A New Approach to Computing Maximum Flows using Electrical Flows

Yin Tat Lee
MIT
yintat@mit.edu

Satish Rao
UC Berkeley
satishr@cs.berkeley.edu

Nikhil Srivastava
Microsoft Research, India
niksri@microsoft.com

ABSTRACT

We give an algorithm which computes a $(1-\epsilon)$-approximately maximum $st-$flow in an undirected uncapacitated graph in time $O(\frac{1}{\epsilon}\sqrt{\frac{m}{F}} \cdot m \log^2 n)$ where F is the flow value. By trading this off against the Karger-Levine algorithm for undirected graphs which takes $\tilde{O}(m + nF)$ time, we obtain a running time of $\tilde{O}(mn^{1/3}/\epsilon^{2/3})$ for uncapacitated graphs, improving the previous best dependence on ϵ by a factor of $O(1/\epsilon^3)$. Like the algorithm of Christiano, Kelner, Madry, Spielman and Teng, our algorithm reduces the problem to electrical flow computations which are carried out in linear time using fast Laplacian solvers. However, in contrast to previous work, our algorithm does not reweight the edges of the graph in any way, and instead uses *local* (i.e., non $s-t$) electrical flows to reroute the flow on congested edges. The algorithm is simple and may be viewed as trying to find a point at the intersection of two convex sets (the affine subspace of st-flows of value F and the ℓ_∞ ball) by an accelerated version of the method of alternating projections due to Nesterov.

By combining this with Ford and Fulkerson's augmenting paths algorithm, we obtain an exact algorithm with running time $\tilde{O}(m^{5/4}F^{1/4})$ for uncapacitated undirected graphs, improving the previous best running time of $\tilde{O}(m+\min(nF, m^{3/2}))$.

We give a related algorithm with the same running time for approximate minimum cut, based on minimizing a smoothed version of the ℓ_1 norm inside the cut space of the input graph. We show that the minimizer of this norm is related to an approximate blocking flow and use this to give an algorithm for computing a length k approximately blocking flow in time $\tilde{O}(m\sqrt{k})$.

Categories and Subject Descriptors

F.2.0 [**Analysis of Algorithms**]: General

Keywords

Maximum flow, minimum cut, electrical flow, Nesterov

STOC'13, June 1-4, 2013, Palo Alto, California, USA.

1. INTRODUCTION

The $st-$maximum flow problem and its dual, the $st-$minimum cut problem, are central in combinatorial optimization and have a wide range of applications. Several decades of work on these problems have yielded a host of combinatorial algorithms for solving them exactly and approximately, the fastest of which run in time $O(n^{3/2})$ on an $n-$vertex graph with $O(n)$ edges. Recently, the breakthrough work of Christiano et al. [CKM$^+$11] improved this bound significantly by giving a $(1-\epsilon)$-approximation algorithm for maximum flow which runs in time $\tilde{O(mn^{1/3}\epsilon^{-11/3})}$ on an undirected graph with m edges. Their algorithm uses the multiplicative weights framework [AHK12] to reduce the maximum flow computation to a series of *electrical flow* computations, which are solved in $\tilde{O(m)}$ time using fast Laplacian solvers.

In this paper, we present an alternate approach to these problems which also relies on electrical flows, but does not use multiplicative weights. Consider the following procedure for finding a feasible st-flow in a unit capacity undirected graph G. Identify G with a resistor network in which all edges have resistance 1Ω. Route F units of current from s to t. For each edge $e = (u, v)$ with too much flow $f(u, v) > 1$, create a current source at u of with value equal to the excess flow, $f(u, v) - 1$, and a current sink at v of the same value. Route this flow electrically. Repeat.

We observe that this rather simple algorithm corresponds to gradient descent in a certain space, and computes a $(1+\epsilon)$-approximately feasible flow (if there is one) in $O(\frac{m}{\epsilon^2})$ iterations. Moreover, it can be accelerated using the optimal gradient method of Nesterov [Nes05] to run in $O(\frac{\sqrt{m}}{\epsilon})$ iterations. If we are willing to settle for a cruder flow which is not approximately feasible but merely has small total flow volume violating the edge capacities, then this may be further reduced to $O(\frac{1}{\epsilon}\sqrt{\frac{m}{F}})$ iterations; the crude flow can then be fixed using a (nearly-linear time) combinatorial procedure to obtain a feasible $(1-\epsilon)$-approximately maximum flow.

Together, these observations reduce the $(1-\epsilon)$-approximately maximum flow computation to $\frac{1}{\epsilon}\sqrt{\frac{m}{F}}$ electrical flow compuations. We use the linear time Laplacian linear system solvers of Koutis, Miller and Peng [KMP11] to compute these electrical flows and trade off our procedure with the $\tilde{O(m + nF)}$ algorithm of Karger and Levine [KL02], which is fast for small F, to give a $(1-\epsilon)$-approximate maximum flow algorithm for unit capacity graphs with a worst case runtime of $\tilde{O}(\frac{mn^{1/3}}{\epsilon^{2/3}})$. Thus, our algorithm is faster than that of [CKM$^+$11] for unit capacity graphs by a factor of

$O(1/\epsilon^3)$, and asymptotically faster in the input size for large flow values.

The method described above is quite intuitive, but proving its correctness is based on understanding the flow problem geometrically. In particular, the simple algorithm and the acceleration method can be viewed as alternately projecting a flow of value F onto the ℓ_∞ ball (which may violate conservation constraints) and then back into the affine subspace of $st-$flows. Each alternation step amounts to a projected gradient step in the space of flows, which in turn corresponds to electrically rerouting excess flow on congested edges.

In contrast, the algorithm of [CKM+11] routes an $st-$electrical flow, penalizes congested edges by increasing their resistances (and occasionally through removal), and repeats. Eventally, the average of the flows computed is output. The error dependence of at least $1/\epsilon^2$ is fundamental in this approach as any use of multiplicative weights has this type of behavior [KY99]. On the other hand, this method works for graphs with arbitrary capacities, whereas our method is significantly slower with iteration count $\frac{1}{\epsilon}\sqrt{\frac{W}{F}}$ for a graph of total capacity W. This type of polynomial dependence on weights is normally the case for methods that are based purely on gradients, eg. [BI04].

In addition, we show how the resulting approximate flow can be quickly rounded to an integral flow and corrected by the Ford-Fulkerson algorithm [FF56]. This gives an exact maximum flow algorithm with running time $\tilde{O}(m^{5/4}F^{1/4})$, which is the fastest known for simple undirected uncapacitated graphs whenever $n^4F^3 > m^5$.

We also present an algorithm for finding minimum cuts. As in the case of our flow algorithm, the workhorse of our method is Nesterov's accelerated gradient method, which combines gradients that correspond to certain electrical flows, this time with *nonzero excesses* at the vertices, to arrive at a (non $s-t$) electrical flow whose potential differences give a good embedding of the vertices into $\mathbf{R}$. This embedding is then swept to obtain a $(1+\epsilon)$-minimum cut. Unlike the flow algorithm, this procedure does not have an alternating projections interpretation and amounts to minimizing a certain smoothing of the ℓ_1 norm. We show that the vector of potential differences of the minimizer is an approximate maximum flow; this is surprising because it is not known how to find a maximum flow from a minimum cut. Finally, we show that the flow computed in this manner has the property that every sufficiently short flow path contains an almost saturated edge; we thus call it an *approximate blocking flow*.

While the multiplicative weights framework has been widely applied in computer science [AHK12], the methods of Nesterov [Nes05] have received only modest attention. We believe that it deserves more attention in light of its generality and improved dependence on ϵ. A particularly interesting direction is to combine the mild dependence of multiplicative weights on capacities with the ϵ-dependence of Nesterov. We discuss this possibility at length in the conclusion,

We believe that the main contribution of this work is the flexible yet structured geometric viewpoint that we use to derive and analyze our algorithms. The use of affine subspaces to represent cuts and flows, which is classical in algebraic graph theory, interacts naturally with the recently developed fast Laplacian solvers. We hope that this viewpoint provides useful tools towards designing a nearly linear time algorithm.

1.1 Related Work

Maximum flow algorithms have a long history dating from Ford and Fulkerson's result establishing the maximum flow minimum cut theorem and giving an $O(Fm)$ algorithm for finding a flow of value F. A recent survey can be found in [Gol98]. The currently fastest known algorithm [GR98] runs in $\tilde{O}(m\min(n^{2/3}, \sqrt{m}))$ time. Karger [Kar98] introduced a graph smoothing method that can be used in conjuction with [GR98] to give an $\tilde{O}(\frac{m\sqrt{n}}{\epsilon})$ time $1+\epsilon$ approximation algorithm for maximum flow on undirected graph. Karger and Levine [KL02] gave a method to apply random sampling in residual graphs of undirected graphs and it leads to an $\tilde{O}(m+nF)$ time algorithm for capacitated undirected graphs, normalized so that the minimum capacity is 1.

