Computing the Degeneracy of Large Graphs*

Martín Farach-Colton and Meng-Tsung Tsai

Rutgers University, New Brunswick NJ 08901, USA
{farach,mtsung.tsai}@cs.rutgers.edu

Abstract. Any ordering of the nodes of an n-node, m-edge simple undirected graph G defines an acyclic orientation of the edges in which each edge is oriented from the earlier node in the ordering to the later. The *degeneracy* on an ordering is the maximum outdegree it induces, and the *degeneracy* of a graph is smallest degeneracy of any node ordering. Small-degeneracy orderings have many applications.

We give an algorithm for generating an ordering whose degeneracy approximates the minimum possible, that is, it approximates the degeneracy of the graph. Although the optimal ordering itself can be computed in $\mathcal{O}(m)$ time and $\mathcal{O}(m)$ space, such algorithms are infeasible for large graphs. Our approximation algorithm is semi-streaming: it uses less space, can achieve a constant approximation ratio, and accesses the graph in logarithmic read-only passes.

1 Introduction

Any ordering of the nodes of an n-node, m-edge simple undirected graph G defines an acyclic orientation of the edges in which each edge is oriented from the earlier node in the ordering to the later. The *degeneracy* of an ordering is the maximum outdegree it induces. The *degeneracy*, $d(G)$, of G is the smallest degeneracy of any ordering[1], and an ordering whose degeneracy is $d(G)$ is called a *degenerate ordering*. An ordering is *d-degenerate* if it has degeneracy at most d.

Degenerate orderings have many uses. Given a degenerate ordering, one can: decompose a graph into at most twice to the minimum number of disjoint forests [2,4]; decompose a graph into at most six times to the minimum number of disjoint planar graphs [4,11]; speed up the counting of the number of short paths or cycles [2], for example, counting the exact number of 3-cycles in $\mathcal{O}(md(G))$ time; find a component of density at least half the maximum density of any subgraph, i.e. a 1/2-approximation [7]; identify a dominating set of cardinality at most $\mathcal{O}(d^2(G))$ times the cardinality of minimum dominating set [19] and some variations of dominating set [12], e.g. k-dominating set; etc. Although most of

* Work supported by NSF Grants IIS-1247750 and CCF-1114930.

[1] The degeneracy of a graph was originally defined to be the maximum of minimum degree among all subgraphs [2,4,5,7,14,22,26]. The definition here is a slight modification of the *coloring number* [4,5,14] of a graph, a dual definition of degeneracy. The coloring number of a graph was shown to be one larger than the degeneracy [4,5,14], and our definition yields the same value as the original definition of degeneracy.

A. Pardo and A. Viola (Eds.): LATIN 2014, LNCS 8392, pp. 250–260, 2014.

these problems can be solved exactly in polynomial time and space $\mathcal{O}(n)$ to $\mathcal{O}(m)$ [7,15,16,18], the approximation algorithms based on degenerate orderings are faster, use less space or yield better approximation factors for large graphs. For example, such orderings yield a better approximation algorithm for decomposing a graph into minimum number of planar subgraphs than other algorithms using $O(n)$ space [17, 21]. Although all of the results listed originally relied on (optimally) degenerate orderings, we show that orderings that are nearly degenerate, that is, orderings whose degeneracy approximates rather than matching the graph degeneracy, also yield good approximation algorithms.

Known algorithms to compute a low-degeneracy ordering do not scale to graphs that are larger than memory. Several models of computation have been proposed for computing on large graphs, such as the restrictive *semi-streaming* [13, 23–25, 27] model that allows only $\mathcal{O}(n \operatorname{polylog} n)$ space and sequential read-only passes through the graph, the *W-stream* [24,27] model which is similar but also allows the algorithm $\mathcal{O}(m)$-size read-write space on disk, the *Stream-Sort* [24, 25, 27] model, in which sorting the graph needs only one pass, and the *Stream-with-annotations* [6] model that as *semi-streaming* but assume a powerful helper can be queried for a small number of annotations. When read-write space is restricted to $\mathcal{O}(n \operatorname{polylog} n)$ space and accessing the graph is restricted to a constant or logarithmic number of sequential passes on entire the graph, some graph problems, e.g. connectivity or minimum spanning tree, have known optimal solutions. Other graph problems, e.g. counting the number of 3-cycles and maximum matching, can be approximated [1,3]. Some fundamental problems, such as breath-first search, depth-first search, topological sorting, and directed connectivity, are believed to be difficult [24,25].

