Brief Announcement: New Streaming Algorithms for Parameterized Maximal Matching & Beyond *

Rajesh Chitnis
Dept. of Computer Science & Applied Mathematics
Weizmann Institute of Science, Israel
rajesh.chitnis@weizmann.ac.il

Graham Cormode
Dept. of Computer Science.
University of Warwick, UK
g.cormode@warwick.ac.uk

Hossein Esfandiari
Dept. of Computer Science
University of Maryland,
College Park, USA
hossein@cs.umd.edu

MohammadTaghi Hajiaghayi
Dept. of Computer Science
University of Maryland,
College Park, USA
hajiagha@cs.umd.edu

Morteza Monemizadeh
Dept. of Computer Science.
Charles University, Prague,
Frankfurt, Czech Republic
monemi@iuuk.mff.cuni.cz

ABSTRACT

Very recently at SODA'15 [2], we studied maximal matching via the framework of *parameterized streaming*, where we sought solutions under the promise that no maximal matching exceeds k in size. In this paper, we revisit this problem and provide a much simpler algorithm for this problem. We are also able to apply the same technique to the *Point Line Cover* problem [3].

1. INTRODUCTION

The streaming model for processing graphs is an attractive one: we keep a compact data structure that summarizes all data seen to date, and incrementally update it as new edges are observed. However, for many problems it is known that any such data structure must be large, in the worst case proportional to the total size of the graph. An ongoing line of research asks what can be computed using resources much less than simply storing the data in full. In our recent work [2], we studied graph streaming problems in the parameterized setting: we seek solutions provided that the size of the solution is bounded by a parameter k. Of particular interest is finding a matching in a graph, a fundamental question in graph algorithms. Under the promise that no maximal matching exceeds k in size, we gave an intricate algorithm to process a dynamic stream (one with both edge insertions and deletions) that required $O(k^2 \operatorname{polylog} n)$ space, and showed that any algorithm for this problem needs $\Omega(k^2)$ space.

In this paper, we revisit this problem and provide a much simpler algorithm for this problem, which meets the same bounds. We further apply the same technique to the Point Line Cover problem [3] where we are given n points on the plane and a parameter k, and the goal is to see if we can cover all points using k lines. Our belief is that the insights gained from this algorithm could apply to other problems in this area, and that the reduced complexity of the description also makes the algorithm easier to understand and apply.

2. MAXIMAL MATCHING

Definition 1. (k-sparse recovery algorithm) *A k-sparse recovery algorithm is a data structure which accepts insertions and deletions of elements so that, if the current number of elements stored in it is at most k, then these can be recovered in full.*

Such algorithms have been designed to operate deterministically, and require $O(k \operatorname{polylog} n)$ space [1].

Definition 2. (k-sample algorithm) *A k-sample algorithm is a data structure which accepts insertions and deletions of elements so that at any moment it can provide a sample of size k from the elements stored in it, provided there are at least k such elements.*

Randomized constructions of k-sample algorithms are known (which use k-sparse recovery algorithms within them), and require $O(k \operatorname{polylog} n)$ space [1].

Assumption 3. (promise) *We assume that, at every point during the execution of the algorithm, the maximal matching of the graph has size at most k.*

*R.C. was supported by a postdoctoral fellowship from I-CORE ALGO. G.C. was supported by the Yahoo Faculty Research and Engagement Program and a Royal Society Wolfson Research Merit Award. H.E. and M.H. were supported in part by NSF CAREER award 1053605, NSF Grant CCF-1161626, ONR YIP award N000141110662, DARPA/AFOSR grant FA9550-12-1-0423, and a Google Faculty Research award. M.M. was supported by the project 14-10003S of GA ČR. Part of this work was done when M.M>. was at Goethe-Universität Frankfurt, Germany and supported in part by MO 2200/1-1

SPAA'15, June 13–15, 2015, Portland, OR, USA.
ACM 978-1-4503-3588-1/15/06.
http://dx.doi.org/10.1145/2755573.2755618.

Our algorithm for finding a maximal matching works as follows. It keeps information about edges in two forms. A data structure L keeps a set of edges exactly. We also maintain up to $2k+1$ data structures S_i $(1 \leq i \leq 2k+1)$ which are capable of sampling up to $2k+1$ edges from a set (i.e. instances of $(2k+1)$-sample recovery algorithms). Each S_i is associated with a timestamp t_i (index of an update in the stream), which represents when the structure was created.