Prior to [CKM+11], Daitch and Spielman [DS08] used Laplacian solvers to provide a maximum flow algorithm along with improved algorithms for minimum cost flow and generalized flow. The running time of their flow algorithm was $\tilde{O}(m^{3/2})$.

The previously mentioned algorithm in [CKM+11] combined with techniques of Karger [Kar98], yields $O(mn^{1/3}/\epsilon^{11/3})$ time approximation algorithm for capacitated undirected graphs. This is faster than our approach for graphs with large capacities.

2. PRELIMINARIES AND NOTATION

2.1 Cuts and Flows

Let $G = (V, E)$ be an undirected connected graph with n vertices and m edges. Orient the edges arbitrarily so that each edge has a head and a tail. Given a distinguished *source* $s \in V$ and *sink* $t \in V$, an $s-t$ *flow* is a vector $f : E \to \mathbf{R}$ satisfying the flow conservation constraints

$$\sum_{e:\text{head}(e)=v} f(e) - \sum_{e:\text{tail}(e)=v} f(e) = 0$$

at all $v \in V \setminus \{s, t\}$. The total flow out of the source s is called the *value* of f. In a unit capacity graph, we call any edge e with $|f(e)| > 1$ *congested*, and we refer to $|f(e)|-1$ as the *overflow* on e. A flow in which no edges are congested, i.e., $\|f\|_\infty \leq 1$, is called *feasible*.

The *maximum $st-$flow* problem consists of finding a feasible st-flow of maximum value. An *$st-$cut* is a subset $S \subset V$ with $s \in V$ and $t \notin V$. The *value* of a cut is the total capacity of the edges crossing it. The *minimum st-cut* problem seeks to find the $st-$cut of minimum value. It is the dual of the maximum $st-$flow problem, i.e., the value of the minimum $st-$cut is equal to the value of the maximum $st-$flow [FF56].

Given an $st-$flow f, the *$\delta-$approximate residual graph* $G_{f,\delta}$ is defined by deleting all edges of G with $|f(e)| > 1-\delta$. We say that f is an approximate (k, δ) blocking flow if the distance between s and t in $G_{f,\delta}$ is at least k.

2.2 The Incidence Matrix, The Laplacian, and Fast Laplacian Solvers

Given an arbitrary orientation of the edges, we define the *signed edge-vertex incidence matrix* $B_{m\times n}$ as:

$$B(e, v) = \begin{cases} 1 & \text{if } v \text{ is } e\text{'s head} \\ -1 & \text{if } v \text{ is } e\text{'s tail} \\ 0 & \text{otherwise} \end{cases}$$

Thus the rows of B are the signed incidence vectors $\chi_u - \chi_v$ of edges $uv \in E$, where $\chi_u \in \mathbf{R}^n$ is the canonical basis vector with a 1 in coordinate u.

The *Laplacian matrix* of G is the $n \times n$ symmetric matrix

$$L := B^T B = \sum_{uv \in E} (\chi_u - \chi_v)(\chi_u - \chi_v)^T.$$

It is easy to see that for a connected graph, $\mathrm{im}(L) = \mathrm{im}(B^T) = \mathrm{im}(\mathbf{1})^\perp$. We will frequently use the pseudoinverse of the Laplacian, which is the unique matrix L^+ satisfying

$$LL^+x = L^+Lx = x \qquad \forall x \perp \mathbf{1}.$$

The incidence matrix B splits $\mathbf{R}^m$ into two important subspaces which we will use extensively: the *cut space* $\mathrm{im}(B)$ and its orthogonal complement the *cycle space* $\mathrm{im}(B)^\perp = \ker(B^T)$. It is classical in algebraic graph theory that the cut space is the span of the signed incidence vectors of cuts in G and the flow space is the span of the signed incidence vectors of cycles. The orthogonal projection onto the cut space is given by

$$\Pi = BL^+B^T$$

and the projection onto the cycle space is

$$\Pi^\perp = I - BL^+B^T.$$

We will crucially use the fact that linear equations in Laplacian matrices $Lx = b$ can be solved to high precision in nearly-linear time. Specifically, we will use the following theorem of Koutis-Miller-Peng [KMP11]:

THEOREM 1 (FAST LAPLACIAN SOLVER [KMP11]). *There is an algorithm* LSolve *which on input a Laplacian matrix $L \in \mathbf{R}^{n\times n}$, a vector $b \in \mathbf{R}^n$, and a parameter $\epsilon > 0$ outputs a vector $\tilde{x} \in \mathbf{R}^n$ satisfying*

$$\|\tilde{x} - L^+b\|_L \le \epsilon \|L^+b\|_L, \tag{1}$$

where $\|v\|_L = \sqrt{v^T Lv}$ is the $L-$norm. The algorithm runs in time $O(m \log n \log(1/\epsilon))$ where m is the number of nonzero entries in L.

We will mainly use LSolve to approximately compute the projections $\Pi y = BL^+B^Ty$ and $\Pi^\perp y$ for various y. For technical reasons, it will be convenient to require that the approximate projections we compute lie *exactly* in $\mathrm{im}(\Pi)$ and $\mathrm{im}(\Pi^\perp)$, respectively. This additional requirement can be easily met in nearly linear time by applying a simple post-processing procedure which was described in [CKM+11] (Appendix A). We encapsulate this fact below; the notation $\mathrm{poly}(n)$ denotes a sufficiently large polynomial n^c in n, whose exact value does not matter as it goes under the logarithm.

COROLLARY 1. *There is an algorithm which on input $B \in \mathbf{R}^{m\times n}$ with $2m$ nonzeros and a vector $y \in \mathbf{R}^m$ with $\|y\|_2 = O(\mathrm{poly}(n))$ outputs a vector $q \in \mathbf{R}^m$ satisfying*

$$\|\Pi^\perp y - q\|_2 \le 1/\mathrm{poly}(n) \qquad \text{and} \qquad \Pi^\perp q = q$$

in time $O(m \log^2 n)$. As $\Pi y = (I - \Pi^\perp)y$, the same kind of guarantee is immediate for Πy.

It is well known (see for instance the monograph [Vis]) that solving a Laplacian system corresponds to solving an electrical flow problem. Specifically, for a vector $b \perp \mathbf{1}$ of injected currents, $L^+b \in \mathbf{R}^n$ gives the induced potential differences at the vertices, and $BL^+b \in \mathbf{R}^m$ gives the induced potential differences across the edges.

2.3 Nesterov's Gradient Method

Consider the unconstrained minimization problem

$$\min_{y \in \mathbf{R}^m} f(y)$$

where f is convex and differentiable. Such problems can be solved to arbitrary precision in polynomial time by first-order methods; the convergence rates of these methods can be quantified in terms of the initial distance to the optimum, the *Lipschitz constant of the gradient* ∇f and the *strong convexity parameter*. The Lipschitz constant of the gradient ∇f is defined as the least c_L for which

$$\|\nabla f(y) - \nabla f(z)\|_2 \le c_L \|y - z\|_2 \quad \forall y, z \in \mathbf{R}^m.$$

The strong convexity parameter is defined as the largest c_{sc} for which

$$\langle \nabla f(y) - \nabla f(z), y - z \rangle \ge \frac{c_{sc}}{2} \|y - z\|_2^2 \quad \forall y, z \in \mathbf{R}^m.$$

The simplest first-order scheme is the *gradient descent* method, which is defined by the fixed step size iteration

$$y_{k+1} = y_k - \frac{1}{c_L} \nabla f(y_k).$$

Let y^* be an optimal point. Then it is well known that given a starting point y_0, a point y_T with error $f(y_T) - f(y^*) \le \epsilon$ is obtained in at most $T = \frac{2c_L \|y_0 - y_*\|_2^2}{\epsilon}$ iterations. A remarkable thirty year old result of Yu. Nesterov shows that it is possible to obtain a much faster rate of convergence by taking the last *two* iterates into account. We encapsulate Nesterov's method and its convergence rate below; as is generally the case with gradient methods, the rate of convergence is logarithmic when f is strongly convex, i.e., $c_{sc} > 0$.

