2018 IEEE 59th Annual Symposium on Foundations of Computer Science

Coordinate Methods for Accelerating ℓ_∞ Regression and Faster Approximate Maximum Flow

Aaron Sidford
Department of Management Science & Engineering
Stanford University Stanford, CA USA
Email: sidford@stanford.edu

Kevin Tian
Department of Computer Science
Stanford University Stanford, CA USA
Email: kjtian@stanford.edu

***Abstract*—In this paper we provide faster algorithms for approximately solving ℓ_∞ regression, a fundamental problem prevalent in both combinatorial and continuous optimization. In particular we provide an accelerated coordinate descent method which converges in k iterations at a $O\left(\frac{1}{k}\right)$ rate independent of the dimension of the problem, and whose iterations can be implemented cheaply for many structured matrices. Our algorithm can be viewed as an alternative approach to the recent breakthrough result of Sherman [She17] which achieves a similar running time improvement over classic algorithmic approaches, i.e. smoothing and gradient descent, which either converge at a $O\left(\frac{1}{\sqrt{k}}\right)$ rate or have running times with a worse dependence on problem parameters. Our running times match those of [She17] across a broad range of parameters and in certain cases, improves upon it.**

We demonstrate the efficacy of our result by providing faster algorithms for the well-studied maximum flow problem. We show how to leverage our algorithm to achieve a runtime of $\tilde{O}\left(m + \frac{\sqrt{ns}}{\epsilon}\right)$ to compute an ϵ-approximate maximum flow, for an undirected graph with m edges, n vertices, and where s is the squared ℓ_2 norm of the congestion of any optimal flow. As $s = O(m)$ this yields a running time of $\tilde{O}\left(m + \frac{\sqrt{nm}}{\epsilon}\right)$, generically improving upon the previous best known runtime of $\tilde{O}\left(\frac{m}{\epsilon}\right)$ in [She17] whenever the graph is slightly dense. Moreover, we show how to leverage this result to achieve improved exact algorithms for maximum flow on a variety of unit capacity graphs.

We achieve these results by providing an accelerated coordinate descent method capable of provably exploiting dynamic measures of coordinate smoothness for smoothed versions of ℓ_∞ regression. Our analysis leverages the structure of the Hessian of the smoothed problem via a simple bound on its trace, as well as techniques for exploiting column sparsity of the constraint matrix for faster sampling and improved smoothness estimates. We hope that the work of this paper can serve as an important step towards achieving even faster maximum flow algorithms.

***Keywords*-ℓ_∞ regression; structured linear programming; accelerated coordinate descent; maximum flow**

I. Introduction

The classical problem of ℓ_∞ *regression* corresponds to finding a point x^* such that

$$x^* = \text{argmin}_{x \in \mathbb{R}^m} \|Ax - b\|_\infty, A \in \mathbb{R}^{n \times m}, \text{ and } b \in \mathbb{R}^n$$

In this work, we are primarily concerned with developing iterative algorithms for the approximate minimization of ℓ_∞ regression. We use OPT to denote $\|Ax^* - b\|_\infty$ and our goal is to find an ϵ-approximate minimizer to the ℓ_∞-regression function, i.e. a point $x \in \mathbb{R}^m$ such that

$$\mathsf{OPT} \leq \|Ax - b\|_\infty \leq \mathsf{OPT} + \epsilon$$

This regression problem has fundamental implications in many areas of statistics and optimization [She13], [LS14], [LS15a], [SWWY18]. In many of these settings, it is also useful to design iterative method machinery for the following more general problem of *box-constrained ℓ_∞ regression*:

$$x^* = \text{argmin}_{x \in S} \|Ax - b\|_\infty, A \in \mathbb{R}^{n \times m}, b \in \mathbb{R}^n$$

with $S = \{x \in \mathbb{R}^m : x_i \in [l_i, r_j] \ \forall j \in [m]\}$ for some m pairs of scalar $l_j \leq r_j$ (possibly infinite). Note that the constrained optimization problem is strictly more general than the standard one as setting $l_j \equiv -\infty, r_j \equiv \infty$ recovers the ℓ_∞ regression problem. For this work, the domain constraint will only be of the form $x \in B_\infty^c$ where we use B_∞^c to denote $[-c, c]^m$ for some constant $c > 0$ (though our results apply to the more general case). For short, we will simply refer to this problem as the *constrained ℓ_∞ regression* problem.