We partition the nodes into two classes: low-degree, meaning that they have at most $2k$ incident edges, and high-degree, meaning that they have more than $2k$ incident edges. We maintain a (sub)set H of high degree nodes, corresponding to the set of nodes which are represented with the sketches S_i. Initially, L and H are empty, the S_i are empty, and all nodes are considered low-degree.

Edge Insertion: For each edge that is inserted, we ensure that it is reflected in one of the data structures. If the edge is incident on two nodes in H, then we store the edge in the sketch S_i associated with the oldest summary, i.e. the one with the oldest timestamp. If the edge is incident on one node $v \notin H$, and another $w \in H$, then we store it in the sketch associated with w. If the edge is incident on two nodes neither of which is in H, then we store the edge explicitly in L. If this insertion causes one of the low degree nodes v to have more than $2k$ edges in L incident on it, then we create a sketch for v, and assign it the current timestamp. We extract all edges in L that are incident on v, and insert them into the sketch of v. We update H to include v.

Edge Deletion: For each edge that is deleted, we identify where it is represented in the data structures, and remove it. If the edge is present in L, then we remove it. Else, at least one of its endpoints must have a sketch. If only one has a sketch, then we remove the edge from this sketch. Else, we remove the edge from the sketch with the oldest timestamp. If removing an edge from a sketch causes the node, v to become low degree, then we interrogate the sketch to find all the edges incident on v. For each extracted edge, if it is also incident on a high degree node w, then it is placed in the sketch for w, else it is placed in L. The sketch can then be removed, and further v is removed from H.

Invariants: The update procedures above ensure that a number of invariants hold over the course of the stream.

1. (Unique representation) Each edge is represented in exactly one place within the data structures L, S_i.
2. (Old sketch) If an edge is incident on two sketches, then it is always represented in the older of these two.

These invariants can be checked easily. The property of unique representation follows from the description of the insertion and deletion procedures: when a new edge is inserted, it is stored in only one data structure; when an edge is deleted, it is removed. The operations which move edges from L into sketches, from one sketch into another sketch, or from a sketch into L also preserve this property.

The property of an edge being associated with the older of its two sketched nodes similarly follows by consideration of the update operations. The only cases which require some care are when an edge is currently associated with two sketched nodes, and the older of these sketches is removed. The deletion process makes it clear that this edge is then moved to the sketch of the other node.

Note that the algorithm does not explicitly maintain the degree of nodes that are not high degree, only those in H. It is consequently possible that a node may reach a degree of greater than k and remain outside of H (for example, if several of its edges are incident on nodes that are in H). That is, every node in H is a high degree node, but not every high degree node necessarily has a sketch and is in H. This does not affect the correctness of the algorithm.

We now make some observations about the algorithm.

LEMMA 4. *Under Assumption 3, we have*

- *The number of high degree nodes is at most* $2k+1$
- *The number of edges in* L *is bounded by* $4k^2$.

PROOF. We prove the two statements in the lemma as follows:

- If there are more than $2k+1$ high degree nodes, then there must be a matching of size more than k. Such a matching can be found by picking a high degree node and an arbitrary neighbor. This still leaves at least $2k-1$ nodes whose degree is at least $2k-1$, and so the procedure can be iterated. This will result in a matching of size $k+1$.
- This follows similarly, since we could otherwise greedily find a large matching within L. Each matched edge removes at most $4k$ other edges from L that are incident on the matched nodes, since we ensure that all nodes in L have at most $2k$ edges from L incident on them from. If we match $2k$ nodes, then we remove at most $4k^2$ edges; hence if there are more than $4k^2$ edges in L, then these can participate in the matching, violating the promise.

□

From Lemma 4 we can conclude the following theorem

THEOREM 5. *The space cost of this algorithm is* $\tilde{O}(k^2)$.

PROOF. From Assumption 3, we have that $|L|$ is $O(k^2)$, and the number of sketches stored is always $O(k)$, the size of which is bounded by $O(k \operatorname{polylog} n)$ (see Definitions 1 and 2). Hence, the total space cost is $O(k^2 \operatorname{polylog} n)$. □

Algorithm to find a matching: Given these data structures, the algorithm to find a matching is straightforward: it recovers a graph G' which is a subgraph of the graph G described by the stream. We then find a maximal matching on G', and argue that this is also maximal for G.

Graph G' is formed by extracting as many edges from the data structures as possible. That is, for each sketch S_i we extract up to $(2k+1)$ edges incident on the corresponding node, and set G' to be the union of all these edges plus those stored in L. We now argue that this information is sufficient to find a maximal matching for G.

