

SORTING NETWORKS ON 13 INPUTS NEED 45 COMPARATORS

TEJASVI SINGH TOMAR

ABSTRACT. Let $S(N)$ be the least number of comparators in a sorting network on N inputs. The values are known for $N \leq 12$. For 13 inputs the smallest known sorting network, found by Juillé, has 45 comparators, and the best proved lower bound has been 44. We prove $S(13) \geq 45$, so $S(13) = 45$. The proof extends a two-value bound to partial sorting. If a network with L comparators sorts a set X of 0-1 vectors, then for suitable pairs of inputs the largest and the second largest value meet at most $L - c$ comparators, where c is the cost of sorting the residual set left when their paths are deleted. Whether a single network stays within all these budgets is decided by a finite search over the tree of paths of the largest value, the comparators on these paths, and a forest of comparators that carries the second largest value. A breadth-first search over the sets produced by prefixes of a hypothetical sorting network with 44 comparators discards each set that fails this test. The costs of the residual sets are bounded below using $S(11) = 35$ and exact costs of 9- and 7-channel sets. No set survives at depth 25. A separate verifier, with its own implementation of the test, checks the certificate written by the search. It trusts Harder’s program for the exact 9- and 7-channel costs, the bound $S(11) = 35$, and this implementation of the test. A second proof, in Lean 4, checks the same certificate with another implementation of the test, which it proves sound, and proves the 9- and 7-channel bounds that it uses; its large computations run as compiled code. A third proof, also in Lean, has every computation checked by Lean’s kernel and uses only the standard axioms.

1. INTRODUCTION

A comparator network on N inputs is a circuit of comparators, each of which outputs the larger and the smaller of its two input values on two separate leads. A sorting network delivers every input sequence in sorted order. The Bose–Nelson problem [1] asks for $S(N)$, the least number of comparators in a sorting network on N inputs. Floyd and Knuth determined $S(N)$ for $N \leq 8$ [3], Codish, Cruz-Filipe, Frank and Schneider-Kamp proved $S(9) = 25$ and $S(10) = 29$ [2], and Harder proved $S(11) = 35$ and $S(12) = 39$ [5]. Harder’s bound $S(11) \geq 35$ is backed by a certificate, which is checked by a program verified in Isabelle/HOL [4]. The smallest known sorting network on 13 inputs, Juillé’s [7], has 45 comparators. Van Voorhis’s bound $S(N) \geq S(N - 1) + \lceil \log_2 N \rceil$ [12] gives $S(13) \geq 43$. Following the two largest values instead of one gives $S(N) \geq S(N - 2) + P(2, N)$, where $P(2, N)$ is the least possible worst-case number of comparators met by these two values, and the computation $P(2, 13) = 9$ then gives $S(13) \geq 44$ [11].

This bound cannot reach 45, because $P(2, 13) = 9$ is exact. It treats the comparators that the two values do not meet only through the single number $S(11)$. Our starting point is to apply the two-value argument not to a whole network but to its suffix. Let T be a sorting network with at most 44 comparators, and let X be the set of 0-1 vectors produced by its first k comparators. The remaining comparators sort X , and there are at most $44 - k$ of them. Suppose that X contains the vector with a single 1 on a channel x and the vector with 1s on x and on another

Date: 21 October 2026.

2020 Mathematics Subject Classification. Primary 68P10; Secondary 68Q06, 05C05, 68V05.

Key words and phrases. Sorting networks, Bose–Nelson problem, comparator networks, lower bounds, computer-assisted proof.

channel y (Definition 2.6 gives the precise condition). Run the suffix with the largest value on x and the second largest on y . Deleting the comparators that these two values meet leaves a network that sorts a set Y_{xy} on 11 channels, the *core* of the pair. The suffix therefore meets the two values at most $44 - k - s(Y_{xy})$ times in the run from (x, y) , where $s(Y)$ denotes the least size of a network that sorts Y . Different pairs now have different budgets. We call the resulting condition, that a single network stays within all these budgets, a *weighted top-two instance*. If no network meets it, the prefix cannot be completed within 44 comparators.

Deciding a weighted instance is a finite problem. Section 4 extends the structure theory of [11] from all pairs of inputs to an arbitrary set of pairs: the costs of all runs of a network are determined by the tree formed by the paths of the largest value, the comparators that these paths pass, and, for each place where the second largest value leaves the tree, the root and the depth of the forest of comparators that carries it onward. These depths satisfy Kraft's inequality. We decide such instances by a computer search over the possible configurations, and Section 5 proves that each of its tests is valid.

The costs of the cores are bounded below in two ways. Deleting the comparators that the two values meet in the prefix leaves a network on 11 channels whose outputs lie in the core, so $s(Y_{xy}) \geq S(11) - (k - w)$ when the two values meet w of the first k comparators. Sharper bounds come from the same test one level down: a core Y on 11 channels has cores on 9 channels, and if the weighted instance of Y with budget $v - 1$ is infeasible, then $s(Y) \geq v$. Its weights are exact costs of sets on 9 channels, computed by Harder's program [6], or bounds obtained from the instance on 9 channels with exact costs of sets on 7 channels.

The search runs over prefixes, one layer per comparator, with each set taken up to permutation and complementation of the channels. Each comparator that is useful for the set of a prefix (Section 2) gives a child, and a child of depth k is discarded when its weighted instance with budget $44 - k$ is infeasible for one of the two polarities of Definition 2.3. The surviving sets form the next layer. The layers grow to 142,881 sets at depth 14 and then shrink, and layer 25 is empty. The search writes a certificate: a record for each child, which states whether the child survives, repeats an earlier child, or is discarded together with the weights that make its instance infeasible, and a record for each bound on a core that these weights use. A verifier, written from the format of the certificate and the mathematical description without the search code, checks all of them. The verifier and the search share only Harder's program for exact values.

Throughout, a dagger ($\dagger$) marks a statement whose proof includes a computation.

Theorem 1.1. $S(13) = 45$. ^{$\dagger$}

The upper bound is Juillé's network [7]. The lower bound follows from Theorem 7.1 and the checks of Section 8.

Corollary 1.2. For $13 \leq N \leq 32$, $S(N) \geq \Lambda(N) + 1$, where $\Lambda(N)$ is the lower bound of [11, Corollary 1.3]. ^{$\dagger$}

The verification relies on three components that it does not check: Harder's program for the exact costs of the 3,624,738 sets on 9 channels and the 4,028,993 sets on 7 channels that the certificate uses, the bound $S(11) = 35$, and the program that decides weighted instances. Section 10 states what each of them is trusted for and how it was tested. Section 11 gives a second proof of the lower bound, in Lean 4, which checks the same certificate without the verifier and without the program that decides weighted instances. It derives $S(11) \geq 35$ from Harder's certificate [4] and proves every bound on 9 and 7 channels that it needs. Its large computations run as compiled code rather than in Lean's kernel. Section 12 gives a third proof in Lean, in which the kernel checks every computation itself. It uses only the standard axioms of Lean's logic.

Section 2 recalls partial sorting networks and defines cores. Section 3 states the weighted two-value bound, Section 4 reduces it to configurations, and Section 5 describes the search over configurations. Section 6 gives the bounds on cores, Section 7 the search over prefixes and the proof of Theorem 1.1, Section 8 the certificate and its verification, and Section 9 the results. Section 10 lists the trusted components and the cross-checks, Section 11 describes the proof in Lean, Section 12 the proof checked by the kernel, and Section 13 proves Corollary 1.2. The programs, the certificate and the logs are in the supplement [14], and the proof of Section 12 in [15].

2. PARTIAL SORTING NETWORKS AND CORES

2.1. Comparator networks and runs. We use the circuit model of [11, Section 2.1]. A *comparator network* T on a finite set A of channels is a finite acyclic circuit made of input leads, one for each channel, and of comparators. A comparator g has two input leads and two output leads, $\text{hi}(g)$ and $\text{lo}(g)$. Each lead is either an input lead of T or an output lead of exactly one comparator, and it enters at most one comparator. The leads that enter no comparator are the *outputs* of T , and there are $|A|$ of them. Given values on the input leads, each comparator puts the larger of its two input values on $\text{hi}(g)$ and the smaller on $\text{lo}(g)$. The size $|T|$ is the number of comparators. Drawing T on channels along a topological order of its comparators gives a sequence of standard comparators, defined below, followed by a permutation of the channels [5, Section 2]. Conversely, a sequence of comparators on channels is a network: the comparator (i, j) , for channels $i \neq j$, puts the maximum on channel i and the minimum on channel j , and the outputs are identified with the channels. The comparator (i, j) is *standard* if $i > j$.

For distinct channels x and y , the *run* (x, y) puts the largest value M on x , the second largest value m on y , and smaller distinct values on the other inputs. The value M leaves every comparator it enters on hi . The value m leaves a comparator on lo if M is on the other input lead, and on hi otherwise. Hence the leads that carry M and m depend only on (x, y) . We write Π_x for the set of leads that carry M , which depends only on x , $\delta(x)$ for the number of comparators that M enters, and $W(x, y)$ for the set of comparators with M or m on an input lead. A comparator that m enters while M is on its other input lead is a *meeting*.

Deleting a path is the operation of [5, Definition 13]. For a channel x , the network T/x on $A \setminus \{x\}$ is obtained from T by deleting the leads of Π_x and the comparators that they enter, and joining the other input lead of each such comparator to its lo output lead. The input lead x and the output lead at the end of Π_x disappear.

Lemma 2.1. *Let $x \neq y$. In the run (x, y) the value m never occupies a lead of Π_x . The comparators of $W(x, y)$ are the $\delta(x)$ comparators that M enters and the comparators of T/x on the path that the maximum of T/x takes from y , which are the comparators that m enters and that are not meetings. Deleting this path from T/x gives the network $(T/x)/y$, which has $|T| - |W(x, y)|$ comparators.*

Proof. Since M occupies every lead of Π_x in the run (x, y) , the value m occupies none of them. When m enters a comparator g of Π_x , the value M is on the other input lead, and m leaves on $\text{lo}(g)$, which is where the join of T/x leads. At any other comparator m leaves on hi , as the maximum of T/x does. Thus the leads of m are those of the path of the maximum of T/x from y , with the joins expanded, and the comparators that m enters outside Π_x are the comparators of T/x on that path. $\square$

2.2. Partial sorting networks. For a set of channels A , a vector is an element $v \in \{0, 1\}^A$, e_c denotes the vector with a single 1 on channel c , and $\mathbf{1}$ the vector of all 1s. A network T on A *sorts* a set $X \subseteq \{0, 1\}^A$ if its outputs can be listed as $o_0, \dots, o_{|A|-1}$ so that, for each input

vector in X , the output values read in this order are nondecreasing. Let $s(X)$ be the least size of a network that sorts X . These are Harder's partial sorting networks [5, Definition 1]: his networks may contain exchanges of channels, which the circuit model absorbs into the wiring. A permutation π of the channels acts on vectors by moving coordinates, and $\bar{X} = \{\mathbf{1} - v : v \in X\}$ is the complement of X . Sets related by a permutation of the channels, possibly combined with complementation, are *similar*.

Lemma 2.2. (a) $s(\{0, 1\}^n) = S(n)$.
 (b) If $X \subseteq X'$ then $s(X) \leq s(X')$.
 (c) Similar sets have the same value of s .

Proof. (a) A network sorts every input sequence if and only if it sorts all 0-1 sequences [8, Section 5.3.4], and a network in the circuit model that sorts can be untangled into a standard sorting network of the same size [8, Exercise 5.3.4–16]. (b) and (c) are Lemma 2 and Corollary 6 of [5]. $\square$

A standard comparator (i, j) maps a set X to the set $X(i, j)$ of its images. It is *useful* for X if X contains a vector v with $(v_i, v_j) = (0, 1)$ and a vector with $(v_i, v_j) = (1, 0)$. If no vector of X has $(v_i, v_j) = (0, 1)$, the comparator leaves X unchanged. If some vector does but none has $(1, 0)$, it acts on X as the exchange of channels i and j .

2.3. Pruning and cores.

Definition 2.3. Let $X \subseteq \{0, 1\}^A$ and $\epsilon \in \{0, 1\}$, the *polarity*. A channel $c \in A$ is *extremal* in X for polarity 0 if $e_c \in X$, and for polarity 1 if $\mathbf{1} - e_c \in X$. For polarity 0 and an extremal channel c , let R be the set of vectors $u \in X$ that can be reached from e_c by a sequence of vectors of X in which each vector arises from the previous one by changing one coordinate from 0 to 1. The *pruned set* $X/c \subseteq \{0, 1\}^{A \setminus \{c\}}$ consists of the restrictions of the vectors of R to $A \setminus \{c\}$. For polarity 1, X/c is the complement of $\bar{X}/c$, the latter taken for polarity 0.

