Minimal-Size Sorting Networks for 9 and 10 Inputs

L. Cruz-Filipe[1] M. Codish[2] M. Frank[2]
P. Schneider-Kamp[1]

[1]Dept. Mathematics and Computer Science, Univ. Southern Denmark (Denmark)

[2]Ben-Gurion University of the Negev (Israel)

Roskilde Universitet
June 18th, 2014

Outline

1. Sorting Networks in a Nutshell
2. The Generate-and-Prune approach
3. Conclusions & Future Work

Outline

1. Sorting Networks in a Nutshell
2. The Generate-and-Prune approach
3. Conclusions & Future Work

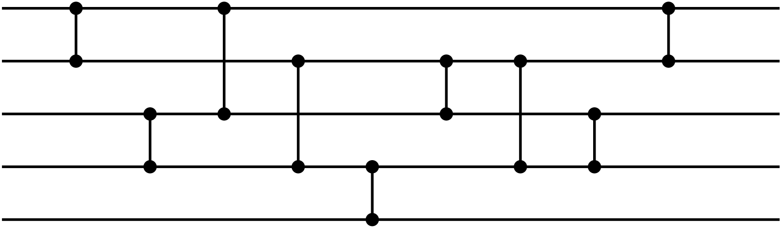
What are sorting networks?

- Oblivious algorithms to sort a given number of inputs
- Easy to implement at the hardware level
- Intrinsically parallel
- Two interesting optimization problems:
  - size (production cost)
  - depth (execution time)

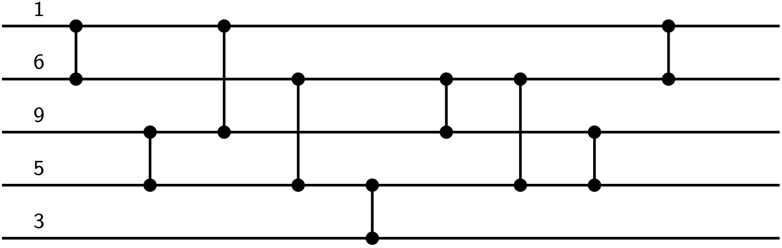
What are sorting networks?

- Oblivious algorithms to sort a given number of inputs
- Easy to implement at the hardware level
- Intrinsically parallel
- Two interesting optimization problems:
  - size (production cost)
  - depth (execution time)
- See Donald E. Knuth, *The Art of Computer Programming*, vol. 3 for more details

