Densest Subgraph in Dynamic Graph Streams

Andrew McGregor[*] David Tench[*] Sofya Vorotnikova[*] Hoa T. Vu[*]

Abstract. In this paper, we consider the problem of approximating the densest subgraph in the dynamic graph stream model. In this model of computation, the input graph is defined by an arbitrary sequence of edge insertions and deletions and the goal is to analyze properties of the resulting graph given memory that is sub-linear in the size of the stream. We present a single-pass algorithm that returns a $(1+\epsilon)$ approximation of the maximum density with high probability; the algorithm uses $O(\epsilon^{-2} n \operatorname{polylog} n)$ space, processes each stream update in $\operatorname{polylog}(n)$ time, and uses $\operatorname{poly}(n)$ post-processing time where n is the number of nodes. The space used by our algorithm matches the lower bound of Bahmani et al. (PVLDB 2012) up to a poly-logarithmic factor for constant ϵ. The best existing results for this problem were established recently by Bhattacharya et al. (STOC 2015). They presented a $(2+\epsilon)$ approximation algorithm using similar space and another algorithm that both processed each update and maintained a $(4+\epsilon)$ approximation of the current maximum density in $\operatorname{polylog}(n)$ time per-update.

1 Introduction

In the dynamic graph stream model of computation, a sequence of edge insertions and deletions defines an input graph and the goal is to solve a specific problem on the resulting graph given only one-way access to the input sequence and limited working memory. Motivated by the need to design efficient algorithms for processing massive graphs, over the last four years there has been a considerable amount of work designing algorithms in this model [1–5,8,9,11,17,19,21,22,24,25]. Specific results include testing edge connectivity [3] and node connectivity [19], constructing spectral sparsifiers [21], approximating the densest subgraph [8], maximum matching [5,9,11,24], correlation clustering [1], and estimating the number of triangles [25]. For a recent survey of the area, see [27].

In this paper, we consider the densest subgraph problem. Let G_U be the induced subgraph of graph $G=(V,E)$ on nodes U. Then the *density* of G_U is defined as

$$d(G_U) = |E(G_U)|/|U| ,$$

[*]University of Massachusetts Amherst. This work was supported by NSF Awards CCF-0953754, IIS-1251110, CCF-1320719, and a Google Research Award. {mcgregor,dtench,svorotni,hvu}@cs.umass.edu.

where $E(G_U)$ is the set of edges in the induced subgraph. We define the *maximum density* as

$$d^* = \max_{U \subseteq V} d(G_U) \ .$$

and say that the corresponding subgraph is the *densest subgraph*. The densest subgraph can be found in polynomial time [10, 15, 18, 23] and more efficient approximation algorithms have been designed [10]. Finding dense subgraphs is an important primitive when analyzing massive graphs; applications include community detection in social networks and identifying link spam on the web, in addition to applications on financial and biological data. See [26] for a survey of applications and existing algorithms for the problem.

1.1 Our Results and Previous Work

We present a single-pass algorithm that returns a $(1 + \epsilon)$ approximation with high probability[1]. For a graph on n nodes, the algorithm uses the following resources:

- *Space:* $O(\epsilon^{-2} n \operatorname{polylog} n)$. The space used by our algorithm matches the lower bound of Bahmani et al. [7] up to a poly-logarithmic factor for constant ϵ.
- *Per-update time:* $\operatorname{polylog}(n)$. We note that this is the worst-case update time rather than amortized over all the edge insertions and deletions.
- *Post-processing time:* $\operatorname{poly}(n)$. This will follow by using any exact algorithm for densest subgraph [10, 15, 18] on the subgraph generated by our algorithm.

The most relevant previous results for the problem were established recently by Bhattacharya et al. [8]. They presented two algorithms that use similar space to our algorithm and process updates in $\operatorname{polylog}(n)$ amortized time. The first algorithm returns a $(2 + \epsilon)$ approximation of the maximum density of the final graph while the second (the more technically challenging result) outputs a $(4 + \epsilon)$ approximation of the current maximum density after every update while still using only $\operatorname{polylog}(n)$ time per-update. Our algorithm improves the approximation factor to $(1+\epsilon)$ while keeping the same space and update time. It is possible to modify our algorithm to output a $(1 + \epsilon)$ approximation to the current maximum density after each update but the simplest approach would require the post-processing step to be run after every edge update and this would not be efficient.

