Streaming Algorithms for Independent Sets in Sparse Hypergraphs[†]

Bjarni V. Halldórsson[‡] Magnús M. Halldórsson[§] Elena Losievskaja[§¶]
Mario Szegedy[‖]

April 26, 2015

Abstract

We give the first treatment of the classic independent set problem in graphs and hypergraphs in the streaming setting. The objective is to find space-efficient algorithms that output independent sets that are "combinatorially optimal", that is, with size guarantee in terms of the degree sequence alone. Our main result is a randomized algorithm that achieves this using space in bits that is linear in the number of vertices.

We use this to examine assumptions about the streaming model, and advocate the study of *output-efficient* algorithms that measure space usage relative to the size of the output solution. In that sense, our main algorithm uses space linear in the output size. We also examine algorithms that use little or no space in addition to the bits storing the output.

Our algorithms fall also into an *online streaming* model, where output-changes can go only in one direction. In particular a feasible solution must be maintained at all times, and items that are removed from the solution can never reenter. We obtain tight bounds on deterministic algorithms for independent sets in graphs in that model.

1 Introduction

In this paper we consider streaming algorithms for the classic independent set problem on graphs and hypergraphs. As input, we are presented with a sequence of (hyper)edges, and we have to output a large set of vertices that contains no (full) edges.

For graphs, a theorem of Paul Turán guarantees an independent set of size $n/(\overline{d}+1)$ [18], where $\overline{d}$ is the average degree of the graph. If the entire degree-sequence, $d_1, \ldots, d_n$, of the graph is available, then $\sum_{i=1}^{n} \frac{1}{d_i+1}$ is a stronger lower bound for the maximum independent set size (in this paper n will always denote the number of nodes of the input graph) [4, 19]. This formula has a generalization for hypergraphs as well (see Section 1.2). We seek algorithms that output solutions of this "combinatorially optimal" size.

[†]A preliminary version of this paper appeared in the Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP), Bordeaux, France, July 2010.

[‡]ICE-TCS, School of Science and Engineering, Reykjavik University, and deCODE genetics, 101 Reykjavik, Iceland, email: `bjarnivh@ru.is`

[§]ICE-TCS, School of Computer Science, Reykjavik University, 101 Reykjavik, Iceland, email: `mmh@ru.is`. Research supported by grants 7000921,90032021 and 12003211 of the Icelandic Research Fund

[¶]Current address: Icelandic Heart Association, Holtasmari 1, 201 Kopavogur, Iceland. email: `ellossie@gmail.com`

[‖]Dept. of Computer Science, Rutgers University, 110 Frelinghuysen Road, Piscataway, New Jersey, USA, email: `szegedy@cs.rutgers.edu`. Supported by NSF grant EMT-0523866.

The motivation for the streaming model [16] comes from practical applications of managing massive data sets such as, e.g., real-time network traffic, on-line auctions, and telephone call records. These data sets are huge and arrive at very high rate, making it impossible to store the entire input and forcing quick decisions on each data item.

However, most graph problems are impossible to solve within the polylogarithmic space bound that the traditional streaming model offers. For instance, testing basic properties like connectivity or bipartiteness requires $\Omega(n)$ space [12].

This observation led to the introduction of the *semi-streaming* model [16], where space for n-vertex graphs is restricted to $n \log^{O(1)} n$. The algorithms then have enough space for some information for each vertex, but not enough to store the whole graph. A number of graph problems have been studied in the semi-streaming model, including bipartite matching (weighted and unweighted cases) [11, 10], diameter and shortest paths [11, 12], min-cut [1], and graph spanners [12]. The independent set (IS) problem, to our best knowledge, has not been studied in the model before.

We propose to study algorithms that are *output efficient*, by measuring space usage relative to the size of the required output. In that sense, our focus is on models that are more restrictive than the semi-streaming, either aiming for space linear in the solution size or minimal space in addition to the solution bits.