All known algorithms for computing degenerate orders have a structure that is similar to topological sort. Therefore, we seek to approximate the graph degeneracy. We use the *semi-streaming* model; that is, $\mathcal{O}(n \operatorname{polylog} n)$ space and constant/logarithm sequentially read-only passes on the entire graph are allowed, which is the most restricted model among the mentioned three. Our goal is to minimize the number of passes on the disk while finding a node ordering of low degeneracy, in particular one whose degeneracy is a good approximation of the degeneracy of the graph.

A simple semi-streaming algorithm can find a $\sqrt{n}d(G)$-degenerate ordering of nodes in one pass, by sorting the node by (undirected) degree [26]. We improve the approximation factor to a constant at the cost of a logarithmic number of passes while maintaining n working space. Our algorithm can be made to use space that is less than n, and as our space usage decreases, our approximation factor degrades. Our algorithm outputs a sequence of some subset of nodes in each pass. At the end of all passes, the concatenation of all sequences is the desired ordering.

Theorem 1. *Given a simple undirected n-node m-edge graph G, an $\alpha d(G)$-degenerate ordering of nodes of G, i.e. an α-approximation, can be computed in $\mathcal{O}((m+n)\mathcal{P})$ time, using $s(n)$ space and $\mathcal{P} = \mathcal{O}(\log_{1+\varepsilon/2} n/s(n) + \log_{\alpha/2} s(n))$*

sequential passes on the entire graph, where $\alpha = (2+\varepsilon)n/s(n)$ *for any* $\varepsilon > 0$ *and* $1 \leq s(n) \leq n$.

Note that α is inversely related to space $s(n)$. For example, if $s(n) = n/10$, then $\alpha = 20(1+\varepsilon)$, and if $s(n) = n/\log n$, let $\alpha = 2\log n(1+\varepsilon)$, for any $\varepsilon > 0$. In addition, it is possible to perform fewer than $\log n$ passes: our algorithm requires $\mathcal{O}(\log\log n/\varepsilon + \log n/\log\log n)$ passes, for small $\varepsilon > 0$, when $s(n) = \mathcal{O}(n/\operatorname{polylog} n)$.

Organizations. In Section 2, we revisit some properties of graph degeneracy and give a sketch of our algorithms. We propose the space-efficient approximation algorithms in Section 3 and analyze their complexities. Last, in Section 4, we discuss how the found ordering be applied to applications.

2 Preliminaries

We begin by reviewing some known results about degeneracy. A greedy algorithm finds $d(G)$ and a corresponding ordering of nodes in $\mathcal{O}(m)$ time using $\mathcal{O}(m)$ space [5, 22]. The greedy algorithm is based on the following two observations: $d(G)$ is at least the minimum degree, $\delta(G)$, of G, because the first node in any ordering has outdegree equal to its (undirected) degree; $d(G) \geq d(H)$ for any subgraph $H \subseteq G$, since one can apply the optimal orientation for G to subgraph H. Hence, the algorithm can greedily pick node v of minimum degree as the first node in the ordering and reduce the graph G to a subgraph $G \setminus v$. This greedy step will not increase the maximum out-degree in the resulting ordering because

$$\max\left\{d(v), d(G \setminus v)\right\} \leq d(G).$$

This greedy algorithm needs to update the degree of nodes next to v in order to find the node of minimum degree in $G \setminus v$, which requires a full graph scan in the semi-streaming model, or $\mathcal{O}(n)$ passes in total. To reduce the number of passes on the graph, one can find a subset of vertices W whose degrees are within the range $[\delta(G), c\delta(G)]$ for any $c \geq 1$. Removing W in a round yields a greedy algorithm that approximates $d(G)$ by a factor of c, since

$$\hat{d}(G) \leq \max\left\{c\delta(G), d(G \setminus W)\right\} \leq cd(G).$$

Although this algorithm finds sets of nodes, rather than single nodes, in each pass, the number of passes remains $\Theta(n)$ in the worst case, as illustrated in Figure 1.

