$$\text{amo}(\{x_0, .., x_m\}) = \bigwedge_{0 \leq i < j \leq m} (\neg x_i \vee \neg x_j) \tag{4.2}$$

$$\text{alo}(\{x_0, .., x_m\}) = \bigvee_{0 \leq i \leq m} x_i \tag{4.3}$$

4.2 State of the Art

The state of the art approach to finding depth lower bounds reduces the sorting network depth decision problem to the Boolean satisfiability problem. Recall that the sorting network depth decision problem asks whether there is a sorting network on n elements with depth d for a given n and d.

The reduction exploits the zero-one principle.

Theorem 8 (Zero-One Principle)
If a network with n inputs sorts all 2^n sequences of 0s and 1s into nondecreasing order, then it sorts any arbitrary sequence of n numbers into nondecreasing order [1, p. 223].

We encode a comparison network, a standard channel network to be exact, on n channels of depth d with a Boolean formula such that a satisfying assignment of that formula corresponds to a *sorting* network with those dimensions.

First we introduce basic definitions that we use throughout the next sections.

- There is a set of n element numbers $C = \{0, .., n-1\}$. We name the set C because most of the time we use them as channel numbers. Likewise they number the n network input nodes or the n network output nodes.
- There is a set of the d layer numbers $L = \{0, .., d-1\}$.
- There is a set of stages $S = \{\text{in}, \text{out}\} \cup L$, which extends the d layers stages by a network input stage and a network output stage. There is a total order on values in S:
$$\text{in} < 0 < .. < d-1 < \text{out}$$
- Sometimes we want to refer to element values or arcs between two immediately consecutive stages. Hence we define the set $B = \{0, .., d\}$ to number the network segments that are between two stages. Number 0 refers to the segment between the network input stage and the first layer etc.. The functions $\text{stBef}, \text{stAft} : B \to S$ yield the stage before or after that segment respectively. Likewise the functions $\text{bef}, \text{aft} : L \to B$ refer to the segment before or after some layer.

Next we introduce three kinds of variables that we use in the encoding:

- The variable $g^k_{i,j}$ models a comparator between channel i and channel j in layer k.
$$\mathcal{G} = \{g^k_{i,j} \mid k \in L, i, j \in C, i < j\}$$
- The variable nc^k_i models that channel i in layer k is not used by any comparator.
$$\mathcal{NC} = \{nc^k_i \mid k \in L, i \in C\}$$

- The variable $v_{i,x}^k$ models the element value on channel i in the kth segment between immediately consecutive stages/layers in the network run of the input sequence x.

$$\mathcal{V} = \{v_{i,x}^k \mid x \in \{0,1\}^n, k \in B, i \in C\}$$

Our first constraint ensures that in each layer k every gate is parallel, i.e. no two gates use the same channel i. Additionally a channel i is either used by a gate in layer k or $nc_i^k \mapsto$ true.

$$\bigwedge_{k \in L} \bigwedge_{i \in C} \text{eo} \begin{pmatrix} \{g_{i,j}^k \mid j \in C, i < j\} \\ \cup \{g_{h,i}^k \mid h \in C, h < i\} \\ \cup \{nc_i^k\} \end{pmatrix} \tag{4.4}$$

The next constraints realize the functional behavior of each layer k. Constraint (4.5) ensures that the outputs of a gate (i, j) are the minimum and maximum of its inputs. Constraint (4.6) ensures that the element value on channel i is the same before and after layer k if i is not used by a gate in k.

$$\bigwedge_{x \in \{0,1\}^n} \left(\begin{array}{ll} \displaystyle\bigwedge_{g_{i,j}^k \in \mathcal{G}} \left(g_{i,j}^k \rightarrow \begin{pmatrix} \left(v_{i,x}^{\text{aft}(k)} \leftrightarrow (v_{i,x}^{\text{bef}(k)} \wedge v_{j,x}^{\text{bef}(k)})\right) \\ \wedge \left(v_{j,x}^{\text{aft}(k)} \leftrightarrow (v_{i,x}^{\text{bef}(k)} \vee v_{j,x}^{\text{bef}(k)})\right) \end{pmatrix} \right) & (4.5) \\ \displaystyle\bigwedge_{nc_i^k \in \mathcal{NC}} \left(nc_i^k \rightarrow \left(v_{i,x}^{\text{aft}(k)} \leftrightarrow v_{i,x}^{\text{bef}(k)} \right) \right) & (4.6) \end{array} \right)$$

Additionally we fix the input sequence and the output sequence for each network run such that the network sorts. That is for an input sequence x we set the truth values of $v_{i,x}^0$ to x_i and the truth values of $v_{i,x}^d$ to y_i where $y = \mathsf{sort}(x)$.

4.2.1 Encoding of Exponential Size

Note that there are 2^n Boolean input sequences such that there would be $2^n \cdot n \cdot d$ element value variables. In practice we only model the network runs of a subset of all input sequences saving many variables and clauses, because an unsatisfiable formula that only models a subset of all runs cannot be extended to a satisfiable formula that models all runs.

With this subset approach we prove a lower bound if the formula is unsatisfiable. On the other hand finding a satisfying assignment gives us a comparison network, which we know sorts at least a certain subset of all input sequences. To prove an upper bound we are short of showing that this network sorts *all* inputs.

We can avoid wasting work by verifying whether the network sorts all inputs. Although sorting network validation is coNP-complete, the verification is comparatively fast especially since we work with very small values for n [12]. Otherwise if there is a counterexample, i.e. an input sequence x that is not sorted, then we add the variables and clauses

that simulate the run of x to the formula and solve again to either obtain a lower bound or synthesize a new network [26]. The previous implementation use SAT solver to find a counterexample, since that is NP-complete. If the network sorts we obtain an upper bound.

This strategy is known more generally as Counterexample Guided Inductive Synthesis [40]. For this purpose there are *incremental* SAT solvers, which after finding a satisfying assignment accept additional clauses to find another satisfying assignment under more constraints. The solver benefits from clauses that it learned in its previous runs [41].

However experience shows that initially encoding more comparison network runs than necessary, but not too many, is the fastest way to prove lower bounds [27]. Therefore we actually strive to avoid the CEGIS loop.

4.2.2 Subnetwork Optimization

An important observation is that leading zeros from an input sequence remain on the first channels in a standard network, because their value is already minimal. Likewise trailing ones remain on the last channels. Thus for a given input sequence of the form $x = 0^a y 1^{n-1-b}$ it suffices to consider a subnetwork, i.e. the network without the first a and last b channels.

Since elements from this input only swap channels within that subnetwork we can dispense with $(a + b) \cdot d$ element value variables and the clauses from the Constraints (4.5) and (4.6) that involve the disregarded channels.

This may be called subnetwork optimization, a term originally introduced by Bundala and Závodný [15]. Ehlers calls the middle part y the window of the sequence x. For the window size $|y|$ we have $a+b+|y| = n$. The window size determines how many variables and clauses the subnetwork optimization saves. Thus Ehlers prefers input sequences with minimal window size [27, 16][11][p. 122]. In the next section we calculate the exact number of variables and clauses that are added to encode the network run of sequence with a certain window size.

4.2.3 Improved Encoding

This section covers an improvement of the encoding by Ehlers and Müller [27, 16] [11][p. 123]. Specifically we introduce additional variables that increase the propagation of variables and reduce the number of clauses significantly.

The variables $\min_{i,j}^k$ and $\max_{i,j}^k$ model that channel i or j is used as a min or max channel of a gate within the subnetwork delimited by i and j. Ehlers et al. refer to these variables as oneUp and oneDown variables, because a min (max) channel is connected to one channel below (above) it in a Knuth diagram [16]. Specifically

- $\min_{i,j}^k$ indicates there is some gate $g_{i,l}^k$ with $i < l \leq j$ in layer k
- $\max_{i,j}^k$ indicates there is some gate $g_{l,j}^k$ with $i \leq l < j$ in layer k

The sets $\mathcal{N}$ and $\mathcal{X}$ contain all mi$\mathcal{N}$ and ma$\mathcal{X}$ variables respectively.

$$\mathcal{N} = \{\min_{i,j}^k \mid k \in L, i, j \in C, i \leq j\}$$
$$\mathcal{X} = \{\max_{i,j}^k \mid k \in L, i, j \in C, i \leq j\}$$

The sets $\mathcal{N}_{l,u}$ and $\mathcal{X}_{l,u}$ contain the variables that model a min or max channel usage in the subnetwork delimited by channels l and u.

$$\mathcal{N}_{l,u} = \{\min_{i,u}^k \mid k \in L, i \in C, l \leq i \leq u\}$$
$$\mathcal{X}_{l,u} = \{\max_{l,j}^k \mid k \in L, j \in C, l \leq j \leq u\}$$

In particular $\min_{i,n-1}^k$ and $\max_{0,i}^k$ model whether i is used as a min or max channel in layer k in the whole network. Further note that $\mathcal{N} \cap \mathcal{G} \cap \mathcal{X} = \{g_{i,j}^k \in \mathcal{G} \mid j - i = 1\}$ because we have $\min_{i,i+1}^k = g_{i,i+1}^k = \max_{i,i+1}^k$.

The $\min_{i,j}^k$ and $\max_{h,i}^k$ variables both model the existence of a gate using channel i in layer k, i.e. if the min or max variables maps to true, then one of several candidate gate variables maps to true. We define gates : $\mathcal{N} \cup \mathcal{X} \to 2^{\mathcal{G}}$ to return the appropriate set of candidate gate variables.

$$\text{gates}(\min_{i,j}^k) = \{g_{i,l}^k \mid l \in C, i < l \leq j\}$$
$$\text{gates}(\max_{i,j}^k) = \{g_{l,j}^k \mid l \in C, i \leq l < j\}.$$

Equipped with this definition we can constrain the min and max variables to be consistent with their intended semantics.

$$\bigwedge_{\min_{i,j}^k \in \mathcal{N}} \left(\min_{i,j}^k \leftrightarrow \bigvee_{g_{i,l}^k \in \text{gates}(\min_{i,j}^k)} g_{i,l}^k \right) \tag{4.7}$$

$$\bigwedge_{\max_{i,j}^k \in \mathcal{X}} \left(\max_{i,j}^k \leftrightarrow \bigvee_{g_{l,j}^k \in \text{gates}(\max_{i,j}^k)} g_{l,j}^k \right) \tag{4.8}$$

Note that for all i in C and all k in L we have $\min_{i,i}^k \leftrightarrow$ false and $\max_{i,i}^k \leftrightarrow$ false such that initially the solver infers by unit propagation that in a satisfying assignment both $\min_{i,i}^k$ and $\max_{i,i}^k$ must map to false. Actually we have only introduced the $\min_{i,i}^k$ and $\max_{i,i}^k$ variables to enable a uniform formal treatment of edge cases. These variables may be omitted in an actual encoding.

We observe how the new variables simplify the Constraint (4.5). As this is a bit technical, we consider the conjunctive normal form of Constraint (4.5). We consider a single

input sequence x where l is the first and u is the last channel of the corresponding subnetwork.

$$\bigwedge_{\substack{g^k_{i,j}\in\mathcal{G}\\ l\le i<j\le u}} \begin{pmatrix} \left(\neg g^k_{i,j} \vee \quad v^{\mathrm{aft}(k)}_{i,x} \vee \neg v^{\mathrm{bef}(k)}_{i,x} \vee \neg v^{\mathrm{bef}(k)}_{j,x}\right) \\ \wedge \left(\neg g^k_{i,j} \vee \neg v^{\mathrm{aft}(k)}_{i,x} \vee \quad v^{\mathrm{bef}(k)}_{i,x}\right) \\ \wedge \left(\neg g^k_{i,j} \vee \neg v^{\mathrm{aft}(k)}_{i,x} \vee \quad v^{\mathrm{bef}(k)}_{j,x}\right) \\ \wedge \left(\neg g^k_{i,j} \vee \neg v^{\mathrm{aft}(k)}_{j,x} \vee \quad v^{\mathrm{bef}(k)}_{i,x} \vee v^{\mathrm{bef}(k)}_{j,x}\right) \\ \wedge \left(\neg g^k_{i,j} \vee \quad v^{\mathrm{aft}(k)}_{j,x} \vee \neg v^{\mathrm{bef}(k)}_{i,x}\right) \\ \wedge \left(\neg g^k_{i,j} \vee \quad v^{\mathrm{aft}(k)}_{j,x} \vee \neg v^{\mathrm{bef}(k)}_{j,x}\right) \end{pmatrix} \tag{4.9}$$

We can avoid quantifying over j in the second clause by replacing the premiss literal $g^k_{i,j}$ with the premiss $\min^k_{i,u}$. Likewise we can avoid quantifying over i in the last clause. This splits Constraint (4.9) into the Constraints (4.10) and (4.11).

$$\bigwedge_{\substack{g^k_{i,j}\in\mathcal{G}\\ l\le i<j\le u}} \begin{pmatrix} \left(\neg g^k_{i,j} \vee \quad v^{\mathrm{aft}(k)}_{i,x} \vee \neg v^{\mathrm{bef}(k)}_{i,x} \vee \neg v^{\mathrm{bef}(k)}_{j,x}\right) \\ \wedge \left(\neg g^k_{i,j} \vee \neg v^{\mathrm{aft}(k)}_{i,x} \vee \quad v^{\mathrm{bef}(k)}_{j,x}\right) \\ \wedge \left(\neg g^k_{i,j} \vee \neg v^{\mathrm{aft}(k)}_{j,x} \vee \quad v^{\mathrm{bef}(k)}_{i,x} \vee v^{\mathrm{bef}(k)}_{j,x}\right) \\ \wedge \left(\neg g^k_{i,j} \vee \quad v^{\mathrm{aft}(k)}_{j,x} \vee \neg v^{\mathrm{bef}(k)}_{i,x}\right) \end{pmatrix} \tag{4.10}$$

$$\begin{aligned} &\bigwedge_{\min^k_{i,u}\in\mathcal{N}_{l,u}} \left(\neg \min^k_{i,u} \vee \neg v^{\mathrm{aft}(k)}_{i,x} \vee v^{\mathrm{bef}(k)}_{i,x}\right) \\ &\bigwedge_{\max^k_{l,j}\in\mathcal{X}_{l,u}} \left(\neg \max^k_{l,j} \vee v^{\mathrm{aft}(k)}_{j,x} \vee \neg v^{\mathrm{bef}(k)}_{j,x}\right) \end{aligned} \tag{4.11}$$

Next we consider the conjunctive normal form of Constraint (4.6):

$$\bigwedge_{\substack{nc^k_i\in\mathcal{NC}\\ l\le i\le u}} \begin{pmatrix} \left(\neg nc^k_i \vee \neg v^{\mathrm{aft}(k)}_{i,x} \vee \quad v^{\mathrm{bef}(k)}_{i,x}\right) \\ \wedge \left(\neg nc^k_i \vee \quad v^{\mathrm{aft}(k)}_{i,x} \vee \neg v^{\mathrm{bef}(k)}_{i,x}\right) \end{pmatrix} \tag{4.12}$$

We simplify further by exploiting the similarity of the Constraints (4.11) and (4.12). For the first clause the premiss is either $\min^k_{i,u}$ or nc^k_i. In other words the premiss is that i is not used as a max channel. This is conveniently modeled by the literal $\neg \max^k_{l,i}$. The simplification for the second clause works analogously. Thus we can combine the Constraints (4.11) and (4.12) to the following constraint:

$$\begin{aligned} &\bigwedge_{\max^k_{l,i}\in\mathcal{X}_{l,u}} \left(\max^k_{l,i} \vee \neg v^{\mathrm{aft}(k)}_{i,x} \vee v^{\mathrm{bef}(k)}_{i,x}\right) \\ &\bigwedge_{\min^k_{j,u}\in\mathcal{N}_{l,u}} \left(\min^k_{j,u} \vee v^{\mathrm{aft}(k)}_{j,x} \vee \neg v^{\mathrm{bef}(k)}_{j,x}\right) \end{aligned} \tag{4.13}$$

Constraint	3-literal clauses	4-literal clauses	total clauses
(4.5)	$4\binom{n}{2}d$	$2\binom{n}{2}d$	$6\binom{n}{2}d$
(4.6)	$2nd$		$2nd$
old encoding (4.5 + 4.6)	$2n^2d$	$2\binom{n}{2}d$	$2\left(n^2+\binom{n}{2}\right)d$
(4.9)	$4\binom{w}{2}d$	$2\binom{w}{2}d$	$6\binom{w}{2}d$
(4.12)	$2wd$		$2wd$
subnetwork optimization (4.9 + 4.12)	$2w^2d$	$2\binom{w}{2}d$	$2\left(w^2+\binom{w}{2}\right)d$
(4.10)	$2\binom{w}{2}d$	$2\binom{w}{2}d$	$4\binom{w}{2}d$
(4.13)	$2wd$		$2wd$
new encoding (4.10 + 4.13)	$2\binom{w+1}{2}d$	$2\binom{w}{2}d$	$2w^2d$

Table 4.1: *Comparison of the number of clauses with 3 or 4 literals in the old encoding, the old encoding after subnetwork optimization and the new encoding*

We conclude that introducing additional min and max variables allows to simplifies the old encoding consisting of the Constraints (4.9) and (4.12) and yields the new encoding with the Constraints (4.10) and (4.13).

Let w be the window size of an input sequence. The encoding adds $(n-1)d$ element value variables or $(w-1)d$ after subnetwork optimization.

Table 4.1 shows the number of clauses added by the new encoding or the old encoding before or after subnetwork optimization. Before the subnetwork optimization the old encoding depends on the number of elements n and the depth d. After applying the subnetwork optimization the old encoding depends on the window size w and the depth d. We see that the new encoding saves $2\binom{w}{2}d$ many 3-literal clauses over the old encoding.

To be pedantic the variables $\min_{l,l}^k$ and $\max_{u,u}^k$ are necessarily mapped to false. As a consequence the Constraint (4.13) contains clauses with only two literals. Similarly the element values before the first and after the last layer are statically known such that the solver knows how to map variables of the form $v_{i,x}^0$ and $v_{i,x}^d$ for all i and x. We gloss over such details.

Further the new encoding leads to increased propagation. Ehlers gives the example that during solving the solver may arrive at the partial assignment $v_{i,x}^{\mathrm{bef}(k)} \mapsto$ false and $g_{i,l}^k \mapsto$ false for every $g_{i,l}$ in gates($\min_{i,u}^k$). Then the improved encoding infers $v_{i,x}^{\mathrm{aft}(k)} \mapsto$ false by unit propagation. The old encoding does not even propagate if $g_{i,l}^k \mapsto$ false for every $g_{i,l}$ in gates($\min_{i,n-1}^k$) instead of gates($\min_{i,u}^k$) [16].

4.2.4 Layer Ambiguity

In general it is ambiguous how to partition the gates into layers. Consider a gate (i, j) in layer k. If the channels i and j are both not used in either layer $k + 1$ or $k - 1$, we may move (i, j) to that layer while retaining the functional behavior of the network.

We want the solver to only consider a single representative partition into layers for each equivalence class of networks that are equivalent modulo reordering these parallel comparators. In my bachelor thesis I highlighted how this corresponds to breaking the symmetry that reorders parallel comparators from the Sections 3.1.2 and 3.1.3 [19]. I pointed out that the following constraint breaks this symmetry.

$$\bigwedge_{\substack{g_{i,j}^k \in \mathcal{G} \\ k<d-1}} \neg nc_i^k \vee \neg nc_j^k \vee \neg g_{i,j}^{k+1} \tag{4.14}$$

Note that this constraint is not satisfied if the channels i and j are not used in the layer k before the gate (i, j) in layer $k + 1$. Thus the partial assignment of $nc_i^k \mapsto$ true, $nc_j^k \mapsto$ true and $g_{i,j}^{k+1} \mapsto$ true leads to a conflict such that the solver does not continue guessing. Yet adding this constraint yields an equisatisfiable formula, because we may move the gate to the earliest layer possible such that Constraint (4.14) is satisfied. Hence we break the symmetry such that the representative partition is the partition where every gate is moved to the earliest layer possible. Bundala et al. dub this constraint "Eager Comparator Placement" [42].

I gave an alternative constraint where we may move the gate to the latest layer possible by adding the constraint

$$\bigwedge_{\substack{g_{i,j}^k \in \mathcal{G} \\ k>0}} \neg nc_i^k \vee \neg nc_j^k \vee \neg g_{i,j}^{k-1} \tag{4.15}$$

Ehlers actually adds the Constraint (4.15) [43]. The code is commented with "If there is a comparator (i, j) in layer l and both nc_i^{l+1} and nc_j^{l+1} are false, add a comparator there". Notably Ehlers does not add the Constraint (4.14) but describes it in joint work with Bundala et al. [42].

Note that adding a comparator is not the same as moving a comparator. I believe Ehlers' justification for adding a comparator (i, j) in layer $l+1$ after an already existing comparator (i, j) in layer l is that the added comparator is clearly redundant with the already existing comparator such that the functional behavior of the network is retained.

In Section 5.2 I propose to combine (4.14) and (4.15).

4.2.5 Prefix

There are some important results about prefixes of comparison networks. A comparison network may be split between any two layers into a prefix network and a suffix network. In the following we assume that every prefix has exactly two layers unless stated otherwise.

Recall that we called the symmetry by permuting and untangling correct, because a comparison network N sorts iff if the network N' obtained by permuting and untangling N sorts. Similarly a prefix N can be extended to a sorting network of depth d iff the prefix N' obtained by permuting and untangling N can be extended to a sorting network of depth d.

The key insight is that it suffices to consider only one representative prefix from an equivalence class of prefixes that are obtained from each other by permuting and untangling.

We define $\text{outputs}(N) = \{N(x) \mid x \in \{0,1\}^n\}$ to denote the set of all output sequences a comparison network N might produce. We write $\pi(\text{outputs}(N))$ to permute the n output positions of every output sequence in $\text{outputs}(N)$ by π. If a (prefix) network N' is obtained from another (prefix) network N by permuting by π and untangling, then we have $\text{outputs}(N') = \rho(\text{outputs}(N))$ for some ρ.

We say N' *subsumes* N if $\text{outputs}(N') \subseteq \pi(\text{outputs}(N))$ [16]. If we can extend a prefix N to a sorting network of depth d, then the same extension makes any prefix N' that is subsumed by N a sorting network depth d. Hence if we consider a prefix (class), then we do not need to consider a prefix (class) if we already consider another prefix (class). Thus it suffices to consider fewer representative prefixes.

Deciding whether a network is the result of permuting and untangling another network is in P, whereas deciding whether a network subsumes another network is GI-complete [44].

Lastly there is another comparison network symmetry called *reflection* that maps every comparator (i, j) to $(n - j - 1, n - i - 1)$. We define $\mathsf{reverse}(x_0, x_1, .., x_{n-1}) = x_{n-1}, .., x_1, x_0$ and $\mathsf{negate}(x_0, x_1, .., x_{n-1}) = \neg x_0, \neg x_1, .., \neg x_{n-1}$ where $\neg : \{0,1\} \to \{0,1\}$ maps 0 to 1 and 1 to 0. Note that $\mathsf{reverse}$ and negate commute under composition. For a network N' that is the reflection of a network N we have $\text{outputs}(N') = \mathsf{negate}(\mathsf{reverse}(\text{outputs}(N)))$ [42, 19]. Again a prefix can be extended to a sorting network of depth d iff its reflection can be extended to a sorting network of depth d. This merges some equivalence classes such that it suffices to consider even fewer representative prefixes.

The state of the art approach fixes the first two layers in the SAT formula to obtain a SAT formula for every representative two-layer prefix. Consequently is runs a SAT solver for every representative prefix. If every formula is unsatisfiable we prove a lower

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$\lvert G_n \rvert$	10^0	10^0	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	1	2	4	5	8	12	22	21	48	50	117	94	262	211	609	411	1367	894

Table 4.2: *G_n is the set of all possible two-layer prefixes. R_n is the number of two-layer representative prefixes that need to be considered [16]. For $|G_n|$ the order of magnitude is listed instead.*

bound, otherwise if a sorting network with a representative prefix is found we prove an upper bound.

Table 4.2 shows the number of representative two-layer prefixes that need to be considered for different n. Notably for larger n there are more representatives for odd n than for even $n+1$. Codish et al. present a technique to generate all representative prefixes for some n efficiently [45, 42].

4.2.6 Prefix Optimization

Since we can pick an arbitrary representative from each equivalence class, it makes sense to optimize our choice.

Recall from Section 4.2.3 that the number of variables and clauses that encode the run of an input sequence x depend on the window size of x.

Ehlers defines the utility of a prefix as the total window size [27, 11]. We define window : $\{0,1\}^n \to \{0,1\}^*$ to strip the leading zeros and trailing ones from a sequence of size n. Then the total window size of a prefix N is

$$\sum_{x \in \text{outputs}(N)} |\text{window}(x)|.$$

With this in mind an optimal prefix is a prefix with a minimal total window size. Note that reflecting a window does not change its size, i.e.

$$|\text{window}(x)| = |\text{window}(\mathsf{negate}(\mathsf{reverse}(x)))|.$$

Next we observe that if prefix N and prefix N' are in the same equivalence class, then we have $|\text{outputs}(N)| = |\text{outputs}(N')|$. Further recall from Section 4.2.1 that we only encode the network runs of a *subset* of all input sequences. For his proof of the depth lower bound for $n = 17$ Ehlers encoded 2000 distinct inputs for every prefix [11]. Thus he summed only the 2000 smallest window sizes of the distinct outputs of a prefix. Optimizing a prefix is difficult, since a naive approach would consider 2^n initial input sequences and $n!$ ways to permute and untangle a prefix. For this purpose Ehlers developed greedy algorithms that permuted and untangled a prefix by random permutations to obtain a more optimized prefix [11].

As an example for $n = 5$ let $N = \{(1,2),(0,4)\},\{(0,1),(3,4)\}$. Permuting the channels of N by $\left(\begin{smallmatrix}0&1&2&3&4\\1&2&4&0&3\end{smallmatrix}\right)$ yields $N' = \{(2,4),(1,3)\},\{(1,2),(0,3)\}$, which is already standard such that we skip the untangling step. Running the $2^5 = 32$ distinct inputs leaves us fewer distinct outputs, that is

$$\begin{aligned}
\text{outputs}(N) &= \mathsf{S} \cup \{00\underbrace{100}_{3}, 00\underbrace{10}_{2}1, 0\underbrace{100}_{3}1, 0\underbrace{10}_{2}11, 0\underbrace{1100}_{4}, 0\underbrace{110}_{3}1, \underbrace{1110}_{4}1\} \\
\text{outputs}(N') &= \mathsf{S} \cup \{00\underbrace{10}_{2}, 00\underbrace{10}_{2}1, 00\underbrace{110}_{3}, \underbrace{10010}_{5}, \underbrace{100}_{3}11, \underbrace{10110}_{5}, \underbrace{10}_{2}111\} \\
\mathsf{S} &= \{00000, 00001, 00011, 00111, 01111, 11111\}
\end{aligned}$$

We disregard the sorted sequences in S. Each prefix has exactly seven inputs with a nonzero window size. We see that N has a smaller total window size than N'.

$$\begin{aligned}
\sum_{x \in \text{outputs}(N)} |\text{window}(x)| &= 3+2+3+2+4+3+4 \\
&= 2+2+3+3+3+4+4 \\
&= 21
\end{aligned}$$

$$\begin{aligned}
\sum_{x \in \text{outputs}(N')} |\text{window}(x)| &= 2+2+3+5+3+5+2 \\
&= 2+2+2+3+3+5+5 \\
&= 22
\end{aligned}$$

However we may only encode four instead of all seven inputs. Since we prefer inputs with a smaller window size, the total encoded window size of the four smallest inputs is $2+2+3+3 = 10$ for N and $2+2+2+3 = 9$ for N'. Therefore we may prefer N' even though N yields a smaller total window size over all inputs.

For upper bounds we do not consider representatives of all equivalence classes. Thus we can fix more than only the first two layers. We may basically guess a large part of the sorting network. Again it makes sense to minimize the total window size. Ehlers successfully used an optimized five-layer prefix to find a sorting network on 24 elements [29].

