The Complexity of Unary Subset Sum

Nutan Limaye[1], Meena Mahajan[2], and Karteek Sreenivasaiah[2]

[1] Indian Institute of Technology, Bombay, India. nutan@cse.iitb.ac.in.
[2] The Institute of Mathematical Sciences, Chennai, India.
{meena,karteek}@imsc.res.in.

Abstract. Given a stream of n numbers and a number B, the subset sum problem deals with checking whether there exists a subset of the stream that adds to exactly B. The unary subset sum problem, USS, is the same problem when the input is encoded in unary. We prove that any p-pass randomized algorithm computing USS with error at most $1/3$ must use space $\Omega(\frac{B}{p})$. For $p \leq B$, we give a randomized p-pass algorithm that computes USS with error at most $1/3$ using space $\tilde{O}(\frac{nB}{p})$. We give a deterministic one-pass algorithm which given an input stream and two parameters B, ϵ, decides whether there exist a subset of the input stream that adds to a value in the range $[(1-\epsilon)B, (1+\epsilon)B]$ using space $O\left(\frac{\log B}{\epsilon}\right)$. We observe that USS is monotone (under a suitable encoding) and give a monotone NC^2 circuit for USS. We also show that any circuit using ε-approximator gates for USS under this encoding needs $\Omega(n/\log n)$ gates to compute the Disjointness function.

1 Introduction

The Subset Sum problem is defined as follows: Given a number $B \in \mathbb{N}$ and a sequence of numbers $a_1, \ldots, a_n \in \mathbb{N}$, decide whether there is a subset $S \subseteq [n]$ such that $\sum_{i \in S} a_i = S$. This problem is one of the earliest problems shown to be NP-complete and can be found in [GJ79].

The Unary Subset Sum problem (USS) is the same problem, but with the input numbers given in unary (for instance $1^B 0 1^{a_1} 0 \ldots 1^{a_n}$.)

USS is known to be in P. In [EJT10], Elberfeld, Jakoby and Tantau showed a powerful meta-theorem for obtaining logspace upper bounds, and used it to conclude that USS is even in LogSpace. In [Kan10] Daniel Kane gave a considerably simplified logspace algorithm. Improving upon this further, in [EJT12] it is shown that under appropriate encodings USS has polynomial-sized formulas and hence is in NC^1. They also show that USS has polynomial-sized constant-depth formulas using MAJORITY and NOT gates and hence is in TC^0. On the other hand, it can be shown easily that MAJORITY reduces to computing USS and hence USS is TC^0-hard. Thus USS is TC^0-complete.

A natural question to ask at this point is: Is it crucial to have access to the entire input at any time in order to be able to solve USS in LogSpace? In other words: how hard, with regard to space, is USS when the inputs are, say, read in a stream? We study the complexity of USS and related questions in different models from this perspective.

In Section 2, we consider the space complexity of USS in the streaming world. The input numbers arrive in a stream, and we want to design a small space algorithm that makes one, or maybe a few, passes over the input stream and decides the instance. We assume that B is the first number in the stream. We assume that the stream contains integers in the range $1, \ldots, B$. We use lower bounds from communication complexity to show that any randomized p-pass streaming algorithm for USS that makes error bounded by $1/3$ must use $\Omega(\frac{B}{p})$ space (Theorem 1). Also, by modifying the algorithm from [Kan10], we obtain, for each $p \leq B$, a randomized streaming algorithm for USS that makes p passes over the input, uses space $O((\frac{nB}{p}) \log^2(Bn))$, and on each input errs with probability at most $1/3$ (Theorem 2).

In Section 3, we consider the complexity of approximating USS. We say that an algorithm ϵ-approximates USS if it outputs Yes exactly when there is a subset $S \subseteq [n]$ such that $|B - \sum_{i \in S} a_i| < \epsilon B$. Note that this problem is not necessarily easier than the exact version of USS since the exact version is not an optimization problem. i.e., if the answer to exact-USS on an input instance is NO, this does not mean that the answer to the approximate version is a NO. However, there is a fully polynomial time approximation scheme (FPTAS) for approximate subset sum, where the goal is to find a subset S such that $B - \epsilon B \leq \sum_{i \in S} a_i \leq B$ (see for instance [CLRS09]). But this is not efficient in terms of streaming space, even when the input is given in unary. We give a simple deterministic 1-pass streaming algorithm that takes input $\epsilon, B, \tilde{a}$ and ϵ-approximates USS on the stream $\tilde{a}$ using space $O(\frac{\log B}{\epsilon})$ (Theorem 3). We also show that this is almost tight (Lemma 3).