Bhattacharya et al. were one of the first to combine the space restriction of graph streaming with the fast update and query time requirements of fully-dynamic algorithms from the dynamic graph algorithms community. Epasto, Lattanzi, and Sozio [14] present a fully-dynamic algorithm that returns a $(2 + \epsilon)$ approximation of the current maximum density. Other relevant work includes papers by Bahmani, Kumar, and Vassilvitskii [7] and Bahmani, Goel, and Munagala [6]. The focus of these papers is on designing algorithms in the MapReduce model but the resulting algorithms can also be implemented in the data stream model if we allow multiple passes over the data.

[1] Throughout this paper, we say an event holds with high probability if the probability is at least $1 - n^{-c}$ for some constant $c > 0$.

1.2 Our Approach and Paper Outline

The approach we take in this paper is as follows. In Section 2, we show that if we sample every edge of a graph independently with a specific probability then we generate a graph that is a) sparse and b) can be used to estimate the maximum density of the original graph. This is not difficult to show but requires care since there are an exponential number of subgraphs in the subsampled graph that we will need to consider.

In Section 3, we show how to perform this sampling in the dynamic graph stream model. This can be done using the ℓ_0 sampling primitive [12,20] that enables edges to be sampled uniformly from the set of edges that have been inserted but not deleted. However, a naive application of this primitive would necessitate $\Omega(n)$ per-update processing. To reduce this to $O(\mathrm{polylog}\, n)$ we reformulate the sampling procedure in such a way that it can be performed more efficiently. This reformulation is based on creating multiple partitions of the set of edges using pairwise independent hash functions and then sampling edges within each group in the partition. The use of multiple partitions is somewhat reminiscent of that used in the Count-Min sketch [13].

2 Subsampling Approximately Preserves Maximum Density

In the section, we consider properties of a random subgraph of the input graph G. Specifically, let G' be the graph formed by sampling each edge in G independently with probability p where

$$p = c\epsilon^{-2} \log n \cdot \frac{n}{m}$$

for some sufficiently large constant $c > 0$ and $0 < \epsilon < 1/2$. We may assume that m is sufficiently large such that $p < 1$ because otherwise we can reconstruct the entire graph in the allotted space using standard results from the sparse recovery literature [16].

We will prove that, with high probability, the maximum density of G can be estimated up to factor $(1+\epsilon)$ given G'. While it is easy to analyze how the density of a specific subgraph changes after the edge sampling, we will need to consider all 2^n possible induced subgraphs and prove properties of the subsampling for all of them.

The next lemma shows that $d(G'_U)$ is roughly proportional to $d(G_U)$ if $d(G_U)$ is "large" whereas if $d(G_U)$ is "small" then $d(G'_U)$ will also be relatively small.

Lemma 1. *Let U be an arbitrary set of k nodes. Then,*

$$\mathbb{P}\left[d(G'_U) \geq pd^*/10\right] \leq n^{-10k} \qquad \text{if } d(G_U) \leq d^*/60$$
$$\mathbb{P}\left[|d(G'_U) - pd(G_U)| \geq \epsilon p d(G_U)\right] \leq 2n^{-10k} \qquad \text{if } d(G_U) > d^*/60\ .$$

Proof. We start by considering the density of the entire graph $d(G) = m/n$ and therefore conclude that the maximum density, d^*, is at least m/n. Hence, $p \geq (c\epsilon^{-2}\log n)/d^*$.

Let X be the number of edges in G'_U and note that $\mathbb{E}[X] = pkd(G_U)$. First assume $d(G_U) \leq d^*/60$. Then, by an application of the Chernoff Bound (e.g., [28, Theorem 4.4]), we observe that

$$\mathbb{P}\left[d(G'_U) \geq pd^*/10\right] = \mathbb{P}\left[X \geq pkd^*/10\right] \leq 2^{-pkd^*/10} < 2^{-ck(\log n)/10}$$

and this is at most n^{-10k} for sufficiently large constant c.

