Efficient Primal-Dual Algorithms for MapReduce

Bahman Bahmani*, Ashish Goel**, and Kamesh Munagala***

Abstract. In this paper, we obtain improved algorithms for two graph-theoretic problems in the popular MapReduce framework. The first problem we consider is the densest subgraph problem. We present a primal-dual algorithm that provides a $(1 + \epsilon)$ approximation and takes $O\left(\frac{\log n}{\epsilon^2}\right)$ MapReduce iterations, each iteration having a shuffle size of $O(m)$ and a reduce-key-complexity of $O(d_{\text{max}})$. Here $m$ is the number of edges, $n$ is the number of vertices, and $d_{\text{max}}$ is the maximum degree of a node. This dominates the previous best MapReduce algorithm, which provided a $(2 + \delta)$-approximation in $O\left(\frac{\log n}{\delta^2}\right)$ iterations, with each iteration having a total shuffle size of $O(m)$ and a reduce-key-complexity of $O(d_{\text{max}})$.

The standard primal-dual technique for solving the above problem results in $O(n)$ iterations. Our key idea is to carefully control the width of the underlying polytope so that the number of iterations becomes small, but an approximate primal solution can still be recovered from the approximate dual solution. We then show an application of the same technique to the fractional maximum matching problem in bipartite graphs. Our results also map naturally to the PRAM model.

1 Introduction

The programming paradigm MapReduce [6] and its main open-source implementation, Hadoop [10], have had an enormous impact on large scale data processing. Over the last two decades, the primal-dual method (eg. [14,9,2,17]) has been used to solve many graph optimization problems. In this paper, our goal is to adapt the primal-dual technique for solving graph optimization problems to the MapReduce framework. We consider the densest subgraph problem as well as the fractional maximum bipartite matching problem, both of which have the following structure: These problems can be written as linear programs; furthermore, either the linear program or its dual can be interpreted as solving...
maximum or concurrently maximum flows on suitably defined bipartite graphs. We present a simple algorithmic technique for solving these optimization problems to arbitrarily good approximations, that yields fast sequential as well as MapReduce algorithms, thus improving the state of the art. Along the way, in Appendix A we also present complexity measures for MapReduce computations that is somewhat different from those extant in literature (for instance [12]). Our measures abstract away the number of processors, and instead focus on the complexity associated with a single map/reduce key as well as with the data in a single MapReduce phase. These measures have the advantage of being detailed enough to make fine-grained distinction between algorithms, while abstracting away implementation details.

1.1 Problem Formulations and Results

We will focus on two graph optimization problems in this paper (although our technique is more widely applicable). The first is the classic densest subgraph problem in general graphs. This can be defined either for undirected or directed graphs. For an undirected graph \( G(V,E) \) with \( n \) vertices and \( m \) edges, the problem is defined as follows. Find a subset \( H \) of vertices such that the induced subgraph \( G'(H,E') \) has maximum density (or average degree), \( |E'|/|H| \). For a directed graph \( G(V,E) \), the densest subgraph problem [11] asks to find two subsets \( S,T \subseteq V \) (not necessarily disjoint) such that if \( E(S,T) \) denotes the set of edges \( (u,v) \) where \( u \in S \) and \( v \in T \), then the density \( D(S,T) = |E(S,T)|/\sqrt{|S||T|} \) is maximized. Such subgraphs have meaning for various kinds of social or authorship networks - these can be used to identify dense communities of nodes, or dense “cliques” of authors. We refer the reader to [14] for several applications.

The other problem we consider is the maximum generalized fractional matchings on bipartite graphs. This problem has several applications in Adword allocation and load balancing [13]. Specifically, we consider the following allocation problem [13]: There are \( n_1 \) advertisers, where advertiser \( i \) has demand \( d_i \); there are \( n_2 \) ad slots, where slot \( j \) has supply \( s_j \); furthermore there is a bipartite graph specifying which advertiser is interested in which ad slots. The goal is to find an allocation of slots to advertisers that maximizes the demand satisfied; if the supply and demands are large, this can be thought of as a fractional allocation without loss of generality.

Our Results. We express our results using complexity measures described in [8] and summarized in appendix A. While we are not describing the full model here, it is important to note that the model is quite detailed and takes into account practical aspects of MapReduce, such as shuffle costs, the fact that phases are very expensive to set up, and the “curse of the last reducer” [16].

In this paper, we develop a simple and general algorithmic technique that achieves a \((1 + \epsilon)\) approximation to the above problems (for any \( \epsilon > 0 \)), with sequential running times of the form \( O(mf(\log n, 1/\epsilon)) \) where \( f \) is a polynomial. Here, \( m \) is the number of edges, and \( n \) the number of vertices in the input graph. We will be interested in making the function \( f \) as small as possible. We
will further design MapReduce implementations that minimize the number of phases subject to two constraints: (1) The key-complexity (i.e. the maximum size/computation/memory associated with any map or reduce key) is $O(d_{\text{max}})$, the maximum degree, and (2) The sequential complexity (the total volume of data transferred, or the total time spent) in any phase is $\tilde{O}(m)$, where the $\tilde{O}$ notation hides low order terms ($\frac{\log n}{\epsilon}$ to be precise). Our results are summarized as follows, where the running time refers to sequential running times.

- **Undirected densest subgraph:** We present an algorithm that runs in $O\left( m \frac{\log n}{\epsilon^2} \right)$ time, and takes $O\left( \frac{\log n}{\epsilon^2} \right)$ MapReduce phases (Section 2). The total running time and shuffle size in any one phase is $O(m)$. The best previous MapReduce implementation is a greedy algorithm in [4]; this algorithm takes $O\left( \frac{\log n}{\epsilon} \right)$ phases to yield a $2 + \delta$ approximation.

- **Directed densest subgraph:** We combine the above approach with a linear programming formulation due to Charikar [5] to obtain $\tilde{O}\left( m \frac{\log^2 n}{\epsilon^3} \right)$ running time, and $O\left( \frac{\log n}{\epsilon^2} \right)$ MapReduce phases (Appendix C). Again, the best previous best MapReduce algorithm [4] was a greedy $(2 + \delta)$ approximation.

- **Fractional matching on bipartite graphs:** We show that exactly the same technique yields $\tilde{O}\left( m \frac{\log^2 n}{\epsilon^3} \right)$ running time, and $O\left( \frac{\log n}{\epsilon^2 \log d_{\text{max}}} \right)$ MapReduce phases (Appendix D). This matches (and in fact, improves by a log $d_{\text{max}}$ factor) that of the natural MapReduce implementation of the semi-streaming algorithm in [1]. Furthermore our sequential running time is comparable to the best sequential algorithms in [12] (which we do not know how to implement on MapReduce). While the improvement in running time is small, note that the previous algorithms were tailored to the bipartite matching problem, whereas our algorithm naturally follows from the more general framework we describe next.

While we have stated our results in the MapReduce computation framework, they also map to (and are novel in) the PRAM framework: the key-complexity corresponds to the parallel running time per phase (given $n$ processors) and the sequential running time corresponds to the total amount of work done.

### 1.2 Technique: Width Modulation

We design our algorithms by exploiting connections to fast algorithms for packing and covering linear programs. We illustrate this for the densest subgraph

---

1. Superficially, the number of phases in [4] might seem smaller. However, note that even $\delta = 0$ for their work corresponds to $\epsilon = 1/2$ for us.

2. Though Ahn and Guha [1] present several improvements to their basic algorithm, these are in the semi-streaming model and do not appear to produce corresponding improvements in the MapReduce model.
problem. We start with a linear programming relaxation of the densest subgraph problem (due to Charikar [5]) and show that it is the dual of a maximum concurrent multi-commodity flow (MCMF) problem on a suitably defined bipartite graph (where the goal is to simultaneously route demand from all sources to a sink). We then proceed to exploit fast approximation algorithms for the MCMF problem due to Plotkin-Shmoys-Tardos [14] and Young [17], which are unified in a framework due to Arora-Hazan-Kale [2]. These algorithms suffer from two significant roadblocks.