In Section 4, we consider the monotone circuit complexity of USS. Note that USS is naturally monotone in the following sense: if the number of occurrences of a number i in the stream is increased, a Yes instance remains a Yes instance. To model this monotonicity, we consider the following encoding of USS: For each positive integer B, the input consists of the frequency of each number in the stream in unary. That is, an instance consists of B blocks of B bits each, where the ith block w_i has as many 1s as the number of occurrences m_i of the number i in the stream. Thus, the input records the multiplicity of each number in $[B]$ (without loss of generality, no multiplicity exceeds B). Call this problem the multiplicity-USS problem, mUSS. We show, by a monotone reduction to reachability, that this problem has monotone circuits of polynomial size and $O(\log^2 B)$ depth (Theorem 5). The circuit we construct can also be used to solve the approximate version of USS.

A related question is: How powerful are ε-approximators when used as gate primitives in circuits? We explore this direction in section 5. We observe that ε-approximators for mUSS (we call them ApproxUSS gates) are at least as powerful as threshold gates. Using a technique introduced by Nisan in [Nis94], we also show that any circuit computing the Disjointness function using ε-mUSS gates requires $\Omega(n/\log n)$ gates. However we have not been able to compare ApproxUSS gates explicitly with Linear Threshold gates.

2 Exact USS in Streaming Model

In the communication problem corresponding to USS, both Alice and Bob are given an integer B. Further, each of them has a multiset of numbers and they have to determine if there is a sub-multiset of numbers among the union of their multisets that adds to B. The goal is to minimize the number of bits exchanged between Alice and Bob. Additionally, there may be constraints on how often the communication exchange changes direction (the number of rounds).

A standard lower bound technique (see [AMS99]) shows that a p-pass space $O(s)$ streaming algorithm yields a protocol with communication complexity $O(ps)$ and $2p-1$ rounds. Thus a communication complexity lower bound yields a streaming space lower bound. We use this technique to show that any 1-pass streaming algorithm for USS needs $\Omega(B)$ space.

Lemma 1. *Any deterministic or randomized 1-pass streaming algorithm for USS uses space $\Omega(B)$.*

Proof. We reduce the INDEX problem to USS. The INDEX_n function is defined as follows: Alice has $x \in \{0,1\}^n$ and Bob has an index $k \in [n]$. The goal is to find x_k. Alice can send one message to Bob, after which Bob should announce what he believes is the value of x_k. It is known that the 1-way randomized communication complexity of INDEX_n is $\Theta(n)$ (see [BYJKS02] or [KNR95]).

The reduction from INDEX_n to USS is as follows: Let $B = 2n$. Alice creates a set $S = \{2n - i | x_i = 1\}$. Bob creates the set $T = \{k\}$. Notice that each number in S is at least n. And so any subset of S that has two or more numbers would have a sum strictly greater than $2n$. Hence any subset of $S \cup T$ that has a sum of $2n$ can have at most one number from S. Now it is easy to see that if $x_k = 1$, the subset $\{(2n-k), k\}$ has sum $2n$. And if $x_k = 0$, there is no subset of $S \cup T$ that has sum $2n$. Thus a protocol that correctly decides the USS instance where $B = 2n$, Alice has S and Bob has T with communication cost c also correctly decides $\text{INDEX}_n(x,k)$ with communication cost c.

Assume that there is a space $s(B)$ 1-pass streaming algorithm for USS. Then there is a cost $s(B)$ protocol for USS, and hence by the above reduction, a cost $s(2n)$ protocol for $\text{INDEX}_n(x,k)$. By the lower bound for INDEX, $s(2n) \in \Omega(n)$, and so $s(B) \in \Omega(B)$. □

A generalization of the above proof gives a space lower bound for streaming USS that depends on the number of passes.

Theorem 1. *Any deterministic or randomized p-pass streaming algorithm for USS uses space $\Omega(B/p)$.*

Proof. We give a reduction from $\overline{\text{DISJ}_n}$ to USS. The Disjointness problem DISJ_n is defined as follows: for $x, y, \in \{0,1\}^n$, $\text{DISJ}_n(x,y) = \wedge_{i=1}^n \neg(x_i \wedge y_i)$. (That is, if x and y are characteristic vectors of sets $X, Y \subseteq [n]$, then $\text{DISJ}_n(x,y) = 1$ if and only if $X \cap Y = \emptyset$.) Its complement $\overline{\text{DISJ}_n}$ is the intersection problem. Alice and Bob are given $x \in \{0,1\}^n$ and $y \in \{0,1\}^n$ respectively. The goal is to determine if there exists an $i \in [n]$ such that $x_i = y_1 = 1$. It is known [KS92,Raz92,BYJKS04]

that any randomized protocol for $\overline{\text{DISJ}_n}$, with any number of rounds, must exchange $\Omega(n)$ bits to bound error probability by $1/3$.