Next assume $d(G_U) > d^*/60$. Hence, by an application of an alternative form of the Chernoff Bound (e.g., [28, Theorem 4.4 and 4.5]), we observe that

$$\begin{aligned}\mathbb{P}\left[|d(G'_U) - pd(G_U)| \geq \epsilon pd(G_U)\right] &= \mathbb{P}\left[|X - pkd(G_U)| \geq \epsilon pkd(G_U)\right] \\ &\leq 2\exp(-\epsilon^2 pkd(G_U)/3) \\ &\leq 2\exp(-\epsilon^2 pkd^*/180) \\ &\leq 2\exp(-ck(\log n)/180) \ .\end{aligned}$$

and this is at most $2n^{-10k}$ for sufficiently large constant c. □

Corollary 1. *With high probability, for all $U \subseteq V$:*

$$d(G'_U) \geq (1-\epsilon)pd^* \quad \Rightarrow \quad d(G_U) \geq \frac{1-\epsilon}{1+\epsilon} \cdot d^* \ .$$

Proof. There are $\binom{n}{k} \leq n^k$ subsets of V that have size k. Hence, by appealing to Lemma 1 and the union bound, with probability at least $1 - 2n^{-9k}$, the following two equations hold,

$$\begin{aligned} d(G'_U) \geq pd^*/10 &\quad \Rightarrow \quad d(G_U) > d^*/60 \\ d(G_U) > d^*/60 &\quad \Rightarrow \quad d(G_U) \geq \frac{d(G'_U)}{p(1+\epsilon)} \end{aligned}$$

for all $U \subseteq V$ such that $|U| = k$. Since $(1-\epsilon)pd^* \geq pd^*/10$, together these two equations imply

$$d(G'_U) \geq (1-\epsilon)pd^* \quad \Rightarrow \quad d(G_U) \geq \frac{d(G'_U)}{p(1+\epsilon)} \geq \frac{1-\epsilon}{1+\epsilon} \cdot d^*$$

for all sets U of size k. Taking the union bound over all values of k establishes the corollary. □

We next show that the densest subgraph in G' corresponds to a subgraph in G that is almost as dense as the densest subgraph in G.

Theorem 1. *Let $U' = \operatorname{argmax}_U d(G'_U)$. Then with high probability,*

$$\frac{1-\epsilon}{1+\epsilon} \cdot d^* \leq d(G_{U'}) \leq d^* \ .$$

Proof. Let $U^* = \operatorname{argmax}_U d(G_U)$. By appealing to Lemma 1, we know that $d(G'_{U^*}) \geq (1-\epsilon)pd^*$ with high probability. Therefore

$$d(G'_{U'}) \geq d(G'_{U^*}) \geq (1-\epsilon)pd^* \ ,$$

and the result follows by appealing to Corollary 1. □

3 Implementing in the Dynamic Data Stream Model

In this section, we show how to sample each edge independently with the prescribed probability in the dynamic data stream model. The resulting algorithm uses $O(\epsilon^{-2} n \operatorname{polylog} n)$ space. The near-linear dependence on n almost matches the $\Omega(n)$ lower bound proved by Bahmani et al. [7]. The main theorem we prove is:

Theorem 2. *There exists a randomized algorithm in the dynamic graph stream model that returns a $(1+\epsilon)$-approximation for the density of the densest subgraph with high probability. The algorithm uses $O(\epsilon^{-2} n \operatorname{polylog} n)$ space and $O(\operatorname{polylog} n)$ update time. The post-processing time of the algorithm is polynomial in n.*

To sample the edges with probability p in the dynamic data stream model there are two main challenges:

1. Any edge we sample during the stream may subsequently be deleted.
2. Since p depends on m, we do not know the value of p until the end of the stream.

