Master Thesis

Javier González García

February 2021

1 Introduction

A sorting network is a formal representation of a sorting algorithm that for any inputs generates monotically increasing outputs. A sorting network is formed by n channels each of them carrying one input, which are connected pairwise by comparators. A comparator compares the inputs from it's 2 channels and outputs them sorted to the same 2 channels. A comparator network is a sorting network if for any input sequence the output is always the sorted sequence. What makes sorting networks special is their high parallelization capacity, we can create parallel layers of comparators as long as none of them is part of the same input channel at once.

The creation of optimal sorting networks can be divided in 2 tasks. Finding sorting networks with less comparators, also called size optimization. And finding sorting networks with less parallel layers, also called depth optimization. In this thesis we will focus in the size optimization.

The search of optimal size sorting networks involves to test all comparator networks of a giving size. For example to prove the optimality of the sorting network with 11 inputs and 35 comparators we should consider 55 = (11 x 10)/2 possibilities to place each comparator in 2 out of 11 channels. Therefore the search space is of $55^{35} \approx 9 * 10^{60}$ comparator networks.

This problem can only be addressed by using symmetry breaking rules to trim the search space. For this matter first I implemented the method used in [4] called generate and prune. This method is formed by 2 phases. In the generate phase, starting with a one comparator network it iteratively creates new networks with one comparator more in all possible positions. In the prune phase the redundant networks (networks equivalent to others in the set) are removed. This way the search space is reduced to $2.2 * 10^{37}$ to around $3.3 * 10^{21}$ for the 9 inputs network.

In the first part of this thesis, focused in improving the implementation using modern programming languages and reducing the memory and CPU consumption. This itself allowed to reproduce the same results than in [4] with only 10 hours of compute time in a 64 cores computer more modest than the one used in [4] which took one week of computing. However, when trying to address the next open problem, this itself is not enough. Due to the combinatory explosion it would be necessary around a year of computing time in this computer to find a solution for the 11 input problem. This led to the second part of this thesis where I apply heuristics to the generate and prune method. During the test of heuristic I discovered without prove a method that allows to discover nets with the same size than all the already best size networks until 13 inputs. For 14 inputs and above again the sets are too big to be tested in the available hardware.

The combination of heuristic functions with generate and prune has dealt promising results, finding networks of the same size than the state of the art smallest networks in the interval 3-16. In the following chapters I will state with further details the work performed in this master thesis.

2 Sorting networks preliminaries

A comparator network with n inputs is a sequence of comparators, each comparator is formed by a tuple of channels $C = (i_1, j_1); ...; (i_k, j_k)$ where $(1 \leq i_l < j_l \leq n)$. We name size k to the number of comparators the network has. An input $\bar{x} = x_1...x_n \epsilon \{0,1\}^n$ outputs the network as follows: $\bar{x_0} = \bar{x}$ for $0 < l \leq k$, $\bar{x^l}$ is a permutation of $\bar{x}^{l-1}$ exchanging $\bar{x}^{l-1}_{i_l}$ and $\bar{x}^{l-1}_{j_l}$ if $\bar{x}^{l-1}_{i_l} > \bar{x}^{l-1}_{j_l}$ A comparator network is a sorting network if for any n inputs the outputs are the ascending ordered sequence. The reason we take only binary sequences is because of the zero-one principle[3] which states that a comparator network orders all sequences in {0,1} if and only if it sorts all sequences in any ordered set such as the integers set. This also allows to test if a comparator network is a sorting network without testing n! combination of sequences. If a comparator network tests all the binary sequences in 1-n it is enough to state that it is indeed a sorting network.

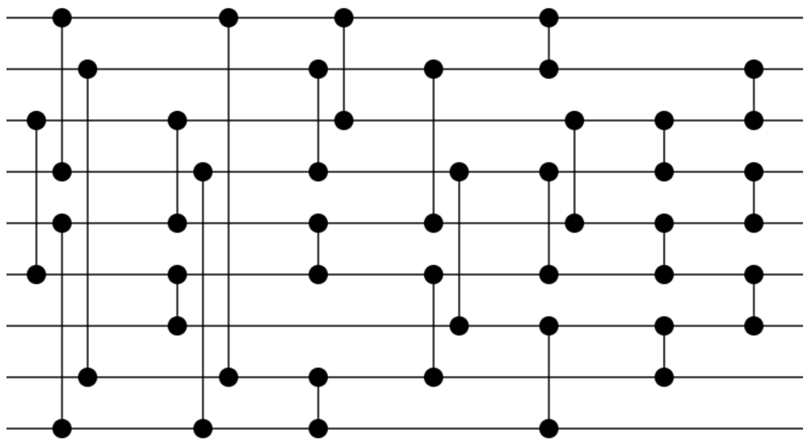


Figure 1: Size 8 sorting network

As stated before the creation of sorting networks with optimal size is the problem of finding networks with the smallest possible set of comparators. Until this date optimal size sorting networks exists for $n \leq 12$, in [1] Floyd and Knuth found optimal size sorting networks for $n \leq 8$. In [4] the optimal size networks for $n = 9 and n = 10$ are proved and [2] proves the optimal size networks for $n = 11 and n = 12$.

