Computing the Maximum Using (min,+) Formulas

Meena Mahajan[1], Prajakta Nimbhorkar[2], and Anuj Tawari[3]

1 **The Institute of Mathematical Sciences, HBNI, Chennai, India**
meena@imsc.res.in
2 **Chennai Mathematical Institute, India**
prajakta@cmi.ac.in
3 **The Institute of Mathematical Sciences, HBNI, Chennai, India**
anujvt@imsc.res.in

Abstract

We study computation by formulas over $(\min, +)$. We consider the computation of $\max\{x_1, \ldots, x_n\}$ over $\mathbb{N}$ as a difference of $(\min, +)$ formulas, and show that size $n + n \log n$ is sufficient and necessary. Our proof also shows that any $(\min, +)$ formula computing the minimum of all sums of $n - 1$ out of n variables must have $n \log n$ leaves; this too is tight. Our proofs use a complexity measure for $(\min, +)$ functions based on minterm-like behaviour and on the entropy of an associated graph.

1998 ACM Subject Classification F.1.1 Models of Computation; F.1.3 Complexity Measures and Classes; F.2.3 Tradeoffs between Complexity Measures

Keywords and phrases formulas, circuits, lower bounds, tropical semiring

Digital Object Identifier 10.4230/LIPIcs.MFCS.2017.74

1 Introduction

A $(\min, +)$ formula is a formula (tree) in which the leaves are labeled by variables or constants. The internal nodes are gates labeled by either min or $+$. A min gate computes the minimum value among its inputs while a $+$ gate simply adds the values computed by its inputs. Such formulas can compute any function expressible as the minimum over several linear polynomials with non-negative integer coefficients.

In this work, we consider the following problem: Suppose we are given n input variables $x_1, x_2, \ldots, x_n$ and we want to find a formula which computes the maximum value taken by these variables, $\max(x_1, x_2, \ldots, x_n)$. If variables are restricted to take non-negative integer values, t is easy to show that no $(\min, +)$ formula can compute max. Suppose now we strengthen this model by allowing minus gates as well. Now we have a very small linear sized formula: $\max(x_1, x_2, \ldots, x_n) = 0 - \min(0 - x_1, 0 - x_2, \ldots, 0 - x_n)$. It is clear that minus gates add significant power to the model of $(\min, +)$ formulas. But how many minuses do we actually need? It turns out that only one minus gate, at the top, is sufficient. Here is one such formula: (Sum of all variables) - $\min_i$ (Sum of all variables except x_i). The second expression above can be computed by a $(\min, +)$ formula of size $n \log n$ using recursion. So, we can compute max using min, $+$ and one minus gate at the top, at the cost of a slightly super-linear size. Can we do any better? We show that this simple difference formula is indeed the best we can achieve for this model.

The main motivation behind studying this question is the following question asked in [8]: Does there exist a naturally occuring function f for which $(\min, +)$ circuits are super-polynomially weaker than $(\max, +)$ circuits? There are two possibilities:

42nd International Symposium on Mathematical Foundations of Computer Science (MFCS 2017).
Editors: Kim G. Larsen, Hans L. Bodlaender, and Jean-Francois Raskin; Article No. 74; pp. 74:1–74:11
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1. Show that max can be implemented using a small $(\min, +)$ circuit.
2. Come up with an explicit function f which has *small* $(\max, +)$ circuits but requires *large* $(\min, +)$ circuits.

Since we show that no $(\min, +)$ formula (or circuit) can compute max, option 1 is ruled out. In the weaker model of formulas instead of circuits, we show that any difference of $(\min, +)$ formulas computing max should have size at least $n \log n$. This yields us a separation between $(\max, +)$ formulas and difference of $(\min, +)$ formulas.

Background

Many dynamic programming algorithms correspond to $(\min, +)$ circuits over an appropriate semiring. Notable examples include the Bellman-Ford-Moore (BFM) algorithm for the single-source-shortest-path problem (SSSP) [2, 5, 14], the Floyd-Warshall (FW) algorithm for the All-Pairs-Shortest-Path (APSP) problem [4, 18], and the Held-Karp (HK) algorithm for the Travelling Salesman Problem (TSP) [6]. All these algorithms are just recursively constructed $(\min, +)$ circuits. For example, both the BFM and the FW algorithms give $O(n^3)$ sized $(\min, +)$ circuits while the HK algorithm gives a $O(n^2 \cdot 2^n)$ sized $(\min, +)$ circuit. Matching lower bounds were proved for TSP in [7], for APSP in [8], and for SSSP in [10]. So, proving tight lower bounds for circuits over $(\min, +)$ can help us understand the power and limitations of dynamic programming. We refer the reader to [8, 9] for more results on $(\min, +)$ circuit lower bounds.

Note that algorithms for problems like computing the diameter of a graph are naturally expressed using $(\min, \max, +)$ circuits. This makes the cost of converting a max gate to a $(\min, +)$ circuit or formula an interesting measure.