To address the first challenge, we appeal to an existing result on the ℓ_0 sampling technique [20]: there exists an algorithm using $\operatorname{polylog}(n)$ space and update time that returns an edge chosen uniformly at random from the final set of edges in the graph. Consequently we may sample r edges uniformly at random using $O(r \operatorname{polylog} n)$ update time and space. To address the fact we do not know p apriori, we could set $r \gg pm = c\epsilon^{-2} n \log n$, and then, at the end of the stream when p and m are known a) choose $X \sim \mathbf{Bin}(m,p)$ where $\mathbf{Bin}(\cdot,\cdot)$ denotes the binomial distribution and b) randomly pick X distinct random edges amongst the set of r edges sampled (ignoring duplicates). This approach will work with high probability if r is sufficiently large since X is tightly concentrated around $\mathbb{E}[X] = pm$. However, a naive implementation of this algorithm would require $\omega(n)$ update time. The main contribution of this section is to demonstrate how to ensure $O(\operatorname{polylog} n)$ update time.

3.1 Reformulating the Sampling Procedure

We first describe an alternative sampling process that, with high probability, returns a set of edges S where each edge in S has been sampled independently with probability p as required. The purpose of this alternative formulation is that it will allow us to argue that it can be emulated in the dynamic graph stream model efficiently.

Basic Approach. The basic idea is to partition the set of edges into different groups and then sample edges within groups that do not contain too many edges. We refer to such groups as "small". We determine which of the edges in a small group are to be sampled in two steps:

– *Fix the number X of edges to sample:* Let $X \sim \mathbf{Bin}(g,p)$ where g is the number of edges in the relevant group.
– *Fix which X edges to sample:* We then randomly pick X edges without replacement from the relevant group.

It is not hard to show that this two-step process ensures that each edge in the group is sampled independently with probability p. At this point, the fate of all edges in small groups has been decided: they will either be returned in the final sample or definitely not returned in the final sample.

We next consider another partition of the edges and again consider groups that do not contain many edges. We then determine the fate of the edges in such groups whose fate has not hitherto been determined. We keep on considering different partitions until every edge has been included in a small group and has had its fate determined.

Lemma 2. *Assume for every edge there exists a partition such that the edge is in a small group. Then the distribution over sets of sampled edges is the same as the distribution had each edge been sampled independently with probability p.*

Proof. The proof does not depend on the exact definition of "small" and the only property of the partitions that we require is that every edge is in a small group of some partition. We henceforth consider a fixed set of partitions with this property.

We first consider the jth group in the ith partition. Let g be the number of edges in this group. For any subset Q of ℓ edges in this group, we show that the probability that Q is picked by the two-step process above is indeed p^ℓ.

$$\begin{aligned}\mathbb{P}\left[\forall e \in Q, e \text{ is picked}\right] &= \sum_{t=\ell}^{g} \mathbb{P}\left[\forall e \in Q, e \text{ is picked } | X = t\right] \mathbb{P}\left[X = t\right] \\ &= \sum_{t=\ell}^{g} \frac{\binom{g-\ell}{t-\ell}}{\binom{g}{t}} \cdot \binom{g}{t} \cdot p^t (1-p)^{g-t} \\ &= p^\ell \sum_{t=\ell}^{g} \binom{g-\ell}{t-\ell} \cdot p^{t-\ell}(1-p)^{g-t} = p^\ell .\end{aligned}$$

and hence edges within the same group are sampled independently with probability p. Furthermore, the edges in different groups of the same partition are sampled independently from each other.

Let $f(e)$ be the first partition in which e is placed in a group that is small and let $W_i = \{e : f(e) = i\}$. Restricting Q to edges in W_i in the above analysis establishes that edges in each W_i are sampled independently. Since $f(e)$ is determined by the fixed set of partitions rather than the randomness of the sampling procedure, we also conclude that edges in different W_i are sampled independently. As we assume that every edge belongs to at least one small group in some partition, if we let r be the total number of partitions, then $\{W_i\}_{i\in[r]}$ partition the set of edges E. Hence, all edges in E are sampled independently with probability p. $\square$

Details of Alternative Sampling Procedure. The partitions considered will be determined by pairwise independent hash functions and we will later argue that it is sufficient to consider only $O(\log n)$ partitions. Each hash function will partition the m edges into $n\epsilon^{-2}$ groups. In expectation the number of edges in a group will be $\epsilon^2 m/n$ and we define a group to be small if it contains at most $t = 4\epsilon^2 m/n$ edges. We therefore expect to sample less than $4p\epsilon^2 m/n = 4c \log n$ edges from a small group. We will abort the algorithm if we attempt to sample significantly more edges than this from some small group. The procedure is as follows:

– Let $h_1, \ldots, h_r : \binom{n}{2} \to [n\epsilon^{-2}]$ be pairwise independent hash functions where $r = 10 \log n$.
– Each h_i defines a partition of E comprising of sets of the form
$$E_{i,j} = \{e \in E : h_i(e) = j\} .$$
Say $E_{i,j}$ is *small* if it is of size at most $t = 4\epsilon^2 m/n$. Let D_i be the set of all edges in the small sets determined by h_i.
– For each small $E_{i,j}$, let
$$X_{i,j} = \mathbf{Bin}(|E_{i,j}|, p)$$
and abort if
$$X_{i,j} \geq \tau \quad \text{where} \quad \tau = 24c \log n .$$
Let $S_{i,j}$ be a set of $X_{i,j}$ edges sampled without replacement from $E_{i,j}$.
– Let S be set of edges that were sampled among some D_i that are not in $D_1 \cup D_2 \cup \ldots \cup D_{i-1}$, i.e., edges whose fate had not already been determined.
$$S = \bigcup_{i=1}^{r} \{e \in D_i : e \in \cup_j S_{i,j} \text{ and } e \notin D_1 \cup D_2 \cup \ldots \cup D_{i-1}\}$$

Analysis. There are two main things that we need to show to establish that the above process emulates our basic sampling approach with high probability. First, we will show that with high probability for every edge e there exists i and j such that $e \in E_{i,j}$ and $E_{i,j}$ is small. This ensures that we will make a decision on whether e is included in the final sample. Second, we will show that it is very unlikely we abort because some $X_{i,j}$ is too large.

Lemma 3. *With probability at least $1 - n^{-8}$, for every edge e there exists i such that $e \in E_{i,j}$ and $E_{i,j}$ is small.*

Proof. Fix $i \in [r]$ and let $j = h_i(e)$. Then $\mathbb{E}[|E_{i,j}|] \leq 1 + \epsilon^2(m-1)/n \leq 2\epsilon^2 m/n$ assuming $m \geq \epsilon^{-2} n$. By an application of the Markov bound:
$$\mathbb{P}\left[|E_{i,j}| \geq 4m\epsilon^2/n\right] \leq 1/2 .$$
Since each h_i is independent,
$$\mathbb{P}\left[|E_{i,h_i(e)}| \geq 4m\epsilon^2/n \text{ for all } i\right] \leq 1/2^r = 1/n^{10} .$$
Therefore by the union bound over all $m \leq n^2$ edges there exists a good partition for each e with probability at least $1 - n^{-8}$. □

Lemma 4. *With high probability, all $X_{i,j}$ are less than $\tau = 24c \log n$.*

Proof. Since $E_{i,j}$ is small then $\mathbb{E}[X_{i,j}] = |E_{i,j}|p \leq 4\epsilon^2 pm/n = 4c \log n$. Hence, by an application of the Chernoff bound,
$$\mathbb{P}[X_{i,j} \geq 24c \log n] \leq 2^{-24c \log n} \leq n^{-10} .$$
Taking the union bound over all $10 \log n$ values of i and $\epsilon^{-2} n$ values of j establishes the lemma. □

3.2 The Dynamic Graph Stream Algorithm

We are now ready to present the dynamic graph stream algorithm. To emulate the above sampling process in the dynamic graph stream model, we proceed as follows:

1. *Pre-Processing:* Pick the hash functions $h_1, h_2, \ldots, h_r$. These define the sets $E_{i,j}$.
2. *During One Pass:*
   - Compute the size of each $E_{i,j}$ and m. Note that m is necessary to define p.
   - Sample τ edges $S'_{i,j}$ uniformly without replacement from each $E_{i,j}$.
3. *Post-Processing:*
   - Randomly determine the values $X_{i,j}$ based on the exact values of $|E_{i,j}|$ and m for each $E_{i,j}$ that is small. If $X_{i,j}$ exceeds τ then abort.
   - Let $S_{i,j}$ be a random subset of $S'_{i,j}$ of size $X_{i,j}$.
   - Return $p^{-1} \max_U d(G'_U)$ where G' is the graph with edges:

$$S = \bigcup_{i=1}^{r} \{e \in D_i : e \in \cup_j S_{i,j} \text{ and } e \notin D_1 \cup D_2 \cup \ldots \cup D_{i-1}\}$$