A notable mention is the Green filter, which is defined for $n = 2^i$ and consists of i layers. Ehlers conjectures that there are no better prefixes in terms of number of outputs [11]. The Green filter is often used as a starting point for prefixes even if n is not a power of 2.

5 Minor Contributions to the State of the Art

This chapter discusses miscellaneous minor variations of the state of the art approach including minor contributions. In Section 5.1 we discuss another utility function for prefix optimization. We consider a notion of obviously redundant comparators in Section 5.2 as well as a related symmetry. We discuss how resolving the layer ambiguity from Section 4.2.4 interacts with breaking this symmetry. And in Section 5.3 we ensure the intended semantics of the gate, min and max variables using alternative more compact encodings of the at-most-one constraint from the literature.

5.1 More Precise Prefix Utility Function

I believe there is a more precise utility function than the total window size from Section 4.2.6 as defined by Ehlers.

We revise the example for $n = 5$ from Section 4.2.6 such that we encode six inputs with the smallest window size instead of only four or all seven. Now both N and N' lead to a total encoded window size of 17.

$$\begin{aligned} 2+2+3+3+3+4 &= 17 \quad \text{for } N \\ 2+2+2+3+3+5 &= 17 \quad \text{for } N' \end{aligned}$$

However this glosses over the fact that we encode one more smaller window of size 2 instead of 3 with prefix N', whereas we do not encode one big window of size 5 but only one of window size 4 with prefix N.

Recall that to encode the run of an input sequence x with window size w we add $w(d-1-p)$ element value variables and $2w^2(d-p)$ clauses where p is the number of layers of the prefix (see Table 4.1). The total number of variables and clauses added by a prefix M a

$$\begin{aligned} \#vars(M) &= \sum_{x \in \text{outputs}(M)} (|\text{window}(x)|(d+1-p)) \\ &= (d+1-p) \sum_{x \in \text{outputs}(M)} |\text{window}(x)| \end{aligned}$$

$$\begin{aligned}\#clauses(M) &= \sum_{x \in \text{outputs}(M)} \Big(2|\text{window}(x)|^2(d-p)\Big) \\ &= 2(d-p) \sum_{x \in \text{outputs}(M)} |\text{window}(x)|^2\end{aligned}$$

Due to the inequality laws we have for any two prefixes M and M' from the same equivalence class that

$$\begin{aligned}\#vars(M) \leq \#vars(M') \quad &\iff \sum_{x \in \text{outputs}(M)} |\text{window}(x)| \leq \sum_{x \in \text{outputs}(M')} |\text{window}(x)| \\ \#clauses(M) \leq \#clauses(M') \iff &\sum_{x \in \text{outputs}(M)} |\text{window}(x)|^2 \leq \sum_{x \in \text{outputs}(M')} |\text{window}(x)|^2\end{aligned}$$

Hence minimizing the total window size minimizes the number of variables that are added.

The different distribution of window sizes is irrelevant for the number of variables, which is linear in w. However the number of clauses is quadratic in w such that the distribution of window sizes makes a difference. In particular for any two prefixes M and M' from the same equivalence class we have that I propose to evaluate the utility of a prefix not only by the total number of variables added but also by the total number of clauses added. That is if the total window size of two prefix candidates is the same we evaluate each prefix M by its squared window size

$$\sum_{x \in \text{outputs}(M)} |\text{window}(x)|^2.$$

This way we evaluate N and N' differently:

$$\begin{aligned}2^2 + 2^2 + 3^2 + 3^2 + 3^2 + 4^2 = 56 \quad &\text{for } N \\ 2^2 + 2^2 + 2^2 + 3^2 + 3^2 + 5^2 = 51 \quad &\text{for } N'\end{aligned}$$

To summarize if we encode the best six inputs we prefer N' over N because N' allows us to get away with adding $2 \cdot (56 - 51) \cdot (d - 2)$ fewer clauses. Let us assume that we want to prove the lower bound 5, i.e. that there exists no sorting network on 5 elements with depth $d = 4$. Then choosing N' over N saves 20 clauses.

5.2 Breaking the Symmetry of Obviously Redundant Comparators

Recall from Section 4.2.4 that Ehlers added the Constraint (4.15), which forced the SAT solver to either move a gate to the latest layer possible or alternatively add redundant gates after the gate such that the two channels of a gate are never both unused in the succeeding layer.

We may additionally add the Constraint (4.14), which forces the SAT solver to either move a gate to the earliest layer possible or alternatively add redundant gates before the gate such that the two channels of a gate are never both unused in the preceding layer. Combined the Constraints (4.14) and (4.15) fill the network with redundant comparators. We exploit this in a brief discussion at the beginning of Section 7.3.

Note that not every redundant comparator is this *obviously* redundant by succeeding or preceding another comparator on the same channels. We consider the symmetry of adding and removing obviously redundant comparators. Adding the Constraints (4.14) and (4.15) does not only completely break the symmetry that reorders parallel parallel comparators [19]. It additionally partially breaks the symmetry of adding and removing these obviously redundant comparators.

To understand why the symmetry break is only partial we consider three channels $i < j < m$. Let there be a gate (i, j) in layer $k-1$ and a gate (j, m) in layer $k+1$. Further let i, j and m be not used by any gate in layer k. Clearly this violates Constraint (4.14) for the gate (j, m) and the Constraint (4.15) for gate (i, j). We may fix this by adding the gate (i, j) in layer k, which is obviously redundant with (i, j) in layer $k-1$. Alternatively we may fix this by adding the gate (j, m) in layer k, which is obviously redundant with (j, m) in layer $k+1$. Clearly both resulting fixed networks are in the same symmetry class of adding and removing obviously redundant comparators. Hence there are more than one representative network from the same symmetry class that satisfy the symmetry breaking Constraints (4.14) and (4.15).

We observe that every symmetry class under the symmetry of adding and removing obviously redundant comparators contains exactly one network without obviously redundant comparators. Thus removing every obviously redundant comparators from all networks in the same symmetry class yields this network. Hence we can break the symmetry of adding and removing obviously redundant comparators by forcing the SAT solver to remove obviously redundant comparators instead of adding them. To this end Bundala et al. add the constraint [42]

$$\bigwedge_{\substack{g^k_{i,j} \in \mathcal{G} \\ k>0}} \neg g^k_{i,j} \vee \neg g^{k-1}_{i,j} \tag{5.1}$$

Additionally we either use Constraint (4.14) to move every gate to the earliest layer possible or alternative Constraint (4.15) to move every gate to the latest layer possible.

This approach can be extended to disallow redundant comparators that are not obviously redundant. In my bachelor's project we gave auxiliary variables and constraints that encode which comparators are redundant [46]. Unfortunately our encoding was incomplete, i.e. not every redundant comparator was inferred to be redundant. Back then we did not add constraints to move a gate to the earliest/latest layer.

The takeway is that we can add only two of the three Constraints (4.14), (4.15) and (5.1) to the encoding. Adding all three yields an incorrect symmetry breaking constraint. In

particular we may combine the first two constraints to fill the network with obviously redundant comparators, but this prevents us to break the symmetry of adding and removing such obviously redundant comparators.

As far as I know this thesis is the first work that discusses the relation between obviously redundant comparators and reordering parallel comparators in depth. Specifically this thesis is the first work that shows a use case for filling the network with obviously redundant comparators. For details on that see the discussion at the beginning of Section 7.3. As such this is the first discussion of combining the Constraints (4.14) and (4.15) that I am aware of.

Again in joint work with Bundala et al. Ehlers gives an encoding with the Constraints (4.14) and (5.1) [42], but in his implementation Ehlers only adds the Constraint (4.15) [43].

5.3 At-Most-One Constraints

Actually the number of clauses added by the at-most-one Constraint (4.2) is quadratic in the number of literals it limits to one. There are various alternative encodings that counteract the quadratic blowup in clauses by adding auxiliary variables such that the number of clauses is reduced [47].

5.3.1 Commander Encoding

One encoding of the at-most-one constraint is the *commander* encoding, which introduces auxiliary commander variables that command disjoint subsets of the set of literals [48].

We use the commander encoding to revise the Constraint (4.4) that ensures that every channel i is used *at most once* in layer k.

$$\bigwedge_{k\in L}\bigwedge_{i\in C} \operatorname{eo}\begin{pmatrix} \{g^k_{i,j} \mid j \in C, i < j\} \\ \cup\, \{g^k_{h,i} \mid h \in C, h < i\} \\ \cup\, \{nc^k_i\} \end{pmatrix} \tag{4.4}$$

A commander encoding ensures the following conditions.

1. At most one literal in a set can be true.
2. If the commander variable of a set is true, then at least one of the literals in the set must be true.
3. If the commander variable of a set is false, then none of the literals in the set can be true.

4. Exactly one of the commander variables is true.

Note that the gate sets in the definition of Constraint (4.4) are captured by the semantics that we ascribed to the min and max variables introduced in Section 4.2.3.

$$\{g^k_{i,j} \mid j \in C, i < j\} = \text{gates}\left(\min^k_{i,n-1}\right)$$
$$\{g^k_{h,i} \mid h \in C, h < i\} = \text{gates}\left(\max^k_{0,i}\right)$$

Hence we can avoid the costs of introducing new auxiliary variables by reusing the already introduced $\min^k_{i,n-1}$ and $\max^k_{0,i}$ variables as commander variables.

By Constraint (4.7) a satisfying assignment maps the variable $\min^k_{i,n-1}$ to true iff a gate variable in gates $\left(\min^k_{i,n-1}\right)$ maps to true. Constraint (4.8) enforces the same for $\max^k_{0,i}$. This satisfies Conditions 2 and 3 of the commander encoding.

To satisfy Condition 1 we add the constraints:

$$\bigwedge_{\min^k_{i,n-1} \in \mathcal{N}_{0,n-1}} \text{amo}\Big(\text{gates}(\min^k_{i,n-1})\Big) \tag{5.2}$$

$$\bigwedge_{\max^k_{0,j} \in \mathcal{X}_{0,n-1}} \text{amo}\Big(\text{gates}(\max^k_{0,j})\Big) \tag{5.3}$$

Further we satisfy Condition 4 by adding the following constraint:

$$\bigwedge_{nc^k_i \in \mathcal{NC}} \text{eo}\left(\{\min^k_{i,n-1}, \max^k_{0,i}, nc^k_i\}\right) \tag{5.4}$$

Note that the Constraints (5.2) and (5.3) add strictly fewer clauses than Constraint (4.4). Specifically they do not add clauses of the form $\neg g^k_{h,i} \vee \neg g^k_{i,j}$. Therefore we save $\binom{n}{3}d$ clauses. The remaining clauses of the amo constraint from Constraint 4.4 are either of the form $\neg g^k_{i,h} \vee \neg g^k_{i,j}$ with $h \neq j$ or of the form $\neg g^k_{h,i} \vee \neg g^k_{j,i}$ with $h \neq j$. Both forms appear $\binom{n}{3}d$ times, since the literal order within a clause is irrelevant such that we can assume $h < j$. Constraint (5.2) adds these $\binom{n}{3}d$ clauses of the former form and Constraint (5.3) adds these $\binom{n}{3}d$ clauses of the latter form.

Both the pairwise Encoding (4.2) of the at-most-one constraint as well as the Commander encoding enforce arc consistency [47]. That means that if a one literal evaluates to true in a partial assignment, then under unit propagation the variables of the other literals are assigned such that the other literals evaluate to false.

Thus we can substitute the Constraints (5.2), (5.3) and (5.4) for Constraint (4.4) and achieve an encoding of at least the same quality but with less clauses. However the improved compactness is negligible as the size of the encoding is still dominated by the exponentially sized encoding of the network runs.

Nonetheless the substitution results in a speedup. However the same speedup is also observed when only adding the clauses of the form $\neg \min_{i,n-1}^{k} \vee \neg \max_{0,i}^{k}$ from Constraint (5.4) to Constraint (4.4) instead of replacing it. This indicates that the decrease in clauses does not cause the speedup. Rather it stems from the new clauses enabling propagation.

5.3.2 Sequential Encoding

We can further reduce the number of clauses by repeating the trick again. The *sequential* encoding (aka relaxed ladder encoding) is another encoding that saves clauses over the naive at-most-one Constraint (4.2) by introducing auxiliary variables [47]. Once again these auxiliary variables turn out to be the min and max variables. Since we add these anyway, the cost of introducing additional variables is already paid for. Specifically we use the sequential encoding for the at-most-one constraints in the Constraints (5.2) and (5.3).

The sequential encoding assumes an order on the literals that are limited to at most one one satisfying one. For the constraint $\text{amo}\Big(\text{gates}(\min_{i,n-1}^{k})\Big)$ we order the gate variables in $\text{gates}(\min_{i,n-1}^{k})$ by

$$g_{i,i+1}^{k} < g_{i,i+2}^{k} < .. < g_{i,n-1}^{k}.$$

For these variables $x_1 < x_2 < .. < x_m$ the sequential encoding introduces auxiliary variables a_i for $1 \leq i < m$. Finally the sequential encoding consists of the following clauses:

$$(\neg x_1 \vee a_1) \wedge (\neg x_m \vee \neg a_{m-1}) \wedge \bigwedge_{1<i<m} \Big((\neg x_i \vee a_i) \wedge (\neg a_{i-1} \vee a_i) \wedge (\neg x_i \vee a_{i-1})\Big).$$

Marques-Silva and Lynce observe that a satisfying assignment for this constraint maps the variable sequence $a_1, .., a_{m-1}$ to $0..01..1.$ and whenever any variable x_i with $1 < i \leq m-1$ is assigned to true (or 1), consequently under unit propagation all other variables x_j with $1 \leq j \leq m$ and $i \neq j$ must be forced to false (or 0) [49].

We observe that the auxiliary variables $a_1 < a_2 < .. < a_{m-1}$ correspond to

$$\min_{i,i+1}^{k} < \min_{i,i+2}^{k} < .. < \min_{i,n-2}^{k}.$$

The same way as a_j implies a_l for $j < l$ we have that $\min_{i,j}^{k}$ implies $\min_{i,l}^{k}$ for $j < l$. Analogously all this holds for max too. We substitute the Constraints (5.2) and (5.3)

with the following constraints:

$$\bigwedge_{\substack{i\in C\\k\in L}} \left((\neg g^k_{i,i+1} \vee \min^k_{i,i+1}) \wedge (\neg g^k_{i,n-1} \vee \neg \min^k_{i,n-2})\right) \\ \bigwedge_{\substack{i,j\in C\\k\in L\\i+1<j<n-1}} \left((\neg g^k_{i,j} \vee \min^k_{i,j}) \wedge (\neg \min^k_{i,j-1} \vee \min^k_{i,j}) \wedge (\neg g^k_{i,j} \vee \neg \min^k_{i,j-1})\right) \tag{5.5}$$

$$\bigwedge_{\substack{i\in C\\k\in L}} \left((\neg g^k_{i-1,i} \vee \max^k_{i-1,i}) \wedge (\neg g^k_{0,i} \vee \neg \max^k_{1,i})\right) \\ \bigwedge_{\substack{i,j\in C\\k\in L\\0<j<i-1}} \left((\neg g^k_{j,i} \vee \max^k_{j,i}) \wedge (\neg \max^k_{j+1,i} \vee \max^k_{j,i}) \wedge (\neg g^k_{j,i} \vee \neg \max^k_{j+1,i})\right) \tag{5.6}$$

5.3.3 Ladder Encoding

We can similarly sequentialize the Constraint (4.7), which ensures that the min variables are consistent with their intended semantics. Remember that $g^k_{i,j} \mapsto$ true already forces the truth value of all $\min^k_{i,l}$ variables with $l \neq n-1$ under unit propagation due to the Constraints (5.5) and (5.6). Thus the $\leftarrow$ direction of Constraint (4.7) is almost completely realized by the Constraint (5.5): We add the following constraint to handle the remaining case $l = n-1$.

$$\bigwedge_{\substack{i\in C\\k\in L\\i<n-1}} \left(\neg g^k_{i,n-1} \vee \min^k_{i,n-1}\right) \tag{5.7}$$

To realize the $\rightarrow$ direction of Constraint (4.7), we can either use one large clause

$$\bigwedge_{\min^k_{i,j}\in\mathcal{N}} \left(\neg \min^k_{i,j} \vee \bigvee_{g^k_{i,l}\in \text{gates}(\min^k_{i,j})} g^k_{i,l}\right). \tag{5.8}$$

Alternatively we can sequentialize this large clause as well:

$$\bigwedge_{g^k_{i,j}\in\mathcal{G}} \left(\neg \min^k_{i,j} \vee g^k_{i,j} \vee \min^k_{i,j-1}\right) \tag{5.9}$$

Notably the clauses in Constraint (5.9) are exactly the clauses that are removed from the ladder encoding to obtain the relaxed ladder encoding from Section 5.3.2 [47]. The

ladder encoding is also known as the regular encoding. The ladder encoding yields the following constraints.

$$\bigwedge_{g^k_{i,j} \in \mathcal{G}} \begin{pmatrix} (\neg \min^k_{i,j-1} \vee \min^k_{i,j}) \wedge (\neg \min^k_{i,j} \vee g^k_{i,j} \vee \min^k_{i,j-1}) \\ \wedge (\neg g^k_{i,j} \vee \min^k_{i,j}) \qquad \wedge (\neg g^k_{i,j} \vee \neg \min^k_{i,j-1}) \end{pmatrix} \tag{5.10}$$

$$\bigwedge_{g^k_{j,i} \in \mathcal{G}} \begin{pmatrix} (\neg \max^k_{j+1,i} \vee \max^k_{j,i}) \wedge (\neg \max^k_{j,i} \vee g^k_{j,i} \vee \max^k_{j+1,i}) \\ \wedge (\neg g^k_{j,i} \vee \max^k_{j,i}) \qquad \wedge (\neg g^k_{j,i} \vee \neg \max^k_{j+1,i}) \end{pmatrix} \tag{5.11}$$

Note that all these clauses are satisfied for $|i-j| = 1$, because $\min^k_{i,i+1} = g^k_{i,i+1} = \max^k_{i,i+1}$ and the fact that both $\min^k_{i,i}$ and $\max^k_{i,i}$ map to false in a satisfying assignment. Therefore each of the two Constraints (5.10) and (5.11) adds $4\binom{n-1}{2}d$ clauses.

In summary the Constraints (5.4), (5.10) and (5.11) replace the Constraints (4.4), (4.7) and (4.8)

6 Encoding Another Permuted and Untangled Network

Note that the subnetwork optimization is actually the repeated application of van Voorhis' lemma in disguise. Van Voorhis points out that given a sorting network we may remove a path to either the first or last network output node together with all comparison nodes on that path to obtain a sorting network on one fewer element than the original one [20, 17]. Removing e.g. the first, second and last channel of a standard network applies van Voorhis' lemma three times. Hence I consider subnetwork optimization a weaker version of van Voorhis' lemma. It is weaker because given either the first or last network output node t_i, we can only remove a single path to t_i instead of all candidate paths, since any path from one of the n network input nodes is suitable for van Voorhis' lemma.

Recall that there are $n!$ permutations π of the channel numbers that correspond to the $n!$ standard π-channel labelings of a sorting network. The important observation is that permuting and untangling enables subnetwork optimization to remove the other candidates such that permuting and untangling recovers some power of van Voorhis' lemma.

Permuting and untangling recovers only some power of van Voorhis' lemma, because some paths through the network to t_i cannot be channels in a *standard* channel labeling. Although replacing permuting and untangling with input twists would solve this, it is a dead end, because the subnetwork optimization does not apply to generalized networks. The crucial difference between van Voorhis' lemma and subnetwork optimization is that subnetwork optimization only removes paths to the minimum (maximum) output where a 0 (1) at the start of the path remains on the path in every run.

In this section we exploit the fact that permuting and untangling enables us to apply the subnetwork optimization for different paths to the outer network output nodes.

6.0.1 Another Permuted and Untangled Sorting Network

The idea is the following. Let N be a sorting network and consider the sorting network R obtained from N by permuting the channels by π and untangling. For an input sequence x holds that N sorts x iff R sorts $\pi(x)$. Hence we either

1. encode the network run of the input sequence x in N

2. encode the network run of the input sequence $\pi(x)$ in R.

Note that we have required N to be a sorting network. However we encode the network run of x (or alternatively $\pi(x)$) in order to prove that the encoded network N sorts, i.e. we want to have the choice between Options 1 and 2 for any comparison network N, not only for any sorting network N.

Recall from Section 3.1.3 that input twists are a strongly correct symmetry, which is finer than and strongly complete relative to permuting channels and twisting outputs. We encode two standard networks N and R and add the constraint that R is obtained from N by input twists that realize the permutation π on R's inputs. I find it intuitive to interpret R as the π-line representation of the id-graph representation of N, although that graph representation is not part of the encoding.

If there is a sorting network, then there is a satisfying assignment for this constraint, because input twists are strongly complete relative to sorting networks. Hence we infer a lower bound if the constraint is unsatisfiable.

But if there is a satisfying assignment for the constraint, then N sorts x iff R sorts $\pi(x)$ for any input sequence x such that we may choose between Options 1 and 2.

6.0.2 Savings

Note that the window sizes for Options 1 and 2 may differ. We prefer the former Option 1 if $|\text{window}(x)| \leq |\text{window}(\pi(x))|$ and the latter Option 2 otherwise. We save variables and clauses over the state of the art approach by reducing the encoded window size of an input x by $|\text{window}(x)| - |\text{window}(\pi(x))|$ if the latter case of $|\text{window}(x)| > |\text{window}(\pi(x))|$ applies.

In some cases we even have $|\text{window}(x)| > 0 = |\text{window}(\pi(x))|$ such that it becomes unnecessary to encode the network run of an input x that had to be encoded in the state of the art approach. In fact we could encode all $n!$ standard π-channel representations for any π of the id-graph representation of N instead of only a single π. Then it would be unnecessary to encode network runs, because for any x there is always some π such that $|\text{window}(x)| > 0 = |\text{window}(\pi(x))|$. This matches our knowledge that a network sorts iff it admits a standard π-channel labeling for any π.

Due to the zero one-principle it even suffices to encode only 2^n instead of all $n!$ standard networks to ensure that a network sorts. However we will see that encoding an input run requires far fewer variables and clauses than encoding another network obtained from input twists. Hence the encoding of 2^n network runs is smaller than the encoding of 2^n standard networks related by input twists.

Rather we exploit that adding more and more standard network related by input twists to the encoding yields less and less savings in the encoding of the network runs over the state of the art approach. Thus we only encode a single standard network related by

input twists, which has the best cost benefit ratio in terms of added and saved variables and clauses.

It makes sense to pick π such that we maximize our savings over the state of the art approach. An input sequence that starts with a 1 and ends with a 0 has a window size of n. There are $\frac{1}{4}2^n$ such inputs in $\{0,1\}^n$ for $n \geq 2$. Recall that reflection preserves the window size, that is $|\text{window}(x)| = |\text{window}(\mathsf{negate}(\mathsf{reverse}(x)))|$. As a consequence we have $|\text{window}(\mathsf{negate}(x))| = |\text{window}(\mathsf{reverse}(x))|$. We save at least $2\frac{1}{4}2^n$, i.e. exponentially many, variables and clauses by choosing $\pi = \left(\begin{smallmatrix} 0 & 1 & \cdots & n-2 & n-1 \\ n-1 & n-2 & \cdots & 1 & 0 \end{smallmatrix}\right)$ assuming we do not fix the first two layers.

6.0.3 Prefix Pair Optimization

However in practice we do fix the first two layers.

Note that if a (prefix) network N' is obtained from another (prefix) network N by permuting by π and untangling, then we have $\text{outputs}(N') = \rho(\text{outputs}(N))$ for some ρ. To understand this we consider the permutation that maps between the sequences at stage k in the run of input sequence x in N and input sequence $\pi(x)$ in N'. Initially the permutation is π. If N is a sorting network, then we have $\rho = \mathsf{id}$ such that N' is a sorting network too.

Further we consider the interpretation that N' is the π-line representation of the id-graph representation of N. In this interpretation the standard π-channel labeling, that determines N', corresponds to a network graph run that sorts an input shuffled by π such that the output is shuffled by ρ. Interpreted this way the permutation between the sequence at stage k in each network is initially shuffled by π and in a sorting network ultimately sorted to be id. If N and N' are two-layer prefixes π is only partially sorted to be some ρ. Hence even if we permute the inputs to the N' by $\left(\begin{smallmatrix} 0 & 1 & \cdots & n-2 & n-1 \\ n-1 & n-2 & \cdots & 1 & 0 \end{smallmatrix}\right)$, the first two layers change that permutation, since any gate swaps the reverse permutation.

The problem of prefix optimization now becomes to find the best *pair* of prefix representatives N and N' where N' is obtained by permuting the channels of N by π and untangling such that we minimize

$$\sum_{x \in \text{outputs}(N)} \min\big(|\text{window}(x)|, |\text{window}(\pi(x))|\big).$$

Keep in mind that we also use this new utility function to determine the best, e.g. 2000 for $n = 17$, inputs.

We refrain from solving this problem. Instead we reuse the prefix representatives from Ehlers' previous work [11, 43]. We pick $\pi = \left(\begin{smallmatrix} 0 & 1 & \cdots & n-2 & n-1 \\ n-1 & n-2 & \cdots & 1 & 0 \end{smallmatrix}\right)$, since it appears to yield the most savings.

6.1 Encoding Another Standard Network with Inputs in Reverse Order

In his dissertation Ehlers writes wrt to future work:

> A drawback of the SAT encodings used so far is that they do not consider symmetries of prefixes, which we would consider a major achievement. Here, the term of symmetries relates to the relation between sets of outputs of different prefixes. It is not clear if these can be encoded into SAT efficiently. [11]

In this section we develop the encoding that we motivated in Section 6.0.1. We use the symbols $\mathbb{I}$ and $\mathbb{R}$ where $\mathbb{I}$ denotes to the standard network from the prior encoding and $\mathbb{R}$ denotes the standard network related to $\mathbb{I}$ by input twists whose inputs are in reverse order. We group them in the set $R = \{\mathbb{I}, \mathbb{R}\}$.

6.1.1 Encoding $\mathbb{R}$

We double the number of g, nc, min and max variables to model both $\mathbb{I}$ and $\mathbb{R}$. For this we revise the definitions of $\mathcal{G}$, $\mathcal{NC}$, $\mathcal{N}$, $\mathcal{X}$, $\mathcal{N}_{l,u}$ and $\mathcal{X}_{l,u}$ to additionally quantify over R.

- The variable $g_{i,j}^{k,r}$ models a comparator between channel i and channel j in layer k in network r.
$$\mathcal{G} = \{g_{i,j}^{k,r} \mid r \in R, k \in L, i, j \in C, i < j\}$$

- The variable $nc_i^{k,r}$ models that channel i in layer k in network r is not used by any comparator.
$$\mathcal{NC} = \{nc_i^{k,r} \mid r \in R, k \in L, i \in C\}$$

- The variables $\min_{i,j}^k$ and $\max_{i,j}^k$ model that channel i or j is used as a min or max channel of a gate within the subnetwork of network r delimited by i and j.
$$\begin{aligned}
\mathcal{N} &= \{\min_{i,j}^{k,r} \mid r \in R, k \in L, i, j \in C, i \leq j\} \\
\mathcal{X} &= \{\max_{i,j}^{k,r} \mid r \in R, k \in L, i, j \in C, i \leq j\} \\
\mathcal{N}_{l,u} &= \{\min_{i,u}^{k,r} \mid r \in R, k \in L, i \in C, l \leq i \leq u\} \\
\mathcal{X}_{l,u} &= \{\max_{l,j}^{k,r} \mid r \in R, k \in L, j \in C, l \leq j \leq u\}
\end{aligned}$$

- The variable $v_{i,x}^k$ models the element value on channel i in the kth segment between two stages in the network run of the input sequence x in the network r.
$$\mathcal{V} = \{v_{i,x}^{k,r} \mid r \in R, x \in \{0,1\}^n, k \in B, i \in C\}$$

Note that only a subset of $\mathcal{V}$ is actually part of the encoding due to the subnetwork optimization and the choice to encode a network run in one or the other network.

This duplicates the Constraints (5.4), (5.10) and (5.11) that now similarly quantify over R such that they apply to both networks.