Many natural optimization problems which arise from combinatorial settings can be written in the form of constrained ℓ_∞ regression, such as the maximum flow problem and other structured linear programs, and thus faster methods for solving regression can imply faster algorithms for common problems in theoretical computer science. Therefore, the central goal of this paper is to provide faster algorithms for computing ϵ-approximate minimizers to constrained ℓ_∞-regression, that when specialized to the maximum flow problem, achieve faster running times.

A. Regression Results

In this paper we show how to apply ideas from the literature on accelerated coordinate descent methods to provide faster algorithms for approximately solving constrained ℓ_∞ regression. We show that by assuming particular sampling and smoothness oracles or sparsity assumptions on A we can obtain a randomized algorithm which improves upon the the classic gradient descent based methods across a broad range of parameters and attains an $\frac{1}{\epsilon}$ dependence in the runtime. Formally, we show the following:

Theorem I.1 (Accelerated constrained ℓ_∞ regression)**.** *There is an algorithm initialized at x^0 that ϵ-approximately minimizes the box-constrained ℓ_∞ regression problem (for any c)*

$$\min_{x \in B_\infty^c} \|Ax - b\|_\infty$$

$\tilde{O}\left(\frac{\sqrt{md}\|A\|_\infty \|x^0 - x^*\|_2}{\epsilon}\right)$ *iterations provided each column of $A \in \mathbb{R}^{n \times m}$ has at most d non-zero entries. Furthermore, after nearly linear preprocessing each iteration can be implemented in $\tilde{O}(d^2)$ time.*

In particular, note that when no column of A has more than $\tilde{O}(1)$ nonzero entries, each iteration can be implemented in $\tilde{O}(1)$ time. The only other known algorithm for achieving such a $\frac{1}{\epsilon}$ dependence in running time for ℓ_∞ regression, that improves upon the classic $\frac{1}{\epsilon^2}$ time without paying a running time penalty in terms of dimension or domain size, is the recent breakthrough result of [She17]. Our running times match those of [She17] across a broad range of parameters, and in certain cases improve upon it, due to our algorithm's tighter dependence on the ℓ_2-norm and therefore sparsity of the optimal solution, as well as a more fine-grained dependence on the problem's smoothness parameters. Because of these tighter dependences, in many parameter regimes including the maximum flow problem for even slightly dense graphs, our result improves upon [She17].

Interestingly, our work provides an alternative approach to [She17] for accelerating ℓ_∞ gradient descent for certain highly structured optimization problems, i.e. ℓ_∞ regression. Whereas Sherman's work introduced an intriguing notion of area convexity and new regularizations of ℓ_∞ regression, our results are achieved by working with the classic smoothing of the ℓ_∞ norm and by providing a new variant of accelerated coordinate descent. We achieve our tighter bounds by exploiting local smoothness properties of the problem and dynamically sampling by these changing smoothnesses.

Our algorithm leverages recent advances in non-uniform sampling for accelerated coordinate descent [AQRY16], [QR16], [NS17] and is similar in spirit to work on accelerated algorithms for approximately solving packing and covering linear programs [AO15] which too works with non-standard notions of smoothness. Our paper overturns some conventional wisdom that these techniques do not extend nicely to ℓ_∞ regression and the maximum flow problem. Interestingly, our algorithms gain an improved dependence on dimension and sparsity over [She17] in certain cases while losing the parallelism of [She17]. It is an open direction for future work as to see whether or not these approaches can be combined for a more general approach to minimizing ℓ_∞-smooth functions.

B. Maximum Flow Results

The classical problem of maximum flow roughly asks for a graph G with m (capacitated) edges and n vertices, with a specified source node and sink node, how to send as many units of flow from the source to the sink while preserving conservation at all other vertices and without violating edge capacity constraints (i.e. the flow cannot put more units on an edge than the edge's capacity).

