Electronic Colloquium on Computational Complexity, Revision 1 of Report No. 183 (2012)

Streaming bounds from difference ramification

András Z. Salamon

LFCS, School of Informatics, University of Edinburgh
Andras.Salamon@ed.ac.uk
(version of 14 Februrary 2013)

Abstract. In graph streaming a graph with n vertices and m edges is presented as a read-once stream of edges. We obtain an $\Omega(n \log n)$ streaming lower bound on the number of bits of space required to decide graph connectivity. This improves the known bound of $\Omega(n)$ bits, and matches the upper bound of $O(n \log n)$ bits. We go on to develop a method of ramifying inessential differences between streams into significant ones. For graph connectivity, this yields a crisp streaming lower bound (with no undetermined constants) of $n \log n - n(\log \log n + 3/2)$ bits, via lower bounds for the Bell numbers and a pigeonhole argument. We apply difference ramification to min-cut, for a crisp lower bound of $n(n-1)/2$ bits. Difference ramification also shows that streaming n-variable SAT requires at least 2^n bits, compared to the $O(n)$ bits that are sufficient for an unrestricted deterministic Turing machine.

1 Introduction

The streaming model was proposed for problems where the input is presented as a stream of elements, and is too large to fit into random-access memory, but where a smaller data structure may be enough to decide whether the input satisfies some property [8]. In graph streaming, the input is a graph with n vertices and m edges, presented as a read-once stream of edges.

Our contributions Lower-bound arguments in streaming usually rely on reductions from problems for which communication complexity lower bounds are known. Such bounds can only be as powerful as the bounds on the problem being reduced from: the known lower bound of $\Omega(n)$ bits of space for deciding if the input graph is connected is the bound for set disjointness [8,5]. With a reduction from a graph connectivity game, we improve this to $\Omega(n \log n)$ bits.[1]

We then formalise a streaming machine model. Our key contribution is a technique to show space lower bounds by *difference ramification*. We use difference ramification to obtain *crisp* streaming space lower bounds, free of the undetermined constant factors implicit in the order notation of previous bounds. We show that at least $n \log n - n(\log \log n + 3/2)$ bits of space must be used to decide graph connectivity. This nearly matches the upper bound of $n\lceil \log n \rceil + 3\lceil \log n \rceil + \lceil \log \lceil \log n \rceil \rceil + c$ bits for updating a partition of vertices

[1] Logarithms are base 2.

ISSN 1433-8092

representing the connected components, merging two components when an edge arrives connecting them. This simple algorithm is therefore essentially optimal.

Graph algorithms based on computing or approximating eigenvectors often rely on the input graph being connected. For a graph with $2^{33} \approx 8.59$ bn vertices presented as a stream of edges (perhaps representing the social network of people alive during the years 2000–2012), our bound shows that at least 26.95GB of random-access memory is needed to decide whether the graph is connected. On the other hand, this can be done with 35.44GB of random-access memory.

We employ a generalized pigeonhole argument to compare the partitions of streams that relate to states of a data structure with partitions of streams that relate to states of a deterministic streaming machine. If the space restriction is so severe that there are not enough possible states, then there is some state of the streaming machine that can be obtained from two different input streams, while also corresponding to two different configurations of the data structure. We start with these two different input streams. We then ramify the differences between the input streams, as reflected in differences in the associated data structures, into a difference in output state. This is done by constructing a single suffix stream that maintains this difference when processed by the stream machine, until one of the states of the data structure has the desired property, while the other fails to have it. Since the deterministic stream machine was in the same state for both input streams, and then received the same suffix stream, it cannot distinguish these two different inputs. Hence any streaming machine must use at least as much space as avoids this outcome. This establishes the desired space lower bound. This sideways argument, deriving bounds for streaming machines by considering a data structure that is associated with the problem and not any particular streaming machine, is the major novelty of our approach.

Using difference ramification, we derive crisp lower bounds for other decision problems in the streaming model: which of two numbers is larger, whether a directed graph has no sinks, whether a graph has a min-cut of given magnitude, and whether a SAT instance has a solution. Table 1 lists bounds for each problem.

Related work In a circuit model, JáJá showed that two parties must exchange $\Omega(n \log n)$ bits to compute graph connectivity [9, Theorem 3.3]. Hajnal, Maass, and Turán showed an $\Omega(n \log n)$ lower bound on the two-party communication complexity of graph connectivity [6]. This result appears to have been overlooked for streaming lower bounds; we apply the result to show an improved lower bound in Proposition 2 on the streaming complexity of graph connectivity.

Feigenbaum et al. showed that streaming directed graph s-t connectivity requires $\Omega(m)$ bits of space [4, Lemma 4]. For sparse graphs with $m = o(n \log n)$ our bounds are an improvement. For instance, for graphs with maximum degree k (for some fixed k), their method yields at most an $\Omega(kn/2) = \Omega(n)$ lower bound, while we show a crisp lower bound of $n \log n - n(\log \log n + 1 + (\log n)/n)$ bits for the easier problem of deciding connectivity for undirected graphs.