All the algorithms we consider have the common feature that they maintain a feasible solution at all times. Also, decisions made on edges are irreversible: once a node is ejected from the solution, it never enters it again. This defines a new *online streaming* model. It is closely related to *preemptive online* algorithms, considered recently by Epstein et al. [10] in a streaming context for weighted matching. The difference is that in our problem, we can view the whole vertex set as belonging to the initial solution, thus the solution of any algorithm is *monotonously non-increasing* with time.

To contrast, in the classical *online* version of the IS problem [14, 2], vertices arrive one by one, along with all incident edges to previous vertices. The online algorithm must then determine once and for all whether the node is to be included in the constructed feasible independent set solution. This problem is very hard to solve online [14, 2]; e.g., a competitive factor of $n-1$ is best possible for deterministic algorithms, even when restricted to trees. However, bounded-degree graphs are comparatively easy, since a factor of Δ is trivial for a deterministic greedy algorithm, where Δ denotes the maximum degree.

By focusing on the space requirements and how the working space can be used, we seek to gain a deeper understanding of the ways that independent sets can be computed.

There have been few works on online graph problems that allow for one-way changes along the lines of the online streaming model (where items can be removed from a solution at any time but not reintroduced). A rare example is a recent (and concurrent) work of [7] that can be viewed as the maintenance of a *strong* independent set (a set of vertices that intersects each edge in at most one node) of a hypergraph when edges arrive in a stream; the focus in that work is however on the competitive ratio, not the space usage. Following the initial publication of our work [13], two papers have explored further streaming algorithms that use space proportional to the output size: interval selection (or independent set in interval graphs) [8], and semi-matchings [9].

Finally, the independent set problem in general graphs was studied in the streaming setting [15], with strong space lower bounds.

1.1 Our Contribution

We present in Sec. 2 a priority-based schema of streaming algorithms for IS in sparse graphs and hypergraphs. It is parameterized by a priority function that rules which vertices from each edge will survive and which are removed from the solution. We instantiate the schema with three priority functions. Our main result is showing that there is an $O(n)$-space algorithm that achieves a combinatorially optimal bound (denoted $\Omega(i(H))$ and defined shortly).

We use this opportunity to examine the appropriate models for streaming algorithms in this context. Our algorithms have the property of being *online* in the sense that a feasible solution must be maintained at all times and a vertex can never reenter the solution once it has been ejected. We study in Section 3 the power of deterministic algorithms in such an online streaming model, which are much weaker than randomized algorithms.

We also examine the tradeoffs between space and solution quality. One conclusion is that a desirable goal for streaming algorithms should be to use space only linear in the size of the *output*, which for IS is n bits. We consider in Section 4 the case when only minimal extra space is available, in addition to the bits needed to store the solution, and show in particular that with no additional space the performance guarantees are exponentially worse.

1.2 Definitions

Given a hypergraph $H = (V, E)$, let n and m be the number of vertices and edges in H, respectively. We assume that H is a *simple* hypergraph, i.e. no edge is a proper subset of another edge. An *independent set* I in H is a vertex subset that contains no edge of H. If I is independent, then $V \setminus I$ is said to be a *hitting set*.

A hypergraph is r-*uniform* if all edges have the same cardinality r. Graphs are exactly the 2-uniform hypergraphs. For graphs and hypergraphs the *degree* $d(v)$ of a vertex v is the number of edges incident on v. We denote by Δ and $\overline{d}$ the maximum and the average degree, respectively. For us, a better measure for the degree of the vertex v of H is the *efficient degree*, $d^*(v)$, which is the real-valued solution to $\sum_{e:v\in e} \frac{1}{(d^*(v))^{|e|-1}} = 1$. For intuition, if all edges incident on vertex v are of size r, then $d^*(v) = \sqrt[r-1]{d(v)}$.

Let $\alpha(H)$ denote the size of a maximum independent set in H. Also, define $i(H) = \sum_v \frac{1}{d^*(v)}$, and note that for a d-regular r-uniform hypergraph H, $i(H) = n/\sqrt[r-1]{d}$. What makes $i(H)$ interesting for us is that $\alpha(H) = \Omega(i(H))$. In fact, as Caro and Tuza have shown that $\Omega(i(H))$ is the strongest lower bound for $\alpha(H)$ that we can obtain from the degree-sequence alone for a large class of hyper-graphs [5], and we are not aware of any class for which an asymptotically better bound could be given. Let IS denote the problem of finding in hypergraphs an independent set of size $\Omega(i(H))$.

