Proc. Int. Cong. of Math. – 2018
Rio de Janeiro, Vol. 3 (3345–3372)

GRADIENTS AND FLOWS: CONTINUOUS OPTIMIZATION APPROACHES TO THE MAXIMUM FLOW PROBLEM

Aleksander Mądry

Abstract

We use the lens of the maximum flow problem, one of the most fundamental problems in algorithmic graph theory, to describe a new framework for design of graph algorithms. At a high level, this framework casts the graph problem at hand as a convex optimization task and then applies to it an appropriate method from the continuous optimization toolkit. We survey how this new approach led to the first in decades progress on the maximum flow problem and then briefly sketch the challenges that still remain.

1 Introduction

The maximum flow problem is one of the most fundamental and extensively studied graph problems in combinatorial optimization Schrijver [2003], Ahuja, Magnanti, and Orlin [1993], and Schrijver [2002]. It has a wide range of applications (see Ahuja, Magnanti, Orlin, and Reddy [1995]), is often used as subroutine in other algorithms (see, e.g., Arora, Hazan, and Kale [2012] and Sherman [2009]), and a number of other important problems – e.g., the minimum s-t cut problem and the bipartite matching problem Cormen, Leiserson, Rivest, and C. Stein [2009] – can be reduced to it. Furthermore, this problem was often a testbed for development of fundamental algorithmic tools and concepts. Most prominently, the max-flow min-cut theorem Elias, Feinstein, and Shannon [1956] and Ford and Fulkerson [1956] constitutes a prototypical primal-dual relation.

Several decades of work resulted in a number of developments on fast algorithms for the maximum flow problem (see Goldberg and S. Rao [1998] for an overview) and many of this problem's generalizations and special cases. The algorithms underlying these developments tended to be combinatorial in spirit. That is, they operated on various combinatorial notions associated with a graph, such as paths, cuts, trees, and partitions, and then used sophisticated data structures to make these operations efficient. Employing this

MSC2010: primary 68Q25; secondary 05C85, 68W40, 65K10.

kind of approaches is fairly natural in this context – after all, graphs are combinatorial objects – and it was very successful too. The resulting techniques were shaping much of our understanding of not only graph algorithms but also algorithms at large.

Still, despite all this effort and successes, the basic problem of computing a maximum s-t flow in general graphs resisted progress for a long time. The best known combinatorial algorithm for that problem runs in time $O(m \min\{m^{\frac{1}{2}}, n^{\frac{2}{3}}\} \log(n^2/m) \log U)$ and it was developed already 20 years ago by Goldberg and S. Rao [1998]. In fact, this running time bound, in turn, matches the $O(m \min\{m^{\frac{1}{2}}, n^{\frac{2}{3}}\})$ bound that Even and Tarjan [1975] – and, independently, Karzanov [1973] – established for unit-capacity graphs almost 40 years ago.[1] It is thus evident that such purely combinatorial techniques have certain fundamental limitations. Consequently, there is a need for development of a different, broader perspective on graph algorithms.

In this survey, we describe such a new perspective. In a sense, this new view can be seen as a more general form of the continuous approaches to understanding graphs that were developed in the context of spectral graph theory Chung [1997]. At a high level, it relies on casting the graph problem at hand as an optimization task that is continuous and then applying to it an appropriate tool from continuous optimization, a field that aims to design efficient algorithms for finding (approximate) minimum of a given (continuous) function over a continuous domain.

Over the last decade this new approach enabled us to make first in decades progress on a number of fundamental graph problems. Most prominently, it provided us with algorithms for the ε-approximate undirected maximum flow problem Christiano, J. Kelner, Mądry, D. Spielman, and Teng [2011], Lee, S. Rao, and Srivastava [2013], Sherman [2013], J. Kelner, Lee, Orecchia, and Sidford [2014], and Sherman [2017a] and the exact, directed maximum flow problem Mądry [2013], Lee and Sidford [2014], and Mądry [2016] that finally improve over the classic bounds due to Even and Tarjan [1975] and Karzanov [1973] as well as Goldberg and S. Rao [1998].

The goal of this exposition is to present a unified view on these developments. In particular, we aim to directly connect the maximum flow algorithms that have been proposed in this context to the underlying methods and notions from the field of continuous optimization. It turns out that this "tale of one problem" enables us to survey a large part of continuous optimization's landscape. Specifically, along the way, we discuss almost all of the most fundamental tools and concepts of that field, such as different variants of the gradient descent method, and the Newton's method. This shows that the maximum flow

[1] Here, m denotes the number of edges, n – the number of vertices, and U is the largest (integer) edge capacity.

problem might end up becoming a fertile testbed for development of new continuous optimization methods, and thus play a role that is similar to the one it has played already in the context of combinatorial methods.

1.1 Organization of the Paper. We begin the technical part of the paper in Section 2 by introducing the basic notions that will be needed in our discussion. Then, in Section 3, we provide a brief overview of basic continuous optimization tools. In Section 4, we show how these tools can be used to obtain fast ε-approximate algorithms for the maximum flow problem in undirected graphs. Next, in Section 5, we describe continuous optimization–based approaches to computing an exact maximum flow in directed graphs. We conclude in Section 6 with a discussion of some of the key challenges that the future work in this area might be able to address.

2 Preliminaries

We will be viewing graphs as having both lower and upper capacities. Specifically, we will denote by $G = (V, E, u)$ a directed graph with a vertex set V, an edge set E, and two (non-negative) integer capacities u_e^- and u_e^+, for each edge $e \in E$. (The role of these capacities is described below.) Usually, m will denote the number $|E|$ of edges of the graph in question, $n = |V|$ the number of its vertices, and U the largest edge capacity. We view each edge e of G as having an orientation (u, v), where u is its *tail* and v is its *head*.

Maximum Flow Problem. Given a graph G, we view a flow in G as a vector $f \in \mathbb{R}^m$ that assigns a value f_e to each edge e of G. When f_e is non-negative (resp. negative) we interpret it as having a flow of $|f_e|$ flowing in (resp. opposite to) the direction of the edge e orientation.

We say that a flow f is *valid* for some *demands* $\sigma \in \mathbb{R}^n$ iff it satisfies *flow conservation constraints* with respect to that demands. That is, we have that

$$\sum_{e \in E^+(v)} f_e - \sum_{e \in E^-(v)} f_e = \sigma_v, \quad \text{for each vertex } v \in V. \tag{1}$$

Here, $E^+(v)$ (resp. $E^-(v)$) is the set of edges of G that are oriented towards (resp. out of) vertex v. Intuitively, these constraints enforce that the net balance of the total in-flow into vertex v and the total out-flow out of that vertex is equal to σ_v, for every $v \in V$. (Observe that this implies, in particular, that $\sum_v \sigma_v = 0$.)

Now, we say that a flow f is *feasible* in G iff f obeys the *the capacity constraints*:

$$-u_e^- \leq f_e \leq u_e^+, \quad \text{for each arc } e \in E. \tag{2}$$

In other words, we want each arc e to have a flow that is at most u_e^+ if it flows in the direction of e's orientation (i.e., $f_e \geq 0$), and at most u_e^-, if it flows in the opposite direction (i.e., $f_e < 0$). Note that orienting the edges accordingly and setting all u_e^-s be equal to zero recovers the standard notion of flows in directed graphs. Similarly, setting $u_e^- = u_e^+$ for each edge e corresponds to the setting of undirected flows.

One type of flows that will be of special interest to us are s-t flows, where s (the *source*) and t (the *sink*) are two distinguish vertices of G. Formally, an s-t *flow* is a σ-flow whose demand vector σ is equal to $F \cdot \chi_{s,t}$, where $F \geq 0$ is called the *value* of f and $\chi_{s,t}$ is a demand vector that has -1 (resp. 1) at the coordinate corresponding to s (resp. t) and zeros everywhere else.

Now, the *maximum flow problem* corresponds to a task in which we are given a graph $G = (V, E, u)$ with integer capacities as well as a source vertex s and a sink vertex t and want to find a *feasible* (in the sense of (2)) s-t flow of maximum value. We will denote this maximum value as F^*.

Vector Norms. We will find it useful to work with various ℓ_p-norms of vectors. To this end, for any $p > 0$, we define the *ℓ_p-norm* $\|h\|_p$ of a vector h as $\|h\|_p := (\sum_i |h_i|^p)^{\frac{1}{p}}$. In particular, the ℓ_∞-norm is given by $\|h\|_\infty := \max_i |h_i|$. Finally, for a given positive definite matrix A, we define $\|h\|_A := \sqrt{h^T A h}$.

3 A Primer on Continuous Optimization

The framework that will be at the center of our considerations is continuous optimization or, more precisely, its part called *convex* optimization. Therefore, in this section, we provide a brief overview of this framework. (For a much more comprehensive treatment of this subject, the reader is referred to Nemirovskii, Yudin, and Dawson [1983], Nesterov [2004], Nocedal and S. Wright [2000], Boyd and Vandenberghe [2004], and Bubeck [2015].) Later, we will discuss how this methodology can be applied to flow problems.

3.1 (Constrained) Minimization Problem. At a high level, one can view continuous optimization as a set of tools designed to solve a single, general task: *(constrained) minimization problem*. In this problem, we are given a continuous *objective function* $g : \mathbb{R}^k \to \mathbb{R}$ and want to solve the following optimization problem.

$$\min_{x \in \mathcal{K}} g(x), \tag{3}$$

where $\mathcal{K} \subseteq \mathbb{R}^k$ is the *feasible set*. In its full generality, the problem (3) is intractable (or even impossible to solve). Therefore, in the context of convex optimization – which is

the context we will focus on here – we assume that both $\mathcal{K}$ and g are convex and that a minimum we intend to find indeed exists. Additionally, whenever we have that $\mathcal{K} = \mathbb{R}^k$, we will call (3) *unconstrained*.