The reduction from $\overline{\text{DISJ}}$ to USS is as follows: We set $B = 12n - 1$. Alice constructs the set $S = \{8n - 2i | x_i = 1\}$. Bob constructs the set $T = \{4n + 2i - 1 | y_i = 1\}$. Notice that all numbers in S are greater than $B/2$, and that all numbers in T lie in the interval $(B/3, B/2)$. Further note that each number in S is even and that each number in T is odd. We claim that $\overline{\text{DISJ}_n} = 1$ exactly when $S \cup T$ has a subset adding to B. To see why, first observe that

1. Using numbers only from S cannot give a sum of B since B itself does not appear in S, and the sum of even two numbers from S exceeds B.
2. Using numbers only from T cannot give a sum of B since (1) B does not appear in T; (2) Any two numbers in T add to an even number greater than $B/2$, but B is odd; and (3) adding three or more numbers from T gives a sum greater than B.

Thus we see that if any subset of $S \cup T$ adds to B, then it must contain exactly one number from S and one from T. That is, it must be of the form $\{8n-2i, 4n+2j-1\}$. To add up to $12n - 1$, it must be the case that $i = j$. Hence such a subset exists if and only if there exists an $i \in [n]$ such that $x_i = y_i = 1$.

Now, as in Lemma 1, assume that there is a space $s(B)$ p-pass streaming algorithm for USS. Then there is a cost $(2p - 1)s(B)$ protocol for USS with p rounds, and hence by the above reduction, a cost $(2p - 1)s(12n - 1)$ protocol for $\overline{\text{DISJ}_n}$. By the lower bound for $\overline{\text{DISJ}_n}$, $(2p - 1)s(12n - 1) \in \Omega(n)$, and so $s(B) \in \Omega(B/p)$. □

We now show a space upper bound for USS, for large number of passes.

Theorem 2. *For every $s \leq B$, there is a randomized streaming algorithm for USS that makes s passes over the input, uses space $O(\frac{nB \log^2(nB)}{s})$, and on each input errs with probability at most $1/3$.*

Proof. The idea is to use the algorithm of [Kan10] for just one prime p. We will pick this prime randomly from a large enough range to ensure that the probability of success is high. We first briefly recapitulate Kane's algorithm.

Let $a_1, \ldots, a_n$ be the given set of numbers, and let A be the number of subsets of $\{a_1, \ldots, a_n\}$ that add to B. We want to determine whether $A = 0$. If $A = 0$, then $A = 0 \pmod p$ for all primes p. If $A \neq 0$, then $A \neq 0 \pmod p$ for all primes that do not divide A; the number of primes p such that $A = 0 \pmod p$ is fewer than $\log A \leq \log 2^n = n$.

The algorithm from [Kan10] proceeds as follows: Define

$$C = |B| + \sum_{i=1}^{n} |a_i| + 1.$$

Let $\mathcal{P}$ be the set of the first n primes beyond C. Compute $A \pmod p$ for each $p \in \mathcal{P}$. Clearly, $A = 0 \iff \forall p \in \mathcal{P} : A = 0 \pmod p$. So it suffices to show how to compute $A \pmod p$ for $p \in \mathcal{P}$. To do this, Kane establishes the following key lemma, which we also use.

Lemma 2 (Lemma 1 from [Kan10]). *For any prime $p > C$:*

$$\sum_{x=1}^{p-1} x^{-B} \prod_{i=1}^{n} (1 + x^{a_i}) \equiv -A \pmod{p}$$

This gives a space-efficient way of computing $A \pmod{p}$, for any fixed p: compute the left-hand-side above modulo p by sequentially accumulating the contributions from each $x \in \{1, \ldots, p-1\}$. This yields the logspace algorithm of [Kan10].

However, this approach seems to require multiple passes over the input, since for each $x \in \{1, \ldots, p-1\}$ we need all the input numbers, and furthermore, we need to compute $A \pmod{p}$ for each $p \in \mathcal{P}$.

To handle the second problem, we choose a single prime p *uniformly at random* from the first $3n$ primes beyond C. More precisely, we choose D such that there are at least $3n$ primes between C and D. Let $\mathcal{Q}$ be the set of primes between C and D. Now we pick a prime $p \in \mathcal{Q}$ uniformly at random. If $A = 0$, then $A = 0 \pmod{p}$. If $A > 0$, then $A = 0 \pmod{p}$ for at most n distinct primes p. Hence the probability that our randomly chosen prime p yields $A = 0 \pmod{p}$ is at most $1/3$. Thus it suffices to compute the left-hand-side of the expression in Lemma 2 for a single randomly chosen prime p. We are left with the problem of dealing with sequential accumulation, each x requiring all inputs.