6.1.2 Encoding Network Runs

Let N and N' be a prefix pair where N' is obtained from N' by permuting channels and untangling. The prefixes N and N' determine one permutation π such that $\pi(\text{outputs}(N)) = \text{outputs}(N')$ where we may either encode the network run of the input sequence x in the extension of N or alternatively the network run of $\pi(x) \in \pi(\text{outputs}(N))$ in the extension of N'.

Given such a prefix pair N and N' and markers for their extension, e.g. $\mathbb{R}$ and $\mathbb{I}$, we define $\text{outputs}(N, N')$ as

$$\text{outputs}(N, N') = \{o(x) \mid x \in \text{outputs}(N)\}$$
$$o(x) = \begin{cases} (x, \mathbb{I}) & \text{if } |\text{window}(x)| \leq |\text{window}(\pi(x))| \\ (\pi(x), \mathbb{R}) & \text{otherwise} \end{cases}$$

We write $\text{outputs}_m(N, N')$ to select the m smallest sequences from $\text{outputs}(N, N')$ ordered by their window size. Ties are broken arbitrarily.

Let $\text{bounds}(x) : \{0,1\}^n \to (C, C)$ denote the element numbers that delimit the window of x. If we encode the best 2000 prefixes we only introduce the following subset of the element value variables $\mathcal{V}$:

$$\{v_{i,x}^{k,r} \mid (x, r) \in \text{outputs}_{2000}(N, N'), k \in B, (l, u) = \text{bounds}(x), i \in \{l, .., u\}\}$$

According to the case distinction in $o(..)$ we encode the network run through the subnetwork suffix of depth $d-2$ delimited by l and u of either network $\mathbb{I}$ or network $\mathbb{R}$. As previously this is done with the Constraints (4.10) and (4.13) where the network is now fixed to either $\mathbb{I}$ or $\mathbb{R}$.

6.1.3 Encoding the Relation between $\mathbb{R}$ and $\mathbb{I}$

In Section 6.0.3 we explored the perspective that permuting and untangling or equivalently input twists determine a permutation of the elements between each stage, which corresponds to a permutation of the channel numbers. We encode these permutations π_k for each segment k in B.

The variable $p^k_{h,i}$ models that $\pi_k(i) = h$, i.e. channel i is mapped to channel h by the channel permutation that is realized on the channels in segment k by the input twists relating $\mathbb{I}$ and $\mathbb{R}$.

$$\mathcal{P} = \{p^k_{h,i} \mid k \in B, h, i \in C\}$$

Note that h is a channel in network $\mathbb{R}$ and that i is a channel in network $\mathbb{I}$.

Since π_k is a permutation, we ensure that the mapping is surjective (Constraint (6.1a)) and injective (Constraint (6.1b)).

$$\bigwedge_{\substack{k\in B\\ h\in C}} \mathrm{eo}\left(\{p^k_{h,i} \mid i \in C\}\right) \tag{6.1a}$$

$$\bigwedge_{\substack{k\in B\\ i\in C}} \mathrm{eo}\left(\{p^k_{h,i} \mid h \in C\}\right) \tag{6.1b}$$

For the segment before the first layer we know that the channels in $\mathbb{R}$ correspond to the channels in the reverse position in the original network $\mathbb{I}$. In the segment after the last layer the outputs of both $\mathbb{I}$ and $\mathbb{R}$ are sorted. Hence we have $\pi_0 = \left(\begin{smallmatrix} 0 & 1 & .. & n-2 & n-1 \\ n-1 & n-2 & .. & 1 & 0 \end{smallmatrix}\right)$ and $\pi_d = \mathsf{id}$. We fix these permutations in our encoding:

$$\bigwedge_{i\in C} p^0_{n-i-1,i} \tag{6.1c}$$

$$\bigwedge_{i\in C} p^d_{i,i} \tag{6.1d}$$

The mapping given by the p variables relates the original network $\mathbb{I}$ with the reversed and untangled network $\mathbb{R}$ such that each gate or other variable in one network is mapped to a counterpart in the other.

$$\bigwedge_{\substack{k\in L\\ h,m,i,j\in C\\ h<m\\ i<j}} \left(\left(p^{\mathrm{aft}(k)}_{h,i} \wedge p^{\mathrm{aft}(k)}_{m,j}\right) \to \left(g^{k,\mathbb{R}}_{h,m} \leftrightarrow g^{k,\mathbb{I}}_{i,j}\right)\right) \tag{6.1e}$$

$$\bigwedge_{\substack{k\in L\\ h,i\in C}} \left(p^{\mathrm{aft}(k)}_{h,i} \to \left(nc^{k,\mathbb{R}}_{h} \leftrightarrow nc^{k,\mathbb{I}}_{i}\right)\right) \tag{6.1f}$$

$$\bigwedge_{\substack{k\in L\\ h,i\in C}} \left(p^{\mathrm{aft}(k)}_{h,i} \to \left(\min^{k,\mathbb{R}}_{h,n-1} \leftrightarrow \min^{k,\mathbb{I}}_{i,n-1}\right)\right) \tag{6.1g}$$

$$\bigwedge_{\substack{k\in L\\ h,i\in C}} \left(p^{\mathrm{aft}(k)}_{h,i} \to \left(\max^{k,\mathbb{R}}_{0,h} \leftrightarrow \max^{k,\mathbb{I}}_{0,i}\right)\right) \tag{6.1h}$$

Note that for two corresponding gates $g^{k,\mathbb{R}}_{h,m}$ and $g^{k,\mathbb{I}}_{i,j}$ the permutation of their outputs $\pi_{\mathrm{aft}(k)}$ always maps $\min(h,m)$ to $\min(i,j)$ and $\max(h,m)$ to $\max(i,j)$. This way the

gates in layer k "sort" the permutation $\pi_{\mathrm{aft}(k)}$. Recall that in this interpretation the permutation id counts as sorted.

Unfortunately Constraint (6.1e) is linear in d and *quartic* in n. That means that this constraint is particularly costly. At first glance it seems like the Constraints (6.1g) and (6.1h) similarly ensure that $\pi_{\mathrm{aft}(k)}$ always maps $\min(h,m)$ to $\min(i,j)$ and $\max(h,m)$ to $\max(i,j)$. However these two constraints alone do not suffice, because that would allow the channels i and j of a gate $g_{i,j}^{k,\mathbb{I}}$ to be mapped by $p_{h,i}^{\mathrm{aft}(k)}$ and $p_{m,j}^{\mathrm{aft}(k)}$ to two different gates $g_{h,o}^{k,\mathbb{R}}$ and $g_{p,m}^{k,\mathbb{R}}$ with $h \neq p$ and $o \neq m$. Instead we only add the Constraints (6.1f), (6.1g) and (6.1h) to increase propagation.

We disallow gates whose counterpart would be nonstandard.

$$\bigwedge_{\substack{k\in L\\ h,m,i,j\in C\\ h>m\\ i<j}} \left(\left(p_{h,i}^{\mathrm{aft}(k)} \wedge p_{m,j}^{\mathrm{aft}(k)}\right) \to \neg g_{i,j}^{k,\mathbb{I}}\right) \tag{6.1i}$$

$$\bigwedge_{\substack{k\in L\\ h,m,i,j\in C\\ h<m\\ i>j}} \left(\left(p_{h,i}^{\mathrm{aft}(k)} \wedge p_{m,j}^{\mathrm{aft}(k)}\right) \to \neg g_{h,m}^{k,\mathbb{R}}\right) \tag{6.1j}$$

$$\bigwedge_{\substack{k\in L\\ i\in C}} \left(\begin{array}{l} \left(p_{n-1,i}^{\mathrm{aft}(k)} \to \neg \min_{i,n-1}^{k,\mathbb{I}}\right)\\ \wedge \left(p_{0,i}^{\mathrm{aft}(k)} \to \neg \max_{0,i}^{k,\mathbb{I}}\right)\end{array}\right) \tag{6.1k}$$

$$\bigwedge_{\substack{k\in L\\ h\in C}} \left(\begin{array}{l} \left(p_{h,n-1}^{\mathrm{aft}(k)} \to \neg \min_{h,n-1}^{k,\mathbb{R}}\right)\\ \wedge \left(p_{h,0}^{\mathrm{aft}(k)} \to \neg \max_{0,h}^{k,\mathbb{R}}\right)\end{array}\right) \tag{6.1l}$$

Next we constrain the relation between $\pi_{\mathrm{bef}(k)}$ and $\pi_{\mathrm{aft}(k)}$ for all layers k. Specifically we have $\pi_k = \pi_{k+1}$ unless the input twists that relate $\mathbb{I}$ and $\mathbb{R}$ twist the inputs of a gate in k. In other words only input twists change the permutation. We encode the input twists in network $\mathbb{R}$.

The variable $t_{h,m}^k$ models that the inputs of a comparator $g_{h,m}^{k,\mathbb{R}}$ are twisted.

$$\mathcal{T} = \{t_{h,m}^k \mid k \in L, h, m \in C, h < m\}$$

We ensure that the inputs of a comparator are only twisted if the comparator actually exists:

$$\bigwedge_{t_{h,m}^k \in \mathcal{T}} \left(t_{h,m}^k \to g_{h,m}^{k,\mathbb{R}}\right) \tag{6.1m}$$

The variable nt_h^k models that if channel h is used by a comparator in layer k in network $\mathbb{R}$, then that comparator's inputs are not twisted. Note that both nt_h^k and $nc_h^{k,\mathbb{R}}$ can be true at the same time.

$$\mathcal{NT} = \{nt_h^k \mid k \in L, h \in C\}$$

Obviously we want twisting and not twisting to be mutually exclusive (we denote the logical XOR operator by $\oplus$):

$$\bigwedge_{t_{h,m}^k \in \mathcal{T}} \left(t_{h,m}^k \oplus \left(nt_h^k \wedge nt_m^k \right) \right) \tag{6.1n}$$

Now any assignment of the twisting and not twisting variables in layer k satisfying Constraint (6.1n) relates $\pi_{\text{bef}(k)}$ and $\pi_{\text{aft}(k)}$. We consider the channel i in layer k in network $\mathbb{I}$ with $\pi_{\text{bef}(k)}(i) = h$. If there is no twist of h in the layer k between the segments in network $\mathbb{R}$, then $\pi_{\text{aft}(k)}$ maps i to h (Constraint (6.1p)). On the other hand if h is twisted with m in layer k, then $\pi_{\text{aft}(k)}$ maps i to m (Constraint (6.1o)).

$$\bigwedge_{\substack{t_{h,m}^k \in \mathcal{T} \\ i \in C}} \begin{pmatrix} t_{h,m}^k \rightarrow \begin{pmatrix} \left(p_{m,i}^{\text{aft}(k)} \leftrightarrow p_{h,i}^{\text{bef}(k)}\right) \\ \wedge \left(p_{h,i}^{\text{aft}(k)} \leftrightarrow p_{m,i}^{\text{bef}(k)}\right) \end{pmatrix} \\ t_{h,m}^k \leftarrow \begin{pmatrix} \left(p_{m,i}^{\text{aft}(k)} \wedge p_{h,i}^{\text{bef}(k)}\right) \\ \vee \left(p_{h,i}^{\text{aft}(k)} \wedge p_{m,i}^{\text{bef}(k)}\right) \end{pmatrix} \end{pmatrix} \tag{6.1o}$$

$$\bigwedge_{\substack{nt_h^k \in \mathcal{NT} \\ i \in C}} \begin{pmatrix} nt_h^k \rightarrow \left(p_{h,i}^{\text{aft}(k)} \leftrightarrow p_{h,i}^{\text{bef}(k)}\right) \\ nt_h^k \leftarrow \left(p_{h,i}^{\text{aft}(k)} \wedge p_{h,i}^{\text{bef}(k)}\right) \end{pmatrix} \tag{6.1p}$$

6.1.4 Increasing Propagation

The Constraints (6.1e) - (6.1l) all relate a layer k in $\mathbb{R}$ and $\mathbb{I}$ with $\pi_{\text{aft}(k)}$. We may similarly relate every layer k in $\mathbb{R}$ and $\mathbb{I}$ with $\pi_{\text{bef}(k)}$.

Recall that layer k "sorts" the permutation $\pi_{\text{aft}(k)}$. Unlike $\pi_{\text{aft}(k)}$ the permutation $\pi_{\text{bef}(k)}$ does not imply whether the counterpart of a standard gate (i, j) in layer k in one network is also standard in the other network or not, because $\pi_{\text{bef}(k)}$ is not yet "sorted" by layer k. Hence we do not disallow gates whose counterpart would be nonstandard as the Constraints (6.1i) - (6.1l) do. Instead we make sure that the permutation $\pi_{\text{bef}(k)}$ maps some twist of the gates' inputs. For example if a gate (i, j) in $\mathbb{I}$ and a gate (h, m) in $\mathbb{R}$ are determined to be related by $\pi_{\text{bef}(k)}$, then we have either $\pi_{\text{bef}(k)}(i) = h$ and $\pi_{\text{bef}(k)}(j) = m$

or we have $\pi_{\text{bef}(k)}(i) = m$ and $\pi_{\text{bef}(k)}(j) = h$.

$$\bigwedge_{\substack{k\in L\\ h,m,i,j\in C\\ h<m\\ i<j}} \left(\left(p_{h,i}^{\text{bef}(k)} \wedge p_{m,j}^{\text{bef}(k)} \right) \rightarrow \begin{pmatrix} (g_{h,m}^{k,\mathbb{R}} \leftrightarrow g_{i,j}^{k,\mathbb{I}}) & \text{(6.1q)} \\ \wedge\, (g_{h,m}^{k,\mathbb{R}} \rightarrow \neg t_{h,m}^{k}) & \text{(6.1r)} \end{pmatrix} \right) \tag{6.1s}$$

$$\bigwedge_{\substack{k\in L\\ h,m,i,j\in C\\ h<m\\ i<j}} \left(\left(p_{m,i}^{\text{bef}(k)} \wedge p_{h,j}^{\text{bef}(k)} \right) \rightarrow \begin{pmatrix} (g_{h,m}^{k,\mathbb{R}} \leftrightarrow g_{i,j}^{k,\mathbb{I}}) & \text{(6.1q)} \\ \wedge\, (g_{h,m}^{k,\mathbb{R}} \rightarrow t_{h,m}^{k}) & \text{(6.1r)} \end{pmatrix} \right) \tag{6.1s}$$

$$\bigwedge_{\substack{k\in L\\ h,i\in C}} \left(p_{h,i}^{\text{bef}(k)} \rightarrow \left(nc_{h}^{k,\mathbb{R}} \leftrightarrow nc_{i}^{k,\mathbb{I}} \right) \right) \tag{6.1t}$$

There is a striking similarity between the $4\binom{n}{2}^2(d-p)$ clauses from the Constraints (6.1e), (6.1i) and (6.1j), which relate $\pi_{\text{bef}(k)}$ with layer k, and the $4\binom{n}{2}^2(d-p)$ clauses from the Constraint (6.1q), which relate layer k with $\pi_{\text{aft}(k)}$.

Note that the Constraints (6.1e) - (6.1l) as well as the Constraints (6.1q) and (6.1t) relate $\pi_{\text{bef}(k)}$ and $\pi_{\text{aft}(k)}$ (via layer k). The motivation for the input twists variables is also to relate $\pi_{\text{bef}(k)}$ and $\pi_{\text{aft}(k)}$. Hence the Constraints (6.1q) and (6.1t) are redundant with input twist variables wrt satisfiability. However (6.1q) is quartic in n such that it adds more clauses than the input twist Constraints (6.1n), (6.1o) and (6.1p), which are cubic in n. Hence we prefer the cubic encoding over its quartic alternative.

Though there is a reason to include the Constraints (6.1s) and (6.1t) and that is to increase propagation.

Consider the Constraint (6.1d), which fixes the permutation after the last layer. The solver infers that some variables in layer k are mapped to false by unit propagation on clauses from Constraints (6.1i) - (6.1l). The clauses from Constraints (6.1e) - (6.1h) also yield equivalences between variables in the last layer. If the solver assigns the truth value of a variable in the last layer in one network, it infers the truth value of the corresponding variable in the other network by unit propagation.

In contrast consider the Constraint (6.1c), which fixes the permutation before the first layer. Adding the Constraints (6.1q) and (6.1t) to the encoding yields equivalences between variables in the first layer. Thus if the solver assigns the truth value of a variable in the first layer in one network, it infers the truth value of the corresponding variable in the other network by unit propagation.

If we add the input twist variables to the encoding, then Constraint (6.1r) enables the solver to infer the truth value of the twist variable by unit propagation. With the twist variables known the solver can even infer the permutation after the first layer by unit propagation on clauses from the Constraints (6.1n), (6.1o) and (6.1p).

There are even more constraints that help with propagation but are quartic in n. The Constraints (6.1u), (6.1v) and (6.1w) are variants of the Constraints (6.1e), (6.1q) and (6.1r) respectively with the direction of the implication inverted.

$$\bigwedge_{\substack{k\in L\\ h,m,i,j\in C\\ h<m\\ i<j}} \left(\left(p_{h,i}^{\mathrm{aft}(k)} \leftrightarrow p_{m,j}^{\mathrm{aft}(k)} \right) \leftarrow \left(g_{h,m}^{k,\mathbb{R}} \wedge g_{i,j}^{k,\mathbb{I}} \right) \right) \tag{6.1u}$$

$$\bigwedge_{\substack{k\in L\\ h,m,i,j\in C\\ h<m\\ i<j}} \left(\left(\begin{array}{c} (p_{h,i}^{\mathrm{bef}(k)} \leftrightarrow p_{m,j}^{\mathrm{bef}(k)}) \\ \wedge\, (p_{m,i}^{\mathrm{bef}(k)} \leftrightarrow p_{h,j}^{\mathrm{bef}(k)}) \end{array} \right) \leftarrow \left(g_{h,m}^{k,\mathbb{R}} \wedge g_{i,j}^{k,\mathbb{I}} \right) \right) \tag{6.1v}$$

$$\bigwedge_{\substack{k\in L\\ h,m,i,j\in C\\ h<m\\ i<j}} \left(\begin{array}{l} \left(\neg p_{h,i}^{\mathrm{bef}(k)} \vee \neg p_{m,j}^{\mathrm{bef}(k)} \right) \leftarrow t_{h,m}^{k} \\ \left(\neg p_{m,i}^{\mathrm{bef}(k)} \vee \neg p_{h,j}^{\mathrm{bef}(k)} \right) \leftarrow \neg t_{h,m}^{k} \end{array} \right) \tag{6.1w}$$

6.2 Variable and Clause Savings

In this section we compare the increased size of the encoding with the savings from encoding runs of input sequences in the network where the input sequence has the smaller window size.

We denote the number of layers we fix at the start of the network by p. The depth upper bounds for $n = 17$ and $n = 20$ were found by fixing the first three and the first four layers respectively [27]. Fixing another layer reduces the number of variables and clauses required to encode a sufficient subset of the network runs in the remaining network. This also reduces the potential savings such that we do not consider upper bounds proofs.

Instead we consider lower bound proofs where we only fix the first two layers. Looking at Table 1.1 we see that the latest lower bound result proves that there is no sorting network with $n = 17$ and $d = 9$. Extrapolating the results for $n \leq 17$ I conjecture that the next depth lower bound $d = 10$ first holds for $n = 21$.

6.2.1 Variable Costs of Encoding $\mathbb{R}$

Table 6.1 shows the number of variables added to the sets $\mathcal{G}$, $\mathcal{NC}$, $\mathcal{N}$, $\mathcal{X}$, $\mathcal{P}$, $\mathcal{T}$ and $\mathcal{NT}$ to encode $\mathbb{R}$ and its relation to $\mathbb{I}$. To encode $\mathbb{R}$ we add additional variables to $\mathcal{G}$, $\mathcal{NC}$, $\mathcal{N}$ and $\mathcal{X}$. To encode the relation between $\mathbb{R}$ and $\mathbb{I}$ we add the additional variables $\mathcal{P}$, $\mathcal{T}$ and $\mathcal{NT}$.

	#Variables	$n = 17, d = 9, p = 2$	$n = 21, d = 10, p = 2$
$\mathcal{G}$	$\binom{n}{2}(d-1-p)$	816	1470
$\mathcal{NC}$	$n(d-1-p)$	102	147
$\mathcal{N}$	$\binom{n-1}{2}(d-1-p)$	720	1330
$\mathcal{X}$	$\binom{n-1}{2}(d-1-p)$	720	1330
$\mathcal{P}$	$n^2(d+1-p)$	2312	3969
$\mathcal{T}$	$\binom{n}{2}(d-p)$	952	1680
$\mathcal{NT}$	$n(d-p)$	119	168
		5741	10 094

Table 6.1: *The number of additional variables to encode another permuted and untangled network.*

Note that the Constraints (6.1d) and (6.1e) imply that

$$\bigwedge_{\substack{i,j \in C \\ i<j}} \left(g_{i,j}^{d,\mathbb{I}} \leftrightarrow g_{i,j}^{d,\mathbb{R}} \right).$$

Hence we may encode the two gates on channels i and j in the last layer of $\mathbb{I}$ and $\mathbb{R}$ as a single variable $g_{i,j}^{k,\mathbb{I}} = g_{i,j}^{k,\mathbb{R}}$. Therefore there are no additional gate variables for the last layer such that we subtract d by 1 in the term of the added gate variables. The same applies to the additional $\mathcal{NC}$, $\mathcal{N}$ and $\mathcal{X}$ variables.

Further recall that variables of the form $\min_{i,i}^k$ and $\max_{i,i}^k$ are known to be false such that they must not be encoded. Finally also recall that $\min_{i,i+1}^k = g_{i,i+1}^k = \max_{i,i+1}^k$ such that we also do not need to encode variables of the form $\min_{i,i+1}^k$ or $max_{i,i+1}^k$.

To encode $\mathbb{R}$ and its relation to $\mathbb{I}$, we add 5741 variables for $n = 17$ and 10 094 variables for $n = 21$.

6.2.2 Clause Costs of Encoding $\mathbb{R}$

Table 6.2a shows the number of clauses added to encode $\mathbb{R}$ and its relation to $\mathbb{I}$. We add 682 547 clauses for $n = 17$ and 1 755 784 clauses for $n = 21$. Unsurprisingly the Constraints (6.1e), (6.1i) and (6.1j) contribute most of the clauses, as they are quartic in n. Table 6.2b shows the number of clauses if we further add the constraints from Section 6.1.4, most of which are quartic in n too, to increase propagation. In this case we add significantly more additional clauses, that is 2 497 178 clauses for $n = 17$ and 6 698 512 clauses for $n = 21$.

6.2.3 Variable and Clause Savings from Encoding $\mathbb{R}$ for $n = 17$

Next we contrast the costs of the encoding in variables and clauses with the number of variables and clauses saved by the encoding. For $n = 17$ we draw on Ehlers' work, who

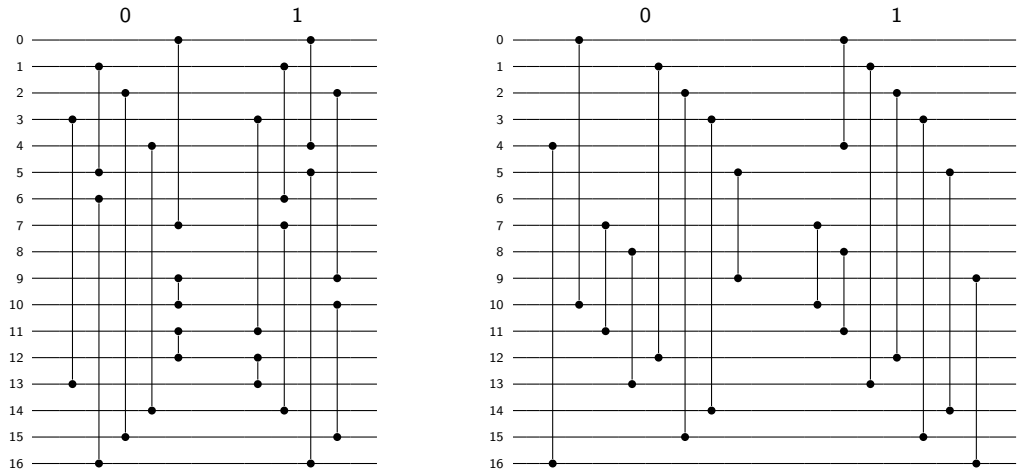
Constraint	#Clauses	$n = 17, d = 9, p = 2$	$n = 21, d = 10, p = 2$
(4.14)	$\binom{n}{2}(d-1-p)$	816	1470
(5.1)	$\binom{n}{2}(d-1-p)$	816	1470
(5.10)	$4\binom{n-1}{2}(d-p)$	3360	6080
(5.11)	$4\binom{n-1}{2}(d-p)$	3360	6080
(5.4)	$6(n-1)(d-p)$	672	960
(6.1a)	$n(1+\binom{n}{2})(d+1-p)$	18 632	39 879
(6.1b)	$n(1+\binom{n}{2})(d+1-p)$	18 632	39 879
(6.1e)	$2\binom{n}{2}^2(d-p)$	258 944	705 600
(6.1f)	$2n^2(d-p)$	4046	7056
(6.1g)	$2n^2(d-p)$	4046	7056
(6.1h)	$2n^2(d-p)$	4046	7056
(6.1i)	$\binom{n}{2}^2(d-p)$	129 472	351 800
(6.1j)	$\binom{n}{2}^2(d-p)$	129 472	351 800
(6.1k)	$2n(d-p)$	238	336
(6.1l)	$2n(d-p)$	238	336
(6.1n)	$3\binom{n}{2}(d-p)$	2856	5040
(6.1o)	$6\binom{n}{2}n(d-p)$	97 104	211 680
(6.1p)	$3n^2(d-p)$	6069	10 584
		682 547	1 755 784

(a) *The number of additional clauses required for the minimum encoding of another permuted and untangled network.*

Constraint	#Clauses	$n = 17, d = 9, p = 2$	$n = 21, d = 10, p = 2$
Table 6.2a		682 547	1 755 784
(6.1s)	$6\binom{n}{2}^2(d-p)$	776 832	2 116 800
(6.1t)	$n^2(d-p)$	2023	3528
(6.1u)	$2\binom{n}{2}^2(d-p)$	258 944	705 600
(6.1v)	$4\binom{n}{2}^2(d-p)$	517 888	1 411 200
(6.1w)	$2\binom{n}{2}^2(d-p)$	258 944	705 600
		2 497 178	6 698 512

(b) *The number of additional clauses to encode another permuted and untangled network with increased propagation.*

Table 6.2



(a) *Prefix* P_0^{17} (b) *Prefix* P_{51}^{17}

Figure 6.1

already optimized representative prefixes for all equivalence classes and solved whether they can be extended to a sorting network on 17 elements of depth 9. In his dissertation he writes "The prefix with index 0 was one of the easiest in our experiments, whereas the one with index 51 was one of the hardest cases" [11]. According to him solving prefix 0 took about 7 hours, whereas solving prefix 51 took about 14 hours. Figure 6.1 shows these prefixes. We denote them by P_0^{17} and P_{51}^{17}. Reversing the order of the channels and untangling yields the prefixes R_0^{17} and R_{51}^{17}.

Notably the equivalence class of prefix 51 produces more outputs than the class of prefix 0. Specifically we have

$$|\text{outputs}(\mathsf{P}_0^{17})| = |\text{outputs}(\mathsf{R}_0^{17})| = |\text{outputs}(\mathsf{P}_0^{17}, \mathsf{R}_0^{17})| = 2592$$
$$|\text{outputs}(\mathsf{P}_{51}^{17})| = |\text{outputs}(\mathsf{R}_{51}^{17})| = |\text{outputs}(\mathsf{P}_{51}^{17}, \mathsf{R}_{51}^{17})| = 4414.$$

Recall from Section 4.2.1 that encoding more comparison network runs than necessary, but not too many, yields the fastest solver runs. Personally I conjecture that theoretically this sweet spot is roughly the same for all prefix classes, e.g. 60 - 65% of $|\text{outputs}(P)|$ for a prefix P. Note that encoding the smallest (by window size) 2000 network runs, covers $\approx 77\%$ of the outputs of prefix 0, whereas it only covers $\approx 45\%$ of the outputs of prefix 51.