The most popular way of solving problem (3) is to apply an iterative approach to it. Specifically, we start with some initial feasible solution $x^0 \in \mathcal{K}$ and then, repeatedly, given a current solution $x^{t-1} \in \mathcal{K}$, we provide a procedure (*update rule*) that produces a new, improved solution x^t. We require that

$$\lim_{t\to\infty} x^t = x^*,$$

for some optimal solution $x^* \in \mathcal{K}$ to problem (3), and are interested in the rate of this convergence. Specifically, for a given $\varepsilon > 0$, we would like to understand the number T_ε of update steps needed to have that

$$\min_{t=0,\ldots,T_\varepsilon} g(x^t) - g(x^*) \leq \varepsilon. \tag{4}$$

That is, the number of steps needed to guarantee that we find an ε-approximate minimizer of g in $\mathcal{K}$.

Convex optimization has developed a number of different update rules. Each of these rules gives rise to a different algorithm and thus a different type of bound on T_ε. Most of these methods – including each one we will discuss – require making additional assumptions on the function g that go beyond assuming that it is convex. The general principle is that the stronger conditions on g we assume, the better convergence bounds we can obtain.

In what follows, we describe the most basic examples of such algorithms: the *subgradient descent* and the *gradient descent* methods. These methods fall into the broad category of so-called *first-order methods*, i.e., algorithms whose update rules rely only on the local first-order information about the objective function g. Later on, in Section 5, we will also discuss more advanced algorithms.

3.2 Subgradient Descent Method. Recall that a *subgradient* of a function g at a point x is any vector $s \in \mathbb{R}^k$ such that

$$g(x) - g(y) \leq s^T(x - y), \tag{5}$$

for every y. So, in other words, s defines a linear function $g(x) + s^T(y - x)$ that lower-bounds the function g everywhere. We denote by $\partial g(x)$ the set of all subgradients of g at the point x.

Now, the key observation is that if $s \in \partial g(x)$ for some (non-optimal) solution $x \in \mathcal{K}$ and x^* is a minimizer of g in $\mathcal{K}$, then, by (5), it must be the case that

$$0 < g(x) - g(x^*) \leq s^T(x - x^*) = (-s)^T(x^* - x), \tag{6}$$

i.e., the direction in which x^* lies with respect to x is positively correlated, i.e., has a positive inner product, with the direction that is *opposite* to the subgradient direction.

The above observation motivates the following natural update rule

$$x^t \leftarrow x^{t-1} - s, \tag{7}$$

where $s \in \partial g(x^{t-1})$.

This update rule has, however, two important issues. First of all, even if moving in the direction of $-s$ might indeed bring us closer to a minimum x^*, it might make the point x^t lay outside of the feasible set $\mathcal{K}$. We thus need to have a way to map such a point outside of $\mathcal{K}$ back to $\mathcal{K}$ (and do it in a way that does not cancel our progress). To this end, we employ the operation of a *projection* $\Pi_{\mathcal{K}}$ on the set $\mathcal{K}$ defined as

$$\Pi_{\mathcal{K}}(x) = \operatorname{argmin}_{y \in \mathcal{K}} \|y - x\|_2, \tag{8}$$

and project our new solution back on the feasible set $\mathcal{K}$ in each step. A key property of such projection that we use here is its contractivity. In particular, we have that, for any point y,

$$\|\Pi_{\mathcal{K}}(x^*) - \Pi_{\mathcal{K}}(y)\|_2 = \|\Pi_{\mathcal{K}}(x^*) - \Pi_{\mathcal{K}}(y)\|_2 \leq \|\Pi_{\mathcal{K}}(x^*) - y\|_2,$$

where the first equality follows from the fact that $\Pi_{\mathcal{K}}(x^*) = x^*$, as $x^* \in \mathcal{K}$. As a result, projecting a point on $\mathcal{K}$ can only bring it closer to a given minimum x^*.

The second shortcoming of the update rule (7) is related to the fact that it is not clear if fully moving in the direction of $-s$ is not too drastic. After all, the correlation expressed by (6) does not tell us much about how *far* from x the minimum x^* lies. It only informs the direction we should take. Consequently, moving by too much could lead to vast "overshooting" of the minimum x^* and thus lack of convergence.

To cope with this problem we need to make (a fairly minimal) assumption about the objective function g. Namely, we require that g is *L-Lipschitz*, i.e., that

$$|g(x) - g(y)| \leq L\|x - y\|_2, \tag{9}$$

for every x and y and some fixed parameter L, and then we modulate the size of our step appropriately.

Specifically, our final form of subgradient descent method update becomes

$$x^t \leftarrow \Pi_{\mathcal{K}}(x^{t-1} - \eta s), \tag{10}$$

where $s \in \partial g(x^{t-1})$ and $\eta > 0$ is the scalar *step size*.

With this update rule in place, one can establish the following convergence bounds for the resulting algorithm.

Theorem 3.1 (see, e.g., Theorem 3.2 in Bubeck [2015]). *If the objective function g is L-Lipschitz then, for any $\varepsilon > 0$, the update rule (10) with $\eta = \frac{\varepsilon}{L^2}$ delivers an ε-approximate solution after at most*

$$T_\varepsilon \leq L^2 R^2 \varepsilon^{-2}$$

iterations, where $R = \|x^0 - x^*\|$ is the distance of the initial solution x^0 to a minimum x^*.

The above bound is fairly remarkable as it requires very minimal assumptions on the objective function g. In particular, we do not even need g to be differentiable. Still, as we will see shortly, once stronger assumptions on g can be made, we can obtain significantly improved bounds.

3.3 Gradient Descent Method. Arguably, the most well-known algorithm in continuous optimization is gradient descent method. In our setting, this algorithm can be viewed as a variant of the subgradient descent method considered above in the case when the objective function g is differentiable everywhere. In this case, the *gradient* $\nabla g(x)$ of g exists at every point x and, consequently, we have that $\partial g(x) = \{\nabla g(x)\}$ everywhere. That is, the gradients are the (unique) subgradients of g.

The update rule (10) thus becomes

$$x^t \leftarrow \Pi_{\mathcal{K}}(x^{t-1} - \eta \nabla g(x^{t-1})), \tag{11}$$

where, again, $\eta > 0$ is the step size.

Clearly, setting η as in Theorem 3.1 immediately recovers the corresponding bounds. (Note that the Lipschitz constant L (cf. Equation (9)) corresponds to the bound on the norm of the gradient.)

However, one can get an even better bound provided g is not only Lipschitz but also has Lipschitz gradients. That is, the additional assumption to make on g is to require that it is β-*smooth*, which is defined as

$$\|\nabla g(x) - \nabla g(y)\|_2 \leq \beta \|x - y\|_2, \tag{12}$$

for every x and y.

To understand how β-smoothness enables us to get a better control over the progress made by the update step (11), let us assume for the sake of exposition that g is infinitely differentiable. (However, every conclusion that follows holds also without this assumption.) Applying Taylor expansion to g around a given solution x, we obtain that

$$g(x + \Delta) = \underbrace{g(x) + \nabla g(x)^T \Delta}_{\varphi_x(\Delta)} + \underbrace{\frac{1}{2}\Delta^T \nabla^2 g(x) \Delta + \ldots}_{\varrho_x(\Delta)}, \tag{13}$$

where Δ is the step that we intend to take and $\nabla^2 g(x)$ is the *Hessian* of g at x.

One should view this expansion as comprising two terms. A linear (in Δ) term $\varphi_x(\Delta) = g(x) + \nabla g(x)^T \Delta$ that corresponds to a (local) linear approximation of our objevtive function g at the point x, and $\varrho_x(\Delta)$ being the "tail error" of this approximation.

Now, the key point is that the convexity and β-smoothness of g enables us to have a fairly tight control of the tail error term $\varrho_x(\Delta)$. Specifically, we have that

$$0 \leq \varrho_x(\Delta) \leq \frac{\beta}{2}\|\Delta\|_2^2, \tag{14}$$

for all x and Δ. That is, the objective function g can be not only lowerbounded by the linear function $\varphi_x(\Delta)$, as before, but it also can be upperbounded by the same linear function after adding a quadratic term $\frac{\beta}{2}\|\Delta\|_2^2$ to it.

Consequently, instead of trying to choose the step Δ so us to directly minimize g, one can aim to minimize this upperbounding function. More precisely, we can choose Δ so as

$$\Delta_x^* = \operatorname{argmin}_\Delta \left(\varphi_x(\Delta) + \frac{\beta}{2}\|\Delta\|_2^2\right) = -\frac{1}{\beta}\operatorname{argmax}_\Delta \left(\nabla g(x)^T \Delta - \frac{1}{2}\|\Delta\|_2^2\right). \tag{15}$$

An elementary calculation shows that $\Delta_x^* = -\frac{1}{\beta}\nabla g(x)$. This, in turn, corresponds to the update step (11) with the setting of $\eta = \frac{1}{\beta}$.

Indeed, with such a setting of η one obtains the following, improved convergence bound.

Theorem 3.2 (see, e.g., Theorem 3.3 in Bubeck [2015])**.** *If the objective function g is β-smooth then, for any $\varepsilon > 0$, the update rule (11) with $\eta = \frac{1}{\beta}$ delivers an ε-approximate solution to problem (3) after at most*

$$T_\varepsilon \leq O\left(\beta R^2 \varepsilon^{-1}\right)$$

iterations, where $R = \|x^0 - x^*\|_2$ is the distance of the initial solution x^0 to a minimum x^*.