Difference ramification is also related to an argument by Zelke [11, Lemma 12 and Theorem 13]. There, the sequence of vertex degrees is used to show that two different graphs must lead to the same configuration of a machine that

Table 1. Summary of results

lower bound	upper bound ($+c$)	$\Theta(.)$	problem
n	n	n	Example 1
n	$n + \lceil \log n \rceil + \lceil \log \lceil \log n \rceil \rceil$	n	Example 9
$n \log n - n(\log \log n + 3/2)$	$n\lceil \log n \rceil + 3\lceil \log n \rceil + \lceil \log \lceil \log n \rceil \rceil$	$n \log n$	Corollary 14
$n(n-1)/2$	$n(n-1)/2 + 2\lceil \log n \rceil$	n^2	Example 15
2^n	$2^n + n + 2\lceil \log n \rceil$	2^n	Example 16

finds a minimum cut, if the machine is permitted too little space. These two different graphs are then extended to graphs with different min-cut values. (The published version omits this argument in favour of a communication complexity reduction [12].) In Section 3.5 we improve the lower bounds implicit in [11, Theorem 13] using our difference ramification machinery.

2 Preliminaries

Let $[n]$ denote $\{0, 1, \ldots, n-1\}$. Let M be a Turing machine with a set $\mathcal{S}$ of possible inputs. We say that M *decides* $L \subseteq \mathcal{S}$ if M accepts when $x \in L$ and rejects when $x \in \mathcal{S} \setminus L$. For a space-bounded Turing machine M, let $M(x)$ denote the final state of M after processing input x, either accept or reject.

A *streaming machine* is a restricted Turing machine. The input tape contains a stream of items, which are read one at a time by a read-only head that travels from the beginning of the tape to the end. The machine also has access to an unbounded working tape (with a binary alphabet) supplied with a read-write head. This is an abstraction of the streaming model introduced by Henzinger et al. [8]. The streaming machines considered here are deterministic.

Write xy for the stream formed by concatenating streams x and y. Let M be a deterministic space-bounded streaming machine. Denote by $s_M(x)$ the *overall state* of M (including the configuration of all storage used) when given as input a stream xy, just prior to it checking for further input past the x prefix of the stream. (The overall state is determined by x, so y can be arbitrary.) There may be several intermediate states after x has been read but before the suffix y can be read. Increasing the number of states would only serve to make our lower bounds stronger, so we aggregate each such sequence of internal states into a single overall state, ignoring the intermediate states. The *size* $|s_M(x)|$ of overall state $s_M(x)$ is the number of bits of working space used in that overall state. We adopt the convention that a streaming machine writes a binary representation of its state to a special area of the working tape when its state changes. The state of the machine is therefore included in its overall state.

The *space* used by M for input z is $|z|_M = \sup\{|s_M(x)| \mid z = xy\}$, and for the set of inputs $\mathcal{S}$ is $|\mathcal{S}|_M = \sup\{|x|_M \mid x \in \mathcal{S}\}$. Our upper bounds include a constant c that depends on the details of the Turing machine and the number of states in its transition table. For general Turing machines, Cobham called this notion of space *capacity* [2]. In practical terms, the space used during a

computation includes the sizes of registers and internal storage used by the computer, enough bits to represent all internal states of its program, and any random-access memory used.

A *partition* P of a set X is a set of disjoint non-empty subsets (known as *blocks*) of X, such that the union of the blocks in P is the entire set X. A partition $X/\equiv$ is said to be *induced by* an equivalence relation $\equiv$ on set X if it contains the blocks $x_{\equiv}$ for each $x \in X$, where $x_{\equiv} = \{y \in X \mid x \equiv y\}$.

The following warm-up exercise illustrates our core pigeonhole argument, which for this simple problem is similar to the crossing sequences technique for general Turing machine space lower bounds [7].

Example 1 (comparing numbers). Let x and y be streams representing n-bit non-negative integers. For an input stream xy, any streaming machine must use at least n bits of space to decide whether $x < y$, and $n + c$ bits is sufficient.

Proof. Suppose a streaming machine M decides using fewer than n bits of space. Then there are two n-bit numbers, say $0 \leq a < b \leq 2^n - 1$, such that $s_M(a) = s_M(b)$. Now consider the instances bb and ab. For each of these, M is in the same state $s_M(b)$ after the n-bit prefix has been read. The suffix of both input streams is identical. As M is deterministic, the state of M for both of these streams must remain identical, so $M(bb) = M(ab)$. Since $a < b$ and $b \not< b$, the machine M must err when deciding at least one of these instances. Hence at least n bits of space are required.

By simply storing x as it is being read, and then checking each bit against the next bit of y, it is possible to decide the problem using $n+c$ bits of space. □

In Example 1, the trivial algorithm is essentially optimal and the lower and upper bounds differ only by a constant. For problems with streams of structured elements as input, some non-constant overhead may be required to summarise the input in a data structure. Section 3 deals with problems where the input elements are graph edges while also developing a technique for general streams.

3 Streaming bounds, with graph streaming applications

A graph stream consists of elements that represent pairs of vertices of the input graph. The pairs may be directed or undirected, depending on the problem setup. In Section 3.1 we note an improved streaming lower bound for graph connectivity, based on a reduction from communication complexity. We then develop the difference ramification technique for general streaming lower bounds in Section 3.2, based on considering partitions of the set of streams induced by various functions. The key technical tools are two properties that data structures used for deciding a problem should have, together with the Comparison Lemma, which is a convenient way to apply a general pigeonhole principle to partitions. In Section 3.3 we derive and apply explicit bounds on the number of partitions of a set, and use these to obtain lower bounds for graph connectivity in Section 3.4. Finally, in Section 3.5 we discuss an improvement to the streaming lower bounds for deciding min-cut of a graph.

3.1 A streaming bound for graph connectivity via a reduction

We first demonstrate an improved lower bound for graph streaming, via a reduction from communication complexity. This leads to an $\Omega(n \log n)$ lower bound.