THEOREM 2 (NESTEROV). *Let $f : \mathbf{R}^m \to \mathbf{R}$ be convex with c_L-Lipschitz gradient and strong convexity parameter $c_{sc} \ge 0$, and let $y^* \in \mathbf{R}^m$ be any vector. Given an initial vector $y_0 \in \mathbf{R}^m$, the algorithm* Nesterov *produces in*

$$T = \min\left(2\sqrt{\frac{c_L}{\epsilon}} \cdot \|y_0 - y^*\|_2, \sqrt{\frac{c_L}{c_{sc}}} \ln\left(\frac{4\|y_0 - y^*\|_2}{\varepsilon}\right)\right) \tag{2}$$

iterations a vector y_T with objective

$$f(y_T) - f(y^*) \le \epsilon. \tag{3}$$

Each iteration requires a single gradient evaluation $\nabla f(y)$.

Algorithm Nesterov
Input: gradient oracle $\nabla f(y)$, Lipschitz constant c_L of ∇f, strong convexity parameter c_{sc}, starting point y_0, iteration count T

Set $z_1 = y_0, a_1 = 1$. For $k = 1 \ldots T$:

- Set $y_k = z_k - \frac{1}{c_L} \nabla f(z_k)$ (descent step).
- If $c_{sc} = 0$, Then set $a_{k+1} = (1 + \sqrt{4a_k^2 + 1})/2$ and
  $z_{k+1} = y_k + \frac{a_k - 1}{a_{k+1}}(y_k - y_{k-1})$ (momentum step).
- Otherwise, set
  $z_{k+1} = y_k + \frac{\sqrt{c_L} - \sqrt{c_{sc}}}{\sqrt{c_L} + \sqrt{c_{sc}}}(y_k - y_{k-1})$ (momentum step).

Output y_T.

REMARK 1. *Unravelling the two-term recurrence which defines z_{k+1} reveals that in each momentum step the algorithm moves in the direction of a certain weighted average of the last few gradients. This has the effect of cancelling the characteristic 'zig zagging' behavior which makes gradient descent slow. The use of the last two iterates and the $\sqrt{\cdot}$ improvement over gradient descent are reminiscent of the Chebyshev method for solving linear equations; indeed, an appropriate analysis [Pol87] of* Nesterov *can be used to recover the best known bound of $\sqrt{\lambda_{max}(A)/\lambda_{min}(A)}\log(1/\epsilon)$ iterations for solving a linear system $Ax = b$.*

In the present paper we will be interested in minimizing convex functions over affine subspaces which correspond to cuts and flows in a graph, specifically in problems of the form

$$\min_{y\in\mathcal{S}} f(y)$$

where f is convex and $\mathcal{S} = \{y : Py = b\}$ is an affine subspace for some projection P. Since $\mathcal{S}$ is itself a copy of $\mathbf{R}^{\mathrm{rank}(P)}$, such problems can easily be reduced to the unconstrainted case due to the following well known fact.

FACT 1. *The restriction $f_{\mathcal{S}} : \mathcal{S} \to \mathbf{R}$ of f to $\mathcal{S} = \{y : Px = b\}$ is also convex and its gradients are given by*

$$\nabla f_{\mathcal{S}}(y) = P^{\perp}\nabla f(y) \qquad \text{for } y \in \mathcal{S},$$

where $P^{\perp}$ is the orthogonal projection with $P + P^{\perp} = I$.

Thus we can reduce constrained minimization of f over $\mathcal{S} \subset \mathbf{R}^n$ to unconstrained minimization of $f_{\mathcal{S}}$ over $R^{\mathrm{rank}(P)} \cong \mathcal{S}$, and the convergence properties of Nesterov continue to hold in the affine-constrained setting as long as we use the projected gradients $\nabla f_{\mathcal{S}}(y) = P^{\perp}\nabla f(x)$.

COROLLARY 2. *Let $\mathcal{S} = \{y : Py = b\} \subset \mathbf{R}^m$ be an affine subspace for some projection $P : \mathbf{R}^m \to \mathbf{R}^m$, let $f : \mathbf{R}^m \to \mathbf{R}$ be convex, and let $y^* \in \mathcal{S}$. Given a starting point $y_0 \in \mathcal{S}$ and gradient oracle $P^{\perp}\nabla f(y)$,* Nesterov *produces a vector $y_T \in \mathcal{S}$ satisfying $f(y_T) - f(y^*) \le \epsilon$ in*

$$T = \min\left(2\sqrt{\frac{c_L}{\epsilon}} \cdot \|y_0 - y^*\|_2, \sqrt{\frac{c_L}{c_{sc}}}\ln\left(\frac{4\|y_0 - y^*\|_2}{\varepsilon}\right)\right)$$

iterations, where c_L is the Lipschitz constant of $P^{\perp}f(y)$ and c_{sc} is the strong convexity parameter of $P^{\perp}f(y)$.

Next, we recall a result of d'Aspremont which states that Nesterov works essentially just as well if we use approximate gradients rather than exact ones. This will be necessary in order to handle the error introduced by the Laplacian solver.

THEOREM 3 (D'ASPREMONT [D'A08], THEOREM 2.2).[1] *Suppose all the iterates generated by* Nesterov *are guaranteed to lie in a Euclidean ball of radius R. If we replace the exact gradient oracle ∇f by an approximate gradient oracle $\tilde{\nabla} f$ satisfying:*

$$\|\nabla f(y) - \tilde{\nabla} f(y)\|_2 \le \frac{\delta}{3R},$$

then the last iterate y_T with T as in (2) has objective value bounded by:

$$f(y_T) - f(y^*) \le \epsilon + \delta.$$

Finally, we record some standard facts about projections onto convex sets, which will be useful in reasoning about the functions we will minimize.

$$\mathrm{Proj}_K(x) = \mathrm{argmin}_{y\in K}\|x - y\|_2.$$

LEMMA 1 (PROJECTIONS ONTO CONVEX SETS). *Let K be convex. Then (i) Proj_K is $1-$Lipschitz, i.e.,*

$$\|\mathrm{Proj}_K(x) - \mathrm{Proj}_K(y)\|_2 \le \|x - y\|_2 \quad \forall x, y \in \mathbf{R}^n.$$

(ii) The function

$$f(x) := \frac{1}{2}\|x - \mathrm{Proj}_K(x)\|_2^2$$

is convex and has $1-$Lipschitz gradient

$$\nabla f(x) = x - \mathrm{Proj}_K(x).$$

3. THE FLOW ALGORITHM

In this section we present our $\frac{1}{\epsilon}\sqrt{\frac{m}{F}}$ iteration algorithm for approximately maximum st-flow. The algorithm has two phases: a numerical phase which uses Nesterov to obtain a flow with small total overflow, and a combinatorial phase which removes this overflow by subtracting congested flow-paths using a fast implementation of depth first search. The numerical phase begins by computing the $s-t$ electrical flow of value F, and its gradient steps correspond to rerouting overflow on congested edges using local (non $s-t$) electrical flows, which are computed in nearly linear time using LSolve. Crucially, we only run Nesterov until the total overflow is about ϵF, so that the number of iterations required varies inversely with F – this gives the $\sqrt{\frac{m}{F}}$ dependence advertised above. When the overflow is subtracted out in the second phase (whose running time is nearly linear, independent of F) we are left with a flow of value $F - \epsilon F = (1-\epsilon)F$, as desired.

We now describe the convex program used in the first phase. Given a source and sink $s, t \in V$, the set of st-flows of value F is an affine subspace of $\mathbf{R}^m$ which may be written succinctly as:

$$\mathfrak{F}_{st,F} := \{y : B^T y = F(\chi_s - \chi_t)\},$$

indicating excesses of F and $-F$ at s and t and flow conservation at all other vertices. Observe that since BL^+ is invertible on $\mathrm{im}(B^T)$, we have

$$\begin{aligned} B^T y = F(\chi_s - \chi_t) &\iff BL^+B^T y = F \cdot BL^+(\chi_s - \chi_t) \\ &\iff \Pi y = F y_{st}, \end{aligned}$$

where we define

$$y_{st} := BL^+(\chi_s - \chi_t)$$

to be the standard electrical flow between s and t with one unit of current. Thus we may write

$$\mathfrak{F}_{st,F} = \{y : \Pi y = F y_{st}\} = F y_{st} + \{y : \Pi y = 0\},$$

revealing that $\mathfrak{F}_{st,F}$ is a translation of the cycle space $\{y : \Pi y = 0\}$ by the electrical flow $F y_{st}$, which thus is the minimum length vector (and therefore minimum energy flow) in $\mathfrak{F}_{st,F}$:

$$F y_{st} = \mathrm{Proj}_{\mathfrak{F}_{st,F}}(0). \tag{4}$$

[1] The original statment of the theorem is adapted to suit our setup. In particular: (i) The ball of radius R here corresponds to the compact feasible set Q in [d'A08]. (ii) We have presented Nesterov using a two term recurrence, whereas in [d'A08] it is written as an unravelled sum over previous gradients. These formulations are equivalent; see [Nes05] for details.