First, the number of parallel iterations of these algorithms depends on the width of the problem, which in our case is proportional to the maximum degree \(d_{\text{max}}\). This dependence arises because the MCMF formulation attempts to route demands of size 1 concurrently in a graph with capacities either infinity or \(d_{\text{max}}\). The algorithms are based on the Lagrangian approach, which converts the MCMF problem to a maximum multi-commodity flow problem ignoring the demands, which can make the Lagrangian route demand that can be a factor \(d_{\text{max}}\) larger than the original demands. We overcome this hurdle by a technique that we term width modulation, whereby we add spurious capacity constraints to make capacities a small constant factor larger than the maximum demand, which is 1.

Though the above method reduces width to a constant and does not affect feasibility, this introduces the second roadblock: Adding capacities to the dual program changes the primal problem. Though this is not an issue for optimal solutions (after all, we did not affect feasibility), an approximately optimal primal solution for the changed problem need not yield an approximately optimal solution to the problem we intended to solve. We need the modulation part to overcome this hurdle – we show that for width being sufficiently large, we can indeed recover an approximately optimal primal solution via developing a problem-specific efficient rounding scheme. In a sense, there is a trade-off between running time (small width helps) and ability of the algorithm to recover the solution to the original problem (large width helps). We show that this trade-off leads to a final solution with constant width, which yields an \(O\left(\frac{\log n}{\epsilon^2}\right)\) phase \textsc{MapReduce} implementation and an \(O\left(m\frac{\log n}{\epsilon^2}\right)\) overall running time.

For directed graphs, we use a parametric linear programming formulation due to Charikar [5]. To adapt this to our framework, we devise a parametric search procedure. We finally show that the same technique applies with small changes to the fractional bipartite matching problem, showing its generality.

In summary, we show that several linear programs which have wide practical applications can be efficiently approximated on \textsc{MapReduce} by considering their dual formulation. We show that there is a tension between reducing the width of the dual problem (which is needed for efficiency) and recovering the primal solution itself, and present a general technique based on width modulation and rounding to achieve this trade-off without sacrificing efficiency or precision.
1.3 Related Work

In addition to presenting the LP formulations that we use, Charikar [5] also presents a greedy 2-approximation for the densest subgraph problem (see also [15]). This algorithm is modified to yield an efficient MAPREDUCE implementation in [4]; this algorithm takes $O\left(\frac{\log n}{\epsilon}\right)$ rounds to yield a $2 + \epsilon$ approximation.

There is a long line of work on fast approximate algorithms for covering linear programs; see [3] for a survey. One closely related line of work are the algorithms for spreading metrics due to Garg-Konemann [7] and their parallel implementation due to Awerbuch et al. [3]. These algorithms can possibly be applied to our dual formulations; however, when implemented in MAPREDUCE these methods will need $\text{poly}(\log n, 1/\epsilon)$ phases for a large degree polynomial, which is a far worse running time than what we show. For example, the algorithm of Awerbuch et al., when applied to our setting, would result in $O\left(\frac{\log^6 n}{\epsilon^4}\right)$ phases. Furthermore, note that the techniques in [7, 3] can also be viewed as width reduction, where the width is reduced by adding constraints that the flow along a path is at most the minimum capacity on that path (see [2] for details). However, our width modulation technique is fundamentally different - we modulate the capacities themselves based on the demand being routed. In contrast with the technique in [7, 3], our technique changes the description of the primal problem and does not preserve approximate optimality. Hence we need a problem-specific rounding scheme to recover the primal solution.

Roadmap. For lack of space, we relegate the description of complexity measures for MAPREDUCE to Appendix A and the description of the multiplicative weight method to Appendix B. We present the FPTAS and MAPREDUCE implementation for undirected densest subgraph in Section 2 that for directed densest subgraph in Appendix C and for fractional bipartite matchings in Appendix D.

2 Undirected Densest Subgraph

For an undirected graph $G(V, E)$ with $n$ vertices and $m$ edges, the DENSEST SUBGRAPH problem is defined as follows. Find a subset $H \subseteq V$ of vertices such that the induced subgraph $G'(H, F)$ has maximum density (or average degree), $|F|/|H|$. We denote the optimum density by $D^\ast$. We present an algorithm that for any $\epsilon > 0$, outputs a subgraph of density $D^\ast(1 - \epsilon)$ in $\tilde{O}\left(\frac{m \log n}{\epsilon^2}\right)$ running time, where the $\tilde{O}()$ notation ignores lower order terms. In the MAPREDUCE model, we show that with key complexity $O(d_{\text{max}})$, where $d_{\text{max}}$ is the maximum degree, and sequential complexity $O(m)$, the algorithm requires $O\left(\frac{\log n}{\epsilon^2}\right)$ phases.

2.1 Linear Program and Duality

Let $D^\ast$ denote the optimal density. We will first present a well-known linear program which is known to compute the optimal solution [5]. For any vertex
\(v \in V\), let \(x_v \in \{0, 1\}\) denote whether \(v \in H\). For any edge \(e \in E\), let \(y_e \in \{0, 1\}\) denote whether \(e \in F\), which are the edges induced by \(H\). We relax \(x_v, y_e\) to be any real number. The value \(D^*\) is the solution to the following linear program:

\[
\text{Maximize } \sum_e y_e
\]

\[
y_e \leq x_v \quad \forall e \in E, e \text{ incident on } v
\]

\[
\sum_v x_v \leq 1
\]

\[
x_v, y_e \geq 0 \quad \forall v \in V, e \in E
\]

To interpret the above program, note that if \(y_e = 1\) for \(e = (u, v)\), then both \(x_u\) and \(x_v\) have to be 1. This implies the first constraint. The objective should maximize \((\sum_e y_e)/(\sum_v x_v)\). We can scale the values so that \(\sum_v x_v = 1\), and enforce this as the second constraint. This means the objective now maximizes \(\sum_e y_e\). Therefore, the value of the above LP is at least \(D^*\). (It is in fact known that it is exactly \(D^*\), but we will not need that fact.) For simplicity, we overload notation and denote the optimal value of the LP by \(D^*\).

We now take the dual of the above program. Let \(\alpha_{e v}\) denote the dual variable associated with the first constraint, and let \(D\) denote the dual variable associated with the second constraint. We parametrize the dual constraints by the variable \(D\), and call these set of constraints \(\text{Dual}(D)\):

\[
\alpha_{e u} + \alpha_{e v} \geq 1 \quad \forall e = (u, v) \in E
\]

\[
\sum_e \alpha_{e v} \leq D \quad \forall v \in V
\]

\[
\alpha_{e v} \geq 0 \quad \forall e, v
\]

Since the dual program is minimizing \(D\), using strong duality, we have:

**Lemma 1.** \(\text{Dual}(D)\) is feasible iff \(D \geq D^*\).

We note that the dual program is a maximum concurrent multi commodity flow (MCMF) problem: Construct a bipartite directed graph \(G'(U', V', E')\) as follows: \(U' = E, V' = V\), and \(E' = \{(e, v) \in E \times V | e \text{ is incident on } v\}\). Each node in \(U'\) has demand 1, and the nodes in \(V'\) are connected to a sink with directed edges of capacity \(D\). The first constraint means that each demand of 1 is completely routed; the second constraint means that for all directed edges to the sink of capacity \(D\), the flow routed is at most the capacity. Therefore, the goal is decide if all demand can be concurrently routed to the sink while satisfying the capacity constraints.