Proposition 2 (graph connectivity bound). *A streaming machine that decides whether a graph is connected, when the input graph is presented as a stream of edges with vertices from $[n]$, requires $\Omega(n \log n)$ bits of space.*

Proof. We use a reduction from the communication complexity bound for a graph connectivity game. In this game, the edges of a graph are partitioned evenly between two parties Alice and Bob, who must deterministically decide whether the graph formed by the union of the two subgraphs is connected. Szemerédi's Regularity Lemma then implies that Alice and Bob must exchange at least $\Omega(n \log n)$ bits to decide this problem [6, Theorem 10].

The parties will simulate identical streaming machines M and M' that use at most $b(n)$ bits of space to decide graph connectivity, for any input graph with vertices from $[n]$. The edges of the input graph are partitioned evenly between Alice and Bob. Each party first writes its edges on the input tape of its machine. We show that this leads to a protocol for the graph connectivity game where at most $b(n) + 1$ bits are exchanged; the state of the streaming machine represents a compressed description of the input.

Alice simulates machine M until it has read the contents of its input tape, but before it detects that there is no more input. Alice then sends a description of the internal state of M to Bob, which requires at most $b(n)$ bits. Bob initializes the internal state of machine M' to that received from Alice, and simulates M' (with the remaining edges as input). Bob then reports the output of M' as the result of the computation. This protocol uses at most $b(n) + 1$ bits. Every protocol must exchange $\Omega(n \log n)$ bits, so $b(n) \in \Omega(n \log n)$. □

The reduction in the proof of Proposition 2 seems to have been overlooked. We now discuss a bounding technique that does not rely on communication complexity reductions. This also avoids undetermined multiplicative constants, such as those arising from the use of the Regularity Lemma in the proof above.

3.2 A general lower bound technique

For a function f with domain X, let X/f denote $\{\{y \in X \mid f(x) = f(y)\} \mid x \in X\}$, the partition of X induced by (the equality relation with respect to) f. Let $\mathcal{S}$ be a semigroup with an associative binary operation that maps x and y to xy. We usually let $\mathcal{S}$ be the set of all valid input streams for a class of streaming machines under consideration. The semigroup operation xy then denotes the stream formed by appending stream y (the suffix) to stream x (the prefix).

For the following definitions, let f be a function with domain $\mathcal{S}$. We use $f^{-1}(z)$ to denote the set $\{x \mid f(x) = z\}$, which may be empty.

Definition 3 (fixpoint maintenance). *f maintains fixed points* with respect to the semigroup operation on $\mathcal{S}$ if whenever $f(x) = f(y)$, then $f(xz) = f(yz)$ for any $z \in \mathcal{S}$.

Definition 4 (factor). h is a *factor* of f if there is a map g such that $f = g \circ h$.

Definition 5 (difference ramification). *f supports difference ramification* with respect to $X \subseteq \mathcal{S}$ if whenever $x, y \in X$ and $f(x) \neq f(y)$, then there is some $z \in \mathcal{S}$ such that $xz \in X$ iff $yz \notin X$. (We say that z *certifies a significant difference* between x and y.)

Definition 6 (inessential differences). *f suppresses inessential differences* with respect to $X \subseteq \mathcal{S}$ if whenever $x, y \in \mathcal{S} \setminus X$, then $f(x) = f(y)$.

Note that if M decides L, then s_M maintains fixed points with respect to stream concatenation, and is also a factor of the function M.

Tying together the two requirements to support difference ramification and to suppress inessential differences, we obtain a means for proving crisp streaming lower bounds. A function satisfying both requirements maps streams satisfying some property to a fixed value, but allows differences between the other streams to be ramified. For good bounds, the function should keep the non-satisfying streams apart as far as possible, while preserving some of the structure of the problem. For instance, if the property in question is closed under isomorphism of structures represented by the streams, then the function must also map all streams without the property but with isomorphic structures to the same value.

Difference ramification relies on the following result.

Lemma 7 (Comparison Lemma). *Suppose $\mathcal{S}$ is a semigroup, $h\colon \mathcal{S} \to \{0,1\}$ is a function, $f\colon \mathcal{S} \to X$ is a function that supports difference ramification and suppresses inessential differences with respect to $h^{-1}(0)$, and g is a factor of h that maintains fixed points. Then $|\mathcal{S}/f| \leq |\mathcal{S}/g|$.*

Proof. Let $L = h^{-1}(1)$. Note that then $\mathcal{S} \setminus L = h^{-1}(0)$.

Suppose there exist $x, y \in \mathcal{S}$ with $g(x) = g(y)$ and $f(x) \neq f(y)$. Then $h(x) = h(y)$ as g is a factor of h. As h can take only one of two values, either both $x \in L$ and $y \in L$, or both $x \in \mathcal{S} \setminus L$ and $y \in \mathcal{S} \setminus L$. If the latter, then as f supports difference ramification with respect to $\mathcal{S} \setminus L$, there is $z \in \mathcal{S}$ such that $xz \in L$ iff $yz \in \mathcal{S} \setminus L$. Hence $h(xz) \neq h(yz)$. However, $g(x) = g(y)$ so $g(xz) = g(yz)$, and g is a factor of h, so $h(xz) = h(yz)$. This is a contradiction. Therefore $x, y \in L$. Now f suppresses inessential differences with respect to $\mathcal{S} \setminus L$, so $f(x) = f(y)$. This is again a contradiction.