We remark that the update rule (11) and the resulting iteration bound provided above is not optimal. Nesterov [1983, 2005] put forth a much more involved update rule: so-called *accelerated scheme*, that enables one to obtain a significantly improved convergence.

Theorem 3.3 (see, e.g., Theorem 3.19 in Bubeck [2015])**.** *If the objective function g is β-smooth then, for any $\varepsilon > 0$, one can compute an ε-approximate solution to problem (3) after at most*

$$T_\varepsilon \leq O\left(\sqrt{\beta} R \varepsilon^{-\frac{1}{2}}\right)$$

iterations, where $R = \|x^0 - x^*\|_2$ is the distance of the initial solution x^0 to a minimum x^*.

3.4 Gradient Descent Method and Strong Convexity. The bound provided by the gradient descent method (cf. Theorem 3.2) is a clear improvement over the convergence bound delivered by the subgradient descent method (cf. Theorem 3.1). Still, this bound is not fully satisfying as it depends polynomially on ε^{-1}. As a result, obtaining a solution that is very close to optimal, say, has $\varepsilon = \frac{1}{n^c}$, for some constant $c > 1$, becomes very expensive computationally.

It turns out, however, that gradient descent, with the exact same update rule (11), can converge *significantly* faster as long as the objective function g has an additional property: it is *strongly* convex. Formally, we say that g is α-*strongly convex*, for some $\alpha > 0$, if we have that

$$g(x + \Delta) \geq g(x) + \nabla g(x)^T \Delta + \frac{\alpha}{2}\|\Delta\|_2^2, \tag{16}$$

for every x and Δ. (Note that standard convexity corresponds to taking $\alpha = 0$ above.)

Clearly, α-strong convexity immediately implies the following straightening of the tail error approximation (14)

$$\frac{\alpha}{2}\|\Delta\|_2^2 \leq \varrho_x(\Delta) \leq \frac{\beta}{2}\|\Delta\|_2^2, \tag{17}$$

for all x and Δ. That is, now, we can both lower- and upperbound the objective g at point x by quadratic functions. This much tighter control of the error tail $\varrho_x(\Delta)$ leads to the following convergence bound.

Theorem 3.4 (see, e.g., Theorem 3.10 in Bubeck [ibid.])**.** *If the objective function g is α-strongly convex and β-smooth then, for any $\varepsilon > 0$, the update rule* (11) *with $\eta = \frac{1}{\beta}$ delivers an ε-approximate solution to problem* (3) *after at most*

$$T_\varepsilon \leq O\left(\frac{\beta}{\alpha}\log\frac{R}{\varepsilon}\right)$$

iterations, where $R = \|x^0 - x^*\|_2$ is the distance of the initial solution x^0 to a minimum x^*.

Observe that the convergence bound in the above theorem is only *logarithmic* in ε^{-1} (and R). So, this dependence is small enough that we can afford getting solutions that are close to optimal. Consequently, the key factor influencing the convergence of gradient descent in this case is the ratio $\frac{\beta}{\alpha}$. This ratio can be seen as expressing the worst-case ill-conditioning of the level sets of g and is thus often referred to as the *condition number* of g.

Finally, as in the previous section, the above iteration bound is not optimal. In particular, the dependence on the condition number that the above theorem presents can be improved

using an accelerated scheme due to Nesterov [1983, 2005], which corresponds to a certain more sophisticated update rule.

Theorem 3.5 (see, e.g., Theorem 3.18 in Bubeck [2015]). *If the objective function g is α-strongly convex and β-smooth then, for any $\varepsilon > 0$, one can compute an ε-approximate solution to problem (3) after at most*

$$T_\varepsilon \leq O\left(\sqrt{\frac{\beta}{\alpha}} \log \frac{R}{\varepsilon}\right)$$

iterations, where $R = \|x^0 - x^*\|_2$ is the distance of the initial solution x^0 to a minimum x^*.

4 First-Order–Based Approaches to Undirected Maximum Flow

The previous section laid down the key components of the continuous optimization framework we will need: the basic first-order convex optimization methods. We are thus ready to demonstrate how these methods can be applied to the maximum flow problem.

For now, our focus will be on solving a very basic variant of the problem: the one that corresponds to the graph being *undirected* and all capacities being *unit*, and in which we are interested in obtaining only an *approximately* optimal solution. Then, in Section 5, we will extend our approach to make it deliver exact solution to the general problem.

One should note that the classic, combinatorial approaches to the maximum flow problem are not known to be able to offer improved performance for this special variant of the problem. Specifically, the best combinatorial algorithm for this setting is still the classic $O(m \min\{\sqrt{m}, n^{\frac{2}{3}}\})$-time algorithm for the (exact) unit-capacity maximum flow problem due to Even and Tarjan [1975] and Karzanov [1973].

4.1 Maximum Flow as an Optimization Problem.

Our point of start is casting the (undirected) maximum flow problem as a continuous optimization task. To this end, we need first to encode our graph and the flows in it in the language of linear algebra, i.e., as vectors and matrices. Conveniently, our definition of a flow already views it as a vector in m-dimensional space. As a result, for a given demand vector $\sigma \in \mathbb{R}^n$, we can compactly express the flow conservation constraints (1) as

$$B^T \boldsymbol{f} = \sigma, \tag{18}$$

where B is an $m \times n$ *edge-vertex incidence matrix* defined as

$$B_{e,v} := \begin{cases} 1 & \text{if } v \text{ is } e\text{'s head} \\ -1 & \text{if } v \text{ is } e\text{'s tail} \\ 0 & \text{otherwise.} \end{cases} \tag{19}$$

Now, one should observe that by employing a simple binary search (and incurring an $O(\log nU) = O(\log n)$ factor running time overhead) we can reduce the task of (ε-approximate) solving of the maximum flow problem to solving an (ε-approximate) *flow feasibility problem*. In the latter problem, we are given a candidate value $F \geq 0$ as well as the desired accuracy $\varepsilon > 0$, and our goal is to either return a flow of value $F/(1+\varepsilon)$ that is feasible or to conclude that $F > F^*$.

The flow feasibility problem can be in turn cast as the following optimization task

$$\begin{aligned} \min_{x} \quad & \|x\|_\infty \\ \text{s.t.} \quad & x \in \mathcal{F}_{s,t}, \end{aligned} \tag{20}$$

where s is the source vertex, t is the sink vertex and

$$\mathcal{F}_{s,t} = \{x \mid B^T x = F\chi_{s,t}\}, \tag{21}$$

i.e., $\mathcal{F}_{s,t}$ is the affine subspace of all the vectors x that represent an s-t flow of value F.

Note that both the set $\mathcal{F}_{s,t}$ and the objective $\|\cdot\|_\infty$ are convex. Furthermore, it is not hard to see that obtaining an ε-approximate solution to the minimization problem (20) (and scaling this solution down by $(1+\varepsilon)$ when needed) gives us a solution the desired ε-approximate flow feasibility problem. So, from now on, we can focus on the former task.

4.2 Projections and Electrical Flows. The most straightforward way to solve the problem (20) is to apply to it the subgradient descent method (cf. Section 3.2). However, to apply the corresponding update rule (10), we need to describe how to compute the projection $\Pi_{\mathcal{F}_{s,t}}$ on the feasible set $\mathcal{F}_{s,t}$ (cf. (8)).

Since $\mathcal{F}_{s,t}$ is an affine subspace (cf. (21)), a simple calculations shows that, for any vector x,

$$\Pi_{\mathcal{F}_{s,t}}(x) = x - B\left(BB^T\right)^{\dagger}\left(B^T x - F\chi_{s,t}\right). \tag{22}$$

The above expression turns out to have a very natural interpretation. It corresponds to canceling out any deviation of the demand vector of the flow represented by x from the

desired demand vector $F\chi_{s,t}$ by routing it in the graph using electrical flows, i.e., flows that minimize the energy (wrt to unit edge resistances). In particular, the matrix BB^T is the Laplacian matrix of our graph and multiplying its pseudoinverse $\left(BB^T\right)^\dagger$ by a vector corresponds to solving a Laplacian system. Importantly, there is now a long line of work D. A. Spielman and Teng [2004], Koutis, Miller, and Peng [2010, 2011], J. A. Kelner, Orecchia, Sidford, and Zhu [2013], Cohen, Kyng, Miller, Pachocki, Peng, A. B. Rao, and Xu [2014], Kyng, Lee, Peng, Sachdeva, and D. A. Spielman [2016], and Kyng and Sachdeva [2016] that builds on an earlier work of Vaidya [n.d.] and D. A. Spielman and Teng [2003] that enables us to solve such a system, and thus also compute the projection $\Pi_{\mathcal{F}_{s,t}}(x)$, in nearly-linear time, i.e., in time $\widetilde{O}((m))$.[2]

Once we know how to implement each iteration of the update rule (11) efficiently, we can use Theorem 3.1 to bound the running time of the resulting algorithm. To this end, observe that our objective function $\|\cdot\|_\infty$ has a Lipschitz constant (cf. (9)) of $L = 1$, and with some care one can show that $R = \sqrt{m}$, provided we start with $x^0 = B\left(BB^T\right)^\dagger F\chi_{s,t}$. Given that each iteration can be implemented in $\widetilde{O}(m)$ time, this yields an overall running time bound of

$$\widetilde{O}(m)\cdot\frac{LR^2}{\varepsilon^2} = \widetilde{O}\left(m^2\varepsilon^{-2}\right), \tag{23}$$

to get an ε-approximate solution to our problem.