### 2.2 Width Modulation

We will apply the multiplicative weight update framework as described in Appendix B to decide feasibility of \(\text{Dual}(D)\) for given \(D\). In particular, we will decide the feasibility of the set of constraints:

\[
\alpha_{e u} + \alpha_{e v} \geq 1 \quad \forall e = (u, v) \in E
\]
subject to the polyhedral constraints $P(D)$ (which depends on parameter $D$):

$$\sum_e \text{ incident on } v \alpha_{ev} \leq D \quad \forall v \in V$$

$$\alpha_{ev} \geq 0 \quad \forall e \in E, v \in V$$

The dual vector corresponding to the constraint $\alpha_{eu} + \alpha_{ev} \geq 1$ is $y_e$, whose dimension is $m$. The main issue with a naive application of the method is the width. Each $\alpha_{ev}$ can be as large as $D$, so that the LHS of the above constraint can be as large as $2D$, which is also the width. Since $D$ is the density and can be as large as $n$, this implies a polynomial number of MapReduce rounds, and a correspondingly large sequential running time. Our goal will now be to reduce the width to a constant.

In order to achieve this, consider the following modified polyhedral constraints, that we term $P(D, q)$. Here, $q \geq 1$ will be a small integer. We will denote the corresponding feasibility problem as $\text{Dual}(D, q)$.

$$\sum_e \text{ incident on } v \alpha_{ev} \leq D \quad \forall v \in V$$

$$\alpha_{ev} \leq q \quad \forall e \in E, v \in V$$

$$\alpha_{ev} \geq 0 \quad \forall e \in E, v \in V$$

Note that the second constraint in $P(D, q)$ is new, and it does not change the feasibility of $\text{Dual}(D)$, since if the original system is feasible, it is also feasible with $\alpha_{ev} \leq 1$ for all $e, v$. In other words, $\text{Dual}(D)$ is feasible iff $\text{Dual}(D, 1)$ is feasible. We will set $q$ to be a small constant that we decide later.

**Lemma 2.** The width $\rho$ of $\text{Dual}(D, q)$ as written above is at most $2q$.

**Proof.** For any $\alpha \in P(D, q)$, we have $\alpha_{eu} \leq q$. Therefore $\alpha_{eu} + \alpha_{ev} \leq 2q$, which implies a width of $2q$.

In order to apply the multiplicative weight update method as described in Appendix B, we need to compute $\text{Oracle}(y)$. This involves solving the following problem for given $y$:

$$C(y, D, q) = \max_{\alpha \in P(D, q)} \sum_v \sum_{e \text{ incident on } v} y_e \alpha_{ev}$$

**Lemma 3.** $\text{Oracle}(y)$ can be computed in $O(m)$ time.

**Proof.** For any $y$, for each $v$, the optimal solution $C(y, D, q)$ sets $\alpha_{ev}$ as follows. Let $r = \lfloor D/q \rfloor$ and let $s = D - rq$. Then $\alpha_{ev} = q$ for the $r$ largest $y_e$ incident on $v$, and $\alpha_{ev} = s$ for the $e$ with the $(r + 1)^{st}$ largest $y_e$. This involves finding the $r^{th}$ and $(r + 1)^{st}$ largest $y_e$ for each vertex, which can be done in linear time. This is followed by a linear time computation to set the values.

Using the above two lemmas, the following theorem is immediate from Theorem 5.

**Theorem 1.** For any integer $D$ and constants $q$ and $\epsilon \in [0, 1]$, in time $O\left(\frac{m \log m}{\epsilon^2}\right)$, the multiplicative weight algorithm either returns that $\text{Dual}(D, q)$ is infeasible, or finds $\alpha$ so that for all $e = (u, v) \in E$: $\alpha_{eu} + \alpha_{ev} \geq 1 - \epsilon$. 

7
2.3 Binary Search for $D^*$

We showed earlier how we can apply the multiplicative weight update algorithm to decide if $\text{Dual}(D, q)$ is feasible. Let $k(\epsilon)$ be the number of phases needed to compute a $(1+\epsilon)$ approximation to $D^*$, and $k'(\epsilon)$ the number of phases needed to compute a $(1+\epsilon)$ approximation given a $(1+2\epsilon)$ approximation, where $\epsilon < 1/2$.

From Theorem 5, it follows that $k'(\epsilon) = O\left(\frac{\log n}{\epsilon^2}\right)$. Since $k(\epsilon) \leq k'(\epsilon) + k(2\epsilon)$, this gives $k(\epsilon) = O\left(\frac{\log n}{\epsilon^2}\right)$. Since $\epsilon$ decreased by a factor of 2 in each recursive step above, the above recurrence can be thought of as a binary search.

Let $(\alpha, y)$ denote the final solution corresponding to running the multiplicative weight procedure on $\text{Dual}(\hat{D}, q)$. Here the approximately optimal dual solution $y$ is found as in Theorem 5.

**Theorem 2.** For $0 \leq \epsilon \leq 1/3$ and any constant $q \geq 1$, the value $\hat{D}$ and the final solution $(\alpha, y)$ satisfy:

1. $D^*(1-\epsilon) \leq \hat{D} \leq D^*(1+\epsilon)$.
2. $\sum_c y_c \geq (1-3\epsilon)C(y, \hat{D}, q)$.

**Proof.** Suppose $\hat{D} < D^*(1-\epsilon)$. Since the multiplicative weight procedure returns an $\epsilon$-optimal solution, we can scale up $\alpha$ by $1/(1-\epsilon)$ so that these values are feasible for $\text{Dual}(D, 2q)$ for $D = \hat{D}/(1-\epsilon) < D^*$, and hence feasible for $\text{Dual}(D)$. This violates the optimality of $D^*$ as the smallest $D$ for which $\text{Dual}(D)$ is feasible. On the other hand, $\text{Dual}(D, q)$ is feasible for any $D \in [D^*, D^*(1+\epsilon)]$, which means the multiplicative weight algorithm cannot declare in-feasibility for any $D$ falling within this range. Therefore $D^*(1-\epsilon) \leq \hat{D} \leq D^*(1+\epsilon)$.

Recall from the discussion preceding Theorem 5 that

$$\lambda^* = \max \{ \lambda \mid \alpha_{eu} + \alpha_{ev} \geq \lambda \ \forall e = (u, v) \text{ is feasible for } \alpha \in P(D, q)\}$$

For $D \leq D^*(1+\epsilon)$, we have $\lambda^* \leq 1+\epsilon$, else by scaling $\alpha$ we can show $\text{Dual}(D, q)$ is feasible for $D < D^*$, which is not possible. Therefore, Theorem 5 implies for $0 \leq \epsilon \leq 1/3$:

$$\sum_e y_e \geq (1-\epsilon)^2 C(y, \hat{D}, q) \geq (1-3\epsilon)C(y, \hat{D}, q)$$

2.4 Rounding Step: Recovering the Densest Subgraph

Using Theorem 2, we have a value $\hat{D} \in [(1-\epsilon)D^*, (1+\epsilon)D^*]$ along with dual variables $y$ that satisfies $\sum_c y_c \geq (1-3\epsilon)C(y, \hat{D}, q)$. We will now use these variables to recover an approximately optimal densest subgraph. We first discuss why this is not straightforward.

**Technical Hurdle.** The problem $\text{Dual}(\hat{D}, q)$ is different from the problem $\text{Dual}(\hat{D})$ in that the corresponding primal problems are different. The primal feasibility problem corresponding to $\text{Dual}(\hat{D}, q)$ is the following:

$$\text{Find } y, x, z \geq 0 \text{ s.t. } \frac{\sum_e y_e}{\sum_{e,v} (Dx_e + qz_{ev})} \geq 1$$
where \( y_e \leq \min(x_u + z_{cu}, x_v + z_{cv}) \) \( \forall e = (u, v) \in E \)

The primal feasibility problem corresponding to Dual(\( \tilde{D} \)) does not have variables \( z \). These problems are equivalent from the perspective of exact feasibility since the exact optimal primal solution of Dual(\( \tilde{D}, q \)) will indeed set \( z = 0 \), and the resulting \( x, y \) are precisely the vertex and edge variables in the LP formulation we began from. The catch is the following: An \( \epsilon \)-approximate solution using Dual(\( \tilde{D}, q \)) need not yield an \( \epsilon \)-approximate solution using Dual(\( \tilde{D} \)). The reason is that an approximately optimal solution to Dual(\( \tilde{D}, q \)) might have large \( z \), so that the resulting \( y, x \) variables have no easy interpretation.