While the solving times of prefix 51 may be negatively affected from encoding too few network runs, I believe the main reason for the longer solving time is simply that $|\text{outputs}(\mathsf{P}_{51}^{17})|$ is much greater than $|\text{outputs}(\mathsf{P}_0^{17})|$. I.e. even encoding the optimal percentage of the outputs of prefix 0 or 51 yields a much slower solving run for prefix 51, as its optimally sized encoding is also much larger than that of prefix 0.

window size	0	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
outputs(P_0^{17})	18	15	27	49	71	122	214	254	296	308	204	274	328	160	120	60	72
outputs$_{61\%}$(P_0^{17})	18	15	27	49	71	122	214	254	296	308	204	3	0	0	0	0	0
outputs$_{2000}$(P_0^{17})	18	15	27	49	71	122	214	254	296	308	204	274	148	0	0	0	0
outputs($\mathsf{P}_0^{17}, \mathsf{R}_0^{17}$)	30	26	44	86	140	214	344	272	256	336	160	208	192	140	144	0	0
outputs$_{61\%}$($\mathsf{P}_0^{17}, \mathsf{R}_0^{17}$)	30	26	44	86	140	214	344	272	256	169	0	0	0	0	0	0	0
outputs$_{2000}$($\mathsf{P}_0^{17}, \mathsf{R}_0^{17}$)	30	26	44	86	140	214	344	272	256	336	160	92	0	0	0	0	0

(a) *The window size distribution of the outputs of prefix 0 from Ehlers.*

window size	0	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
outputs(P_{51}^{17})	18	16	30	48	80	136	224	282	468	558	584	492	292	320	448	418	0
outputs$_{61\%}$(P_{51}^{17})	18	16	30	48	80	136	224	282	468	558	584	249	0	0	0	0	0
outputs$_{2000}$(P_{51}^{17})	18	16	30	48	80	136	224	282	468	558	140	0	0	0	0	0	0
outputs($\mathsf{P}_{51}^{17}, \mathsf{R}_{51}^{17}$)	33	29	49	80	116	207	264	368	576	768	686	340	348	254	198	98	0
outputs$_{61\%}$($\mathsf{P}_{51}^{17}, \mathsf{R}_{51}^{17}$)	33	29	49	80	116	207	264	368	576	768	203	0	0	0	0	0	0
outputs$_{2000}$($\mathsf{P}_{51}^{17}, \mathsf{R}_{51}^{17}$)	33	29	49	80	116	207	264	368	576	278	0	0	0	0	0	0	0

(b) *The window size distribution of the outputs of prefix 51 from Ehlers.*

Table 6.3

Further there may be practical reasons to limit the number of encoded runs at e.g. 2000 no matter what, for example the sizes of the caches in the available cache hierarchy. This focus on the encoded number of network runs of prefix outputs also dubiously equates prefix outputs with different window size. Nonetheless I find it appropriate to compare encodings for prefix 0 and 51 where both encodings cover the same percentage of the outputs of prefix 0 or 51. For this comparison I rather arbitrarily choose the average of 45% and 77%, that is 61%.

Recall the definition of outputs(..) from Section 4.2.5 and the definition of outputs(.. , ..) from Section 6.1.2. Their variants outputs$_m$(..) and outputs$_m$(.. , ..) only return the m inputs with the smallest window size. Likewise Their variants outputs$_{m\%}$(..) and outputs$_{m\%}$(.. , ..) only return the m% of all inputs with the smallest window size. The Table 6.3a shows how many sequences there are of each window size in outputs(P_0^{17}) and outputs($\mathsf{P}_0^{17}, \mathsf{R}_0^{17}$). Here the former assumes that only $\mathbb{I}$ with prefix P_0^{17} is encoded, whereas the latter assumes that both $\mathbb{I}$ with prefix P_0^{17} and $\mathbb{R}$ with prefix R_0^{17} are encoded. The variants that are limited to 2000 inputs or 61% of all inputs consider a realistic setting, whereas the other consider the theoretical case where we encode all network runs, i.e. each in one extension of a prefix. Table 6.3b shows the same numbers for prefix P_{51}^{17} (and R_{51}^{17}).

The distribution of these sequences across different window sizes determines how many variables and clauses are necessary to encode the network runs of either all sequences or 61% of all sequences/2000 sequences of minimal window size. The Tables 6.4a and 6.4b show these numbers. In particular they show how many variables and clauses are saved in the encoding of the network runs by encoding $\mathbb{R}$.

To understand the labels of the tables we recall the definition of *#variables*(..) and

	$\#vars_{2000}(..)$	$\#clauses_{2000}(..)$	$\#vars_{61\%}(..)$	$\#clauses_{61\%}(..)$	$\#vars(..)$	$\#clauses(..)$
P_0^{17}	144 992	2 490 936	103 584	1 594 432	213 504	4 240 208
$\mathsf{P}_0^{17}, \mathsf{R}_0^{17}$	124 992	1 880 032	88 720	1 189 720	189 056	3 405 920
savings	20 000	610 904	14 864	404 712	24 448	834 288

(a) *The savings on prefix 0 from encoding another network with its inputs in reverse order.*

	$\#vars_{2000}(..)$	$\#clauses_{2000}(..)$	$\#vars_{61\%}(..)$	$\#clauses_{61\%}(..)$	$\#vars(..)$	$\#clauses(..)$
P_{51}^{17}	133 488	2 067 380	196 464	3 321 500	393 264	8 289 652
$\mathsf{P}_{51}^{17}, \mathsf{R}_{51}^{17}$	120 824	1 723 862	177 888	2 753 744	353 976	6 752 662
savings	12 664	343 518	18 576	567 756	39 288	1 536 990

(b) *The savings on prefix 51 from encoding another network with its inputs in reverse order.*

Table 6.4

$\#clauses(..)$ from Section 5.1. We define the variants $\#vars_m(..)$ and $\#clauses_m(..)$, which quantify over $\text{outputs}_m(..)$. Further we define the variant $\#vars(M, M')$ and $\#clauses(M, M')$, which quantify over $\text{outputs}(M, M')$.

Note that we save more element value variables and clauses in the encoding of the network runs for prefix 0 if we encode 2000 network runs, but we save more for prefix 51 if we encode 61% of all network runs. In the theoretical case where we encode all network runs we save nearly twice as many element value variables and clauses in the encoding of the network runs for prefix 51 as for prefix 0.

Consider the case where we encode the best 2000 runs from the outputs of P_0^{17}. Additionally encoding $\mathbb{R}$ saves 20 000 element value variables and 610 904 clauses in the encoding of the network runs, which are the greatest savings under realistic conditions in our superficial estimation. On the other hand encoding $\mathbb{R}$ adds 5741 variables and 2 497 178 clauses. In this best case encoding $\mathbb{R}$ saves 14 259 variables but adds 1 886 274 clauses. Even if we drop the optional clauses from Section 6.1.4, which are desirable as they facilitate propagation, we still add 682 547 clauses after accounting for the savings.

6.2.4 Asymptotic Variable and Clause Savings

Saving variables at the cost of adding additional clauses may or may not reduce the solving time. However the number of network runs to encode grows exponentially as there are 2^n possible input sequences. Recall from Section 4.2.6 that the Green filter is conjectured to be optimal in terms of number of outputs. Even in the case of the Green filter equivalence class of prefixes we have

n	1	2	4	8	16	32	64
Layers	0	1	2	3	4	5	6
Figure 6.2	a	b	c	d	e		
$\#outputs(..)$	2	3	6	20	168	7581	7828354

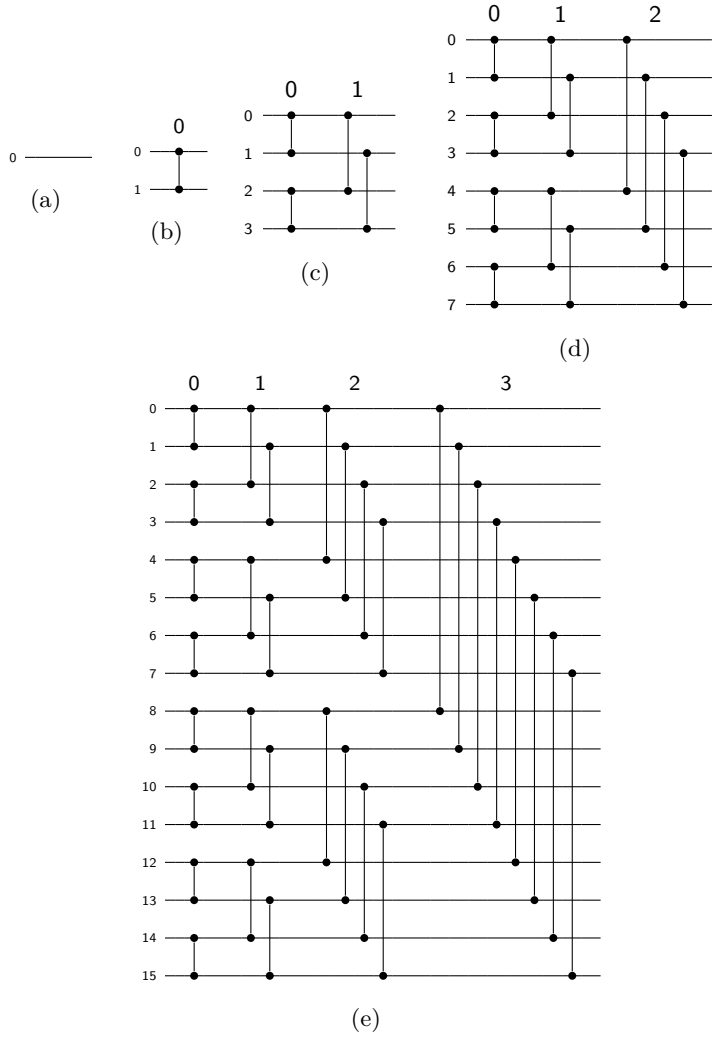


Figure 6.2: *Green filters*

where $2, 3, 6, 20, 169, 7581, 7828354$ are the first seven Dedekind numbers [50]. Note that the Green filter for $n = 2^i$ consists of i layers.

I conjecture that for every prefix equivalence class there is a prefix pair that achieves savings over the optimal prefix that are linear in the number of outputs of that equivalence class. Under this assumption the savings grow exponentially, whereas the overhead of encoding another network with inputs permuted grows only quartic in n. If necessary we may weaken this assumption to only apply to two-layer prefix classes instead of prefix classes with up to $\lfloor \log(n) \rfloor$ layers.

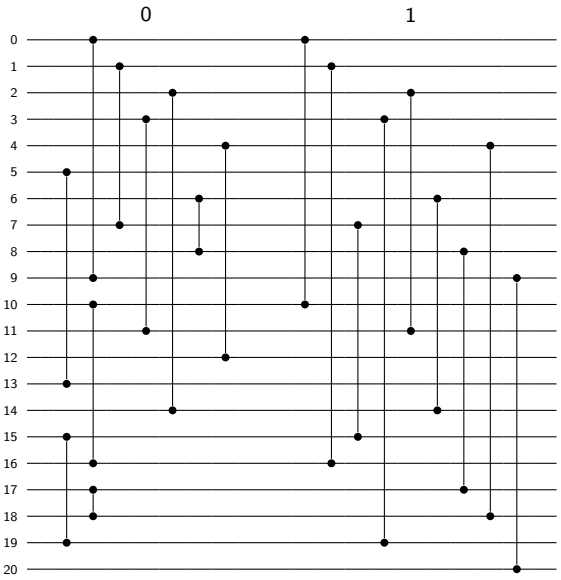


Figure 6.3: *Prefix* P^{21}

6.2.5 Variable and Clause Savings from Encoding $\mathbb{R}$ for $n = 21$

To put this conjecture to the test we consider an optimized two layer prefix for $n = 21$, which we name P^{21}. Figure 6.3 shows P^{21}. To optimize the prefix I have used Ehlers' greedy algorithm, i.e. just like P_0^{17} and P_{51}^{17} it is optimized but not necessarily optimal.

Likewise reversing the order of the channels in P^{21} and untangling yields the prefix R^{21}. The number of outputs of this prefix equivalence class is

$$|\text{outputs}(\mathsf{P}^{21})| = |\text{outputs}(\mathsf{R}^{21})| = |\text{outputs}(\mathsf{P}^{21}, \mathsf{R}^{21})| = 32838.$$

We consider encoding only the smallest (by window size) 20 000 network runs, which is roughly 61% of all 32 838 outputs of the prefix class. Table 6.5 shows the saved number of element value variables and clauses if we encode either only 20 000 network runs or all network runs. In the former less theoretic case we save 203 832 element value variables and 9 268 896 clauses. In contrast encoding $\mathbb{R}$ adds 10 094 variables and 6 698 512 clauses. Overall we save 193 738 variables and 2 570 384 clauses by encoding $\mathbb{R}$.

Unfortunately realistically the lower bound for $n = 21$ will not be tackled anytime soon, considering the effort it took to achieve the current best running times for $n = 17$.

	$\#vars_{20000}(..)$	$\#clauses_{20000}(..)$	$\#vars(..)$	$\#clauses(..)$
P^{21}	2 299 050	54 917 216	4 469 598	127 612 512
$\mathsf{P}^{21}, \mathsf{R}^{21}$	2 095 218	45 648 320	4 074 201	106 382 704
savings	203 832	9 268 896	395 397	21 229 808

Table 6.5: *The savings on prefix* P^{21} *from encoding another network with its inputs in reverse order.*

7 Graph Encoding

In the Sections 6.1.3 and 6.1.4 we gave an encoding of the relation between network $\mathbb{R}$ and network $\mathbb{I}$. In this section we consider an alternative encoding of this relation. Specifically we encode a network graph G such that $\mathbb{I}$ and $\mathbb{R}$ are the standard id- and standard rev-line representation of G respectively where $\mathsf{rev} = \left(\begin{smallmatrix} 0 & 1 & .. & n-2 & n-1 \\ n-1 & n-2 & .. & 1 & 0 \end{smallmatrix}\right)$.

We first consider a faithful encoding of a comparison network graph in Section 7.1. We discuss various variants of the encoding in Section 7.2 that make tradeoffs in the number of variables and clauses. In Section 7.3 we give a detailed encoding of an unfaithful graph encoding that is a strict improvement of the faithful graph encoding.

Furthermore in Section 7.4 we discuss how to utilize the graph encoding to break the symmetry of twisting inputs. Finally we discuss the costs and benefits of the graph encoding in Section 7.5.

7.1 Faithful Graph Encoding

There is a set G of numbers for gates in the same layer. There are at most $\lfloor \frac{n}{2} \rfloor$ gates in the same layer. Hence we have $G = \{0, .., \lfloor \frac{n}{2} \rfloor - 1\}$.

The set of node numbers in stage k is defined as

$$\mathrm{ns}(k) = \begin{cases} G & k \in L \\ C & \text{otherwise} \end{cases}$$

The set of input and output ports of a node in stage k is given by the functions $\mathrm{inpts}, \mathrm{outpts} : S \to 2^P$ respectively, defined as

$$\mathrm{inpts}(k) = \begin{cases} \emptyset & k = \mathrm{in} \\ \{\mathrm{in1}, \mathrm{in2}\} & k \in L \\ \{\mathrm{in}\} & k = \mathrm{out} \end{cases}$$

$$\mathrm{outpts}(k) = \begin{cases} \{\mathrm{out}\} & k = \mathrm{in} \\ \{\min, \max\} & k \in L \\ \emptyset & k = \mathrm{out} \end{cases} .$$

In the following we sometimes quantify over all input or output ports in some stage k. For example we may refer to some $a{:}p$ in $\text{ports}(k)$ to identify port p of the ath node in stage k. For this we define:

$$\begin{aligned} \text{op}(k) &= \text{ns}(k) \times \text{outpts}(k) \\ \text{ip}(k) &= \text{ns}(k) \times \text{inpts}(k) \\ \text{ports}(k) &= \text{ip}(k) \cup \text{op}(k) \end{aligned}$$

Actually a and p are just components of a tuple, but since p identifies a port on the ath node we write such tuples in port notation as $a{:}p$ instead of (a, p).

7.1.1 Encoding of a Comparison Network Graph

The variable ng_a^k models that the ath gate in layer k is not used. That means that no arcs connect with that gate.

$$\mathcal{NG} = \{ng_a^k \mid k \in L, a \in G\}$$

The variable $w_{a:p,b:q}^{k,l}$ models an arc from port p on the ath node in stage k to port q on the bth node in stage l.

$$\mathcal{W} = \{w_{a:p,b:q}^{k,l} \mid k, l \in S, a{:}p \in \text{op}(k), b{:}q \in \text{ip}(l), k < l\}$$

We add the Constraints (7.1a) and (7.1b) to ensure that every input port and every output port respectively has at most one arc attached. Recall that in a comparison network graph every port has exactly one arc attached. In our encoding the ports of the ath node in stage k either all have exactly one arc attached if $ng_a^k \mapsto \text{true}$ or alternatively the node's ports all have no arc attached and the node is not part of the comparison network if $ng_a^k \mapsto \text{false}$.

$$\bigwedge_{\substack{l \in S \\ b:p \in \text{ip}(l)}} \text{eo}\left(\begin{array}{l} \{w_{a:o,b:p}^{k,l} \mid \begin{array}{l} k \in S \\ k < l \end{array}, a{:}o \in \text{op}(k)\} \\ \cup \{ng_b^l \mid l \in L\} \end{array}\right) \tag{7.1a}$$

$$\bigwedge_{\substack{l \in S \\ b:p \in \text{op}(l)}} \text{eo}\left(\begin{array}{l} \{w_{b:p,c:q}^{l,m} \mid \begin{array}{l} m \in S \\ m > l \end{array}, c{:}q \in \text{ip}(m)\} \\ \cup \{ng_b^l \mid l \in L\} \end{array}\right) \tag{7.1b}$$

7.1.2 Encoding of Channel Labelings

We introduce variables that model the two channel labelings $chan_{\mathbb{I}}$ and $chan_{\mathbb{R}}$ that determine the channel networks $\mathbb{I}$ and $\mathbb{R}$ respectively. Recall from Section 2.3 that we can alternatively label the ports instead of the arcs. We label the ports, because it requires fewer variables.

The variable ${}_iy^k_{a:p}$ models that for the ath node in stage k, which we denote by c, we have $chan_r(c{:}p) = i$.

$$\mathcal{Y} = \{{}_iy^{k,r}_{a:p} \mid r \in R, k \in S, a{:}p \in \text{ports}(k), i \in C\}$$

Hence the number of channel label variables is linear in n and linear in d. Note that if we had labeled the arcs instead, we would have had n times as many channel label variables as there are arc variables. The number of arc variables is quadratic in n and and quadratic in d.

Since a channel labeling $chan_r$ for some r in R is a function, any port is labeled with a channel unless it is a port of a gate that is not part of the comparison network.

$$\bigwedge_{\substack{r\in R\\ k\in S\\ a:p\in\text{ports}(k)}} \text{eo}\left(\begin{array}{c}\{{}_iy^{k,r}_{a:p} \mid i \in C\}\\ \cup\{ng^{k,r}_a \mid k \in L\}\end{array}\right) \tag{7.1c}$$

Again recall from Section 2.3 that parallel comparison nodes are labeled by distinct channel numbers, which is why we can consider a channel labeling to be invertible on a per layer basis. Thus we add a constraint to make sure that $chan_{\mathbb{I}}$ and $chan_{\mathbb{R}}$ are injective on a per layer basis. To be specific they are injective on per input ports of a stage basis and on a per output ports of a stage basis. For the output ports we add the Constraint (7.1d) and for the input ports we add the Constraint (7.1e).

$$\bigwedge_{\substack{r\in R\\ k\in S\\ i\in C}} \text{eo}\left(\begin{array}{c}\{{}_iy^{k,r}_{a:o} \mid a{:}o \in \text{op}(k)\}\\ \cup\{nc^{k,r}_i \mid k \in L\}\end{array}\right) \tag{7.1d}$$

$$\bigwedge_{\substack{r\in R\\ k\in S\\ i\in C}} \text{eo}\left(\begin{array}{c}\{{}_iy^{k,r}_{a:p} \mid a{:}p \in \text{ip}(k)\}\\ \cup\{nc^{k,r}_i \mid k \in L\}\end{array}\right) \tag{7.1e}$$

Further according to the definition of channel labelings each channel i is a path from some network input nodes to the ith network output node. Thus we fix the channel labels of the network output nodes:

$$\bigwedge_{\substack{i\in C\\ r\in R}} {}_iy^{\text{out},r}_{i:\text{in}} \tag{7.1f}$$

Theoretically we can choose the encoded graph G to be the π-graph representation of channel network $\mathbb{I}$ for any π such that the channel network $\mathbb{R}$ is the $(\mathsf{rev} \circ \pi)$-line representation of G. We do not want the SAT-solver try out the assignments for every π such that we break this symmetry by choosing $\pi = \mathsf{id}$. Hence as stated before $\mathbb{I}$ and $\mathbb{R}$ are the standard id- and rev-line representations of G. Thus we likewise fix the channel labels of the network input nodes.

$$\bigwedge_{i \in C} {}_i y^{\mathrm{in},\mathbb{I}}_{i:\mathrm{out}} \tag{7.1g}$$

$$\bigwedge_{i \in C} {}_{n-i-1} y^{\mathrm{in},\mathbb{R}}_{i:\mathrm{out}} \tag{7.1h}$$

Recall that we have $chan_r(c{:}p) = chan_r(a) = chan_r(d{:}q)$ for any arc $a = (c{:}p, d{:}q)$ and any network r from R. This is why we may label the ports instead of the arcs. We encode this property with the following constraint.

$$\bigwedge_{\substack{w^{k,l}_{a:o,b:p} \in \mathcal{W} \\ r \in R \\ i \in C}} \left(\begin{array}{l} \left(w^{k,l}_{a:o,b:p} \rightarrow ({}_i y^{k,r}_{a:o} \leftrightarrow {}_i y^{l,r}_{b:p})\right) \\ \wedge \left(w^{k,l}_{a:o,b:p} \leftarrow ({}_i y^{k,r}_{a:o} \wedge {}_i y^{l,r}_{b:p} \wedge \bigwedge_{\substack{m \in L \\ k<m<l}} nc^{m,r}_i)\right) \end{array} \right) \tag{7.1i}$$

Since channels are paths from some network input node to some network output node, a channel must not start or end at a comparison node. Therefore we add the following for every ath comparison node in layer k.

$$\bigwedge_{\substack{r \in R \\ k \in L \\ i \in C \\ a \in G}} \left({}_i y^{k,r}_{a:\min} \vee {}_i y^{k,r}_{a:\max}\right) \leftrightarrow \left({}_i y^{k,r}_{a:\mathrm{in1}} \vee {}_i y^{k,r}_{a:\mathrm{in2}}\right) \tag{7.1j}$$

Additionally because channels are paths and not trees, the two input or the two output ports of a comparison node cannot both have the same channel label. The Constraints (7.1d) and (7.1e) already take care of that. Thus our constraints encode that each channel i is a path from some network input node to the ith network output node such that the label variables $\mathcal{Y}$ encode a proper id-channel labeling for $\mathbb{I}$ and a proper rev-channel labeling for $\mathbb{R}$.

Since we want $chan_{\mathbb{I}}$ and $chan_{\mathbb{R}}$ to be standard channel labelings we add a constraint that disallows channel labels on a comparison node's output ports such that the corresponding gate in the line representation would be a max-min gate.

$$\bigwedge_{\substack{r \in R \\ k \in L \\ a \in G \\ i,j \in C \\ j<i}} \left(\neg\, {}_i y^{k}_{a:\min} \vee \neg\, {}_j y^{k}_{a:\max}\right) \tag{7.1k}$$

Note that we filter the quantified variables with the predicate $j < i$ instead of $i < j$.

7.1.3 Encoding of Graph- and Line-Representations

We encode the graph and line representations between the encoded comparison network graph G and the standard channel networks $\mathbb{I}$ and $\mathbb{R}$ according to their definition.

The ath comparison node in layer k, which we denote by c, corresponds to a gate (i, j) in layer k where $chan(c{:}\min) = i$ and $chan(c{:}\max) = j$. Therefore we add a constraint for gate variables:

$$\bigwedge_{\substack{g_{i,j}^{k,r} \in \mathcal{G} \\ a \in G}} \left(\begin{array}{c} \left(g_{i,j}^{k,r} \rightarrow ({}_i y_{a:\min}^{k,r} \leftrightarrow {}_j y_{a:\max}^{k,r})\right) \\ \wedge \left(g_{i,j}^{k,r} \leftarrow ({}_i y_{a:\min}^{k,r} \wedge {}_j y_{a:\max}^{k,r})\right) \end{array} \right) \tag{7.1l}$$

To increase propagation we also add constraints for min and max variables.

$$\bigwedge_{\min_{i,n-1}^{k,r} \in \mathcal{N}_{0,n-1}} \left(\min_{i,n-1}^{k,r} \leftrightarrow \bigvee_{a \in G} {}_i y_{a:\min}^{k,r} \right) \tag{7.1m}$$

$$\bigwedge_{\max_{0,i}^{k,r} \in \mathcal{X}_{0,n-1}} \left(\max_{0,i}^{k,r} \leftrightarrow \bigvee_{a \in G} {}_i y_{a:\max}^{k,r} \right) \tag{7.1n}$$

Note that Constraint (7.1i) already maps variables in $\mathcal{NC}$. The implication in the $\rightarrow$ direction is incomplete, but we revise this constraint in Section 7.3 anyway. However the representation of nc variables is already achieved by the Constraints (7.1d) and (7.1e), because they imply the Constraints (7.1o) and (7.1p) respectively. Specifically the latter two constraints are the former two constraints without all clauses that contain no variable from $\mathcal{NC}$.

$$\bigwedge_{nc_i^{k,r} \in \mathcal{NC}} \left(nc_i^{k,r} \leftrightarrow \bigwedge_{a:p \in \mathrm{op}(k)} \neg {}_i y_{a:p}^{k,r} \right) \tag{7.1o}$$

$$\bigwedge_{nc_i^{k,r} \in \mathcal{NC}} \left(nc_i^{k,r} \leftrightarrow \bigwedge_{a:p \in \mathrm{ip}(k)} \neg {}_i y_{a:p}^{k,r} \right) \tag{7.1p}$$

7.2 Faithful Graph Encoding without Input Port Labels

In this section we consider alternative variants of the faithful comparison network graph encoding that make trade offs between variables and clauses. We consider variants that remove variables but introduce additional clauses. We also consider variants that intro additional variables to save clauses.

Note that the line representation only uses channel labels on the output ports. Actually we may halve the number of channel label variables by not labeling input ports. In this section we explore such variants of the graph encoding, although we do not actually commit to any such variant of the encoding wrt the following sections.

To this end we change the function ports in the definition of $\mathcal{Y}$ to the function op. We make an exception for the network output stage, whose nodes have no output ports.

$$\text{ports}(k) = \begin{cases} \text{ip}(k) & \text{if } k = \text{out} \\ \text{op}(k) & \text{otherwise} \end{cases}$$

This changes the set of channel label variables $\mathcal{Y}$, which uses ports in its definition.

Note that we still have input ports in our encoding, because the arc variables still connect output ports to input ports.

Removing the channel label variables on input ports affects several constraints:

- We change the Constraint (7.1i) such that the channel label variables that previously referred to input ports in stage r now refer to output ports in stage r. This makes the Constraint (7.1j) redundant such that we remove it.
- We only quantify over op instead of ports in (7.1c).
- We remove the Constraint (7.1e).

In the Sections 7.2.1 and 7.2.2 we consider different variants to combine (7.1i) and (7.1j) to a new constraint.

7.2.1 Alternative Comparison Network Definition

As I alluded to in Section 3.0.1 we may also define comparison networks as graphs where comparison nodes only have a single input port with exactly two arcs attached.

We can easily change to this definition, since we already changed our constraints to ignore the channel labels on input ports. We redefine inpts such that comparison nodes only have a single input port (in) with exactly two arcs attached.

$$\text{inpts}(k) = \begin{cases} \emptyset & k = \text{in} \\ \{\text{in}\} & k \in L \\ \{\text{in}\} & k = \text{out} \end{cases}$$

Additionally we change the exactly-one Constraint (4.1) in the Constraint (7.1a) to an exactly-two Constraint if $l \neq \text{out}$.