The maximum flow problem is known to be easily reducible to the more general problem of *minimum congestion flow*. Instead of specifying s and t this problem takes as input a vector $d \in \mathbb{R}^V$ such that $d^\top \mathbb{1} = 0$, where $\mathbb{1}$ is the all-ones vector. The goal of minimum congestion flow is to find a flow $f \in \mathbb{R}^E$ which routes d meaning, mean that the imbalance of f at vertex v is given by d_v, and subject to this constraint minimizes the *congestion*,

$$\max_{e \in E(G)} \left| \frac{f_e}{c_e} \right|$$

where f_e is the flow on some edge, and c_e is the capacity on that edge. We refer to the vector with entries $\frac{f_e}{c_e}$ as the *congestion vector*.

A recent line of work beginning in [She13], [KLOS14] solves the maximum flow problem by further reducing to constrained ℓ_∞ regression. To give intuition for the reduction used in this work, broadly inspired by [She13], [KLOS14], we note that maximum flow in uncapacitated graphs can be rephrased as asking for the smallest congestion of a feasible flow, namely to solve the problem

$$f^* = \text{argmin}_{Bf=d} \|f\|_\infty$$

where the restriction $Bf = d$ for B the edge-vertex incidence matrix of a graph, and d the demands, enforces the flow constraints. This can be solved up to logarithmic factors in the running time by fixing some value F for $\|f\|_\infty$ and asking to optimally solve the problem

$$f^* = \text{argmin}_{\|f\|_\infty \le F} \|Bf - d\|_\infty$$

where we note that the constraint $\|f\|_\infty \le F$ can be decomposed as the indicator of a box so that this objective matches the form of Equation I. The exact reduction we use has a few modifications: the box constraint is more simply replaced by $\|f\|_\infty \le 1$, and the regression objective is in a matrix RB, where R is a combinatorially-constructed preconditioner whose goal is to improve the condition number (and convergence rate) of the problem, and the problem is scaled for capacitated graphs (for a more detailed description, see Section IV-B).

In this paper we show how to use our improved algorithms for structured ℓ_∞ regression in order to obtain faster algorithms for maximum flow. We do so by leveraging the tighter dependence on the domain size (in the ℓ_2 norm rather than ℓ_∞) and coordinate smoothness properties of the function to be minimized (due to the structure of the regression matrix). In particular we show the following.

Theorem I.2 (ℓ_2 accelerated maximum flow)**.** *There is an algorithm that takes time $\tilde{O}\left(m + \frac{\sqrt{ns}}{\epsilon}\right)$ to find an ϵ-approximate maximum flow, where s is ratio of the the ℓ_2 norm squared of the congestion vector of any optimal flow to the congestion of that flow.*

Our running time improves upon the previous fastest running time of this problem of $\tilde{O}(\frac{m}{\epsilon})$. Since $s \leq m$ we achieve a faster running time whenever the graph is slightly dense, i.e. $m = \Omega(n^{1+\delta})$ for any constant $\delta > 0$.

Interestingly our algorithm achieves even faster running times when there is a sparse optimal flow. Leveraging this we provide several new results on exact undirected and directed maximum flow on uncapacitated graphs as well.

Theorem I.3 (Improved exact maximum flows)**.** *There are algorithms for finding an exact maximum flow in the following types of uncapacitated graphs.*

- *There is an algorithm which finds a maximum flow in an undirected, uncapacitated graph in time $\tilde{O}(ms^{1/4}n^{1/4})$.*
- *There is an algorithm which finds a maximum flow in an undirected, uncapacitated graph with maximum flow value F in time $\tilde{O}(m + \min(\sqrt{mn}F^{3/4}, m^{3/4}n^{1/4}\sqrt{F}))$.*
- *There is an algorithm which finds a maximum flow in an undirected, uncapacitated graph with a maximum flow that uses at most s edges in time $\tilde{O}(m+m^{1/2}n^{1/4}s^{3/4})$.*
- *There is an algorithm which finds a maximum flow in a directed, uncapacitated graph in time $\tilde{O}(mn^{1/4}s^{1/4})$.*

Each of these runtimes improves upon previous work in some range of parameters. For example, the bound of $\tilde{O}(m + m^{3/4}n^{1/4}\sqrt{F})$ for undirected, uncapacitated graphs improves upon the previous best running times of $\tilde{O}(m\sqrt{F})$ achievable by [She17] whenever $n = o(m)$ and of $\tilde{O}(m + nF)$ achievable by [KL02] whenever $m = o(nF^{2/3})$.