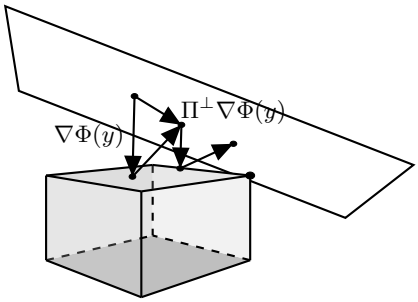

Figure 1: In each step, the point is alternately projected onto the ℓ_∞ ball and then back into the cycle space. This is equal to the gradient in $\mathfrak{F}_{st,F}$.

Finding a feasible flow is equivalent to finding a point in $\mathfrak{F}_{st,F} \cap B^m_\infty$. Such a point may be obtained by minimizing $\|y\|_\infty$ inside $\mathfrak{F}_{st,F}$; this is the standard linear program for max flow, which cannot be solved by using gradients since $\|\cdot\|_\infty$ is not differentiable. Thus, we choose instead to minimize the squared Euclidean distance to B^m_∞:

$$\Phi(y) := \frac{1}{2}\|y - \mathrm{Proj}_{B^m_\infty}(y)\|_2^2 \tag{5}$$

inside $\mathfrak{F}_{st,F}$. Since projection onto B^m_∞ amounts to truncating the coordinates of y to have absolute value at most 1, we have $\Phi(y) = \frac{1}{2}\|\mathsf{over}(y)\|^2$ where $\mathsf{over}(y) := y - \mathrm{Proj}_{B^m_\infty}(y)$ is defined to be the vector of signed overflows on the edges of G.

By Lemma 1, the function Φ is convex with 1−Lipschitz gradient $\nabla\Phi(y) = y - \mathrm{Proj}_{B^m_\infty}(y) = \mathsf{over}(y)$. We will use Nesterov to minimize Φ over $\mathfrak{F}_{st,F}$. Constrained to $\mathfrak{F}_{st,F}$, the projected gradient is given by

$$\nabla\Phi_{\mathfrak{F}_{st,F}}(y) = \Pi^{\perp}\nabla\Phi(y) = (I - BL^{+}B^T)\mathsf{over}(y).$$

Thus each gradient step taken by Nesterov,

$$y - (I - BL^{+}B^T)\mathsf{over}(y) = (y - \mathsf{over}(y)) + BL^{+}B^T\mathsf{over}(y),$$

corresponds cleanly to replacing the overflows $\mathsf{over}(y)$ by the electrical flows obtained by routing the excesses defined by them[2].

After reaching a flow y_T with sufficiently small $\Phi(y_T)$ in the first phase, we use a combinatorial procedure in the second phase to obtain a feasible flow with sufficiently large value. The procedure Drain promised below crucially uses Sleator and Tarjan's *dynamic trees* data structure to quickly remove flowpaths which contribute to the overflow.

LEMMA 2 (OVERFLOW DRAINAGE). *Suppose f is a (not necessarily feasible) st-flow of value F in G. Then there is a feasible st−flow f' of value at least $F - \sum_{e\in E}|\mathsf{over}(y)(e)|$. Moreover, there is an algorithm* Drain *which finds such an f' in time $O(m \log n)$.*

We now state the algorithm in its entirety and prove its correctness.

[2]The projected gradient step inside $\mathfrak{F}_{st,F}$ may also be seen as the projection of the unconstrained step onto $\mathfrak{F}_{st,F}$:

$$y - \Pi^{\perp}\mathrm{Proj}_{B^m_\infty}(y) = \mathrm{Proj}_{\mathfrak{F}_{st,F}}((y - \mathrm{Proj}_{B^m_\infty}(y)),$$

hence the name 'alternating projections'.

Algorithm Maxflow
Input: Graph G, flow value F, error parameter $\epsilon > 0$

1. Compute the st−electrical flow with F units of current
$$y_0 = F \cdot BL^{+}(\chi_s - \chi_t).$$
2. Run Nesterov with starting point y_0, gradients
$$\Pi^{\perp}\nabla\Phi(y) = (I - BL^{+}B^T)(y - \mathrm{Proj}_{B^m_\infty}(y)),$$
Lipschitz constant $c_L = 1$ and strong convexity parameter $c_{sc} = 0$ for $T = \frac{2}{\epsilon}\sqrt{\frac{m}{F}}$ iterations. Let $y_T \in \mathfrak{F}_{st,F}$ be the last point obtained.
3. Apply the Drain procedure from Lemma 2 to the rescaled flow $\frac{y_T}{1+\epsilon}$ to obtain a feasible st-flow f. Output f.

THEOREM 4. *If there is a feasible st-flow of value F, then Maxflow outputs a feasible st−flow f of value at least $(1 - 4\epsilon)F$. Its total running time is at most*

$$O\left(\frac{1}{\epsilon}\sqrt{\frac{m}{F}} \cdot m \log^2 n\right).$$

PROOF. Assume for the moment that the initial electrical flow y_0 and projections $\Pi^{\perp}\Phi(y)$ are computed exactly; we will handle the actual situation with $1/\mathrm{poly}(n)$ error later on.

Let y^* be a feasible flow of value F. The initial point $y_0 = Fy_{st}$ satisfies:

$$\begin{aligned}\|Fy_{st} - y^*\|_2 &= \|\mathrm{Proj}_{\mathfrak{F}_{st,F}}(0) - \mathrm{Proj}_{\mathfrak{F}_{st,F}}(y^*)\|_2 && \text{by (4)} \\ &\le \|0 - y^*\|_2 && \text{by Lemma 1} \\ &\le \sqrt{m}\|y^*\|_\infty && \text{by Cauchy-Schwartz} \\ &\le \sqrt{m} && \text{since } y^* \text{ is feasible.}\end{aligned} \tag{6}$$

Since $\nabla\Phi$ is 1−Lipschitz, the projected gradient $\Pi^{\perp}\nabla\Phi$ inside $\mathfrak{F}_{st,F}$ is also 1−Lipschitz. Therefore by Corollary 2, after $T = \frac{2}{\epsilon\sqrt{F}}\sqrt{m}$ iterations, at the end of step (2) we have an st-flow $y_T \in \mathfrak{F}_{st,F}$ with

$$\begin{aligned}\Phi(y_T) - \Phi(y^*) &= \frac{1}{2}\|y_T - \mathrm{Proj}_{B^m_\infty}(y_T)\|_2^2 - 0 \\ &= \frac{1}{2}\sum_e \mathsf{over}(y_T)(e)^2 \\ &\le \epsilon^2 F.\end{aligned}$$

Setting aside the edges with large overflow, we find that

$$\epsilon \sum_{e:\mathsf{over}(y_T)(e)>\epsilon} |\mathsf{over}(y_T)(e)| \le \sum_e \mathsf{over}(y_T)(e)^2 \le 2\epsilon^2 F,$$

from which we conclude that the total capacity of the set of severely congested edges $D = \{e : \mathsf{over}(y_T)(e) > \epsilon\}$ is at most $2\epsilon F$. This is also true for the rescaled flow $\tilde{y_T} = \frac{y_T}{1+\epsilon}$, which satisfies all capacity constraints outside D. Since it has value at least $\frac{F}{1+\epsilon}$, applying Lemma 2 to $\tilde{y_T}$ yields a feasible st−flow f' of value at least

$$\frac{F}{1+\epsilon} - 2\epsilon F = (1 - 3\epsilon)F,$$

as desired.

The bound on the running time is immediate since each iteration involves one trivial gradient computation $\nabla\Phi(x)$ and one application of $\Pi^{\perp}$ up to $1/\text{poly}(n)$ error, which takes $\tilde{O}(m \log^2 n)$ time by Corollary 1. By Lemma 2 the drainage step also takes nearly linear time.

It remains to show that it is sufficient to compute the starting point y_0 and projections $\Pi^{\perp}\nabla\Phi(y)$ upto $1/\text{poly}(n)$ error. We will achieve this by appealing to Corollary 1 and Theorem 3, which both require an *a priori* bound on the Euclidean norms of the iterates y_k, z_k produced by Nesterov when it is supplied with *exact* y_0 and gradients $\Pi^{\perp}\nabla\Phi(y)$.