Assume that p has been chosen. Define

$$f(x) = x^{-B} \prod_{i=1}^{n} (1 + x^{a_i}) \pmod{p}$$

$$\sigma(i, j) = \sum_{x=i+1}^{j} f(x) \pmod{p}$$

$$-A \equiv \sigma(0, p-1) \pmod{p}$$

$f(x)$ can be computed in 1 pass using $O(\log p)$ space. Hence $\sigma(i, j)$ can be computed in $j - i$ passes with $O(\log p)$ space. But it can also be computed in 1 pass with $O((j-i)\log p)$ space, by computing $f(x)$ for each $x \in [i+1, j]$ in parallel. In fact, we have a trade-off: for any $1 \leq s \leq j - i$, if s passes are allowed, then $\sigma(i, j)$ can be computed in $\frac{(j-i)}{s} \log p$ space.

We use this trade-off to compute $\sigma(0, p-1)$ in s passes. We first compute $K = \lceil \frac{p-1}{s} \rceil$. We then compute $\sigma(0, K)$ in the first pass, $\sigma(K, 2K)$ and hence $\sigma(0, 2K)$ in the second pass and so on. In s passes, we can obtain $\sigma(0, p-1)$, and we use $O(K \log p)$ space throughout. This works provided $s \leq p - 1$; since $p > B$, it works fo all $s \leq B$.

The kth prime is roughly $k \ln k \in O(k \log k)$. The prime we use, p, is at most as large as the $(C + 3n)$th prime. Since $C \in O(nB)$, we see that $p \in O(nB \log(nB))$. Hence the space used is $O(\frac{nB \log^2(nB)}{s})$. □

3 Approximate USS in Streaming Model

As computing exact USS is provably hard (Theorem 1), the next natural question to ask is: can it be approximated? There is a classical approximation algorithm for the following approximation version of the Subset Sum problem: Given a set of numbers and a target B, let B^* be the largest value *smaller than* B expressible as a sum of a subset of the given numbers. Find a subset with sum in the range $[(1-\epsilon)B^*, B^*]$, for a given ϵ. (Note that B^* itself is not explicitly known.) It is known that this problem has a fully polynomial time approximation scheme (an algorithm with run time polynomial in n, B, $1/\epsilon$); see for instance [CLRS09]. This algorithm is one-pass and works even if the input data is given in binary. However, the space used is $O(n)$ even if the input is given in unary. We wish to approximate USS using a small number of passes on the input and using space polylogarithmic in the length of the input. We consider the following variant: For any ϵ and B and input stream $\tilde{a} = a_1, \ldots, a_n$ where each $a_i \in [B]$, we say that set $S \subseteq [n]$ is an *ϵ-approximator of B in $\tilde{a}$* if $\left(\sum_{i \in S} a_i\right) \in [B(1-\epsilon), B(1+\epsilon)]$. Given $\epsilon, B, \tilde{a}$, we want to decide whether there is an ϵ-approximator of B in $\tilde{a}$. We prove the following theorem:

Theorem 3. *There is a deterministic 1-pass streaming algorithm that on an input stream $\epsilon, B, \tilde{a}$, uses space $O(\frac{\log B}{\epsilon})$ and outputs 1 if and only if there exists an ϵ-approximator for B in the stream $\tilde{a}$.*

Proof. Consider the following algorithm $\mathcal{A}$:

Maintain a set of intervals T.
Initialise: $T \leftarrow \{[B(1-\epsilon), B(1+\epsilon)]\}$.
while End of stream not reached **do**
  $a \leftarrow$ Next number in stream.
  if $\exists$ interval $[\alpha, \beta] \in T$ such that $a \in [\alpha, \beta]$ **then**
    Output YES and halt.
  else
    $T' \leftarrow \{[\alpha, \beta], [\alpha - a, \beta - a] \mid [\alpha, \beta] \in T\}$;
    $T \leftarrow T'$.
    Merge overlapping intervals in T to get a set of pairwise disjoint intervals.
    (If $[a, b], [c, d] \in T$ and $a \leq c \leq b \leq d$, remove $[a, b], [c, d]$ and add $[a, d]$.)
  end if
end while

Before seeing why the algorithm is correct, we first consider the space analysis. Note that at the beginning of each iteration, T has a set of disjoint intervals and each interval has size at least $2B\epsilon$. The space required to store the endpoints of each interval is $O(\log B)$. There can be at most $B/(2B\epsilon)$ disjoint intervals from 1 to B, so at any given time, $|T| \leq \frac{1}{\epsilon}$. Since T' has two intervals for each interval of T, $|T'|$ is also $O(\frac{1}{\epsilon})$. So the space used is $O(\frac{\log B}{\epsilon})$.

We now show that $\mathcal{A}$ is correct; that is, $\mathcal{A}$ outputs YES if and only if there exists a subset of the input numbers that has sum in $[l, r]$. The intuition behind the correctness is the following: We maintain the set of intervals T such that if

any number in the union of the intervals in T is seen as input, then there indeed exists a subset that generates B. This is true in the beginning by the way we initialize T. When a number m is read, a copy of each interval in T is shifted down by m to create a new interval. So if a number in any of these new intervals is seen, then it can be combined with m to give a number in one of the older intervals. (The original intervals are also retained, so we can also not use m in creating a subset.) And this property is maintained by updating T with every number seen. Note that no interval in T gets deleted. Intervals in T only get merged into other intervals to become larger intervals and this does not affect the invariant property.