A related question arises in the setting of counting classes defined by arithmetic circuits and formulas. Circuits over $\mathbb{N}$, with specific resource bounds, count accepting computation paths or proof-trees in a related resource-bounded Turing machine model defining a class $\mathcal{C}$. The counting function class is denoted $\#\mathcal{C}$. The difference of two such functions in a class $\#\mathcal{C}$ is a function in the class Diff$\mathcal{C}$. On the other hand, circuits with the same resource bounds, but over $\mathbb{Z}$, or equivalently, with subtraction gates, describe the function class Gap$\mathcal{C}$. For most complexity classes $\mathcal{C}$, a straightforward argument shows that that Diff$\mathcal{C}$ and Gap$\mathcal{C}$ coincide. See [1] for further discussion on this. In this framework, we restrict attention to computation over $\mathbb{N}$ and see that as a member of a Gap class over $(\min, +)$, max has linear-size formulas, whereas as a member of a Diff class, it requires $\Omega(n \log n)$ size.

Our results and techniques

We now formally state our results and briefly comment on the techniques used to prove them.

1. For $n \geq 2$, no $(\min, +)$ formula over $\mathbb{N}$ can compute $\max(x_1, x_2, \ldots, x_n)$. (Theorem 10)
   The proof is simple: apply a carefully chosen restriction to the variables and show that the $(\min, +)$ formula does not output the correct value of max on this restriction.
2. $\max(x_1, x_2, \ldots, x_n)$ can be computed by a difference of two $(\min, +)$ formulas with total size $n + n\lceil \log n \rceil$. More generally, the function computing the sum of the topmost k values amongst the n variables can be computed by a difference of two $(\min, +)$ formulas with total size $n + n(\lceil \log n \rceil)^{\min\{k, n-k\}}$. (Theorem 11)
   Note that the sum of the topmost k values can be computed by the following formula: (Sum of all variables) - $\min_S$ (Sum of all variables except those in S). Here S ranges over all possible subsets of $\{x_1, x_2, \ldots, x_n\}$ of cardinality $n - k$. Using recursion, we obtain the claimed size bound.

3. Let F_1, F_2 be $(\min, +)$ formulas over $\mathbb{N}$ such that $F_1 - F_2 = \max(x_1, x_2, \ldots, x_n)$. Then F_1 must have at least n leaves and F_2 at least $n \log n$ leaves. (Theorem 13)
   A major ingredient in our proof is the definition of a measure for functions computable by constant-free $(\min, +)$ formulas, and relating this measure to formula size. The measure involves terms analogous to minterms of a monotone Boolean function, and uses the entropy of an associated graph under the uniform distribution on its vertices. In the setting of monotone Boolean functions, this technique was used in in [15] to give formula size lower bounds. We adapt that technique to the $(\min, +)$ setting.
   The same technique also yields the following lower bound: Also, any $(\min, +)$ formula computing the minimum over the sums of $n-1$ variables must have at least $n \log n$ leaves. This is tight. (Lemma 12 and Corollary 18)

2 Preliminaries

2.1 Notation

Let X denote the set of variables $\{x_1, \ldots, x_n\}$. We use $\tilde{x}$ to denote $(x_1, x_2, \ldots, x_n, 1)$.

We use e_i to denote the $(n+1)$-dimensional vector with a 1 in the ith coordinate and zeroes elsewhere. For $i \in [n]$, we also use e_i to denote an assignment to the variables $x_1, x_2, \ldots, x_n$ where x_i is set to 1 and all other variables are set to 0.

▶ **Definition 1.** For $0 \leq r \leq n$, the n-variate functions Sum_n, MinSum_n^r and MaxSum_n^r are as defined below.

$$
\begin{aligned}
\mathsf{Sum}_n &= \sum_{i=1}^{n} x_i \\
\mathsf{MinSum}_n^r &= \min\left\{\sum_{i \in S} x_i \mid S \subseteq n, |S| = r\right\} \\
\mathsf{MaxSum}_n^r &= \max\left\{\sum_{i \in S} x_i \mid S \subseteq n, |S| = r\right\}
\end{aligned}
$$

Note that MinSum_n^0 and MaxSum_n^0 are the constant function 0, and MinSum_n^1 and MaxSum_n^1 are just the min and max respectively.

▶ **Observation 2.** *For $1 \leq r < n$, $\mathsf{MinSum}_n^n = \mathsf{MaxSum}_n^n = \mathsf{Sum}_n = \mathsf{MinSum}_n^r + \mathsf{MaxSum}_n^{n-r}$.*

2.2 Formulas

A $(\min, +)$ formula is a directed tree. Each leaf of a formula has a label from $X \cup \mathbb{N}$; that is, it is labeled by a variable x_i or a constant $\alpha \in \mathbb{N}$. Each internal node has exactly two children and is labeled by one of the two operations min or $+$. The output node of the formula computes a function of the input variables in the natural way. The input nodes of a formula are also referred to as gates.

If all leaves of a formula are labeled from X, we say that the formula is constant-free.

A $(\min, +, -)$ formula is similarly defined; the operation at an internal node may also be $-$, in which case the children are ordered and the node computes the difference of their values.

We define the size of a formula as the number of leaves in the formula. For a formula F, we denote by $L(F)$ its size, the number of leaves in it. For a function f, we denote by $L(f)$ the smallest size of a formula computing f. By $L_{cf}(f)$ we denote the smallest size of a constant-free formula computing f.