Hence for all $x, y \in \mathcal{S}$, if $g(x) = g(y)$ then $f(x) = f(y)$. By the Axiom of Choice if $\mathcal{S}/f$ is infinite, or unconditionally if $\mathcal{S}/f$ is finite, there is then an injection from $\mathcal{S}/f$ to $\mathcal{S}/g$. Hence $|\mathcal{S}/f| \leq |\mathcal{S}/g|$. □

For a problem decided by a deterministic streaming machine M, let $g = s_M$, let $h(x) = 1$ if $M(x)$ is an accepting state and let $h(x) = 0$ otherwise. The following form of the Comparison Lemma is then usually more convenient.

Lemma 8 (Streaming Comparison Lemma). *If a function f supports difference ramification and suppresses inessential differences with respect to $\mathcal{S} \setminus L$, then any streaming machine deciding L must use at least $\lceil \log |\mathcal{S}/f| \rceil$ bits of space.*

For a problem decided by a deterministic streaming machine M, there is always a function f satisfying the conditions of Lemma 8, as $f(x) = M(x)$ can be used to satisfy the conditions vacuously. However, collapsing all streams into just two values gives a bound that is not useful. The challenge in applying the Comparison Lemma is in finding a non-trivial function f that keeps apart as many streams as possible, and proving that f supports difference ramification.

In the following example illustrating stream difference ramification with the Comparison Lemma, we use a function which maps directed graph streams to arrays with a fixed number of bits per vertex (in fact, just one bit).

Example 9 (sink-free digraphs). Consider the property "this directed graph has no sinks". Fix some positive integer n. Each stream in $\mathcal{S}$ consists of directed edges of a graph with vertices $[n]$. This property can be decided by keeping an array of n bits, initially all set to 0; whenever a directed edge (u, v) is seen, then the bit corresponding to u is set to 1 indicating that u has at least one successor and is therefore not a sink. After all edges have been read, the array will contain all 1 elements precisely when the graph has no sinks. A streaming machine can decide this property using $n + \lceil \log n \rceil + \lceil \log \lceil \log n \rceil \rceil + c$ bits, for the indicator array, one index variable, and a way to iterate over the bits of the index variable.

Let $L \subseteq \mathcal{S}$ be the set of those streams representing directed graphs with no sinks. Let f map each stream to such an array, indicating which of its vertices is known not to be a sink. With respect to $\mathcal{S} \setminus L$, this function suppresses inessential differences since every stream in L is mapped to the all-1 array.

We claim that f also supports difference ramification with respect to $\mathcal{S} \setminus L$. Suppose $x, y \in \mathcal{S} \setminus L$ such that $f(x) \neq f(y)$. Then (without loss of generality) there is some bit set to 0 in $f(x)$ and to 1 in $f(y)$, corresponding to some vertex u. Now create a stream z of edges containing each edge (v, u) where $v \neq u$. By construction of stream z, the graph $G(xz)$ contains sink u, while $G(yz)$ contains no sinks. Hence $xz \in L$ while $yz \in \mathcal{S} \setminus L$, proving our claim.

Now $|\mathcal{S}/f| \leq 2^n$ as f can take at most 2^n different values, and $|\mathcal{S}/f| \geq 2^n$ as each of the 2^n possible states of the array can be obtained by a stream containing as many edes as the number of 1 bits in the array. By the Comparison Lemma, a streaming machine deciding this problem must use at least n bits of space. □

We now apply the stream difference ramification method to two other problems. For graph connectivity, our next problem, it is crucial to count partitions of a set of distinct objects. We therefore first examine bounds on this quantity.

3.3 Bounds on Bell numbers

The n-th *Bell number* B_n is the number of distinct partitions of $[n]$ (these are sometimes called set partitions). Counting partitions induced by different functions yields the following bounds.

Proposition 10 (Bell number bounds). *For any integer $n \geq 2$,*

$$n \log n - n(\log \log n + 1 + (\log n)/n) < \log B_n < n \log n.$$

For $n \geq 2$, rephrase B_n as $\log c_n = (\log B_n)/n - (\log n - \log\log n)$. Better bounds follow from [1, Theorem 2.1] and [3, Section 6.2].

Corollary 11. *For any integer $n \geq 2$, $-1.5 < \log c_n < 0.1924$. Moreover, for every $\varepsilon > 0$ there is $n_0 = n_0(\varepsilon)$ such that $-0.9139\cdots < \log c_n < -0.9139\cdots + \varepsilon$ for all $n \geq n_0$. (The constant is $-0.9139\cdots = \log\log e - \log e$.)*

Since $\lim_{n\to\infty}(\log n)/n = 0$, the lower bound can be made arbitrarily close to -1 by just using the bounds in Proposition 10 and choosing the threshold for n large enough: for $n \geq 2^{10}$, the bound already exceeds -1.01. No effective values are known for the threshold n_0 for the asymptotic value $-0.9139\ldots$, but the lower bound has the form $n\log n - o(n\log n)$, so it can be expressed simply.

Corollary 12. *For any $\varepsilon > 0$, there is some positive integer $n_0 = n_0(\varepsilon)$ such that for any integer $n \geq n_0$, $(1-\varepsilon)n\log n < \log B_n < n\log n$.*

3.4 Bounds for graph connectivity

We now consider bounds for graph connectivity.

Theorem 13. *For an input stream x of edges of a graph with vertices from $[n]$, a deterministic streaming machine that correctly decides whether $G(x)$ is connected must use at least $\lceil \log B_n \rceil$ bits of space. A deterministic streaming machine can decide this problem using $n\lceil\log n\rceil + 3\lceil\log n\rceil + \lceil\log\lceil\log n\rceil\rceil + c$ bits.*

Proof. Let $\mathcal{S}$ be the set of all streams of edges with vertices from $[n]$. Denote by $G(x)$ the graph described by the stream of edges $x \in \mathcal{S}$. Let $L \subseteq \mathcal{S}$ consist of those streams x such that $G(x)$ is connected. Let $f(x)$ denote the partition of the vertices of $G(x)$ that represents the connected components of $G(x)$.