We can generalize all of the definitions for weighted (hyper)graphs. A vertex-weighted hypergraph has a non-negative weight function on its vertices. The notions of average degree etc. carry over to the weighted case in a natural way, for instance $i(H)$ becomes $\sum_v \frac{w(v)}{d^*(v)}$, where w is the weight function. Most of our results will carry over to the weighted case with obvious modifications.

Let $[n]$ denote $\{1, 2, \ldots, n\}$.

Model We assume that the algorithm knows in advance n, the number of vertices, but has no other information about the input. The edges arrive in a stream, in an adversarial order. We assume that the vertices are labelled $0, \ldots, n-1$; this assumption can be avoided by simply maintaining a lookup table, but the storage requirements for such lookup are beyond the scope of our considerations here.

For this article, the most precious resource is space, and we model and regard memory as a linear array of bits with direct access (as in standard C programming). When using big-oh notation, we mean constants independent of the edge sizes and degree parameters.

We consider online streaming algorithms that maintain at all times a single valid solution, in the form of a single bit for each vertex. The solution is monotonously non-increasing with time: initially, it contains all vertices, but once a vertex is removed from the solution it never reappears. Note that vertices of degree 0 will automatically be included in any solution output.

2 Priority-Based Streaming Algorithms

We consider the following algorithm schema that operates on a stream of (hyper)edges. It is parameterized by a priority function f mapping vertices to natural numbers, with higher values representing lower priority, meaning less chances of being part of the solution output.

```
$S \leftarrow \emptyset$
For each edge $e$ in the edge stream do
    Let $v$ be the vertex in $e$ with the largest value $f(v)$, resolving conflicts arbitrarily
    $S \leftarrow S \cup \{v\}$
Output $I = V \setminus S$
```

Figure 1: Algorithm schema Priority, parameterized by a priority function $f : V \to \mathbb{N}$

Priority places a lowest-priority vertex of each edge into a set S, that then forms a hitting set (vertex cover). Thus, the set I output is an independent set. The quality of the solution depends on the priority function f. The space requirements depend also only on the compressibility of the function f (besides $O(\log n)$ bits for local variables), and note that this working space is only written to before the stream is processed. Finally, the time complexity is the aggregate complexity of looking up the f-values of the vertices in each edge, in addition to whatever precomputation is needed for constructing f.

We shall present three different instantiations of Priority with different functions f.

2.1 RandomPermute

In the first instantiation of Priority, we choose f to be a random permutation, i.e., drawn uniformly at random from bijective functions $f : V \to [n]$. The resulting algorithm, RandomPermute, assigns the last (according to the random permutation) vertex of each edge to a vertex cover S. As featured in the book of Alon and Spencer [3], it achieves the Turán bound $\sum_{i=1}^{n} \frac{1}{d_i+1}$ on graphs; on hypergraphs, it was shown by Shachnai and Srinivasan [17] to achieve the bound $\Omega(i(H))$. As presented, the algorithm is clearly a streaming algorithm with space complexity $O(n \log n)$, which becomes our benchmark.

Theorem 2.1 ([3, 17]) *There is a randomized streaming algorithm* RandomPartialPermute *that given an edge stream, finds an independent set of expected size at least* $\sum_{i=1}^{n} \frac{1}{d_i+1}$ *on graphs and* $\Omega(i(H))$ *on hypergraphs, using expected* $O(n \log n)$ *bits of space.*

2.2 Random subset algorithm

Our key algorithm, and our second example, is based on the well-known observation that random sets of the right size in a sparse graph are nearly independent. It uses the complemented characteristic function of a random subset X of pn vertices, for a given p; namely, $f_p(v)$ is 0 if $v \in X$ and 1 otherwise. Let us refer to this algorithm — Priority run with f_p, parameterized by p — as RandomSubset(p).