2.3 Graph Entropy

The notion of the entropy of a graph or hypergraph, with respect to a probability distribution on its vertices, was first defined by Körner in [11]. In that and subsequent works (e.g. [12, 13, 3, 15]), equivalent characterizations of graph entropy were established and are often used now as the definition itself, see for instance [16, 17]. In this paper, we use graph entropy only with respect to the uniform distribution, and simply call it graph entropy. We use the following definition, which is exactly the definition from [17] specialised to the uniform distribution.

▶ **Definition 3.** Let G be a graph with vertex set $V(G) = \{1, \ldots, n\}$.

The *vertex packing polytope* $VP(G)$ of the graph G is the convex hull of the characteristic vectors of independent sets of G.

The entropy of G is defined as

$$H(G) = \min_{\vec{a} \in VP(G)} \sum_{i=1}^{n} \frac{1}{n} \log \frac{1}{a_i} \quad .$$

It can easily be seen that $H(G)$ is a non-negative real number, and moreover, $H(G) = 0$ if and only if G has no edges. We list non-trivial properties of graph entropy that we use.

▶ **Lemma 4** ([12, 13]). *Let $F = (V, E(F))$ and $G = (V, E(G))$ be two graphs on the same vertex set. The following hold:*

1. ***Monotonocity.*** *If $E(F) \subseteq E(G)$, then $H(F) \leq H(G)$*
2. ***Subadditivity.*** *Let Q be the graph with vertex set V and edge set $E(F) \cup E(G)$. Then $H(Q) \leq H(F) + H(G)$.*

▶ **Lemma 5** (see for instance [16, 17]). *The following hold:*

1. *Let K_n be the complete graph on n vertices. Then $H(K_n) = \log n$.*
2. *Let G be a graph on n vertices, whose edges induce a bipartite graph on m (out of n) vertices. Then $H(G) \leq \frac{m}{n}$.*

3 Transformations and Easy bounds

We consider the computation of $\max\{x_1, \ldots, x_n\}$ over $\mathbb{N}$ using $(\min, +)$ formulas.

To start with, we describe some properties of $(\min, +)$ formulas that we use repeatedly. The first property, Proposition 7 below, is expressing the function computed by a formula as a depth-2 polynomial where $+$ plays the role of multiplication and min plays the role of addition. The next properties, Proposition 8 and 9 below, deal with removing redundant sub-expressions created by the constant zero or moving common parts aside.

▶ **Definition 6.** Let F be a $(\min, +)$ formula with leaves labeled from $X \cup \mathbb{N}$. For each gate $v \in F$, we construct a set $S_v \subseteq \mathbb{N}^{n+1}$ as described below.

We construct the sets inductively based on the depth of v.

1. Case 1. v is a leaf labeled α for some $\alpha \in \mathbb{N}$. Then $S_v = \{\alpha \cdot e_{n+1}\}$. (Recall, e_i is the unit vector with 1 at the ith coordinate and zero elsewhere).
2. Case 2: v is a leaf labeled x_i for some $i \in [n]$. Then $S_v = \{e_i\}$.
3. Case 3: $v = \min\{u, w\}$. Then $S_v = S_u \cup S_w$.
4. Case 4: $v = u + w$. Then $S_v = \{\tilde{a} + \tilde{b} \mid \tilde{a} \in S_u,\ \tilde{b} \in S_w\}$ (coordinate-wise addition).

Let r be the output gate of F. We denote by $S(F)$ the set S_r so constructed.

It is straightforward to see that if F has no constants (so Case 1 is never invoked), then a_{n+1} remains 0 throughout the construction of the sets S_v. Hence if F is constant-free, then for each $\tilde{a} \in S(F)$, $a_{n+1} = 0$.

By construction, the set $S(F)$ describes the function computed by F. Thus we have the following:

▶ **Proposition 7.** *Let F be a formula with* min *and* + *gates, with leaves labeled by elements of $\{x_1, \ldots, x_n\} \cup \mathbb{N}$. For each gate $v \in F$, let f_v denote the function computed at v.*

Then $f_v = \min\{\langle \tilde{a} \cdot \tilde{x} \rangle \mid \tilde{a} \in S_v\}$.

The following proposition is an easy consequence of the construction in Definition 6.

▶ **Proposition 8.** *Let F be a $(\min, +)$ formula over $\mathbb{N}$. Let G be the formula obtained from F by replacing all constants by the constant 0. Let H be the constant-free formula obtained from G by eliminating 0s from G through repeated replacements of $0 + A$ by A, $\min\{0, A\}$ by 0. Then*
1. *$L(H) \leq L(G) = L(F)$,*
2. *$S(G) = \{\tilde{b} \mid b_{n+1} = 0, \exists \tilde{a} \in S(F), \forall i \in [n], a_i = b_i\}$, and*
3. *G and H compute the same function $\min\{\langle \tilde{b} \cdot \tilde{x} \rangle \mid \tilde{b} \in S(G)\}$.*

(Note: It is not the claim that $S(G) = S(H)$. Indeed, this may not be the case. eg. let $F = x + \min\{1, x+y\}$. Then $S(F) = \{101, 210\}$, $S(G) = \{100, 210\}$, $S(H) = \{100\}$, However, the functions computed are the same.)