The vectors of R have a 1 on channel c , so X/c is contained in Harder's pruned set $\{u|_{A \setminus \{c\}} : u \in X, u_c = 1\}$ [5, Definition 18].

Lemma 2.4 (pruning a channel). *Let T sort $X \subseteq \{0, 1\}^A$ with outputs $o_0, \dots, o_{|A|-1}$, and let c be extremal in X for polarity 0. Then Π_c ends on $o_{|A|-1}$, and T/c sorts X/c with its outputs listed as $o_0, \dots, o_{|A|-2}$.*

Proof. On input e_c the single 1 follows Π_c , and since the output is sorted, Π_c ends on $o_{|A|-1}$. Harder proves that T/c sorts his pruned set [5, Lemma 20], which contains X/c . $\square$

A *maximal chain* of $\{0, 1\}^A$ is a sequence $u_0, \dots, u_{|A|}$ that starts at the zero vector and in which each vector arises from the previous one by changing one coordinate from 0 to 1. Harder calls a set *well-behaved* if it is a union of maximal chains [5, Definitions 27 and 28]. His program computes $s(X)$ for well-behaved sets [5, Corollary 56].

Lemma 2.5 (well-behaved sets). *Let $X \subseteq \{0, 1\}^A$ be well-behaved and let c be extremal in X for polarity 0. Then X/c is the union of the maximal chains of $\{0, 1\}^{A \setminus \{c\}}$ that are contained in Harder's pruned set. In particular, X/c is well-behaved. For polarity 1, too, X/c is well-behaved. Consequently, the well-behaved sets are closed under comparators, similarity and pruning an extremal channel of either polarity, and every set obtained from $\{0, 1\}^n$ by these operations is well-behaved.*

Proof. Let $u \in X/c$ be the restriction of $\bar{u} \in R$. Some maximal chain of $\{0, 1\}^A$ inside X passes through $\bar{u}$. Its part above $\bar{u}$, preceded by a sequence from e_c to $\bar{u}$ as in Definition 2.3, runs from

e_c to $\mathbf{1}$ inside X , and its restriction to $A \setminus \{c\}$ is a maximal chain through u inside Harder's pruned set. Conversely, setting coordinate c to 1 in the vectors of a maximal chain of $\{0, 1\}^{A \setminus \{c\}}$ inside Harder's pruned set gives a sequence in X from e_c as in Definition 2.3, so the chain lies in X/c . Polarity 1 follows by complementation. The set $\{0, 1\}^n$ is well-behaved, and comparators, permutations of the channels and complementation map well-behaved sets to well-behaved sets [5, Lemma 35 and Corollary 36]. $\square$

Definition 2.6. Let $X \subseteq \{0, 1\}^n$ and let ϵ be a polarity. A *scenario* of (X, ϵ) is a pair (x, y) of channels such that x is extremal in X and y is extremal in X/x , both for polarity ϵ . Its *core* is the set $Y_{xy} = (X/x)/y \subseteq \{0, 1\}^{[n] \setminus \{x, y\}}$, where $[n] = \{0, \dots, n-1\}$.

Remark 2.7. A channel is extremal in X for polarity 1 exactly when it is extremal in $\bar{X}$ for polarity 0, and X/c for polarity 1 is the complement of $\bar{X}/c$ for polarity 0. Hence the scenarios of $(X, 1)$ are those of $(\bar{X}, 0)$, and their cores are the complements of the cores of $(\bar{X}, 0)$. By Lemma 2.2(c) they have the same values of s .

Lemma 2.8 (two values). *Let T sort $X \subseteq \{0, 1\}^n$ with outputs $o_0, \dots, o_{n-1}$, and let (x, y) be a scenario of $(X, 0)$. In the run (x, y) of T , the value M ends on o_{n-1} and m ends on o_{n-2} , and $|T| - |W(x, y)| \geq s(Y_{xy})$.*

Proof. By Lemma 2.4, Π_x ends on o_{n-1} and T/x sorts X/x with outputs $o_0, \dots, o_{n-2}$. Since y is extremal in X/x , the same lemma applied to T/x shows that the path of the maximum of T/x from y ends on o_{n-2} and that $(T/x)/y$ sorts Y_{xy} . By Lemma 2.1 this path is the path of m , and $(T/x)/y$ has $|T| - |W(x, y)|$ comparators. $\square$

2.4. The trivial bound. The bound of the next lemma uses only the comparators that produce X .

Lemma 2.9 (trivial bound). *Let P be a sequence of k comparators on n channels, and let X be the set of outputs of P on $\{0, 1\}^n$. For channels $i \neq j$, let x and y be the channels on which M and m end in the run (i, j) of P , and let $w = |W(i, j)|$. Then (x, y) is a scenario of $(X, 0)$, and*

$$s(Y_{xy}) \geq S(n-2) - (k-w).$$

The same holds for polarity 1 with the run in which M is the smallest value and m the second smallest.

Proof. Let $P_{ij} = (P/i)/j$. By Lemma 2.1 it has $k-w$ comparators. Its input channels are $[n] \setminus \{i, j\}$, and its outputs are the channels other than x and y . On an input vector with a 1 on channel i , the leads of Π_i carry 1, because comparator networks are monotone. Hence P and P/i agree on the remaining leads [5, proof of Lemma 16]. Applying the same argument to P/i and channel j shows, for every $z \in \{0, 1\}^{[n] \setminus \{i, j\}}$, that $P(e_i + e_j + z)$ has 1s on x and y and agrees with $P_{ij}(z)$ on the other channels.

Since the map P is monotone and preserves the number of 1s, a vector u' that arises from u by changing one coordinate from 0 to 1 is mapped to a vector that arises from $P(u)$ in the same way. Starting from e_i , adding the 1 on j and then the 1s of z one at a time gives vectors whose images form a sequence in X from $P(e_i) = e_x$ through $P(e_i + e_j) = e_x + e_y$ to $P(e_i + e_j + z)$, each arising from the previous one by changing one coordinate from 0 to 1. Hence x is extremal in X , y is extremal in X/x , and $P_{ij}(z) \in Y_{xy}$ for all z . If a network N sorts Y_{xy} , then P_{ij} followed by N sorts $\{0, 1\}^{[n] \setminus \{i, j\}}$, so $s(Y_{xy}) + k - w \geq S(n-2)$ by Lemma 2.2(a). For polarity 1, apply this to the sequence $\bar{P}$ obtained from P by replacing each comparator (a, b) with (b, a) . Since it maps $\mathbf{1} - u$ to $\mathbf{1} - P(u)$, its set of outputs is $\bar{X}$, and its run with the largest values on i and j is the run of P with the smallest values there. By Remark 2.7, the scenarios and cores of $(\bar{X}, 0)$ give those of $(X, 1)$. $\square$

3. THE WEIGHTED TWO-VALUE BOUND

Definition 3.1. A *weighted instance* I on a set of n channels consists of a budget $L \geq 0$ and a finite list of *scenarios* (x, y, v) , where $x \neq y$ are channels and $v \geq 0$ is an integer, the *weight*. Let E be the set of pairs (x, y) that occur in scenarios, and for $(x, y) \in E$ let $V(x, y)$ be the largest weight of a scenario on (x, y) and $b(x, y) = L - V(x, y)$. The instance is *feasible* if there are a comparator network T on the n channels and two distinct outputs o_M and o_m of T such that, for each $(x, y) \in E$, the run (x, y) of T ends with M on o_M and m on o_m , and $|W(x, y)| \leq b(x, y)$.

For a set $X \subseteq \{0, 1\}^n$, a polarity ϵ , integers v_t indexed by the scenarios (x_t, y_t) of (X, ϵ) , and a budget L , let $I(X, \epsilon; v; L)$ denote the weighted instance with budget L and scenarios (x_t, y_t, v_t) .

Proposition 3.2 (weighted two-value bound). *Let $X \subseteq \{0, 1\}^n$, let ϵ be a polarity, and let v_t be integers with $0 \leq v_t \leq s(Y_t)$ for each scenario (x_t, y_t) of (X, ϵ) with core Y_t . If $s(X) \leq L$, then $I(X, \epsilon; v; L)$ is feasible. Hence, if $I(X, \epsilon; v; L)$ is infeasible, then $s(X) \geq L + 1$.*

Proof. For polarity 0, let T sort X with $s(X)$ comparators and outputs $o_0, \dots, o_{n-1}$. By Lemma 2.8, each run (x_t, y_t) ends with M on o_{n-1} and m on o_{n-2} , and $|W(x_t, y_t)| \leq s(X) - s(Y_t) \leq L - v_t$. With $o_M = o_{n-1}$ and $o_m = o_{n-2}$, the network T shows that the instance is feasible. For polarity 1, Remark 2.7 shows that $I(X, 1; v; L) = I(\bar{X}, 0; v; L)$ and that the cores of $(\bar{X}, 0)$ have the same values of s , and $s(\bar{X}) = s(X)$. $\square$

For $X = \{0, 1\}^n$ and polarity 0, every pair of distinct channels is a scenario, and every core is the full set on the other $n - 2$ channels. With all weights equal to $S(n - 2)$, the instance with budget L is feasible exactly when some top-two correct network has $p_2 \leq L - S(n - 2)$, in the notation of [11, Definition 2.1]. The weighted bound thus generalizes the two-value bound $S(n) \geq S(n - 2) + P(2, n)$ of [11, Lemma 2.2]. In our application X is the set produced by a prefix of a network, the weights are lower bounds on the costs of the cores (Section 6), and different scenarios have different budgets.

4. FROM FEASIBLE NETWORKS TO CONFIGURATIONS

In this section we show that a feasible weighted instance has a *valid configuration*, a finite object that the search of Section 5 enumerates. The argument extends [11, Section 3], where every pair of inputs is a scenario, to an arbitrary set of scenarios.

Throughout, I is a weighted instance with $E \neq \emptyset$, and T with outputs o_M and o_m shows that I is feasible. Let $P' = \{x : (x, y) \in E\}$. The channels $y \notin P'$ with $(x, y) \in E$ for some x are the *blue starts*. For a channel y let $E^y = \{x : (x, y) \in E\}$, and for $x \in P'$ let $E_x = \{y : (x, y) \in E\}$.

4.1. The tree of maximum paths. A lead of T is *red* if it lies in Π_x for some $x \in P'$, and *blue* otherwise.

Lemma 4.1. (a) *The input lead of a channel x is red exactly when $x \in P'$. Each lo output is blue, and $\text{hi}(g)$ is red if and only if g has a red input lead.*

(b) *Join each red lead $\ell \neq o_M$ to $\text{hi}(g)$, where g is the comparator that ℓ enters. Then the red leads form a rooted tree with root o_M whose leaves are the input leads of the channels in P' .*

(c) *Call a comparator a branch node if both of its input leads are red, a pass-through if exactly one is, and blue if neither is. The blue input lead of a pass-through is its port. There are exactly $|P'| - 1$ branch nodes. Contracting the vertices with one child in the tree of (b) gives a binary tree B whose leaves are the channels of P' and whose internal vertices are the branch nodes. Each pass-through lies on an edge of B or on the edge from the root of B to o_M .*

Proof. The proof of [11, Lemma 3.1] applies to the paths Π_x with $x \in P'$. Each of them ends on o_M , because some scenario has M on x . The input lead of a channel outside P' lies on none of them, since a path Π_x starts at the input lead of x and input leads are outputs of no comparator. $\square$

For a vertex u of B , $\text{seg}(u)$ denotes the edge from u to its parent, or from u to o_M if u is the root, and $k(u)$ denotes the number of pass-throughs on $\text{seg}(u)$. We number them $(u, 1), \dots, (u, k(u))$ from below. The branch nodes and the pass-throughs are the *red comparators*. For a red comparator v , let $u(v) = v$ if v is a branch node and $u(v) = u$ if $v = (u, i)$, and let $A(v)$ be the set of red comparators above v on the path to o_M , with v excluded. For $x \in P'$, the red comparators on the path from x to o_M are the comparators that M enters in the runs from x , so

$$\delta(x) = k(x) + \sum_u (1 + k(u)),$$

where u runs over the proper ancestors of x in B . For a branch node c and a leaf y below c , let $\text{pre}(y, c) = k(y) + \sum_u (1 + k(u))$ over the vertices u strictly between y and c . For a pass-through $q = (w, i)$ and a branch node e that is a proper ancestor of w , let

$$\text{cl}(q, e) = 1 + (k(w) - i) + \sum_z (1 + k(z)),$$

where z runs over the vertices strictly between w and e . This is the number of red comparators from q up to e , with e excluded. Note that $q = (w, i)$ lies on the path from $x \in P'$ to o_M exactly when w is x or an ancestor of x .

4.2. The runs. For a blue lead ℓ , the *walk* from ℓ starts at ℓ and, while the current lead enters a blue comparator g , moves to $\text{hi}(g)$. It stops at a lead $\rho(\ell)$ that is an output or enters a comparator that is not blue, and $d(\ell)$ denotes the number of blue comparators that it passes. Since a blue lead does not enter a branch node, $\rho(\ell)$ is the port of a pass-through whenever it is not an output.