Remark. By a random set of size pn we shall mean either of two things: 1. We select a subset of $V = [n]$ of size pn uniformly and randomly. 2. We create a random subset X of V by the procedure that selects each node in V randomly and independently with probability p. While we prove Lemma 2.3 below only for case 2, our applications will use the lemma for case 1, where it also holds.

We need the following special case of the FKG-inequality, as used in [17].

Theorem 2.2 (FKG) *Given a vector $\vec{Y} = (Y_1, Y_2, \ldots, Y_\ell)$ of independent Bernoulli random variables and an event F that is completely determined by the Y_i's, call F* increasing *if and only if the following holds: for any $\vec{a}$ such that F holds when $\vec{Y} = \vec{a}$, F also holds when $\vec{Y} = \vec{b}$ that coordinate-wise dominates $\vec{a}$ (i.e., $a_i \le b_i$, for all i). Then, for any collection of* increasing *events $F_1, F_2, \ldots, F_t$, it holds that*

$$\Pr\left[\bigwedge_{i=1}^{t} F_i\right] \ge \prod_{i=1}^{t} \Pr[F_i] \ .$$

We now analyze RandomSubset(p).

Lemma 2.3 *Let H be a hypergraph, p be a number and A be the set of vertices of H with efficient degree $\le \frac{1}{2p}$. Then, running* RandomSubset(p) *on H yields an independent set I such that $\mathbb{E}[|A \cap I|] \ge p|A|/2$. In fact, each node in A is included in I with probability at least $p/2$.*

Proof: For a node v in A, let χ_v be the indicator random variable of the event that node v is selected into I. Then $|A \cap I| = \sum_{v \in A} \chi_v$. We aim to show that $\mathbb{E}[\chi_v] \ge p/2$, from which the lemma follows.

For $v \in I$ to hold, it suffices that two things occur: v is chosen into X, and for each edge e incident on v it holds that $e \not\subseteq X$. Let us define events that capture this: Y_v is the event that $v \in X$ and F_e^v is the event $e \not\subseteq X | v \in X$, i.e., conditioned on v being in X, some element in e is outside X. Observe that the events Y_v and F_e^v are independent, for any vertex v and edge e containing v. The probability of Y_v is p. Thus,

$$\Pr[\chi_v = 1] = \Pr[Y_v] \cdot \Pr\left[\bigwedge_{e \ni v} F_e^v\right] = p \cdot \Pr\left[\bigwedge_{e \ni v} F_e^v\right] \ .$$

The probability of the conditional event F_e^v is

$$\Pr[F_e^v] = 1 - \Pr[e \subseteq X | v \in X] = 1 - p^{|e|-1} \ ,$$

If the edges incident on v have only the vertex v in common and are otherwise disjoint, then this gives us that

$$P[\chi_v = 1] = p \prod_{e \ni v} \Pr[F_e^v] = p \prod_{e:\, v \in e} \left(1 - p^{|e|-1}\right) \ge p \left(1 - \sum_{e:\, v \in e} p^{|e|-1}\right) \ . \tag{1}$$

Now, note that since the efficient degree of any node v in A is at most $1/(2p)$, by our assumption, it holds that

$$\sum_{e:\, v \in e} (2p)^{|e|-1} \le \sum_{e:\, v \in e} (1/d^*(v))^{|e|-1} \le 1 \ .$$

Thus, since there are no singleton edges, it holds that $2 \le 2^{|e|-1}$ and therefore that $\sum_{e:\, v\in e} p^{|e|-1} \le 1/2$. Applying this bound in (1), yields $\mathbb{E}[\chi_v] = P[\chi_v] \ge p/2$, as desired. When the (hyper)edges incident on v intersect, the FKG-inequality (Thm. 2.2) implies the same lower bound. The lemma now follows from the additivity of expectation. ■

Clearly, RandomSubset is very space efficient, as it uses only one bit per vertex. Intuitively, it works well for the "right" value of p.