We also separately include the following result (which has no dependence on the sparsity s) for finding exact flows in general uncapcitated directed graphs, as it improves upon the running time of $\tilde{O}(m \cdot \max\{m^{1/2}, n^{2/3}\})$ achieved by [GR98] whenever $m = \omega(n)$ and $m = o(n^{5/3})$.

Theorem I.4 (Directed uncapacitated graphs)**.** *There is an algorithm which finds a maximum flow in a directed, uncapacitated graph in time $\tilde{O}(m^{5/4}n^{1/4})$.*

Although the runtime of [GR98] has been improved upon by the recent works of [Mad13] achieving a running time of $O(m^{10/7})$ and of [LS14] achieving a running time $\tilde{O}(m\sqrt{n})$, which dominate our $\tilde{O}(m^{5/4}n^{1/4})$ running time, they do

Author	Method	Complexity	Cost	Norm
[Nes05]	Smoothing	$O(\epsilon^{-2})$ $O(\epsilon^{-1})$	$O(m)$ $O(m)$	ℓ_∞ ℓ_2
[Nem04]	Mirror prox	$O(\epsilon^{-2})$	$O(m)$	ℓ_∞
[Nes07]	Dual extrapolation	$O(\epsilon^{-2})$ $O(m\epsilon^{-1})$	$O(m)$ $O(m)$	ℓ_∞ ℓ_∞
[She17]	Area-convexity	$O(\epsilon^{-1})$	$O(m)$	ℓ_∞
This paper	Local smoothness	$O(\sqrt{md}\epsilon^{-1})$	$\tilde{O}(d^2)$	ℓ_2

Table I
DEPENDENCIES OF ALGORITHMS FOR ℓ_∞ REGRESSION IN $A \in \mathbb{R}^{n\times m}$ ON PROBLEM PARAMETERS. NOTE THERE IS UP TO AN $O(\sqrt{m})$ DISCREPANCY BETWEEN THE ℓ_2 AND ℓ_∞ NORMS. d IS THE MAXIMUM NUMBER OF NONZERO ENTRIES IN ANY COLUMN OF A, COMPLEXITY IS THE NUMBER OF ITERATIONS, AND COST IS PER ITERATION.

Author	Complexity	Weighted	Directed
[GR98]	$\tilde{O}(\min(m^{3/2}, mn^{2/3}))$	Yes	Yes
[Kar98]	$\tilde{O}(m\sqrt{n}\epsilon^{-1})$	Yes	No
[KL02]	$\tilde{O}(m + nF)$	Yes	No
[CKM+11]	$\tilde{O}(mn^{1/3}\epsilon^{-11/3})$	Yes	No
[LRS13]	$\tilde{O}(mn^{1/3}\epsilon^{-2/3})$	No	No
[She13]	$\tilde{O}(m^{1+o(1)}\epsilon^{-2})$	Yes	No
[KLOS14]	$\tilde{O}(m^{1+o(1)}\epsilon^{-2})$	Yes	No
[Mad13]	$\tilde{O}(m^{10/7})$	No	Yes
[LS14]	$\tilde{O}(mn^{1/2})$	Yes	Yes
[Pen16]	$\tilde{O}(m\epsilon^{-2})$	Yes	No
[She17]	$\tilde{O}(m\epsilon^{-1})$	Yes	No
This paper	$\tilde{O}(m + \sqrt{ns}\epsilon^{-1})$	Yes	No

Table II
COMPLEXITY OF MAXIMUM FLOW ALGORITHMS SINCE [GR98] FOR GRAPHS WITH n VERTICES, m EDGES, WHERE s AND F ARE THE ℓ_2^2 OF THE CONGESTION AND VALUE OF THE MAXIMUM FLOW.

it using sophisticated advances in interior point methods, whereas our algorithm operates using a *first-order method* which only queries gradient information of the objective function, rather than second-order Hessian information. In particular, our algorithm is the first to improve runtimes for directed graphs while relying only on first-order information of the objective function. We find it interesting that our result achieves any running time improvement for unit capacity maximum flow over [GR98] without appealing to interior point machinery and think this may motivate further research in this area, namely designing first-order methods for structured linear programs.