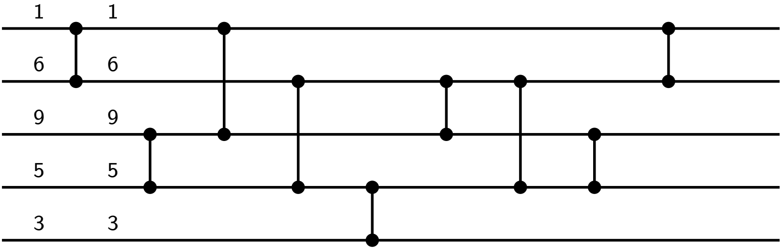
A sorting network



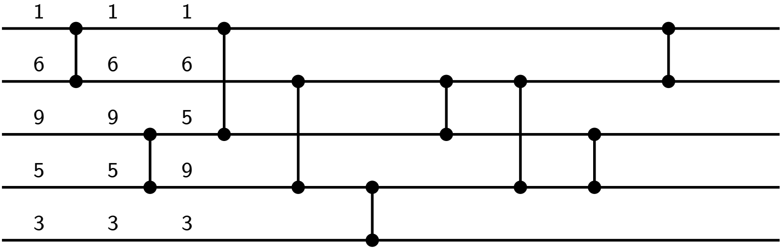
A sorting network



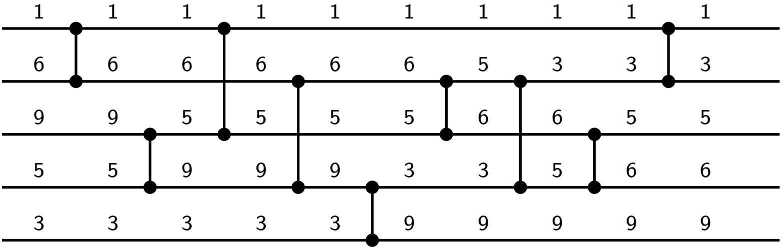
A sorting network



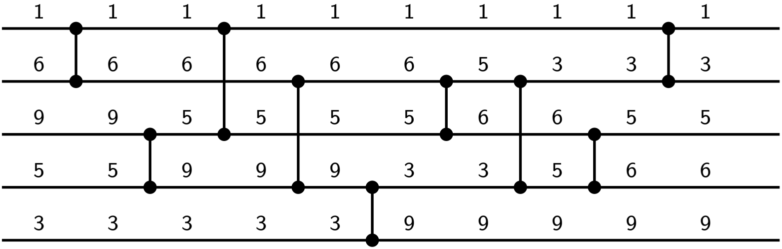
A sorting network



A sorting network



A sorting network



Size

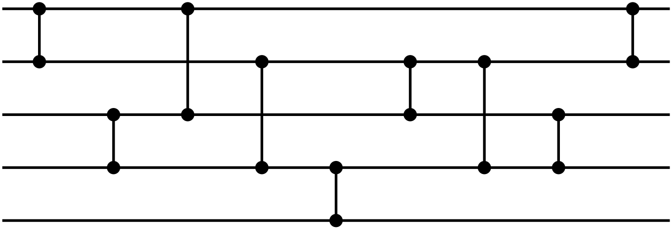
This net has 5 *channels* and 9 *comparators*.

A sorting network

Some of the comparisons may be performed in parallel:

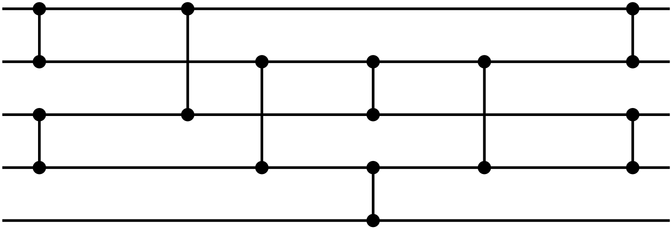
A sorting network

Some of the comparisons may be performed in parallel:



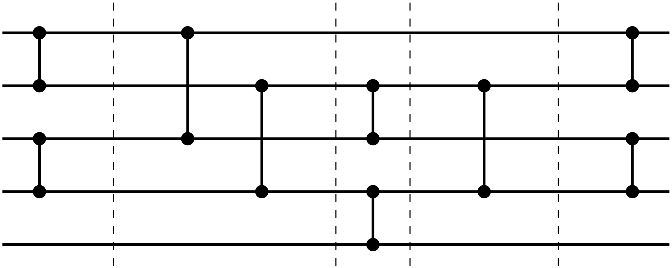
A sorting network

Some of the comparisons may be performed in parallel:



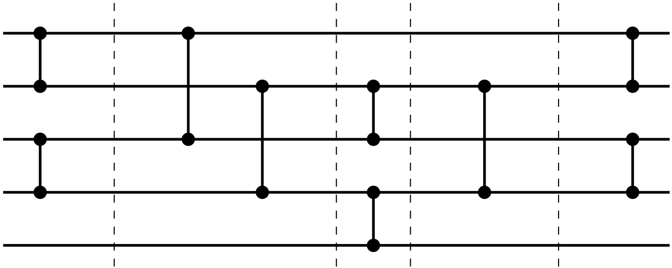
A sorting network

Some of the comparisons may be performed in parallel:



A sorting network

Some of the comparisons may be performed in parallel:



Depth

This net has 5 *layers*.

The optimization problems

The size problem

What is the minimal number of *comparators* on a sorting network on n channels (S_n)?

The depth problem

What is the minimal number of *layers* on a sorting network on n channels (T_n)?

The optimization problems

The size problem

What is the minimal number of *comparators* on a sorting network on n channels (S_n)?

The depth problem

What is the minimal number of *layers* on a sorting network on n channels (T_n)?

Knuth 1973

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$S_n \leq$	3	5	9	12	16	19	25	29	35	39	45	51	56	60
$S_n \geq$	3	5	9	12	16	19	23	27	31	35	39	47	51	55
$T_n \leq$	3	3	5	5	6	6	7	7	8	8	9	9	9	9
$T_n \geq$	3	3	5	5	6	6	6	6	6	6	6	6	6	6

The optimization problems

The size problem

What is the minimal number of *comparators* on a sorting network on n channels (S_n)?