4.3 Smoothing Technique. The running time bound (23) is hardly satisfying given that classic algorithms Even and Tarjan [1975] and Karzanov [1973] run in $O(m\min\{\sqrt{m}, n^{\frac{2}{3}}\})$ time and deliver an *exact* solution to the problem. In fact, even the most basic Ford-Fulkerson algorithm Ford and Fulkerson [1956] and Elias, Feinstein, and Shannon [1956] offers a running time of $O(mn)$, which is also superior to the bound (23).

Still, this should not be viewed as the evidence that continuous optimization is an inadequate toolkit in this context. To the contrary! After all, the algorithm we just obtained came out of a very straightforward attempt. This attempt almost completely ignored the structure of the problem. It turns out that there is a principled methodology that one can apply here to address the shortcomings of this first attempt.

More precisely, a problematic issue with formulation (20) is that the objective function is convex and Lipschitz but not differentiable, let alone smooth. At first, this might seem to be an insurmountable obstacle. After all, (20) captures exactly the problem we want to solve! However, this turns out to not be entirely correct. Even though the objective itself captures our problem and is non-differentiable, it is still possible to *approximate* it with a

[2]The notation $\widetilde{O}(t(n))$ suppresses factors that are polylogarithmic in $t(n)$. Also, we ignore here the fact that the solutions computed by the solver are not exact, as this shortcoming can be alleviated in a standard way.

different "proxy" objective function that is differentiable and, in fact, smooth. (Note that we are looking here for approximate solution anyway, so this additional approximation is not detrimental.) This approach of substituting the original objective function with such a proxy function is known as *smoothing* Nesterov [2005]. In the context of our specific objective function, the corresponding proxy function is the *softmax* function defined as

$$\operatorname{smax}_\delta(x) := \delta \ln\left(\frac{\sum_{i=1}^{m} e^{\frac{x_i}{\delta}} + e^{\frac{-x_i}{\delta}}}{2m}\right), \tag{24}$$

for any $\delta > 0$.

Elementary calculation shows that $\operatorname{smax}_\delta$ is $\frac{1}{\delta}$-smooth and we have that

$$\|x\|_\infty - \delta \ln 2m \leq \operatorname{smax}_\delta(x) \leq \|x\|_\infty,$$

for any vector x. So, the parameter δ trades off the quality of approximation of the l_∞ norm and the smoothness of the resulting function. In particular, it is not hard to see that setting $\delta = \frac{\varepsilon}{2 \ln 2m}$ suffices for our purposes.

Now, by applying gradient descent update (11) to such *smoothened* version of the problem (20), by Theorem 3.2, we get an ε-approximation to the maximum flow problem in time

$$\widetilde{O}(m) \cdot O\left(\frac{\beta R^2}{\varepsilon}\right) \leq \widetilde{O}(m) \cdot O\left(\frac{R^2}{\delta\varepsilon}\right) = \widetilde{O}\left(m^2 \varepsilon^{-2}\right). \tag{25}$$

This new running time bound unfortunately matches the bound (23) we obtained using subgradient descent method. Still, the important differences is that now we are able to work directly with gradients of our objective function. As a result, the range of possible tools we can apply to the problem is much broader.

In particular, Christiano, J. Kelner, Mądry, D. Spielman, and Teng [2011] showed that one can use a certain variant of gradient descent method: the so-called *multiplicative weights update method* Plotkin, Shmoys, and Tardos [1995], N. E. Young [1995], N. Young [2001], and Arora, Hazan, and Kale [2012] to obtain an improved running time of $\widetilde{O}\left(m^{\frac{3}{2}} \varepsilon^{-\frac{5}{2}}\right)$. This running time, in a sense, already matches the classic $O(m \min\{\sqrt{m}, n^{\frac{2}{3}}\})$ running time bound of Even and Tarjan [1975] and Karzanov [1973] whenever the graph is sparse, i.e., $m = O(n)$.

4.4 Importance of Choosing the "Right" Geometry. As we have seen above, fairly standard gradient descent–based approaches are able to largely recover the best known classic running time bounds. At least when we only aim for an ε-approximate solution to the undirected variant of the maximum flow problem. Naturally, the fact that we merely

recover these bounds might not be fully satisfying. So, it is important to understand what are the key obstacles preventing us from obtaining an actual improvement here.

The choice that turns out to play a key role in this context is the choice of the geometry we use when projecting back onto the feasible space in the update rule (11) of the gradient descent method. Specifically, we defined our projection $\Pi_{\mathcal{F}_{s,t}}$ to be an ℓ_2-projection, i.e., to always project a given point x to a feasible point y that is the closest one with respect to the ℓ_2 distance $\|x-y\|_2$ (see (8)). This choice was convenient since it enables us to compute this projection fast via Laplacian system solvers (see (22)). However, the ℓ_2–based geometry this projection works with ends up being ill-suited to the maximum flow problem.

The root of the problem here is that the maximum flow problem corresponds to an ℓ_∞–based geometry. In particular, as made explicit in formulation (20), its objective is to find a minimum ℓ_∞ norm point in an affine space ($\mathcal{F}_{s,t}$). It is not hard to see, however, that the ℓ_2– and ℓ_∞–based notions of distance can sometimes vary significantly. So, points that are "close" with respect to ℓ_∞–based distance might seem far when computing the projection with respect to ℓ_2–based distance. More precisely, if we are working in m-dimensional space, which is the case for us, we have that

$$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{m}\,\|x\|_\infty, \tag{26}$$

and both these inequalities are tight, e.g., when x has just a single non-zero coordinate or is an all-ones vector, respectively.

This $\sqrt{m}$ discrepancy manifests itself directly when establishing the upper bound R on the initial distance to the optimum solution (see Theorem 3.2). In a sense, the fact that our bound on R was only $O(\sqrt{m})$ is tied closely to the worst-case discrepancy captured by (26).

This realization prompted Christiano, J. Kelner, Mądry, D. Spielman, and Teng [2011] to change the geometry used in the projection. Specifically, instead of working with the distance induced by the ℓ_2 norm, they work with an ℓ_2 norm that is coordinate-wise reweighted in an *adaptive* manner. That is, in each step t, one projects with respect to the distance induced by the norm $\|\cdot\|_{D_t}$, where each D_t is a positive diagonal matrix and, additionally, $D_{t+1} \succeq D_t$ for each t. The choice of these matrices D_t is such that the corresponding distance measure approximates the ℓ_∞–based distances well (at least, in the directions relevant for the current solution) and thus avoids the $O(\sqrt{m})$ worst-case discrepancy discussed above. Also, since D_t is a diagonal matrix, $\|\cdot\|_{D_t}$ is still at its core an ℓ_2 norm. So, this enables us to use the Laplacian solver–based approach to make the projections step remain fast.

These ideas give rise to the following result that finally improves over the classic running time bounds.

Theorem 4.1 (Christiano, J. Kelner, Mądry, D. Spielman, and Teng [ibid.]). *For any $\varepsilon > 0$, one can compute an ε-approximate maximum flow in undirected graph in time $\widetilde{O}\left(m^{\frac{4}{3}}\varepsilon^{-3}\right)$.*

Christiano, J. Kelner, Mądry, D. Spielman, and Teng [ibid.] also demonstrate how to use sparsification techniques D. A. Spielman and Teng [2008] and D. A. Spielman and Srivastava [2008] to get an $\widetilde{O}\left(mn^{\frac{1}{3}}\varepsilon^{-\frac{11}{3}}\right)$–time algorithm. This algorithm is more favorable for dense graph setting, at the expense of slightly higher dependence on $\frac{1}{\varepsilon}$. Finally, it is worth noting that Lee, S. Rao, and Srivastava [2013] subsequently demonstrated that one can obtain a similar result using the accelerated gradient descent method (see Theorem 3.3).

4.5 Interlude: Gradient Descent Method for General Norms. As discussed in the previous section, the key to obtaining faster algorithms for the maximum flow problem is to understand and exploit the interplay between the geometries of the problem and the one used by the gradient descent method. Once we manage to align these two geometries better, we can obtain an improved running time.

This gives rise to a question: how much flexibility does the gradient descent framework have in terms of the geometry it can work in? So far, all our considerations revolved around the ℓ_2 geometry (and its coordinate-wise reweightings). However, it turns out that gradient descent method can be applied to *any* geometry that is induced by a norm.

In fact, our treatment of gradient descent method presented in Section 3.3 can be translated into this broader, general norm setting almost syntactically. (Although there are certain important differences.) The point of start is extending the notion of β-smoothness (12). We say that an objective function is *β-smooth* (with respect to a general norm $\|\cdot\|$) iff

$$\|\nabla g(x) - \nabla g(y)\|^* \leq \beta\|x - y\|, \tag{27}$$

for every x and y, where $\|\cdot\|^*$ denotes the *dual norm* of $\|\cdot\|$, defined as

$$\|y\|^* = \max_{x \neq 0} \frac{y^T x}{\|x\|}. \tag{28}$$

Observe that if $\|\cdot\|$ is the ℓ_2 norm then $\|\cdot\|^*$ is also the ℓ_2 norm. (This corresponds to the fact that ℓ_2 norm is self-dual.) Thus, the definition (12) is a special case of the above definition. In general, the primal norm $\|\cdot\|$ and its dual norm $\|\cdot\|^*$ are different. In particular, $\|\cdot\|_p^* = \|\cdot\|_q$, where $\frac{1}{p} + \frac{1}{q} = 1$.

Now, similarly as in Section 3.3, the fact that the objective function g is β-smooth (and convex) enables us to derive the following analogue of the bound (14) on the behavior of

the tail error term $\varrho_x(\Delta)$ of the linear Taylor approximation $\varphi_x(\Delta)$ (see (13)).

$$0 \leq \varrho_x(\Delta) \leq \frac{\beta}{2}\|\Delta\|^2, \tag{29}$$

for all x and Δ.