If $G(x)$ is connected, then $f(x)$ contains just one block, so f suppresses inessential differences with respect to $\mathcal{S} \setminus L$. We claim that f also supports difference ramification with respect to $\mathcal{S} \setminus L$.

Suppose $x, y \in \mathcal{S} \setminus L$ with $f(x) \neq f(y)$. Then there are two vertices u and v in the same block W of (say) $f(x)$ but in different blocks $U \ni u$ and $V \ni v$ of $f(y)$. We construct the suffix stream z explicitly as a concatenation $z = z_0 z' z''$ of three streams that all start empty. First colour vertices in U with ultramarine, vertices in V with vermilion. The idea is now to add edges while keeping the ultramarine and vermilion blocks apart. For each vertex w in $[n] \setminus (U \cup V)$, add an edge $\{v, w\}$ to z_0, and colour w vermilion. If $f(xz_0)$ has only one block, then we are done. Otherwise it has at least two blocks. For each vermilion vertex w in the blocks of $f(xz_0)$ not containing u and v, add an edge $\{v, w\}$ to z'. Note that both endpoints in such edges are vermilion, so $f(yz_0) = f(yz_0z')$. If $f(xz_0z')$ has only one block, we are done, so suppose it has at least two blocks. By the construction of z', all its blocks not containing u, v contain only ultramarine vertices. Now add edges between these blocks, forming a stream z'', collapsing them into a single block. Again, these new edges do not affect the blocks of $f(yz_0) = f(yz_0z') = f(yz_0z'z'')$. Now add an edge to z'' connecting u and one of

these ultramarine vertices. Then $f(xz_0z'z'')$ has a single block, while $f(yz_0z'z'')$ still has two. It follows that f supports difference ramification.

Every partition of $[n]$ can be obtained from a stream of edges, so $|\mathcal{S}/f| = B_n$. The Comparison Lemma then implies that $\lceil \log B_n \rceil$ bits of space are necessary for any deterministic streaming machine that decides L.

For the upper bound, the following straightforward algorithm decides connectivity. For any prefix x of the input stream of edges, maintain $f(x)$, and update it as new edges arrive. When there are no more edges to read, return YES if every vertex is in the same block, or NO otherwise. We now sketch how to implement this algorithm on a streaming machine. A simple upper bound on the amount of space it uses will then serve as our bound.

The partition can be represented using n slots each of $\lceil \log n \rceil$ bits, or $n\lceil \log n \rceil$ bits total. Each slot u indicates which block $p(u)$ vertex u belongs to, and the slots are initialized so that $p(u) = u$ for each u. For each new edge $\{u, v\}$ in the input stream, if u and v currently belong to the same block then nothing is done, otherwise merge the blocks $p(u)$ and $p(v)$. Specifically, when merging take the larger of the partition number $p(u)$ and $p(v)$ (for argument's sake, say $p(v)$) and for every element w with $p(w) = p(v)$, set $p(w)$ to $p(u)$. This requires at most $3\lceil \log n \rceil + \lceil \log \lceil \log n \rceil \rceil$ bits in addition to the partition, to keep an index variable which is used to iterate over the vertices, two registers for $p(v)$ and $p(u)$, and a way of keeping track of the bits when comparing two values with $\lceil \log n \rceil$ bits. Hence the space usage is at most $n\lceil \log n \rceil + 3\lceil \log n \rceil + \lceil \log \lceil \log n \rceil \rceil + c$ bits. □

Theorem 13 and Proposition 10 together imply the following result.

Corollary 14 (streaming connectivity bounds). *For integer $n \geq 2$, a streaming machine deciding if an n-vertex graph is connected requires at least $n \log n - n(\log \log n + 1 + (\log n)/n)$ bits of space. A streaming machine exists that decides this problem using at most $n\lceil \log n \rceil + 3\lceil \log n \rceil + \lceil \log \lceil \log n \rceil \rceil + c$ bits of space.*

Corollary 12 and Theorem 13 together imply that $(1 - \varepsilon)n \log n$ bits are necessary, with ε arbitrarily close to 0 for large enough n.

3.5 Bounds for graph min-cut

Given a graph G, a *cut* is a partition $\{V_1, V_2\}$ of the vertices $V(G)$ with two blocks. The value of the cut $\{V_1, V_2\}$ is the total number of edges of $E(G)$ with one endpoint in V_1 and the other in V_2. The min-cut problem requires finding a cut with minimum value. We work with the decision version, where a threshold value is given as the first part of the input, and it must be determined whether the min-cut is at least as small as the threshold.

Zelke uses the $2^{(n/8)(n/8-1)/2}$ graphs on n vertices to argue for an $\Omega(n^2)$ lower bound (see [11, Lemma 12 and Theorem 13]). If less than $(n/8)(n/8-1)/2$ bits are used by the streaming machine, then two distinct graphs from this set result in the same state, and these are obtained from two different streams x and y. Zelke then constructs a stream of edges z to ramify the difference between x and

y. From the proof a crisp lower bound of $n^2/512 - n/16$ bits therefore follows. The construction can be improved by a factor of roughly 256 to $n(n-1)/2$, using difference ramification by a function that distinguishes between all graphs with min-cut at most the threshold value.

Example 15 (Min-cut). Min-cut requires at least $n(n-1)/2$ bits to decide on a streaming machine with its input a stream of edges of a graph with vertices from $[n]$. Min-cut can be decided with $n(n-1)/2 + 2\lceil \log n \rceil + c$ bits.