Note that is possible to compute $|E_{i,j}|$ using a counter that is incremented or decremented whenever an edge e is added or removed respectively that satisfies $h_i(e) = j$. We may evaluate pairwise independent hash functions in $O(\text{polylog}\, n)$ time. The exact value of $\max_U d(G'_U)$ can be determined in polynomial time using the result of Charikar [10]. To prove Theorem 2, it remains to describe how to sample τ edges *without replacement* from each $E_{i,j}$.

Sampling Edges Without Replacement Via ℓ_0-Sampling. To do this, we use the ℓ_0-sampling algorithm of Jowhari et al. [20]. Their algorithm returns, with high probability, a random edge from $E_{i,j}$ and the space and update time of the algorithm are both $O(\text{polylog}\, n)$. Running τ independent instantiations of this algorithm immediately enables us to sample τ edges uniformly from $E_{i,j}$ *with replacement*.

However, since their algorithm is based on linear sketches, there is an elegant way (at least, more elegant than simply over sampling and removing duplicates) to ensure that all samples are distinct. Specifically, let $\mathbf{x}$ be the characteristic vector of the set $E_{i,j}$. Then, τ instantiations of the algorithm of Jowhari et al. [20] generate random projections

$$\mathcal{A}_1(\mathbf{x})\ ,\ \mathcal{A}_2(\mathbf{x})\ ,\ \ldots\ ,\ \mathcal{A}_\tau(\mathbf{x})$$

of $\mathbf{x}$ such that a random non-zero entry of $\mathbf{x}$ (which corresponds to an edge from $E_{i,j}$) can be identified by processing each $\mathcal{A}_i(\mathbf{x})$. Let e_1 be the edge reconstructed from $\mathcal{A}_1(\mathbf{x})$. Rather than reconstructing an edge from $\mathcal{A}_2(\mathbf{x})$, which could be the same as e_1, we instead reconstruct an edge e_2 from

$$\mathcal{A}_2(\mathbf{x}) - \mathcal{A}_2(\mathbf{i}_{e_1}) = \mathcal{A}_2(\mathbf{x} - \mathbf{i}_{e_1})$$

where $\mathbf{i}_{e_1}$ is the characteristic vector of the set $\{e_1\}$. Note that e_2 is necessarily different from e_1 since $\mathbf{x} - \mathbf{i}_e$ is the characteristic vector of the set $E_{i,j} \setminus \{e_1\}$. Similarly we reconstruct e_j from

$$\mathcal{A}_j(\mathbf{x}) - \mathcal{A}_j(\mathbf{i}_{e_1}) - \mathcal{A}_j(\mathbf{i}_{e_2}) - \ldots - \mathcal{A}_j(\mathbf{i}_{e_{j-1}}) = \mathcal{A}_2(\mathbf{x} - \mathbf{i}_{e_1} - \ldots - \mathbf{i}_{e_{j-1}})$$

and note that e_j is necessarily distinct from $\{e_1, e_2, \ldots, e_{j-1}\}$.

4 Conclusion

We presented the first algorithm for estimating the density of the densest subgraph up to a $(1+\epsilon)$ factor in the dynamic graph stream model. Our algorithm used $O(\epsilon^{-2} n \operatorname{polylog} n)$ space, $\operatorname{polylog}(n)$ per-update processing time, and $\operatorname{poly}(n)$ post-processing to return the estimate. The most relevant previous results, by Bhattacharya et al. [8], were a $(2+\epsilon)$ approximation in similar space and a $(4+\epsilon)$ approximation with $\operatorname{polylog}(n)$ per-update processing time that also outputs an estimate of the maximum density after each edge insertion or deletion. A natural open question is whether it is possible to use ideas contained in this paper to improve the approximation factor for the problem of maintaining a running estimate of the maximum density.