THEOREM 6. *The above procedure indeed finds a maximal matching for* G

PROOF. The proof hinges on the fact that high degree nodes have sufficiently high degree that it does not matter which edges we remember for them beyond a point. Specifically, suppose we have found a maximal matching of size at most k on G'. Assume that there is a node in H (of degree more than $2k$) that is unmatched. At most $2k$ of its neighbors are matched, due to the promise that the matching is

size at most k. Then, it must have an unmatched neighbor which was can find from the (k sample recovery) sketch, since the sketch recovers up to $2k+1$ neighbors of the node. This contradicts the claim that the matching was maximal.

Then, all nodes in H are matched. Thus, any edge which is incident on a node in H cannot extend the matching. L consists of the set of all remaining edges, hence this represents sufficient information to ensure that the whole matching is maximal. Consequently, the matching we find on G' is also maximal on G. □

Note: Note that the operation of the algorithm as updates are processed depends only on deterministic algorithms (such as the k-sparse recovery algorithms - see Definition 1). The only time a randomized data structure is made use of is in the algorithm to find a matching, where we probe the k-sample algorithms. This greatly simplifies the analysis as compared to that in [2]

3. POINT LINE COVER

In the *Point Line Cover* problem, we are given a set P of n points and the question is to find minimum number of lines which can cover all the given n points. In the parametereized version of the problem, we are given an integer k and the question is whether there exists a set of k lines which can cover all the n points. There is an (known) easy kernel of size $O(k^2)$ (see next paragraph), which was shown to be essentially tight by Kratsch et al. [3]

$O(k^2)$ **kernel for Point Line Cover**: We perform the following procedure iteratively:

- **Step 1**: Check if there exists a line ℓ which contains at least $k+1$ points
- **Step 2**: If YES then include this line in the solution, delete all the points which lie on this line and decrease k by 1. If $k > 0$, then go to Step 1. If $k = 0$ and there are still some points remaining then the given instance is a NO instance.
- **Step 3**: Otherwise if $n > k^2$ then the given instance is a NO instance.

We can check if a given line contains at least $k+1$ points in time $O(n)$: for each of the n points we just check individually if they lie on the line or not. There are $\binom{n}{2} = O(n^2)$ lines: one determined by each pair of points. So each execution of Step 1 takes $O(n^3)$ time. Since we execute Step 1 at most k times (note the parameter decreases each time we get a YES answer for Step 1, which is the only time we run Step 1 again). Hence the total running time is $O(n^3 \cdot k)$.

Now we show correctness of the kernelization algorithm. If there is a line which contains at least $k+1$ points, then we must include it in our solution; since otherwise we will need at lest $k+1$ different lines to cover each of these points. We continue this process until Step 1 answers NO or k becomes 0. If $k = 0$ and we have any points remaining to cover then the instance is obviously a NO instance. Otherwise if Step 1 answers NO, then this means that any line can cover at most k points. Hence an instance which can be covered with at most k lines can have at most k^2 points. This justifies Step 3, and shows the correctness of the kernelization algorithm.

Algorithm for Point Line Cover under Promise: We now show how to obtain an $\tilde{O}(k^2)$ algorithm for finding minimum Point Line Cover under the following promise:

ASSUMPTION 7. (promise) *We assume that, at every point during the execution of the algorithm, the number of lines needed to cover all the points is at most k.*

Given a stream of insertions and deletions of points, we maintain a kernel as follows. Let L be a set of lines such that any line in L contains at most k points. We denote set of lines containing more than k points each by H. For a line in L we just keep all the points which lie on it, and for each line in H we sample $k+1$ points using a $(k+1)$-sparse recovery algorithm. The maintenance of sets L and H, and commuting of lines in sets S and H upon point insertions and deletions is similar to the algorithm in Section 2.

4. REFERENCES

[1] N. Barkay, E. Porat, and B. Shalem. Efficient sampling of non-strict turnstile data streams. In *FCT 2013*, pages 48–59.

[2] R. H. Chitnis, G. Cormode, M. T. Hajiaghayi, and M. Monemizadeh. Parameterized streaming: Maximal matching and vertex cover. In *SODA 2015*, pages 1234–1251.

[3] S. Kratsch, G. Philip, and S. Ray. Point line cover: The easy kernel is essentially tight. In *SODA 2014*, pages 1596–1606.