RandomSubset in parallel Interestingly, we can run algorithm RandomSubset "in parallel" for many different p's at the same time in the same *sequential* algorithm. This happens implicitly in the case of algorithm RandomPermute. For a random permutation π, define

$$X_k = \{\text{first } k \text{ elements of } \pi\} \ .$$

Now X_k is a random set of size pn, where $p = k/n$. Notice that upon running RandomPermute we also run RandomSubset indirectly for $p = i/n$, $i = 1, 2, \ldots, n$ in parallel, giving X_k for $k = 1, \ldots, n$. Indeed, it holds that when a hyperedge e is processed, we add the last vertex, v, of e to S. Thus, unless $e \subseteq X_k$, vertex v is not in X_k.

Using this property, we see that for each vertex v, the expectation of χ_v in Lemma 2.3 is at least $\frac{1}{4d^*(v)}$, just by picking X_k with $k = n/d^*(v)$, and applying the same argument to bound $\mathbb{E}[\chi_v]$ from below.

Using the linearity of expectation, we therefore obtain an alternative proof of the result of Shachnai and Srinivasan [17](aside from a constant factor). The space needed is only the $O(n \log n)$ bits needed to store the random permutation π.

Theorem 2.4 *Parallel executions of* RandomSubset *yield a randomized streaming algorithm that, given a hypergraph H, finds an independent set of expected size $\Omega(i(H))$ using $O(n \log n)$ space.*

2.3 Linear Space Algorithm

We now create a more efficient algorithm, RandomMap, based on the observation that the above argument goes through even when we only use the properties of X_k for $k = 1, 2, 4, 8, \ldots, n$. Indeed, if v has efficient degree d^* then choose k such that $\frac{n}{2d^*} \le k \le \frac{n}{d^*}$. Then, we get that the expectation χ_v of Lemma 2.3 is at least $\frac{1}{8d^*(v)}$. Assume in the following, for simplicity of exposition, that n is a power of 2.

To provide the X_k, for all powers of two, we use a map $\rho : V \to \{1, 2, \ldots, \log n\}$, where the probability of $\rho(v) = i$ is $1/2^i$ (to make the total probability equal to one, we set the probability of $\rho(v) = \log n$ to be $2/n$ instead of $1/n$). This can be obtained by counting unbiased coin flips until flipping a head. Then, $X_{2^j} = \{v : \rho(v) \ge \log n - j\}$, for $j = 0, 1, \ldots, \log n$.

We now describe how to store and access ρ using small space. Since the domain of ρ has size $\log n$, there is a straightforward random access implementation that uses only deterministic space $n \log\log n$ by storing $\rho(i)$ in the memory segment $[(i-1)\lceil \log\log n \rceil + 1, i\lceil \log\log n \rceil]$. We can use the space even more efficiently, however. Note that $\mathbb{E}[\rho(v)] = O(1)$, using that $\sum_{i\geq 1} i/2^i = 2$, and thus by the linearity of expectation, the expected bit-length of the sequence

$$\rho(1)\ \rho(2)\ \ldots \rho(n)$$

is linear in n. We write the ρ-values in a prefix-free code, which at most triples the length. To facilitate binary search, we pack these items into odd-numbered words of size $\log n$, and store in the even-numbered words the index of the first item packed in the next succeeding word. This again at most doubles the space requirements. We can now implement a binary search to find a given $\rho(i)$, until we reach the right word, in which we need only examine $\log n$ bits. Hence, the running time is $O(\log n)$. Since the total bit-length of any sequence of $\Theta(\log n)$, adjacent ρ-values will be highly concentrated around its mean, so interpolation search could also be applied to reduce the complexity further still.

Theorem 2.5 *There is a randomized streaming algorithm that finds in a given hypergraph an independent set of expected size $\Omega(i(H))$, using expected $O(n)$ bits of space and deterministic $O(r \log n)$ time per edge of size r.*

3 Deterministic Online Streaming Algorithms

All the algorithms considered in this paper have the following two properties: (a) a feasible solution I is maintained at all times, and (b) rejections (i.e., the removal of a node from I, or alternatively addition to S) are irrevocable. We refer to such algorithms as *online streaming* algorithms.