The next proposition shows how to remove "common" contributors to $S(F)$ without increasing the formula size.

▶ **Proposition 9.** *Let F be a $(\min, +)$ formula computing a function f.*

If, for some $i \in [n]$, $a_i > 0$ for every $\tilde{a} \in S(F)$, then $f - x_i$ can be computed by a $(\min, +)$ formula F' of size at most size(F).

If $a_{n+1} > 0$ for every $\tilde{a} \in S(F)$, then $f - 1$ can be computed by a $(\min, +)$ formula F' of size at most size(F).

In both cases, $S(F') = \{\tilde{b} \mid \exists \tilde{a} \in S(F), \tilde{b} = \tilde{a} - e_i\}$.

Proof. First consider $i \in [n]$. Let X be the subset of nodes in F defined as follows:

$$X = \{v \in F \mid \forall \tilde{a} \in S_v : a_i > 0\}$$

Clearly, the output gate r of F belongs to X. By the construction of the sets S_v, whenever a min node v belongs to X, both its children belong to X, and whenever a + node belongs to X, at least one of its children belongs to X. We pick a set $T \subseteq X$ as follows. Include r in T. For each min node in T, include both its children in T. For each + node in T, include in T one child that belongs to X (if both children are in X, choose any one arbitrarily). This gives a sub-formula of F where all leaves are labeled x_i. Replace these occurrences of x_i in F by 0 to get formula F'. It is easy to see that $S(F') = \{\tilde{a} - e_i \mid \tilde{a} \in S\}$. Hence F' computes $f - x_i$.

For $i = a_{n+1}$, the same process as above yields a subformula where each leaf is labeled by a positive constant. Subtracting 1 from the constant at each leaf in T gives the formula computing $f - 1$. ◀

It is intuitively clear that no $(\min, +)$ formula can compute max. A formal proof using Proposition 7 appears below.

▶ **Theorem 10.** *For $n \geq 2$, no (min, +) formula over $\mathbb{N}$ can compute $\max\{x_1, \ldots, x_n\}$.*

Proof. Suppose, to the contrary, some formula C computes max. Then its restriction D to $x_1 = X$, $x_2 = Y$, $x_3 = x_4 = \ldots = x_n = 0$, correctly computes $\max\{X, Y\}$. Since all leaves of D are labeled from $\{x_1, x_2\} \cup \mathbb{N}$, the set $S(D)$ is a set of triples. Let $S \subseteq \mathbb{N}^3$ be this set. For all $X, Y \in \mathbb{N}$, $\max\{X, Y\}$ equals $E(X, Y) = \min\{AX + BY + C \mid (A, B, C) \in S\}$.

Let K denote the maximum value taken by C in any triple in S. If for some $B, C \in \mathbb{N}$, the triple $(0, B, C)$ belongs to S, then $E(K + 1, 0) \leq C \leq K < K + 1 = \max\{0, K + 1\}$. So for all $(A, B, C) \in S$, $A \neq 0$, so $A \geq 1$. Similarly, for all $(A, B, C) \in S$, $B \geq 1$. Hence for all $(A, B, C) \in S$, $A + B \geq 2$.

Now $E(1, 1) = \min\{A + B + C \mid (A, B, C) \in S\} \geq 2 > 1 = \max\{1, 1\}$. So $E(X, Y)$ does not compute $\max(X, Y)$ correctly. ◀

However, if we also allow the subtraction operation at internal nodes, it is very easy to compute the maximum in linear size; $\max(x_1, \ldots, x_n) = -\min\{-x_1, -x_2, \ldots, -x_n\}$. Here $-a$ is implemented as $0 - a$, and if we allow only variables, not constants, at leaves, we can compute $-a$ as $(x_1 - x_1) - a$.

Thus the subtraction operation adds significant power. How much? Can we compute the maximum with very few subtraction gates? It turns out that the max function can be computed as the difference of two $(\min, +)$ formulas. Equivalently, there is a $(\min, +, -)$ formula with a single $-$ gate at the root, that computes the max function. This formula is not linear in size, but it is not too big either; we show that it has size $\mathrm{O}(n \log n)$. A simple generalisation allows us to compute the sum of the largest k values.

▶ **Theorem 11.** *For each $n \geq 1$, and each $0 \leq k \leq n$, the function MaxSum_n^k can be computed by a difference of two $(\min, +)$ formulas with total size $n + n(\lceil \log n \rceil)^{\min\{k, n-k\}}$.*

In particular, the function $\max\{x_1, \ldots, x_n\}$ can be computed by a difference of two $(min, +)$ formulas with total size $n + n\lceil \log n \rceil$.