The depth problem

What is the minimal number of *layers* on a sorting network on n channels (T_n)?

Parberry 1991

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$S_n \leq$	3	5	9	12	16	19	25	29	35	39	45	51	56	60
$S_n \geq$	3	5	9	12	16	19	23	27	31	35	39	47	51	55
$T_n \leq$	3	3	5	5	6	6	7	7	8	8	9	9	9	9
$T_n \geq$	3	3	5	5	6	6	**7**	**7**	**7**	**7**	**7**	**7**	**7**	**7**

The optimization problems

The size problem

What is the minimal number of *comparators* on a sorting network on n channels (S_n)?

The depth problem

What is the minimal number of *layers* on a sorting network on n channels (T_n)?

Bundala & Závodný 2013

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$S_n \leq$	3	5	9	12	16	19	25	29	35	39	45	51	56	60
$S_n \geq$	3	5	9	12	16	19	23	27	31	35	39	47	51	55
$T_n \leq$	3	3	5	5	6	6	7	7	8	8	9	9	9	9
$T_n \geq$	3	3	5	5	6	6	7	7	**8**	**8**	**9**	**9**	**9**	**9**

The optimization problems

The size problem

What is the minimal number of *comparators* on a sorting network on n channels (S_n)?

The depth problem

What is the minimal number of *layers* on a sorting network on n channels (T_n)?

Codish, Cruz-Filipe, Frank & Schneider-Kamp (CCFS) 2014

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$S_n \leq$	3	5	9	12	16	19	25	29	35	39	45	51	56	60
$S_n \geq$	3	5	9	12	16	19	**25**	**29**	**33**	**37**	**41**	**45**	**49**	**53**
$T_n \leq$	3	3	5	5	6	6	7	7	8	8	9	9	9	9
$T_n \geq$	3	3	5	5	6	6	7	7	8	8	9	9	9	9

An exponential explosion

- Upper bounds obtained by concrete examples (1960s)
- Lower bounds obtained by mathematical arguments
- HUGE number of nets

An exponential explosion

- Upper bounds obtained by concrete examples (1960s)
- Lower bounds obtained by mathematical arguments
- HUGE number of nets
- Parberry (1991)
  - exploration of symmetries
  - fixed first layer
  - 200 hours of computation

An exponential explosion

- Upper bounds obtained by concrete examples (1960s)
- Lower bounds obtained by mathematical arguments
- HUGE number of nets
- Parberry (1991)
- Bundala & Závodný (2013)
  - exploration of symmetries
  - reduced set of two-layer prefixes
  - intensive SAT-solving

An exponential explosion

- Upper bounds obtained by concrete examples (1960s)
- Lower bounds obtained by mathematical arguments
- HUGE number of nets
- Parberry (1991)
- Bundala & Závodný (2013)
- These techniques are not directly applicable to the size problem
  36 possibilities for each layer when $n = 9$, so
  $36^{24} \approx 2.2 \times 10^{37}$ 24-comparator nets

An exponential explosion

- Upper bounds obtained by concrete examples (1960s)
- Lower bounds obtained by mathematical arguments
- HUGE number of nets
- Parberry (1991)
- Bundala & Závodný (2013)
- These techniques are not directly applicable to the size problem
- CCFS (2014)
  - generate-and-prune
  - combine brute-force generation with optimal (?) reduction
  - compromise between time and space

Outline

Comparator networks

- A *(generalized) comparator network* C on n channels is a sequence of pairs (i, j) (the comparators) such that $1 \leq i \neq j \leq n$.
- A *standard* comparator network C is a generalized comparator network such that $i < j$ for every comparator $(i, j) \in C$.
- The *output* of comparator (i, j) on $\vec{x} = x_1 \dots x_n$ is $\vec{x}'$, where $x'_i = \min(x_i, x_j)$, $x'_j = \max(x_i, x_j)$, and $x'_k = x_k$ for $k \neq i, j$.
- The *output* of C on a sequence $x_1 \dots x_n$, is defined inductively:
  - if C is empty, then $C(x_1 \dots x_n) = x_1 \dots x_n$;
  - if C is $(i, j); C'$ then $C(x_1 \dots x_n) = C'((i, j)(x_1 \dots x_n))$.
- A comparator network C is a *sorting network* if $C(\vec{x})$ is sorted for every input $\vec{x}$.