Finally we combine the Constraints (7.1i) and (7.1j) to the following constraint:

$$\bigwedge_{\substack{w^{j,l}_{a:o,c:\mathrm{in}}\in\mathcal{W}\\ w^{k,l}_{b:p,c:\mathrm{in}}\in\mathcal{W}\\ j<k\ \vee\ a<b\\ l\neq\mathrm{out}}}\ \bigwedge_{\substack{r\in R\\ i\in C}}\left(w^{k,l}_{a:o,c:\mathrm{in}}\wedge w^{k,l}_{b:p,c:\mathrm{in}}\rightarrow\left({}_iy^{k,r}_{a:o}\vee {}_iy^{k,r}_{b:p}\leftrightarrow \bigvee_{q\in\mathrm{outpts}(l)} {}_iy^{l,r}_{c:q}\right)\right) \tag{7.2}$$

$$\bigwedge_{\substack{w^{j,l}_{a:o,c:\mathrm{in}}\in\mathcal{W}\\ l=\mathrm{out}}}\ \bigwedge_{\substack{r\in R\\ i\in C}}\left(w^{k,l}_{a:o,c:\mathrm{in}}\rightarrow\left({}_iy^{k,r}_{a:o}\leftrightarrow {}_iy^{l,r}_{c:\mathrm{in}}\right)\right) \tag{7.3}$$

$$\bigwedge_{w^{k,l}_{a:o,c:\mathrm{in}}\in\mathcal{W}}\ \bigwedge_{\substack{r\in R\\ i\in C}}\ \bigwedge_{q\in\mathrm{outpts}(l)}\left(w^{k,l}_{a:o,c:\mathrm{in}}\leftarrow\left({}_iy^{k,r}_{a:o}\wedge {}_iy^{l,r}_{c:q}\wedge \bigwedge_{\substack{m\in L\\ k<m<l}} nc^{m,r}_i\right)\right) \tag{7.4}$$

The $\leftarrow$ direction is similar to (7.1i). However we split the $\rightarrow$ direction into two constraints that distinguish the cases $l \neq \mathrm{out}$ and $l = \mathrm{out}$. In the former case we have two ingoing arcs to a comparison node such that the premiss includes both arcs. Only a single arc does not suffice to infer to which channel label variables (7.1j) applies. Unfortunately this means that the Constraint (7.2) is now quartic in n, whereas before neither (7.1i) nor (7.1j) was.

To be exact if we consider the CNF of Constraint (7.1j), we see that only 2 of the 4 clauses require both arcs in the premiss. Thus fortunately only 2 of the 4 clauses in the Constraint (7.2) are quartic in n while the other 2 are cubic in n:

$$\bigwedge_{\substack{w^{k,l}_{a:o,c:\mathrm{in}}\in\mathcal{W}\\ l\neq\mathrm{out}}}\ \bigwedge_{\substack{r\in R\\ i\in C}}\left(w^{k,l}_{a:o,c:\mathrm{in}}\rightarrow\left(\quad {}_iy^{k,r}_{a:o}\rightarrow \bigvee_{q\in\mathrm{outpts}(l)} {}_iy^{l,r}_{c:q}\right)\right)$$

$$\bigwedge_{\substack{w^{j,l}_{a:o,c:\mathrm{in}}\in\mathcal{W}\\ w^{k,l}_{b:p,c:\mathrm{in}}\in\mathcal{W}\\ j<k\ \vee\ a<b\\ l\neq\mathrm{out}}}\ \bigwedge_{\substack{r\in R\\ i\in C}}\ \bigwedge_{q\in\mathrm{outpts}(l)}\left(w^{k,l}_{a:o,c:\mathrm{in}}\wedge w^{k,l}_{b:p,c:\mathrm{in}}\rightarrow\left({}_iy^{k,r}_{a:o}\vee {}_iy^{k,r}_{b:p}\leftarrow {}_iy^{l,r}_{c:q}\right)\right)$$

7.2.2 Encoding Swap Variables

Note that combining the Constraints (7.1i) and (7.1j) yields a new constraint that is quartic in n even with an encoding of a normally defined comparison network graph.

But if we distinguish the input ports of comparison nodes, we can introduce additional swap variables to achieve a constraint that is only cubic in n. Specifically the variable $s^{k,r}_a$ models that $chan_r(a\colon \mathrm{min}) = chan_r(a\colon \mathrm{in1})$ and $chan_r(a\colon \mathrm{max}) = chan_r(a\colon \mathrm{in2})$ for every ath comparison node in layer k and any network r in R. If the variable is mapped

to false, it instead models that $chan_r(a\colon \min) = chan_r(a\colon \text{in2})$ and $chan_r(a\colon \max) = chan_r(a\colon \text{in1})$.

$$\mathcal{S} = \{s_a^{k,r} \mid r \in R, k \in L, a \in G\}$$

Let the functions $\text{swap}, \text{nosw} : P \to P$ be defined as

$$\text{swap}(p) = \begin{cases} \max & p = \text{in1} \\ \min & p = \text{in2} \end{cases}$$

$$\text{nosw}(p) = \begin{cases} \min & p = \text{in1} \\ \max & p = \text{in2} \end{cases}$$

The procedure to combine the Constraints (7.1i) and (7.1j) is similar to before. The constraint constraint corresponds to (7.1i). We make a case distinction whether the arc connects to a comparison node in a layer or to a network output node in the output stage. In the former case the swap variable determines, which channel label variables on the output ports are part of the constraint.

$$\bigwedge_{\substack{w_{a:o,b:p}^{k,l} \in \mathcal{W} \\ l \neq \text{out}}} \bigwedge_{\substack{r \in R \\ i \in C}} \left(\begin{array}{l} \left(s_b^{l,r} \rightarrow \left(\begin{array}{l} \left((w_{a:o,b:p}^{k,l}) \rightarrow ({}_i y_{a:o}^{k,r} \leftrightarrow {}_i y_{b:\text{swap}(p)}^{l,r}) \right) \\ \wedge \left((w_{a:o,b:p}^{k,l}) \leftarrow ({}_i y_{a:o}^{k,r} \wedge {}_i y_{b:\text{swap}(p)}^{l,r} \wedge \bigwedge_{\substack{m \in L \\ k<m<l}} nc_i^{m,r}) \right) \end{array} \right) \right) \\ \wedge \left(\neg s_b^{l,r} \rightarrow \left(\begin{array}{l} \left((w_{a:o,b:p}^{k,l}) \rightarrow ({}_i y_{a:o}^{k,r} \leftrightarrow {}_i y_{b:\text{nosw}(p)}^{l,r}) \right) \\ \wedge \left((w_{a:o,b:p}^{k,l}) \leftarrow ({}_i y_{a:o}^{k,r} \wedge {}_i y_{b:\text{nosw}(p)}^{l,r} \wedge \bigwedge_{\substack{m \in L \\ k<m<l}} nc_i^{m,r}) \right) \end{array} \right) \right) \end{array} \right) \tag{7.5}$$

$$\bigwedge_{\substack{w_{a:o,b:\text{in}}^{k,l} \in \mathcal{W} \\ l = \text{out}}} \bigwedge_{\substack{r \in R \\ i \in C}} \left(\begin{array}{l} \left(w_{a:o,b:\text{in}}^{k,l} \rightarrow ({}_i y_{a:o}^{k,r} \leftrightarrow {}_i y_{b:\text{in}}^{l,r}) \right) \\ \wedge \left(w_{a:o,b:\text{in}}^{k,l} \leftarrow ({}_i y_{a:o}^{k,r} \wedge {}_i y_{b:\text{in}}^{l,r} \wedge \bigwedge_{\substack{m \in L \\ k<m<l}} nc_i^{m,r}) \right) \end{array} \right) \tag{7.6}$$

7.3 Decomposition into Arcs between Adjacent Stages

We do not commit to any of the alternative encodings of a comparison network graph outlined in Section 7.2. Instead we consider yet another alternative graph encoding. Again we use the faithful graph encoding from Section 7.1 as a base template. However this time we do not faithfully encode the comparison network graph. Instead we encode a graph that is obtained from decomposing arcs in the comparison network graph. Unlike the encodings from Section 7.2 I believe that the variant in this section is strictly better, as it reduces both the number of variables and clauses.

Note that the number of arc variables $|\mathcal{W}|$ is quadratic in d. As a consequence the number of clauses in each of the Constraints (7.1a), (7.1b) and (7.1i) is quadratic in d. Specifically consider the clauses from the Constraint (4.2) in the Constraints (7.1a) and (7.1b). Although these clauses for e.g. Constraint (7.1a) are of the form $\neg w^{k,l}_{a:o,b:p} \vee \neg w^{m,l}_{c:q,b:p}$, which is cubic in d, we can theoretically get away with only adding them for $\max(k,m)+1=l$ if we combine the Constraints (4.14) and (4.15) to fill both $\mathbb{I}$ and $\mathbb{R}$ with redundant gates such that at most one input is not used by a gate in the preceding layer (and similarly that at most one output is not used by a gate in the succeeding layer for Constraint (7.1b). While this propagates worse, it actually reduces the number of clauses in the Constraints (7.1a) and (7.1b) from cubic in d to quadratic in d. Recall that we know from Section 5.2 that it is legal to combine the Constraints (4.14) and (4.15). Of course this trick also works for e.g. Constraint (7.2).

In contrast the encoding of the relation between $\mathbb{R}$ and $\mathbb{I}$ from Chapter 6 gets by with variable and clause costs that are linear in d. It turns out that there is a way to encode a comparison network graph with a number of arc variables that is linear in d such that all constraints are linear in d too.

The idea is to decompose arcs between nodes in nonadjacent stages into a path of arcs that connect nodes from adjacent stages. This way there are only arcs between adjacent stages k and $k+1$ such that the number of arc variables is linear in d. We introduce additional phantom nodes with a single input port and a single output port in each layer to function as intermediate nodes on such a path that is the decomposition of single arc.

One way to introduce phantom nodes is to simply add n additional nodes to every layer. This would double the input ports and output ports in every layer and it would quadruple the number of arc variables for arcs between two adjacent layers. We choose a different way to introduce the phantom nodes to avoid this increase in arc variables.

We change the semantics of the variables in $\mathcal{NG}$. As before $ng^k_a \mapsto \text{false}$ indicates that the ath node in layer k is part of the comparison network graph and functions as a comparison node. However we change $ng^k_a \mapsto \text{true}$ to mean that the ath node in layer k is split into two phantom nodes where one phantom node has input port $a{:}\text{in1}$ and output port $a{:}\text{min}$ and the other phantom node has input port $a{:}\text{in2}$ and output port $a{:}\text{max}$. In particular every port now has exactly one arc attached invariant of whether ng^k_a is mapped to true or false.

Additionally if n is odd we introduce a single phantom node to each layer. We name this node unused and we refer to its ports as unused:in and unused:out.

We use a different definition for op and ip to include the ports of the unused node. We name the new versions opn and ipn due to the invariant $|\text{opn}(k)| = n = |\text{ipn}(k)|$ for any

stage k. Likewise we obtain portsn, opnBef and ipnAft.

$$\operatorname{opn}(k) = \begin{cases} \operatorname{op}(k) \cup \{\text{unused}\colon\text{out}\} & \text{if } n \text{ is odd} \\ \operatorname{op}(k) & \text{otherwise} \end{cases}$$

$$\operatorname{ipn}(k) = \begin{cases} \operatorname{ip}(k) \cup \{\text{unused}\colon\text{in}\} & \text{if } n \text{ is odd} \\ \operatorname{ip}(k) & \text{otherwise} \end{cases}$$

$$\operatorname{portsn}(k) = \operatorname{ipn}(k) \cup \operatorname{opn}(k)$$

$$\operatorname{opnBef} = \operatorname{opn} \circ \operatorname{stBef}$$

$$\operatorname{ipnAft} = \operatorname{ipn} \circ \operatorname{stAft}$$

We redefine the set of arc variables $\mathcal{W}$ to use $k \in B$, opnBef and ipnAft.

$$\mathcal{W} = \{w^k_{a:p,b:q} \mid k \in B, a\colon p \in \operatorname{opnBef}(k), b\colon q \in \operatorname{ipnAft}(k)\}$$

Note that we now quantify k over B, whereas before we quantified two variables over S. Since we restrict the encoding to arcs between adjacent layers, the two variables from S are now $\operatorname{stBef}(k)$ and $\operatorname{stAft}(k)$. Crucially $|\mathcal{W}|$ is now linear in d.

In the following we only give constraints for the case that n is odd. The constraints for the case that n is even can be obtained by dropping all clauses whose variables refer to unused:in or unused:out. We recall the constraints from the faithful graph encoding on the left side and show the corresponding revised version on the right side.

As mentioned before now every port has exactly one arc attached invariant of whether ng^k_a is mapped to true or false. Thus we remove the variables from $\mathcal{NG}$ from the encoding of the respective constraints. In a satisfying assignment any two adjacent stages are connected by a perfect matching of size n.

$$\bigwedge_{\substack{l \in S \\ b:p \in \operatorname{ip}(l)}} \operatorname{eo}\left(\begin{array}{l} \{w^{k,l}_{a:o,b:p} \mid \begin{smallmatrix} k \in S \\ k < l \end{smallmatrix},\ a\colon o \in \operatorname{op}(k)\} \\ \cup \{ng^l_b \mid l \in L\} \end{array}\right) \tag{7.1a}$$

$$\bigwedge_{\substack{l \in S \\ b:p \in \operatorname{op}(l)}} \operatorname{eo}\left(\begin{array}{l} \{w^{l,m}_{b:p,c:q} \mid \begin{smallmatrix} m \in S \\ m > l \end{smallmatrix},\ c\colon q \in \operatorname{ip}(m)\} \\ \cup \{ng^l_b \mid l \in L\} \end{array}\right) \tag{7.1b}$$

(7.7a)
$$\bigwedge_{\substack{k \in B \\ b:p \in \operatorname{ipnAft}(k)}} \operatorname{eo}\Big(\{w^k_{a:o,b:p} \mid a\colon o \in \operatorname{opnBef}(k)\}\Big)$$

(7.7b)
$$\bigwedge_{\substack{k \in B \\ b:p \in \operatorname{opnBef}(k)}} \operatorname{eo}\Big(\{w^k_{b:p,c:q} \mid c\colon q \in \operatorname{ipnAft}(k)\}\Big)$$

We also redefine the set of channel label variables $\mathcal{Y}$ to use portsn:

$$\mathcal{Y} = \{{}_i y^{k,r}_{a:p} \mid r \in R, k \in S, a{:}p \in \text{portsn}(k), i \in C\}$$

Constraint (7.1c) ensures that every port that is part of the network has exactly one channel label. Now every port has exactly one channel label, since every port is now part of the network. Again we remove the variables from $\mathcal{NG}$ from this constraint, because the ports of the ath node in layer k are now labeled even if $ng^k_a \mapsto \text{true}$, because then they are a port of a phantom node.

$$\bigwedge_{\substack{r \in R \\ k \in S \\ a:p \in \text{ports}(k)}} \text{eo}\left(\begin{array}{c} \{{}_i y^{k,r}_{a:p} \mid i \in C\} \\ \cup \{ng^{k,r}_a \mid k \in L\} \end{array}\right) \tag{7.1c}$$

(7.7c)
$$\bigwedge_{\substack{r \in R \\ k \in S \\ a:p \in \text{portsn}(k)}} \text{eo}\Big(\{{}_i y^{k,r}_{a:p} \mid i \in C\}\Big)$$

Every channel is associated with exactly one of the input/output ports of every stage. Unlike previously this is the case even if a channel is not used by a comparator in a layer. Note that we include the unused node by quantifying over $\text{opn}(k)$ and $\text{ipn}(k)$ instead of $\text{op}(k)$ and $\text{ip}(k)$.

$$\bigwedge_{\substack{r \in R \\ k \in S \\ i \in C}} \text{eo}\left(\begin{array}{c} \{{}_i y^{k,r}_{a:o} \mid a{:}o \in \text{op}(k)\} \\ \cup \{nc^{k,r}_i \mid k \in L\} \end{array}\right) \tag{7.1d}$$

$$\bigwedge_{\substack{r \in R \\ k \in S \\ i \in C}} \text{eo}\left(\begin{array}{c} \{{}_i y^{k,r}_{a:p} \mid a{:}p \in \text{ip}(k)\} \\ \cup \{nc^{k,r}_i \mid k \in L\} \end{array}\right) \tag{7.1e}$$

(7.7d)
$$\bigwedge_{\substack{r \in R \\ k \in S \\ i \in C}} \text{eo}\Big(\{{}_i y^{k,r}_{a:o} \mid a{:}o \in \text{opn}(k)\}\Big)$$

(7.7e)
$$\bigwedge_{\substack{r \in R \\ k \in S \\ i \in C}} \text{eo}\Big(\{{}_i y^{k,r}_{a:p} \mid a{:}p \in \text{ipn}(k)\}\Big)$$

The Constraints (7.1f), (7.1g) and (7.1h) that fix the channel labels in the input and the output stage remain the same.

We change the Constraints (7.1i) such that we quantify over the new arc variables, i.e. we quantify k over B instead of quantifying over two variables from S. These two variables now become $\text{stBef}(k)$ and $\text{stAft}(k)$.

$$\bigwedge_{\substack{w^{k,l}_{a:o,b:p}\in\mathcal{W}\\ r\in R\\ i\in C}} \left(\begin{array}{c} \left(w^{k,l}_{a:o,b:p} \rightarrow ({}_iy^{k,r}_{a:o} \leftrightarrow {}_iy^{l,r}_{b:p})\right) \\ \wedge \left(w^{k,l}_{a:o,b:p} \leftarrow ({}_iy^{k,r}_{a:o} \ \wedge \ {}_iy^{l,r}_{b:p} \wedge \bigwedge_{\substack{m\in L\\ k<m<l}} nc^{m,r}_i)\right) \end{array} \right) \tag{7.1i}$$

(7.7i)
$$\bigwedge_{\substack{w^{k}_{a:o,b:p}\in\mathcal{W}\\ r\in R\\ i\in C}} \left(\begin{array}{c} \left(w^{k}_{a:o,b:p} \rightarrow ({}_iy^{\text{stBef}(k),r}_{a:o} \leftrightarrow {}_iy^{\text{stAft}(k),r}_{b:p})\right) \\ \wedge \left(w^{k}_{a:o,b:p} \leftarrow ({}_iy^{\text{stBef}(k),r}_{a:o} \ \wedge \ {}_iy^{\text{stAft}(k),r}_{b:p})\right) \end{array} \right)$$

We change the Constraint (7.1j) to handle the case that a comparison node is split into two phantom node. Then the channel label of the phantom node's input port must match the channel label of the phantom node's output port. This also holds for the unused node.

$$\bigwedge_{\substack{r\in R\\ k\in L\\ i\in C\\ a\in G}} \left({}_iy^{k,r}_{a:\min} \vee {}_iy^{k,r}_{a:\max}\right) \leftrightarrow \left({}_iy^{k,r}_{a:\text{in1}} \vee {}_iy^{k,r}_{a:\text{in2}}\right) \tag{7.1j}$$

(7.7j)
$$\bigwedge_{\substack{r\in R\\ k\in L\\ i\in C}} \left(\begin{array}{c} \bigwedge_{a\in G} \left(\begin{array}{c} \left({}_iy^{k,r}_{a:\min} \vee {}_iy^{k,r}_{a:\max}\right) \leftrightarrow \left({}_iy^{k,r}_{a:\text{in1}} \vee {}_iy^{k,r}_{a:\text{in2}}\right) \\ \wedge \left(ng^k_a \rightarrow \left(\begin{array}{c} ({}_iy^{k,r}_{a:\min} \leftrightarrow {}_iy^{k,r}_{a:\text{in1}}) \\ \wedge ({}_iy^{k,r}_{a:\max} \leftrightarrow {}_iy^{k,r}_{a:\text{in2}}) \end{array} \right)\right) \end{array} \right) \\ \wedge \left({}_iy^{k,r}_{\text{unused:out}} \leftrightarrow {}_iy^{k,r}_{\text{unused:in}}\right) \end{array} \right)$$

Note that we do not add $\neg ng^k_a$ as a premiss to the first line. This is because the implication in the second line implies the first line. Hence we can omit $\neg ng^k_a$ as the premiss in the first line, because the implication holds even if $ng^k_a \mapsto \text{true}$.

We still want $\mathbb{R}$ and $\mathbb{I}$ to be standard. Thus the Constraint (7.1k) remains the same.

Before we had for a comparison node that the channel labels of its output ports corresponds to a gate in the line representation. We modify the Constraint (7.1l) such in the graph the comparison node must not be split into two phantom node.

$$\bigwedge_{\substack{g^{k,r}_{i,j}\in\mathcal{G}\\ a\in G}} \left(\begin{array}{c} \left(g^{k,r}_{i,j} \rightarrow ({}_iy^{k,r}_{a:\min} \leftrightarrow {}_jy^{k,r}_{a:\max})\right) \\ \wedge \left(g^{k,r}_{i,j} \leftarrow ({}_iy^{k,r}_{a:\min} \ \wedge \ {}_jy^{k,r}_{a:\max})\right) \end{array} \right) \tag{7.1l}$$

$$\bigwedge_{\substack{g_{i,j}^{k,r}\in\mathcal{G}\\ a\in G}} \left(\begin{array}{c} \left(g_{i,j}^{k,r} \rightarrow \left(\begin{array}{c} \left({}_iy_{a:\min}^{k,r} \leftrightarrow {}_jy_{a:\max}^{k,r} \right) \\ \wedge \left({}_iy_{a:\min}^{k,r} \leftrightarrow \neg ng_a^k \right) \\ \wedge \left({}_jy_{a:\max}^{k,r} \leftrightarrow \neg ng_a^k \right) \end{array} \right) \right) \\ \wedge \left(g_{i,j}^{k,r} \leftarrow ({}_iy_{a:\min}^{k,r} \ \wedge \ {}_jy_{a:\max}^{k,r} \wedge \neg ng_a^k) \right) \end{array} \right) \tag{7.7l}$$

Similarly we also ensure for the Constraints (7.1m) and (7.1n) that the comparison node in the graph is not split into two phantom nodes.

$$\bigwedge_{\min_{i,n-1}^{k,r}\in\mathcal{N}_{0,n-1}} \left(\min_{i,n-1}^{k,r} \leftrightarrow \bigvee_{a\in G} {}_iy_{a:\min}^{k,r} \right) \tag{7.1m}$$

$$\bigwedge_{\max_{0,i}^{k,r}\in\mathcal{X}_{0,n-1}} \left(\max_{0,i}^{k,r} \leftrightarrow \bigvee_{a\in G} {}_iy_{a:\max}^{k,r} \right) \tag{7.1n}$$

$$\bigwedge_{\min_{i,n-1}^{k,r}\in\mathcal{N}_{0,n-1}} \left(\begin{array}{c} \left(\min_{i,n-1}^{k,r} \rightarrow \bigvee_{a\in G} {}_iy_{a:\min}^{k,r} \right) \\ \wedge \bigwedge_{a\in G} \left(\begin{array}{c} \left(\min_{i,n-1}^{k,r} \rightarrow ({}_iy_{a:\min}^{k,r} \rightarrow \neg ng_a^k) \right) \\ \wedge \left(\min_{i,n-1}^{k,r} \leftarrow ({}_iy_{a:\min}^{k,r} \ \wedge \ \neg ng_a^k) \right) \end{array} \right) \end{array} \right) \tag{7.7m}$$

$$\bigwedge_{\max_{0,i}^{k,r}\in\mathcal{X}_{0,n-1}} \left(\begin{array}{c} \left(\max_{0,i}^{k,r} \rightarrow \bigvee_{a\in G} {}_iy_{a:\max}^{k,r} \right) \\ \wedge \bigwedge_{a\in G} \left(\begin{array}{c} \left(\max_{0,i}^{k,r} \rightarrow ({}_iy_{a:\max}^{k,r} \rightarrow \neg ng_a^k) \right) \\ \wedge \left(\max_{0,i}^{k,r} \leftarrow ({}_iy_{a:\max}^{k,r} \ \wedge \ \neg ng_a^k) \right) \end{array} \right) \end{array} \right) \tag{7.7n}$$

In contrast representing a variable nc_i^k in the graph means that the node in layer k whose port is labeled with channel i is a phantom node. That means it is either the ath comparison node with $ng_a^k \mapsto$ true or it is the unused node.

$$\bigwedge_{nc_i^{k,r}\in\mathcal{NC}} \left(nc_i^{k,r} \leftrightarrow \bigwedge_{a:p\in \mathrm{op}(k)} \neg\, {}_iy_{a:p}^{k,r} \right) \tag{7.1o}$$

$$\bigwedge_{nc_i^{k,r}\in\mathcal{NC}} \left(nc_i^{k,r} \leftrightarrow \bigwedge_{a:p\in \mathrm{ip}(k)} \neg\, {}_iy_{a:p}^{k,r} \right) \tag{7.1p}$$

$$\bigwedge_{nc_i^{k,r} \in \mathcal{NC}} \left(\bigwedge_{a:p \in \mathrm{op}(k)} \left(\begin{array}{c} \left(nc_i^{k,r} \rightarrow ({}_i y_{a:p}^{k,r} \rightarrow ng_a^k)\right) \\ \wedge \left(nc_i^{k,r} \leftarrow ({}_i y_{a:p}^{k,r} \wedge ng_a^k)\right) \end{array} \right) \wedge \left(nc_i^{k,r} \leftarrow {}_i y_{\text{unused:out}}^{k,r}\right) \right) \tag{7.7o}$$

$$\bigwedge_{nc_i^{k,r} \in \mathcal{NC}} \left(\bigwedge_{a:p \in \mathrm{ip}(k)} \left(\begin{array}{c} \left(nc_i^{k,r} \rightarrow ({}_i y_{a:p}^{k,r} \rightarrow ng_a^k)\right) \\ \wedge \left(nc_i^{k,r} \leftarrow ({}_i y_{a:p}^{k,r} \wedge ng_a^k)\right) \end{array} \right) \wedge \left(nc_i^{k,r} \leftarrow {}_i y_{\text{unused:in}}^{k,r}\right) \right) \tag{7.7p}$$

7.4 Breaking the Input Twist Symmetry

In this section we consider the symmetry of twisting inputs, which on standard sorting networks matches the symmetry of permuting and untangling. In Section 4.2.6 we choose a representative two-layer prefix from the class of two-layer prefixes that are symmetric under permuting and untangling. One may think that committing to a representative prefix breaks the symmetry by permuting and untangling on standard sorting networks. But this commitment only partially breaks the symmetry, i.e. there are symmetry classes where two or more networks have the same two-layer prefix.

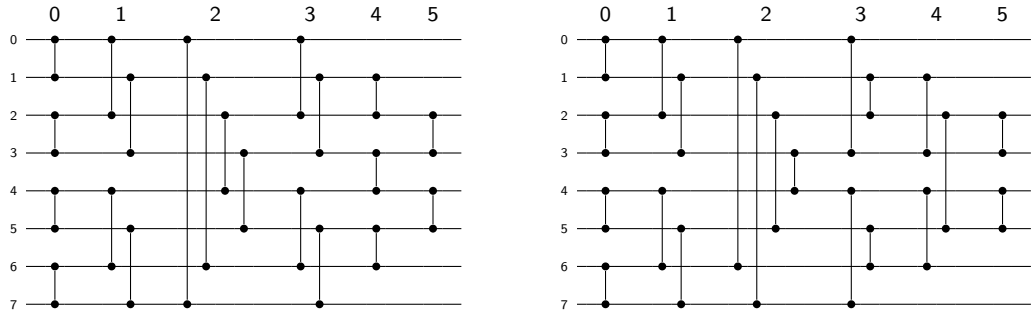
Ehlers writes wrt future work:

> When using prefixes on two layers, some symmetry breaks could be applied on the third layer of a sorting network. Consider the Green Filter on 8 channels and 2 layers in Figure 7.1, which consists of 2 segments, each of which is a Green Filter on 4 channels. Connecting the top-most channel to the bottom-most channelis symmetric to connecting it to the second channel from the bottom from, thus, one of these comparators could be forbidden. [11]

Actually Ehlers gives an example for $n = 12$, but I reduced it to a smaller example for $n = 8$. We see that Figure 7.1 shows two standard sorting networks N_1^8 and N_2^8 on 8 elements of minimal depth 6 that have the same two-layer prefix. Notably they are in the same symmetry class. For example permuting the channels of network N_1^8 by $\left(\begin{smallmatrix} 0&1&2&3&4&5&6&7 \\ 4&5&6&7&0&1&2&3 \end{smallmatrix}\right)$ and untangling the result yields network N_2^8. Alternatively we may reverse the channels with rev and untangle to transform one network into the other.