Proof. Note that $\mathsf{MaxSum}_n^k = \mathsf{Sum}_n - \mathsf{MinSum}_n^{n-k}$. Lemma 12 below shows that MinSum_n^{n-k} can be computed by a formula of size $n(\lceil \log n \rceil)^{\min\{k, n-k\}}$ for $0 \leq k \leq n$. Since Sum_n can be computed by a formula of size n, the claimed upper bound for MaxSum_n^k follows. ◀

▶ **Lemma 12.** *For all n, k such that $n \geq 1$ and $0 \leq k < n$, the functions MinSum_n^k, MinSum_n^{n-k} can be computed by a $(\min, +)$ formula of size $n(\lceil \log n \rceil)^k$.*

Hence the functions MinSum_n^k, MinSum_n^{n-k} can be computed by $(\min, +)$ formulas of size $n(\lceil \log n \rceil)^{\min\{k, n-k\}}$.

Proof. We prove the upper bound for MinSum_n^{n-k}. The bound for MinSum_n^k follows from an essentially identical argument.

We prove this by induction on k.

- Base Case: $k = 0$. For every $n \geq 1$, $\mathsf{MinSum}_n^{n-k} = \mathsf{Sum}_n$ and can be computed with size n.
- Inductive Hypothesis: For all $k' < k$, and all $n > k'$, $\mathsf{MinSum}_n^{n-k'}$ can computed in size $n(\lceil \log n \rceil)^{k'}$.
- Inductive Step: We want to prove the claim for k, where $k \geq 1$, and for all $n > k$. We proceed by induction on n.
  - Base Case: $n = k + 1$. $\mathsf{MinSum}_n^{n-k} = \mathsf{MinSum}_n^1$ is the minimum of the n variables, and can be computed in size n.
  - Inductive Hypothesis: For all $k < m < n$, MinSum_m^{m-k} can be computed in size $m(\lceil \log m \rceil)^k$.

- Inductive Step: Let $m' = \lfloor n/2 \rfloor$, $m'' = \lceil n/2 \rceil$, Let X, X_l, X_r denote the sets of variables $\{x_1, \ldots, x_n\}$, $\{x_1, \ldots, x_{m'}\}$, $\{x_{m'+1}, \ldots, x_n\}$. Note that $|X_l| = m'$, $|X_r| = m''$, $m' + m'' = n$. Let p denote $\lceil \log n \rceil$. Note that $\lceil \log m' \rceil = \lceil \log m'' \rceil = p - 1$.
  To compute MinSum_n^{n-k} on X, we first compute, for various values of t, $\mathsf{MinSum}_{m'}^{m'-t}$ on X_l, $\mathsf{MinSum}_{m''}^{m''-(k-t)}$ on X_r, and add them up. We then take the minimum of these sums. Note that if $m' = t$ or $m'' = k - t$, then that summand is simply 0 and we only compute the other summand. Now $\mathsf{MinSum}_n^{n-k}(X)$ can be computed as

  $$\min \left\{ \mathsf{MinSum}_{m'}^{m'-t}(X_l) + \mathsf{MinSum}_{m''}^{m''-(k-t)}(X_r) \mid \max\{0, k - m''\} \leq t \leq \min\{m', k\} \right\}$$

  For all the sub-expressions appearing in the above construction, we can use inductively constructed formulas. Using the inductive hypotheses (both for $t < k$ and for $t = k$, $m'' < n$), we see that the number of leaves in the resulting formula is given by

  $$\begin{aligned} & \sum_{t=\max\{0,k-m''\}}^{\min\{m',k\}} \left[m'(p-1)^t + m''(p-1)^{k-t} \right] \\ \leq\ & \sum_{t=0}^{k} \left[m'(p-1)^t + m''(p-1)^{k-t} \right] \\ =\ & \left[\sum_{t=0}^{k} m'(p-1)^t \right] + \left[\sum_{t=0}^{k} m''(p-1)^t \right] \\ =\ & (m' + m'') \left[\sum_{t=0}^{k} (p-1)^t \right] \\ \leq\ & n\left[(p-1)+1\right]^k = np^k \end{aligned}$$ ◀

In the rest of this paper, our goal is to prove a matching lower bound for the max function. Note that the constructions in Theorem 11 and Lemma 12 yield formulas that do not use constants at any leaves. Intuitively, it is clear that if a formula computes the maximum correctly for all natural numbers, then constants cannot help. So the lower bound should hold even in the presence of constants, and indeed our lower bound does hold even if constants are allowed.

4 The main lower bound

In this section, we prove the following theorem:

▶ **Theorem 13.** *Let F_1, F_2 be $(min, +)$ formulas over $\mathbb{N}$ such that $F_1 - F_2 = \max(x_1, \ldots, x_n)$. Then $L(F_1) \geq n$, and $L(F_2) \geq n \log n$.*

The proof proceeds as follows: we first transform F_1 and F_2 over a series of steps to formulas G_1 and G_2 no larger than F_1 and F_2, such that $G_1 - G_2$ equals $F_1 - F_2$ and hence still computes max, and G_1 and G_2 have some nice properties. These properties immediately imply that $L(F_1) \geq L(G_1) \geq n$. We further transform G_2 to a constant-free formula H no larger than G_2. We then define a measure for functions computable by constant-free $(\min, +)$ formulas, relate this measure to formula size, and use the properties of G_2 and H to show that the function h computed by H has large measure and large formula size.