Note that these bounds correspond to the threshold $n-2$; smaller thresholds lead to smaller lower bounds and may also lead to smaller upper bounds.

4 Discussion and open questions

The Comparison Lemma yields lower bounds on the space required to decide a problem. This relies on a function f that both supports difference ramification and suppresses inessential differences. A lower bound should be available for the cardinality of the set of values taken by f, and for the largest possible bounds, f should collapse as few elements as possible. The main challenge lies in ensuring that f supports difference ramification. For two arbitrary streams x and y that do not have the desired property, this requires constructing an appropriate suffix stream z that ramifies a difference in the data structures $f(x)$ and $f(y)$ associated with the two streams into an essential difference between the two streams xz and yz. The construction of the suffix stream does not rely on the generic machine M, but only on the data structures associated with the problem. If the data structure has some redundancy, then there may be multiple distinct configurations that do not significantly differ for ramification. Difference ramification can therefore be regarded as a method that unifies proving lower bounds with finding appropriate data structures.

We have considered a selection of graph streaming problems chosen to illustrate the power of stream difference ramification. An argument analogous to Example 1 shows that deciding set disjointness on a streaming machine requires at least (a crisp) n bits of space, compared with the previous $\Omega(n)$ bound[8]. We now outline a lower bound for streaming SAT. This shows that, as expected, deterministic Turing machines are strictly more powerful than deterministic one-pass streaming machines. However, being able to access the input in an unrestricted manner yields at least an *exponential* benefit in terms of space usage.

Example 16 (streaming SAT). The streaming Boolean satisfiability problem requires deciding if a Boolean formula in conjunctive normal form with n variables is satisfiable, when it is presented as a stream of clauses. This requires at least 2^n bits of space to decide, and can be decided with $2^n + n + 2\lceil \log n \rceil + c$ bits of space. (In contrast, it can be decided by a deterministic Turing machine with $n + 3\lceil \log n \rceil + c$ bits of space.) In outline, the lower bound follows because given two input instances with different sets of solutions, a common suffix stream can be constructed from clauses that each disallow precisely one solution, removing every solution of one input instance while keeping the other satisfiable. □

Some questions remain open; a selection follows.

Undirected s-t connectivity Directed s-t connectivity requires $\Omega(m)$ bits to decide with a one-pass streaming machine [5]. For dense graphs with $m = \Omega(n^2)$ edges this is $\Omega(n^2)$ bits. In contrast, undirected s-t connectivity can be decided using $O(n \log n)$ bits with a one-pass streaming machine, regardless of whether the graphs are dense or sparse, by distinguishing partitions formed by connected components. In a model without restrictions on how the input may be read, logarithmic space is sufficient to decide undirected s-t connectivity [10]. This prompts us to ask: how many passes are needed for a logspace-bounded streaming machine to decide undirected s-t connectivity?

Optimal upper bounds To reduce the gap between the upper and lower bounds for graph connectivity, the data structure could use $\lceil \log B_n \rceil$ bits to represent a canonical name for each $P(x)$. The difficulty is how to create such a naming scheme. Whenever a new edge arrives, then it must be used to derive the new partition name without using a significant amount of additional space.

For the min-cut example an adjacency matrix achieves the bound for threshold $n - 2$. For smaller values of the threshold, it may be more efficient to assign a canonical name to each graph with min-cut at most the threshold; again the question is how to efficiently update this data structure as new edges arrive.

Knowing the number of vertices We assumed here, as is common in the streaming literature, that n is known a priori. This is a benign assumption when seeking lower bounds, since with less information the lower bounds may only become larger. However, not knowing the range of values also may require larger upper bounds. It would be interesting to study the case when the set of vertices appearing in the stream of edges is from the set $[n]$ for some unknown n. For instance, an algorithm may have to rearrange data structures if they have been built with the assumption of a particular n, which then turns out to have been too small. Allowing for such rearrangement may require unavoidable overhead.

Certificates In prior work on graph streaming, certificates are subgraphs that can be used to quotient the set of streams [5]. We have extended this notion via the map f, allowing more general data structures as certificates. For the connectivity lower bound, partitions of the vertices serve as certificates, and for the sinkless digraph and number comparison lower bounds an n-bit string suffices. What other kinds of certificates are generally useful for (graph) streaming?

Multiple passes When multiple passes over the input are allowed, a lower bound of $b(n)$ bits becomes at least $b(n)/k$ bits with k passes, but it is not clear whether this is actually achievable. Is it possible to solve 2-pass streaming graph connectivity with $n \log n - n(\log \log n + 0.9139\ldots) - 1$ bits of space? It is not clear how to ramify, even if two streams lead to the same state.

Dividing by the number of passes may also not yield the best possible multi-pass lower bound. As an example, for the problem in Example 1 three $\lceil \log k \rceil$

bit counters can track the number of passes and store every k-th bit of the first number using $\lceil n/k \rceil$ bits; these are then compared to the corresponding bits of the second number. With k passes, $\lceil n/k \rceil + 3\lceil \log k \rceil + c$ bits suffice, which is $1 + 3\lceil \log n \rceil + c$ bits for $n = k$. It does not seem likely that an algorithm exists that can decide this problem with n passes and a constant amount of space.

More general models Difference ramification splits the input into a fixed prefix, and a suffix that an adversary can manipulate to ramify differences. This is not possible if the streaming machine is nondeterministic, or if the input may be read multiple times. (An unrestricted Turing machine can simply be modified to scan the entire input tape before it begins, foiling such an adversary.) Can a form of difference ramification be applied to more general kinds of computation?