Note that additionally encoding another standard network with its inputs permuted, e.g. reversed, as outlined in Chapter 6 does not complete the symmetry break. Assuming that we commit to this two-layer prefix and encode $\mathbb{R}$ and $\mathbb{I}$ there are still two satisfying assignments. One assignment where $\mathbb{I} = \mathsf{N}_1^8$ and $\mathbb{R} = \mathsf{N}_2^8$ and another assignment where $\mathbb{R} = \mathsf{N}_1^8$ and $\mathbb{I} = \mathsf{N}_2^8$.

In general a two-layer prefix can be split into connected components. For example the two-layer prefix of the networks 7.1a and 7.1b in Figure 7.1 has two connected



(a) *Sorting network* N_1^8 (b) *Sorting network* N_2^8

Figure 7.1

components. The first four channels line-represent one connected component and the last four channels line-represent the other connected component. Clearly a standard two-layer prefix cannot be extended to two distinct standard sorting networks with isomorphic graphs if none of its connected components in the two-layer prefix are pairwise isomorphic. In contrast the $\frac{n}{4}$ connected components of a two-layer Green filter, for which n is a power of 2 greater than 8, are all pairwise isomorphic. Thus fixing the first two layers by a Green filter reduces the number of permutable components by a constant factor of 4.

Note that there is a symbolic representation of a symmetry class of two-layer prefixes, which is used to generate representative prefixes of all classes that we need to consider to prove a depth lower bound [45, 21]. In this symbolic representation a prefix class is represented by a multi-set of words where each word represents a connected component. Therefore we can identify the prefix classes, for which some connected components are pairwise isomorphic, by comparing the words of their symbolic representation.

Ehlers likely envisioned a symmetry break that fixes the truth value of either $g^2_{0,7}$ or $g^2_{0,6}$ to false. Such a symmetry breaking constraint basically adds no clauses, since the solver immediately assigns the truth value and marks the constraint as satisfied. We do not devise such a symmetry breaking constraint in this section. Instead we devise symmetry breaking constraints, which are much more costly in terms of the number of clauses added, in order to break various symmetries of the graph encoding. In particular we first point out a relation between these graph symmetries and the symmetry by twisting inputs. Then we argue that completely breaking the graph symmetries also breaks the symmetry of twisting inputs such that the symmetry of permuting and untangling on standard sorting networks is also completely broken.

7.4.1 Symmetries of the Graph Encoding

In Section 2.3 we defined the π-line representation as the channel network determined by a given graph network with a π-channel labeling and a partition into layers. Further we

may interpret permuting and untangling or twisting inputs as a transformation of the π-channel labeling. Consider the assignment where $\mathbb{I} = \mathsf{N}_1^8$ and $\mathbb{R} = \mathsf{N}_2^8$. Permuting N_1^8 by e.g. rev and untangling to obtain N_2^8 transforms the id-channel labeling that determines $\mathbb{I}$ into a rev-channel labeling that determines $\mathbb{I}$. This violates the Constraint (7.1g). This raises the question: How can the other assignment with $\mathbb{I} = \mathsf{N}_2^8$ and $\mathbb{R} = \mathsf{N}_1^8$ satisfy all constraints including Constraint (7.1g)?

To answer that we consider the rev-graph representation of N_1^8. Permuting N_1^8 by rev and untangling to obtain N_2^8 transforms the rev-channel labeling into an id-channel labeling such that we can use this other graph with this id-channel labeling for a satisfying assignment where $\mathbb{I} = \mathsf{N}_2^8$. The problem is that

We observe that while there is a single graph for both $\mathbb{R}$ and $\mathbb{I}$ in each satisfying assignment, the graphs in the two satisfying assignments are different.

We formalize this observation. For simplicity we assume that we only encode $\mathbb{I}$. Specifically we do not encode $\mathbb{R}$ and its relation to $\mathbb{I}$, but we encode the the channel network $\mathbb{I}$ as well as a graph G with an id-channel labeling such that $\mathbb{I}$ is the id-line representation of G.

Let a be a satisfying assignment, i.e. it encodes a standard channel network $\mathbb{I}$. Further let us assume there are variables $g_{i,j}^k$, $\min_{i,j}^k$ and $\max_{i,j}^k$ with $i \geq j$. We assume that there is an additional constraint that fixes their truth values to false such that it corresponds to our usual encoding. Then there are two kinds of variable permutations that determine another assignment b, which satisfies every constraint except the Constraint (7.1g).

- The first kind interprets the aforementioned transformation on the channel labeling, which itself is an interpretation of permuting and untangling or twisting inputs, as a permutation of the Boolean variables in the encoding. Specifically let $\pi_k : C \to C$ be the permutation of the channels labels between each layer for every k from B. Then this determines a variable permutation σ:

$$\begin{aligned}
\sigma(g_{i,j}^{k,r}) &= g_{\pi_{\mathrm{aft}(k)}(i),\pi_{\mathrm{aft}(k)}(j)}^{k,r} \\
\sigma(w_{a:p,b:q}^k) &= w_{a:p,b:q}^k \\
\sigma({}_i y_{a:p}^{k,r}) &= \begin{cases} {}_{\pi_{\mathrm{bef}(k)}(i)} y_{a:p}^{k,r} & a{:}p \in \mathrm{ipn}(k) \\ {}_{\pi_{\mathrm{aft}(k)}(i)} y_{a:p}^{k,r} & a{:}p \in \mathrm{opn}(k) \end{cases} \\
\sigma(nc_i^{k,r}) &= nc_{\pi_{\mathrm{aft}(k)}(i)}^{k,r} \\
\sigma(ng_a^k) &= ng_a^k \\
\sigma(v_{i,x}^{k,r}) &= v_{\pi_k(i),x}^{k,r}
\end{aligned} \tag{7.8}$$

  This variable permutation σ determines the assignment b where for any variable v holds that b maps $\sigma(v)$ to true iff a maps v to true. The truth values of the variables $\mathcal{N}$ and $\mathcal{X}$ under b are determined by the Constraints (5.4), (5.5) and (5.6), as there

is only one mapping to truth values that satisfies these constraints. The Constraint (7.1g) is violated because the channel labels of the ports in the stage are permuted by π_0. Note that if we permute the channels by π and untangle, then we have $\pi_0 = \pi$.

- The second kind of variable permutation simply reconnects the network input nodes according to a permutation $\pi : C -> C$.

$$\begin{aligned}
\sigma(g_{i,j}^{k,r}) &= g_{i,j}^{k,r} \\
\sigma(\min_{i,j}^{k,r}) &= \min_{i,j}^{k,r} \\
\sigma(\max_{i,j}^{k,r}) &= \max_{i,j}^{k,r} \\
\sigma(w_{a:p,b:q}^{k}) &= \begin{cases} w_{\pi(a):p,b:q}^{k} & k = 0 \\ w_{a:p,b:q}^{k} & \text{otherwise} \end{cases} \\
\sigma({}_{i}y_{a:p}^{k,r}) &= \begin{cases} {}_{\pi(i)}y_{a:p}^{k,r} & k = \text{in} \\ {}_{i}y_{a:p}^{k,r} & \text{otherwise} \end{cases} \\
\sigma(nc_{i}^{k,r}) &= nc_{i}^{k,r} \\
\sigma(ng_{a}^{k}) &= ng_{a}^{k} \\
\sigma(v_{i,x}^{k,r}) &= v_{i,x}^{k,r}
\end{aligned} \tag{7.9}$$

Again this variable permutation σ determines the assignment b where for any variable v holds that b maps $\sigma(v)$ to true iff a maps v to true. The Constraint (7.1g) is violated because the channel labels of the ports in the stage are permuted by π.

Both variable permutations permute the channel labels in the input stage violating. Our observation is that we can combine these two kinds of variable permutations such that their permutations of the channel labels in the input stage cancel each other out. This way we can satisfy the Constraint (7.1g).

If we permute the channels of a sorting network given by assignment a by π and untangle, then the following variable permutation determines another assignment b that like a

satisfies all constraints in the encoding:

$$\begin{aligned}
\sigma(g_{i,j}^{k,r}) &= g_{\pi_{\mathrm{aft}(k)}(i),\pi_{\mathrm{aft}(k)}(j)}^{k,r} \\
\sigma(w_{a:p,b:q}^{k}) &= \begin{cases} w_{\pi_0{}^{-1}(a):p,b:q}^{k} & k = 0 \\ w_{a:p,b:q}^{k} & \text{otherwise} \end{cases} \\
\sigma({}_i y_{a:p}^{k,r}) &= \begin{cases} \pi_{\mathrm{aft}(k)}(i) y_{a:p}^{k,r} & a{:}p \in \mathrm{opn}(k) \\ {}_i y_{a:p}^{k,r} & k = \mathrm{in} \\ \pi_{\mathrm{bef}(k)}(i) y_{a:p}^{k,r} & \text{otherwise} \end{cases} \\
\sigma(nc_i^{k,r}) &= nc_{\pi_{\mathrm{aft}(k)}(i)}^{k,r} \\
\sigma(ng_a^k) &= ng_a^k \\
\sigma(v_{i,x}^{k,r}) &= v_{\pi_k(i),x}^{k,r}
\end{aligned} \tag{7.10}$$

Recall from Section 3.1.2 that the graphs of N_1^8 and N_2^8 are isomorphic. Reconnecting the network input nodes retains this property, because we can change the first component of the network input nodes, which maps between network input nodes.

Note that we can we can similarly reconnect the network output nodes. We can likewise cancel such a reconnection with a series of output twists instead of input twists/permuting and untangling. However this does not yield a proper sorting network but a sorting network under permuted outputs, which violates the encoding. We exploit this in the next section, but as a consequence our symmetry break is only complete on symmetry classes of satisfying assignments.

Notably there is a strict notion of symmetry in the context of (Boolean) SAT solving, which defines a symmetry as a permutation of variable value pairs [19, 51]. The variable permutations above are not SAT symmetries by this strict notion, because we defined them to only apply to satisfying assignments and because we did not define the permutation of min max variables. In all of Section 7.4 we are more liberal with what we call a symmetry.

Further we also weaken the term complete symmetry breaking constraint to mean that such a constraint is satisfied by at most one assignment from each symmetry class of assignments that all satisfy the set of constraints that the symmetry breaking constraint is added to. In contrast the at most one condition of a proper complete symmetry breaking constraint applies to all symmetry classes.

We conclude that the two graphs from the assignments with $\mathbb{I} = \mathsf{N}_1^8$ and $\mathbb{R} = \mathsf{N}_2^8$ or alternatively with $\mathbb{R} = \mathsf{N}_1^8$ and $\mathbb{I} = \mathsf{N}_2^8$ only differ in their connection to the network input nodes.

Note that the two graphs only differ in their connection to the network input nodes if they are defined by Definition 2 where the nodes are partitioned into layers in a way that is uniqely determined (see Section 4.2.4). Recall that layers are sets of comparison

nodes. But our encoding numbers the comparison nodes in each layer. Hence a layer in our encoding is a sequence of comparison nodes. As a consequence the two graphs may also differ in that the comparison nodes are reordered within a layer.

For any family of permutations $\pi_k : G \to G$ that specifies a permutation of the comparison nodes in each layer k there is the variable permutation

$$\begin{aligned}
\sigma(g^{k,r}_{i,j}) &= g^{k,r}_{i,j} \\
\sigma(w^k_{a:p,b:q}) &= \begin{cases} w^k_{a:p,\pi_{\mathrm{stAft}(k)}(b):q} & k = 0 \\ w^k_{\pi_{\mathrm{stBef}(k)}(a):p,b:q} & k = d \\ w^k_{\pi_{\mathrm{stBef}(k)}(a):p,\pi_{\mathrm{stAft}(k)}(b):q} & \text{otherwise} \end{cases} \\
\sigma({}_i y^{k,r}_{a:p}) &= \begin{cases} {}_i y^{k,r}_{\pi_k(a):p} & k \in L \text{ and } a \neq \text{unused} \\ {}_i y^{k,r}_{a:p} & \text{otherwise} \end{cases} \\
\sigma(nc^{k,r}_i) &= nc^{k,r}_i \\
\sigma(ng^k_a) &= ng^k_{\pi_k(a)} \\
\sigma(v^{k,r}_{i,x}) &= v^{k,r}_{i,x}
\end{aligned} \tag{7.11}$$

Again for any assignment a this variable permutation σ determines another assignment b where for any variable v holds that b maps $\sigma(v)$ to true iff a maps v to true. The truth values of the variables $\mathcal{N}$ and $\mathcal{X}$ under b are determined by the Constraints (5.4), (5.5) and (5.6), as there is only one mapping to truth values that satisfies these constraints. Then a satisfies all constraints iff b satisfies all constraints.

To be very exact the graph encoding from Section 7.3 admits two kinds of node reorderings within a layer. The first kind reorders comparison nodes and the second kind reorders phantom nodes. In a faithful graph encoding there is only the first kind, because such an encoding has no phantom nodes.

Notably we can treat the ath comparison node in layer k with $ng^k_a \mapsto$ true as either a single comparison node or two phantom nodes. This means that we may reorder the phantom node at the unused position to any other position in the layer if at least one comparison node in the layer is split into two phantom nodes. Because that allows us to swap the unused phantom node with another phantom node. This phantom node is the result of splitting a comparison nodes a into two phantom nodes. We can also reorder a with other comparison nodes.

Finally any ath comparison node in some layer k, denoted as e, with $ng^k_a \mapsto$ true may reconnect the arcs to its input ports such that the arcs $(b{:}p, e\,\mathrm{in1})$ and $(c{:}q, e{:}\,\mathrm{in2})$ become $(b{:}p, e\,\mathrm{in2})$ and $(c{:}q, e{:}\,\mathrm{in1})$. Again for a given satisfying assignment a there are corresponding variable permutations that determine a satisfying assignment b, which encodes the same graph of a but with the arcs to the input ports of some comparison node(s) reconnected. Note that the graph encoding variant from Section 7.2.1 does not have this symmetry, because it does not distinguish between input ports.

We conclude that our graph encoding has three symmetries.

- The first symmetry twists inputs while at the same time it reconnects the network input nodes.
- The second symmetry reorders comparison nodes within a layer.
- The third symmetry reorders phantom nodes within a layer.
- The fourth symmetry reconnects the arcs to input ports of comparison nodes.

7.4.2 Breaking the Graph Symmetries

We consider the symmetry of twisting inputs of a standard network $\mathbb{I}$ or equivalently the symmetry of twisting inputs on the channel labeling of a graph that determines the standard network $\mathbb{I}$. This symmetry divides the satisfying assignments into symmetry classes. In this section we devise a symmetry breaking constraint φ such that only one representative assignment of each class satisfies φ.

In theory to break the symmetry of twisting inputs we only need to break the symmetry that reconnects the network input nodes while twisting the inputs on channel labeling. Unfortunately fixing the arcs from the network input nodes in the input stage to the comparison and phantom nodes in the first layer does not fix the way to connect the network input nodes. The argument for that is similar to before. For a satisfying assignment a both the Variable Permutations (7.10) and (7.11) determine another assignment b that satisfies every other Constraint except the fixed arcs. We may combine these variable permutations such that they cancel each other and determine another assignment with the same fixed arcs as a.

Likewise the symmetries of reordering nodes in adjacent stages may cancel each other wrt fixed arcs between those stages. In general we cannot relate e.g. the ath comparison node in layer k in one satisfying assignment to the ath comparison node in layer k in another satisfying assignment, because the comparison nodes that they connect to in the next and previous layer may be unrelated due to reordering.

Thus in order to break the symmetry by twisting inputs we also break the graph symmetries of reordering comparison and phantom nodes within layers. Additionally we give a symmetry breaking constraint for reconnecting the arcs to the input ports of a comparison node, because these constraints happen to be very cheap. In summary we break all graph symmetries from the previous section.

Fortunately as mentioned before reconnecting the nodes from the output stage transforms a sorting network into a network that only sorts under permuted outputs. Because such networks do not satisfy our encoding, there is no symmetry of reconnecting the network output nodes that cancels the symmetry of reordering the nodes within the last layer wrt to the arcs between the last layer and the output stage.

Thus we start with the perfect matching between the nodes in the last layer and the network output nodes. Reordering the nodes in the last layer divides the perfect matching into symmetry classes. The idea is to add a symmetry breaking constraint φ such that exactly one representative matching satisfies φ. This breaks the symmetry of reordering nodes in the last layer. With that order fixed the symmetry of reordering nodes in the second to last layer divides the perfect matchings between the second to last and the last layer into symmetry classes. This way we repeatedly break the symmetries for every matching between two stages where the symmetry classes of the matchings between the input stage and the first layer are determined by the symmetry of reconnecting inputs.

To make this work we need to be careful with our choice of symmetry breaking criteria, i.e. the criteria that determine the representative matching of each symmetry class. Clearly we need symmetry breaking criteria for each symmetry in each layer. The problem is that symmetry breaking criteria are the truth values of other variables in the encoding. In general the choice of a representative in one symmetry class X may affect the symmetry breaking criteria for another symmetry class, whose representative may in turn affect the symmetry breaking criteria for class X. Such cyclic interference may cause the conjunction of two symmetry breaking constraints, which are correct and complete on their own, to be incorrect or incomplete.

To avoid such cyclic interference we choose our symmetry breaking criteria for a symmetry in some layer such that they are determined by the partial assignment of all variables succeeding that layer. In particular breaking the symmetry of reordering nodes in the last layer determines the symmetry breaking criteria for the symmetry of reordering nodes in the second to last layer and so on. This way we only need to make sure that the breaks of different symmetries within the same layer do not interfere with each other.

7.4.3 Non-Solutions

For simplicity let us assume that n is even and every channel is used by a gate in every layer. In other words we assume that there are no phantom nodes.

A first attempt is the following. First we break the symmetry of reconnecting the arcs to the input ports of the ath comparison node in layer k we consider the two candidate assignments

$$w^{\mathrm{bef}(k)}_{b:p,a:\mathrm{in1}} \mapsto \mathrm{true}$$
$$w^{\mathrm{bef}(k)}_{c:q,a:\mathrm{in2}} \mapsto \mathrm{true}$$

or

$$w^{\mathrm{bef}(k)}_{c:q,a:\mathrm{in1}} \mapsto \mathrm{true}$$
$$w^{\mathrm{bef}(k)}_{b:p,a:\mathrm{in2}} \mapsto \mathrm{true}$$

Without loss of generality let $b < c$ where $\lfloor \frac{n}{2} \rfloor - 1 <$ unused. Then we always decide for the former candidate. This could be realized with a symmetry breaking constraint that disallows the latter candidate.

$$\bigwedge_{\substack{k \in B \setminus \{d\} \\ a \in \mathrm{ns}(\mathrm{stAft}(k)) \\ b:p \in \mathrm{opnBef}(k) \\ c:q \in \mathrm{opnBef}(k) \\ b < c}} \left(\neg w^k_{c:q,a:\mathrm{in1}} \vee \neg w^k_{b:p,a:\mathrm{in2}} \right) \tag{7.12}$$

Next we break the symmetry of reordering comparison nodes in layer k by choosing a representative matching for $\mathrm{aft}(k)$. We choose symmetry breaking criteria that order the comparison nodes in layer k. To order the nodes we consider their properties that are invariant under each symmetry class. Namely in all candidate matchings from the same symmetry class a comparison node's min port is always connected to the same node in the next stage. The same holds for a comparison node's max port. For a comparison node c we denote the positions of these nodes in the next stage by $\mathrm{toNodePos}(c{:}\mathrm{min})$ and $\mathrm{toNodePos}(x{:}\mathrm{max})$.

We consider two orders on the comparison nodes. We may order them such that a node c is smaller than another comparison node e

1. if it holds that $\mathrm{toNodePos}(c{:}\mathrm{min}) < \mathrm{toNodePos}(e{:}\mathrm{min})$
2. if it holds that $\mathrm{toNodePos}(c{:}\mathrm{min}) < \mathrm{toNodePos}(e{:}\mathrm{min})$ or alternatively that we have that $\mathrm{toNodePos}(c{:}\mathrm{min}) = \mathrm{toNodePos}(e{:}\mathrm{min})$ and an order by the max port, i.e. that $\mathrm{toNodePos}(c{:}\mathrm{max}) < \mathrm{toNodePos}(e{:}\mathrm{max})$.

Hence for two comparison nodes at positions a and b with $a < b$ in k we want the node at position a to be smaller than the node at position b. Thus for Order 1 we disallow the case that the min port of the ath comparison node connects to a greater node position in the next stage than the min port of the bth node.

$$\bigwedge_{\substack{k \in B \setminus \{0\} \\ a,b \in \mathrm{ns}(\mathrm{stBef}(k)) \\ a < b \\ e:p \in \mathrm{ipnAft}(k) \\ f:q \in \mathrm{ipnAft}(k) \\ e > f}} \left(\neg w^k_{a:\mathrm{min},e:p} \vee \neg w^k_{b:\mathrm{min},f:q} \right) \tag{7.13}$$

The idea is that there is exactly only one matching that satisfies the order, which is the representative matching.

We also have to check that the composition of the Symmetry Breaking Constraints (7.12) and (7.13) is correct and complete. In my bachelor thesis I called two correct and complete symmetry breaking constraints whose composition is correct and complete *compatible*. Let T and U be equivalence relations (symmetries) on the assignments of a formula and let φ and ψ be symmetry breaking constraints for that formula that break the symmetry T and U respectively. In my bachelor thesis I show that φ and ψ are

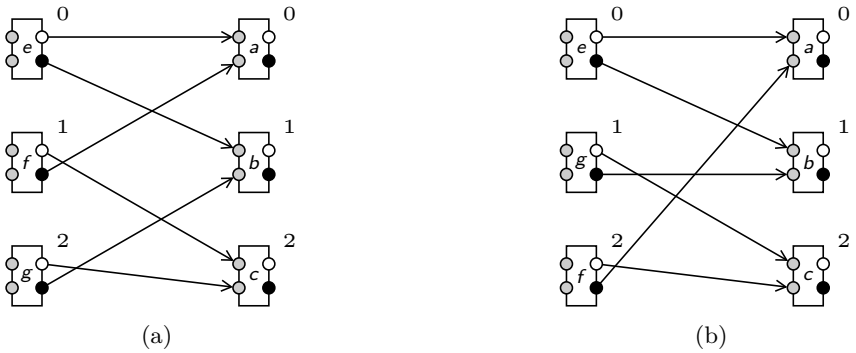


Figure 7.2: *Two matchings symmetric under swapping* f *and* g

compatible if for any two assignments x and y that are symmetric under T and U their representative assignments under φ are also symmetric under U [19]. In other words φ and ψ are compatible if one of them preserves the other symmetry.

Note that under the assumption that the Symmetry Breaking Constraints (7.12) and (7.13) are correct and complete on their own they are compatible, because the Constraint (7.13) preserves the symmetry of reconnecting the arcs to the input ports of each comparison node in aft(k).

Figure 7.2 shows two matchings from the same symmetry class of reordering nodes in the earlier layer and reconnecting the two arcs to the input ports of comparison nodes in the later layer. Specifically the nodes f and g are swapped. In Matching 7.2a the node f is at position 1 and g is at position 2, whereas in Matching 7.2b the node f is at position 2 and g is at position 1.

Since both matchings satisfy the Constraint (7.12) as well as the constraint for Order 1, the Symmetry Breaking Constraint (7.13) is incomplete. Hence we may use Order 2 instead whose representative also preserves the symmetry of reconnecting the arcs to the input ports of each comparison node. This disallows the Matching 7.2b.

Alternatively we may replace the Symmetry Breaking Constraint (7.12) with a constraint that chooses a different representative from each symmetry class of reconnecting the arcs to the input ports of a comparison node in stAft(k) such that Order 1 suffices to break the symmetry of reordering nodes in stBef(k). Specifically in any matching that is symmetric under reordering the nodes in stBef(k) the following property of every comparison node c in stAft(k) is invariant.

We consider an arc that connects to the input port in1 of c. That arc is attached to the output port of a comparison node e in stBef(k) and there is another arc from the other output port of e to a comparison node in stAft(k), which we denote by viaBef(c:min) because it is connected to c:min via some node in stBef(k). Likewise there is a node viaBef(c:max) in stAft(k) that is connected to c:max via some node in stBef(k). Note that they are only connected if we ignore the direction of the arcs. We observe that for

every node c in stAft(k) the position of the nodes viaBef(c:min) and viaBef(c:max) in stAft(k) is invariant under reordering the nodes in stBef(k).

Clearly reconnecting the arcs to the input nodes of a comparison node c swaps the nodes viaBef(c:min) and viaBef(c:max) and hence also swaps their positions in stAft(k), which we denote by viaBefPos(c:min) and viaBefPos(c:max). Thus we may break the symmetry of reconnecting the arcs to the input ports of a comparison node c in stAft(k) by choosing the representative matching where viaBefPos(c:min) $\leq$ viaBefPos(c:max). Let us assume that we do this with a symmetry breaking constraint φ. Note that this choice of representative by φ preserves the symmetry of reordering nodes in stBef(k), which is a sufficient condition for compatibility of the symmetry breaking constraints φ and Constraint (7.13) under the assumption that both constraints are correct and complete on their own.

Using φ we may consider a variant of Order 1 that does not compare the position of the nodes but the position of the ports in stAft(k). Let c be a comparison node in stBef(k). We define toPortPos(c:min) = (toNodePos(c:p), q) where q is either in1 or in2 depending on which input port of the node at toNodePos(c:p) in stAft(k) the port p of node c in stBef(k) is connected to. We compare a tuple of a node position and an input port first by their node position and if those are equal by their input port position where in1 $<$ in2. We obtain a variant of Order 1 that use toPortPos instead of toNodePos.

Breaking the symmetry of reconnecting the arcs to the input ports of each comparison node with the viaBefPos criteria determines the toPortPos values for every output port in stBef(k). In this context the variant of Order 1 that uses toPortPos suffices such that we do not need the finer Order 2.

As a consequence there are two alternative symmetry breaks:

- A symmetry break $\mathfrak{B}_1$ where
  - breaking the symmetry of reordering nodes in stBef(k) preserves the symmetry of reconnecting the arcs to the input ports of a comparison node in stAft(k)
  - the symmetry of reconnecting the arcs to the input ports of a comparison node in stAft(k) is broken by Constraint (7.13).
  - the symmetry of reordering nodes in stBef(k) is broken by Order 2 using toNodePos
- A symmetry break $\mathfrak{B}_2$ where
  - breaking the symmetry of reconnecting the arcs to the input ports of a comparison node in stAft(k) preserves the symmetry of reordering nodes in stBef(k)
  - the symmetry of reconnecting the arcs to the input ports of a comparison node in stAft(k) is broken by the viaBefPos criteria.