To further reduce the number of passes to log, let the *density* of a G be m/n, and let $d^*(G)$ be the maximum density among all subgraphs of G. It is known that $d^*(G) \leq d(G) \leq 2d^*(G)$ [14]. The first inequality is true by the pigeon-hole principle: if m edges are assigned as out-edges to n nodes, then some node will get at least m/n edges, and this is true of all subgraphs as well. The second inequality is true because every n-node m-edge graph G has a node of degree

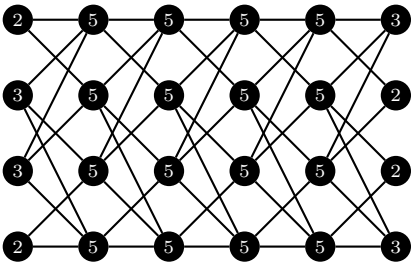

Fig. 1. In the above graph G, if one removes nodes of degree within $[\delta(G), 2\delta(G)]$ in each round, then the removal of all nodes requires three rounds. Each round removes the extreme column at either end, leaving a structure with "internal" nodes of degree 5 and end columns of degree 2/3/3/2. This graph can be extended to an arbitrary length by concatenating multiple instances of G. This longer graph shares the property that only the first and last column are removed in each round. Therefore, greedily removing nodes of degree with $[\delta(G), 2\delta(G)]$ requires $\Theta(n)$ rounds.

no more than $2m/n$; hence, one can always remove a vertex of degree at most $2d^*(G)$, or, equivalently, $d(G) \leq 2d^*(G)$.

If one iteratively removes a subset of vertices whose degrees are no more than cm/n, where $c = 2 + \varepsilon$ for any $\varepsilon > 0$, then the approximated degeneracy $\hat{d}(G)$ is at most $cd^*(G)$, i.e., we achieve a c-approximation of both $d^*(G)$ and $d(G)$. After the removal of vertices, the number of surviving vertices is at most $2n/c$ because the sum of degree is at most $2m$, which means that there are at most $2n/c$ nodes of degree more than cm/n. Thus, the number of passes is logarithmic, specifically, $\mathcal{O}(\log n/(\log c - 1))$, yielding a tradeoff between the approximation factor and the number of passes.

The space can be further reduced based on the ideas of counting sketches used in streaming algorithms [8–10]. That is, we use $s(n) < n$ space to count the degree of each node. Since $s(n)$ is not sufficient to count the degree of n nodes individually, a counter is possibly shared among some nodes. Indeed, the counter is the sum of degrees of nodes assigned to this counter. Therefore, the counter is an overestimate of degree. If the overestimate is bounded, then we get a bounded approximation. We explore these ideas more fully below.

3 Algorithms

We present two algorithms: one that uses n space and achieves a constant approximation factor and one that uses less than n space and yields a smooth tradeoff between the space used and the approximation achieved. These algorithms share some high-level structure, which we present first.

The proposed algorithms require multiple passes on graph $G(V, E)$. Let V_1 be V. In the i^{th} pass, the algorithms identify a subset $V_{i+1} \subseteq V_i$. Let $n_i = |V_i|$ and m_i be the number of edges in the subgraph of G induced by V_i. Step i outputs the nodes in $V_i \setminus V_{i+1}$ in any arbitrary order. The algorithms terminate when

there are no more nodes to output, that is, when $n_{i+1} = 0$. The desired ordering of nodes is the concatenation of the output of all phases.