In this section, we study the power of this model in the deterministic case. We restrict our attention here to the case of graphs.

Deterministic algorithms: The next result shows that no deterministic algorithm can attain a performance ratio in terms of the average degree $\overline{d}$ alone, nor a ratio of $o(2^{\Delta})$.

Theorem 3.1 *The performance ratio of any deterministic algorithm in the online streaming model is $\Omega(n)$. This holds even for trees of maximum degree $\log n$, giving also a lower bound of $\Omega(2^{\Delta})$. It also holds even if the algorithm is allowed to use arbitrary extra space.*

Proof: Assume that $n = 2^k$, for an integer k. Let A be any deterministic algorithm. We construct a tree on n vertices; thus at any stage, the graph induced by the edges seen so far induces a forest.

We maintain at any stage the invariant that the independent set selected by A contains at most one node from each tree in the forest. The construction proceeds in k rounds. In round i, for $i = 1, 2, \ldots, k$, $n/2^i$ edges are presented. Each edge connects two components; in either component, we choose as endpoint the node that is currently in A's solution, if there is one, and otherwise use any node in the component. This ensures that the algorithm cannot keep both vertices in its solution, maintaining the invariant.

In the end, the resulting graph is a tree of maximum degree at most k, and A's solution contains at most one node. ■

When allowing additional space, we can match the previous lower bound in terms of Δ.

Theorem 3.2 *There is a deterministic algorithm in the online streaming model with a performance ratio $O(2^{\Delta})$.*

Proof: We consider an algorithm that maintains additional information in the form of a counter c_v for each node v, initialized as zero.

When an edge arrives between two nodes in the current solution I, we compare the counters of the nodes. The node whose counter is smaller, breaking symmetry arbitrarily, is then removed from the current solution I. The counter of the other node, e.g. u, is incremented. We then say that u *eliminated* v. We say that a node u is *responsible* for a vertex x if u eliminated x, or, inductively, if u eliminated a node that was responsible for x.

Let $r(v)$ denote the number of nodes for which node v is responsible, and let $R(k)$ denote the maximum $r(v)$ over nodes v with $c_v = k$. We claim that $R(k) \leq 2^k - 1$. It then follows that the size of I is at least $n/2^{\Delta}$, since c_v is at most the degree of v. For the base case $R(0) = 0$, since the node never eliminated another vertex. Assume now that $R(t) \leq 2^t - 1$, for all $t < k$. Consider a node v with $c_v = k$, and let $u_1, u_2, \ldots, u_k$ denote the vertices eliminated by v in order. Before the i-th elimination, the value of c_v was $i-1$, hence the value of c_{u_i} was at most $i-1$. Once eliminated, the counter c_{u_i} for node u_i stays unchanged. Hence, by the inductive hypothesis, $r(u_i) \leq R(i-1) \leq 2^i - 1$. We then have that v was responsible for at most

$$r(v) \leq \sum_{t=1}^{k}(r(v_t)+1) \leq \sum_{t=1}^{k}(R(t-1)+1) = \sum_{t=0}^{k-1} 2^t = 2^k - 1 \ .$$

Since this holds for any give v with $c_v = k$, we have established that $R(k) \leq 2^k - 1$. ■

4 Minimal Space Algorithms

In the most restricted case, we have no extra space available. We can refer to such algorithm as *memoryless*, since they cannot store anything about previous events. Can we still obtain reasonable approximations to IS?

We show that there exists a function g allowing us to find an $n/g(d)$-independent set in this model, but that g must now be exponential.

A simple modification to Priority, which we shall call RandomDelete, is to always select the covering vertex v at random. This algorithm is clearly memoryless, as it even does not check whether it has already added some other vertex from the edge into the cover.

Intuitively, a memoryless algorithm would seem to be unable to do significantly better than randomly selecting the vertex to be eliminated. We restrict our attention in this section to the case of graphs.