Transformation 1. For $b \in \{1,2\}$, let S_b denote the set $S(F_b)$. For $i \in [n+1]$, let A_i be the minimum value appearing in the ith coordinate in any tuple in $S_1 \cup S_2$. Let $\tilde{A}$ denote the tuple $(A_1, \ldots, A_n, A_{n+1})$. By repeatedly invoking Proposition 9, we obtain formulas G_b computing $F_b - \langle \tilde{A} \cdot \tilde{x} \rangle$, with $L(G_b) \le L(F_b)$. For $b \in \{1,2\}$, let T_b denote the set $S(G_b)$.

We now establish the following properties of G_1 and G_2.

▶ **Lemma 14.** *Let F_1, F_2 be $(\min,+)$ formulas such that $F_1 - F_2$ computes $\max$. Let G_1, G_2 be obtained as described above. Then*
1. $L(G_1) \le L(F_1)$, $L(G_2) \le L(F_2)$,
2. $\max(X) = F_1 - F_2 = G_1 - G_2$,
3. *For every $i \in [n]$, for every $\tilde{a} \in T_1$, $a_i > 0$. Hence $L(G_1) \ge n$.*
4. *For every $i \in [n]$, there exists $\tilde{a} \in T_2$, $a_i = 0$.*
5. *There exist $\tilde{a} \in T_1$, $\tilde{b} \in T_2$, $a_{n+1} = b_{n+1} = 0$.*
6. *For every $i, j \in [n]$ with $i \ne j$, for every $\tilde{a} \in T_2$, $a_i + a_j > 0$.*

Proof. 1. This follows from proposition 9.
2. Obvious.
3. Suppose for some $\tilde{a} \in T_1$ and for some $i \in [n]$, $a_i = 0$. Consider the input assignment $\tilde{d}$ where $d_i = 1 + a_{n+1}$ and $d_j = 0$ for $j \in [n] \setminus \{i\}$. Then $\max\{d_1, \ldots, d_n\} = 1 + a_{n+1}$. However, $\langle \tilde{a} \cdot \tilde{d} \rangle = a_{n+1}$. Therefore on input $\tilde{d}$, $G_1(\tilde{d}) \le a_{n+1}$. Since $G_2 \ge 0$ on all assignments, we get $G_1(\tilde{d}) - G_2(\tilde{d}) \le a_{n+1} < \max(\tilde{d})$, contradicting the assumption that $G_1 - G_2$ computes max.
4. This follows from the previous point and the choice of A_i for each i.
5. From the choice of A_{n+1}, we know that there is an $\tilde{a}$ in $T_1 \cup T_2$ with $a_{n+1} = 0$. Suppose there is such a tuple in exactly one of the sets T_1, T_2. Then exactly one of $G_1(\tilde{0})$, $G_2(\tilde{0})$ equals 0, and so $G_1 - G_2$ does not compute $\max(\tilde{0})$.
6. Suppose to the contrary, some $\tilde{a} \in T_2$ has $a_i = a_j = 0$. Consider the input assignment $\tilde{d}$ where $d_i = d_j = 1 + a_{n+1}$ and $d_k = 0$ for $k \in [n] \setminus \{i, j\}$. Then $\max\{d_1, \ldots, d_n\} = 1 + a_{n+1}$. Since every x_k figures in every tuple of T_1, $G_1(\tilde{d}) \ge d_i + d_j = 2a_{n+1} + 2$. But $G_2(\tilde{d}) \le a_{n+1}$. Hence $G_1(\tilde{d}) - G_2(\tilde{d})$ does not compute $\max(\tilde{d})$.

◀

We have already shown above that $L(F_1) \ge L(G_1) \ge n$. Now the more tricky part: we need to lower bound $L(G_2)$.

Transformation 2. Let H' be the formula obtained by simply replacing every constant in G_2 by 0. Let H be the constant-free formula obtained from H' by eliminating the zeroes, repeatedly replacing $0 + A$ by A, $\min\{0, A\}$ by 0. Let h be the function computed by H. Then, $L_{cf}(h) \le L(H) \le L(H') = L(G_2) \le L(F_2)$. It thus suffices to show that $L_{cf}(h) \ge n \log n$. To this end, we define a complexity measure μ, relate it to constant-free formula size, and show that it is large for the function h.

▶ **Definition 15.** For an n-variate function f computable by a constant-free $(\min,+)$ formula, we define

$$\begin{aligned} (f)_1 &= \{i \mid f(e_i) \ge 1, f(0) = 0\}. \\ (f)_2 &= \{(i,j) \mid f(e_i + e_j) \ge 1, f(e_i) = 0, f(e_j) = 0\}. \end{aligned}$$

We define $G(f)$ to be the graph whose vertex set is $[n]$ and edge set is $(f)_2$.

The measure μ for function f is defined as follows:

$$\mu = \frac{|(f)_1|}{n} + H(G(f))$$

The following lemma relates $\mu(f)$ with $L(f)$. This relation has been used before, see for instance [15] for applications to monotone Boolean circuits. Since we have not seen an application in the setting of $(\min, +)$ formulas, we (re-)prove this in detail here; however, it is really the same proof.