So far, this algorithm follows the same outline as the other greedy algorithms. The difference will be in which nodes we output in each phase. A node becomes a *candidate* node in phase i if $d_i(u) \leq \alpha m_i/n_i$ for some $\alpha = 2 + \varepsilon$, $\varepsilon > 0$, where $d_i(u)$ is the degree of u in the subgraph of G induced by V_i. Once a node is a candidate, it remains a candidate. Both of our algorithms output only candidate nodes, and they both output candidates as soon as they detect that a node becomes a candidate. The difference between our algorithms is that if we use linear space, we can detect that a node is a candidate as soon as it becomes one, so that we output nodes more aggressively. We achieve a smaller number of phases and a better approximation ratio. If we have sublinear space, it will be more difficult to detect candidacy.

In the first algorithm, we will go further and guarantee to output all nodes u with $d_i(u) \leq \alpha m_i/n_i$, that is, we will detect candidate nodes as soon as they become candidates. In that case,

$$n_{i+1} \leq (2m_i)/(\alpha m_i/n_i) = 2n_i/\alpha.$$

This implies that the algorithm terminates after a logarithmic number of phases. When we use sublinear space, we will not be able to guarantee that all low-degree nodes are output, so we will need to be more careful in order to guarantee a logarithmic number of phases.

Using Space of Size n**.** We start with an easy version such that the memory has size of n^2, which is used for n counters, $d_i(u)$ for $u \in V$, where $d_i(u)$ is reused as $d_{i+1}(u)$ for all i. Each $d_i(u)$ keeps the information whether node u is output before the i^{th} pass, in which case it is set to -1. If the node has not be output yet, $d_i(u)$ is used to keep track of the degree of u in the subgraph of G induced by V_i. The value of $d_i(u)$ for all u can be updated in one pass through the graph, as follows. If $d_i(u)$ is negative, do nothing; otherwise, reset $d_i(u)$ to 0. During the scan, process the edge, one by one. When processing edge (u, v), if $d_i(u) \geq 0$ and $d_i(v) \geq 0$, then it means edge (u, v) is still the subgraph induced by V_i; increment $d_i(u)$ and $d_i(v)$ by one.

Having $d_i(u)$ for $u \in V_i$, check each u to see if $d_i(u) \geq \alpha m_i/n_i$. Output any such node u as it is identified, and set $d_i(u) = -1$.

Lemma 1. *An $\alpha d(G)$-degenerate ordering of nodes, for $\alpha = 2+\varepsilon, \varepsilon > 0$, can be generated by a semi-streaming algorithm using n space, $\mathcal{P} = \mathcal{O}(\log n/\log(\alpha/2))$ passes, and $\mathcal{O}((m+n)\mathcal{P})$ time.*

Proof. Consider an ordering produced by the algorithm. If node u is output in the i^{th} pass, then u belongs to $V_i \setminus V_{i+1}$ and has outdegree at most $\alpha m_i/n_i$. The n counters allows the algorithm to output any such candidate node during

[2] Any algorithm will require a constant amount of memory, so when we report space usage of n, we mean in addition to the $O(1)$ overhead for running the algorithm.

the phase that it becomes a candidate. Therefore, $n_{i+1} \leq 2n_i/\alpha$. The number of passes is therefore $\mathcal{O}(\log n/\log(\alpha/2))$.

As for the approximation factor, since each output node u in the i^{th} pass has degree $d_i(u)$, then the eventual outdegree of u is at most

$$d_i(u) \leq \alpha m_i/n_i \leq \alpha d^*(G) \leq \alpha d(G),$$

yielding an $\alpha d(G)$-degenerate ordering of nodes. □

Using Space of Size $s(n) \leq n$**.** The small-space algorithm proceeds in two sections. At first, there is not enough space to count the degree of every one. As some nodes are output, the number of remaining nodes n_i drops. When n_i drops to no more than $s(n)/2$, then we can use a hash table to count the degrees and proceed as that using space of size n. Therefore, we focus on the first part of the algorithm.