This, in turn, enables us to derive the optimal update Δ_x^* (see (15)) to be

$$\Delta_x^* = \operatorname{argmin}_\Delta \left(\varphi_x(\Delta) + \frac{\beta}{2}\|\Delta\|^2\right) = -\frac{1}{\beta}\operatorname{argmax}_\Delta \left(\nabla g(x)^T\Delta - \frac{1}{2}\|\Delta\|^2\right) = -\frac{1}{\beta}\left(\nabla g(x)\right)^\sharp, \tag{30}$$

where $\cdot^\sharp$ operator is defined as

$$y^\sharp = \operatorname{argmax}_\Delta \left(y^T\Delta - \frac{1}{2}\|\Delta\|^2\right). \tag{31}$$

Observe that, again, if $\|\cdot\|$ is the ℓ_2 norm then $(y)^\sharp = y$ and (15) becomes a special case of (30). In general, however, $(y)^\sharp \neq y$ and, for example, when $\|\cdot\|$ is the ℓ_∞ norm, we have that

$$(y)_i^\sharp = \operatorname{sign}(y_i)\cdot|y|_1,$$

for each coordinate i.

The final step is to make our projection $\Pi_{\mathcal{K}}$ correspond to the distance induced by our general norm $\|\cdot\|$. Namely, we take (cf. (8))

$$\Pi_{\mathcal{K}}(x) = \operatorname{argmin}_{y\in\mathcal{K}}\|y - x\|, \tag{32}$$

for any x. As a result, our overall update rule becomes

$$x^t \leftarrow \Pi_{\mathcal{K}}\left(x^{t-1} - \eta\left(\nabla g(x^{t-1})\right)^\sharp\right), \tag{33}$$

where, again, $\eta > 0$ is the step size.

Once we put all these elements together, we obtain a direct analogue of Theorem 3.2.

Theorem 4.2. *If the objective function g is β-smooth with respect to norm $\|\cdot\|$ then, for any $\varepsilon > 0$, the update rule (33) with $\eta = \frac{1}{\beta}$ delivers an ε-approximate solution after at most*

$$T_\varepsilon \leq \beta R^2 \varepsilon^{-1}$$

iterations, where $R = \|x^0 - x^*\|$ is the distance of the initial solution x^0 to a minimum x^*.

It is important to remember that even though the above theorem seems to be an almost verbatim repetition of Theorem 3.2, the fact that all the notions correspond to a general norm is crucial and, as we will see shortly, it significantly enhances the power of this framework.

Finally, we note that once we extend the notion of α-strong convexity (16) in an analogous manner, i.e., define the objective function to be α-*strong convex* (with respect to a general norm $\|\cdot\|$) iff

$$g(x+\Delta) \geq g(x) + \nabla g(x)^T \Delta + \frac{\alpha}{2}\|\Delta\|^2, \tag{34}$$

for every x and Δ, we can obtain a corresponding tighter control on the behavior of the tail error term $\varrho_x(\Delta)$ of our (local) linear approximation $\varphi_x(D)$ of the objective function g at x (cf (13)). Specifically, as a direct analogue of the inequalities (17), we have that

$$\frac{\alpha}{2}\|\Delta\|^2 \leq \varrho_x(\Delta) \leq \frac{\beta}{2}\|\Delta\|^2, \tag{35}$$

and thus the following extension of Theorem 3.4 can be established.

Theorem 4.3. *If the objective function g is α-strongly convex and β-smooth with respect to a norm $\|\cdot\|$ then, for any $\varepsilon > 0$, the update rule (33) with $\eta = \frac{1}{\beta}$ delivers an ε-approximate solution after at most*

$$T_\varepsilon \leq \frac{\beta}{\alpha}\log\frac{R}{\varepsilon}$$

iterations, where $R = \|x^0 - x^*\|$ is the distance of the initial solution x^0 to a minimum x^*.

4.6 Solving the ε-approximate Undirected Maximum Flow in $\widetilde{O}\left(m\varepsilon^{-1}\log U\right)$ Time. As first noted by Sherman [2013] and J. Kelner, Lee, Orecchia, and Sidford [2014] independently, the framework described in the previous section is particularly well-suited to tackle the ε-approximate undirected maximum flow problem.[3] Indeed, let us consider applying an ℓ_∞ norm variant of the gradient descent method to a version of the problem (20) that was smoothened as described in Section 4.3. One can readily notice that such smoothened objective has a smoothness of

$$\beta = \frac{1}{\delta} = \frac{2\ln 2m}{\varepsilon}$$

[3] Strictly speaking, the variant of the framework presented here corresponds to the one employed in the work of J. Kelner, Lee, Orecchia, and Sidford [2014]. Sherman [2013] relied on its slightly different, dual variant.

also with respect to the ℓ_∞ norm. Additionally, one can bound the distance to the optimum R in that norm to be only $O(1)$. As a result, Theorem 4.2 yields an iteration bound of only $O(\frac{\ln m}{\varepsilon^2})$!

Unfortunately, this alone does not yet provide us with a nearly-linear time algorithm. The issue is that now in each iteration of the algorithm we need to compute a projection $\Pi_{\mathcal{F}_{s,t}}$ with respect to the ℓ_∞ norm – instead of the (reweighted) ℓ_2 norm. This is problematic as, in general, computing this projection is as hard as solving the *exact* maximum flow problem. Specifically, if we consider a projection $\Pi_{\mathcal{F}_{s,t}}(0)$ of the all-zero vector, we note that

$$\Pi_{\mathcal{F}_{s,t}}(0) = \operatorname{argmin}_{y \in \mathcal{F}_{s,t}} \|y\|_\infty,$$

which is exactly the *optimal* solution to our problem (20).

To circumvent this conundrum, Sherman [2013] and J. Kelner, Lee, Orecchia, and Sidford [2014] make a crucial observation: we do not need to compute the projection $\Pi_{\mathcal{F}_{s,t}}(x)$ exactly. It suffices to compute some γ-approximation of it. Even if the value of γ is fairly large, its only impact is on the running time, i.e., the running time will depend polynomially on γ, but *not* on the quality of the solution one obtains in the end. With this insight in mind, they build on the work of Mądry [2010] that gave an $\widetilde{O}\left(m^{1+o(1)}\right)$-time $n^{o(1)}$-approximation algorithm for the *value* of the undirected maximum flow, to compute such $\gamma = n^{o(1)}$-approximate projection in $\widetilde{O}\left(m^{1+o(1)}\right)$ time. This gives rise to the following result.

Theorem 4.4 (Sherman [2013] and J. Kelner, Lee, Orecchia, and Sidford [2014])**.** *For any $\varepsilon > 0$, one can compute an ε-approximate maximum flow in undirected graph in time $O\left(m^{1+o(1)}\varepsilon^{-2}\log U\right)$.*

Note that the above running time already refers to the general, not necessary unit-capacity, version of the undirected maximum flow problem. (The necessary adjustment boils down to applying an appropriate coordinate-wise scaling that corresponds to edge capacities. It also introduces the $O(\log U)$ term in the running time due to the need to perform a binary search in the reduction of the maximum flow problem to the flow feasibility problem – see Section 4.1.)

In follow up work, Peng [2016] provided an improved variant of the algorithm that runs in $\widetilde{O}\left(m\varepsilon^{-2}\right)$ time. Finally, Sherman [2017a] further improved the dependence on $\frac{1}{\varepsilon}$, giving rise to the following theorem.

Theorem 4.5 (Sherman [ibid.])**.** *For any $\varepsilon > 0$, one can compute an ε-approximate maximum flow in undirected graph in time $\widetilde{O}\left(m\varepsilon^{-1}\log U\right)$.*

It is worth noting that in the unit capacity setting (i.e., when $U = 1$), if we wanted to obtain an *exact* solution to the undirected maximum flow problem, it suffices to set $\varepsilon = \frac{1}{\sqrt{n}}$

and then "fix" the resulting approximate solution by rounding the flow and computing at most $\sqrt{n}$ augmenting paths, which would take $O(m\sqrt{n})$ time. So, the above result also matches (up to polylogarithmic factors) – and, in fact, for dense graphs improves – the classic $O(m \min\{\sqrt{m}, n^{\frac{2}{3}}\})$-time algorithms of Even and Tarjan [1975] and Karzanov [1973] also in the regime of exact answers.

5 Computing Maximum Flows with Second-Order Methods

In the previous section, we demonstrated how continuous optimization–based approaches such as gradient descent method can make substantial progress on the problem of computing ε-approximately maximum flows in undirected graphs (cf. Theorem 4.5). Can though some of this progress be leveraged to get improved algorithms also for the general variant of the maximum flow problem, i.e., the task of computing *exact* maximum flows in *directed* graphs?

At first glance, the key challenge in adapting the techniques from previous section to this new setting might be that these techniques were defined only in the context of undirected graphs. So, it is unclear how to extend them to the directed graph context. It turns out, however, that this alone is not a problem. Specifically, one can show that the maximum flow problem in directed graphs can be *efficiently* reduced to the maximum flow problem in *undirected* graphs. At least, in the regime of sparse graph. (See Theorem 3.6.1 in Mądry [2011] for details.)

Crucially, however, this reduction only holds if we are able to solve the undirected maximum flow problem (almost) exactly. The key shortcoming of the methods from Section 4 is thus that, due to their running time bounds being polynomial in ε^{-1}, they do not offer sufficiently good accuracy here. This deficiency is, in a sense, inherent to the first-order framework that these methods rely on. Specifically, this shortcoming is tied to the fact that our objective function g in problem (20) is *not* strongly convex (and smoothening does not affect this aspect). As a result, we are unable to benefit from the corresponding, much improved convergence bound given by Theorem 4.3.