Lemma 4.2. *Suppose that m is on a blue lead ℓ in the run (x, y) , where $(x, y) \in E$. Then m follows the walk from ℓ , none of the $d(\ell)$ blue comparators on it is a meeting, and one of the following holds.*

- (i) $\rho(\ell)$ is an output. Then $\rho(\ell) = o_m$, and m ends there.
- (ii) $\rho(\ell)$ is the port of a pass-through q that lies on Π_x . Then q is a meeting, and m leaves q on $\text{lo}(q)$.
- (iii) $\rho(\ell)$ is the port of a pass-through $q = (w, i)$ that does not lie on Π_x . Then $e = \text{lca}_B(w, x)$ is a proper ancestor of w , m passes the $\text{cl}(q, e)$ red comparators from q up to e , none of which is a meeting, meets M at e , and leaves e on $\text{lo}(e)$.

Proof. The other input lead of a blue comparator is blue and does not carry M . The value m therefore leaves on hi , which is blue by Lemma 4.1(a), and it follows the walk. In case (i), m ends on $\rho(\ell)$, which is o_m because $(x, y) \in E$. In case (ii), M is on the red input lead of q . In case (iii), M is not at q . Then m leaves on the red lead $\text{hi}(q)$, and w is neither x nor an ancestor of x . From then on m climbs the tree of Lemma 4.1(b). At a pass-through on the way, the other input lead is the port, which never carries M . At a branch node z strictly between w and e , the other input lead comes from the subtree of z that does not contain w . Since z is not an ancestor of x , that subtree does not contain x , and the lead does not carry M . Hence m leaves each of these comparators on hi . At e the value M arrives from the side of x and m from the side of w , and m leaves on $\text{lo}(e)$. $\square$

Lemma 4.3. *Let $(x, y) \in E$.*

- (a) If $y \in P'$ and $c = \text{lca}_B(x, y)$, then m passes the $\text{pre}(y, c)$ red comparators below c on the path from y , none of which is a meeting, meets M at c and leaves c on $\text{lo}(c)$.
- (b) If m leaves a red comparator v on $\text{lo}(v)$, then from there on, the comparators that m enters, and which of them are meetings, depend only on v . We write $\tau(v)$ for the number of those comparators that are not meetings.
- (c) $|W(x, y)| = \delta(x) + \text{pre}(y, c) + \tau(c)$ if $y \in P'$ and $c = \text{lca}_B(x, y)$, and $|W(x, y)| = \delta(x) + r_x(y)$ if y is a blue start, where $r_x(y)$ is the number of comparators that m enters in the run (x, y) and that are not meetings.

Proof. (a) The proof of [11, Lemma 3.3(a)] applies. (b) After v the value M passes the comparators of $A(v)$. The comparators of Π_x below v come before v in the circuit, and m enters only comparators after v . Whether m leaves a comparator on lo therefore depends only on its current lead and on $A(v)$. (c) follows from (a), (b) and Lemma 2.1. $\square$

For a red comparator v that m leaves on $\text{lo}(v)$ in some run of a scenario, we call v *live*. Then $\text{lo}(v)$ is blue, and Lemma 4.2 applies to it. If $\rho(\text{lo}(v))$ is the port of a pass-through q , then q comes after v in the circuit. The comparators of Π_x that come after v are those of $A(v)$, so q lies on Π_x exactly when $q \in A(v)$. If $q \notin A(v)$, then $u(q)$ is not $u(v)$ or a descendant of $u(v)$: otherwise q would lie in $A(v)$, be v or come before v . As x is $u(v)$ or lies below it, the vertex $\text{lca}_B(u(q), x)$ of Lemma 4.2(iii) is then $\text{lca}_B(u(q), u(v))$, a proper ancestor of $u(v)$. This gives the recursion

$$(1) \quad \tau(v) = \begin{cases} d(v) & \text{if } \rho(v) = o_m, \\ d(v) + \tau(q) & \text{if } \rho(v) \text{ is the port of } q \in A(v), \\ d(v) + \text{cl}(q, e) + \tau(e) & \text{if } \rho(v) \text{ is the port of } q \notin A(v), \text{ with } e = \text{lca}_B(u(q), u(v)), \end{cases}$$

where $\rho(v) = \rho(\text{lo}(v))$ and $d(v) = d(\text{lo}(v))$. For a blue start y and $x \in E^y$, Lemma 4.2 applied to the input lead y gives $r_x(y) = d(y)$ if $\rho(y) = o_m$; $r_x(y) = d(y) + \tau(q)$ if $\rho(y)$ is the port of $q = (w, i)$ and x lies below w or equals it; and $r_x(y) = d(y) + \text{cl}(q, e) + \tau(e)$ with $e = \text{lca}_B(w, x)$ otherwise. Here $\rho(y) = \rho(\ell)$ and $d(y) = d(\ell)$ for the input lead ℓ of y .

4.3. Three properties of feasible networks. A pass-through q has a *climbing run* if, in the run of some scenario, m enters the port of q while M is not at q .

Lemma 4.4. *If I is feasible, then some network T with outputs o_M and o_m shows this and has a climbing run for each of its pass-throughs. In such a network no pass-through lies on $\text{seg}(\text{root of } B)$.*

Proof. Let q be a pass-through without a climbing run, with red input lead a and port b . Delete q , and join a to $\text{hi}(q)$ and b to $\text{lo}(q)$. In the run of a scenario, if M is at q then it is on a and leaves on $\text{hi}(q)$. If m enters q through b , then M is at q , because there is no climbing run, and m leaves on $\text{lo}(q)$. If m enters on a , then M is not at q , since M is not on a and never on the blue lead b , and m leaves on $\text{hi}(q)$. In every scenario run, M and m therefore follow the same leads as before, with q skipped, and the new network shows that I is feasible. Since the runs are the same, the other comparators keep their kinds and the remaining pass-throughs keep their climbing runs. Repeating this ends, since each step removes a comparator. A pass-through on the segment of the root has no climbing run, because every path Π_x with $x \in P'$ passes it. $\square$

The *sources* are the leads $\text{lo}(v)$ of the live red comparators v and the input leads of the blue starts.

Lemma 4.5 (Kraft). *For each lead r , the sources s with $\rho(s) = r$ satisfy $\sum_s 2^{-d(s)} \leq 1$.*

Proof. Each lead on a walk, except its first lead, is $\text{hi}(g)$ for a blue comparator g . A source is the lo output of a red comparator or an input lead, so it lies on no walk except its own, as the first lead. The walks to r form a tree with root r whose leaves are the sources s with $\rho(s) = r$, and the depth of s in it is $d(s)$. A vertex $\text{hi}(g)$ of this tree has at most two children, the input leads of g . Give the root weight 1, and let each vertex pass its weight to its child or split it equally between its two children. The weight of a leaf is $2^{-j} \geq 2^{-d(s)}$, where j is the number of vertices with two children above it, and the weights of the leaves sum to 1. $\square$

Lemma 4.6. *If each pass-through has a climbing run, then each port is $\rho(s)$ for some source s .*

Proof. Let q be a pass-through and consider a climbing run of q . Before m enters the port of q , it is on blue leads. Let s be the last lead before the port at which m either started or arrived from a red comparator v on $\text{lo}(v)$. In the first case s is the input lead of a blue start, because input leads of P' are red. In the second case v is live. In both cases s is a source. After s , the value m passes only blue comparators until it reaches the port: it never enters a pass-through through its port on the way, since a meeting there would give a later lo output, and so would the meeting that ends a climb (Lemma 4.2). The walk from s thus ends at the port of q . $\square$

Lemma 4.7 (cap). *For each $x \in P'$, $2^{\delta(x)} \sum_{y \in E_x} 2^{-b(x,y)} \leq 1$.*

Proof. Fix x . In the runs (x, y) with $y \in E_x$ the value M follows Π_x , and the next lead of m depends only on its current lead. The leads of m in these runs therefore form a tree with root o_m whose leaves are the input leads of the channels in E_x , since an input lead is the output of no comparator. Call a comparator *binary* if both of its input leads lie in this tree. A binary comparator does not lie on Π_x , since one of its input leads would then carry M in all these runs. In particular, it is not a meeting. With weights passed down from the root as in the proof of Lemma 4.5, the leaf y receives 2^{-j_y} , where j_y is the number of binary comparators that m enters in the run (x, y) , and these weights sum to 1. By Lemma 2.1, $j_y \leq |W(x, y)| - \delta(x) \leq b(x, y) - \delta(x)$. Hence $\sum_y 2^{-(b(x,y)-\delta(x))} \leq 1$. $\square$

4.4. Configurations.

Definition 4.8. A *configuration* Γ for I consists of

- (i) a binary tree B whose leaves are the channels of P' , each once, with an edge $\text{seg}(\text{root})$ from its root to o_M ;
- (ii) numbers $k(u) \geq 0$ for the vertices u of B , with $k(\text{root}) = 0$; the *red nodes* are the internal vertices of B , called *branch nodes*, and the pass-throughs (u, i) with $1 \leq i \leq k(u)$;
- (iii) for each red node v , either the mark **dead** or a *source* $(\rho(v), d(v))$ consisting of a *root* $\rho(v)$, which is o_m or a pass-through, and a *depth* $d(v) \geq 0$; the red nodes that are not **dead** are *live*;
- (iv) for each blue start y , a source $(\rho(y), d(y))$.

The quantities δ , pre , cl , $u(\cdot)$ and $A(\cdot)$ are defined from (B, k) as above. A root that is a pass-through q stands for the port of q . For a live red node v whose root is a pass-through q , the *successor* of v is q if $q \in A(v)$, and $\text{lca}_B(u(q), u(v))$ otherwise. For a blue start y whose root is a pass-through $q = (w, i)$ and for $x \in E^y$, the successor of y for x is q if x is w or lies below w , and $\text{lca}_B(w, x)$ otherwise. The values $\tau(v)$ of the live red nodes are defined by the recursion (1), and the values $r_x(y)$ of the blue starts by the formulas that follow it; a value is ∞ if the recursion reaches a **dead** node. The *LCA nodes* are the branch nodes $\text{lca}_B(x, y)$ with $(x, y) \in E$ and $y \in P'$. For an LCA node c let

$$\beta(c) = \min\{b(x, y) - \delta(x) - \text{pre}(y, c) : (x, y) \in E, y \in P', \text{lca}_B(x, y) = c\}.$$

Definition 4.9. A configuration is *valid* if it satisfies the following conditions.

- (V1) For each root r , the sources with root r , of live red nodes and of blue starts, satisfy $\sum 2^{-d} \leq 1$.
- (V2) Each pass-through is the root of some source.
- (V3) The directed graph on the red nodes with an arc from each red node to the next red node above it, and an arc from each live red node v to its root whenever that root is a pass-through, is acyclic.
- (V4) Each LCA node is live, the successor of each live red node is live, and the successor of each blue start for each $x \in E^y$ is live.
- (V5) $\tau(c) \leq \beta(c)$ for each LCA node c , and $\delta(x) + r_x(y) \leq b(x, y)$ for each blue start y and each $x \in E^y$.
- (V6) $2^{\delta(x)} \sum_{y \in E_x} 2^{-b(x, y)} \leq 1$ for each $x \in P'$.

By (V3) the successor of a live red node v lies in $A(v)$. If the root q of v is not in $A(v)$, then q is neither v nor below v , since the arc from v to q would close a cycle with the path from q up to v . As in the paragraph before (1), $u(q)$ is then not $u(v)$ or a descendant of $u(v)$, so $\text{lca}_B(u(q), u(v))$ is a proper ancestor of $u(v)$. Each step of the recursion therefore moves strictly up, and τ is well defined. For $(x, y) \in E$ the *cost* is $\delta(x) + \text{pre}(y, c) + \tau(c)$ with $c = \text{lca}_B(x, y)$ if $y \in P'$, and $\delta(x) + r_x(y)$ if y is a blue start. Condition (V5) says that every cost is at most $b(x, y)$.