References

1. K. J. Ahn, G. Cormode, S. Guha, A. McGregor, and A. Wirth. Correlation clustering in data streams. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, July 6 - 11, 2015*, 2015.
2. K. J. Ahn, S. Guha, and A. McGregor. Analyzing graph structure via linear measurements. In *Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012*, pages 459–467, 2012.
3. K. J. Ahn, S. Guha, and A. McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 5–14, 2012.
4. K. J. Ahn, S. Guha, and A. McGregor. Spectral sparsification in dynamic graph streams. In *APPROX*, pages 1–10, 2013.
5. S. Assadi, S. Khanna, Y. Li, and G. Yaroslavtsev. Tight bounds for linear sketches of approximate matchings. *CoRR*, abs/1505.01467, 2015.
6. B. Bahmani, A. Goel, and K. Munagala. Efficient primal-dual graph algorithms for mapreduce. In *Algorithms and Models for the Web Graph - 11th International Workshop, WAW 2014, Beijing, China, December 17-18, 2014, Proceedings*, pages 59–78, 2014.
7. B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and mapreduce. *PVLDB*, 5(5):454–465, 2012.
8. S. Bhattacharya, M. Henzinger, D. Nanongkai, and C. E. Tsourakakis. Space- and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *STOC*, 2015.
9. M. Bury and C. Schwiegelshohn. Sublinear estimation of weighted matchings in dynamic data streams. *CoRR*, abs/1505.02019, 2015.
10. M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Approximation Algorithms for Combinatorial Optimization, Third International Workshop, APPROX 2000, Saarbrücken, Germany, September 5-8, 2000, Proceedings*, pages 84–95, 2000.
11. R. H. Chitnis, G. Cormode, H. Esfandiari, M. Hajiaghayi, A. McGregor, M. Monemizadeh, and S. Vorotnikova. Kernelization via sampling with applications to dynamic graph streams. *CoRR*, abs/1505.01731, 2015.
12. G. Cormode and D. Firmani. A unifying framework for ℓ_0-sampling algorithms. *Distributed and Parallel Databases*, 32(3):315–335, 2014.
13. G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

14. A. Epasto, S. Lattanzi, and M. Sozio. Efficient densest subgraph computation in evolving graphs. In *WWW*, 2015.
15. G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.*, 18(1):30–55, 1989.
16. A. C. Gilbert and P. Indyk. Sparse recovery using sparse matrices. *Proceedings of the IEEE*, 98(6):937–947, 2010.
17. A. Goel, M. Kapralov, and I. Post. Single pass sparsification in the streaming model with edge deletions. *CoRR*, abs/1203.4900, 2012.
18. A. V. Goldberg. Finding a maximum density subgraph. Technical report, Berkeley, CA, USA, 1984.
19. S. Guha, A. McGregor, and D. Tench. Vertex and hypergraph connectivity in dynamic graph streams. In *PODS*, 2015.
20. H. Jowhari, M. Saglam, and G. Tardos. Tight bounds for lp samplers, finding duplicates in streams, and related problems. In *PODS*, pages 49–58, 2011.
21. M. Kapralov, Y. T. Lee, C. Musco, C. Musco, and A. Sidford. Single pass spectral sparsification in dynamic streams. In *FOCS*, 2014.
22. M. Kapralov and D. P. Woodruff. Spanners and sparsifiers in dynamic streams. In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 272–281, 2014.
23. S. Khuller and B. Saha. On finding dense subgraphs. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pages 597–608, 2009.
24. C. Konrad. Maximum matching in turnstile streams. *CoRR*, abs/1505.01460, 2015.
25. K. Kutzkov and R. Pagh. Triangle counting in dynamic graph streams. In *Algorithm Theory - SWAT 2014 - 14th Scandinavian Symposium and Workshops, Copenhagen, Denmark, July 2-4, 2014. Proceedings*, pages 306–318, 2014.
26. V. Lee, N. Ruan, R. Jin, and C. Aggarwal. A survey of algorithms for dense subgraph discovery. In C. C. Aggarwal and H. Wang, editors, *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*, pages 303–336. Springer US, 2010.
27. A. McGregor. Graph stream algorithms: a survey. *SIGMOD Record*, 43(1):9–20, 2014.
28. M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, New York, NY, USA, 2005.