Theorem 4.1 RandomDelete *finds an independent set of expected size $n/2^{\overline{d}}$. This is tight for this algorithm in that even if the algorithm avoids eliminating vertices unnecessarily: there is a graph instance in which it only finds an independent set of expected size at most $n/2^{\Theta(d)}$.*

Proof: *Positive result.* Each vertex v belongs to the final solution $V \setminus S$ with probability $2^{-d(v)}$. Therefore, the expected size of $V \setminus S$ is $\sum_{v \in V} 2^{-d(v)} \geq n/2^{\overline{d}}$, using the linearity of expectation and Jensen's inequality.

Limitation result. Consider the graph stream S_n with vertex set $V = \{v_1, v_2, \cdots, v_n\}$ and edges $\{v_i, v_j\}$ for any $|i - j| \leq k$, where the edges arrive in the stream in lexicographic order: (v_1, v_2), (v_1, v_3), $\ldots$, (v_1, v_k), $(v_2, v_3), \ldots, (v_2, v_{k+1})$, (v_3, v_4), etc. Note, that all but the first and the last k vertices have degree $2k$. Thus, the average degree $\overline{d} \leq \Delta = 2k$.

Suppose first that the algorithm avoids eliminating vertices unnecessarily, i.e., if it ignores an edge is one endpoint has already been selected into a cover. We claim that the expected size of the independent set I found by the algorithm on S_n is at most $1 + n/2^k$. We prove the claim by induction on n. For $n \leq k$, the claim holds since graph is a clique.

For the inductive step, consider the first vertex v_1 of the stream. There are two cases, depending on whether v_1 ends up in I, the independent found by the algorithm.

Case 1: $v_1 \in I$. It means that during the processing of the edges (v_1, v_j), for $j = 2, 3, \ldots, k$, all the neighbors of v_1, namely $v_2, v_3, \ldots, v_k$, are added to the cover S. The probability of this event occurring is $P[v_1 \in I] = 2^{-k}$. All the edges incident on $v_2, v_3, \ldots, v_k$ are then ignored. The remaining stream is identical to the graph stream S_{n-k}. By the induction hypothesis, the expected size of the solution found on S_{n-k} is at most $1 + (n-k)/2^k \leq 1 + (n-1)/2^k$.

Case 2: $v_1 \notin I$. Suppose v_1 was selected to cover the t-th edge incident on v_1 for some $t \in [1, k]$. Then, the first $t-1$ neighbors of v_1 were selected to cover the first $t-1$ edges incident on v_1. All later edges incident on $v_1, v_2, \ldots, v_t$ are then ignored by the algorithm. The remaining stream is then identical to the graph stream S_{n-t}. By the induction hypothesis, the expected size of the solution found is at most $1 + (n-t)/2^k \leq 1 + (n-1)/2^k$.

Thus, a vertex $v_i \in V$ is inserted in I only in Case 1 and the probability of this event is 2^{-k}, for any $i \in [1, n-k]$. Combining the two cases, we obtain that the expected size of the solution found is

$$2^{-k}(1 + 1 + \frac{n-k}{2^k}) + (1 - 2^{-k})(1 + \frac{n-1}{2^k}) \leq 2^{-k} + 1 + \frac{n-1}{2^k} = 1 + n/2^k \ ,$$

as desired.

Now, when the algorithm is memoryless, the probability that a given vertex v remains in the independent set solution is $2^{-d(v)} \leq 2^{-k}$, since the minimum degree of the graph is k. By the linearity of expectation, the expected size of the solution is at most $n/2^k$. We conclude by observing that the average degree of the graph is $2k(1 + o(1))$. ■

Remark. The algorithm has the special property of being *oblivious* in that the solution maintained, or any other part of memory, is never consulted in the operation of the algorithm until it is output.

4.1 The utility of advice

When the average efficient degree $d^*(H)$ is known in advance, we can obtain from the RandomSubset schema an algorithm that requires only logarithmic space in addition to the solution bits. Initially, the solution bit for vertex v is set if v is in X; as we proceed, the bit records whether a v is contained in the current independent set solution, i.e. in $\overline{S} \cap X$. When $p = 1/d^*(H)$, the reciprocal of the efficient degree, Lemma 2.3 yields the following bound that is slightly weaker than $i(H)$.