Theorem 4.10 (soundness). *If I is feasible and $E \neq \emptyset$, then I has a valid configuration.*

Proof. By Lemma 4.4 we may assume that every pass-through of T has a climbing run, and then $k(\text{root}) = 0$. Lemma 4.1 gives B and the numbers $k(u)$. Give each live red comparator v the source $(\rho(\text{lo}(v)), d(\text{lo}(v)))$, mark the other red comparators **dead**, and give each blue start y the source $(\rho(\ell), d(\ell))$ of its input lead ℓ . By Lemma 4.2(i) and the discussion before (1), each root is o_m or the port of a pass-through, and the successors in the configuration are the comparators at which m meets M next in the network. These are live, which gives (V4) together with Lemma 4.3(a). The values τ and r_x of the configuration are those of the network, by (1). By Lemma 4.3(c) the costs are then the numbers $|W(x, y)|$, and (V5) holds because T shows that I is feasible. (V1), (V2) and (V6) are Lemmas 4.5, 4.6 and 4.7. Each arc of the graph of (V3) is a directed path in the circuit: from a red comparator to the next one above it, or from v through $\text{lo}(v)$ and the blue comparators of its walk into the pass-through. A cycle in this graph would give a cycle in the circuit. $\square$

Remark 4.11. Conversely, every valid configuration is realized by a network whose runs cost at most the costs of the configuration: build the red tree with its pass-throughs, and for each root a tree of blue comparators on the lo outputs of its sources, at depths at most the given ones, by Kraft's inequality [11, Lemma 2.4(c)]. For budgets at most 56, the program of Section 5 uses this construction to confirm each feasible answer by simulating every run. No part of the proof of Theorem 1.1 uses this converse.

5. DECIDING WEIGHTED INSTANCES

The program **lb2cr** decides a weighted instance by searching for a valid configuration. If $E = \emptyset$ it answers that the instance is feasible, and if some $b(x, y)$ is negative it answers that it is infeasible. Otherwise it searches the configurations in a normal form, discards partial data that violate necessary conditions, and answers *infeasible* only if the search finishes without finding one. By Theorem 4.10, the instance is then infeasible. A third answer, *unknown*, is given when a limit on the work is reached; the verification of Section 8 treats it as a failure. This section states

the normal form and the necessary conditions, proves each of them, and describes the search. We fix an instance I with $E \neq \emptyset$ and $b(x, y) \geq 0$ for all $(x, y) \in E$.

5.1. Normal forms. A pass-through $q = (w, i)$ is *readable* if some red node lies below it on the path to o_M , or some blue start y has an $x \in E^y$ that is w or lies below w . A branch node e is readable if some pass-through lies on the segment of a proper descendant of e . By the recursion (1) and the formulas for $r_x(y)$, the value $\tau(q)$ of a pass-through enters only the values of sources that meet at q , which lie below q , and of blue starts with x equal to w or below w , while $\tau(e)$ of a branch node enters only the values of sources whose root is a pass-through on the segment of a proper descendant of e . Only readable nodes can thus be successors. For a source s whose successor is s' , we call s a *predecessor* of s' ; a blue start y is a predecessor of each of its successors for the $x \in E^y$. Let

$$B_{\max} = \max\left(\{\beta(c) : c \text{ an LCA node}\} \cup \{b(x, y) - \delta(x) : y \text{ a blue start, } x \in E^y\}\right).$$

A configuration is in *normal form* if, besides being valid, it satisfies:

- (N3) each live red node that is not readable is an LCA node, and an LCA node c that is not readable has the largest depth for which $\tau(c) \leq \beta(c)$, given its root;
- (N4) $\tau(v) \leq B_{\max}$ for each live red node v that is not an LCA node;
- (N5) each blue start y has the largest depth for which $\delta(x) + r_x(y) \leq b(x, y)$ for all $x \in E^y$, given its root.

The numbering follows (N1), $k(\text{root}) = 0$, and (N2), which is (V2).

Lemma 5.1 (deletion). *Let Γ be a configuration that satisfies all conditions of validity except possibly (V2). Then deleting, one at a time, pass-throughs that are the root of no source gives a valid configuration. If Γ violates (V2), the result has fewer pass-throughs than Γ .*

Proof. Let $q = (w, i)$ be the root of no source, and delete it: $k(w)$ drops by one and the pass-throughs above q on $\text{seg}(w)$ move down one place. No successor is q , since a pass-through is a successor only of sources whose root it is. Since membership in $A(\cdot)$ is unchanged for the other red nodes and B is unchanged, every other successor stays the same. The numbers δ , pre and cl count red nodes on paths and do not increase. By induction along the recursion, no value τ or r_x increases, and no $\beta(c)$ decreases. Conditions (V4), (V5) and (V6) therefore still hold. The source of q , if any, leaves the sum at its root, so (V1) holds. A path in the new graph of (V3) comes from a path in the old one, so (V3) holds. Only (V2) can fail, at the root of q , and then we repeat. This ends, since each step removes a pass-through. $\square$

The formal configurations of Section 11 also require that every live red node can be reached from an LCA node or a blue start by a chain of successors. Deletions keep this condition: the deleted pass-through is the successor of no source, so it is not reached and hence **dead**, and no other successor changes.

Lemma 5.2 (normal form). *Let Γ be a valid configuration whose number of pass-throughs is the least among all valid configurations for I . Then some configuration with the same B and k is in normal form.*

Proof. Each step below keeps (B, k) , only marks nodes **dead** or increases depths, and changes the values τ and r_x only where stated. This keeps (V1), (V3) and (V6). If it left a pass-through that is the root of no source, Lemma 5.1 would give a valid configuration with fewer pass-throughs, contrary to the choice of Γ . Condition (V2) is kept as well, and we check (V4) and (V5) at each step.

(N4) Let Z be the set of live red nodes v that are not LCA nodes and have $\tau(v) > B_{\max}$. A predecessor s of a node $v \in Z$ has $\tau(s) \geq \tau(v)$, or $r_x(s) \geq \tau(v)$ if s is a blue start. It is not an LCA

node, since those have $\tau \leq \beta \leq B_{\max}$, and not a blue start, since $r_x(y) \leq b(x, y) - \delta(x) \leq B_{\max}$. Thus Z contains every predecessor of its members. Mark the nodes of Z **dead**. No live node outside Z has its successor in Z , so (V4) holds, and no value outside Z changes, so (V5) holds.

(N3) A live red node v that is not readable is the successor of no source. If it is not an LCA node, mark it **dead**: no other value changes, and (V4) holds. If it is an LCA node, its value $\tau(v)$ enters only the condition $\tau(v) \leq \beta(v)$ of (V5), and we raise its depth as far as that condition allows. This changes no other value. These changes do not affect (N4).

(N5) Since no source has a blue start as its successor, raising the depth of a blue start changes only its own values $r_x(y)$. We raise each depth as far as (V5) allows. $\square$

Lemma 5.3. *Let Γ be as in Lemma 5.2. Then each pass-through $q = (w, i)$ of Γ has an affordable climber: an LCA node c such that q neither lies on the path from c to o_M nor below c , and $\beta(c) \geq \text{cl}(q, \text{lca}_B(w, c))$; or a blue start y and an $x \in E^y$ that is not w and not below w , with $b(x, y) - \delta(x) \geq \text{cl}(q, \text{lca}_B(w, x))$.*

Proof. A source with root q climbs through q if its successor is not q ; for a blue start this means that its successor for some $x \in E^y$ is a branch node. First, some source climbs through q . Otherwise every source s with root q has successor q , and q is live by (V4). Give each such s the root $\rho(q)$ and the depth $d(s) + d(q)$. If $\rho(q)$ is o_m or a pass-through above q , then s meets there as q did. If $\rho(q)$ is a pass-through q' not above q , then q climbs to $e = \text{lca}_B(u(q'), w)$, and since s lies below q , or x lies below w for a blue start, the climb from q' also ends at e for s . In each case the value of s is unchanged. The sum at $\rho(q)$ changes by $2^{-d(q)}(\mu - 1) \leq 0$, where $\mu \leq 1$ is the sum at q . Each new arc of (V3) replaces a path through q . Then mark q **dead**. Now q is the root of no source, and Lemma 5.1 gives a valid configuration with fewer pass-throughs, a contradiction.

Let U be the set of sources that climb through q , and let Z be the set of sources from which a chain of successors reaches a member of U . Suppose that Z contains an LCA node or a blue start c , whose chain reaches $t \in U$. If t is a red node, then $u(q) = w$ is not $u(t)$ or a descendant of $u(t)$ (see the remarks after Definition 4.9). If c is an LCA node, then c is t or lies below t , so the climb from q ends at the same branch node e for c as for t . Moreover q does not lie on the path from c to o_M : it would then lie above t , so that t meets at q , or at or below t , which (V3) excludes. Nor does q lie below c , since it would then lie below t . If $c = y$ is a blue start, choose $x \in E^y$ as follows: if $t = y$, an x for which the successor of y is a branch node; otherwise an x to which the chain from y to t belongs. In the second case every node of the chain lies on the path from x to o_M , so x lies in the subtree of $u(t)$, and w is not x and not an ancestor of x , since q would otherwise lie in $A(t)$. In both cases x is not w and does not lie below w , and the climb from q ends at the same branch node e for c as for t . In all cases the value of c is at least that of t , which is at least $\text{cl}(q, e)$, and the budget of c is at least its value. Hence c is an affordable climber. Otherwise Z contains no LCA node and no blue start. Mark the nodes of Z **dead**. No live node outside Z has its successor in Z , so (V4) holds, and no value outside Z changes. If a pass-through became the root of no source, Lemma 5.1 would contradict the choice of Γ . The result is therefore valid, with the same (B, k) , and no source climbs through q , which the first part excludes. $\square$

5.2. Necessary conditions on the tree and the pass-throughs. For $x \in P'$ let A_x be the largest integer a with $2^a \sum_{y \in E_x} 2^{-b(x, y)} \leq 1$. Let $D(u)$ denote the depth of a vertex u of B , the root having depth 0.

Lemma 5.4. *Every valid configuration satisfies the following.*

- (a) $D(x) \leq \delta(x) \leq A_x$ for each $x \in P'$. Moreover, if numbers $c(x) \geq D(x)$ are given for the leaves x of a subtree of B whose root has depth t , then $\sum_x 2^{-(c(x)-t)} \leq 1$ over these leaves, and $c(x) - t \geq 1$ if the subtree has at least two leaves.
- (b) $\delta(x) + \text{pre}(y, c) \leq b(x, y)$ for each $(x, y) \in E$ with $y \in P'$ and $c = \text{lca}_B(x, y)$. In particular $D(x) + D(y) \leq b(x, y) + D(c) + 1$.
- (c) $\beta(c) \geq 0$ for each LCA node c , and $\delta(x) \leq b(x, y)$ for each blue start y and $x \in E^y$.

Proof. (a) Each proper ancestor of x contributes at least 1 to $\delta(x)$, and (V6) gives $\delta(x) \leq A_x$. The leaves of a subtree satisfy Kraft's equality $\sum 2^{-(D(x)-t)} = 1$ [11, Lemma 2.4(a)], and $D(x) - t \geq 1$ if there are at least two of them. (b) and (c) follow from (V5), since τ and r_x are nonnegative, and from $\text{pre}(y, c) \geq D(y) - D(c) - 1$. $\square$

Since $\delta(x) = D(x) + \sum_u k(u)$ over x and its proper ancestors, these conditions are linear inequalities in the numbers $k(u)$, with coefficients 0 and 1. For a segment w , let $k_{\max}(w)$ be the least right-hand side among the inequalities of (a) and (b) that contain $k(w)$.

5.3. Credit bounds. The following bounds compare the budgets with what the forest of sources can carry. Fix a valid configuration. Give each live red node v the *credit* $\kappa(v) = 2^{-\tau(v)}$. For a blue start y fix $x^*(y) \in E^y$ with the least value of $b(x, y) - \delta(x)$, put $\beta(y) = b(x^*(y), y) - \delta(x^*(y))$, and let $\kappa(y) = 2^{-\tau_{x^*(y)}(y)}$. The *parent* of a live red node is o_m if its root is o_m , and its successor otherwise. The parent of a blue start y is defined in the same way, with its successor for $x^*(y)$. Parents are live by (V4), and they lie above their children on the path to o_M , so the parents form a tree with root o_m , the *flow tree*. Put $\kappa(o_m) = 1$. By (1), each child s of a node p satisfies

$$(2) \quad \kappa(s) = \kappa(p) 2^{-d(s)-\gamma(s)},$$

where $\gamma(s) = \text{cl}(q, p)$ if s climbs to p through its root q , and $\gamma(s) = 0$ otherwise. The LCA nodes and the blue starts are *required*. They are live, and their total credit R^* satisfies

$$(3) \quad R^* \geq \Omega := \sum_c 2^{-\beta(c)} + \sum_y 2^{-\beta(y)}$$

by (V5), where c runs over the LCA nodes and y over the blue starts.

Lemma 5.5 (gain bound). $R^* \leq 1 + \sum_w (1 - 2^{-k(w)})$, where w runs over the vertices of B .