3 Generate and Prune

4 Generate and Prune Implementation

Algorithm 1 Generate

```
$result \leftarrow \emptyset$
$N \leftarrow networks$
$C \leftarrow comparators$
**for** $n$ in $N$ **do**
  **for** $c$ in $C$ **do**
    $n' \leftarrow n \bigcup c$
    **if** $n'$ is not redundant **then**
      $result \leftarrow result \bigcup n'$
    **end if**
  **end for**
  $n' \leftarrow$
**end for**
**return** $result$
```

Algorithm 2 Prune

```
$R \leftarrow \emptyset$
$N \leftarrow networks$
**for** $n$ in $N$ **do**
  **for** $r$ in $R$ **do**
    **if** $r$ subsumes $n$ **then**
      $subsumed \leftarrow$ **true**
      break
    **end if**
    **if** $n$ subsumes $r$ **then**
      $R \leftarrow R \setminus r$
    **end if**
  **end for**
  **if** $subsumed$ is **false then**
    $R \leftarrow R \bigcup n$
  **end if**
**end for**
**return** $R$
```

Algorithm 3 Parallel Prune

```
$N \leftarrow networks$
$C \leftarrow Divide(N)$ {Divide N in as many Clusters as processor}
Each processor performs:
PRUNE($C_i$)
**for** $c$ in $C$ **do**
    Remove($c, C \setminus c$)
**end for**
**return** $N$
```

Algorithm 4 Remove

```
$result \leftarrow \emptyset$
$N_i \leftarrow networks$
$N_j \leftarrow networks$
**for** $n_i$ in $N_i$ **do**
    **for** $n_j$ in $N_j$ **do**
        **if** $n_i$ subsumes $n_j$ **then**
            $N_j \leftarrow N_j \setminus n_j$
        **end if**
    **end for**
**end for**
**return** $N_j$
```

5 Subsume Implementations

5.1 Permutations Enumeration

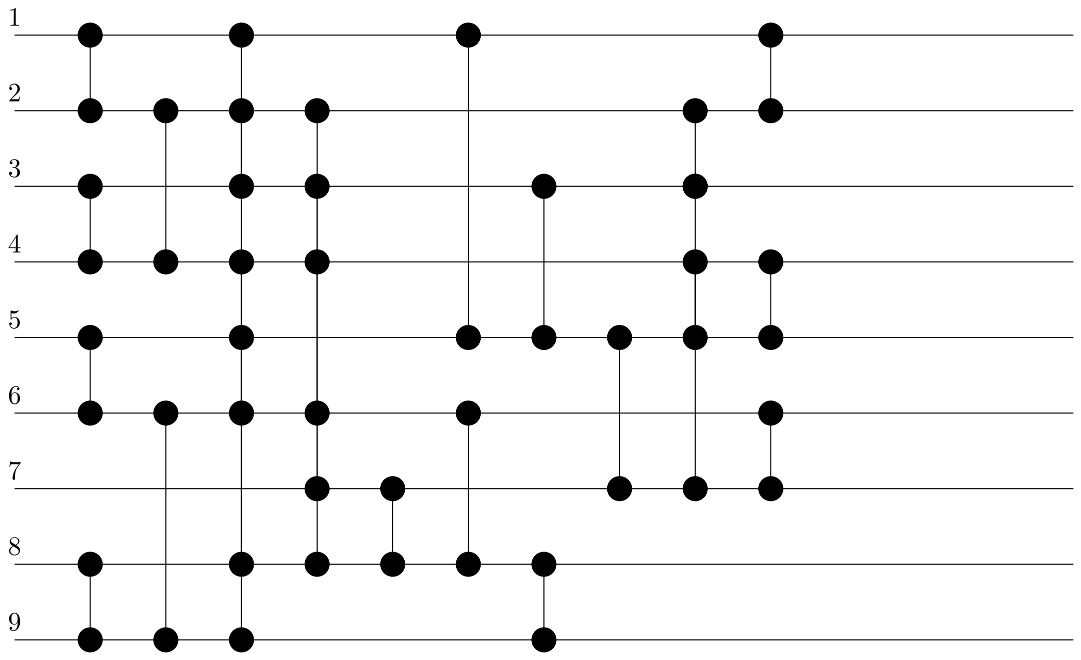
5.2 Bigraph Perfect Matchings

6 Heuristics

In the following table I compare the sets for n=9 of both implementations:

k	1	2	3	4	5	6	7	8	9	10	11	12	13
R9k	1	3	7	20	59	208	807	3415	14,343	55,951	188,730	480,322	854,638
R9k’	1	2	3	7	13	22	41	77	136	229	302	403	531

k	14	15	16	17	18	19	20	21	22	23	24	25
R9k	914,444	607,164	274,212	94,085	25,786	5699	1107	250	73	27	8	1
R9k'	586	570	519	413	314	230	179	123	57	24	8	1



7 Conclusion

References

[1] ROBERT W. FLOYD and DONALD E. KNUTH. Chapter 15 - the bose-nelson sorting problem††the preparation of this report has been supported in part by the national science foundation, and in part by the office of naval research. In JAGDISH N. SRIVASTAVA, editor, *A Survey of Combinatorial Theory*, pages 163–172. North-Holland, 1973.

[2] Jannis Harder. An answer to the bose-nelson sorting problem for 11 and 12 channels, 2021.

[3] Donald Ervin Knuth. *The art of computer programming*, volume 3. Pearson Education, 1997.

[4] Michael Codish; Luís Cruz-Filipe; Michael Frank; Peter Schneider-Kamp. Sorting nine inputs requires twenty-five comparisons. *Journal of Computer and System Sciences*, 2016.