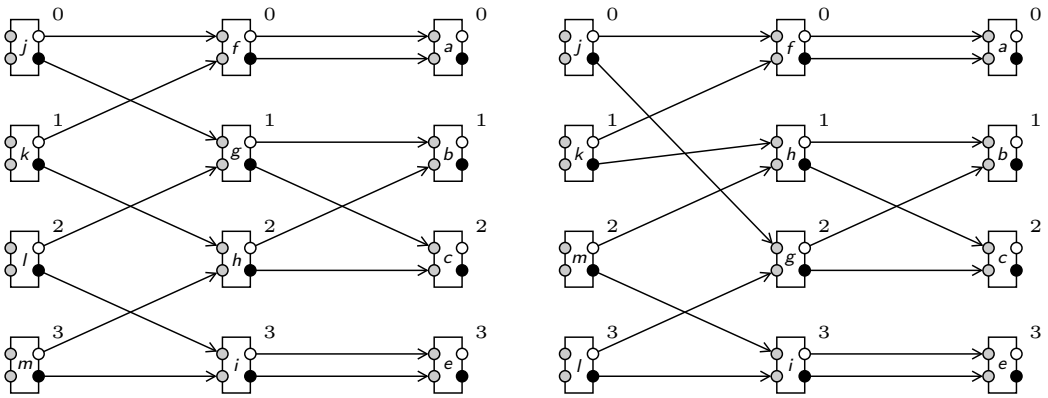


Figure 7.3: *Two matchings symmetric under swapping* g *and* h

– the symmetry of reordering nodes in stBef(k) is broken by Order 1 using toPortPos

Note that we have not shown either of the four symmetry breaks to be complete, which is a sufficient condition for the two alternative composed symmetry breaks $\mathfrak{B}_1$ or $\mathfrak{B}_2$ to be compatible. In particular there is a case that none of the aforementioned symmetry can handle: They do not break the case that toNodePos(c:min) = toNodePos(e:min) and toNodePos(c:max) = toNodePos(e:max). This case is particularly tricky, because we cannot break the symmetry of reordering c and e by swapping them, since both orders are represented by the same matching. As a consequence we must consider that c and e in in layer k may be swapped when choosing the representative matching between layer k and the stage before k. This is a problem, because we assume that the order of the nodes is fixed when when we choose the representative matching between layer k and the stage before k, because we use the node positions in k as the symmetry breaking criteria.

Figure 7.3 shows an example for $\mathfrak{B}_1$ where this case is problematic. It shows two assignments symmetric under reordering nodes $\{j, k, l, m\}$ and $\{f, g, h, i\}$ as well as reconnecting the two arcs to the input ports of a comparison node from $\{a, b, c, e, f, g, h, i\}$. In particular both assignments satisfy the Constraint (7.12) as well as the constraint for Order 2. Specifically the earlier matching in both 7.3a and 7.3b is the representative matching under these constraints assuming that the order of the nodes f, g, h and i is determined by representative matchings from symmetry breaks in the later layers. This assumption is violated, because we have toNodePos(g:min) = toNodePos(h:min) and toNodePos(g:max) = toNodePos(h:max) such that the representative matching between $\{f, g, h, i\}$ and $\{a, b, c, e\}$ admits a swap of g and h. Note that due to violated assumptions even the symmetry break of the matching between $\{j, k, l, m\}$ and $\{f, g, h, i\}$ is incomplete, as both earlier matchings in 7.3a and 7.3b satisfy the constraints. As a consequence this also admits a swap of l and m.

7.4.4 Strengthening the Symmetry Break

I propose to encode another id-channel labeling that determines a generalized id-line representation $\mathbb{G}$ in order to break the graph symmetries. This fixes the channel labels in the output stage by Definitions 5 and 3.

The idea is that when we break a matching of k in B, we use the channel labels of the input ports in $\mathrm{stAft}(k)$ as symmetry breaking criteria to determine the representative matching as well as the input ports in $\mathrm{stBef}(k)$. Previously we divided matchings of k into classes of matchings that are symmetric under reordering nodes in $\mathrm{stBef}(k)$. Now we divide matchings with channel labels where the input ports in $\mathrm{stAft}(k)$ are labeled the same into classes that are symmetric under reordering the nodes in $\mathrm{stBef}(k)$. In other words we strengthen the symmetry break by extending it from unlabeled matching to labeled matchings such that the induction hypothesis in our induction over each k in B allows us to assume that the input ports in $\mathrm{stAft}(k)$ are labeled the same in each class of (labeled) matchings that are symmetric under reordering nodes $\mathrm{stBef}(k)$.

We observe that any satisfying assignment encodes a sorting network graph. Additionally the channel labels for $\mathbb{G}$ in the output stage are predetermined. Thus all satisfying assignments that encode channel labelings for $\mathbb{G}$ are related by twisting inputs of gates in $\mathbb{G}$. As a consequence it suffices to additionally break the symmetry of twisting inputs in the channel labeling for $\mathbb{G}$.

I do not know how to break the symmetry of twisting inputs on $\mathbb{G}$ such that the representative assignment for $\mathbb{G}$ is standard. However it is straightforward if we allow the channel labeling for $\mathbb{G}$ to be a generalized channel labeling. This is the reason why we cannot reuse the channel labels that determine $\mathbb{I}$ or $\mathbb{R}$ but have to introduce another channel labeling that determines a generalized network $\mathbb{G}$, because we want $\mathbb{I}$ and $\mathbb{R}$ to be standard to utilize the subnetwork optimization.

Extending the symmetry break to include a channel labeling allows us to handle the aforementioned case where we have both $\mathrm{toNodePos}(c{:}\min) = \mathrm{toNodePos}(e{:}\min)$ and $\mathrm{toNodePos}(c{:}\max) = \mathrm{toNodePos}(e{:}\max)$ for two comparison nodes c and e. Because while reordering c and e by swapping them does not change the matching of k in B, it changes the channel labeling of the output ports in $\mathrm{stBef}(k)$. Now our symmetry break only admits a single representative assignment with certain channel labels of the input ports in $\mathrm{stBef}(k)$. The idea is that this way at most one of the two orders of c and e is admitted.

We consider the symmetry class of reordering nodes in $\mathrm{stAft}(k)$. By our induction hypothesis we assume that the break of this symmetry fixes the channel labels of the input ports in $\mathrm{stAft}(k)$ to a representative channel labeling. The matching between $\mathrm{stAft}(k)$ and $\mathrm{stBef}(k)$ might only determine a partial order of the nodes in $\mathrm{stBef}(k)$, since for two comparison nodes c and e in $\mathrm{stBef}(k)$ we may have $\mathrm{toNodePos}(c{:}\min) = \mathrm{toNodePos}(e{:}\min)$ and $\mathrm{toNodePos}(c{:}\max) = \mathrm{toNodePos}(e{:}\max)$. In contrast if for example we assume that the channel labels of the input ports in $\mathrm{stAft}(k)$ determine the same input port labels for

the same node (not node position) for every member of the same symmetry class under reordering nodes in stBef(k), then the channel labels of the output ports in stBef(k) determine a total order on the nodes in stBef(k).

As the order of of the comparison nodes in stBef(k) is only totally defined by the channel labels, it is crucial that we do not choose the node positions, whose order may be partial, as symmetry breaking criteria for the representative matching between stBef(k) and the stage k' before stBef(k) and the channel labels of the ports in k'. Instead we pick the channel labels in stBef(k) as the symmetry breaking criteria that determine this representative.

There are no comparison nodes in the input stage such that we never have two comparison nodes c and e in the input stage with toNodePos(c:min) = toNodePos(e:min) and toNodePos(c:max) = toNodePos(e:max). Likewise the matching between the input stage and the first layer must depend on the channel labels of the input ports in the first layer.

Instead of a node's position we consider a node's channel label as symmetry breaking criteria, which for comparison nodes we define to be the channel label of port in1. We define toNodeLabel(c:p) and viaBefLabel(c:p) to return the the channel label of port in1 of the node at position toNodePos(c:p) and viaBefPos(c:p) respectively. We define toPortLabel(c:p) to be the channel label of the port at toPortPos(c:p). We further strengthen the symmetry break by stating that the following invariant holds for the representative assignment that encodes $chan_{\mathbb{G}}$. We have $chan_{\mathbb{G}}(c{:}\text{in1}) < chan_{\mathbb{G}}(c{:}\text{in2})$ for any comparison node c.

7.4.5 Encoding of the Symmetry Break

The encoding of the symmetry break may either added to an encoding of with $R = \{\mathbb{I}, \mathbb{R}\}$ or alternatively it may be added to an encoding with $R = \{\mathbb{I}\}$.

We add variables for the generalized channel labeling that determines a generalized π-line representation $\mathbb{G}$ for some π. We only encode the channel labeling on the encoded graph that determines $\mathbb{G}$, but we neither encode the channel network $\mathbb{G}$ itself nor any network runs of $\mathbb{G}$. Again we label the ports instead of the arcs. We extend the set $\mathcal{Y}$ of port channel label variables as follows:

$$\mathcal{Y} = \{{}_i y^{k,r}_{a:p} \mid r \in R \cup \{\mathbb{G}\}, k \in S, a{:}p \in \text{portsn}(k), i \in C\}$$

Next we add constraints to ensure that the generalized channel labeling for $\mathbb{G}$ is a proper π-channel labeling for some π. We replace R with the set $R \cup \{\mathbb{G}\}$ in the Constraints (7.7c), (7.7d), (7.7e), (7.1f), (7.7i), (7.7j). Notably we do not replace R in the Constraint (7.1k), because $\mathbb{G}$ is a generalized network.

Next we break the symmetry of reconnecting the arcs to the input ports as well as the symmetry of reordering nodes within layers. We need to care that the symmetry

breaking constraints for these two symmetries do not interfere with each other. Obvious options are to

- break the symmetry of reordering nodes in stBef(k) preserving the symmetry of reconnecting the arcs to the input ports of a comparison node in stAft(k) similar to $\mathfrak{B}_1$
- break the symmetry of reconnecting the arcs to the input ports of a comparison node in stAft(k) preserving the symmetry of reordering nodes in stBef(k) similar to $\mathfrak{B}_2$

Breaking like $\mathfrak{B}_1$

We consider the first option that is similar to $\mathfrak{B}^1$.

The Symmetry Breaking Constraint (7.12) disallows

$$w^{\mathrm{bef}(k)}_{c:q,a:\mathrm{in1}} \mapsto \mathrm{true}$$
$$w^{\mathrm{bef}(k)}_{b:p,a:\mathrm{in2}} \mapsto \mathrm{true}$$

if $b < c$. We disallow this if the channel label of b is smaller than the channel label of c. Hence we add the symmetry breaking constraint

$$\bigwedge_{\substack{k\in B\setminus\{d\}\\ a\in\mathrm{ns}(\mathrm{stAft}(k))\\ b:p\in\mathrm{opnBef}(k)\\ c:q\in\mathrm{opnBef}(k)\\ i,j\in C\\ i<j}} \left(\left({}_j y^{\mathrm{bef}(k),\mathbb{G}}_{c:\mathrm{in1}} \wedge {}_i y^{\mathrm{bef}(k),\mathbb{G}}_{b:\mathrm{in1}}\right) \to \left(\neg w^k_{c:q,a:\mathrm{in1}} \vee \neg w^k_{b:p,a:\mathrm{in2}}\right)\right) \tag{7.14}$$

This breaks the symmetry by reconnecting the arcs to the inputs of the ath comparison node in stAft(k).

Next we break the symmetry by reordering nodes in stBef(k). For simplicity let us assume that n is even and every channel is used by a gate in every layer. In other words we assume that there are no phantom nodes. Then we add a constraint that is similar to Constraint (7.13) except that we do not compare $e > f$ but their channel labels i and j.

$$\bigwedge_{\substack{k\in B\setminus\{0\}\\ a,b\in\mathrm{ns}(\mathrm{stBef}(k))\\ a<b\\ e:p\in\mathrm{ipnAft}(k)\\ f:q\in\mathrm{ipnAft}(k)\\ j>i}} \left(\left({}_j y^{\mathrm{bef}(k),\mathbb{G}}_{e:\mathrm{in1}} \wedge {}_i y^{\mathrm{bef}(k),\mathbb{G}}_{f:\mathrm{in1}}\right) \to \left(\neg w^k_{a:\min,e:p} \vee \neg w^k_{b:\min,f:q}\right)\right) \tag{7.15}$$

For Order 2 we would also compare toNodeLabel(e:max) with toNodeLabel(e:max) if toNodeLabel(e:min) = toNodeLabel(e:min). However there is a problem. While

Constraint (7.13) (partially) orders the nodes in stBef(k) such that Constraint (7.12) may use that order to break the symmetry of reconnecting the arcs to the input ports of each comparison node in stAft(k), the Constraint (7.15) does not determine the node labels in stBef(k), which in turn totally determine the order of the nodes in stBef(k). Thus the symmetry breaking criteria for Constraint (7.14) are not determined.

Breaking like $\mathfrak{B}_2$

We consider the second option that is similar to $\mathfrak{B}^2$.

We introduce auxiliary variables $vbp^k_{a:p,b}$ that indicate that viaBefPos(c:p) = b where c is the ath comparison node in layer k.

$$\mathcal{B} = \{vbp^k_{a:p,b} \mid k \in L \setminus \{0\}, a{:}p \in \mathrm{ip}(k), b \in \mathrm{ns}(k) \cup \{\mathrm{unused}\}, a \neq b\}$$

We make sure that in a satisfying assignment the truth values of these variables are consistent with their intended semantics by adding the following constraint:

$$\bigwedge_{\substack{k\in L\setminus\{0\}\\ a\in G\\ b:p,c:q\in ip(k)\\ b\neq c}} \left(\left(vbp^k_{b:p,c} \wedge vbp^k_{c:q,b}\right) \leftarrow \left(\neg ng^{k-1}_a \wedge w^{\mathrm{bef}(k)}_{a:\min,b:p} \wedge w^{\mathrm{bef}(k)}_{a:\max,c:q}\right)\right) \tag{7.16}$$

We add the following constraint for odd n if for $vbp^k_{b:p,c}$ the position c is the position of the unused node.

$$\bigwedge_{\substack{k\in L\setminus\{0\}\\ a\in G\\ b:p,\in ip(k)}} \left(vbp^k_{b:p,\mathrm{unused}} \leftarrow \left(\neg ng^{k-1}_a \wedge \left(\begin{array}{l} \left(w^{\mathrm{bef}(k)}_{a:\min,b:p} \wedge w^{\mathrm{bef}(k)}_{a:\max,\mathrm{unused:in}}\right)\\ \vee \left(w^{\mathrm{bef}(k)}_{a:\max,b:p} \wedge w^{\mathrm{bef}(k)}_{a:\min,\mathrm{unused:in}}\right)\end{array}\right)\right)\right) \tag{7.17}$$

Since the constraints only cover the $\leftarrow$ direction they admits false positives, i.e. that $vbp^k_{a:p,b}$ is assigned true but we do not have viaBefPos(c:p) = b where c is the ath comparison in layer k.

We break the symmetry of reconnecting the arcs to the input ports of the ath comparison in layer k except for the first layer. For that we make sure that in a satisfying assignment we have viaBefLabel(e:in1) < viaBefLabel(e:in2) where e is the ath comparison node in layer k. Hence we disallow viaBefLabel(e:in1) > viaBefLabel(e:in2). In the following constraint we have i = viaBefLabel(e:in2) and j = viaBefLabel(e:in1). Obviously we only need to break the symmetry if the ath comparison node is not split into two phantom nodes.

$$\bigwedge_{\substack{k\in L\setminus\{0\}\\ a,b,c\in G\\ a\neq b\\ a\neq c\\ b\neq c\\ i,j\in C\\ i<j}} \left(ng^k_a \vee \neg vbp^k_{a:\mathrm{in2},b} \vee \neg vbp^k_{a:\mathrm{in1},c} \vee \neg_i y^{k,\mathbb{G}}_{b:\mathrm{in1}} \vee \neg_j y^{k,\mathbb{G}}_{c:\mathrm{in1}}\right) \tag{7.18}$$

By our induction hypothesis we assume that the channel labels in layer k are already determined. Thus Constraint (7.18) eliminates the aforementioned false positives.

Next we break the symmetry of reordering nodes within a layer. For simplicity let us first assume that n is even and every channel is used by a gate in every layer. In other words we assume that there are no phantom nodes.

Similar to Constraint (7.13) we make sure that for Order 1 we disallow the case that for node positions $b < a$ the min port of the bth comparison node connects to a greater port position in the next stage than the min port of the ath node. Note that unlike Constraint (7.13) we consider the port position in the next stage instead of the node position, because we break similar to $\mathfrak{B}_2$. Further note that due to the Constraint (7.7e) the compared channel labels are never equal such that we completely break the symmetry of reordering nodes in $\mathrm{stBef}(k)$.

$$\bigwedge_{\substack{k\in B\setminus\{0\}\\ i,j\in C\\ i<j}} \bigwedge_{\substack{e:p\in\mathrm{ipnAft}(k)\\ f:q\in\mathrm{ipnAft}(k)}} \bigwedge_{\substack{a,b\in G\\ b<a}} \left(\left({}_i y^{\mathrm{stAft}(k),\mathbb{G}}_{e:p} \wedge {}_j y^{\mathrm{stAft}(k),\mathbb{G}}_{f:q}\right) \rightarrow \left(\neg w^k_{a:\min,e:p} \vee \neg w^k_{b:\min,f:q}\right)\right) \tag{7.19}$$

Note that our induction hypothesis assumes that the channel labels of the nodes in $\mathrm{stAft}(k)$ are already determined. Due to Constraint (7.7i) we can therefore simplify the symmetry breaking constraint to:

$$\bigwedge_{k\in L} \bigwedge_{\substack{i,j\in C\\ i<j}} \bigwedge_{\substack{a,b\in G\\ b<a}} \left(\neg\, {}_i y^{k,\mathbb{G}}_{a:\min} \vee \neg\, {}_j y^{k,\mathbb{G}}_{b:\min}\right) \tag{7.20}$$

The symmetry break remains complete due to the Constraint (7.7d).

Now let us assume that there may also be phantom nodes. We use the Constraint (7.21) to order comparison nodes that are split into two phantom nodes after comparison nodes that are not. If two comparison nodes are not split into two phantom nodes, then the Constraint (7.22) orders them like Constraint (7.20). Phantom nodes that both stem from splitting a comparison node are ordered by the Constraint (7.23). Recall that we order a tuple of node position and output port type first by the position and if the positions are equal we order by output port type. Finally the Constraint (7.24) ensures that for odd n the unused node is last, i.e. there is no other phantom node with a greater

channel label than the channel label of the phantom node at the unused position.

$$\bigwedge_{k\in L}\ \bigwedge_{\substack{a,b\in G\\ b<a}} \left(\neg ng_a^k \vee ng_b^k\right) \tag{7.21}$$

$$\bigwedge_{k\in L}\ \bigwedge_{\substack{a,b\in G\\ b<a}}\ \bigwedge_{\substack{i,j\in C\\ i<j}} \left(\left(ng_a^k \wedge ng_b^k\right) \rightarrow \left(\neg\, {}_i y_{a:\min}^{k,\mathbb{G}} \vee \neg\, {}_j y_{b:\min}^{k,\mathbb{G}}\right)\right) \tag{7.22}$$

$$\bigwedge_{k\in L}\ \bigwedge_{\substack{a:p,b:q\in \mathrm{op}(k)\\ b:q<a:p}}\ \bigwedge_{\substack{i,j\in C\\ i<j}} \left(\left(\neg ng_a^k \wedge \neg ng_b^k\right) \rightarrow \left(\neg\, {}_i y_{a:p}^{k,\mathbb{G}} \vee \neg\, {}_j y_{b:q}^{k,\mathbb{G}}\right)\right) \tag{7.23}$$

$$\bigwedge_{k\in L}\ \bigwedge_{\substack{i,j\in C\\ i<j}}\ \bigwedge_{a:p\in \mathrm{op}(k)} \left(ng_a^k \rightarrow \left(\neg\, {}_j y_{a:p}^{k,\mathbb{G}} \vee \neg\, {}_i y_{\mathrm{unused:out}}^{k,\mathbb{G}}\right)\right) \tag{7.24}$$

These constraints break all labeled matchings k in B with comparison nodes in $\mathrm{stBef}(k)$. Hence the only labeled matching remaining is $k = 0$, that is the labeled matching between the input stage and the first layer.

To break the symmetry of reconnecting the arcs to the input ports in the first layer we add the following constraint that is similar to Constraint (7.14):

$$\bigwedge_{\substack{a\in \mathrm{ns}(0)\\ b:p\in \mathrm{opn}(0)\\ c:q\in \mathrm{opn}(0)\\ i,j\in C\\ i<j}} \left(\left({}_j y_{c:\mathrm{in1}}^{\mathrm{bef}(k),\mathbb{G}} \wedge {}_i y_{b:\mathrm{in1}}^{\mathrm{bef}(k),\mathbb{G}}\right) \rightarrow \left(\neg w_{c:q,a:\mathrm{in1}}^{k} \vee \neg w_{b:p,a:\mathrm{in2}}^{k}\right)\right) \tag{7.25}$$

Notably since the input stage has no comparison nodes, this determines the channel labels of the nodes in the input stage.

Hence to break the symmetry of reordering the network input nodes in the input stage we add the following constraint such that $\mathbb{G}$ is an id-line representation. This is Constraint (7.1g) but for $\mathbb{G}$ instead of $\mathbb{I}$.

$$\bigwedge_{i\in C} {}_i y_{i:\mathrm{out}}^{\mathrm{in},\mathbb{G}} \tag{7.26}$$

Similarly if two arcs that connect to the same comparison node c in layer $\mathrm{stAft}(k)$ both start at phantom nodes in layer $\mathrm{stBef}(k)$, then the Constraint (7.23) breaks the symmetry of reconnecting the arcs to the input ports of c. If one of these two arcs starts at a phantom node in layer $\mathrm{stBef}(k)$ while the other starts at a comparison node in layer $\mathrm{stBef}(k)$, then we break the symmetry of reconnecting the arcs to the input ports with the following constraint:

$$\bigwedge_{\substack{k\in L\setminus\{0\}\\ a\in G\\ b:p,c:q\in \mathrm{opn}(k-1)}} \left(ng_a^k \vee \neg w_{b:p,a:\mathrm{in2}}^{\mathrm{bef}(k)} \vee \neg w_{c:q,a:\mathrm{in1}}^{\mathrm{bef}(k)} \vee \neg ng_b^{k-1} \vee ng_c^{k-1}\right) \tag{7.27}$$

Further note that the break of the symmetry of reconnecting the arcs to the input ports of each comparison node outside the first layer is not complete, because we may have viaBefLabel(c:in1) = viaBefLabel(c:in2) for some comparison node c.

If both arcs from the same comparison node in layer $k-1$ connect to the same comparison node in layer k, then we break the symmetry with the constraint:

$$\bigwedge_{\substack{k\in L\setminus\{0\}\\ a,b\in G}} \left(\neg ng_a^k \wedge \neg ng_b^{k-1}\right) \rightarrow \left(\neg w^{\mathrm{bef}(k)}_{b:\mathrm{min},a:\mathrm{in2}} \vee \neg w^{\mathrm{bef}(k)}_{b:\mathrm{max},a:\mathrm{in1}}\right) \tag{7.28}$$

Otherwise we may have viaBefLabel(c:in1) = viaBefLabel(c:in2) for some comparison node c (and e) if for some k in B we have that two comparison nodes a and b in stBef(k) connect to the same two comparison nodes c and e in stAft(k). We make a case distinction:

- If both the min port of a and the min port of b connect to the same node f in stBef, then we know that the symmetry of reconnecting the arcs to the input ports of f is broken by Constraint (7.22) and our invariant that $chan_{\mathbb{G}}(f\text{:in1}) < chan_{\mathbb{G}}(f\text{:in2})$. Without loss of generality we assume that $c = f$. Then we still need to break the symmetry of reconnecting the arcs to the input ports of e. Similar to before we may sort the arcs to the input ports of e by viaBefLabel except that we do not consider the channel label of node e, which we defined to be e:in1 but rather we compare the channel labels of the two input ports of e that the nodes a and b connect to.

  We introduce auxiliary variables $e^k_{a:p,b:q}$ that indicate that the arcs to a:p and b:q in layer k connect to the same comparison node in the preceding stage.

  $$\mathcal{E} = \{e^k_{a:p,b:q} \mid k \in L \setminus \{0\}, a\text{:}p, b\text{:}q \in \mathrm{ip}(k), a < b\}$$

  We make sure that in a satisfying assignment the truth values of these variables are consistent with their intended semantics by adding the following constraint:

  $$\bigwedge_{e^k_{b:p,c:q}\in\mathcal{E}} \left(e^k_{b:p,c:q} \leftarrow \left(\neg ng_a^{k-1} \wedge \left(\begin{array}{r}\left(w^{\mathrm{bef}(k)}_{a:\mathrm{min},b:p} \wedge w^{\mathrm{bef}(k)}_{a:\mathrm{max},c:q}\right)\\ \vee \left(w^{\mathrm{bef}(k)}_{a:\mathrm{max},b:p} \wedge w^{\mathrm{bef}(k)}_{a:\mathrm{min},c:q}\right)\end{array}\right)\right)\right) \tag{7.29}$$

  Again since this constraint only covers the $\leftarrow$ direction it admits false positives, i.e. that $e^k_{a:p,b}$ is assigned true but the arcs to a:p and b:q in layer k do not connect to the same comparison node in layer $k-1$.

  We return to our case where a and b in stBef(k) connect to the same two comparison nodes c and e in stAft(k) where we assumed $c = f$ such that we still need to break the symmetry of reconnecting the arcs to the input ports of e. Further we assume

that the position g of comparison node f comes before the position h of comparison node e. Reconnecting the arcs changes

$$e^k_{h:\text{in1},g:p} \mapsto \text{true}$$
$$e^k_{h:\text{in2},g:q} \mapsto \text{true}$$

to

$$e^k_{h:\text{in1},g:q} \mapsto \text{true}$$
$$e^k_{h:\text{in2},g:p} \mapsto \text{true}\,.$$

As planned we break similar to Constraint (7.18) by comparing the the channel labels of the two input ports of f.

$$\bigwedge_{\substack{k\in L\setminus\{0\}\\ r,s\in G\\ r\neq s\\ g,h\in G\\ g<h\\ i,j\in C\\ i<j}} \left(\begin{array}{l} \left(w^{\text{bef}(k)}_{r:\max,a:\text{in1}} \wedge w^{\text{bef}(k)}_{s:\max,a:\text{in2}}\right) \\ \rightarrow \left(ng^k_a \vee \neg e^k_{g:\text{in1},h:\text{in2}} \vee \neg e^k_{g:\text{in2},h:\text{in1}} \vee \neg\, {}_i y^{k,\mathbb{G}}_{g:\text{in1}} \vee \neg\, {}_j y^{k,\mathbb{G}}_{g:\text{in2}}\right) \end{array} \right) \tag{7.30}$$

Otherwise if the position g of comparison node f comes after the position h of comparison node e we merely have to swap the port positions in the index of the variables from $\mathcal{E}$.

$$\bigwedge_{\substack{k\in L\setminus\{0\}\\ r,s\in G\\ r\neq s\\ g,h\in G\\ g>h\\ i,j\in C\\ i<j}} \left(\begin{array}{l} \left(w^{\text{bef}(k)}_{r:\max,a:\text{in1}} \wedge w^{\text{bef}(k)}_{s:\max,a:\text{in2}}\right) \\ \rightarrow \left(ng^k_a \vee \neg e^k_{h:\text{in2},g:\text{in1}} \vee \neg e^k_{h:\text{in1},g:\text{in2}} \vee \neg\, {}_i y^{k,\mathbb{G}}_{g:\text{in1}} \vee \neg\, {}_j y^{k,\mathbb{G}}_{g:\text{in2}}\right) \end{array} \right) \tag{7.31}$$

Note that we can avoid quantifying over r and s by introducing additional auxiliary variables that indicate whether the two ingoing arcs of the ath comparison node in layer k both start at a max port in layer $k-1$.

- Otherwise the min port of a and the min port of b connect to different nodes in stBef. I leave this for future work.

Breaking the Symmetry of Twisting Inputs of Gates in $\mathbb{G}$

Finally we break the symmetry of twisting inputs of gates in $\mathbb{G}$. For that we twist the inputs of a gate such that in the channel labeling $chan_{\mathbb{G}}$ that determines $\mathbb{G}$ the channel labels of the input ports of the comparison node corresponding to that gate satisfy

$chan_{\mathbb{G}}(c{:}\text{in1}) < chan_{\mathbb{G}}(c{:}\text{in2})$. Coincidentally this is the invariant that we decided to maintain for every comparison node c. We add a symmetry breaking constraint that disallows $chan_{\mathbb{G}}(c{:}\text{in1}) > chan_{\mathbb{G}}(c{:}\text{in2})$ where c is the ath gate in layer k.

$$\bigwedge_{k\in L}\ \bigwedge_{a\in G}\ \bigwedge_{\substack{i,j\in C\\ j<i}} \left(\neg\, {}_i y^{k,\mathbb{G}}_{a:\text{in1}} \vee \neg\, {}_j y^{k,\mathbb{G}}_{a:\text{in2}}\right) \tag{7.32}$$

This is basically (7.1k) but instead of sorting the channel labels of the output ports we sort the channel labels of the input ports. In other words we sort the generalized id-channel labeling $\mathbb{G}$ in the reverse direction.