We proceed to derive such a bound. Applying (2) and (3) from Theorem 2, we find that the iterates y_k satisfy

$$\begin{aligned}\frac{1}{2}\|y_k - \text{Proj}_{B_\infty^m}(y_k)\|_2^2 &= \Phi(y_k) - \Phi(y^*) \\ &\leq \frac{2\|y_0 - y^*\|_2^2}{k^2} \\ &\leq 2\|Fy_{st} - y^*\|_2^2 \\ &\leq 2m \qquad \text{from (6).}\end{aligned}$$

By the triangle inequality this implies that

$$\|y_k\|_2 \leq 2m + \sqrt{2}\|\text{Proj}_{B_\infty^m}(y_k)\|_2 = O(\sqrt{m}).$$

Applying the triangle inequality to the definition of the auxilliary iterates z_k, we obtain

$$\begin{aligned}\|z_k\|_2 &\leq \left(1 + \frac{a_k - 1}{a_{k+1}}\right)\|y_k\|_2 + \|y_{k-1}\|_2 \\ &\leq 2\|y_k\|_2 + \|y_k\|_2 = O(\sqrt{m}).\end{aligned}$$

Thus all vectors produced have length at mostpoly(n).

We can thus use the procedure of Corollary 1 to compute projections, guaranteeing that $y_0 \in \mathfrak{F}_{st,F}$ and that all the approximate $\Pi^{\perp}\nabla\Phi(y)$ lie exactly in $\text{im}(\Pi^{\perp})$. This in turn means that all iterates produced during the course of the algorithm are exactly inside $\mathfrak{F}_{st,F}$. We can now appeal to Corollary 2 and take $R = O(\sqrt{m})$ in Theorem 3, which tells us that computing the projected gradients $\Pi^{\perp}\nabla\Phi(y)$ upto $1/\text{poly}(n)$ error adds at most $1/\text{poly}(n)$ to our final error, yielding the final bound of $(1 - 4\epsilon)F$, as advertised. □

We now furnish the details of the Drain procedure used to remove the overflows. We remark that this procedure works in exactly the same way for capacitated graphs, but we have chosen to present the unit capacity version for clarity.

Proof of Lemma 2. Orient all the edges in G in the direction of flow to obtain a directed graph. Let $D = \{(s_1, t_1), \ldots, (s_k, t_k)\}$ be the set of congested edges. Create a new distinguished vertex d, and for each $(s_i, t_i) \in D$ add a path of length 2 consisting of two edges $(s_i, d), (d, t_i)$ of capacity $\mathsf{over}(f)(s_i, t_i))$ passing through d. Call this augmented graph $\tilde{G}$, and define a modified $st-$flow $\tilde{f}$ in $\tilde{G}$ in which the overflow on each (s_i, t_i) is rerouted through the newly added path, so that the original edges in G are no longer congested.

We will now drain the total flow from s to d and d to t, yielding a feasible flow f' in G. This is very easy when f is integral: simply trace edges backwards from d to s and forwards from d to t to obtain an $st-$flow path, subtract it from f, and repeat until d is isolated. Since no edge is used by more than one path the total running time is linear.

In our setting the flow f is fractional. However, the above idea can be implemented almost as quickly using Sleator and Tarjan's dynamic trees data structure, described below.

Dynamic Trees (Sleator-Tarjan [ST83]). There is a data structure which maintains set of *disjoint rooted real edge-weighted trees* on a vertex set V, initially a singleton tree for every vertex, and supports the following operations in $O(\log n)$ time:

- $\mathsf{root}(v)$. Returns the root of the tree containing v.
- $\mathsf{link}(v, u, c)$. Makes v the child of u, connected by an edge of weight c.
- $\mathsf{cut}(v)$. Cuts the edge between v and its parent, creating a new tree rooted at v.
- $\mathsf{mincost}(v)$. Returns the last edge of minimum cost on the path $v \longrightarrow \mathsf{root}(v)$.
- $\mathsf{update}(v, c)$ Adds c (which may be negative) to every edge on the path $v \longrightarrow \mathsf{root}(v)$.

We will use this to perform a sped-up depth first search to find and remove flow paths from d to t and update the flow accordingly while maintaining conservation. Initialize the data structure with single-vertex trees for every $v \in V$. Maintain a vector $r \in \mathbf{R}^m$ of residual flows $r(e)$ for every edge $e \in E$, initially $r = f$. Repeat the following loop starting with $v = d$:

Let $w = \mathsf{root}(v)$.

- If $w = t$, then let $q = \mathsf{mincost}(d)$ be the child of the smallest edge on the path $d \longrightarrow t$. Now $\mathsf{update}(d, -r(q))$, $\mathsf{cut}(q)$, and set $v = q$, charging the cost of all of these operations to the deleted edge.
- If $w \neq t$ then find an outgoing edge (w, z) in $\tilde{G}$ with positive residual flow $r(w, z) > 0$. If no such edge exists and $w = d$ then terminate. Otherwise such an edge must exist since we have maintained a nonzero flow path from d to w and flow conservation is satisfied at all vertices on this path. In this case, $\mathsf{link}(w, z, r(w, z))$, set $v = z$, and charge the cost of all operations, including the $\mathsf{root}(v)$, to the newly linked edge.

It is easy to check that the above correctly implements depth first search and that the update step preserves conservation and capacity constraints at all nodes other than d and t, since it subtracts entire $d \longrightarrow t$ flowpaths. The total number of iterations of the loop is at most $2m$ since each iteration either links or cuts an edge and every edge can be linked at most once and cut at most once. Thus the total running time is $O(m \log n)$.

After draining the flow from d to t we repeat the same procedure with edges reversed from d to s. The final result is a flow which respects all capacities in G. The total amount of flow removed is exactly

$$\sum_{i \leq k} \tilde{f}(s_i, d) = \sum_{i \leq k} \mathsf{over}(f)(s_i, t_i)$$

so the output f' has value at least

$$F - \sum_{i \leq k} \mathsf{over}(f)(s_i, t_i),$$

as desired. □

To obtain the worst-case running time of $\tilde{O}(mn^{1/3}/\epsilon^{2/3})$, we trade off the above procedure against the Karger-Levine

[KL02] algorithm which takes time $\tilde{O}(m+nF)$ on uncapacitated graphs: we simply use that algorithm for small flow values $F \leq m/(n\epsilon)^{2/3}$, and our algorithm Maxflow otherwise.

REMARK 2. *It is also possible to obtain the results in this section by minimizing over $\mathfrak{F}_{st,F}$ a certain smoothing of the following 'relaxed' ℓ_∞ norm:*

$$\|y\|_{\infty,F} = \max_{z \in B_1^m \cap \frac{1}{F} B_\infty^m} \langle z, y \rangle.$$

REMARK 3. *The following example shows that the $\sqrt{\frac{m}{F}}$ dependence in Theorem 4 is tight for* Maxflow. *Let s and t be connected by $F/2$ single edges and $F/2$ parallel paths of length $2m/F$, for a total of $O(m)$ edges. As the ratio of the resistance of a long path to that of a short (single edge) path is $m/F : 1$, the initial electrical flow of value F sends about $2F/m$ flow on each long path and $2(1-F/m)$ flow on each short path, causing overflows of $1-2F/m$ on the short edges. In each gradient step, rerouting this overflow amounts to computing another st-electrical flow of value $F/2(1-2F/m)$, which again puts $O(F/m)$ flow on the long paths. Thus, in order to route $\Omega(1)$ flow along the long paths we need $\Omega(m/F)$ rerouting steps; this becomes $\sqrt{m/F}$ when we factor in the the momentum steps taken by* Nesterov.

REMARK 4. *Essentially the same proof shows that a variant of the above algorithm computes a $(1-\epsilon)$-approximately maximum flow in a* capacitated *graph in $O(\frac{1}{\epsilon}\sqrt{\frac{W}{F}})$ iterations, where W is the total capacity.*

4. THE EXACT FLOW ALGORITHM

In this section, we discuss how to obtain an exact flow algorithm from our approximate flow algorithm. The idea is to use Maxflow to find a $(1-\epsilon)-$approximate flow, round this fractional flow into an integral flow, and apply the Ford-Fulkerson algorithm [FF56] to route the remaining ϵF flow. We prove below that can round the fractional flow in $\tilde{O}(m)$; assuming this result, the total running time is given by

$$\tilde{O}\left(\frac{m}{\epsilon}\sqrt{\frac{m}{F}} + \epsilon m F\right) = \tilde{O}\left(m^{5/4}F^{1/4}\right), \quad \text{for } \epsilon = \frac{m^{1/4}}{F^{3/4}}.$$

The following rounding algorithm and its proof are due to Lau and Kwok (personal communication).