Proof. Sum (2) over all nodes of the flow tree other than o_m , and group the terms by the parent. The children of o_m contribute at most 1, by (V1). The children of a live pass-through q meet at q and contribute $\kappa(q)\mu_q$, where $\mu_q \leq 1$ is their part of the sum at q . The nodes that climb contribute a total C . Since every live pass-through is a node of the flow tree that is not required,

$$R^* \leq 1 + \sum_q \kappa(q)\mu_q + C - \sum_q \kappa(q) \leq 1 + C.$$

A source that climbs through $q = (w, i)$ to p has $\gamma = \text{cl}(q, p) \geq 1 + k(w) - i$ and $\kappa(p) \leq 1$, and the sources with root q have total sum at most 1. Hence $C \leq \sum_{(w,i)} 2^{-(1+k(w)-i)} = \sum_w (1 - 2^{-k(w)})$. $\square$

The search fixes the numbers $k(w)$ one segment at a time. Suppose some segments are *fixed* and the others are *free*, with value 0 for now; a completion gives the free segments values $k'(w) \leq k_{\max}(w)$. For each required node o let λ_o be the inequality of Lemma 5.4(b) or (c) for a pair that attains the minimum in the definition of $\beta(o)$, so that its right-hand side minus its left-hand side at the current k equals $\beta(o)$, and let $\Delta_w = \sum 2^{-\beta(o)}$ over the o whose λ_o contains $k(w)$. Write $\Omega(k)$ for the value of (3) at the current k .

Lemma 5.6 (lifted gain bound). *If some completion has a valid configuration, then*

$$1 - \Omega(k) + \sum_{w \text{ fixed}} (1 - 2^{-k(w)}) + \sum_{w \text{ free}} \max_{0 \leq k' \leq k_{\max}(w)} \left((1 - 2^{-k'}) - \Delta_w(2^{k'} - 1) \right) \geq 0.$$

Proof. In a completion k' , each $\beta(o)$ is at most the value of λ_o (its right-hand side minus its left-hand side), which is $\beta(o)$ at k minus the sum of $k'(w)$ over the free segments in λ_o ; if the completion has a valid configuration, these values are nonnegative by Lemma 5.4. Since $2^{a+b} - 1 \geq (2^a - 1) + (2^b - 1)$ for $a, b \geq 0$, this gives $\Omega(k') \geq \Omega(k) + \sum_{w \text{ free}} \Delta_w(2^{k'(w)} - 1)$. Combine with (3) and Lemma 5.5 at k' . $\square$

For a vertex u of B , the *region* of u is the set of red nodes v with $u(v)$ in the subtree of u . Define Ψ on the vertices of B from the leaves up: $\Psi(u) = 1$ for a leaf, and for an internal vertex e with children a and b ,

$$\Phi_e = [e \text{ is an LCA node}] + \Psi(b)S_a(e) + \Psi(a)S_b(e), \quad \Psi(e) = \max(\Psi(a), \Psi(b), \Phi_e),$$

where $S_a(e) = \sum 2^{-\text{cl}(q,e)}$ over the pass-throughs q in the region of a , and similarly $S_b(e)$.

Lemma 5.7 (amplification bound). $R^* \leq \Psi(\text{root})$. *If the root ϱ of B is an LCA node, put $a_{\min} = 2^{-\beta(\varrho)}$ and $\Psi' = \max(\Psi(a), \Psi(b))$ for the children a, b of ϱ . Then $R^* \leq \Phi_\varrho$ if $\Phi_\varrho \geq \Psi'$, and $R^* \leq a_{\min}\Phi_\varrho + (1 - a_{\min})\Psi'$ otherwise.*

Proof. For a node s of the flow tree other than o_m , let $\phi(s)$ be the total credit of the required nodes in the subtree of s , divided by $\kappa(s)$. We show $\phi(v) \leq \Psi(u(v))$ for each live red node v , by induction from the bottom of the flow tree; note that Ψ does not decrease towards the root and that $\Psi \geq 1$. A blue start has no children and has $\phi = 1$. A live pass-through q is not required, and its children meet at q . They lie below q in the region of $u(q)$, and their part of the sum at q is at most 1, so $\phi(q)$ is at most the largest ϕ of a child, hence at most $\Psi(u(q))$. A live branch node e with children a and b has as its children in the flow tree the sources that climb to e . Such a source s climbs through a pass-through q in the region of one child, say a . Its successor e is $\text{lca}_B(u(q), u(s))$, and $u(q)$ is not $u(s)$ or a descendant of $u(s)$ (see the remarks after Definition 4.9), so s lies in the region of b ; or s is a blue start. Its value of ϕ is then at most $\Psi(b)$. Since the sources with root q have total sum at most 1,

$$\phi(e) \leq [e \text{ is an LCA node}] + \Psi(b)S_a(e) + \Psi(a)S_b(e) = \Phi_e \leq \Psi(e).$$

Now $R^* = \sum_s \kappa(s)\phi(s) = \sum_s 2^{-d(s)}\phi(s)$ over the children s of o_m . By (V1) this is at most the largest $\phi(s)$, hence at most $\Psi(\text{root})$.

If the root ϱ of B is an LCA node, its root is o_m : since $k(\varrho) = 0$, no pass-through lies above it, and a pass-through below it cannot be its root (remarks after Definition 4.9). So ϱ does not climb. Its term is $a\phi(\varrho)$ with $a = 2^{-d(\varrho)} \geq 2^{-\tau(\varrho)} \geq a_{\min}$ and $\phi(\varrho) \leq \Phi_\varrho$. The other children of o_m lie in the regions of a and b or are blue starts, so their terms total at most $(1 - a)\Psi'$. Being linear in $a \in [a_{\min}, 1]$, the bound $a\Phi_\varrho + (1 - a)\Psi'$ is largest at an endpoint. $\square$

With $H(e) = \sum_z (1 + k(z))$ over the internal vertices z from e up to the root, the pass-throughs on the segment of a vertex w in the region of a contribute $(1 - 2^{-k(w)})2^{-(H(\text{par } w) - H(e))}$ to $S_a(e)$, where $\text{par } w$ is the parent of w . For a partial assignment of the numbers $k(w)$, the search evaluates the bound with $k_{\max}(w)$ pass-throughs on each free segment, with H and $\beta(\varrho)$ at the current k . In every completion, Ω is at least its current value and R^* is at most this bound: more pass-throughs give more terms, the current H gives the shortest climbs, and the current $\beta(\varrho)$ gives the smallest $a_{\min}$.

During the search over sources, the red nodes are assigned in an order in which each node comes after the nodes above it. At a given point, let $\text{free}(r)$ be 1 minus the sum of 2^{-d} over the

sources already assigned to the root r . For a pass-through $q = (w, i)$ put $F_q = 2^{-(1+k(w)-i)}$, or $F_q = \max(\kappa(q), 2^{-(1+k(w)-i)})$ if q is assigned and live.

Lemma 5.8 (credit capacity). *Every valid completion of the current partial assignment satisfies*

$$\sum_s 2^{-\beta(s)} \leq \text{free}(o_m) + \sum_q \text{free}(q) F_q,$$

where s runs over the required nodes not yet assigned.

Proof. For an unassigned required node s , follow the parents in the flow tree as long as the step goes from a child that meets at a pass-through not yet assigned. Let p be the node where this stops. Then p is unassigned, and either its parent is o_m , or it meets at an assigned pass-through q , which is live by (V4), or it climbs through some pass-through q . By (2), $\kappa(p)$ is at most $2^{-d(p)}$, $2^{-d(p)}\kappa(q)$ or $2^{-d(p)}2^{-(1+k(w)-i)}$ in these three cases. Group the required nodes by this node p . In the subtree of p formed by the steps that meet at unassigned pass-throughs, the required nodes are leaves, since sources meet only at pass-throughs, while the required nodes are branch nodes or blue starts. The sums at the pass-throughs are at most 1, so the required nodes in this subtree have total credit at most $\kappa(p)$. Since the nodes p are unassigned, their sums $2^{-d(p)}$ at a root r total at most $\text{free}(r)$. Finally $2^{-\beta(s)} \leq \kappa(s)$ by (V5). $\square$

5.4. Symmetry. Two channels are *interchangeable* if exchanging them maps the function V on E to itself. This is an equivalence relation, and the product G of the symmetric groups of its classes consists of automorphisms of I . A class that meets P' lies inside P' , and $V(gx, gy) = V(x, y)$ for $g \in G$. Relabelling a valid configuration by an element of G gives a valid configuration, with the same number of pass-throughs, whose tree is the image of B , and the conditions (N3)–(N5) and the conclusion of Lemma 5.3 are preserved. Two trees whose leaves are labelled by the channels of P' lie in one G -orbit exactly when they are isomorphic as trees with leaves coloured by the classes. It therefore suffices to examine one labelled tree for each coloured isomorphism class. Blue starts in one class have the same sets E^y and the same budgets. Exchanging their sources therefore keeps a configuration valid and in normal form, and we may assume:

- (N6) within each maximal run of consecutive blue starts of one class, in the order of the search, the roots do not decrease in a fixed order of the roots.

The program computes the classes from the scenarios themselves.

5.5. The search. The program proceeds as follows.

- (1) Compute the numbers A_x . Answer infeasible if some $A_x < 0$ or $\sum_x 2^{-A_x} > 1$ (Lemma 5.4(a)).
- (2) Enumerate the coloured binary trees on P' , one for each isomorphism class, from the root down. Each leaf x has a cap on its depth, at first A_x . When the second subtree of a vertex c is enumerated, the depths of the leaves of the first subtree are known, and the cap of each leaf y of the second subtree is lowered to the least value of $b + D(c) + 1 - D(x)$, where x runs over the leaves of the first subtree and b over the budgets $b(x, y)$ and $b(y, x)$ of the pairs in E (Lemma 5.4(b), whose left side is symmetric in x and y). Because V is invariant under G , these budgets depend only on the classes of x and y , and the program uses the deepest leaf of each class. The subtree condition of Lemma 5.4(a) is applied with the current caps.
- (3) For each tree, enumerate the numbers $k(w)$ on the segments other than that of the root, from the top down, under the linear inequalities of Lemma 5.4. At each node of this enumeration, including the first, discard the partial assignment if Lemma 5.6 or the lifted form of Lemma 5.7 fails.

- (4) For each (B, k) , compute δ , pre, β and the budgets of the blue starts, and discard (B, k) if one of them is negative, if some pass-through has no affordable climber (Lemma 5.3), or if Lemma 5.5 fails with R^* replaced by Ω . Then search the red nodes depth first, in an order in which each node comes after the nodes above it. A node that is not an LCA node may be **dead**. Otherwise it receives a root, o_m or a pass-through allowed by (V3), whose successor is live, and a depth. Its value τ is then determined, and it must satisfy $\tau \leq \beta$ for an LCA node and $\tau \leq B_{\max}$ otherwise. A node that is not readable receives only the choices allowed by (N3). Depths are tried from the largest allowed value down, and the loop stops when the sum at the root would exceed 1. The condition of Lemma 5.8 is tested whenever some red node is still unassigned. Condition (V3) is enforced in part during the search, by excluding roots at or below the node, and in full in step (6).
- (5) Then search the blue starts depth first. Each receives a root whose successors are live and the largest depth allowed by (V5), subject to (N6) and to (V1).
- (6) At a complete assignment, check (V2) and (V3), evaluate every cost again with a separate implementation of the recursion, build the network of Remark 4.11, and simulate the run of every scenario. Answer feasible. If the two evaluations disagree or a simulated run fails, the program stops with an error.

If the search ends without a complete assignment that passes step (6), the answer is infeasible.

Proposition 5.9. *If `lb2cr` answers infeasible for an instance whose budgets are all at most 56, then the instance is infeasible.*

Proof. Suppose the instance is feasible. Then $E \neq \emptyset$ and all $b(x, y) \geq 0$, since the empty network and $|W| \geq 0$ show the other cases. By Theorem 4.10 there is a valid configuration. Take one with the fewest pass-throughs, put it in normal form by Lemma 5.2, and relabel it by an element of G so that its tree is the one listed in step (2), then apply (N6). By Lemmas 5.3 to 5.8, no test of steps (1)–(5) discards it or one of its partial assignments, and each choice it makes is offered. The search then reaches it in step (6). $\square$

The bound 56 comes from the arithmetic. Kraft sums are computed exactly as integers in units of 2^{-56} , and a depth above 56 gets weight 0, which only weakens each test. The credit bounds use exact 128-bit integers, except the amplification bound. That bound is computed in floating point from nonnegative dyadic terms, and the program prunes only if Ω exceeds it by a relative 10^{-9} plus 10^{-12} , far more than the rounding error. Rounding is toward the weaker test except in four places: the lifted gain bound maximizes over $k' \leq 56$ only, the amplification bound would overflow once H exceeds about 1000 and caps $\beta(\varrho)$ at 60, and the number 2^{28} marks the absence of a budget. None of them matters when every budget is at most 56, because $k_{\max}(w)$, H and $\beta(\varrho)$ are at most $\max b(x, y)$. With such budgets every depth in the search is at most 56 as well, so step (6) builds and simulates a network for every feasible answer. The program also answers unknown for malformed input. Every instance that the verification passes to it has budgets at most 41 (Section 9).

6. LOWER BOUNDS FOR CORES

The weights of the instances in the search are lower bounds on the costs of cores. They come from three sources.