We now describe the proof more formally: For a set of intervals T, define $R(T) = \{a \mid \exists [\alpha, \beta] \in T : a \in [\alpha, \beta]\}$; $R(T)$ is the union of all the intervals in T. Let $l = B(1-\epsilon)$ and $r = B(1+\epsilon)$. Initially, $R(T) = \{a \mid l \leq a \leq r\}$.

$\Rightarrow$*:* Assume that $\mathcal{A}$ outputs YES. Let T_k denote the collection of intervals after reading k numbers from the stream. $\mathcal{A}$ accepts at a stage k when it reads a number $a_k \in R(T_{k-1})$. We show below, by induction on k, that if $a \in R(T_k)$, then there is a subset of $\{a_1, \ldots, a_k\} \cup \{a\}$ with sum in $[l, r]$. This establishes that the YES answers are correct.

In the beginning, T_0 is initialized to $\{[l, r]\}$. Thus $a \in R(T_0) \Rightarrow a \in [l, r]$.

Now assume that the property holds after reading $k-1$ numbers. That is, if $a \in R(T_{k-1})$, then there is a subset of $\{a_1, \ldots, a_{k-1}\} \cup \{a\}$ with sum in $[l, r]$.

If $a_k \in R(T_{k-1})$, the algorithm terminates here and there is nothing more to prove. Otherwise, $a_k \notin R(T_{k-1})$, and the algorithm goes on to construct T_k. The update sets $R(T_k)$ to contain all of $R(T_{k-1})$ as well as all numbers b such that $a_k + b \in R(T_{k-1})$. Now consider an $a \in R(T_k)$. If it also holds that $a \in R(T_{k-1})$, then we can pretend that a_k was not read at all, and using induction, pull out a subset of $\{a_1, \ldots, a_{k-1}\} \cup \{a\}$ with sum in $[l, r]$. If $a \notin R(T_{k-1})$, then $a_k + a \in R(T_{k-1})$. By induction, we have a subset of $\{a_1, \ldots, a_{k-1}\} \cup \{a_k + a\}$ with sum in $[l, r]$. Hence we have a subset of $\{a_1, \ldots, a_{k-1}, a_k\} \cup \{a\}$ with sum in $[l, r]$, as desired.

$\Leftarrow$*:* Let S, $|S| = k$, be the first subset of numbers in the input stream that has sum in $[l, r]$. That is,

- $S = \{a_{i_1}, a_{i_2}, \ldots, a_{i_k}\}$ for some $i_1 < i_2 < \ldots < i_k$,
- $\sum_{j=1}^{k} a_{i_j} = B - B\lambda$ for some $|\lambda| < |\epsilon|$, and
- there is no such subset in $a_1, \ldots a_{i_k - 1}$.

We will show that $\mathcal{A}$ outputs YES on reading a_{i_k}.

To simplify notation, let s_j denote a_{i_j}.

Observe that if a number a enters $R(T)$ at any stage, then it remains in $R(T)$ until the end of the algorithm. This is because an interval is deleted only when an interval containing it is added.

Now we observe the way T gets updated. After reading s_1, $R(T)$ will contain the intervals $\{[l, r], [l - s_1, r - s_1]\}$. (It may contain more numbers too, but

that is irrelevant.) After reading s_2, $R(T)$ will contain $\{[l, r], [l - s_1, r - s_1], [l - s_2, r - s_2], [l - s_1 - s_2, r - s_1 - s_2]\}$. Proceeding in this way, and using the above observation that $R(T)$ never shrinks, after reading $s_1, s_2, \cdots, s_{k-1}$, $R(T)$ will contain $[l - (s_1 + \cdots + s_{k-1}), r - (s_1 + s_2 + \cdots + s_{k-1})]$. But this interval is the following:

$$\begin{aligned}
&[(B(1-\epsilon) - (s_1 + \cdots + s_{k-1})), (B(1+\epsilon) - (s_1 + \cdots + s_{k-1}))] \\
&= [(B(1-\epsilon) - (B - B\lambda - s_k)), (B(1+\epsilon) - (B - B\lambda - s_k))] \\
&= [(s_k + B\lambda - B\epsilon), (s_k + B\lambda + B\epsilon)] \\
&= [(s_k - B(\epsilon - \lambda)), (s_k + B(\epsilon + \lambda))]
\end{aligned}$$

Since $\epsilon > 0$ and $|\lambda| < |\epsilon|$, $s_k \in [(s_k + B(\lambda - \epsilon)), (s_k + B(\lambda + \epsilon))]$. Hence $\mathcal{A}$ will output YES when s_k is read. □

The following lemma shows that the simple streaming algorithm discussed above is pretty much tight.