If $n_i > s(n)/2$, then the $s(n)$ space is not sufficient to keep track of which nodes are still active and to keep track of $d_i(u)$ individually for each $u \in V_i$. Therefore, the algorithm switches over when $n_i \leq s(n)/2$. To count degrees with fewer counter, as with counting sketches [8–10], we map n_i nodes to $s(n)$ space by a hash function h, and $\hat{d}_i(h(u))$ is used to count the degree of node u, although other nodes v may share the same counter if $h(v) = h(u)$. Formally,

$$\hat{d}_i(h(u)) = \sum_{v \in V_i, h(v)=h(u)} d_i(v).$$

Note that by $d_i(u)$ and $\hat{d}_i(h(u))$ we denote the real degree of node u in the subgraph induced by V_i and its estimate, respectively. Clearly, $\hat{d}_i(h(u)) \geq d_i(u)$. We use a simple, deterministic hash function,

$$h(u) = u \bmod s(n).$$

The small-space procedure mimics the n-counter algorithm. First, recall that when a counter is negative, it means that a node has been output. Since several nodes can share the same counter, if we output one node u that maps to a counter, we will output all nodes that map to that counter, and we will set that counter $\hat{d}(h(u)) = -1$, as before.

Lemma 2. *An $\alpha d(G)$-degenerate ordering of nodes, for $\alpha = (2+\varepsilon)n/s(n), \varepsilon > 0$, can be generated by a semi-streaming algorithm using $s(n)$ space, $\mathcal{P} = \mathcal{O}(\log n/\log(\alpha s(n)/(2n)))$ passes, and $\mathcal{O}((m+s(n))\mathcal{P})$ time.*

Proof. Recall that $n_1(=n), n_2, \ldots, n_{\mathcal{P}+1}$ is the number of nodes remaining after each pass. In the i^{th} pass, the expectation of estimators $\hat{d}_i(x)$ for $x \in \{1, 2, \ldots, s(n)\}$ is $\mathrm{E}[\hat{d}_i(x)] = 2m_i/s(n)$. By Markov's inequality, we have

$$p_1 \equiv \Pr\left[\hat{d}_1(x) > \frac{\alpha s(n)}{2n_1}\mathrm{E}[\hat{d}_1(x)]\right] < \frac{2n_1}{\alpha s(n)} < 1.$$

The number of estimators that have value more than $(\alpha s(n))/(2n_1)\mathrm{E}[\hat{d}_1(x)]$ is at most

$$s_2 < s(n)\frac{2n_1}{\alpha s(n)} < s(n).$$

Then V_2, the identified subset of V_1, has cardinality at most $n_2 = s_2\lceil n/s(n)\rceil$ or precisely $s_2 n/s(n)$, assuming that n is a multiple of $s(n)$. The assumption can be handled by adding some isolated nodes to increase n to a multiple of $s(n)$ and ignoring the nodes of index more than the original n as they are output. Similarly, due to $n_2 < n_1 = n$,

$$p_2 \equiv \Pr\left[\hat{d}_2(x) > \frac{\alpha s(n)}{2n_2}\mathrm{E}[\hat{d}_2(x)]\right] < \frac{2n_2}{\alpha s(n)} < 1.$$

The number of estimators that have value more than $(\alpha s(n))/(2n_2)\mathrm{E}[\hat{d}_2(x)]$ is

$$s_3 = s(n)p_2 < s(n)\frac{2}{\alpha s(n)}n_2 = s(n)\frac{2}{\alpha s(n)}s_2\frac{n}{s(n)} < s_2.$$

Therefore, $\mathcal{P} = \mathcal{O}(\log n/\log(\alpha s(n)/(2n)))$ because the above derivation is

$$s_{i+1} < \frac{2n}{\alpha s(n)}s_i < s_i$$

in general. The outdegree of nodes output in the i^{th} pass is bounded by

$$\hat{d}_i(h(u)) \le \frac{\alpha s(n)}{2n_i}\mathrm{E}[\hat{d}_i(x)] \le \frac{\alpha s(n)}{2n_i}\frac{2m_i}{s(n)} \le \alpha d^*(G) \le \alpha d(G). \qquad \square$$