THEOREM 5. *Given a fractional st-flow f with flow value F on an integer capacitated graph, an integral st-flow with flow value $\lfloor F \rfloor$ can be found in $O(m \log m)$ time.*

PROOF (SKETCH). This algorithm is inspired by [GKK10]. In this algorithm, we keep a partial integral solution and a new directed graph. Initially, the partial integral solution is empty and the directed graph $\tilde{G}$ is created by adding a new vertex $\tilde{s}$ to the original graph. For any edge e in the original graph, we set the weight on $\tilde{G}$ to be $f(e)$. Add directed edges $t\tilde{s}$ and $\tilde{s}s$ to $\tilde{G}$ with both weights F. Note that all edges in $\tilde{G}$ are directed because the flow sends in only one direction on any edge. Our target is to find a cycle in $\tilde{G}$ from $\tilde{s}$ to $\tilde{s}$ using the random walk according to the weights. After we pick a path P from $\tilde{s}$ to $\tilde{s}$ using random walk, we add it to our partial integral solution as an augmenting path. Then, we reduce the edge weight along the path P by 1. If the edge weight is negative, we reverse its direction and the sign of the weight. Initially, the indegree and the outdegree on every edge are same because the fractional flow f satisfies the flow conservation law. And the reweighting process maintains this property. Also, the indegree of $\tilde{s}$ is $F-k+1$ at step k. Following the same analysis of the random walk in [GKK10], the hitting time from $\tilde{s}$ to $\tilde{s}$ is at least $m/(F-k+1)$ at step k. Hence, this algorithm finds a integral flow with flow value $\lfloor F \rfloor$ in at most $m(1+1/2+\cdots+1/m) = O(m \log m)$ time. □

5. THE CUT ALGORITHM

In this section we present a $\frac{1}{\epsilon}\sqrt{\frac{m}{F}}$ iteration algorithm for finding an approximately minimum st-cut. The algorithm is even simpler than Maxflow, and consists of approximately minimizing a smoothed version of the ℓ_1 norm over the affine subspace of $\mathbf{R}^m$ spanned by the incidence vectors of $st-$cuts. The vector of potential differences thus obtained yields to an embedding of the vertices into $\mathbf{R}$, which is then swept in the standard way to obtain a cut of small value.

We begin by writing the standard linear program for minimum $st-$cut as ℓ_1 minimization over a subspace of $\mathbf{R}^m$:

$$\begin{aligned}
&\min_{x: x_s - x_t = 1} \sum_{ij \in E} |x_i - x_j| \qquad (LP)\\
&= \min_{x: \langle x, \chi_s - \chi_t \rangle = 1} \|Bx\|_1\\
&= \min_{x: \langle x, B^T B L^+ (\chi_s - \chi_t)\rangle = 1} \|Bx\|_1\\
&\qquad \text{since } \chi_s - \chi_t \in \mathrm{im}(B^T) \qquad (7)\\
&= \min_{x: \langle Bx, BL^+(\chi_s - \chi_t)\rangle = 1} \|Bx\|_1\\
&= \min_{y \in \mathfrak{C}_{st}} \|y\|_1
\end{aligned}$$

where

$$\mathfrak{C}_{st} := \mathrm{im}(B) \cap \{\langle y, y_{st}\rangle = 1\}$$

is the $n-2$ dimensional affine subspace of $\mathbf{R}^m$ spanned by the incidence vectors $b_S = B\chi_S$ of cuts $S \subset V$ separating s and t. It will be convenient to write $\mathfrak{C}_{st}$ in the standard form $Px = b$ for

$$\text{projection } P = \Pi^\perp + \frac{y_{st} y_{st}^T}{\|y_{st}\|^2} \quad \text{and} \quad b = \frac{y_{st}}{\|y_{st}\|^2},$$

which also tells us that

$$\frac{y_{st}}{\|y_{st}\|^2} = \mathrm{Proj}_{\mathfrak{C}_{st}}(0). \qquad (8)$$

As before, the ℓ_1 objective is non-differentiable, so we consider the smoothed optimization problem

$$\min_{y \in \mathfrak{C}_{st}} \ell_\mu(y)$$

where ℓ_μ is the following smoothing of the ℓ_1 norm:

$$\ell_\mu(y) := \sum_{e \in E} \sqrt{y^2(e) + \mu^2}.$$

We choose this particular smoothing of ℓ_1 because it is simple to manipulate and because it is easy to see that

$$\frac{\partial}{\partial y}\sqrt{y^2+\mu^2} = \frac{y}{\sqrt{y^2+\mu^2}} = \left(1+\frac{\mu^2}{y^2}\right)^{-1/2} \qquad (9)$$

$$\text{and} \quad \frac{\partial}{\partial y}\left(1+\frac{\mu^2}{y^2}\right)^{-1/2} = \frac{\mu^2}{(y^2+\mu^2)^{3/2}}$$

whence the gradient $\nabla \ell_\mu(y)$ can be computed in $O(m)$ time and has Lipschitz constant $c_L = 1/\mu$.

We are now in a position to present our algorithm.

Algorithm Mincut
Input: Graph G, flow value F, error $\epsilon > 0$.

1. Compute the $st-$electrical flow with unit potential difference between s and t:
$$y_0 = \frac{y_{st}}{\|y_{st}\|_2^2} = \frac{BL^+(\chi_s - \chi_t)}{\|BL^+(\chi_s - \chi_t)\|^2}.$$
2. Set $\mu = \epsilon F/m$. Run Nesterov with starting point y_0, Lipschitz constant $c_L = 1/\mu$, strong convexity parameter $c_{sc} = 0$, and gradients
$$P^\perp \nabla \ell_\mu(y) = \left(\Pi - \frac{y_{st} y_{st}^T}{\|y_{st}\|_2^2}\right) \nabla \ell_\mu(y)$$
for $T = \frac{4}{\epsilon}\sqrt{\frac{2m}{F}}$ iterations. Let y_T be the last iterate.
3. Let $x_T = L^+ B^T y_T \in \mathbf{R}^n$ be the embedding of the vertices corresponding to y_T. Output the best sweep cut
$$S = \{i : x_T(i) \geq \rho\}$$
over all thresholds $\rho \in (x_T(s), x_T(t))$.

THEOREM 6. *If G has an $st-$cut of value at most F, then* Mincut *outputs a cut of value at most $(1+\epsilon)F$. Moreover, its running time is bounded by*
$$O\left(\frac{1}{\epsilon}\sqrt{\frac{m}{F}} \cdot m \log^2 n\right).$$

PROOF. Suppose there is a minimum cut S of value F. The signed incidence vector $y^* = B\chi_S$ of this cut lies in $\mathfrak{C}_{st}$ and has
$$\ell_\mu(y^*) \leq \|y^*\|_1 + \mu m = F + \mu m. \tag{10}$$
Moreover, its ℓ_2 distance to the initial point y_0 may be bounded as
$$\begin{aligned}\|y_0 - y^*\|_2 &= \|\mathrm{Proj}_{\mathfrak{C}_{st}}(0) - \mathrm{Proj}_{\mathfrak{C}_{st}}(y^*)\|_2 \\ &\leq \|0 - y^*\|_2 = \sqrt{F},\end{aligned} \tag{11}$$
since y^* has exactly F entries that are all ± 1[3] .

There are two sources of error in our setup: a smoothing error in approximating $\|y\|_1$ by $\ell_\mu(y)$, and an optimization error in attempting to minimize $\ell_\mu(y)$ by a finite number of iterations of Nesterov. In order to obtain a total error of at most ϵF, we may set the smoothing parameter μ according to
$$\mu m = \epsilon F/2 \Rightarrow \mu = \epsilon F/2m,$$
and run Nesterov with target error $\epsilon F/2$. Plugging these parameters and the bound (11) into Corollary 2, we see that the required number of iterations of Nesterov is
$$T = 2\sqrt{\frac{2m}{\epsilon F \cdot \epsilon F/2}} \cdot \sqrt{F} \leq \frac{4}{\epsilon}\sqrt{\frac{2m}{F}},$$
as prescribed in Mincut. After this many iterations, y_T satisfies
$$\begin{aligned}\|y_T\|_1 &\leq \ell_\mu(y_T) \leq \ell_\mu(y^*) + \epsilon F/2 \\ &\leq (1+\epsilon)F \qquad \text{by (10)}.\end{aligned}$$
Since $y_T \in \mathfrak{C}_{st}$ there exists a vector of potentials $x_T \in \mathbf{R}^n$ with $y_T = Bx_T$. Solving this system of linear equations, we see that x_T may be written as
$$x_T = L^+ B^T y.$$
Moreover, we have $x_T(s) - x_T(t) = 1$. If we choose a threshold ρ uniformly at random in $[x_s, x_t]$, then a standard calculation reveals that the expected number of edges cut by the random sweep cut $S = \{i : x_i \geq \rho\}$ is exactly
$$\sum_{ij \in E} |x_T(i) - x_T(j)| = \|Bx\|_1 \leq (1+\epsilon)F,$$
as desired.