Lemma 3. *Let f be any real-valued function. If $f(2x) \log x \in o(x)$, then there is no randomized 1-pass streaming algorithm that ϵ-approximates USS and uses only $O(f(\frac{1}{\epsilon}) \log B)$ space.*

Proof. Assume to the contrary that $\mathcal{A}$ is a randomized 1-pass algorithm that ϵ-approximates USS and uses space $O(f(\frac{1}{\epsilon}) \log B)$. Choose $\epsilon = \frac{1}{2B}$. Now for this value of ϵ, and for every stream $\tilde{a}$, $\mathcal{A}$ will behave like an exact algorithm for USS. The lower bound from Lemma 1 now implies that $f\left(\frac{1}{\epsilon}\right) \log B \in \Omega(B)$, and hence $f(2B) \log B \in \Omega(B)$. But the last relation cannot hold if $f(2x) \log x \in o(x)$. □

4 Multiplicity USS (mUSS) and Monotone Circuits

In this section, we consider the monotone circuit complexity of USS. Without the monotone restrictions, it is known that USS is complete for the circuit class TC^0 ([EJT12]). However, in a very natural sense, Subset Sum is a monotone problem, and so we can consider monotone circuits for it. The encoding of the input becomes crucial for achieving monotonicty. We choose the following encoding:

For each positive integer B, the input consists of the frequency of each number in the stream in unary. An instance $w \in \{0,1\}^{B^2}$ consists of B blocks of B bits each. For each $k \in [B]$, if k occurs in the stream m_k times, then the kth block w_k has exactly m_k 1s; that is, $\sum_{j=1}^{B} w_{kj} = m_k$. Thus the input records the multiplicity of each number in $[B]$ (we assume that no multiplicity exceeds B).

Define the transitive relation $\preceq$: For $u = (u_1, u_2, \ldots, u_B), v = (v_1, v_2, \ldots, v_B)$ with $u_k, v_k \in \{0,1\}^B$, $u \preceq v$ if and only if $\forall k \in [B]$, $\sum_{j=1}^{B} u_{kj} \leq \sum_{j=1}^{B} v_{kj}$.

We define the multiplicity-USS problem, denoted as mUSS, and its approximation variant ϵ-mUSS, as follows.

$$\mathsf{mUSS}(w,B)=1 \iff \exists y=(y_1,y_2,\ldots,y_B):$$
$$y_k \in \{0,1\}^B \quad \forall k \in [B], \quad y \preceq w, \text{ and}$$
$$B = \left(\sum_{k=1}^{B} k \left(\sum_{j=1}^{B} y_{kj}\right)\right)$$
$$\epsilon\text{-}\mathsf{mUSS}(w,B)=1 \iff \exists y=(y_1,y_2,\ldots,y_B):$$
$$y_k \in \{0,1\}^B \quad \forall k \in [B], \quad y \preceq w, \text{ and}$$
$$B(1-\epsilon) \leq \left(\sum_{k=1}^{B} k \left(\sum_{j=1}^{B} y_{kj}\right)\right) \leq B(1+\epsilon)$$

We call such a y a *witness* for (w,B). The vector y represents a subset of the multi-set represented by w such that the elements in y sum to B (or to a number within ϵ of B, respectively).

For example, for $B=4$, the stream 1 3 2 2 1 4 3 can be encoded by any of the following strings (and by many more): 1100 1100 1100 1000, 1010 0101 0011 0010. Some witnesses for this instance are 1100 1000 0000 0000 (use two 1s and a 2), 0100 0000 0001 0000 (use a 1 and a 3), 0000 0000 000 1000 (use the 4).

Fact 4 mUSS *is a monotone function, i.e. for each positive integer B, and for each $u=(u_1,u_2,\ldots,u_B)$, if* $\mathsf{mUSS}(u,B)=1$, *and if $v=(v_1,v_2,\ldots,v_B)$ is obtained from u by changing some* 0*s to* 1*s, then* $\mathsf{mUSS}(v,B)=1$.
*Similarly, for each ϵ and B, ϵ-*mUSS*is a monotone function.*

It has been known for over three decades ([MS80]) that USS is in nondeterministic logspace; hence USS reduces to the problem Reach defined below:

> Given: a layered directed acyclic graph G, two designated nodes s,t
> Output: 1 if there is a path from s to t in G, 0 otherwise.

It is well-known that Reach has monotone circuits of depth $O(\log^2 n)$, where n is the number of vertices in the input instance. (This follows from the construction of [Sav70]. See for example [AB09].) We show that with the encoding described above, exact and approximate versions of mUSS reduce to Reach via monotone projections, and hence have small depth monotone circuits.