We compare the proposed algorithms in Table 1. The approximation factor α and the required number of passes $\mathcal{P}$ of the proposed sublinear-space algorithm are those of the linear-space algorithm multiplied by a factor of $n/s(n)$, respectively. There is therefore a tradeoff between α and $\mathcal{P}$ on the one hand and $s(n)$ on the other. We can combine the two algorithms as follows. When the space is not sufficient to accommodate all nodes, we use the sublinear-space algorithm. Once the remaining nodes are at most $s(n)/2$, we use the linear-space algorithm, which converges faster, changing the base of the log from $\alpha s(n)/(2n)$ (a constant close to 1) to $\alpha/2$ (a constant if $s(n) = \Theta(n)$ or logarithmic if $s(n) = \Theta(n)(n/\text{polylog})$). Hence, the required number of passes is bounded by

$$\begin{aligned}&\mathcal{O}(\log_{\alpha s(n)/(2n)} n/s(n) + \log_{\alpha/2} s(n))\\ =&\mathcal{O}(\log(n/s(n))/\log(1+\varepsilon/2) + \log s(n)/\log((1+\varepsilon/2)n/s(n))).\end{aligned}$$

It is remarkable that, for small ε, only $\mathcal{O}(\log\log n/\varepsilon + \log n/\log\log n)$ passes are needed to achieve an approximation factor of $\mathcal{O}(\log^k n)$ when $s(n) = n/\log^k n$.

4 Applications

Here we show how to use a small degeneracy ordering to approximate some problems in streaming models.

Table 1. Comparison of proposed algorithms. Approximations are for small ε.

space	approximation factor (α)	# of passes ($\mathcal{P}$) for any ε	$\mathcal{P}$ for small ε
n	$2+\varepsilon$	$\log n/\log((2+\varepsilon)/2)$	$\approx 2\log n/\varepsilon$
$s(n)$	$(2+\varepsilon)n/s(n)$	$\log n/(\log(\alpha s(n)/(2n)))$	$\approx 2\log n/\varepsilon$

Forest/planar subgraph decomposition. Let $a(G)$, the *arboricity* of G, be the minimum number of disjoint forests into which graph G can be decomposed. Let $\theta(G)$, the *thickness* of G, be the minimum of disjoint planar graph into which graph G can be decomposed. The relationships among degeneracy, arboricity and thickness are illustrated in Figure 2.

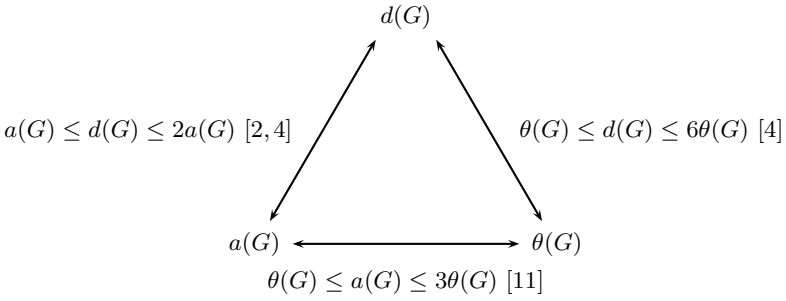

Fig. 2. The relationships among $d(G), a(G)$ and $\theta(G)$

One can use a $(2+\varepsilon)d(G)$-degenerate ordering to approximate the optimal decomposition of a graph into disjoint forests/planar subgraphs, as follows. For each u, assign a distinct color to each edge connecting a neighbor that appears later in the ordering. Since the ordering is $(2+\varepsilon)d(G)$-degenerate, $(4+\varepsilon)d(G)$ are required, i.e., a 4-approximation. Monochrome edges never form a cycle, that is, they form a forest. Since a forest is also a planar graph, forest-decomposition can also be applied to planar-decomposition, yielding a $(12+\varepsilon)$-approximation.

Given a small-degeneracy ordering, subgraph decomposition can therefore be approximated in the Sort-stream model, using the space $s(n)$, as follows. Associate with each edge the rank of its incident nodes in the ordering. This can be done in $n/s(n)$ sequential writable passes. Then, sort the stream by the smaller rank of incident nodes in one pass, as is assumed to be allowable in the Sort-stream model. Perform one more sequential scan to assign a distinct color to each edge that match in the smaller rank of incident nodes. Last, sort the stream by the edge color.