Each iteration involves a trivial gradient calculation $\nabla \ell_\mu(y)$, a projection $\Pi^\perp$, and two vector-vector operations. As the projections may be computed approximately upto $1/\mathrm{poly}(n)$ error in time $O(m \log^2 n)$ using Corollary 1, the total running time is as stated.

The argument for why everything works when we use approximate projections and gradients is completely analogous to the one for Maxflow, and we omit it. □

As before, by trading this off against Karger-Levine algorithm, we obtain a worst-case running time of $\tilde{O}(mn^{1/3}/\epsilon^{2/3})$.

REMARK 5. *In the previous proof, we used a crude bound of μm on the smoothing error of ℓ_μ. It is possible to improve this bound by relating it to the $s-t$ distance in G: assign each edge of G a unit cost and let L be the corresponding min-cost $st-$maxflow value. Then, It can be shown that*
$$F - 3\mu L + \mu m \leq \ell_\mu(y) \leq ||y||_1 + \mu m.$$
Using this improved bound, we can obtain a minimum cut / maximum flow algorithm with a running time of $\tilde{O}(\frac{1}{\varepsilon}\sqrt{\frac{L}{F}})$ iterations. The quantity $\frac{L}{F}$ can be understood as the average flow length and is hence always bounded by n. This approach yields a polynomial time algorithm in the general case of graphs with arbitrary capacities, whereas the naïve generalization of our proof gives a bound of $O(\frac{1}{\varepsilon}\sqrt{\frac{W}{F}})$ iterations, where W is the total capacity of G.

REMARK 6. *It is possible to obtain an algorithm with the same running time by minimizing*
$$\psi(y) := \frac{1}{2}\|y - F \cdot \mathrm{Proj}_{F \cdot B_1^m}(y)\|_2^2,$$
instead of $\ell_\mu(y)$ over $\mathfrak{C}_{st}$, where $F \cdot B_1^m$ is the scaled ℓ_1 ball. This leads to an alternating projections scheme similar to Maxflow. *We have chosen to present the smoothing version because this setup can also be used to produce an approximately blocking flow by duality, as discussed in the next section.*

6. THE APPROXIMATE BLOCKING FLOW ALGORITHM

In this section, we present an approximate blocking flow algorithm based on the energy ℓ_μ proposed in the cut algorithm. The algorithm is remarkable in two ways: it obtains

[3]This last bound uses the structure of $\mathfrak{C}_{st}$ and is a major improvement over the trivial inequality $\|y^*\|_2 \leq \|y\|_1 = F$. It is the reason we get a $\sqrt{m/F}$ iteration algorithm rather than a $\sqrt{m}$ iteration one, which is the best one can hope for for minimization over a general affine subspace.

a flow from the dual of a convex program for minimum cut, and it is almost purely numerical — in particular, it has a very simple drainage step which does not require any data structures like the previous algorithm.

Let us begin by studying the properties of the minimizer y_μ of ℓ_μ over $\mathfrak{C}_{st}$. Let $x_\mu = L^+B^Ty_\mu$ be the embedding of vertices corresponding to y_μ. Note that x_μ is a minimizer of

$$\min_{x:\langle x,\chi_s-\chi_t\rangle=1}\sum_{e\in E}\sqrt{(Bx(e))^2+\mu^2}. \tag{12}$$

Taking derivatives as in (9), we find that

$$\nabla\ell_\mu(y_\mu) = B^T\left(\frac{(Bx_\mu)(e)}{\sqrt{(Bx_\mu)(e)^2+\mu^2}}\right)_e = c\cdot(\chi_s-\chi_t)$$

for some constant c. Hence, the vector of rescaled potential differences,

$$f_\mu(e) := \frac{(Bx_\mu)(e)}{\sqrt{(Bx_\mu)(e)^2+\mu^2}}$$

is a flow from s to t. Since x drops 1 unit from source to sink, it must drop non-negligibly on some edges on any path from s to t. Therefore, if μ is sufficient small, the flow f_μ must be almost 1 on some edges on any path from s to t. In other words, any path from s to t has been almost blocked several times.

Thus, solving (12) *exactly* would allow us to exploit first-order optimality conditions and obtain a blocking flow, for sufficiently small μ. It is not possible to do this quickly because ℓ_μ is not strongly convex; however, we can make it strongly convex by adding a quadratic term as follows:

$$\ell_{\mu,\eta}(y) := \ell_\mu(y) + \frac{\eta}{2}\|y\|^2,$$

after which we will have the optimality condition $B^Tf_{\mu,\eta} = c\cdot(\chi_s-\chi_t)$ for

$$f_{\mu,\eta}(e) := f_\mu(e) + \eta y_\mu(e).$$

Moreover, function $\ell_{\mu,\eta}$ has strong convexity parameter $c_{sc} = \eta$ and can therefore be solved to $1/\mathrm{poly}(n)$ accuracy in $O(\sqrt{\frac{1}{\mu\eta}}\mathrm{poly}\log n)$ iterations by Theorem 2.

It turns out that the above is sufficient to obtain an approximately blocking flow. We now state the algorithm and prove its correctness.

Algorithm ApproxBlockingFlow
Input: Graph G, distance k, error δ.

1. Compute the $st-$electrical flow y_0 with unit potential difference between s and t.
2. Set $\mu = \frac{\sqrt{\delta}}{k}$ and $\eta = \frac{\delta}{3}$. Run Nesterov with starting point y_0, Lipschitz constant $c_L = 1/\mu+\eta$, strong convexity parameter $c_{sc} = \eta$ and gradients $P^\perp\nabla\ell_{\mu,\eta}(y)$ for $T = 30\frac{k^{1/2}}{\delta^{3/4}}\log(\frac{m}{\delta})$ iterations. Let y_T be the last iterate.
3. Let
$$z(e) := \frac{y_T(e)}{1+\eta}\left(\frac{1}{\sqrt{y_T(e)^2+\mu^2}}+\eta\right).$$
Take any spanning tree of G and drain the non-terminal excesses of z along it to satisfy conservation. Output this flow.

THEOREM 7. ApproxBlockingFlow *outputs an approximate (k,δ) blocking flow (as defined in section 2.1) with flow value at least $(1-\frac{2\delta}{3})F - \frac{\sqrt{\delta}}{k}m$. Moreover, its running time is bounded by*

$$O\left(m\frac{k^{1/2}}{\delta^{3/4}}\log(\frac{m}{\delta})\log^2 n\right).$$

PROOF. Let $y_{\mu,\eta}$ be the output of Nesterov and $x_{\mu,\eta} = L^+B^Ty_{\mu,\eta}$ be the embedding of vertices corresponding to $y_{\mu,\eta}$. We have chosen T large enough so that

$$\ell_{\mu,\eta}(Bx_{\mu,\eta}) \le \min_{x:x_s-x_t=1}\ell_{\mu,\eta}(Bx) + \frac{\mu\eta^2}{8m^4}. \tag{13}$$

Noting that $c_L = 1/\mu$ for $\ell_{\mu,\eta}$, examining the gradient step $y = x_{\mu,\eta} - \frac{\mu}{4}B^T\ell'_{\mu,\eta}(Bx_{\mu,\eta})$ gives:

$$\ell_{\mu,\eta}(y) \le \ell_{\mu,\eta}(Bx_{\mu,\eta}) - \frac{\mu}{8}||\mathrm{Proj}_{\perp\chi_s-\chi_t)}(B^T\ell'_{\mu,\eta}(Bx_{\mu,\eta}))||^2.$$

Together with the bound (13) , this implies the following ℓ_2 bound on the excess at the vertices:

$$||B^T\ell'_{\mu,\eta}(Bx_{\mu,\eta}) - c\cdot(\chi_s-\chi_t)||^2 \le \frac{\eta^2}{m^4}.$$

Thus, the vector $\ell'_{\mu,\eta}(Bx_{\mu,\eta})$ violates the conservation constraints by at most $\frac{\eta}{m^2}$ at each vertex $i\neq s,t$. It is also easy to check that $\|\ell'_{\mu,\eta}(Bx_{\mu,\eta})\|_\infty \le 1+\eta$. Thus, we can obtain a valid flow z by scaling down by a $1+\eta$ factor to satisfy capacity constraints, and then draining the excesses at $i\neq s,t$ along any spanning tree of G to satisfy conservation.