Well-known results

0–1 lemma (Knuth 1973)

C is a sorting network on n channels iff C sorts all inputs in $\{0,1\}^n$.

Well-known results

0–1 lemma (Knuth 1973)

C is a sorting network on n channels iff C sorts all inputs in $\{0,1\}^n$.

"C is a sorting network on n channels" is co-NP (complete).

Well-known results

0–1 lemma (Knuth 1973)

C is a sorting network on n channels iff C sorts all inputs in $\{0,1\}^n$.

Standardization theorem (Knuth 1973)

If C is a generalized sorting network, then there is a standard sorting network C' with the same size and depth as C.

Well-known results

0–1 lemma (Knuth 1973)

C is a sorting network on n channels iff C sorts all inputs in $\{0, 1\}^n$.

Standardization theorem (Knuth 1973)

If C is a generalized sorting network, then there is a standard sorting network C' with the same size and depth as C.

We will only consider binary inputs and use generalized comparator networks whenever needed.

Well-known results

0–1 lemma (Knuth 1973)

C is a sorting network on n channels iff C sorts all inputs in $\{0,1\}^n$.

Standardization theorem (Knuth 1973)

If C is a generalized sorting network, then there is a standard sorting network C' with the same size and depth as C.

Output lemma (Parberry 1991)

Let C and C' be comparator networks such that $\text{outputs}(C) \subseteq \text{outputs}(C')$. If $C'; N$ is a sorting network, then so is $C; N$.

Permutations (Bundala & Závodný 2013)

Permuted output lemma (I)

If:

- C and C' are standard comparator networks of depth 2;
- π is a permutation of $1..n$ mapping outputs(C) into outputs(C');
- C' can be extended to a sorting network;

then C can also be extended to a standard sorting network of the same depth.

Permutations (Bundala & Závodný 2013)

Permuted output lemma (I)

If:

- C and C' are standard comparator networks **of depth** 2;
- π is a permutation of $1..n$ mapping outputs(C) into outputs(C');
- C' can be extended to a sorting network;

then C can also be extended to a standard sorting network of the same **depth**.

Permutations revisited (CCFS 2014)

Permuted output lemma (II)

If:

- C and C' are standard comparator networks **of equal size**;
- π is a permutation of $1..n$ mapping outputs(C) into outputs(C');
- C' can be extended to a sorting network;

then C can also be extended to a standard sorting network of the same **size**.

Permutations revisited (CCFS 2014)

Permuted output lemma (II)

If:

- C and C' are standard comparator networks **of equal size**;
- π is a permutation of $1..n$ mapping outputs(C) into outputs(C');
- C' can be extended to a sorting network;

then C can also be extended to a standard sorting network of the same **size**.

We say that $C \preceq C'$ when $\pi(\text{outputs}(C)) \subseteq \text{outputs}(C')$ for some permutation π.

Permutations revisited (CCFS 2014)

Permuted output lemma (II)

If:

- C and C' are standard comparator networks **of equal size**;
- π is a permutation of $1..n$ mapping outputs(C) into outputs(C');
- C' can be extended to a sorting network;

then C can also be extended to a standard sorting network of the same **size**.

We say that $C \preceq C'$ when $\pi(\text{outputs}(C)) \subseteq \text{outputs}(C')$ for some permutation π. Note that $\preceq$ is reflexive and transitive. Also, if C is a sorting network, then $C \preceq C'$ for every other standard network C' on the same number of channels.

The algorithms (I)

Generate-and-prune

1. (Init) Set $R_0^n = \{\emptyset\}$ and $k = 0$.
2. Repeat:
   - (Generate) Extend every net in R_k^n with one comparator in every possible way. Let N_{k+1}^n be the set of all results.
   - (Prune) Keep only one element of each minimal equivalence class w.r.t. the transitive closure of $\preceq$. Let R_{k+1}^n be the resulting set.
   - Increase k.

   until $k > 1$ and $|R_k^n| = 1$.