The trivial bound. Let X be produced by a prefix P of k standard comparators on 13 channels, and let (x, y) be a scenario of (X, ϵ) . By Lemma 2.9 and $S(11) = 35$ [5], its core satisfies

$$s(Y_{xy}) \geq 35 - (k - w_\epsilon(x, y)),$$

where $w_\epsilon(x, y)$ is the largest value of $|W(i, j)|$ over the runs (i, j) of polarity ϵ of P that end on (x, y) . If no such run ends on (x, y) , the trivial bound is 0. Since $k \leq 25$ in the search, the trivial bound is 0 or at least 10.

Instances on 11 channels. A core Y on 11 channels has scenarios of its own, whose cores lie on 9 channels. Proposition 3.2 applied to Y gives: if $u_t \leq s(Z_t)$ for the cores Z_t of the scenarios of (Y, ϵ') , and the instance $I(Y, \epsilon'; u; V - 1)$ is infeasible, then $s(Y) \geq V$. Each weight u_t is justified in one of two ways.

- (E) u_t is at most the exact value $s(Z_t)$, computed by Harder's program.
- (B) For some polarity ϵ'' , the instance $I(Z_t, \epsilon''; z; u_t - 1)$ on 9 channels is infeasible, where z consists of the exact values of the 7-channel cores of (Z_t, ϵ'') , computed by Harder's program. Then $s(Z_t) \geq u_t$ by Proposition 3.2.

Sharing. By Lemma 2.2(c), a bound on $s(Y)$ holds for every set similar to Y . A bound proved for the core of one scenario of one prefix thus serves every scenario, of any prefix, whose core is similar to it. In particular, a trivial bound found for one prefix can be larger than the trivial bound of another prefix with a similar core.

Harder's program computes $s(Z)$ for a well-behaved set Z by the recursion $s(Z) = 1 + \min s(Z')$ over the successors Z' of Z , taken up to similarity, with lower bounds from his Huffman bound [5, Lemma 12, Theorem 26, Corollary 56]. By Lemma 2.5 all sets on 9 and 7 channels that we pass to it are well-behaved. We use his program with our changes, which let it handle sets on up to 13 channels, and add a command-line front end that prints $s(Z)$ for a set given as a bitmap. Further changes, such as query servers, options for seeding bounds and upper bounds for 12 and 13 channels, are not used in the computation of the tables. On that computation the changed program runs his search unchanged, except that a state that is stored again keeps the larger of its two lower bounds and the smaller of its two upper bounds. The values are not certified. They form one of the trusted components of Section 10.

The search computes these bounds as needed. For a child at depth k it first uses the trivial bounds and the bounds already proved for similar cores. If the instance on 13 channels is still feasible, the instance solver of the search returns a configuration, and for the scenarios whose costs in it reach their budgets, the search tries to raise the bounds on their cores through instances on 11 channels. For the weights on 9 channels it uses the exact values already known, including a table loaded at the start, and bounds of type (B) otherwise, and it computes exact values for the scenarios on 9 channels whose costs reach their budgets.

7. THE SEARCH OVER PREFIXES

A *state* of depth k is a set $X \subseteq \{0, 1\}^{13}$ produced by a prefix of k standard comparators, stored together with that prefix. The search produces *layers* $L_0, L_1, \dots$, which are finite lists of states, the states of L_k having depth k , with $L_0 = \{\{0, 1\}^{13}\}$. For a state $R \in L_{k-1}$ and a standard comparator (i, j) that is useful for R , the *child* $R(i, j)$ has depth k , the prefix of R followed by (i, j) , and the budget $44 - k$. Given the layers, a *record* of a child is *valid* if it is one of the following:

- *cut*: for some polarity ϵ and weights v that are lower bounds on the costs of the cores (Section 6), the instance $I(R(i, j), \epsilon; v; 44 - k)$ is infeasible, so $s(R(i, j)) \geq 45 - k$ by Proposition 3.2;
- *alive*: a set similar to the child lies in L_k ;
- *duplicate*: the record names another child. Each duplicate record on the way names a child that is similar to its own child and has at most its depth, and after finitely many steps one reaches a valid cut or alive record.

The search builds L_k from the alive children of depth k , one for each canonical form, and stops at the first empty layer.

Theorem 7.1. *Suppose that every child of every state in $L_0, \dots, L_{24}$ has a valid record, that no state in these layers has 14 or fewer vectors, and that L_{25} is empty. Then $S(13) \geq 45$.*

Proof. Suppose $S(13) = s \leq 44$, and let $g_1, \dots, g_s$ be a standard sorting network on 13 channels with s comparators (Lemma 2.2(a)). Let X_j be the set produced by $g_1, \dots, g_j$. Then $s(X_j) = s - j$: the remaining comparators sort X_j , and a shorter network sorting X_j , preceded by $g_1, \dots, g_j$, would sort $\{0, 1\}^{13}$ with fewer than s comparators. Each g_{j+1} is useful for X_j : otherwise X_{j+1} equals X_j or arises from it by exchanging two channels, and $s(X_{j+1}) = s(X_j)$ by Lemma 2.2(c).

We show by induction on $j \leq s$ that some layer L_d with $d \leq \min(j, 24)$ contains a state similar to X_j . For $j = 0$ this is L_0 . Let $R \in L_d$, $d \leq j$, be the image of X_j under a permutation of the channels possibly combined with complementation, and let $g_{j+1} = (a, b)$. The same map takes X_{j+1} to the image of R under a comparator (a', b') , where $\{a', b'\}$ is the image of $\{a, b\}$. If $a' < b'$, this image arises from the image under the standard comparator (b', a') by exchanging channels a' and b' . Hence R has a child $C = R(i, j)$, with $\{i, j\} = \{a', b'\}$ and $i > j$, that is similar to X_{j+1} . The comparator (i, j) is useful for R , because g_{j+1} is useful for X_j and the map preserves this property. The child C has depth $d + 1 \leq j + 1$, and $s(C) = s(X_{j+1}) = s - j - 1 \leq 44 - (d + 1)$. Hence its record is not a cut. If it is alive, L_{d+1} contains a set similar to C . If it is a duplicate, the names lead to a child C' of depth $d' \leq d + 1$ that is similar to C and has a valid cut or alive record. Since $s(C') = s(C) \leq 44 - d'$, that record is not a cut but alive, and $L_{d'}$ contains a set similar to C . In both cases this layer has depth at most $d + 1 \leq 25$ and is not empty, so its depth is at most 24.

For $j = s$ the set X_s consists of the 14 sorted vectors, so some layer L_d with $d \leq 24$ contains a state with 14 vectors. This contradicts the hypothesis. $\square$

Proof of Theorem 1.1. Juillé's network [7] gives $S(13) \leq 45$. The certificate of Section 8 records the layers $L_0, \dots, L_{25}$ and a record for every child of every state in $L_0, \dots, L_{24}$, and the verification of Section 8 confirms the hypotheses of Theorem 7.1, using the components listed in Section 10. Hence $S(13) \geq 45$. $\square$

7.1. The program. The search is the program `1b2w`, which builds on Harder's library for sets of vectors [6]: his representation of sets as bitmaps, his canonical form of a set up to similarity, and his search for exact values. It decides the instances with a solver of its own, which the verification does not use. When that solver reaches a limit on its work, the child is kept. For each child it builds the two instances on 13 channels, one for each polarity, first with the trivial bounds and the bounds already known for similar cores, and then with bounds raised as described at the end of Section 6. Each process keeps a table of the best bound known for each core in canonical form, so that a bound found once serves all later scenarios of the same process with a similar core.

Duplicates are detected within one process: the canonical form of each child is looked up among those of the children already generated by the same process, at any depth, and a repeated child refers to the first one. We computed layers 1 to 12 with one process, which used only the trivial bounds and their sharing. From depth 13 on, we dealt out the states of each layer in turn to several servers, each running one process at a time, which kept its tables from layer to layer until the process was restarted. A coordinator merged the alive children of each layer, kept one state for each canonical form, and dealt the result out as the next layer. When a server failed, its share of a layer was computed again by fresh processes, split into up to eight parts, and the coordinator's log names the output that it used.

8. THE CERTIFICATE AND ITS VERIFICATION

8.1. The records. Each process of the search writes, for each layer, one record for each child that it generates, and a file of bounds on cores, the *claims*. A set is written as a bitmap. A map between similar sets is written as a flag for complementation followed by a permutation of the channels. The records are of four kinds.

- A The child is alive. The record gives a map that takes the child to a set listed in the merged set of its layer.
- D The child is a duplicate. The record names a child of the same process, by layer, parent and comparator, and gives two maps that take the two children to the same set.
- T, C The child is cut. The record gives the polarity ϵ and the weights v_t of the infeasible instance, and for each scenario whose weight exceeds its trivial bound, a claim whose bound is at least v_t and a map that takes the core of the scenario to the core of the claim. The letter T marks cuts that use trivial bounds and shared bounds only, and C the cuts found after bounds were raised by instances on 11 channels.

The claims are of three kinds.

- Q A core on 11 channels, with an identifier.
- K T A bound $s(Y) \geq V$ for a core Y , justified by the trivial bound of a scenario of a child of the same process: the claim names the child, the polarity, the scenario and a map that takes its core to Y .
- K L A bound $s(Y) \geq V$ justified by an instance on 11 channels: the claim gives the polarity ϵ' and the weights u_t of the scenarios of (Y, ϵ') , each with a tag E or B for the justifications (E) and (B) of Section 6.

Each process also writes a manifest, which identifies the input of each of its layers by its length and a 64-bit hash. The coordinator writes the list of states of each layer, the share of each server, the merged set of each layer, and its log. The merged set lists the states of the layer, each with its canonical form, which is the image of the state under the map of the alive record that produced it. In layers 1 to 12, which one process computed, the alive children of that process form the layers. The format is specified in full in the supplement, so that a verifier needs nothing from the code of the search.

8.2. The verifier. The verifier consists of a program *cv*, which checks the records and claims of the processes, and a script *gv*, which checks the layers as a whole. Both were written from the format specification and the mathematics of Sections 2 to 6, without the search code. They decide every instance with *1b2cr*, which was written from a separate derivation of the mathematics of Sections 4 and 5, extending the model of [11], without the code of the instance solver of the search, and they ask Harder's program for exact values. For each layer of each process, *cv* checks the following.

- (i) In layers 2 to 12 the input of the layer is the list of alive children of the previous layer of the same process. In the other layers it has the length and hash that the manifest records, and *gv* matches it with the share named in the coordinator's files. Each state of the input is the set produced by its prefix, which consists of standard comparators and has the length of the depth. The only state of depth 0 is the full set.
- (ii) Each standard comparator that is useful for a state gives exactly one record, and no record names a comparator that is not useful.
- (iii) For a cut, it computes the trivial bounds of all scenarios of the given polarity from the prefix, checks that each weight above its trivial bound is covered by the named claim and map, and checks that *1b2cr* answers infeasible for the instance. The program computes the symmetries of each instance itself and ignores any hint in the record.

- (iv) For an alive child, it checks that the child has more than 14 vectors and that its image under the map is listed in the merged set of its layer.
- (v) For a duplicate, it checks that the two maps give the same set, that the named child has a depth at most that of the duplicate, and that following the names ends at a verified alive or cut record.

For each claim that a record uses, `cv` checks the following.

- (vi) For `KT`, it recomputes the trivial bound of the named scenario and applies the map.
- (vii) For `KL`, it checks that `1b2cr` answers infeasible for the instance on 11 channels with budget $V - 1$. For a weight tagged `E`, it checks that the weight is at most the exact value of the core on 9 channels, or at most a bound of type (B) for it. For a weight tagged `B`, it checks (B) for one of the two polarities on a set similar to the core, with exact values of the cores on 7 channels.

In all of these, maps must be permutations of the channels, with or without complementation, and the identifiers of claims must be unique. Every instance passed to `1b2cr` must have budgets between 0 and 56. The verifier records the largest budget of each kind of instance. The script `gv` checks that the shares of each layer split it as the coordinator's files say; that for each share exactly one output is used, as named in the log; that every state of a layer is covered; that layers 1 to 12 end in the states of depth 12 from which layers 13 on start; that every line of a merged set comes from an alive record of a used output, and that the images of these alive records are exactly the listed canonical forms; that each layer is the merged set of the previous one; and that layer 25 is empty. No option switches off any of these checks, and the job files of the released run list the merged set of every layer, which the check of alive children needs. The supplement records the invocations of both programs.

Nine shares of layers 14 to 17 were computed again after a server failed, and for each of them the last line of the log that concerns it names the output used. For two shares of layer 14, the processes whose outputs were replaced went on to later layers, whose duplicate records may name children of the replaced outputs. Those children are checked like all others, but an alive child among them can end a chain of duplicates only if its image is listed in the merged set of layer 14. In the replaced outputs, 37 alive records and 8 duplicate records do not satisfy the conditions above. The verifier reports them, but no record of a used output depends on them.