This realization motivates us to consider a more powerful continuous optimization methodology: the so-called *second-order methods*, i.e., approaches that, in contrast to the first-order approaches that rely solely on the information conveyed by the gradients of the objective function, also take advantage of probing the *Hessians* of that function.

5.1 Barrier–Based Maximum Flow Problem Formulation. Our point of start here is casting the maximum flow problem as a special type of a constrained minimization: linear program (LP). That is, instead of formulating the maximum flow problem as a problem of ℓ_∞ norm minimization over an affine subspace – as was the case in (20), we phrase it

as a task of minimizing a simple, *linear* objective function over a more complex, convex feasible set.

Specifically, the maximum flow problem can be cast as the following LP.

$$\begin{aligned} &\min_x c^T x \\ &s.t. B^T x = 0 \\ &\quad x_e \leq u_e^+ \forall_e \\ &\quad x_e \geq u_e^- \forall_e. \end{aligned} \tag{36}$$

Here, B is an edge-vertex incidence matrix (cf. (19)) of the input graph after we added to it an edge $\widehat{e}$ that connects the sink t to the source s and has its lower capacity $u_{\widehat{e}}^-$ be 0 and its upper capacity $u_{\widehat{e}}^+$ be unbounded. Also, c is a vector in which $c_{\widehat{e}} = -1$ and $c_e = 0$ for all other edges e. (This added edge $\widehat{e}$ ensures that the circulation x that we find in the augmented graph corresponds to a valid s-t flow in the original graph, and the penalty we apply to that edge ensures that in the optimal solution the corresponding s-t flow is indeed a maximum one.)

As the LP (36) is an example of a constrained minimization problem, we are, in principle, able to apply to it the gradient descent framework we described in Section 3. However, in that case, the update rule (11) would require us to compute in each step the projection onto the feasible set of this LP, and it is difficult to implement that task efficiently. The key reason here is the presence of the inequality constraints. (Note that projection on the kernel of the matrix B would again correspond to solving a Laplacian system – cf. (22), and thus could be computed fast.)

To cope with this problem, we will employ a technique that is inspired by one of the most popular family of approaches to solving linear programs: the *interior point methods* (see Boyd and Vandenberghe [2004], S. J. Wright [1997], and Ye [1997]). This technique, instead of maintaining the inequality constraints explicitly, enforces them implicitly by introducing an appropriate term to the objective. Specifically, instead of solving the LP (36), we aim to solve the following constrained minimization problem.

$$\begin{aligned} &\min_x c^T x + \psi_\mu(x) \\ &s.t. B^T x = 0, \end{aligned} \tag{37}$$

where

$$\psi_\mu(x) = -\mu \sum_e \left(\ln(u_e^+ - x_e) + \ln(x_e - u_e^-) \right), \tag{38}$$

is the *barrier function* and $\mu > 0$ is a positive number.

Observe that as long as $\mu > 0$ and our initial solution is feasible, the barrier function will ensure that any solution we find using an iterative minimization scheme, such as gradient descent method, remains feasible. Also, it is not hard to see that the smaller $\mu > 0$ is the closer the optimal solution to the problem (37) is to the desired optimal solution to the LP (36). In particular, one can show that to get an ε-approximate solution to the LP (36) it suffices to set $\mu \leq \frac{\varepsilon}{2m}$ and find the optimal solution to the corresponding problem (37).

Finally, observe that the Hessian of the objective function of (37) at some point x is equal to the Hessian $\nabla^2 \psi_\mu(x)$ of the barrier function. The latter turns out to be a diagonal matrix with

$$\left(\nabla^2 \psi_\mu(x)\right)_{e,e} = \mu \left(\frac{1}{(u_e^+ - x_e)^2} + \frac{1}{(x_e - u_e^-)^2} \right),$$

for each edge e. Thus, the Hessian of our objective function is positive definite (provided $\mu > 0$).

Now, one should note that the fact that this Hessian is positive definite implies that the objective function is *strongly* convex. So, if we attempted to solve problem (37) using gradient descent method we can take advantage of the improved iteration bound described by Theorem 3.4. Importantly, this bound depends only logarithmically on ε^{-1}, which is what we need in order to be able to compute the (almost) exact solutions.

Unfortunately, even though this objective function is indeed α-strongly convex and β-smooth, its condition number $\frac{\beta}{\alpha}$ might be very large. The bound delivered by Theorem 4.3 would thus still be prohibitively large, even if it would have the "right" dependence on ε^{-1}. It turns out that in order to get a more efficient algorithm we need to resort to a more powerful technique: the Newton's method.

5.2 Interlude: Netwon's Method. Recall that the key quantity that impacts the iteration bound given by Theorem 4.3 is the condition number $\frac{\beta}{\alpha}$. This number captures the degree of control we have on the behavior of the tail error $\varrho_x(\Delta)$ – see (35), in terms of the norm $\|\cdot\|$ we work with. However, in principle, we have a complete freedom in choosing this norm. So, one might wonder: what is the "best" norm to choose in order to make this condition number be as small as possible?

Observe that, for a given point x and sufficiently small Δ, Taylor expansion (13) gives us that

$$\varrho_x(\Delta) \approx \frac{1}{2}\Delta^T \nabla^2 g(x) \Delta = \frac{1}{2}\|\Delta\|^2_{\nabla^2 g(x)},$$

where the norm $\|\cdot\|_{\nabla^2 g(x)}$ is called the *local norm* of g at x. (Note that, crucially, this norm might be different at each point x.)

So, as long as the objective function g is strongly convex for *some* α with respect to the ℓ_2 norm (and thus $\nabla^2 g(x) \succ 0$), the local norm is well-defined at each point x and

locally, i.e., in a sufficiently close neighborhood of that point x, the condition number of g with respect to $\|\cdot\|_{\nabla^2 g(x)}$ is close to best possible, i.e., close to 1!

This suggests employing an iterative method in which we repeatedly perform a gradient descent update (33) with respect to such local norm (at the current point). This method is known as the *Newton's method*. Clearly, its most attractive feature is that whenever the "sufficiently close neighborhood" of x contains the minimum x^* of the objective function g, Theorem 4.3 ensures very fast convergence *regardless of the "natural", i.e., ℓ_2–based, condition number of g*. (In fact, one can show that in this case the convergence can be even faster than the one promised by that theorem.)

Unfortunately, this method has also two important shortcomings. First of all, it has no meaningful convergence guarantees if the minimum x^* of the objective function g is not sufficiently close to the current point x. In fact, even the notion of "sufficiently close" is in general not well defined. One thus usually is able to analyze Newton's method only for special class of functions such as self-concordant functions Nesterov and Nemirovskii [1994]. (The barrier function (38) is self-concordant by design.)

Also, the other shortcoming is that each update step of Newton's method can be computationally expensive. Specifically, recall that by (33) and (31), we have that this method's update rule becomes

(39)
$$x^t \leftarrow \Pi_{\mathcal{K}} \left(x^{t-1} - \eta \left(\nabla g(x^{t-1}) \right)^{\sharp} \right) = \Pi_{\mathcal{K}} \left(x^{t-1} - \eta \left(\nabla^2 g(x^{t-1}) \right)^{-1} \nabla g(x^{t-1}) \right),$$

where we used the fact that $y^{\sharp} = A^{-1} y$ when we work with respect to the norm $\|\cdot\|_A$. So, implementing each step of the Newton's method requires solving a linear system in the local Hessian of the objective function.

5.3 Faster Algorithms for the Maximum Flow Problem. The shortcomings of the Newton's method that we identified above severely limit its usefulness as a general optimization procedure. Still, this method turns out to be very powerful when carefully applied. In particular, it is a key element of the interior point method–based approaches to LP solving Boyd and Vandenberghe [2004], Ye [1997], and S. J. Wright [1997].

More concretely, the so-called path-following variants of the interior point methods solve the LP (36) by solving a *sequence* of barrier problems (37) (instead of solving directly the one that corresponds to the desired, sufficiently small value of μ). Specifically, they start by obtaining a (near) optimal solution to the problem (37) for a *large* value of μ. (One can show that after appropriate preprocessing of the problem, such a solution is readily available.) Then, these algorithms repeatedly use the Newton's method to compute a (near) optimal solution to the barrier problem (37) for a slightly smaller value of μ *while using the previously obtained solution as a warm start*. (This warm starting is crucial as it ensures that the Newton's method is always in its rapid convergence stage.)

Now, one of the most central theoretical challenges in continuous optimization is establishing bounds on the number of iterations that the above path-following procedure requires. That is, bounding the number of such barrier subproblems that need to be solved in order to obtain an ε-approximate solution to the LP (36). In 1988, Renegar [1988] established a general iteration bound of $O(\sqrt{m}\log\frac{1}{\varepsilon})$. Also, Daitch and D. A. Spielman [2008] observed that when one solves flow LPs such as (36), the Newton's method update step (39) for the corresponding barrier problem (37) can be implemented in nearly-linear time using a Laplacian solver. This enabled them to obtain an $\widetilde{O}\left(m^{\frac{3}{2}}\log U\right)$-time algorithm for the maximum flow problem as well as a host of its generalizations.

However, this running time is hardly satisfying. In particular, it is still inferior to the running time of $O(m\min\{m^{\frac{1}{2}}, n^{\frac{2}{3}}\}\log(n^2/m)\log U)$ due to Goldberg and S. Rao [1998]. Unfortunately, obtaining an improvement here hinges on going beyond the Renegar's $O(\sqrt{m}\log\frac{1}{\varepsilon})$ iteration bound, which is one of the longstanding open problems in the field.