Acknowledgements Kousha Etessami suggested counting partitions by using half the elements as names. Juan Reutter made useful suggestions on presentation. I thank Ewan Klein and Stratis Viglas for discussions that inspired this work, and I am grateful to Peter Buneman and Rahul Santhanam as my academic hosts.

References

1. D. Berend and T. Tassa. Improved bounds on Bell numbers and on moments of sums of random variables. *Probability and Mathematical Statistics*, **30**(Fasc. 2), 185–205, 2010. Electronic version available from http://www.math.uni.wroc.pl/~pms/files/30.2/Article/30.2.1.pdf.
2. A. Cobham. The recognition problem for the set of perfect squares. *SWAT*, 78–87. IEEE Computer Society, 1966. doi:10.1109/SWAT.1966.30.
3. N. G. de Bruijn. *Asymptotic methods in analysis*. Dover, 1981. ISBN 0486642216.
4. J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *TCS*, **348**(2–3), 207–216, 2005. doi:10.1016/j.tcs.2005.09.013.
5. J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the data-stream model. *SIAM J. Comput.*, **38**(5), 1709–1727, 2009. doi:10.1137/070683155.
6. A. Hajnal, W. Maass, and G. Turán. On the communication complexity of graph properties. *STOC*, 186–191. ACM, 1988. doi:10.1145/62212.62228.
7. F. Hennie. One-tape, off-line Turing machine computations. *Information and Control*, **8**(6), 553–578, 1965. doi:10.1016/S0019-9958(65)90399-2.
8. M. R. Henzinger, P. Raghavan, and S. Rajagopalan. Computing on data streams. In J. M. Abello and J. S. Vitter, editors, *External memory algorithms*, 107–118. American Mathematical Society, 1999. ISBN 0821811843.
9. J. JáJá. The VLSI complexity of selected graph problems. *JACM*, **31**(2), 377–391, Mar. 1984. doi:10.1145/62.70.
10. O. Reingold. Undirected connectivity in log-space. *JACM*, **55**(4), 17:1–17:24, 2008. doi:10.1145/1391289.1391291.
11. M. Zelke. *Algorithms for Streaming Graphs*. PhD thesis, Humboldt-Universität zu Berlin, Feb. 2009.
12. M. Zelke. Intractability of min- and max-cut in streaming graphs. *IPL*, **111**(3), 145–150, 2011. doi:10.1016/j.ipl.2010.10.017.

Appendix

We have elided some proofs in the text. Where these proofs are not completely straightforward, details are provided here.

Proof (Lemma 8). Suppose f supports difference ramification and suppresses inessential differences with respect to $\mathcal{S} \setminus L$.

Let M be a deterministic streaming machine that decides L, let $g = s_M$, let $h(x) = 1$ if $M(x)$ is an accepting state and let $h(x) = 0$ otherwise. Then s_M is a factor of M that maintains fixed points, and $\mathcal{S} \setminus L = h^{-1}(0)$.

By Lemma 7 we have $|\mathcal{S}/f| \leq |\mathcal{S}/s_M|$, and hence $\lceil \log |\mathcal{S}/f| \rceil \leq \lceil \log |\mathcal{S}/s_M| \rceil$. Further, $\lceil \log |\mathcal{S}/s_M| \rceil \leq |\mathcal{S}|_M$. Since M was arbitrary, the bound $\lceil \log |\mathcal{S}/f| \rceil$ holds for any streaming machine deciding L. □

Proof (Proposition 10). For the first inequality, fix some subset $S \subseteq [n]$ containing $k = \lceil n/\log n \rceil$ elements. Consider a partition of S containing k singleton blocks. Each of the remaining $n - k$ elements of $[n]$ can then be added to any of these blocks. Each such assignment of the remaining elements leads to a distinct partition of $[n]$, and there are k^{n-k} such assignments; however, not all partitions of $[n]$ can be obtained in this way. Hence $B_n > k^{n-k}$ and therefore $\log B_n > (n-k)\log k$. Note that $n/\log n \leq k < n/\log n + 1$, so $-k > -n/\log n - 1$. Since $n \geq 2$, we also have that $\log k \geq \log n - \log\log n > 0$, and $\log\log n \geq 0$. Hence $-k(\log n - \log\log n) > -n - \log n$, so

$$\log B_n > (n-k)(\log n - \log\log n) > n\log n - n\log\log n - n - \log n.$$

For the upper bound, consider a partition of $S = [2n] \setminus [n]$ consisting of n singletons. Each of the elements of $[n]$ can then be added to any of the blocks of the partition in one of n^n ways. Each such assignment forms a partition of $[2n]$, and by now removing the elements of S from the blocks, then removing any empty blocks, what remains is a partition of $[n]$. Some partitions of $[n]$ can be formed in different ways, but each partition can be obtained in this way. Hence $B_n < n^n$ and the required upper bound follows. □

Proof (Corollary 11). For the lower bound, $\log B_n > n\log n - n(\log\log n + 1 + (\log n)/n)$ by Proposition 10. Note then that $(\log n)/n$ decreases for $n \geq 2$, with its maximum at $n = 2$. For the upper bound, rewrite [1, Theorem 2.1] as $\log B_n < n(\log n + 0.1924 - \log\log(n+1))$ and simplify.