Theorem 4.2 *When $d^*(H)$ (or its approximation) is known in advance, there is an online streaming algorithm that finds an independent set of expected size $\Omega(n/d^*(H))$ in $O(\log n)$ extra space using $O(r)$ time to process each edge.*

In comparison with the earlier zero-space algorithms, this suggests that knowledge of the input parameters is highly useful for IS. This relates to the recent *annotation model* of [6], although the assumption there is that the advice is dispensed only after the stream is given.

5 Open questions

What is the right computational model for graph problems in the streaming context? All of our algorithms for IS use: A read-once-only input tape + A tape storing precomputation + A tape for the output stream with very limited access (in some case write-only) + Poly-logarithmic work space. Is (poly-logarithmic work space + restricted storage types) the right way to capture a range of graph problems that do not fit conveniently into existing streaming models? If other graph problems can be also captured by this model, this could grow into a new brand of research.

Acknowledgments

We thank Páll Melsted for helpful discussions.

References

[1] K. J. Ahn and S. Guha. Graph sparsification in the semi-streaming model. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings*, pages 328–338, 2009.

[2] N. Alon, U. Arad, and Y. Azar. Independent sets in hypergraphs with applications to routing via fixed paths. In *Proc. of Third International Workshop on Randomization and Approximation Techniques in Computer Science, and Second International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, RANDOM-APPROX'99*, pages 16–27, 1999.

[3] N. Alon and J. Spencer. *The Probabilistic Method.* John Wiley, 1992.

[4] Y. Caro. New results on the independence number. Technical report, Tel-Aviv University, 1979.

[5] Y. Caro and Z. Tuza. Improved lower bounds on k-independence. *Journal of Graph Theory*, 15(1):99–107, 1991.

[6] G. Cormode, M. Mitzenmacher, and J. Thaler. Streaming graph computations with a helpful advisor. *Algorithmica*, 65(2):409–442, 2013.

[7] Y. Emek, M. M. Halldórsson, Y. Mansour, B. Patt-Shamir, J. Radhakrishnan, and D. Rawitz. Online set packing. *SIAM J. Comput.*, 41(4):728–746, 2012.

[8] Y. Emek, M. M. Halldórsson, and A. Rosén. Space-constrained interval selection. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, pages 302–313, 2012.

[9] Y. Emek and A. Rosén. Semi-streaming set cover - (extended abstract). In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 453–464, 2014.

[10] L. Epstein, A. Levin, J. Mestre, and D. Segev. Improved approximation guarantees for weighted matching in the semi-streaming model. *SIAM Journal on Discrete Mathematics*, 25(3):1251–1265, 2011.

[11] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. *Theoretical Computer Science*, 348(2):207–216, 2005.

[12] 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, 2008.

[13] B. V. Halldórsson, M. M. Halldórsson, E. Losievskaja, and M. Szegedy. Streaming algorithms for independent sets. In *Automata, Languages and Programming, - 37th International Colloquium, ICALP, Bordeaux, France, Proceedings*, pages 641–652. Springer, 2010.

[14] M. M. Halldórsson, K. Iwama, S. Miyazaki, and S. Taketomi. Online independent sets. *Theoretical Computer Science*, 289(2):953 – 962, 2002.

[15] M. M. Halldórsson, X. Sun, M. Szegedy, and C. Wang. Streaming and communication complexity of clique approximation. In *Automata, Languages, and Programming - 39th International Colloquium, ICALP 2012, Warwick, UK, July 9-13, 2012, Proceedings, Part I*, pages 449–460, 2012.

[16] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[17] H. Shachnai and A. Srinivasan. Finding large independent sets in graphs and hypergraphs. *SIAM J. Discrete Math.*, 18(3):488–500, 2004.

[18] P. Turán. On an extremal problem in graph theory (in Hungarian). *Mat. Fiz. Lapok*, 48:436–452, 1941.

[19] V. K. Wei. A lower bound on the stability number of a simple graph. Technical Memorandum No. 81-11217-9, Bell Laboratories, 1981.