▶ **Lemma 16.** *Let f be an n-variate function computable by a constant-free $(\min, +)$ formula. Then $L_{cf}(f) \geq n \cdot \mu(f)$.*

Proof. The proof is by induction on the depth of a witnessing formula F that computes f and has $L_{cf}(F) = L_{cf}(f)$.

Base case F is an input variable, say x_i. Then $(f)_1 = \{x_i\}$, and $G(f)$ is the empty graph, so $\mu(f) = \frac{1}{n}$. Hence $1 = L_{cf}(f) = n\mu(f)$.

Inductive step F is either $F' + F''$ or $\min\{F', F''\}$ for some formulas F', F'' computing functions f', f'' respectively. Since F is an optimal-size formula for f, F' and F'' are optimal-size formulas for f' and f'' as well. So $L_{cf}(f) = L(F) = L(F') + L(F'') = L_{cf}(f') + L_{cf}(f'')$.

Case a. $F = F' + F''$. Then $(f)_1 = (f')_1 \cup (f'')_1$ and $G(f) \subseteq G(f') \cup G(f'')$. Hence,

$$\begin{aligned}
\mu(f) &\leq \frac{|(f')_1 \cup (f'')_1|}{n} + H(G(f') \cup G(f'')) && \text{(Lemma 4)}\\
&\leq \frac{|(f')_1|}{n} + \frac{|(f'')_1|}{n} + H(G(f')) + H(G(f'')) && \text{(Lemma 4)}\\
&= \mu(f') + \mu(f'')\\
&\leq \frac{1}{n} \cdot L_{cf}(f') + \frac{1}{n} \cdot L_{cf}(f'') && \text{(Induction)}\\
&= \frac{1}{n} \cdot L_{cf}(f) && (L_{cf}(f) = L_{cf}(f') + L_{cf}(f''))
\end{aligned}$$

Case b. $F = \min(F', F'')$. Let $(f')_1 = A$ and $(f'')_1 = B$. Then $(f)_1 = A \cap B$ and $G(f) \subseteq G(f') \cup G(f'') \cup G(A \setminus B, B \setminus A)$. Here, $G(P, Q)$ denotes the bipartite graph G with parts P and Q. Hence,

$$\begin{aligned}
\mu(f) &\leq \frac{1}{n}(|A \cap B|) + H(G(f') \cup G(f'') \cup G(A \setminus B, B \setminus A)) && \text{(Lemma 4)}\\
&\leq \frac{1}{n}(|A \cap B|) + H(G(f')) + H(G(f'')) + H(G(A \setminus B, B \setminus A)) && \text{(Lemma 4)}\\
&\leq \frac{1}{n}(|A \cap B|) + H(G(f')) + H(G(f'')) + \frac{1}{n}(|A \setminus B| + |B \setminus A|) && \text{(Lemma 5)}\\
&\leq \frac{1}{n}(|A| + |B|) + H(G(f')) + H(G(f''))\\
&= \mu(f') + \mu(f'')\\
&\leq \frac{1}{n} \cdot L_{cf}(f') + \frac{1}{n} \cdot L_{cf}(f'') && \text{(Induction)}\\
&= \frac{1}{n} \cdot L_{cf}(f) && (L_{cf}(f) = L_{cf}(f') + L_{cf}(f''))
\end{aligned}$$

Hence, $\mu(f) \leq \frac{1}{n} \cdot L_{cf}(f)$. ◀

Using this measure, we can now show the required lower bound.

▶ **Lemma 17.** *For the function h obtained after Transformation 2, $\mu(h) \geq \log n$.*

Proof. Recall that we replaced constants in G_2 by 0 to get H', then eliminated the 0s to get constant-free H computing h. By Proposition 8, we know that $S(H') = \{\tilde{b} \mid b_{n+1} = 0, \exists \tilde{a} \in T_2, a_i = b_i \forall i \in [n]\}$ and that $h = \min\{\tilde{x} \cdot \tilde{b} \mid \tilde{b} \in S(H')\}$.

From item 4 in Lemma 14, it follows that $(h)_1 = \emptyset$. (For every i, there is a $\tilde{b} \in S(H')$ with $b_i = 0$. So $h(e_i) \leq \langle e_i \cdot \tilde{b} \rangle = 0$.)

Since $(h)_1$ is empty, $(i, j) \in G(h)$ exactly when $h(e_i + e_j) \geq 1$. From item 6 in Lemma 14, it follows that every pair (i, j) is in $G(h)$. Thus $G(h)$ is the complete graph K_n.

From Lemma 5 we conclude that $\mu(h) = \log n$. ◀

Lemmas 16 and 17 imply that $L_{cf}(h) \geq n \log n$. Since $L_{cf}(h) \leq L(H) \leq L(H') = L(G_2) \leq L(F_2)$, we conclude that $L(F_2) \geq n \log n$.

This completes the proof of Theorem 13.

A major ingredient in this proof is using the measure μ. This yields lower bounds for constant-free formulas. For functions computable in a constant-free manner, it is hard to see how constants can help. However, to transfer a lower bound on $L_{cf}(f)$ to a lower bound on $L(f)$, this idea of "constants cannot help" needs to be formalized. The transformations described before we define μ do precisely this.