The asymptotic result is a restatement of an expression discussed by de Bruijn [3, Section 6.2]. The asymptotic expression for B_n can be written as $\log c_n - 0.9139\ldots + o(c_n)$ (see also [1, (2.4)]), with the constant $-0.9139\ldots$ being $\log\log e - \log e$ and the lower order terms positive. Hence for large enough n it is always possible to bound the lower order terms in the interval $(0, \varepsilon)$ for any desired $\varepsilon > 0$. □

Proof (Example 15). For convenience consider the complement of the problem, which requires deciding whether the min-cut of the input exceeds the threshold.

By inverting the output of the streaming machine, we obtain a machine for deciding min-cut that uses the same amount of space. (In fact, observe that the class of languages accepted by a streaming machine using b bits of space is closed under complementation.)

Let $\mathcal{S}$ be the set of streams of edges with vertices from $[n]$, and let L_k be the set of streams x such that $G(x)$ does not have a min-cut with value at most as large as the threshold value k. Let f map a stream of edges x to graph $G(x)$ if $G(x)$ has a min-cut with value at most k, and to the complete graph on n vertices otherwise. Map f suppresses inessential differences for min-cut with respect to $\mathcal{S} \setminus L_k$ by definition. We now show that f supports difference ramification with respect to $\mathcal{S} \setminus L_k$.

Suppose $x, y \in \mathcal{S} \setminus L_k$ with $f(x) \neq f(y)$. Then there is some edge $\{u, v\}$ which exists in one of these two graphs (say $f(x)$) but not the other. We have to construct a stream of edges z that increases the min-cut of the graph containing edge $\{u, v\}$ to $k+1$, while maintaining the min-cut of the graph not containing $\{u, v\}$ at k or less. First, create a stream z_0 containing all edges in y that are not in x. This maintains the min-cut of $G(yz_0) = G(y)$; if the min-cut of $G(xz_0)$ exceeds k then we are done by setting $z = z_0$, so assume not. Any further edges we consider do not occur in either graph. Note that $G(yz)$ is a subgraph of $G(xz)$ for $z = z_0$, and that adding any further edges to z maintains this relationship. Since the min-cut of $G(xz_0)$ does not exceed k, the degree d of u in $G(xz_0)$ is at most k. If $d < k-1$ then choose $k-1-d$ non-neighbours of u in $G(yz_0)$, other than v, and for each such vertex w, add an edge between u and w to a new stream z'. Since $k \leq n-1$, this ensures that the degree of u is precisely $k-1$ in $G(yz_0z')$, and at least k in $G(xz_0z')$. Now create a stream z'' containing all missing edges in $G(xz_0z')$, except those that involve u, and let $z = z_0z'z''$.

At this point, $G(yz)$ has a cut $\{\{u\}, [n] \setminus \{u\}\}$ with value $k-1$, and may have a min-cut that is even smaller, while this cut in $G(xz)$ has a value at least k and every other cut has value at least $n-1 \geq k$. Hence z certifies a significant difference between x and y. The Comparison Lemma then yields a crisp lower bound of $\lceil \log(2^{n(n-1)/2} - 1 + 1) \rceil = n(n-1)/2$ bits for $k = n-2$.

For an upper bound for threshold $n-2$, since there is only one graph (up to isomorphism) that has a min-cut of $n-1$, it is enough to recognize whether the input contains all possible edges of a complete graph. This can be done by keeping an adjacency matrix of the graph, using $n(n-1)/2$ bits of space, and using two index variables using $2\lceil \log n \rceil$ bits to access the relevant bit of the adjacency matrix when an edge is read. When the end of the edge stream has been reached, the machine simply checks whether any of the edges is missing. This is a total of $n(n-1)/2 + 2\lceil \log n \rceil + c$ bits.

Smaller thresholds only require keeping track of the number of distinct graphs with min-cut at most k, so the lower bound is smaller for $k < n-2$. We leave open the question of how to implement an efficient data structure that only distinguishes the graphs with min-cut at most k when $k < n-2$; the adjacency matrix representation will suffice for any threshold. □

Proof (Example 16). For streaming SAT, let $\mathcal{S}$ be the set of all streams of clauses featuring variables from $[n]$. Note that the stream will in general contain a number of clauses that is exponential in n.

Let $L \subseteq \mathcal{S}$ be the set of streams representing instances that are satisfiable, and let f be the map that maps a stream x of clauses to the array of 2^n bits representing all possible assignments of the n variables that satisfy the formula represented by x. The entry with index i in this array ($i = 0, 1, \ldots, n-1$) has value 1 when the n-bit binary representation of i, considered as a list of assignments to the n variables, constitutes a solution to x. Every unsatisfiable formula is mapped to the all-0 array, so f suppresses inessential differences with respect to L. We also claim it supports difference ramification with respect to L.

Suppose $x, y \in L$ such that $f(x) \neq f(y)$. Without loss of generality, suppose that $f(x)$ (when considered as a set of assignments) is a not a subset of $f(y)$. Stream z contains clauses (each containing up to n literals) that forbid each of the satisfying assignments in $f(y)$ but no others. Since $f(x)$ is not a subset of $f(y)$, there is some assignment in $f(xz)$ that is satisfying, so $xz \in L$, while $f(yz)$ has no satisfying assignments, so $yz \in \mathcal{S} \setminus L$. Stream z therefore certifies a significant difference between x and y.

Now each possible value of f can be obtained by some stream of clauses, for instance with each clause containing n literals and forbidding precisely one assignment. Hence $\log |\mathcal{S}/f| = 2^n$.

For the upper bound, it is enough to implement the array described by f, using an n-bit index into the array, and a way to process each literal in the current clause. The general deterministic Turing machine upper bound follows from trying each assignment in turn. □

ECCC ISSN 1433-8092

http://eccc.hpi-web.de