Despite this difficulty, we show that for \( q = 2 \), we can round an \( \epsilon \)-approximate solution to Dual(\( \tilde{D}, q \)) into an \( \epsilon \)-approximate solution to Dual(\( \tilde{D} \)), and hence recover the approximate densest subgraph. We note that the primal problem itself is a fractional densest subgraph that must be further converted (or rounded) into an integer solution. We fold both the rounding steps into one in the proof below, noting that even recovering the fractional densest subgraph would need our new rounding method.

First recall that \( C(y, \tilde{D}, q) \) is computed as follows: Let \( \tilde{r} = |\tilde{D}/q| \), and \( \tilde{s} = \tilde{D} - q\tilde{r} \). For simplicity in the proof below, we assume \( \tilde{s} > 0 \). For any vertex \( v \), we sort the \( y_e \) values incident on \( v \) in decreasing order, and denote these \( y_1(v) \geq y_2(v) \geq \cdots \geq y_n(v) \). Then:

\[
C(y, \tilde{D}, q) = \sum_v \left( \sum_{k=1}^{\tilde{r}} qy_k(v) + \tilde{s}y_{\tilde{r}+1}(v) \right)
\]

The important point is that this is a linear function of \( y \).

**Step 1: Discretization.** This step is mainly to improve efficiency. Let \( Y = \max_v y_e \). Scale up or down the \( y_e \) values so that \( Y = 1 \). Consider all edges \( e \) with \( y_e \leq \epsilon/m^2 \). The contribution of these edges to the summation \( \sum_e y_e \) and to \( C(y, \tilde{D}, q) \) is at most \( \epsilon/m \). We set all these \( y_e = 0 \). Since we originally had \( \sum_e y_e \geq (1 - 3\epsilon)C(y, \tilde{D}, q) \), the new vector \( y \) satisfies: \( \sum_e y_e \geq (1 - 4\epsilon)C(y, \tilde{D}, q) \).

Now round each \( y_e \) down to the nearest power of \((1 + \epsilon) \). This does not change any individual \( y_e \) by more than a factor of \((1 + \epsilon) \). Therefore, the resulting \( y \) satisfies: \( \sum_e y_e \geq (1 - 6\epsilon)C(y, \tilde{D}, q) \). At this point, note that there are only \( O \left( \frac{\log m}{\epsilon} \right) \) distinct values of \( y_e \).

**Step 2: Line Sweep.** Fix any \( \gamma \geq 0 \). Let \( I(z) = 1 \) if \( z \geq \gamma \). Consider the process that includes edge \( e \) if \( y_e \geq \gamma \). Let \( G(\gamma) \) denote the subgraph induced by these edges; let \( E(\gamma) \) denote the set of induced edges; \( V(\gamma) \) denote the set of induced vertices; and let \( d_v(\gamma) \) denote the degree of \( v \) in \( G(\gamma) \). Note that \( d_v(\gamma) = \sum_{e \in N(v)} I(y_e) \), and \( |E(\gamma)| = \sum_e I(y_e) \). Furthermore, let:

\[
H_v(\gamma) = \sum_v \left( \sum_{k=1}^{\tilde{r}} I(y_k(v)) + \frac{\tilde{s}}{q} I(y_{\tilde{r}+1}(v)) \right)
\]
Lemma 4. There exists $\gamma$ such that $G(\gamma)$ is non-empty, and $|E(\gamma)| \geq q(1 - 6\epsilon) \sum_v H_v(\gamma)$. Furthermore, this value of $\gamma$ can be computed in $O\left(\frac{m \log m}{\epsilon} \right)$ time.

Proof. We note that:

$$\sum_{e} y_e = \int_{\gamma=0}^{1} |E(\gamma)| d\gamma$$

$$C(y, \tilde{D}, q) = q \int_{\gamma=0}^{1} \sum_v H_v(\gamma) d\gamma$$

Since $\sum_e y_e \geq (1 - 6\epsilon)C(y, \tilde{D}, q)$, this implies the existence of a $\gamma$ that satisfies the condition of the lemma. There are only $O\left(\log \frac{m}{\epsilon} \right)$ distinct values of $\gamma$, and computing $|E(\gamma)|$, $\sum_v \min(\tilde{D}/d, d_v(\gamma))$ takes $O(m)$ time for any $\gamma$.

Start with the value of $\gamma$ that satisfies $|E(\gamma)| \geq q(1 - 6\epsilon) \sum_v H_v(\gamma)$. Let $V_1$ denote the set of vertices such that for $v \in V_1$, we have $y_{\gamma_{v+1}}(v) \geq \gamma$. For these vertices, $H_v(\gamma) = \sum_{k=1}^{\tilde{D}} 1 + \delta/q = \tilde{D}/q$. Let $V_2$ denote the remaining vertices; for these we have $d_v(\gamma) = H_v(\gamma)$. Therefore, we have

$$\sum_v H_v(\gamma) = \tilde{D}/q \times |V_1| + \sum_{v \in V_2} d_v(\gamma)$$

Suppose we delete all vertices in $V_2$ simultaneously. Let $G(V_1, E_1)$ denote the subgraph induced on $V_1$. Then:

$$|E_1| \geq |E(\gamma)| - \sum_{v \in V_2} d_v(\gamma) \geq q(1 - 6\epsilon) \left( \tilde{D}/q \times |V_1| + \sum_{v \in V_2} d_v(\gamma) \right) - \sum_{v \in V_2} d_v(\gamma)$$

Therefore, $|E_1| \geq (1 - 6\epsilon) \tilde{D}|V_1|$ for $q \geq 2$ and $\epsilon < 1/12$.

The final technicality is to show that $G(V_1, E_1)$ is non-empty. There are now two cases: (a) If $\sum_{v \in V_2} d_v(\gamma) = 0$, then $|E_1| \geq |E(\gamma)| > 0$, so that $G(V_1, E_1)$ is non-empty. (b) Otherwise, the final inequality is strict and we again have $|E_1| > 0$. This implies $G(V_1, E_1)$ is always non-empty. The density of $G(V_1, E_1)$ is at least $\tilde{D}(1 - 6\epsilon) \geq D^*(1 - 7\epsilon)$, and we finally have the following theorem.

Theorem 3. For $\epsilon \in (0, 1/12)$, a subgraph of density $D^*(1 - \epsilon)$ can be computed in $O\left(\frac{m \log m}{\epsilon^2} \right)$ time.

The key point in the above proof is that the very final inequality crucially needs $q > 1 + 6\epsilon$; indeed for smaller values of $q$, there are examples where an approximately optimal solution to $\text{DUAL}(\tilde{D}, q)$ does not imply an approximately optimal densest subgraph in any natural way.
2.5 Summary of the Algorithm

Before presenting the MapReduce implementation details, we summarize the algorithm as follows:

- Define Dual($D, q$) and Oracle($y$) = $C(y, D, q)$ for $q = 2$.
- Decide feasibility of Dual($D, q$) using multiplicative weight method; wrap this in a discretized binary search to find the smallest $\hat{D}$ = $\bar{D}$ for which the problem is approximately feasible.
- The output of the previous step is a value $\tilde{D}$ and dual vector $y$ such that $\sum_y y_e \geq (1 - 3\epsilon) C(y, \tilde{D}, q)$.
- Discretize $y_e$ and throw away values at most $\epsilon/m^2$ times the largest value.
- Perform a line sweep to find $\gamma$ for which the subgraph induced by edges $e$ with $y_e \geq \gamma$ satisfies the condition in Lemma 4.
- Remove vertices with degree at most $\lfloor \tilde{D}/q \rfloor$ from this subgraph, and output the remaining subgraph.