8.3. Exact values. The exact values were computed in advance for the cores on 9 and 7 channels whose values the verifier needs, and stored in two tables with 3,624,738 and 4,028,993 entries. Each entry gives the value for one image of the core under a map of the kind above, which the verifier computes before the look-up. This image is not always a canonical form, but by Lemma 2.2(c) any image will do. Of the 15,430,581 cores on 9 channels that the claims use, counted up to the verifier's representatives, 3,624,738 needed exact values. For the others, bounds of type (B) cover every weight of the claims that use them. For each core with an exact value e , the verifier also passes the two instances on 9 channels with budget e to `1b2cr`, with the exact values on 7 channels as weights, and requires that neither is infeasible. An infeasible answer would show that `1b2cr` or Harder's program is wrong.

9. RESULTS

Table 1 lists the layers. The search went extinct at depth 25: the three states of depth 24 have 134 children, all of them cut. In total the outputs used contain 41,142,659 records of children, of which 31,899,147 are cuts, 2,120,883 alive and 7,122,629 duplicates. The two replaced outputs of layer 14 add 369,927 records. The layers $L_1, \dots, L_{24}$ contain 662,820 states. The cuts use 11,708,977 bounds of kind `KT` and 12,381,230 of kind `KL`. The claims of kind `KL` have 148,319,131

weights tagged E and 78,082,756 tagged B. The verifier covered 56,549,674 of the weights tagged E by bounds of type (B), and looked up the other 91,769,457 in the table of exact values.

k	$ L_{k-1} $	children	alive	duplicate	cut T	cut C	$ L_k $
1	1	78	1	77	0	0	1
2	1	77	2	75	0	0	2
3	2	152	5	146	1	0	5
4	5	375	13	358	4	0	13
5	13	961	36	902	23	0	36
6	36	2,624	106	2,389	129	0	106
7	106	7,617	300	6,616	701	0	300
8	300	21,234	823	17,186	3,225	0	823
9	823	57,392	2,398	42,095	12,899	0	2,398
10	2,398	164,723	7,311	107,586	49,826	0	7,311
11	7,311	494,205	21,956	282,325	189,924	0	21,956
12	21,956	1,459,781	65,659	717,716	676,406	0	65,659
13	65,659	4,292,552	318,344	1,163,812	2,544,340	266,056	100,762
14	100,762	6,475,632	490,716	1,138,486	4,241,800	604,630	142,881
15	142,881	9,006,443	489,270	1,243,569	5,938,250	1,335,354	133,426
16	133,426	8,213,431	358,766	1,072,034	5,263,682	1,518,949	94,630
17	94,630	5,667,481	212,574	719,135	3,513,826	1,221,946	54,471
18	54,471	3,167,491	99,158	374,536	1,942,775	751,022	24,920
19	24,920	1,405,142	37,391	156,818	859,050	351,883	9,250
20	9,250	504,626	11,567	53,685	312,013	127,361	2,812
21	2,812	147,750	3,403	16,976	92,216	35,155	807
22	807	40,768	892	4,743	25,922	9,211	208
23	208	10,114	174	1,132	6,635	2,173	40
24	40	1,876	18	232	1,232	394	3
25	3	134	0	0	103	31	0

TABLE 1. The layers of the search. Row k counts the children of the states of L_{k-1} by the kind of their record. Layers 1 to 12 were computed by one process. From layer 13 on, alive children of different processes can be similar, and $|L_k|$ counts their similarity classes. Duplicates are found within one process.

The first process used only trivial bounds and needed no exact values. From layer 13 on, the cuts of kind C make up 9% of the cuts at depth 13, and between 22% and 30% from depth 16 on. The verifier decided with **1b2cr** 32,176,451 instances on 13 channels, 12,381,230 on 11 channels and 43,451,218 on 9 channels for the bounds of type (B), and 7,249,476 more on 9 channels for the test at the exact values. The largest budgets were 41, 28 and 25 on 13, 11 and 9 channels. These counts, and those of claims and weights above, include the records of the two replaced outputs of layer 14 (Section 8).

Computation. All servers had 384 hardware threads unless stated otherwise. Layers 1 to 12 took under 9 minutes on one server. Layers 13 to 25 took 10 hours and 50 minutes of wall-clock time on 35 to 55 servers, with two more that ran only backup processes. The processing times that the coordinator recorded for the layers add up to about 190 server-hours. The exact values took under an hour of one server for the sets on 7 channels and about 110 server-hours for the sets on 9 channels. The verification of the released files took 20 minutes on a server with 192 threads.

10. TRUSTED COMPONENTS AND CROSS-CHECKS

The proof of Theorem 1.1 rests on Sections 2 to 7 and on the verification of Section 8. The verification checks every record and every claim that a record uses. It does not check the following three components.

- (1) *Harder’s program*, with the changes described in Section 6, for the exact values of the 3,624,738 sets on 9 channels and the 4,028,993 sets on 7 channels in the two tables. These sets are well-behaved (Lemma 2.5), as Harder’s proof of correctness of his algorithm requires. The same search computes $S(11) = 35$ in [5], where its results are backed by certificates; the values in our tables have no certificates.
- (2) *The bound* $S(11) = 35$, used by every trivial bound. Harder’s certificate for $S(11) \geq 35$ is checked by a program verified in Isabelle/HOL [4], and again by a checker whose soundness is proved in Lean [11, Section 7].
- (3) *The program lb2cr*, for the answer infeasible on each instance that the verifier passes to it. Its correctness for budgets at most 56 is the content of Sections 4 and 5 (Proposition 5.9).

We trust the verifier itself, the compilers and the hardware in the usual way, but not the search, its instance solver or its bookkeeping. Table 2 lists the tests of these components. Section 11.4 states what the proof of Section 11 trusts, and Section 12.4 what the proof of Section 12 relies on.

check	scope	outcome
lb2cr at budget equal to the exact value, on the instances on 9 channels with the exact values on 7 channels as weights	all 3,624,738 cores with exact values, both polarities	never infeasible
an exact solver written separately, by search over sets up to similarity, against the 7-channel table values of the 9-channel table computed twice	every 200th entry, 20,145 in all	all agree
lb2cr against an exhaustive search over networks, which gives the least feasible budget L^*	9,481 cores, 9,348 of them on two different machines, among them 3,294 computed again after a machine went down	no conflict
feasible answers of lb2cr	7,340 random instances with 4 to 8 channels, 28,877 answers at budgets from $L^* - 2$ to $L^* + 1$	no disagreement
prefixes of Juillé’s network [7] and of its mirror image, at budget $45 - k$	every feasible answer, for budgets at most 56	network built and every run simulated
Theorem 4.10	528 instances of both polarities, from 3 orders of the comparators of each	none infeasible
the proof in Lean of Section 11	formal proof in Lean 4	standard axioms only
the proof in Lean of Section 12	the records of the outputs used and the claims that they use, with its own search	all checks pass
the verifier on a small certificate with single corruptions	the same records, with every computation checked by Lean’s kernel	all checks pass, standard axioms only
	20 corruptions, 19 of the certificate and one of its table of exact values	each detected by the expected check

TABLE 2. Tests of the trusted components. The supplement contains the programs and logs for each row except the third, which its README describes, and the proof of Section 12, which is archived with its build log in [15].

11. A PROOF IN LEAN

We checked the certificate a second time, by a proof in Lean 4 [9] with Mathlib [10]. It has its own search over configurations and its own checks of the records. A proof in Lean shows that the instances this search rejects are infeasible. The Lean project is part of the supplement [13], with scripts that regenerate the converted certificate and tables from the released files.

11.1. The statement. The definitions of networks and of sorting are those of the proof of $S(13) \geq 44$ in [11, Section 7], which use only Lean’s core library. A `Network 13` is a list of standard comparators on 13 channels, and `net.Sorts` says that the output of `net` is nondecreasing on every input vector of natural numbers. The main theorem is

```
theorem Sortnet.S13_ge_45 : ∀ net : Network 13, net.Sorts → 45 ≤ net.length
```

As in Harder’s certificates [5], the proof works with networks whose comparators may be any pairs of channels, and such a network *works* on a set V of 0-1 vectors if any two members of V whose outputs have the same weight have the same output. In what follows, a lower bound b for a set means that every network that works on the set has at least b comparators.

11.2. What is proved. The following results are proved in Lean from the three standard axioms of its logic, together with the computations of Section 11.3 where stated.

- (i) *Theorem 4.10.* The formal instance is a budget and a list of scenarios (x, y, v) on channels `Fin n`. A network is a list of pairs of distinct channels, and the run of a scenario is computed by a function that moves M and m as in Section 2.1 and counts the comparators that it touches:

```
def Feasible {n : Nat} (I : Inst n) : Prop :=
  ∃ net : List (Fin n × Fin n), Proper net ∧ ∃ oM om : Fin n, oM ≠ om ∧
    ∀ s ∈ I.sc, (tokRun net s.x s.y).M = oM ∧ (tokRun net s.x s.y).m = om ∧
    (tokRun net s.x s.y).w + s.v ≤ I.L
```

```
theorem feasible_valid {n : ℕ} (I : Inst n) (hwf : I.WF) (hne : I.sc ≠ [])
  (hf : Feasible I) : HasValid I
```

Here `I.WF` says that each scenario has $x \neq y$, and `HasValid I` says that some configuration satisfies a structure `Valid`. Its fields are the requirements of Definition 4.8 on the tree, the numbers k and the roots, the conditions (V1)–(V6), the recursion (1) for the values τ , which the formal configuration carries as data, and a relevance condition: every live red node can be reached from an LCA node or a blue start by a chain of successors. The checker `1b2cr` does not use the relevance condition. The search in (ii) derives from it (N4) and the part of (N3) that marks nodes *dead*, in place of the deletion argument of Lemma 5.2.

- (ii) *A search over configurations.* A function `Comp.solve` performs the search of Section 5.5 without the tests of Lemmas 5.3 and 5.5 to 5.8 and without the conditions (N5) and (N6), and in step (6) it only checks (V2) and (V3). Leaving out a test can only make the search accept more instances. The theorem `solve_complete` states that `Comp.solve` accepts every instance on at most 13 channels that has $x \neq y$ in each scenario and a valid configuration. With (i), an instance on at most 13 channels that `Comp.solve` rejects is infeasible. Kraft sums are exact integers in units of 2^{-56} , with weight 0 for a depth above 56. All other arithmetic is exact.
- (iii) *The weighted two-value bound* for the sets of the search. If the instance of a state for one polarity, with weights that are lower bounds for the cores of its scenarios, is infeasible with budget L , then every network that works on the state has at least $L + 1$ comparators.

- (iv) *The trivial bound* of Section 6, from $S(11) \geq 35$. The bound $S(11) \geq 35$ comes from a computation that checks Harder’s certificate [4], which has 12,659,079 steps. The computation uses the checker of [11, Section 7.4], whose soundness is proved in Lean.
- (v) *The induction over the layers*. Suppose that the empty network works on no listed state, and that for each listed state X of depth d and each standard comparator c , the set $c(X)$ is X , or the image of X under the exchange of the two channels of c , or similar to a listed state of depth at most $d + 1$, or needs at least $44 - d$ comparators. Then each listed state of depth d needs at least $45 - d$ comparators, and in particular every sorting network on 13 channels has at least 45 comparators. A reference to a state of depth $d + 1$ uses the bound for a shorter network, so cycles among the references are harmless.
- (vi) *The bounds on cores*. A claim KL holds if the search of (ii) rejects its instance on 11 channels and each weight is a lower bound for its core on 9 channels. Such a bound comes from the route (B) of Section 6, by a rejected instance on 9 channels whose weights are proved bounds on 7 channels, or from the table of values on 9 channels, whose entries are proved in (vii). All 4,028,993 values on 7 channels are proved as lower bounds, in computations by a checker whose soundness is proved in Lean. Each value follows from the bounds for the successors of the set, by the successor rule of [11, Section 7.2], down to sets for which route (B) applies one level lower, with an instance on 7 channels and values on 5 channels that the checker computes.
- (vii) *The bounds on 9 channels*. Each of the 3,624,738 values on 9 channels in the table is proved as a lower bound, in computations by checkers whose soundness is proved in Lean. Here a set X contains a set Z if X contains an image of Z under a permutation of the channels, possibly combined with complementation. If X contains Z , a lower bound for Z is one for X , and if X contains $c(Z)$ for a comparator c , a lower bound b for Z gives the bound $b - 1$ for X . Of the entries, 3,344,121 follow in this way, directly or through one or two comparators, from a step on 9 channels of Harder’s certificate [4] or from other entries, and a rank in each record excludes cycles. The other 280,617 are proved by the successor rule, down to sets for which route (B) applies, as for the values on 7 channels in (vi).