For a more comprehensive discussion of the histories of ℓ_∞ regression, first-order methods for convex optimization, and the maximum flow problem, we refer the reader to the full version of our paper [ST18].

C. Organization

The rest of this paper is organized as follows.

- **Section II: Overview.** We introduce the definitions and notation we use throughout the paper, and give a general framework motivating our work.

- **Section III: Regression.** We give a framework for accelerated randomized algorithms which minimize the constrained ℓ_∞ regression function based on non-uniform sampling which assumes access to a coordinate smoothness and sampling oracle, then give efficient implementations for these oracles for structured problems.
- **Section IV: Maximum Flow.** We show how to attain a faster algorithm for maximum flow by providing implementations for the oracles via exploiting combinatorial structure of the flow regression problem. Furthermore, we give the exact maximum flow runtimes achieved via rounding the resulting approximate flow of our method.

II. Overview

A. Basic Definitions

General Notations. We use $\tilde{O}(f(n))$ to denote runtimes of the following form: $O(f(n)\log^c f(n))$ where c is a constant. With an abuse of notation, we let $\tilde{O}(1)$ denote runtimes hiding polynomials in $\log n$ when the variable n is clear from context, and refer to such runtimes as "nearly constant."

Generally, we work with functions whose arguments are vector-valued variables in m-dimensional space, and may depend on a linear operator $A : \mathbb{R}^m \to \mathbb{R}^n$. Correspondingly we use $j \in [m]$ and $i \in [n]$ to index into these sets of dimensions, where $[m]$ is the set $\{1, 2, \ldots n\}$. We use e_j to denote the standard basis vector, i.e. the vector in $\mathbb{R}^m$ which is 1 in dimension j and 0 everywhere else. We use $u \circ v$ to denote the vector whose j^{th} coordinate is $u_j v_j$.

Matrices. Generally in this work, we will be dealing with matrices $A \in \mathbb{R}^{n\times m}$ unless otherwise specified. Accordingly, we index into rows of A with $i \in [n]$, and into columns with $j \in [m]$. We refer to rows of A via $A_{i:}$ or a_i when it is clear from context, and columns via $A_{:j}$.

We use diag(w) to denote the diagonal matrix whose diagonal entries are the coordinates of a vector w. We call a square symmetric matrix A positive semi-definite if for all vectors x, $x^\top Ax \geq 0$ holds. For positive semi-definite matrices A, B we apply the Loewner ordering and write $A \preceq B$ if for all vectors x, $x^\top Ax \leq x^\top Bx$ holds.

Finally, we say that a matrix is d-column-sparse if no column of A has more than d nonzero entries.

Norms. We use $\|\cdot\|$ to denote an arbitrary norm when one is not specified. For scalar valued $p \geq 1$, including $p = \infty$, we use $\|x\|_p \stackrel{\text{def}}{=} (\sum_j x_j^p)^{1/p}$ to denote the ℓ_p norm. For vector valued $w \in \mathbb{R}^m_{\geq 0}$, we use $\|x\|_w^2 \stackrel{\text{def}}{=} \sum_j w_j x_j^2$ to denote the weighted quadratic norm.

For a norm $\|\cdot\|$, we write the dual norm as $\|\cdot\|_*$, defined as $\|x\|_* \stackrel{\text{def}}{=} \max_{\|y\|\leq 1} y^\top x$. It is well known that the dual norm of ℓ_p is ℓ_q for $1 = \frac{1}{p} + \frac{1}{q}$. For a matrix A and a vector norm $\|\cdot\|$, we correspondingly define the matrix norm $\|A\| \stackrel{\text{def}}{=} \max_{\|x\|=1} \|Ax\|$.