2.6 Number of MapReduce Phases

We now show how to implement the above algorithm in $O\left(\log{m \epsilon^2}\right)$ MapReduce phases, where each phase operates on $O(m)$ total (key, value) pairs.

Oracle $C(y, D, q)$ Computation. The mappers take as input $<e, \alpha_{eu}, \alpha_{ev}, y_{old}>$, and produce $<u, y_{new}>$ and $<v, y_{new}>$. These are shuffled using the vertex as the key. The reducer for vertex $v$ needs to compute the $\lceil D/q \rceil$th largest $y_e$ and set $\alpha_{ev}$ for edges $e$ with larger $y_e$. This takes linear time in $d_{max}$, the maximum degree; this determines the reduce key complexity. The reducers output $\{<e, \alpha_{ev}, y_e>\}$ and $<v, S_v>$, where $S_v$ is the contribution of $v$ to $C(y, D, q)$.

The next map phase does a Combine step summation of the $S_v$ to compute the value of the oracle; the next shuffle phase shuffles the $<e, \alpha_{ev}, y_e>$ using the edge as the key; and the next reduce phase produces $<e, \alpha_{eu}, \alpha_{ev}, y_e>$. Therefore, the oracle can be implemented in two MapReduce phases.

Binary Search. As mentioned before, this takes $k(\epsilon) = O\left(\log{m \epsilon^2}\right)$ phases of the oracle computation.

Rounding Steps. The scaling requires estimating the maximum of the $y_e$, which requires one Combine phase over the vector $y$. The scaling and discretization can be done in the next Map phase. The various choices of $\gamma$ can be tried in sequence, one in each phase of MapReduce, giving $O\left(\log{m \epsilon^2}\right)$ phases. For each $\gamma$, the subgraph computed involves filtering the edges by their $y_e$ and throwing away small degree vertices; this takes $O(m)$ sequential complexity and $O(d_{max})$ key complexity. One final MapReduce phase can pick the best density from the various values of $\gamma$. We therefore have the following theorem:

**Theorem 4.** For $\epsilon \in (0, 1/12)$, a subgraph of density $D^*(1 - \epsilon)$ can be computed with $O(d_{max})$ key complexity, $O(m)$ sequential complexity per phase, and $O\left(\log{m \epsilon^2}\right)$ phases.
Note that restricting $\epsilon$ to be less than $1/12$ poses no problem in the binary search since we can always start with an $O(1)$ approximation using [4]. We finally note that the same technique applies to the weighted version of the problem, where edge $e$ has weight $w_e$ and the goal is to find the subgraph whose ratio of weight of induced edges to number of vertices is maximized. The details are simple and omitted.

References

A MapReduce Complexity Measures for Graph Problems

In this section, we present complexity measures for MapReduce computation restricted to graph problems. We refer the reader to a companion manuscript [8] for a detailed discussion. As an abstract model of parallel computation, there are two types of computing nodes. The input to either set of nodes are <key, value> pairs. A map operation takes a pair <k, v> and maps it to a set of new pairs {<k', v'>}. After a map phase, there is a shuffle phase, where all values with a given key k are collected and sent to one reduce operation. The reduce operation (working on a key and the corresponding set of values) generates a new set of pairs {<k', v'>}, which are fed back to the mappers to start the next phase. In a real implementation, several map and reduce operations are assigned to each physical machine of the compute cluster.

We characterize the performance of this model in terms of three complexity measures, which we define independent of the implementation details such as number of processors, Mappers, and Reducers (unlike abstractions such as in [12]).

Key Complexity. This captures the maximum input size, maximum memory, and maximum running time for a single map or reduce key per phase. The input size for a key simply captures the number of values associated with that key.

Sequential Complexity. This captures the total output size and total running time of all mappers and reducers in a phase. We assume the size of the output produced also captures the complexity of the shuffle operation (ignoring implementation details).

Number of Phases. Given a setting of the parameters described above, we wish to optimize the number of phases taken by the algorithm.

While a detailed description is beyond the scope of this paper, some important points to note are:

1. The key complexity measures capture the behavior of the system had there been “infinite parallelism” where each map or reduce operation could have been assigned its own processor.
2. The sequential complexity measures capture the behavior of the system if there were only one processing node.
3. The cost of a “shuffle” operation (i.e. sending data from mappers to reducers) is captured in the sequential complexity, since the shuffle size is the same as the output size.
4. This model is similar to the PRAM model (though there are important differences, as outlined in [12]). Hence, not surprisingly, our results also map naturally to the PRAM model.

In this paper, the mappers and reducers are linear (or near-linear) time and space algorithms. Therefore, the distinction between input size, memory, and
running time is not important. In this situation, we will use a simplified complexity model, where we report just a single number for key complexity (resp. sequential complexity) which is the maximum of the three detailed measures for key complexity (resp., sequential complexity). As a further simplification throughout this paper, we assume the map-key complexity is $O(1)$, i.e., each map-key has $O(1)$ values associated with it. In this setting, the key complexity will simply be the complexity of a reduce key.

We assume a graph $G(V,E)$ with $m$ edges, $n$ vertices, and maximum degree $d$. In the above framework, we would ideally like the key complexity to be $O(d)$, the sequential complexity to be $\tilde{O}(m)$, and minimum number of phases. For instance, our algorithm for undirected densest subgraph uses $O\left(\frac{\log n}{\varepsilon^2}\right)$ phases.

**Special Combine Phase.** Consider how a commutative and associative aggregation operation over all the data, such as max or sum is implemented with $K$ mappers and one reducer. Suppose there are $O(n)$ records (where $n$ is the number of vertices) that need to be aggregated. Each mapper can perform a streaming map-side aggregation of $n/K$ records and these results are passed to the reducer for further aggregation. Therefore, the key complexity is $O(n/K)$ input size, $O(1)$ memory, and $O(n/K)$ running time, while the sequential complexity is $O(K)$ output size, $O(K)$ memory and $O(n)$ running time. This feature is available in all MapReduce implementations, and is typically not a bottleneck compared to the other phases. We will simply assume a commutative and associative Combine operation on $O(n)$ records can be performed in one phase and ignore the exact running time and implementation details.

**B The Multiplicative Weights Update Framework**

We next present a simple algorithm for deciding feasibility of covering linear programs due to Young [17]; we use the exposition and bounds due to Arora-Hazan-Kale [2]. We first define the generic covering problem.

| COVERING: $\exists \mathbf{x} \in P$ such that $A\mathbf{x} \geq \mathbf{1}$, where $A$ is an $r \times s$ matrix and $P$ is a convex set in $\mathbb{R}^s$ such that $A\mathbf{x} \geq \mathbf{0}$ for all $\mathbf{x} \in P$. |

The running time of the algorithm is quantified in terms of the Width defined as:

$$\rho = \max_i \max_{\mathbf{x} \in P} a_i \mathbf{x}$$

The algorithm assumes an efficient oracle that solves the Lagrangian of the constraints $A\mathbf{x} \geq \mathbf{1}$. Given dual multipliers $y_i$ associated with the constraint $a_i \mathbf{x} \geq 1$, the oracle takes a linear combination of the constraints, and maximizes the LHS with respect to $\mathbf{x}$.

| ORACLE($y$): Given an $r$-dimensional dual vector $\mathbf{y} \geq 0$, solve $C(\mathbf{y}) = \max \{\mathbf{y}^T A\mathbf{z} : \mathbf{z} \in P\}$. |
By duality theory, and as explained in [2], it is easy to see that if there exists $y \geq 0$, $C(y) < ||y||_1$ (where $||y||_1 = y^t1 = \sum_{r=1}^{r} y_r$ is the $l_1$ norm of vector $y$), then $Ax < 1$ for all $x \in P$, and hence COVERING is infeasible. The multiplicative weight update procedure described in Fig. 1 iteratively updates the vector $y$ so that it either correctly declares infeasibility of the program or finds an $x \in P$ with $Ax \geq (1 - \epsilon)1$.