Auxiliary variables

We may introduce an auxiliary variable $z^k_{a:p,b:q}$ to indicate that $chan_{\mathbb{G}}(c{:}p) < chan_{\mathbb{G}}(e{:}q)$ where $chan_{\mathbb{G}}$ is the channel labeling that determines $\mathbb{G}$ and c and e are the ath and bth comparison node in layer k respectively.

$$\mathcal{Z} = \{z^k_{a:p,b:q} \mid k \in S, (a{:}p, b{:}q) \in \text{ipn}(k)^2 \cup \text{opn}(k)^2\}$$

We make sure that in a satisfying assignment the truth values of these variables are consistent with their intended semantics by adding the following constraint:

$$\bigwedge_{z^k_{a:p,b:q}\in\mathcal{Z}} \left(z^k_{a:p,b:q} \leftarrow \bigvee_{\substack{i,j\in C\\ i<j}} \left({}_i y^{k,\mathbb{G}}_{a:p} \wedge {}_j y^{k,\mathbb{G}}_{a:p}\right)\right) \tag{7.33}$$

$$\bigwedge_{z^k_{a:p,b:q}\in\mathcal{Z}} \left(z^k_{a:p,b:q} \rightarrow \bigwedge_{\substack{i,j\in C\\ i>j}} \left(\neg\, {}_i y^{k,\mathbb{G}}_{a:p} \wedge \neg\, {}_j y^{k,\mathbb{G}}_{a:p}\right)\right) \tag{7.34}$$

By using these auxiliary variables we can avoid quantifying over i and j in the Constraints (7.18), (7.19), (7.20), (7.22), (7.23), (7.24), (7.25), (7.30), (7.31) and (7.32). This reduces the number of clauses added by these constraints by a factor that is quadratic in n.

Summary

We considered various ways to break the symmetries of

- reconnecting the arcs to the input ports of each comparison node
- reordering phantom/comparison nodes within a layer
- reconnecting the network input nodes
- twisting inputs of gates in $\mathbb{G}$

in order to break the symmetry of twisting inputs of gates in $\mathbb{I}$ (and $\mathbb{R}$). In particular we have motivated an additional id-channel labeling that enables a complete symmetry break under the assumption that we handle the case viaBefLabel(c:in1) = viaBefLabel(c:in2) completely.

7.5 Discussion of the Graph Encoding

The graph encoding is noteworthy, as the graph encoding from Section 7.3 encodes the relation between $\mathbb{R}$ and $\mathbb{I}$ with a number of clauses that are cubic in n, whereas the encoding of the relation between $\mathbb{R}$ and $\mathbb{I}$ from Section 6.1.3 adds a number of clauses that is quartic in n.

However focusing on these clause savings is misleading, because the encoding of the relation between $\mathbb{R}$ and $\mathbb{I}$ via their common graph-representation does not propagate well.

Recall from Section 3.1.3 that given π one can permute the channels of a standard sorting network N by π and standardize the result to obtain another standard sorting network N'. In contrast given π we do not know on the first try how to twist the inputs of each gate to obtain the same network, even though we know it is possible. Hence there is some kind of inference possible that only works in the forward direction, i.e. from the first layers.

Given π encoded by the Constraints (6.1c) or (7.1h) and (7.1g) and a partial assignment of the first few layer of one network, we expect our encoding to infer the same number of layers in the other network.

For the encoding of the relation between $\mathbb{R}$ and $\mathbb{I}$ from Section 6.1.3 we discussed in Section 6.1.4 that this is possible with additional clauses that facilitate propagation. In contrast the graph encoding of the relation between $\mathbb{R}$ and $\mathbb{I}$ does not perform this forward inference even under the assumption that the arc variables $\mathcal{W}$ are partially assigned for these first layers. In particular the Constraint (7.7j) does not infer the twist of the channel labels after a layer given a partial assignment of the channel labels before the layer, whereas the Constraints (6.1r) and (6.1w) do just that.

Note that we may combine the different encodings of the relation between $\mathbb{R}$ and $\mathbb{I}$ with the following relation between variables from $\mathcal{P}$ and channel label variables from $\mathcal{Y}$. This

is one way to recover forward inference on the graph encoding.

$$\bigwedge_{\substack{p^k_{h,i}\in\mathcal{P}\\ a:p\in\mathrm{opnBef}(k)}} \begin{pmatrix} p^k_{h,i} \to \left({}_h y^{\mathrm{stBef}(k),\mathbb{R}}_{a:p} \leftrightarrow {}_i y^{\mathrm{stBef}(k),\mathbb{I}}_{a:p}\right) \\ p^k_{h,i} \leftarrow \left({}_h y^{\mathrm{stBef}(k),\mathbb{R}}_{a:p} \wedge {}_i y^{\mathrm{stBef}(k),\mathbb{I}}_{a:p}\right) \end{pmatrix} \tag{7.35}$$

$$\bigwedge_{\substack{p^k_{h,i}\in\mathcal{P}\\ a:p\in\mathrm{ipnAft}(k)}} \begin{pmatrix} p^k_{h,i} \to \left({}_h y^{\mathrm{stAft}(k),\mathbb{R}}_{a:p} \leftrightarrow {}_i y^{\mathrm{stAft}(k),\mathbb{I}}_{a:p}\right) \\ p^k_{h,i} \leftarrow \left({}_h y^{\mathrm{stAft}(k),\mathbb{R}}_{a:p} \wedge {}_i y^{\mathrm{stAft}(k),\mathbb{I}}_{a:p}\right) \end{pmatrix} \tag{7.36}$$

Nonetheless we had to assume that the arc variables are partially assigned for these first layers. In particular encoding the arc variables $\mathcal{W}$ introduces the graph symmetries of reconnecting the input nodes, reconnecting the arcs to the input ports of each comparison node and reordering nodes within a layer. We would like that a partial assignment of the gate variables $\mathcal{G}$ of either $\mathbb{I}$ or $\mathbb{R}$ infers an assignment of the arc variables $\mathcal{W}$ by unit propagation. But due to the graph symmetries it is not clear which assignment of $\mathcal{W}$ should be inferred. This lack of propagation due to the graph symmetries is a major weakness of the graph encoding.

At the same time the graph symmetries may be considered a redeeming quality of the graph encoding. By making the symmetries explicit we can break them such that they do not cancel with the symmetry of twisting inputs of comparison nodes in the channel labelings that determine $\mathbb{I}$ and $\mathbb{R}$. This enables us to break the symmetry of twisting inputs on standard network with a two-layer prefix!

For comparison in my bachelor thesis I break the symmetry of twisting inputs of gates in the SAT encoding of a channel network [19]. There I observe that twisting the inputs of a gate c may change gates in the prefix network before c from min-max to max-min. In my bachelor thesis I concluded from this that we can only partially break the atomic symmetry of twisting inputs of a single gate on a standard network. To completely break the symmetry of twisting inputs we need to encode a generalized network instead. Even worse the approach in my bachelor thesis is incompatible with fixing the first two layers.

Considering this it is a notable result that we are able to break the symmetry of twisting inputs with the first two layer fixed by breaking the graph symmetries in the graph encoding. Technically the symmetry break is only complete wrt classes of assignments that represent sorting networks, as we rely on the constraints that encode the network runs to disallow sorting networks that only sort under permuted outputs. This way we can be sure that the symmetry of reconnecting the network output nodes is broken.

It may also be that simply fixing some gate variables to false in the third layer as mentioned at the beginning of Section 7.4 is the best approach. I speculate that $n = 8$ requires fixing gate variables in at most the first three layers, $n = 16$ requires fixing gate variables in at most the first four layers and so on. I did not investigate this further,

because I initially devised the graph encoding not to break the symmetry by twisting inputs but to relate $\mathbb{I}$ and $\mathbb{R}$. Later on I developed the encoding from Section 6.1.3 and deemed it superior, because it propagates better.

Perhaps the graph encoding can be developed further such that it achieves the same propagation properties as the encoding from Section 6.1.3 and 6.1.4 with only a cubic number of clauses. In particular the (graph) symmetry breaking constraints should at least infer the arc variables under the assumption that the gate variables in either the first few or last few layers in $\mathbb{I}$ or $\mathbb{R}$ are partially assigned.

8 Conclusion

In this thesis we considered the sorting network depth optimization problem that asks for the minimum depth of a sorting network on a given number of elements n. In particular we looked at depth lower bound proofs for $2^4 \leq n \leq 2^5$. Depth lower bound proofs show that there is no sorting network of a given depth on n elements.

In the approach that yielded the latest result for $n = 17$ we reduce the problem to Boolean SAT for different equivalence classes that determine the graph structure of the first two layers of the network. A network may either be conceptualized as channels, i.e. data locations of the sequence that is sorted, or alternatively by its graph structure. It suffices to consider standard networks, which are a special kind of channel network. To prove the depth lower bound we solve a SAT instance for a representative standard prefix network from each equivalence class of two-layer prefixes with a SAT solver.

The SAT encoding includes the network runs of an exponentially sized subset of all 2^n Boolean input sequences. Due to this exponential size the encoding of the network runs dominates all other parts of the encoding. By the subnetwork optimization we only encode the network run on a subrange of the channels ignoring leading zeros and trailing ones in a Boolean input sequence. We call such a subrange a window. Previous work employs greedy algorithms to optimize the representative prefix. These algorithms pick a prefix that minimizes the total window size of the encoded runs.

8.1 Minor Contributions

As the first minor contribution of this thesis we distinguished two distinct objective functions namely the total window size and the total squared window size. The former optimizes the number of variables added by the encoding of the network runs, whereas the latter optimizes the number of clauses. We considered a case where two distinct prefixes yield the same total window size and further optimizing by the total squared window size saves clauses in the encoding.

As the second minor contribution we utilized different encodings of the at-most-one cardinality constraint from the SAT solving literature in the sorting network SAT encoding. Specifically in the encoding of a channel network a channel may be used by at most one gate in every layer.

We used the ladder encoding to encode variables that indicate whether there is a gate that outputs the smaller (greater) element on a given channel in a given layer in different given subnetworks on an increasing number of elements. Additionally we used the commander encoding to enforce that either there is no gate using a channel in a layer or there is a gate that outputs either the smaller or the greater element on that channel in that layer.

The different at-most-one constraints from the literature fit these use cases very naturally. Moreover the systematic study of the "at most one gate uses a channel per layer" constraint revealed clauses of a certain form from the commander encoding that speed up solving by increasing propagation. This is an improvement to the state of the art, because encodings from previous work do not include such clauses.

8.2 Encoding Another Network with its Entrances Reordered

As the main contribution I proposed to additionally encode the standard network that is isomorphic to the already encoded standard network, but whose entrances are in reverse order. Thus a network run of an input sequence in one network corresponds to the network run of the reverse sequence in the other network. We exploit this by conveniently encoding network runs in one or the other network according to whether the window size of a sequence is smaller than the window size of the reverse sequence.

More generally the method works for any permutation of the entrances, not only reversal. Hence it generalizes the subnetwork optimization to van Voorhis lemma on standard networks. In theory we can encode more than two isomorphic channel networks with more than two permutations of the entrances, but this is not viable for the next outstanding depth lower bound conjectured to be at $n = 21$ due to the costly encoding of their isomorphism relation.

With this approach the problem of prefix optimization becomes the problem of finding an optimal representative prefix pair where the permutation between the prefix entrances is freely selectable for each prefix class. Prefix pair optimization is likely harder than prefix optimization due to the free choice of the permutation. Note that currently there are only non-optimal greedy algorithms for prefix optimization by Ehlers, who as far as I can tell is also the only person that tackled this problem. I leave it to future work to devise a better/optimal algorithm for prefix optimization in particular an algorithm for prefix pair optimization that considers different permutations of the prefix entrances.

8.2.1 Encodings

We considered two ways to encode that two standard networks are isomorphic with a certain permutation between their inputs.

- Relating the two networks directly
- Relating the two network indirectly via an encoding of their common graph structure

Both encodings add additional variables and clauses. We concerned ourselves with the number of clauses specifically, as too many clauses slow down recurring solving steps such as unit propagation. The direct encoding adds a number of clauses quartic in n and linear in d. In contrast we only add a number of clauses cubic in n and linear in d to relate a channel network with its graph structure, although there are variants that are quartic n. Thus the indirect encoding also only adds a number of clauses cubic in n and linear in d.

Notably the faithful encoding of the graph adds a number of clauses cubic in n and *cubic* (or under certain assumptions quadratic) in d. We reduced the number of additional clauses to be linear in d by decomposing the arcs across multiple layers into subarcs such that there are only arcs between adjacent layers.

Both encodings add a number of additional variables quadratic in n and linear in d. Every additional variable doubles the size of the worst case binary search tree of the solver, but this is irrelevant if the solver deterministically infers the variable's truth value by unit propagation. Our study showed that the direct encoding propagates well, whereas the indirect graph encoding does not. For example the graph encoding introduces additional symmetries such that it is not determined, which partial assignments should be inferred in the first place.

Therefore the direct encoding is preferable. However if we can recover the inference behavior of the direct encoding on the indirect encoding by revising it, then the indirect encoding is preferable, because the number of clauses is quartic in n in the direct encoding but only cubic in n in the indirect encoding.

In summary both encodings add a polynomial number of additional variables and clauses. I conjecture that for every (two-layer) prefix equivalence class there is a prefix pair that achieves exponential variable and clause savings over the optimal prefix by saving more through the subnetwork optimization.

8.2.2 Cost-Benefit Analysis for Example Prefixes

Clearly the conjectured exponential savings outweigh the polynomial costs for larger n. As evidence we considered example prefix classes for the latest depth lower bound for $n = 17$ and the conjectured next lower bound for $n = 21$. Encoding the second

isomorphic network with the direct encoding saves variables but adds additional clauses for $n = 17$, whereas for $n = 21$ the direct encoding saves both variables and clauses. Unfortunately the number of clauses for $n = 21$ remains greater than the number of clauses for $n = 17$ by an order of magnitude such that $n = 21$ is most likely still unfeasible.

It should be pointed out that we made some debatable assumptions in the study of the examples. For instance we considered an optimized prefix and its counterpart obtained by entrance reversal instead of finding an optimized prefix pair. Further we only encode runs of inputs from a subset of all inputs, but the ideal subset or a good size of the subset is unknown. In his depth lower bound proof for $n = 17$ Ehlers encoded the runs of the 2000 inputs with the smallest window size for each representative prefix. But one of his evaluations for $n = 16$ suggests that there is an ideal size of the subset that is neither too small nor too great. I believe this ideal size may depend on the difficulty of the prefix (class) such that it is not the same for every class on the same number of elements.

There are several candidate heuristics for the difficulty of a prefix. One heuristic is the number of outputs of a prefix, whose runs we may encode in the prefix's extension. Alternatively we may take the number of variables or clauses that are required to encode all these runs as a heuristic for a prefix's difficulty. Notably the number of prefix outputs is the same for all prefixes from the same equivalence class, whereas the variable or clause number heuristics may differ for different prefixes from the same class.

As another option it may be more practical to encode as many input runs as a fixed budget of clauses or inputs allows depending on available RAM. I do not know whether that was Ehler's reasoning. I leave the evaluation of these alternatives for future work. Likewise I leave the evaluation of my proposal to encode another network with its inputs reordered for future work.

8.3 Symmetry Breaking

Symmetry breaking constraints aim to speed up solving by reducing the solver's search space to the representative assignments from each symmetry class. We devised symmetry breaking constraints that break the graph symmetries introduced by the graph encoding. In particular we broke the symmetries of reordering nodes within a layer, the symmetry of reconnecting the two arcs to the input ports of each comparison node and the symmetry of reconnecting the network input nodes. We did not break the symmetry of reconnecting the network output nodes, as this is an incorrect symmetry that changes the functional behavior of the network. Reconnecting the network input nodes may also change a network's functional behavior but never if the network sorts. This break of the graph symmetries in the indirect encoding does not propagate well, i.e. it does not recover the inference properties of the direct encoding. Perhaps it can be improved.

It may be that there are two distinct isomorphic sorting networks with the same two-layer prefix. A necessary condition for this is that the two-layer prefix has a graph structure with permutable connected components. While many prefix classes have a structure without permutable components, some two-layer prefix classes e.g. the Green filter have a number of permutable components that is constant in n.

It turns out that the symmetry breaking constraints that break the graph symmetries interact with the symmetry of networks with isomorphic graph structure. Specifically they only admit a single representative from each isomorphism class of sorting networks. Unfortunately they do not similarly restrict the satisfying assignments from isomorphism classes of comparison networks that do not sort to only a single representative assignment. Hence the symmetry breaking constraints only satisfy a weakened notion of completeness wrt network graph isomorphism.

Bibliography

[1] D. E. Knuth, *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*, 2nd ed. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1998.

[2] D. G. O'connor and R. J. Nelson, "Sorting system with n-line sorting switch," U.S. Patent 3 029 413, Apr. 10, 1962. [Online]. Available: https://patents.google.com/patent/US3029413A/en

[3] K. E. Batcher, "Sorting networks and their applications," in *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, ser. AFIPS '68 (Spring). New York, NY, USA: ACM, 1968, pp. 307–314. [Online]. Available: http://doi.acm.org/10.1145/1468075.1468121

[4] D. Hoyte. (2012) Introduction to sorting networks. [Online]. Available: https://hoytech.github.io/sorting-networks/

[5] M. Codish, L. Cruz-Filipe, M. Nebel, and P. Schneider-Kamp, "Applying sorting networks to synthesize optimized sorting libraries," *CoRR*, vol. abs/1505.01962, 2015. [Online]. Available: http://arxiv.org/abs/1505.01962

[6] I. Parberry, *Parallel Complexity Theory*. New York, NY, USA: John Wiley & Sons, Inc., 1987.

[7] G. Perrot, S. Domas, and R. Couturier, "Fine-tuned high-speed implementation of a gpu-based median filter," *Journal of Signal Processing Systems*, vol. 75, no. 3, pp. 185–190, Jun. 2014. [Online]. Available: https://doi.org/10.1007/s11265-013-0799-2

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[9] I. Parberry, "The pairwise sorting network," *Parallel Processing Letters*, vol. 2, pp. 205–211, 1992. [Online]. Available: https://doi.org/10.1142/S0129626492000337

[10] B. Dobbellaere. (2020, Jul.) Smallest and fastest sorting networks for a given number of inputs. https://github.com/bertdobbelaere/SorterHunter. [Online]. Available: http://users.telenet.be/bertdobbelaere/SorterHunter/sorting_networks.html

[11] T. Ehlers, "Sat and cp - parallelisation and applications," Ph.D. dissertation, Kiel, 2017. [Online]. Available: https://macau.uni-kiel.de/receive/diss_mods_00021179

[12] I. Parberry, *On the Computational Complexity of Optimal Sorting Network Verification*. Berlin, Heidelberg: Springer-Verlag, 1991, pp. 252–269.

[13] S. Arora and B. Barak, *Computational Complexity: A Modern Approach (Draft)*, Jan. 2007. [Online]. Available: http://theory.cs.princeton.edu/complexity/

[14] G. Bilardi, "Merging and sorting networks with the topology of the omega network," *IEEE Trans. Comput.*, vol. 38, no. 10, p. 1396–1403, Oct. 1989. [Online]. Available: https://doi.org/10.1109/12.35835

[15] D. Bundala and J. Závodný, "Optimal sorting networks," *CoRR*, vol. abs/1310.6271, 2013. [Online]. Available: http://arxiv.org/abs/1310.6271

[16] M. Codish, L. Cruz-Filipe, T. Ehlers, M. Müller, and P. Schneider-Kamp, "Sorting networks: to the end and back again," *CoRR*, vol. abs/1507.01428, 2015. [Online]. Available: http://arxiv.org/abs/1507.01428

[17] J. Harder, "An answer to the bose-nelson sorting problem for 11 and 12 channels," *CoRR*, vol. abs/2012.04400, 2020. [Online]. Available: https://arxiv.org/abs/2012.04400

[18] S.-S. Choi and B.-R. Moon, "Isomorphism, normalization, and a genetic algorithm for sorting network optimization," in *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO'02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 327–334. [Online]. Available: http://dl.acm.org/citation.cfm?id=2955491.2955548

[19] T. Haslop, "Minimal depth sorting networks," 2020, Bachelor's thesis.

[20] D. C. van Voorhis, "Toward a lower bound for sorting networks," in *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, 1972, pp. 119–129. [Online]. Available: https://doi.org/10.1007/978-1-4684-2001-2_12

[21] J. A. R. Fonollosa, "Joint size and depth optimization of sorting networks," *CoRR*, vol. abs/1806.00305, 2018. [Online]. Available: http://arxiv.org/abs/1806.00305

[22] M. Codish, L. Cruz-Filipe, M. Frank, and P. Schneider-Kamp, "Twenty-five comparators is optimal when sorting nine inputs (and twenty-nine for ten)," *CoRR*, vol. abs/1405.5754, 2014. [Online]. Available: http://arxiv.org/abs/1405.5754

[23] R. W. Floyd and D. E. Knuth, "The bose-nelson sorting problem," in *A Survey of Combinatorial Theory*, J. N. Srivastava, Ed. North-Holland, 1973, ch. 15, pp. 163–172. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978072042262750020X

[24] I. Parberry, "A computer assisted optimal depth lower bound for sorting networks with nine inputs," in *Proceedings of the 1989 ACM/IEEE Conference on Supercomputing*, ser. Supercomputing '89. New York, NY, USA: ACM, 1989, pp. 152–161. [Online]. Available: http://doi.acm.org/10.1145/76263.76280

[25] A. Morgenstern and K. Schneider, "Synthesis of parallel sorting networks using sat solvers," in *Berichte aus der Informatik, MBMV, Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen, 14*. Oldenburg: OFFIS;, 2011, pp. 71–80. [Online]. Available: https://www.tib.eu/de/suchen/id/tema%3ATEMA20121101663

[26] T. Ehlers and M. Müller, "Faster sorting networks for 17, 19 and 20 inputs," *CoRR*, vol. abs/1410.2736, 2014. [Online]. Available: http://arxiv.org/abs/1410.2736

[27] ——, "New bounds on optimal sorting networks," *CoRR*, vol. abs/1501.06946, 2015. [Online]. Available: http://arxiv.org/abs/1501.06946

[28] S. W. A. Baddar and K. E. Batcher, "An 11-step sorting network for 18 elements," *Parallel Processing Letters*, vol. 19, no. 1, pp. 97–103, 2009. [Online]. Available: https://doi.org/10.1142/S0129626409000092

[29] T. Ehlers, "Merging almost sorted sequences yields a 24-sorter," *Inf. Process. Lett.*, vol. 118, no. C, p. 17–20, Feb. 2017. [Online]. Available: https://doi.org/10.1016/j.ipl.2016.08.005

[30] M. Marinov and D. Gregg, "Higher lower bounds for the minimal depth of n-input sorting networks," 2014.

[31] ——, "Sorting networks: The final countdown," *CoRR*, vol. abs/1502.05983, 2015. [Online]. Available: http://arxiv.org/abs/1502.05983

[32] V. K. Valsalam and R. Miikkulainen, "Using symmetry and evolutionary search to minimize sorting networks," *J. Mach. Learn. Res.*, vol. 14, no. 1, pp. 303–331, Feb. 2013. [Online]. Available: http://dl.acm.org/citation.cfm?id=2567709.2502591

[33] L. K. Graham and F. Oppacher, "Symmetric comparator pairs in the initialization of genetic algorithm populations for sorting networks," *2006 IEEE International Conference on Evolutionary Computation*, pp. 2845–2850, Jul. 2006.

[34] S. W. A. Baddar, "Finding better sorting networks," Ph.D. dissertation, Kent State University, Kent, OH, USA, May 2009.

[35] (2002) Sat competition. [Online]. Available: http://www.satcompetition.org/

[36] M. Davis and H. Putnam, "A computing procedure for quantification theory," *J. ACM*, vol. 7, no. 3, pp. 201–215, Jul. 1960. [Online]. Available: http://doi.acm.org/10.1145/321033.321034

[37] J. Smock. (2016) A peek inside sat solvers. Clojure/conj. [Online]. Available: http://jonsmock.com/sat-talk/

[38] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: engineering an efficient sat solver," in *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, Jun. 2001, pp. 530–535.

[39] H. Katebi, K. A. Sakallah, and J. a. P. Marques-Silva, "Empirical study of the anatomy of modern sat solvers," in *Proceedings of the 14th International Conference on Theory and Application of Satisfiability Testing*, ser. SAT'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 343–356. [Online]. Available: http://dl.acm.org/citation.cfm?id=2023474.2023510

[40] A. Solar-Lezama, "Program synthesis by sketching," Ph.D. dissertation, Berkeley, CA, USA, 2008.

[41] N. Eén and N. Sörensson, "An extensible sat-solver," in *Theory and Applications of Satisfiability Testing*, E. Giunchiglia and A. Tacchella, Eds. Berlin, Heidelberg: Springer-Verlag, 2004, pp. 502–518.

[42] D. Bundala, M. Codish, L. Cruz-Filipe, P. Schneider-Kamp, and J. Závodný, "Optimal-depth sorting networks," *CoRR*, vol. abs/1412.5302, 2014. [Online]. Available: http://arxiv.org/abs/1412.5302

[43] T. Ehlers. (2018, Aug.) JCSS. [Online]. Available: https://github.com/the-kiel/JCSS

[44] M. Marinov and D. Gregg, "Itemset isomorphism: Gi-complete," *CoRR*, vol. abs/1507.05841, 2015. [Online]. Available: http://arxiv.org/abs/1507.05841

[45] M. Codish, L. Cruz-Filipe, and P. Schneider-Kamp, "The quest for optimal sorting networks: Efficient generation of two-layer prefixes," *CoRR*, vol. abs/1404.0948, 2014. [Online]. Available: http://arxiv.org/abs/1404.0948

[46] J. Krystofiak, G. Marquardt, and T. Haslop, "Sorting networks," in *Project Report - Tackling Practical Problems with Solvers*, 2017, pp. 62–82, not published.

[47] S. Hölldobler and V. H. Nguyen, "An efficient encoding of the at-most-one constraint," TU Dresden, Tech. Rep., 2013.

[48] W. Klieber and G. Kwon, "Efficient cnf encoding for selecting 1 from n objects," in *the Fourth Workshop on Constraint in Formal Verification(CFV)*, 2007.

[49] J. Marques-Silva and I. Lynce, "Towards robust cnf encodings of cardinality constraints," in *Principles and Practice of Constraint Programming – CP 2007*, C. Bessière, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 483–497.

[50] N. J. A. Sloane. (2021) OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences. [Online]. Available: https://oeis.org/A000372

[51] I. P. Gent, K. E. Petrie, and J.-F. Puget, "Symmetry in constraint programming," in *Handbook of Constraint Programming*, F. Rossi, P. van Beek, and T. Walsh, Eds. New York, NY, USA: Elsevier Science Inc., Aug. 2006, ch. 10, pp. 329–376.

Universität Bremen

Nachname _____ Matrikelnr. _____

Vorname/n _____

Diese Erklärungen sind in jedes Exemplar der Bachelor- bzw. Masterarbeit mit einzubinden.

Urheberrechtliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.
Alle Stellen, die ich wörtlich oder sinngemäß aus anderen Werken entnommen habe, habe ich unter Angabe der Quellen als solche kenntlich gemacht.

_____ _____

Datum Unterschrift

Erklärung zur Veröffentlichung von Abschlussarbeiten

Die Abschlussarbeit wird zwei Jahre nach Studienabschluss dem Archiv der Universität Bremen zur dauerhaften Archivierung angeboten.
Archiviert werden:

1) Masterarbeiten mit lokalem oder regionalem Bezug sowie pro Studienfach und Studienjahr 10 % aller Abschlussarbeiten
2) Bachelorarbeiten des jeweils der ersten und letzten Bachelorabschlusses pro Studienfach und Jahr.

○ Ich bin damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

○ Ich bin damit einverstanden, dass meine Abschlussarbeit nach frühestens 30 Jahren (gem. §7 Abs. 2 BremArchivG) im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

○ Ich bin nicht damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

_____ _____

Datum Unterschrift