Functions. This paper is concerned with minimizing convex functions $f(x)$ subject to a box constraint, where the domain will be $x \in B_\infty^c$ unless otherwise specified. Whenever the function is clear from context, x^* will refer to any minimizing argument of the function. We use the term ϵ-approximate minimizer of a function f to mean any point x such that $f(x^*) \leq f(x) \leq f(x^*) + \epsilon$.

For differentiable functions f we let $\nabla f(x)$ be the gradient and let $\nabla^2 f(x)$ be the Hessian. We let $\nabla_j f(x)$ be the value of the j^{th} partial derivative; we also abuse notation and use it to denote the vector $\nabla_j f(x) e_j$ when it is clear from context.

Properties of functions. We say that a function is L-smooth with respect to some norm $\|\cdot\|$ if it obeys $\|\nabla f(x) - \nabla f(y)\|_* \leq L\|x - y\|$, the dual norm of the gradient is Lipschitz continuous. It is well known in the optimization literature that when f is convex, this is equivalent to $f(y) \leq f(x) + \nabla f(x)^\top(y-x) + \frac{L}{2}\|y-x\|^2$ for $y, x \in \text{dom}(f)$ and, for twice-differentiable f, $z^\top \nabla^2 f(x) z \leq \frac{L}{2}\|z\|^2$.

We say that a function is L_j-coordinate smooth in the j^{th} coordinate if the restriction of the function to the coordinate is smooth, i.e. $|\nabla_j f(x + ce_j) - \nabla_j f(x)| \leq L_j|c|$ $\forall x \in \text{dom}(f), c \in \mathbb{R}$. Equivalently, for twice-differentiable convex f, $\nabla^2_{jj} f(x) \leq L_j$.

Graphs. We primarily study capacitated undirected graphs $G = (V, E, c)$ with edge set $E \subseteq V \times V$, edge capacities $c : E \to \mathbb{R}_+$. When referring to graphs, we let $m = |E|$ and $n = |V|$. Throughout this paper, we assume that G is strongly connected.

We associate the following matrices with the graph G. The matrix of edge weights $C \in \mathbb{R}^{E\times E}$ is defined as $C \stackrel{\text{def}}{=} \text{diag}(c)$. Orienting the edges of the graph arbitrarily, the vertex-edge incidence matrix $B \in \mathbb{R}^{V\times E}$ is defined as $B_{s,(u,v)} \stackrel{\text{def}}{=} -1$ if $s = u$, 1 if $s = v$ and 0 otherwise. Finally, define the Laplacian matrix $L \in \mathbb{R}^{V\times V} \stackrel{\text{def}}{=} BCB^\top$.

B. Overview of our algorithms

Here, we give an overview of the main ideas used in our algorithms for approximately solving ℓ_∞ regression problems. The main ideological contribution of this work is that it uses a new variation of coordinate descent which uses the novel concept of *local coordinate smoothness* in order to get tighter guarantees for accelerated algorithms. The result for ℓ_∞ regression in particular follows from a bound on the local coordinate smoothnesses for an ℓ_∞-smooth function, which is described in full detail in Section III-D. Finally, in order to implement the steps of the accelerated algorithm, it is necessary to efficiently compute overestimates to the square roots of the local coordinate smoothnesses, and furthermore sample coordinates proportional to these overestimates. This procedure is fully described in Lemma III.13.

Local coordinate smoothness. In this work, we introduce the concept of *local coordinate smoothness* at a point x. This generalizes the concept of global coordinate smoothness to a particular point. This definition is crucial to the analysis throughout the rest of the paper.