The flow value of f can be found by considering the amount of flow leaving a random level set $C_t := \{x_{\mu,\eta} > t\}$:

$$\begin{aligned}
&\text{Flow value of } z\\
=\ &\int_0^1\sum_{e\in E(C_t,C_t^c)}z(e)dt\\
\ge\ &(1-\eta)\int_0^1\sum_{e\in E(C_t,C_t^c)}\left(\ell'_{\mu,\eta}(Bx_{\mu,\eta})(e)-\frac{\eta}{m}\right)dt\\
\ge\ &(1-\eta)\cdot\\
&\sum_{e\in E(C_t,C_t^c)}\int_0^1\left(\frac{(Bx_{\mu,\eta})(e)}{\sqrt{(Bx_{\mu,\eta})(e)^2+\mu^2}}+\eta(Bx_{\mu,\eta})(e)\right)dt-\eta\\
=\ &(1-\eta)\sum_{e\in E}\left(\frac{(Bx_{\mu,\eta})(e)^2}{\sqrt{(Bx_{\mu,\eta})(e)^2+\mu^2}}+\eta(Bx_{\mu,\eta})(e)^2\right)-\eta\\
\ge\ &(1-\eta)\sum_{e\in E}\left(\sqrt{(Bx_{\mu,\eta})(e)^2+\mu^2}-\frac{\mu^2}{\sqrt{(Bx_{\mu,\eta})(e)^2+\mu^2}}\right)-\eta\\
\ge\ &(1-\eta)\sum_{e\in E}\sqrt{(Bx_{\mu,\eta})(e)^2+\mu^2}-\mu m-\eta\\
\ge\ &(1-2\eta)F-\mu m.
\end{aligned}$$

where the third line accounts for the effect of modifying the flow.

To prove the blocking flow property, note that

$$\left|\frac{(Bx_{\mu,\eta})(e)}{\sqrt{(Bx_{\mu,\eta}(e))^2+\mu^2}}+\eta(Bx_{\mu,\eta})(e)\right| \ge 1-\eta$$

whenever $|(Bx_{\mu,\eta})(e)| > \frac{\mu}{\sqrt{3\eta}}$. Since the corrected flow z is different from x by at most a $1+2\eta$ factor, we have $|z(e)| \ge 1-3\eta$ as long as $|(Bx_{\mu,\eta})(e)| > \frac{\mu}{\sqrt{3\eta}}$. This means that

if the potential $x_{\mu,\eta}$ drops a little bit on an edge e, the corresponding flow z is almost congested. Consider any path from s to t with length at least $\frac{\sqrt{3\eta}}{\mu}$. Since $x_{\mu,\eta}$ drops 1 unit from s to t, there is at least one edge e which drops more than $\frac{\mu}{\sqrt{3\eta}}$, and hence $|z(e)| \geq 1 - 3\eta$. Since this is true for all paths, z is an approximate $\left(\frac{\sqrt{3\eta}}{\mu}, 3\eta\right)$ blocking flow. The theorem follows by setting $\mu = \frac{\sqrt{\delta}}{k}$, $\eta = \frac{\delta}{3}$ and $T = 30\frac{k^{1/2}}{\delta^{3/4}}\log(\frac{m}{\delta})$. $\square$

REMARK 7. *We can use* ApproxBlockingFlow *to get an approximate blocking flow, round it to an integral flow by Theorem 5, and route the remaining flow in $G_{f,\delta}$ by the Ford-Fulkerson algorithm. This gives a $(1-\delta)$-approximately maximum flow because $G_{f,\delta}$ differs from G only on $(1-\delta)$-congested edges. A simple tradeoff gives a running time of $\tilde{O}(m^{4/3}/\sqrt{\delta})$, which is the fastest approximate maximum flow algorithm for sparse undirected uncapacitated graphs.*

7. DISCUSSION AND CONCLUSION

We conclude by discussing the possibility of speeding up our algorithms by *preconditioning*, by which we mean broadly any change of variables or scaling of the coordinates which speeds up the convergence of Nesterov. This is a common method of speeding up gradient-based iterations, especially for solving linear equations, since gradients are not affine-invariant and their quality can be improved by applying an appropriate linear transformation.

In fact, our algorithm Mincut already uses the change of variables $y = Bx$ in (7) in a crucial way. This transformation provides the improved distance bound of $\|y_0 - y^*\|_2 \leq \sqrt{F}$ and yields a faster rate of convergence than if we just applied Nesterov to minimize (a smoothing of) $\|Bx\|_1$ in the untransformed coordinates $x \in \mathbf{R}^n$. We are hopeful that more general transformations, for instance of type $y = CBx$ for some diagonal scaling matrix C, can be used to obtain even better convergence. As such transformations map the cut space $\mathrm{im}(B)$ to the cut space of a weighted graph, we can still use fast Laplacian solvers to compute projections in the transformed spaces quickly.

Another example of this phenomenon is the following. The initial point y_0 in Maxflow is obtained by projecting 0 onto $\mathfrak{F}_{st,F}$ in the standard ℓ_2 norm. If we use the weighted ℓ_2 norm $\|y\|_C = (\sum_e C(e)y(e)^2)^{1/2}$ for some edge weights $C(e) \geq 0$ instead, then the initial projection

$$y_0 = \mathrm{argmin}_{z \in \mathfrak{F}_{st,F}} \|0 - z\|_C$$

corresponds to the electrical flow in G with conductances of $C(e)\ \Omega^{-1}$ on the edges. It is easy to show that when there is a feasible $st-$flow of value f, then there *exists* a setting of conductances $C(e)$ for which $\|y_0\|_\infty \leq 1$, i.e., the process converges to the optimal point in *one* iteration. This is not very useful, since we do not know of any way to compute $C(e)$ other than solving the original flow problem. However, the multiplicative weights method approach of [CKM+11] may be seen as attempting to find a good $C(e)$ by iteratively reducing the conductance (i.e., increasing the resistance) on edges which are congested by the current flow.

We are optimistic that combining multiplicative update techniques as well as combinatorial insights with the techniques in this paper will yield nontrivial preconditioners and faster algorithms.

8. ACKNOWLEDGEMENTS

Our thanks to Lap Chi Lau and Tsz Chiu Kwok for the suggestion of the rounding algorithm (Theorem 5). We also thank Jittat Fakcharoenphol, Jon Kelner, and Nisheeth Vishnoi for helpful discussions.

9. REFERENCES

[AHK12] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: A meta-algorithm and applications. *Theory OF Computing*, 8:121–164, 2012.

[BI04] D. Bienstock and G. Iyengar. Solving fractional packing problems in o ast $(1/\varepsilon)$ iterations. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 146–155. ACM, 2004.

[CKM+11] P. Christiano, J.A. Kelner, A. Madry, D.A. Spielman, and S.H. Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *STOC 2011*, pages 273–282. ACM, 2011.

[d'A08] A. d'Aspremont. Smooth optimization with approximate gradient. *SIAM Journal on Optimization*, 19(3):1171–1183, 2008.

[DS08] S.I. Daitch and D.A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 451–460. ACM, 2008.

[FF56] L.R. Ford and D.R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956.

[GKK10] A. Goel, M. Kapralov, S. Khanna. Perfect matchings in o (n log n) time in regular bipartite graphs In *Proceedings of the 42th annual ACM symposium on Theory of computing*, pages 39–46. ACM, 2010.

[Gol98] A. Goldberg. Recent developments in maximum flow algorithms. *Algorithm TheoryâĂŤSWAT'98*, pages 1–10, 1998.

[GR98] Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998.

[Kar98] David R. Karger. Better random sampling algorithms for flows in undirected graphs. In *SODA*, pages 490–499, 1998.

[KL02] David R. Karger and Matthew S. Levine. Random sampling in residual graphs. In *STOC 2002*, pages 63–66. ACM, 2002.

[KMP11] I. Koutis, G.L. Miller, and R. Peng. A nearly-m log n time solver for sdd linear systems. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 590–598. IEEE, 2011.

[KY99] P. Klein and N. Young. On the number of iterations for dantzig-wolfe optimization and packing-covering approximation algorithms. *Integer Programming and Combinatorial Optimization*, pages 320–327, 1999.

[Nes05] Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, 2005.

[Pol87] B.T. Poljak. *Introduction to optimization*. Optimization Software, 1987.

[ST83] D.D. Sleator and R. Endre Tarjan. A data structure for dynamic trees. *Journal of computer and system sciences*, 26(3):362–391, 1983.

[Vis] Nisheeth Vishnoi. $Lx = b$, book manuscript.