Theorem 5. *For every positive integer B,* $\mathsf{mUSS}(\cdot,B)$ *and* ϵ-$\mathsf{mUSS}(\cdot,B)$ *have monotone circuits of depth $O(\log^2 B)$.*

Proof. (Sketch) We prove this by reducing an instance of mUSS into an instance of Reach via a monotone projection.

For every integer B, and given $w \in \{0,1\}^{B^2} = (w_1,w_2,\ldots,w_B)$ we create a graph with B^2+1 layers. The zero-th layer consists of the source vertex and the other B^2 layers have $(B+1)$ vertices each. We further partition B^2 layers into B blocks of B consecutive layers each.

Let $v^i_{j,k}$ denote the i-th vertex in the layer j in the block k. Intuitively, each layer corresponds to a bit position in the input string. We add edges in order to ensure that a vertex $v^i_{k,j}$ is reachable from the source vertex if and only if the stream corresponding to the first $k-1$ blocks of w and j bits from the kth block has a subset that adds to i.
If after reading l bits of w there is a subset that adds to i then this subset continues to exist even after reading more bits. To capture this phenomenon, we add *horizontal edges* from every vertex v in layer l to the copy of v in layer $l+1$. If the bit $w_{kj} = 1$, then using this copy of k, for each existing subset sum s, the subset sum $s + k$ can also be created. To capture this, we include *slanted* edges from each $v^i_{j,k}$ to $v^{i+k}_{j+1,k}$.
Thus, there is a path form the source to vertex $v^i_{B,B}$ exactly when there is a subset that sums to i. By connecting $v^i_{B,B}$ for appropriate i to a new target node t, we reduce mUSS or ϵ-mUSS to Reach. □

5 Circuits with ε-approximators for mUSS as gates

We now examine the power of ϵ-approximators for mUSS when used as a primitive to compute other functions. In [Nis94], Nisan showed that any circuit for DISJ_n using linear threshold gates requires $\Omega(n/\log n)$ gates. We introduce a new kind of gate, an ApproxUSS gate, that we show is at least as powerful as a Threshold or Majority gate, and show that any circuit that uses ApproxUSS gates to compute Disjointness needs size $\Omega(n/\log n)$. However, we do not know whether linear threshold gates can simulate ApproxUSS gates with at most sub-logarithmic blowup in the number of gates or vice versa.

We define approximate USS gates, denoted ApproxUSS, as gates that solve the ϵ-mUSS problem defined in Section 4. An $\text{ApproxUSS}_{\epsilon,B}$ gate takes a bit string x of length B^2 as input, and outputs 1 exactly when $\epsilon\text{-mUSS}(x, B) = 1$.

While it is trivial to see that majority can be computed with a single call to an oracle for mUSS, it is not immediately clear that oracle access to ϵ-mUSS when $\epsilon > 0$ is also sufficient. We show that this is indeed the case, by showing that ApproxUSS gates are at least as powerful as standard threshold gates. Specifically, we show that an ApproxUSS gate can simulate majority with only a polynomial blowup in the number of wires.

Lemma 4. *The MAJ_{2n+1} function can be computed by an $\text{ApproxUSS}_{\varepsilon,B}$ gate with $B = O(n^3)$ and a suitable non-zero value for ε.*

On the other hand, it is not known whether ApproxUSS, for $\epsilon \neq 0$, can be decided with a single oracle call to majority. It is conceivable that demanding a YES answer for a wider range of inputs (the approximation) makes the problem harder. It is therefore interesting to examine the power of circuits using ApproxUSS gates. We follow this thread below.

The communication problem ccApproxUSS corresponding to ApproxUSS can be described as follows. Let $S \subseteq [B^2]$. Both Alice and Bob know S, B and ϵ.

Alice knows the bits x_i for $i \in S$, and Bob knows the remaining bits of x. They must decide whether $\mathsf{ApproxUSS}_{\epsilon,B}(x) = 1$, that is, whether ϵ-$\mathsf{mUSS}(x, B) = 1$.

In Theorem 3 we proved that for every ϵ, there is a one-pass streaming algorithm that ϵ-approximates USS using space $O((\log B)/\epsilon)$. The algorithm works for every possible ordering of the numbers in the input stream. This implies that there is a $O((\log B)/\epsilon)$ bit one-round protocol for $\mathsf{ccApproxUSS}$ for worst case partitioning of the input (for every $S \subseteq [B^2]$). (The string x determines the multi-set of numbers. For any partition of x, Alice and Bob can construct a stream of numbers forming this multi-set, where Alice has the initial part of the stream and Bob has the latter part. Treat the indices in B^2 as pairs k, j where k is the block number and j is the index within the block. Alice includes in her stream a copy of k for each bit $x_{kj} = 1$ in her part. Bob does the same.) Therefore, using an argument similar to that of [Nis94], we can prove the following lemma:

Lemma 5. *Let C be a circuit that computes $DISJ_n$ using s $\mathsf{ApproxUSS}_{\varepsilon,B}$ gates, where $\epsilon \in \Theta(1)$, and the value of B at each $\mathsf{ApproxUSS}_{\varepsilon,B}$ is bounded above by a polynomial in n. Then $s \in \Omega(n/\log n)$.*

Proof. Let C be such a circuit, with s $\mathsf{ApproxUSS}$ gates. Let t denote the maximum value of B in any of the gates. We use C to obtain a protocol for DISJ_n as follows. Alice and Bob evaluate C bottom-up, reaching a gate only after all its children have been evaluated. At each $\mathsf{ApproxUSS}$ gate, we know that $\log B \in O(\log t) \subseteq O(\log n)$. When an $\mathsf{ApproxUSS}$ gate has to be evaluated, an efficient protocol of $O((\log t)/\epsilon)$ bits for $\mathsf{ccApproxUSS}$ is invoked with the appropriate partition of the inputs of the gate. As there are s $\mathsf{ApproxUSS}$ gates, the entire protocol for computing DISJ_n uses $O(((\log t)/\epsilon) \times s)$ bits of communication. However, we know that any protocol for DISJ_n requires $\Omega(n)$ bits of communication ([KS92,Raz92,BYJKS04]). Hence, $s \log t = \Omega(\epsilon n)$. By assumption, $\epsilon \in \Theta(1)$, and $\log t = O(\log n)$. Hence $s = \Omega(n/\log n)$. □

6 Discussion

We now discuss a few aspects of our results and some open questions.

- The upper and lower bounds from Theorems 1 and 2, for s-pass streaming algorithms for USS, do not quite match. There is a gap of $n \log^2(nB)$. Closing this gap will be interesting.
- If the multiplicities of all numbers are restricted to be between $\{0, 1\}$, then the problem does not become easier. In fact, our lower bound proof for USS in Section 2 (Theorem 1) generates such instances.
- We know that USS is in TC^0 [EJT12] and we have proved that mUSS is in monotone NC^2. It is known that there exists a monotone formula of polynomial size which cannot be computed by constant depth polynomial sized monotone threshold circuits [Yao89]. However, the question of whether monotone TC^0 is contained in monotone NC^1 is open (see for instance [Ser04]). Majority is known to be in monotone NC^1 [Val84]. And it is easy to observe that majority reduces to USS. In the light of these results, the question of whether mUSS is contained in monotone NC^1 is interesting.

References

AB09. Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

AMS99. Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

BYJKS02. Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. Information theory methods in communication complexity. In *Proceedings of the 17th Annual IEEE Conference on Computational Complexity*, pages 93–102, 2002.

BYJKS04. Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, and D. Sivakumar. An information statistics approach to data stream and communication complexity. *Journal of Computer and System Sciences*, 68(4):702–732, 2004. (preliminary verion in FOCS 2002).

CLRS09. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

EJT10. Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *FOCS*, pages 143–152, 2010. ECCC TR 62, 2010.

EJT12. Michael Elberfeld, Andreas Jakoby, and Till Tantau. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. In *29th STACS)*, volume 14 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 66–77, 2012. See also ECCC TR 128, 2011.

GJ79. M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

Kan10. Daniel M. Kane. Unary subset-sum is in logspace. *CoRR (arxiv)*, abs/1012.1336, 2010.

KNR95. Ilan Kremer, Noam Nisan, and Dana Ron. On randomized one-round communication complexity. *Computational Complexity*, 8:596–605, 1995.

KS92. Bala Kalyanasundaram and Georg Schnitger. The probabilistic communication complexity of set intersection. *SIAM Journal on Discrete Mathematics*, 5(4):545–557, 1992.

MS80. Burkhard Monien and I.H. Sudborough. The interface between language theory and complexity theory. In R.V. Book, editor, *Formal Language Theory*. Academic Press, 1980.

Nis94. Noam Nisan. The communication complexity of threshold gates. In *In Proceedings of Combinatorics, Paul Erdos is Eighty*, pages 301–315, 1994.

Raz92. Alexander A. Razborov. On the distributional complexity of disjointness. *Theoretical Computer Science*, 106(2):385–390, 1992.

Sav70. W. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177 – 192, 1970.

Ser04. Rocco A. Servedio. Monotone boolean formulas can approximate monotone linear threshold functions. *Discrete Applied Mathematics*, 142(1-3):181–187, 2004.

Val84. L.G Valiant. Short monotone formulae for the majority function. *Journal of Algorithms*, 5(3):363 – 366, 1984.

Yao89. A. C. Yao. Circuits and local computation. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, STOC, pages 186–196, NY, USA, 1989. ACM.