Sever-Client Load Balancing. For this application, we need to generalize the result on ordinary graph G to a simple r-hypergraph $\mathcal{G}$. In a hyergraph, each hyperedge is a subset of nodes. An r-hypergraph is a hypergraph where each edge has at least two nodes (no self loops) and at most r. In this case, $d^*(G) \leq d(G) \leq rd^*(G)$ because, on one hand, n nodes are assigned by m

edges and there exists one node has been assigned no fewer than m/n edges by the Pigeon-hole principle; on the other hand, there exists one node with degree no more than rm/n in each r-hypergraph, which, by the optimal greedy construction of degeneracy, shows the right-hand inequality.

Corollary 1. *Given a simple undirected n-node m-edge r-hypergraph $\mathcal{G}$, using space $s(n)$, where $1 \leq s(n) \leq n$, an $\alpha d(\mathcal{G})$-degenerate ordering of nodes of $\mathcal{G}$, i.e. an α-approximation, can be computed in $\mathcal{O}((m+n)\mathcal{P})$ time, using $s(n)$ space and $\mathcal{P} = \mathcal{O}(\log_{1+\varepsilon/r}(n/s(n)) + \log_{\alpha/r} s(n))$ sequential passes on the entire graph, where $\alpha = (r+\varepsilon)n/s(n)$ for any $\varepsilon > 0$.*

Consider the server-client load balancing problem [20]. A sever-client model is a bipartite graph such that all edges are incident on one server node and one client node. The task is to assign each client to a server while minimizing the number clients assigned to a single server.

Suppose that the degree of each client is at most r, and consider the following induced r-hypergraph: the nodes are the server nodes and each hyperedge in the induced hypergraph is the set of servers adjacent to a client node. Then an $(r+\varepsilon)d(\mathcal{G})$-degenerate ordering of the hypergraph is a $(r+\varepsilon)$-approximation of the server-client load balancing problem, where the approximated assignment of each client to a server is then obtained by the degenerate ordering.

An $(r+\varepsilon)d(G)$-degenerate ordering induces an assignment of each client to a server. As mentioned above, the induced assignment minimizes the heaviest server load to within a factor of $(r+\varepsilon)$ of optimal. In addition, based on the algorithm introduced in [20], a good approximation algorithm can be used to speedup their exact computation. In [20], they prove that, given a sub-optimal assignment, there exists an alternating path connecting the server of maximum load to another server. The better the initial solution, the fewer alternating-path rewrites that are needed, thus improving the time complexity of finding an optimal solution.

Finding an Approximated Dominating Set. In [19], it was shown how to identify a dominating set based on a d-degenerate ordering, for any d. Let the neighbor nodes appear earlier (later) in the ordering be earlier (later) neighbors. The algorithm initializes an empty dominating set $\hat{D}$ and adds some nodes to this set greedily as follows. Traverse the nodes in the ordering from the first to the last. When processing node u, if u is dominated by $\hat{D}$, do nothing; otherwise, some nodes need to be added to $\hat{D}$. The choice of added nodes is made depending on whether u has a later neighbor. If so, add all later neighbors to $\hat{D}$; otherwise, add u to $\hat{D}$. The set $\hat{D}$ so constructed is a dominating set. The greedy process requires the space $\mathcal{O}(n)$ and one sequential pass on the adjacency list, each list sorted by lower rank, can be handled as in the graph decomposition.

In [19], it is shown that $|\hat{D}| \leq \mathcal{O}(d^2)|D|$, where D is the smallest dominating set. More applications that apply small degeneracy ordering to some variations of dominating set, e.g. k-dominating set, are introduced in [12].