To address this challenge, Mądry [2013] developed a more fine-grained understanding of the convergence behavior of such interior point methods. This understanding showed that, similarly as it was the case in the context of undirected maximum flow problem (see Section 4.4), the slower convergence is directly tied to an underlying "geometry mismatch". In particular, the $O(\sqrt{m})$ term in Renegar's bound arises due to the compounded worst-case discrepancies between the ℓ_∞ and ℓ_4 as well as ℓ_4 and ℓ_2 norms. Mądry then builds on the idea of adaptive coordinate-wise reweighting of the ℓ_2 norm put forth in the work of Christiano, J. Kelner, Mądry, D. Spielman, and Teng [2011] (see Section 4.4) to alleviate this worst case discrepancies.

Specifically, after translating to our setting, the approach of Mądry [2013] can be viewed as considering a coordinate-wise reweighted version of the barrier function ψ_μ defined as

$$\psi_\mu(x) = -\mu\sum_e v_e\left(\ln(u_e^+ - x_e) + \ln(x_e - u_e^-)\right), \tag{40}$$

where each $v_e \geq 1$ is an individual weight tied to edge e's constraints. Then, Mądry develops an adaptive scheme to update the weights v_e that is inspired by (but more involved than) the reweighting scheme due to Christiano, J. Kelner, Mądry, D. Spielman, and Teng [2011]. This scheme together with a careful analysis enables one to obtain the following result.

Theorem 5.1 (Mądry [2013, 2016]). *The maximum flow problem can be solved in* $\widetilde{O}\left(m^{\frac{10}{7}}U^{\frac{1}{7}}\right)$ *time.*

Observe that the resulting algorithm constitutes the first improvement over the classic $O(m\min\{m^{\frac{1}{2}}, n^{\frac{2}{3}}\})$-time results of Even and Tarjan [1975] and Karzanov [1973] for

unit capacity graphs as well as the general capacity result of Goldberg and S. Rao [1998], provided the capacities are not too large and the graph is sufficiently sparse. In fact, the improved variant of interior point method developed in Mądry [2013] can be applied to solving *any* linear program. Unfortunately, the underlying reweighting scheme involves permanent perturbation of the cost vector of that solved LP. This method is thus of limited use if one cannot control the impact of these perturbations on the final solution we find, as one can in the context of the maximum flow problem. (It is, however, possible that introducing these perturbations is not truly necessary and an alternative reweighting scheme could avoid it.)

Later on, Lee and Sidford [2014] provided a different barrier function reweighting scheme. Their result builds on the work of Vaidya [1989] and delivers an interior point method that converges in only $\widetilde{O}\left(\sqrt{n}\log\frac{1}{\varepsilon}\right)$ iterations, which yields the following theorem.

Theorem 5.2 (Lee and Sidford [2014])**.** *The maximum flow problem can be solved in* $\widetilde{O}\left(m\sqrt{n}\log U\right)$ *time.*

This result thus improves over the classic work of Even and Tarjan [1975], Karzanov [1973] and Goldberg and S. Rao [1998] for dense graph case. Importantly, this new interior point method can be readily applied to *any* LP. (In fact, it was developed directly in the general LP setting.) In particular, the iteration bound it gives matches – and in some cases even outperforms – the bound stemming from the seminal self-concordant barrier of Nesterov and Nemirovskii [1994] and, in contrast to the latter, the method of Lee and Sidford is efficiently computable. Due to this generality, Lee and Sidford [2014] is able to provide, in particular, improved running times for a number of generalizations of the maximum flow problem too.

6 Open Problems

Last decade has brought us a significant progress on algorithms for the maximum flow problem (see Theorems 4.5, 5.1 and 5.2) as well as for the related graph problems such as (weighted) bipartite matching and general shortest path problems Cohen, Mądry, Sankowski, and Vladu [2017]. However, there is still a number of key open problems that remain unsolved and, hopefully, further work on them will lead to a better understanding of both the graph algorithms as well as the continuous optimization toolkit we employ in this context. We briefly discuss some of these questions below.

Arguably, the most progress so far has been made in the context of the ε-approximate algorithms for the undirected maximum flow problem. In particular, Theorem 4.5 delivers a solution that could be viewed, essentially, as the best possible in this regime. Still, it is interesting to understand how flexible the underlying framework is. In particular, if it is

possible to apply it with a similar success to other, related flow problems. Indeed, Sherman [2017b] showed that this framework can deliver an almost linear time ε-approximate algorithm for a variant of the problem (20) in which we minimize the ℓ_1 norm of the vector instead of its ℓ_∞ norm. This corresponds to solving a special case of the *minimum-cost* flow problem in which the capacities are *unbounded* and costs are all unit. It is thus natural to attempt to use this approach to tackle the general undirected minimum-cost flow problem as well.

Challenge 6.1. *Solve the general minimum-cost flow problem in undirected graphs in $O(m^{1+o(1)}\varepsilon^{-2}\log(C+U))$ time, where C is the largest (integer) cost and U, as usual, is the largest (integer) capacity.*

Intuitively, solving this problem corresponds to solving the problem (20) in the case when the objective is a *linear combination* of ℓ_∞ and ℓ_1 norms, with each one of them being reweighted coordinate-wise according to the capacities and costs, respectively. Since these two norms are dual to each other, it is unclear how to "reconcile" them to get the desired running time improvement. The current state of the art for this problem is the $\widetilde{O}\left(m\sqrt{n}\log(U+C)\right)$-time *exact* algorithm due to Lee and Sidford [2014] and the $\widetilde{O}\left(m^{\frac{10}{7}}\log C\right)$-time exact algorithm of Cohen, Mądry, Sankowski, and Vladu [2017] for the *unit capacity* (but general costs) variant of the problem.

In the context of the maximum flow problem, the most central direction is to extend further the progress made on the exact algorithms for the general, directed graph setting. (See Theorems 5.1 and 5.2.) In particular, given the fact that Theorem 5.1 builds on the adaptive coordinate-wise ℓ_2 norm reweighting idea of Christiano, J. Kelner, Mądry, D. Spielman, and Teng [2011], one could wonder if it is possible to match the type of running time improvement that was achieved in the latter work.

Challenge 6.2. *Obtain an $\widetilde{O}\left(m^{\frac{4}{3}}\log U\right)$ time algorithm for the (exact) maximum flow problem.*

Note that the above challenge is interesting also in the setting of unit capacity maximum flow, i.e., when $U=1$. As it would still constitute an improvement over the current best $\widetilde{O}\left(m^{\frac{10}{7}}U^{\frac{1}{7}}\right)$-time algorithm due to Mądry [2013, 2016].

Finally, even though our treatment so far focused on applying the continuous optimization framework to graph algorithms, this connection can be also used in the opposite way. That is, we can view graph algorithmic problems as a useful "testbed" for understanding the full power and limitations of the current continuous optimization tools. This understanding can, in turn, be translated into progress on the challenges in core continuous optimization.

In fact, the advances on the maximum flow problem made so far already yielded important progress on such core questions. On one hand, the ε-approximation result described in Theorem 4.5 required going beyond the standard gradient descent–based framework to overcome its fundamental limitation: the inability to obtain the acceleration (as in Theorem 3.3) for the ℓ_∞ norm based variant of gradient descent. On the other hand, the progress on the exact maximum flow problem captured by Theorems 5.1 and 5.2 brought us key advances on the convergence bounds for general interior point methods. In the light of this, the natural next goal here would be to use the adaptive ℓ_2 norm reweighing ideas to get the following improvement.

Challenge 6.3. *Develop an interior point method that computes an ε-approximate solution to any linear program using $\widetilde{O}\left(m^{\frac{4}{3}} \log \frac{1}{\varepsilon}\right)$ iterations, with each iteration requiring only $O(1)$ linear system solves*[4].

One should note that this challenge subsumes Challenge 6.2. This is so as the maximum flow problem can be cast as a linear program in which the linear systems to be solved in each iteration of the interior point method are Laplacian (and thus can be solved efficiently) – see Section 5.1. We also view tackling the above challenge as the current most promising way to approach Challenge 6.2.

References

Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin (1993). *Network flows: theory, algorithms, and applications*. Prentice-Hall (cit. on p. 3345).

Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin, and M. R. Reddy (1995). *Applications of Network Optimization*. Vol. 7. Handbooks in Operations Research and Management Science. North-Holland, pp. 1–75 (cit. on p. 3345).

Sanjeev Arora, Elad Hazan, and Satyen Kale (2012). "The Multiplicative Weights Update Method: a Meta-Algorithm and Applications". *Theory of Computing* 8.1, pp. 121–164 (cit. on pp. 3345, 3357).

Stephen Boyd and Lieven Vandenberghe (2004). *Convex Optimization*. Cambridge University Press (cit. on pp. 3348, 3364, 3366).

Sébastien Bubeck (2015). "Convex Optimization: Algorithms and Complexity". *Found. Trends Mach. Learn.* 8.3-4, pp. 231–357 (cit. on pp. 3348, 3351–3354).

[4] To be precise, the linear systems to solve should be in a matrix of a form $\boldsymbol{A}^T \boldsymbol{D}\boldsymbol{A}$, where $\boldsymbol{A}$ is the constraint matrix of the linear program and $\boldsymbol{D}$ is some positive diagonal matrix.

Paul Christiano, Jonathan Kelner, Aleksander Mądry, Daniel Spielman, and Shang-Hua Teng (2011). "Electrical Flows, Laplacian Systems, and Faster Approximation of Maximum Flow in Undirected Graphs". In: *STOC'11: Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, pp. 273–281 (cit. on pp. 3346, 3357–3359, 3367, 3369).