The algorithms (I)

Generate-and-prune

1. (Init) Set $R_0^n = \{\emptyset\}$ and $k = 0$.
2. Repeat:
    - (Generate) Extend every net in R_k^n with one comparator in every possible way. Let N_{k+1}^n be the set of all results.
    - (Prune) Keep only one element of each minimal equivalence class w.r.t. the transitive closure of $\preceq$. Let R_{k+1}^n be the resulting set.
    - Increase k.

   until $k > 1$ and $|R_k^n| = 1$.

If $|R_k^n| > 1$, then there can be no sorting network of size k on n channels. If the algorithm finishes with $|R_k^n| = \{C\}$ and C is a sorting network, then $S_n = k$.

The algorithms (II)

`Generate` (Input R_k^n; output N_{k+1}^n)

- (Init) $N_{k+1}^n = \emptyset$, $C_n = \{(i,j) \mid 1 \leq i < j \leq n\}$
- `for` $C \in R_k^n$ `and` $c \in C_n$: $N_{k+1}^n = N_{k+1}^n \cup \{C; c\}$

The algorithms (II)

Generate (Input R^n_k; output N^n_{k+1})

- (Init) $N^n_{k+1} = \emptyset$, $C_n = \{(i,j) \mid 1 \leq i < j \leq n\}$
- for $C \in R^n_k$ and $c \in C_n$: $N^n_{k+1} = N^n_{k+1} \cup \{C; c\}$

Prune (Input N^n_k; output R^n_k)

- (Init) $R^n_k = \emptyset$
- for $C \in N^n_k$ do
  - for $C' \in R^n_k$: if $(C' \preceq C)$ then mark C
  - if (not_marked(C)) then
    - for $C' \in R^n_k$: if $(C \preceq C')$ then $R^n_k = R^n_k \setminus \{C'\}$
    - $R^n_k = R^n_k \cup \{C\}$

Some numerology

R_k^n	3	4	5	6	7	8
1	1	1	1	1	1	1
2	2	3	3	3	3	3
3	**1**	4	6	7	7	7
4		2	11	17	19	20
5		**1**	10	36	51	57
6			7	53	141	189
7			6	53	325	648
8			4	44	564	2,088
9			**1**	23	678	5,703
10				8	510	11,669
11				4	280	16,095
12				**1**	106	13,305
13					33	6,675
14					11	2,216
15					6	503
16					**1**	77
17						18
18						9
19						**1**

Optimizing Generate

Redundant comparators

A comparator (i, j) is *redundant* w.r.t. C if $x_i \leq x_j$ for every $\vec{x} \in \text{outputs}(C)$.

Optimizing Generate

Redundant comparators

A comparator (i, j) is *redundant* w.r.t. C if $x_i \leq x_j$ for every $\vec{x} \in \text{outputs}(C)$.

Redundant comparators:

- do nothing;
- may not occur in minimal-size sorting networks;
- are easy to detect;
- can be avoided at generation time.

Optimizing `Generate`

Redundant comparators

A comparator (i, j) is *redundant* w.r.t. C if $x_i \leq x_j$ for every $\vec{x} \in \text{outputs}(C)$.

Redundant comparators:

- do nothing;
- may not occur in minimal-size sorting networks;
- are easy to detect;
- can be avoided at generation time.

`Generate` is much faster than `Prune`, so it pays off to do this test at generation time.

Optimizing `Prune` (I)

The big cost in `Prune` is searching for a candidate permutation in the subsumption test.

Optimizing `Prune` (I)

The big cost in `Prune` is searching for a candidate permutation in the subsumption test.

Lemma 1

If the number of sequences with k 1s in outputs(C_a) is greater than that in outputs(C_b) for some k, then $C_a \not\preceq C_b$.

Optimizing `Prune` (I)

The big cost in `Prune` is searching for a candidate permutation in the subsumption test.

Lemma 1

If the number of sequences with k 1s in outputs(C_a) is greater than that in outputs(C_b) for some k, then $C_a \not\preceq C_b$.

This very simple test actually eliminates some 70% unsuccessful subsumption tests!

Optimizing Prune (II)

“Where” sets

$w(C, x, k)$ denotes the set of positions i such that there exists a vector in outputs(C) containing k ones with x at position i.

Optimizing Prune (II)

“Where” sets

$w(C, x, k)$ denotes the set of positions i such that there exists a vector in outputs(C) containing k ones with x at position i.