11.3. Data and computations. An untrusted program converts the released certificate files into one stream of 1,011,669,465 bytes, which the proof reads at build time as 16 string constants. It keeps the records of the outputs that the coordinator used and the 12,315,738 claims KL that these records use. Chains of duplicates are replaced by their final targets. The checking functions recompute from the prefixes the states, the useful comparators, the children, the scenarios, the cores and the trivial bounds. The data supply the rest: for each listed state its parent and comparator, and for each child a listed state with a map, a cut child with a map, or a cut with its claims. The tables of values on 9 and 7 channels and Harder’s certificate are read in the same way. Each constant is pinned by a 64-bit hash, and the build script compares the regenerated data with their SHA-256 digests. An error of the converter can make a check fail, but the soundness theorems hold for any data.

The checks run by the tactic `native_decide`, which compiles a Boolean computation, runs it, and records its result as an axiom. The proof uses 1,724 of them: 36 for the listed states, 161 for the claims KL, 24 for the values on 7 channels, 1,499 for the entries of (vii) proved by the successor rule, one that checks the records of the other entries, two for the header of the stream and the table of listed states, and one for Harder’s certificate. The records of (vii), one of 15 characters for each entry, and the list of the entries proved by the successor rule are read in the same way as the other data. They form a separate file of the supplement. Programs that are not part of the release found them, and the proof checks every record. `#print axioms`

lists these and the three standard axioms, and nothing else. No definition of the project is marked `implemented_by` or `extern`, so the code that each computation runs is compiled from the definitions that the soundness theorems are about; the only overrides are those of Lean’s core and standard libraries. A few short functions outside the precompiled libraries, in one chunk theorem of (vii) and in the check of the records of (vii), run in Lean’s interpreter; they call the compiled checking functions.

11.4. Trust and tests. Besides Lean’s kernel, the proof relies, for its 1,724 computations, on the Lean compiler and its interpreter, the C compiler and the runtime. It reads the same certificate files and tables as the verifier, but it trusts none of the three components listed at the start of Section 10.

We compared the search of (ii) with `lb2cr` with the same tests switched off, on 1,585,789 instances (1,359,753 distinct) on 13, 11, 9 and at most 8 channels, most of them instances of cuts. The two programs gave the same answer on every instance, and on the 35,344 instances on at most 8 channels the search of (ii) also agreed with an exhaustive search over networks. In 33 tests we made one field or one record of the converted data wrong, and each change made the check of the affected part fail. These tests ran on an earlier ordering of the same data, with the same checking functions.

The build scripts regenerate the converted certificate and the tables from the released files and compare them with their pinned SHA-256 digests, and the commands of the build instructions build the proof. The build instructions give the SHA-256 digest of Harder’s certificate. The proof was built on 17 servers with 192 hardware threads each. Sixteen of them proved the entries of (vii) that need the successor rule, in 86 to 147 minutes each and about 1,120 hours of processor time in all, and the seventeenth built the rest. From the start of these builds to the output of `#print axioms` took 2 hours and 33 minutes.

12. A PROOF CHECKED BY THE KERNEL

The computations of the proof of Section 11 run as compiled code, so that proof also relies on Lean’s compiler. We wrote a third proof of the lower bound, in which Lean’s kernel checks every computation itself. It follows the same certificate, but its checking functions and its data are written so that the kernel can evaluate them, and it uses neither `native_decide` nor any other way of running compiled code. The Lean project, its data and the record of its build are archived on Zenodo [15].

12.1. The statement. The theorem has the same name and the same statement as in Section 11: `theorem Sortnet.S13_ge_45 : ∀ net : Network 13, net.Sorts → 45 ≤ net.length`

The final module checks that its type is the same Lean expression as the statement of Section 11, not only equal up to unfolding of definitions. That statement is copied verbatim and elaborated over the same file of definitions, which is byte-identical to the one of Section 11. `#print axioms` lists `propext`, `Classical.choice` and `Quot.sound`, and nothing else.

12.2. How the kernel checks the computations. Each check is a Boolean function applied to data. It is stated as `f a = true`, or as `f a = false` for the refutations, and proved by `Eq.refl` or by `decide +kernel`. The kernel accepts such a proof only after it has evaluated `f a` itself. To make this feasible, the checking functions avoid strings, arrays and hash maps; they are written with natural numbers, Booleans, lists and small inductive types. Sets of 0-1 vectors are natural numbers read as bitmaps, and the data are tables of numerals inside the declarations. The kernel reads the string constants of Section 11.3 far too slowly: a string of one kilobyte takes it about 20 seconds. This proof therefore has its own table of the values on 9 channels, written as numerals from the same released file, and it proves that every entry of this table is a lower bound. The

final theorem does not mention the table. Every data file is pinned by its SHA-256 digest. The soundness theorems of the checking functions hold for any data, so wrong data can only make a check fail.

12.3. The parts of the proof. The proof has five parts, which discharge the hypotheses of a theorem that assembles them.

- (a) $S(11) \geq 35$, from a check of each of the 12,659,079 steps of Harder’s certificate [4].
- (b) The table of values on 9 channels. Of its 3,624,738 entries, 3,344,121 follow from steps of Harder’s certificate or from other entries, as in Section 11 (vii). Each of the other 280,617 entries has a certificate: either Harder’s Huffman bound for the entry’s set, or a set of nodes, each the image of the entry’s set under a path of comparators. The set of every node has two different vectors of the same weight, so that some comparator is still needed, and the set that each useful comparator reaches from it is bounded through a step of Harder’s certificate, an entry of lower level, Harder’s Huffman bound, another node of the certificate, or the set that another comparator reaches from the same node.
- (c) A second table of values on 9 channels, which the claims KL use besides the table of (b). Its 8,977,498 entries follow from steps of Harder’s certificate, from entries of the table of (b) and from certificates of the same kind.
- (d) The 12,315,738 claims KL, grouped into 6,139,636 classes. Their instances on 11 channels, 2,249,283 in all, are refuted.
- (e) The induction over the layers. Its 31,899,147 cuts use 5,290,420 distinct instances on 13 channels, and each of them is refuted in its own declaration.

The instances are refuted by a search over configurations that is written for the kernel, with a proof that every instance it rejects is infeasible. For the instances on 11 channels it also rejects, once all numbers $k(w)$ are fixed, the choices of k for which Ω exceeds the bound of Lemma 5.5 or of Lemma 5.7. These two lemmas are proved in Lean as part of the project, and so are Lemmas 5.3, 5.6 and 5.8, which the search does not use.

12.4. Build and trust. The build comprises 102,624 Lean modules. It started from a frozen source archive and 22 data archives and ran on 40 machines. From the first module to the last it took 6 hours and 35 minutes, with a pause of about an hour before the largest part, and about 8,800 hours of processor time. A final check found no module out of date. The final checks import the compiled files of all modules, which Lean does not check again, and no independent replay of the declarations was run.

The proof relies on Lean’s kernel, as run on each module by the machine that built it, on the definitions of networks and sorting, which use only Lean’s core library, and on the three standard axioms. It does not rely on Lean’s compiler, on the data or on the programs that produced them. The build takes the compiled files of Mathlib from its official cache. We compared them with a rebuild of Mathlib and its dependencies from source, done as Mathlib’s continuous integration builds it: they are identical, except for one module whose compiled output varies from run to run, and repeated runs reproduce the cached version of that module as well.

13. CONSEQUENCES FOR LARGER NETWORKS

Proof of Corollary 1.2. For $N \geq 3$, $S(N) \geq S(N-1) + \lceil \log_2 N \rceil$ [12] and $S(N) \geq S(N-2) + P(2, N)$ [11, Lemma 2.2], and $P(2, N) = \lceil \log_2 F(N) \rceil$ for $3 \leq N \leq 32$ [11, Theorem 1.2]. The bounds $\Lambda(N)$ of [11, Corollary 1.3] are defined by

$$\Lambda(N) = \max(\Lambda(N-1) + \lceil \log_2 N \rceil, \Lambda(N-2) + \lceil \log_2 F(N) \rceil), \quad \Lambda(11) = 35, \Lambda(12) = 39.$$

Let Λ' satisfy the same recursion with $\Lambda'(12) = 39$ and $\Lambda'(13) = 45$. Then $S(N) \geq \Lambda'(N)$ for $13 \leq N \leq 32$, by Theorem 1.1 and induction. Since $\Lambda(13) = 44$, we have $\Lambda'(14) = \max(49, 48) = 49 = \Lambda(14) + 1$, and if $\Lambda' = \Lambda + 1$ at $N - 1$ and $N - 2$, then also at N . $\square$

N	13	14	15	16	17	18	19	20	21	22
$\Lambda(N) + 1$	45	49	54	58	64	69	74	79	85	90
N	23	24	25	26	27	28	29	30	31	32
$\Lambda(N) + 1$	96	101	107	112	118	124	130	136	142	148

TABLE 3. The lower bounds $S(N) \geq \Lambda(N) + 1$ of Corollary 1.2.

DATA AND CODE AVAILABILITY

The certificate, the verifier with `1b2cr`, the two tables of exact values, the logs of the verification, the specification of the certificate format, the search program and the Lean project of Section 11 with its data are archived on Zenodo as [14] (doi:10.5281/zenodo.23504361). Harder's repository [6] carries no licence, so the supplement contains our changes to his program as a patch against commit `0b5d09c`, and its build instructions fetch his code. The build instructions of the Lean project fetch Harder's certificate [4] and give its SHA-256 digest. The proof of Section 12, with its data and the record of its build, is archived on Zenodo as [15] (doi:10.5281/zenodo.23552145). This paper is doi:10.5281/zenodo.23552143. Our code is under the Apache License 2.0, and the text and the data under CC BY 4.0.

USE OF AI TOOLS

Generative AI tools were used to find and prove the results, to write the programs and the formal proofs, to search the literature, and to draft and revise the text. The sources found were checked directly. The author takes full responsibility for the content of the paper, including the parts produced with these tools.

REFERENCES

- [1] R. C. Bose and R. J. Nelson, *A sorting problem*, J. ACM **9** (1962), no. 2, 282–296, doi:10.1145/321119.321126.
- [2] M. Codish, L. Cruz-Filipe, M. Frank, and P. Schneider-Kamp, *Sorting nine inputs requires twenty-five comparisons*, J. Comput. System Sci. **82** (2016), no. 3, 551–563, doi:10.1016/j.jcss.2015.11.014.
- [3] R. W. Floyd and D. E. Knuth, *The Bose–Nelson sorting problem*, A Survey of Combinatorial Theory (J. N. Srivastava et al., eds.), North-Holland, Amsterdam, 1973, pp. 163–172, doi:10.1016/B978-0-7204-2262-7.50020-X.
- [4] J. Harder, *Certificate of the minimal size of 11-channel sorting networks*, Zenodo, 2019, dataset, doi:10.5281/zenodo.4108365.
- [5] J. Harder, *An answer to the Bose–Nelson sorting problem for 11 and 12 channels*, arXiv:2012.04400v3 [cs.DS], 2022.
- [6] J. Harder, *Lower size bounds for sorting networks*, source code, 2020, <https://github.com/jix/sortnetopt>, commit `0b5d09c`.
- [7] H. Juillé, *Evolution of non-deterministic incremental algorithms as a new approach for search in state spaces*, Proceedings of the Sixth International Conference on Genetic Algorithms (L. J. Eshelman, ed.), Morgan Kaufmann, San Francisco, 1995, pp. 351–358.
- [8] D. E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, 2nd ed., Addison-Wesley, Reading, MA, 1998.
- [9] L. de Moura and S. Ullrich, *The Lean 4 theorem prover and programming language*, Automated Deduction – CADE 28, Lecture Notes in Comput. Sci., vol. 12699, Springer, 2021, pp. 625–635, doi:10.1007/978-3-030-79876-5_37.

- [10] The mathlib Community, *The Lean mathematical library*, Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020), ACM, 2020, pp. 367–381, doi:10.1145/3372885.3373824.
- [11] T. S. Tomar, *Sorting networks on 13 inputs need at least 44 comparators*, Zenodo, 2026, doi:10.5281/zenodo.23498528.
- [12] D. C. Van Voorhis, *An improved lower bound for sorting networks*, IEEE Trans. Comput. **C-21** (1972), no. 6, 612–613, doi:10.1109/TC.1972.5009021.
- [13] T. S. Tomar, *A Lean proof that sorting networks on 13 inputs need 45 comparators*, part of the supplement [14].
- [14] T. S. Tomar, *Certificate and verifier for “Sorting networks on 13 inputs need 45 comparators”*, Zenodo, 2026, doi:10.5281/zenodo.23504361.
- [15] T. S. Tomar, *A kernel-checked Lean proof that sorting networks on 13 inputs need 45 comparators*, Zenodo, 2026, doi:10.5281/zenodo.23552145.

ORCID: <https://orcid.org/0000-0003-4668-3639>