References

1. Ahn, K.J., Guha, S.: Linear programming in the semi-streaming model with application to the maximum matching problem. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part II. LNCS, vol. 6756, pp. 526–538. Springer, Heidelberg (2011)
2. Alon, N., Yuster, R., Zwick, U.: Finding and counting given length cycles. Algorithmica 17(3), 209–223 (1997)
3. Bar-Yossef, Z., Kumar, R., Sivakumar, D.: Reductions in streaming algorithms, with an application to counting triangles in graphs. In: Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2002, pp. 623–632. Society for Industrial and Applied Mathematics (2002)
4. Bollobás, B.: Extremal graph theory. Academic Press (1978)
5. Bollobás, B.: The evolution of sparse graphs. In: Graph Theory and Combinatorics, Proc. Cambridge Combinatorial Conf., pp. 35–57. Academic Press (1984)
6. Chakrabarti, A., Cormode, G., McGregor, A.: Annotations in data streams. In: Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.) ICALP 2009, Part I. LNCS, vol. 5555, pp. 222–234. Springer, Heidelberg (2009)
7. Charikar, M.: Greedy approximation algorithms for finding dense components in a graph. In: Jansen, K., Khuller, S. (eds.) APPROX 2000. LNCS, vol. 1913, pp. 84–95. Springer, Heidelberg (2000)
8. Charikar, M., Chen, K., Farach-Colton, M.: Finding frequent items in data streams. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 693–703. Springer, Heidelberg (2002)
9. Cormode, G., Hadjieleftheriou, M.: Finding frequent items in data streams. Proc. VLDB Endow. 1(2), 1530–1541 (2008)
10. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. J. Algorithms 55(1), 58–75 (2005)
11. Dean, A.M., Hutchinson, J.P., Scheinerman, E.R.: On the thickness and arboricity of a graph. Journal of Combinatorial Theory, Series B 52(1), 147–151 (1991)
12. Dvořák, Z.: Constant-factor approximation of the domination number in sparse graphs. European Journal of Combinatorics 34(5), 833–840 (2013)
13. Feigenbaum, J., Kannan, S., McGregor, A., Suri, S., Zhang, J.: On graph problems in a semi-streaming model. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 531–543. Springer, Heidelberg (2004)
14. Frank, A., Gyarfas, A.: How to orient the edges of a graph. In: Combinatorics Volume I, Proc. of the Fifth Hungarian Colloquium on Combinatorics, vol. I, pp. 353–364 (1976)
15. Gabow, H., Westermann, H.: Forests, frames, and games: algorithms for matroid sums and applications. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, STOC 1988 pp. 407–421. ACM (1988)
16. Goldberg, A.V.: Finding a maximum density subgraph. Tech. rep. (1984)
17. Kawano, S., Yamazaki, K.: Worst case analysis of a greedy algorithm for graph thickness. Information Processing Letters 85(6), 333–337 (2003)
18. Kowalik, Ł.: Approximation scheme for lowest outdegree orientation and graph density measures. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 557–566. Springer, Heidelberg (2006)
19. Lenzen, C., Wattenhofer, R.: Minimum dominating set approximation in graphs of bounded arboricity. In: Lynch, N.A., Shvartsman, A.A. (eds.) DISC 2010. LNCS, vol. 6343, pp. 510–524. Springer, Heidelberg (2010)

20. Liu, P., Wang, D.W., Wu, J.J.: Efficient parallel i/o scheduling in the presence of data duplication. In: International Conference on Parallel Processing, pp. 231–238 (2003)
21. Mansfield, A.: Determining the thickness of graphs is NP-hard. Math. Proc. Cambridge Philos. Soc. 93, 9–23 (1983)
22. Matula, D.W., Beck, L.L.: Smallest-last ordering and clustering and graph coloring algorithms. J. ACM 30(3), 417–427 (1983)
23. Muthukrishnan, S.: Data streams: Algorithms and applications. Tech. rep. (2003)
24. O'Connell, T.C.: A survey of graph algorithms under extended streaming models of computation. In: Fundamental Problems in Computing, pp. 455–476. Springer, Netherlands (2009)
25. Ruhl, J.M.: Efficient Algorithms for New Computational Models. Ph.D. thesis, Massachusetts Institute of Technology (September 2003)
26. Schank, T., Wagner, D.: Finding, counting and listing all triangles in large graphs, an experimental study. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 606–609. Springer, Heidelberg (2005)
27. Zhang, J.: A survey on streaming algorithms for massive graphs. In: Managing and Mining Graph Data, Advances in Database Systems, vol. 40, pp. 393–420. Springer, US (2010)