Lemma 2

If for some $|w(C_a, x, k)| > |w(C_b, x, k)|$ for some x and k, then $C_a \not\preceq C_b$.

Optimizing Prune (II)

“Where” sets

$w(C, x, k)$ denotes the set of positions i such that there exists a vector in outputs(C) containing k ones with x at position i.

Lemma 2

If for some $|w(C_a, x, k)| > |w(C_b, x, k)|$ for some x and k, then $C_a \not\preceq C_b$.

Lemma 3

If $\pi(\text{outputs}(C_a)) \subseteq \text{outputs}(C_b)$, then $\pi(w(C_a, x, k)) \subseteq w(C_b, x, k)$ for all x and k.

Network representation

For efficiency, we store not only the comparator networks (seen as sequences as comparators, but also their set of outputs:

- each output is represented as an integer (the sequence “read” as a binary number)
- outputs are partitioned according to the number of 1s
- each partition is annotated with its “where” sets

Network representation

For efficiency, we store not only the comparator networks (seen as sequences as comparators, but also their set of outputs:

- each output is represented as an integer (the sequence “read” as a binary number)
- outputs are partitioned according to the number of 1s
- each partition is annotated with its “where” sets

This data is computed at generation time, so that it will be readily available every time it is needed for a subsumption test.

Parallelization (I)

With all these optimizations in place, the known values for S_n ($n \leq 8$) could be checked in under one day.

- $n = 6$: two seconds
- $n = 7$: two minutes
- $n = 8$: several hours

Parallelization (I)

With all these optimizations in place, the known values for S_n ($n \leq 8$) could be checked in under one day.

- $n = 6$: two seconds
- $n = 7$: two minutes
- $n = 8$: several hours

A rough estimate of the computation time for $n = 9$ yielded 10–20 years.

Parallelization (I)

With all these optimizations in place, the known values for S_n ($n \leq 8$) could be checked in under one day.

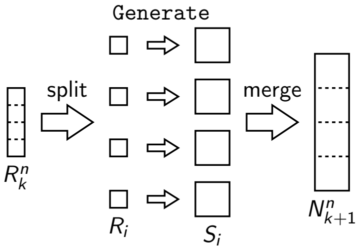
- $n = 6$: two seconds
- $n = 7$: two minutes
- $n = 8$: several hours

A rough estimate of the computation time for $n = 9$ yielded 10–20 years.
With a 288-core cluster available, the precise computation of S_9 became feasible for the first time.

Parallelization (II)

Parallel-Generate
(Input R_k^n; output N_{k+1}^n)

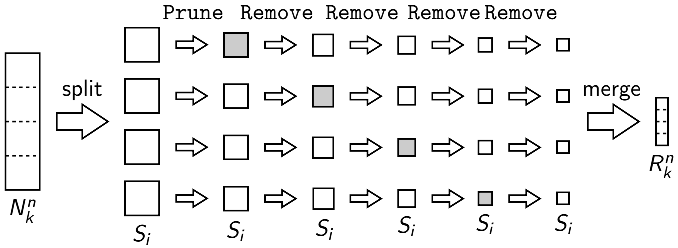
- split R_k^n into sets $R_1, \ldots, R_p$
- for$||^p$ $i \in \{1, \ldots, p\}$ do
  - $S_i = \texttt{Generate}(R_i)$
- $N_{k+1}^n = \biguplus_{1 \le i \le p} S_i$;



Parallelization (III)

Parallel-Prune (Input N_k^n; output R_k^n)

- split N_k^n into sets $S_1, \ldots, S_p$
- $\texttt{for}||^p$ $i \in \{1, \ldots, p\}$: $S_i = \texttt{Prune}(S_i)$
- $\texttt{for}$ $j \in \{1, \ldots, p\}$ do
  - $\texttt{for}||^{p-1}$ $i \neq j$: $S_i = \texttt{Remove}(S_i, S_j)$
- $R_k^n = \biguplus_{1 \leq i \leq p} S_i$;



Outline

1. Sorting Networks in a Nutshell
2. The Generate-and-Prune approach
3. Conclusions & Future Work

Results & Future work

- Exact values of S_9 and S_{10}
- Technique may be adapted to settle higher values which are still unknown
- Algorithms may be useful for *finding* smaller-than-currently-known networks
- Further theoretical results may help proving optimality of best known upper bounds

Thank you!