**Multiplicative Weight Update Algorithm**

Let $T \leftarrow 4\rho \log \frac{\epsilon}{\rho}$, $y_1 = 1$

For $k = 1$ to $T$ do:

Find $x_k$ using ORACLE($y_k$).

If $C(y_k) < ||y_k||_1$, then declare infeasible and stop.

Update $y_{k+1} \leftarrow y_k \left(1 - \frac{\lambda^* x_k}{\rho}\right)$ for $i = 1, 2, \ldots, r$.

Return $x^* = (\sum_k x_k)/T$.

Fig. 1. The Multiplicative Weight Update Algorithm for Covering Linear Programs.

The above procedure not only provides a guarantee on the final value $x^*$, but also yields a guarantee on the dual variables $y$ that it computes. Define an optimization version of COVERING as follows:

$$\lambda^* = \max \{\lambda : Ax \geq \lambda 1 \text{ and } x \in P\}$$

The problem COVERING is equivalent to deciding $\lambda^* \geq 1$. By the definition of $\lambda^*$, we have that for any $y \geq 0$, $\lambda^*||y||_1 \leq C(y)$: the multiplicative weight method makes this inequality approximately tight as well if it runs for $T$ steps. The next theorem is implicit in the analysis of [2], and is explicitly presented in [17,7].

**Theorem 5.** The multiplicative weight procedure either correctly outputs $Ax \geq 1$ is infeasible for $x \in P$, or finds a solution $x^* \in P$ such that $Ax^* \geq (1 - \epsilon)1$. Furthermore, in the latter case, there exists an iteration $k$ such that

$$\lambda^* \times ||y_k||_1 \geq (1 - \epsilon)C(y_k)$$

### C Densest Subgraph in Directed Graphs

Let $G(V, E)$ denote a directed graph. The densest subgraph problem asks to find two subsets $S, T \subseteq V$ (not necessarily disjoint) such that if $E(S, T)$ denotes the

---

iii

---

3 Young [17] shows that if $Ax \geq A1$ for the final solution, then the average over $k$ of $C(y_k)/||y_k||_1$ converges to at most $\lambda/(1 - \epsilon) \leq \lambda^*/(1 - \epsilon)$. This implies one of these values is at most $\lambda^*/(1 - \epsilon)$, implying the claim. The exact same guarantee is also explicitly obtained by Plotkin-Shmoys-Tardos [14], though their update procedure is somewhat different and much more involved.
set of edges \((u, v)\) where \(u \in S\) and \(v \in T\), then the density \(D(S, T) = \frac{|E(S, T)|}{\sqrt{|S||T|}}\) is maximized. Let \(OPT\) denote the optimal density. We present an algorithm with running time \(O\left(m \log^2 m \epsilon^{-3}\right)\) that outputs a subgraph of density \((1 - \epsilon)OPT\). We will only outline the portions different from the undirected case.

C.1 Parametric LP Formulation

We start with a modification of an ingenious parametrized linear program of Charikar [5]. Consider the following linear program \(Primal(z)\). There is a variable \(y_e\) for each edge \(e \in E\), and variables \(s_v, t_v\) for each vertex \(v \in V\). The value \(z\) is a parameter that we will search over.

\[
\text{Maximize } \sum_{e} y_e
\]

\[
y_e \leq s_u \quad \forall e = (u, v) \in E
\]

\[
y_e \leq t_v \quad \forall e = (u, v) \in E
\]

\[
\sum_v (zs_v + \frac{1}{2} t_v) \leq 2
\]

\[
y_e, s_v, t_v \geq 0 \quad \forall e, v
\]

The difference with the LP in [5] is that our penultimate constraint is obtained by taking a linear combination of two constraints in their LP. This leads to a nicer dual formulation where the dual objective can be directly interpreted as the density.

\[\text{Lemma 5 ([5]), } OPT \leq \max_z \text{Primal}(z).\]

**Proof.** Let \(S, T \subseteq V\) denote the densest subgraph. Set \(z = \sqrt{|T|/|S|}\). Set \(y_e = 1/\sqrt{|S||T|}\) for all \(e\) within the subgraph; set \(s_u = 1/\sqrt{|S||T|}\) for all \(u \in S\), and set \(t_v = 1/\sqrt{|S||T|}\) for all \(v \in T\). It is easy to check that this is a feasible solution with objective value exactly equal to the density of this subgraph.

Let \(D^* = \max_z \text{Primal}(z)\). Consider the following dual linear program \(\text{Dual}(D, z)\):

\[
\alpha_{eu} + \alpha_{ev} \geq 1 \quad \forall e = (u, v) \in E
\]

\[
\sum_{e \mid e = (v, w)} \alpha_{ev} \leq Dz/2 \quad \forall v \in V
\]

\[
\sum_{e \mid e = (u, v)} \alpha_{ev} \leq D/(2z) \forall v \in V
\]

\[
\alpha_{ev} \geq 0 \quad \forall e, v
\]

Using strong duality, we immediately have:

\[\text{Lemma 6. } D^* = \max_z \min\{D | \text{Dual}(D, z) \text{ is feasible}\}.\]
C.2 Covering Program and Width Modulation

In order to decide if \textsc{Dual}(D, z) is feasible, we use the multiplicative weight framework to decide the feasibility of the set of constraints:

\[
\alpha_{eu} + \alpha_{ev} \geq 1 \quad \forall e = (u, v) \in E
\]

subject to the polyhedral constraints \( P(D, z) \):

\[
\begin{align*}
\sum_{e | e = (v, w)} \alpha_{ev} &\leq Dz/2 \quad \forall v \in V \\
\sum_{e | e = (u, v)} \alpha_{ev} &\leq D/(2z) \quad \forall v \in V \\
\alpha_{ev} &\leq 2 \quad \forall e, v \\
\alpha_{ev} &\geq 0 \quad \forall e, v
\end{align*}
\]

As before, the constraints \( \alpha_{ev} \leq 2 \) have been added to reduce the width of the program. Note that the width is at most 4. The \textsc{Oracle}(y) problem assigns dual vector \( y \) to the set of constraints \( \alpha_{eu} + \alpha_{ev} \geq 1 \) and solves:

\[
C(y, D, z) = \max_{\alpha \in P(D, z)} \sum_v \sum_{e \text{ incident on } v} y_e \alpha_{ev}
\]

This can be solved in \( O(m) \) time: For each vertex \( v \), we only need to find the top \( Dz/4 \) values \( y_e \) for edges leaving \( v \), and the top \( D/(4z) \) values \( y_e \) for edges entering \( v \); we set \( \alpha_{ev} = 2 \) for the corresponding edges. We assume for simplicity of exposition that \( Dz/4 \) and \( D/(4z) \) are integers; the proof extends to the general case with minor modification.

**Lemma 7.** For any \( D, z \) and \( \epsilon \in [0, 1] \), in time \( O\left(m \log \frac{m}{\epsilon^2}\right) \), the multiplicative weight update algorithm either returns that \( \textsc{Dual}(D, z) \) is infeasible, or finds \( \alpha \in P(D, z) \) such that \( \alpha_{eu} + \alpha_{ev} \geq 1 - \epsilon \) for all \( e = (u, v) \in E \).