Fan R. K. Chung (1997). *Spectral Graph Theory*. American Mathematical Society (cit. on p. 3346).

Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup B. Rao, and Shen Chen Xu (2014). "Solving SDD Linear Systems in Nearly m $\log^{1/2} n$ Time". In: *STOC'14: Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, pp. 343–352 (cit. on p. 3356).

Michael B. Cohen, Aleksander Mądry, Piotr Sankowski, and Adrian Vladu (2017). "Negative-Weight Shortest Paths and Unit Capacity Minimum Cost Flow in $\tilde{O}(m^{10/7} \log W)$ Time". In: *SODA'17: Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms* (cit. on pp. 3368, 3369).

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein (2009). *Introduction to Algorithms*. 3rd. The MIT Press (cit. on p. 3345).

Samuel I. Daitch and Daniel A. Spielman (2008). "Faster approximate lossy generalized flow via interior point algorithms". In: *STOC'08: Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pp. 451–460 (cit. on p. 3367).

Peter Elias, Amiel Feinstein, and Claude E. Shannon (1956). "A note on the maximum flow through a network". *IRE Transactions on Information Theory* 2 (cit. on pp. 3345, 3356).

Shimon Even and R. Endre Tarjan (1975). "Network Flow and Testing Graph Connectivity". *SIAM Journal on Computing* 4.4, pp. 507–518 (cit. on pp. 3346, 3354, 3356, 3357, 3363, 3367, 3368).

Lester R Ford and Delbert R Fulkerson (1956). "Maximal Flow Through a Network". *Canadian Journal of Mathematics* 8, pp. 399–404 (cit. on pp. 3345, 3356).

Andrew V. Goldberg and Satish Rao (1998). "Beyond the flow decomposition barrier". *Journal of the ACM* 45.5, pp. 783–797 (cit. on pp. 3345, 3346, 3367, 3368).

Alexander V. Karzanov (1973). "O nakhozhdenii maksimal'nogo potoka v setyakh spetsial'nogo vida i nekotorykh prilozheniyakh". *Matematicheskie Voprosy Upravleniya Proizvodstvom* 5. (in Russian; title translation: On finding maximum flows in networks with special structure and some applications), pp. 81–94 (cit. on pp. 3346, 3354, 3356, 3357, 3363, 3367, 3368).

Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu (2013). "A Simple, Combinatorial Algorithm for Solving SDD Systems in Nearly-Linear Time". In: *STOC'13: Proceedings of the 45th Annual ACM Symposium on the Theory of Computing*, pp. 911–920 (cit. on p. 3356).

Jonathan Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford (2014). "An Almost-Linear-Time Algorithm for Approximate Max Flow in Undirected Graphs, and its Multicommodity Generalizations". In: *SODA'14: Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 217–226 (cit. on pp. 3346, 3361, 3362).

Ioannis Koutis, Gary L. Miller, and Richard Peng (2010). "Approaching optimality for solving SDD systems". In: *FOCS'10: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, pp. 235–244 (cit. on p. 3356).

– (2011). "A Nearly $m \log n$-Time Solver for SDD Linear Systems". In: *FOCS'11: Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*, pp. 590–598 (cit. on p. 3356).

Rasmus Kyng, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Daniel A. Spielman (2016). "Sparsified Cholesky and Multigrid Solvers for Connection Laplacians". In: *STOC'16: Proceedings of the 48th Annual ACM Symposium on Theory of Computing* (cit. on p. 3356).

Rasmus Kyng and Sushant Sachdeva (2016). "Approximate Gaussian Elimination for Laplacians: Fast, Sparse, and Simple". In: *FOCS'16: Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science* (cit. on p. 3356).

Yin Tat Lee, Satish Rao, and Nikhil Srivastava (2013). "A New Approach to Computing Maximum Flows using Electrical Flows". In: *STOC'13: Proceedings of the 45th Annual ACM Symposium on the Theory of Computing*, pp. 755–764 (cit. on pp. 3346, 3359).

Yin Tat Lee and Aaron Sidford (2014). "Path Finding Methods for Linear Programming: Solving Linear Programs in $\tilde{O}(\sqrt{rank})$ Iterations and Faster Algorithms for Maximum Flows". In: *FOCS'14: Proceedings of the 55th Annual IEEE Symposium on Foundations of Computer Science*, pp. 424–433 (cit. on pp. 3346, 3368, 3369).

Aleksander Mądry (2010). "Fast Approximation Algorithms for Cut-based Problems in Undirected Graphs". In: *FOCS'10: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, pp. 245–254 (cit. on p. 3362).

– (2011). "From Graphs to Matrices, and Back: New Techniques for Graph Algorithms". PhD thesis. Massachusetts Institute of Technology (cit. on p. 3363).

– (2013). "Navigating Central Path with Electrical Flows: from Flows to Matchings, and Back". In: *FOCS'13: Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science*, pp. 253–262 (cit. on pp. 3346, 3367–3369).

– (2016). "Computing Maximum Flow with Augmenting Electrical Flow". In: *FOCS'16: Proceedings of the 57th Annual IEEE Symposium on Foundations of Computer Science*, pp. 593–602 (cit. on pp. 3346, 3367, 3369).

Arkadii Nemirovskii, David Borisovich Yudin, and Edgar Ronald Dawson (1983). *Problem complexity and method efficiency in optimization*. Wiley (cit. on p. 3348).

Yurii Nesterov (1983). "A method of solving a convex programming problem with convergence rate $O(1/k^2)$". *Soviet Mathematics Doklady* 27.2, pp. 372–376 (cit. on pp. 3352, 3354).
– (2004). *Introductory lectures on convex optimization*. Vol. 87. Applied Optimization. A basic course. Kluwer Academic Publishers, Boston, MA, pp. xviii+236. MR: 2142598. IA: springer_10.1007-978-1-4419-8853-9 (cit. on p. 3348).
– (2005). "Smooth minimization of non-smooth functions". *Mathematical Programming* 103.1, pp. 127–152 (cit. on pp. 3352, 3354, 3357).
Yurii Nesterov and Arkadi Nemirovskii (1994). *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics (cit. on pp. 3366, 3368).
Jorge Nocedal and Stephen Wright (2000). *Numerical Optimization*. Springer New York (cit. on p. 3348).
Richard Peng (2016). "Approximate Undirected Maximum Flows in $O(mpolylog(n))$ Time". In: *SODA'16: Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1862–1867 (cit. on p. 3362).
Serge A. Plotkin, David B. Shmoys, and Eva Tardos (1995). "Fast Approximation Algorithms for Fractional Packing and Covering Problems". *Mathematics of Operations Research* 20, pp. 257–301 (cit. on p. 3357).
James Renegar (1988). "A polynomial-time algorithm, based on Newton's method, for linear programing". *Mathematical Programming* 40, pp. 59–93 (cit. on p. 3367).
Alexander Schrijver (2002). "On the history of the transportation and maximum flow problems". *Mathematical Programming* 91, pp. 437–445 (cit. on p. 3345).
– (2003). *Combinatorial Optimization: Polyhedra and Efficiency*. Springer (cit. on p. 3345).
Jonah Sherman (2009). "Breaking the Multicommodity Flow Barrier for $O(\sqrt{\log n})$-Approximatio to Sparsest Cuts". In: *FOCS'09: Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, pp. 363–372 (cit. on p. 3345).
– (2013). "Nearly Maximum Flows in Nearly Linear Time". In: *FOCS'13: Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science*, pp. 263–269 (cit. on pp. 3346, 3361, 3362).
– (2017a). "Area-convexity, ℓ_∞ regularization, and undirected multicommodity flow". In: *STOC'17: Proceedings of the 49th Annual ACM Symposium on Theory of Computing*, pp. 452–460 (cit. on pp. 3346, 3362).
– (2017b). "Generalized Preconditioning and Undirected Minimum-Cost Flow". In: *SODA'17: Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 772–780 (cit. on p. 3369).
Daniel A. Spielman and Nikhil Srivastava (2008). "Graph sparsification by effective resistances". In: *STOC'08: Proceedings of the 40th Annual ACM Symposium on the Theory of Computing*, pp. 563–568 (cit. on p. 3359).

Daniel A. Spielman and Shang-Hua Teng (2003). "Solving Sparse, Symmetric, Diagonally-Dominant Linear Systems in Time $O(m^{1.31})$". In: *FOCS'03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pp. 416–427 (cit. on p. 3356).

– (2004). "Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems". In: *STOC'04: Proceedings of the 36th Annual ACM Symposium on the Theory of Computing*, pp. 81–90 (cit. on p. 3356).

– (2008). "Spectral Sparsification of Graphs". *CoRR* abs/0808.4134 (cit. on p. 3359).

Pravin M. Vaidya (n.d.). "Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners". Unpublished manuscript, UIUC 1990. A talk based on the manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991, Mineapolis. (cit. on p. 3356).

– (1989). "Speeding-up linear programming using fast matrix multiplication". In: *FOCS'89: Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pp. 332–337 (cit. on p. 3368).

Stephen J. Wright (1997). *Primal-Dual Interior-Point Methods*. Society for Industrial and Applied Mathematics (cit. on pp. 3364, 3366).

Yinyu Ye (1997). *Interior Point Algorithms: Theory and Analysis*. John Wiley & Sons (cit. on pp. 3364, 3366).

N. Young (2001). "Sequential and Parallel Algorithms for Mixed Packing and Covering". In: *FOCS'01: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science* (cit. on p. 3357).

Neal E. Young (1995). "Randomized rounding without solving the linear program". In: *SODA'95: Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, pp. 170–178 (cit. on p. 3357).

Received 2018-03-05.

Aleksander Mądry
MIT
madry@mit.edu