For the MinSum_n^{n-1} function, applying the measure technique immediately yields the lower bound $L_{cf}(\mathsf{MinSum}_n^{n-1}) \geq n \log n$. Transferring this lower bound to formulas with constants is a corollary of our main result, and with it we see that the upper bound from Lemma 12 is tight for MinSum_n^{n-1}.

▶ **Corollary 18.** *Any* $(\min, +)$ *formula computing* MinSum_n^{n-1} *must have size at least* $n \log n$.

Proof. Let F be any formula computing MinSum_n^{n-1}. Applying Theorem 13 to $F_1 = x_1 + \ldots + x_n$ and $F_2 = F$, we obtain $L(F) \geq n \log n$. ◀

5 Discussion

Our results hold when variables take values from $\mathbb{N}$. In the standard $(\min, +)$ semi-ring, the value ∞ is also allowed, since it serves as the identity for the min operation. The proof of our main result Theorem 13 does not carry over to this setting. The main stumbling block is the removal of the "common" part of $S(F)$. However, if we allow ∞ as a value that a variable can take, but not as a constant appearing at a leaf, then the lower bound proof still seems to work. However, the upper bound no longer works; while taking a difference, what is $\infty - \infty$?

Apart from the many natural settings where the tropical semiring $(\min, +, \mathbb{N} \cup \{\infty\}, 0, \infty)$ crops up, it is also interesting because it can simulate the Boolean semiring for monotone computation. The mapping is straightforward: $0, 1, \vee, \wedge$ in the Boolean semiring are replaced by $\infty, 0, \min, +$ respectively in the tropical semiring. Proving lower bounds for $(\min, +)$ formulas could be easier than for monotone Boolean formulas because the $(\min, +)$ formula has to compute a function correctly at all values, not just at $0, \infty$. Hence it would be interesting to extend our lower bound to this setting with ∞ as well.

Our transformations crucially use the fact that there is a minimum element, 0. Thus, we do not see how to extend these results to computations over integers. It appears that we will need to include $-\infty$, and since we are currently unable to handle even $+\infty$, there is already a barrier.

The lower bound method uses graph entropy which is always bounded above by $\log n$. Thus this method cannot give a lower bound larger than $n \log n$. It would be interesting to obtain a modified technique that can show that all the upper bounds in Theorem 11 and

Lemma 12 are tight. It would also be interesting to find a direct combinatorial proof of our lower bound result, without using graph entropy.

References

1 Eric Allender. Arithmetic circuits and counting complexity classes. In Jan Krajicek, editor, *Complexity of Computations and Proofs*, Quaderni di Matematica Vol. 13, pages 33–72. Seconda Universita di Napoli, 2004. An earlier version appeared in the Complexity Theory Column, SIGACT News 28, 4 (Dec. 1997) pp. 2-15.
2 Richard Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1956.
3 Imre Csiszár, János Körner, László Lovász, Katalin Marton, and Gábor Simonyi. Entropy splitting for antiblocking corners and perfect graphs. *Combinatorica*, 10(1):27–40, 1990.
4 Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345, 1962.
5 Lester R Ford Jr. Network flow theory. Technical Report P-923, Rand Corporation, 1956.
6 Michael Held and Richard M Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.
7 Mark Jerrum and Marc Snir. Some exact complexity results for straight-line computations over semirings. *Journal of the ACM (JACM)*, 29(3):874–897, 1982.
8 Stasys Jukna. Lower bounds for tropical circuits and dynamic programs. *Theory of Computing Systems*, 57(1):160–194, 2015.
9 Stasys Jukna. Tropical complexity, Sidon sets, and dynamic programming. *SIAM Journal on Discrete Mathematics*, 30(4):2064–2085, 2016.
10 Stasys Jukna and Georg Schnitger. On the optimality of Bellman–Ford–Moore shortest path algorithm. *Theoretical Computer Science*, 628:101–109, 2016.
11 János Körner. Coding of an information source having ambiguous alphabet and the entropy of graphs. In *Transactions of 6th Prague Conference on Information Theory*, pages 411–425. Academia, Prague, 1973.
12 János Körner. Fredman-Komlós bounds and information theory. *SIAM. J. on Algebraic and Discrete Methods*, 7(4):560–570, 1986.
13 János Körner and Katalin Marton. New bounds for perfect hashing via information theory. *European Journal of Combinatorics*, 9(6):523–530, 1988.
14 Edward F Moore. *The shortest path through a maze.* Bell Telephone System., 1959.
15 Ilan Newman and Avi Wigderson. Lower bounds on formula size of boolean functions using hypergraph entropy. *SIAM Journal on Discrete Mathematics*, 8(4):536–542, 1995.
16 Gábor Simonyi. Graph entropy: A survey. *Combinatorial Optimization*, 20:399–441, 1995.
17 Gábor Simonyi. Perfect graphs and graph entropy: An updated survey. In *Perfect Graphs*, pages 293–328. John Wiley and Sons, 2001.
18 Stephen Warshall. A theorem on Boolean matrices. *Journal of the ACM (JACM)*, 9(1):11–12, 1962.