C.3 Parametric Search

We apply the multiplicative weight update algorithm within the following parametric search procedure. Discretize \( z \) and \( D \) in powers of \((1 + \delta)\). For each discretized \( z \), perform a binary search to find the smallest discretized \( D \) (call this \( \tilde{D}(z) \)) for which the multiplicative weight algorithm returns a feasible \((\delta\)-optimal) solution. Find that \( z \) which maximizes \( \tilde{D}(z) \); denote this value of \( z \) as \( \tilde{z} \) and this value of \( \tilde{D}(z) \) as \( \tilde{D} \). Since the density is upper bounded by \( m \), and since we can assume \( D/z \) and \( Dz \) lie in \([1, 2n]\), the number of parameters we try is \( O\left(\frac{\log m}{\delta}\right) \) ignoring lower order terms. This increases the running time by the corresponding factor compared to the undirected case.

Let \((\alpha, y)\) denote the final solution found by the above procedure, where the dual vector \( y \) is found as in Theorem 5. The following theorem is proved analogously to Theorem 2 by choosing \( \delta \) to be a sufficiently small constant fraction of \( \epsilon \). (The only additional observation we need is that modifying \( z \) by a factor of \((1 + \delta)\) changes \( \tilde{D}(z) \) by at most that factor.)
Theorem 6. For $0 \leq \epsilon \leq 1/3$, the values $\tilde{D}, \tilde{z}$ and the final solution $(\alpha, y)$ satisfy:

1. $D^*(1 - \epsilon) \leq \tilde{D} \leq D^*(1 + \epsilon)$.
2. $\sum_e y_e \geq (1 - 3\epsilon)C(y, \tilde{D}, \tilde{z})$

C.4 Rounding Step: Recovering the Densest Subgraph

Using Theorem 6, we have a value $\tilde{D} \in [(1 - \epsilon)D^*, (1 + \epsilon)D^*]$ along with a value $\tilde{z}$ and dual variables $y$ that satisfy $\sum_e y_e \geq (1 - 3\epsilon)C(y, \tilde{D}, \tilde{z})$. We will now use these variables to recover an approximately optimal densest subgraph.

Step 1: Discretization. First note that $C(y, \tilde{D}, \tilde{z})$ is computed as follows: For each $v \in V$, sum up the largest at most $\tilde{D}\tilde{z}/4 y_e$ for $e = (v, w)$, and the largest $\tilde{D}/(4\tilde{z}) y_e$ for $e = (w, v)$, and double this value. Let $Y = \max_e y_e$. Scale up or down the $y_e$ values so that $Y = 1$. We eliminate all edges with $y_e \leq \epsilon/m^2$ and round each $y_e$ down to the nearest power of $(1 + \epsilon)$. As shown before, the resulting $y$ satisfies:

$$\sum_e y_e \geq (1 - 6\epsilon)C(y, \tilde{D}, \tilde{z})$$

At this point, note that there are only $O\left(\log \frac{m}{\epsilon}\right)$ distinct values of $y_e$.

Step 2: Line Sweep. Let $G(\gamma)$ denote the subgraph induced by edges with $y_e \geq \gamma$. Let $E(\gamma)$ denote the set of induced edges, $S(\gamma)$ denote the set of induced source vertices, and $T(\gamma)$ denote the set of induced destination vertices. Let $d^S_v(\gamma)$ and $d^T_v(\gamma)$ denote the out-degree and in-degree respectively of $v$ in $G(\gamma)$.

Lemma 8. There exists $\gamma$ such that

$$|E(\gamma)| \geq 2(1 - 6\epsilon) \sum_v \left(\min \left(\tilde{D}\tilde{z}/4, d^S_v(\gamma)\right) + \min \left(\tilde{D}/(4\tilde{z}), d^T_v(\gamma)\right)\right)$$

Furthermore, this value of $\gamma$ can be computed in $O\left(m \frac{\log m}{\epsilon}\right)$ time.

Proof. We note that:

$$\sum_e y_e = \int_{\gamma=0}^1 |E(\gamma)| d\gamma$$

and

$$C(y, \tilde{D}, \tilde{z}) = 2 \int_{\gamma=0}^1 \sum_v \left(\min \left(\tilde{D}\tilde{z}/4, d^S_v(\gamma)\right) + \min \left(\tilde{D}/(4\tilde{z}), d^T_v(\gamma)\right)\right) d\gamma$$

Since $\sum_e y_e \geq (1 - 6\epsilon)C(y, \tilde{D}, \tilde{z})$, this implies the existence of a $\gamma$ that satisfies the condition of the lemma. There are only $O\left(\frac{\log m}{\epsilon}\right)$ distinct values of $\gamma$, and computing $|E(\gamma)|$ and the degrees of the vertices takes $O(m)$ time for any $\gamma$. 

vi
Start with the value of $\gamma$ that satisfies the condition of the previous lemma. Let $S_1$ denote the set of vertices with $d^S_v(\gamma) \geq \tilde{D}/(4\tilde{\varepsilon})$ and $T_1$ denote the set of vertices with $d^T(\gamma) \geq \tilde{D}/(4\tilde{\varepsilon})$. We have:

$$\sum_v \left( \min \left( \frac{\tilde{D}}{4\tilde{\varepsilon}}, d^S_v(\gamma) \right) + \min \left( \frac{\tilde{D}}{4\tilde{\varepsilon}}, d^T_v(\gamma) \right) \right) = \frac{\tilde{D}}{4} \left( \tilde{\varepsilon}|S_1| + \frac{|T_1|}{\tilde{\varepsilon}} \right) + \sum_{v \in V \setminus S_1} d^S_v(\gamma) + \sum_{v \in V \setminus T_1} d^T_v(\gamma)$$

Consider the sets $(S_1, T_1)$ and the edge set $E(S_1, T_1)$ that goes from $S_1$ to $T_1$. Since this edge set is obtained by deleting edges out of $V \setminus S_1$ and edges into $V \setminus T_1$, the above conditions imply:

$$|E(S_1, T_1)| \geq |E(\gamma)| - \left( \sum_{v \in V \setminus S_1} d^S_v(\gamma) + \sum_{v \in V \setminus T_1} d^T_v(\gamma) \right)$$

$$\geq 2(1 - 6\epsilon) \left( \frac{\tilde{D}}{4} \left( \tilde{\varepsilon}|S_1| + \frac{|T_1|}{\tilde{\varepsilon}} \right) + \sum_{v \in V \setminus S_1} d^S_v(\gamma) + \sum_{v \in V \setminus T_1} d^T_v(\gamma) \right)$$

$$= (1 - 6\epsilon) \frac{\tilde{D}}{2} \left( \tilde{\varepsilon}|S_1| + |T_1|/\tilde{\varepsilon} \right)$$

for $\epsilon \leq 1/12$. Now observe that for any $\tilde{\varepsilon}$, we have $\tilde{\varepsilon}|S_1| + |T_1|/\tilde{\varepsilon} \geq 2\sqrt{|S||T|}$. Therefore:

$$\text{Density of } (S_1, T_1) = \frac{|E(S_1, T_1)|}{\sqrt{|S||T|}} \geq (1 - 6\epsilon)\tilde{D} \geq (1 - 7\epsilon)D^*$$

**Theorem 7.** For any $\epsilon \in (0, 1/12)$, a directed subgraph with density $D^*(1 - \epsilon)$ can be computed in time $O\left(m \log^2 \frac{m}{\epsilon^2}\right)$.

**MapReduce Implementation.** The details of the Oracle computation and the rounding step are the same as the undirected case. The parametric search can be done in parallel for all values of $D, z$; the sequential complexity per phase is now $O\left(m \log^2 \frac{m}{\epsilon^2}\right)$ since the parametric search is over both $D$ and $z$ instead of just over $D$. Therefore, the overall implementation still takes $O\left(\log \frac{m}{\epsilon^2}\right)$ phases with $O\left(m \log^2 \frac{m}{\epsilon^2}\right)$ sequential complexity per phase, and $O(d_{\max})$ key complexity, where $d_{\max}$ is the maximum degree.

**D Fractional Matchings in Bipartite Graphs**

In this section, we show the broader applicability of width modulation by presenting an FPTAS for fractional maximum size matchings in bipartite graphs;
incidentally, we also obtain a slight decrease in the number of phases. As motivation, consider the following Adword allocation problem [13]: There are $n_1$ advertisers, where advertiser $u$ has demand $d_u$; there are $n_2$ ad slots, where slot $v$ has supply $s_v$; furthermore there is a bipartite graph specifying which advertiser is interested in which ad slots. The goal is to find a (possibly fractional) allocation that maximizes the demand satisfied.

We present an efficient algorithm for the case where all supplies and demands are 1 (the maximum fractional matching case); this algorithm extends with minor modification to the general case. On a graph with $n$ vertices and $m$ edges, our algorithm takes $O \left( \frac{m \log n}{\epsilon^3} \right)$ running time, and can be implemented in $O \left( \frac{\log^2 n}{\epsilon^3 \log d} \right)$ phases of MapReduce with key complexity $O(d)$, the maximum degree, and sequential complexity $O(m)$ per phase. The number of phases improves by a log $d$ factor that which can be achieved by the algorithm of Ahn-Guha [1], which was designed for the semi-streaming model.

Formally, we are given a bipartite graph $G(V_1, V_2, E)$ on $n$ vertices. Let $K^*$ denote the size of the optimal fractional matching. For vertex $v$, let $N(v)$ denote the set of edges incident on $v$. The maximum fractional matching is the solution to the following linear program.

Maximize $\sum_e y_e$

subject to the polyhedral constraints $P(K)$.

As before, we write the following dual program $\text{Dual}(K)$.

$x_u + x_v \geq 1 \forall e = (u, v) \in E$

$\sum_{v \in V_1 \cup V_2} x_v \leq K$

It is well-known that the dual is fractional Vertex Cover, where variable $x_v$ captures whether vertex $v$ is in the cover. By strong duality, $\text{Dual}(K)$ is feasible iff $K \geq K^*$.

D.1 Covering Program, Width Modulation, and Binary Search

We will now check the feasibility of the constraints:

$x_u + x_v \geq 1 \forall e = (u, v) \in E$

subject to the polyhedral constraints $P(K)$.

$\sum_v x_v \leq K$

$x_v \leq 1/\epsilon \forall v \in V_1 \cup V_2$

As before, the second set of constraints is added to reduce the width of the program. Without these constraints, the width can be as large as $2K$. With
these constraints, it becomes at most $2/\epsilon$. For simplicity of exposition, we assume from now that $\epsilon K$ is an integer; the proof only needs a minor modification if this assumption is removed.

The problem ORACLE($y$) assigns dual variables $y$ to the set of constraints $x_u + x_v \geq 1$, and computes:

$$C(y, K) = \max_{x \in P(K)} \sum_{v \in V_1 \cup V_2} x_v \left( \sum_{e \in N(v)} y_e \right)$$

This involves the following $O(m)$ time computations: (1) Compute $S_e(y) = \sum_{e \in N(v)} y_e$ for each vertex $v$; and (2) Compute the top $\epsilon K$ values $S_e(y)$; sum these; and multiply this result by $1/\epsilon$.

The following theorem is now analogous to Theorem 5:

**Theorem 8.** For any integer $K$ and $\epsilon \in [0, 1]$, in time $O\left( \frac{\log n}{\epsilon^2} \right)$, the multiplicative weight algorithm either returns that DUAL($K$) is infeasible, or finds $x \in P(K)$ such that $x_u + x_v \geq 1 - \epsilon$ for all $e = (u, v) \in E$.

As before, we apply the multiplicative weight algorithm in the following Binary Search procedure: Round the values $K$ in powers of $(1 + \epsilon)$. Perform binary search over these discretized values to find the smallest $K$ (call this $\tilde{K}$) for which the multiplicative weight algorithm returns a feasible solution. Let $(x, y)$ denote the final solution found by the above procedure, where the dual solution $y$ is found as in Theorem 5. The following theorem is now analogous to Theorem 2:

**Theorem 9.** For $0 \leq \epsilon \leq 1/3$, the value $\tilde{K}$ and the final solution $(x, y)$ satisfy: (1) $K^*(1 - \epsilon) \leq \tilde{K} \leq K^*(1 + \epsilon)$; and (2) $\sum_e y_e \geq (1 - 3\epsilon)C(y, \tilde{K})$.

### D.2 Rounding Step: Recovering the Fractional Matching

We start with the $\tilde{K} \in [K^*(1 - \epsilon), K^*(1 + \epsilon)]$ and $y$ computed above. Recall that $S_v(y) = \sum_{e \in N(v)} y_e$. Therefore, $\sum_{e \in E} y_e = \frac{1}{2} \sum_v S_v(y)$. Let $\kappa = \epsilon \tilde{K}$. Suppose the vertices are sorted in decreasing order of $S_v(y)$. Let $Q$ denote the set of first $\kappa$ vertices. The previous theorem implies:

$$\sum_{e} y_e \geq (1 - 3\epsilon)C(y, \tilde{K}) = \frac{1 - 3\epsilon}{\epsilon} \sum_{v \in Q} S_v(y)$$

where the final equality follows from the definition of $C(y, \tilde{K})$. Let $v^* \in V \setminus Q$ have the largest $S_v(y)$. Since $S_v(y) \geq S_{v^*}(y)$ for all $v \in Q$ and since $|S_v(y)| = \kappa$, we have $\sum_{v \in Q} S_v(y) \geq \kappa S_{v^*}(y)$.

Construct $z$ as follows: For every $e$ incident on a vertex in $Q$, we set $z_e = 0$; otherwise, set $z_e = y_e$. Note that $S_v(z) \leq S_v(y)$ and $\max_v S_v(z) = S_{v^*}(y)$. We have the following sequence of inequalities:

$$\sum_e z_e \geq \sum_{e} y_e - \sum_{v \in Q} S_v(y) \geq \left( \frac{1 - 3\epsilon}{\epsilon} - 1 \right) \sum_{v \in Q} S_v(y) \geq \frac{1 - 4\epsilon}{\epsilon} \tilde{K} \cdot S_{v^*}(y) = (1 - 4\epsilon)\tilde{K} S_{v^*}(y)$$
Since \( \max_v S_v(z) = S_{v^*}(y) \) by construction, we have:
\[
\sum_{e} z_e \geq \frac{1}{\max_v S_v(z)} \geq (1 - 4\epsilon)K \geq (1 - 5\epsilon)K^*
\]
for \( \epsilon \in (0, 1/6) \). By scaling so that \( \max_v S_v(z) = 1 \), it is easy to check that the new vector \( z \) is a feasible fractional matching with value at least \( (1 - 5\epsilon)K^* \). We therefore have:

**Theorem 10.** For \( \epsilon \in (0, 1/6) \), a fractional matching of value \( K^*(1 - \epsilon) \) can be computed in \( O\left(\frac{\log m}{\epsilon^3}\right) \) time.

**MapReduce Implementation.** The key difference in the Oracle computation is that it requires summing the top \( K\epsilon \) values \( S_v(y) \). Since we assume reduce key complexity \( O(d) \), the maximum degree which can be much smaller than \( n \), this step will in general take \( O(\log_d n) \) Combine phases to execute via random partitioning. Therefore, for any given \( K \), the multiplicative weight procedure will require \( O\left(\frac{\log^2 n}{\epsilon^3 \log d}\right) \) phases to execute. The remaining details are the same as before, so that the overall implementation takes \( O\left(\frac{\log^2 n}{\epsilon^3 \log d}\right) \) MapReduce phases with \( O\left(m \frac{\log n}{\epsilon^3}\right) \) sequential complexity per phase and \( O(d) \) key complexity.