Definition II.1 (Local coordinate smoothness)**.** *We say a function f is $L_j(x)$ locally coordinate smooth in coordinate j at a point x, if for $|c| \le \left|\frac{\nabla_j f(x)}{L_j(x)}\right|$, $|\nabla_j f(x + ce_j) - \nabla_j f(x)| \le L_j(x)|c|$ $\forall x \in \mathrm{dom}(f), c \in \mathbb{R}$. Equivalently, for differentiable convex f, for y between $x \pm \frac{1}{L_j(x)}\nabla_j f(x)$, $f(y) \le f(x) + \nabla f_j(x)(y_j - x_j) + \frac{L_j(x)}{2}|y_j - x_j|^2$.*

The proof of this fact is the same as the proof of equivalence for the standard definition of smoothness, restricted to an interval. Note that this says that a coordinate descent step using the local smoothness at a point exhibits essentially the same behavior as a single step of coordinate descent with global smoothnesses. In particular, for the point which the coordinate descent algorithm would step to, the function values exhibit the same quadratic-like upper bound along the coordinate. We will drop the x from the notation $L_j(x)$ when the point we are discussing is clear, i.e. a particular iterate of one of our algorithms.

Bounding the progress of coordinate descent in ℓ_∞-smooth functions. Here, we sketch the main idea underlying our accelerated methods. Why is it possible to hope to accelerate gradient methods in the ℓ_∞ norm via coordinate descent? One immediate reason is that smoothness in this norm is a strong assumption on the sum S of the local coordinate smoothness values of f.

It is well known in the literature that gradient descent for a ℓ_∞-smooth function initialized at $x^0 \in \mathbb{R}^m$ takes roughly $\frac{L\|x^0 - x^*\|_\infty^2}{\epsilon}$ iterations to converge to a solution which has ϵ additive error, whereas coordinate descent with appropriate sampling probabilities $\frac{L_j}{S}$, for $S = \sum_j L_j$, takes $\frac{S\|x^0 - x^*\|_2^2}{\epsilon}$ iterations to converge to the same quality of solution.

Note that when the norm in the gradient descent method is $\|\cdot\|_\infty$, we have $\|x^0 - x^*\|_2^2 \le m\|x^0 - x^*\|_\infty^2$, but the iterates can be m times cheaper because they do not require a full gradient computation. Furthermore, the coordinate method can be accelerated. So, if we can demonstrate $S \le L$, we can hope to match and improve the runtime. Of course, there are several caveats: we can only implement such an algorithm if we are able to efficiently update overestimates the local smoothnesses and sample efficiently by them, issues which we will discuss later. To be more concrete, we will demonstrate the following fact.

Lemma II.2. *Suppose for some point x, $f : \mathbb{R}^m \to \mathbb{R}$ is convex and L-smooth with respect to $\|\cdot\|_\infty$, $\Lambda_j(x) = \nabla_{jj}^2 f(x)$, and $S = \sum_j \Lambda_j(x)$. Then $S \le L$.*

Proof: Throughout, fix x, and define $M \stackrel{\text{def}}{=} \nabla^2 f(x)$ and $S \stackrel{\text{def}}{=} \mathrm{Tr}(M)$. Consider drawing y uniformly at random from $\{-1, 1\}^m$. By the smoothness assumption, we have $y^\top M y \le L\|y\|_\infty^2 = L$. Also, note that

$$\mathbb{E}[y^\top M y] = \mathbb{E}\left[\sum_{i,j} M_{ij} y_i y_j\right] = \mathrm{Tr}(M) = S$$

Thus, by the probabilistic method, there exists some y such that $S \le y^\top M y \le L$, as desired. ■

While this gives a bound on the number of iterations required by a coordinate descent algorithm, it requires being able to compute and sample by the $L_j(x)$. Note that as we take coordinate descent steps, it is not clear how the local coordinate smoothnesses $L_j(x^k)$ will change, and how to update and compute them. Naively, at each iteration, we could recompute the local smoothnesses, but this requires as much work as a full gradient computation if not more. Furthermore, we need to implement sampling the coordinates in an appropriate way, and show how the algorithm behaves under acceleration. However, a key idea in our work is that if we can take steps within regions where the smoothness values do not change by much, we can still make iterates computationally cheap, which we will show.

Implementation of local smoothness estimates. One useful property of coordinate descent is that as long as we implement the algorithm with overestimates to the *local smoothness* values, the convergence rate is still the same, with a dependence on the overestimates. Our full algorithm for ℓ_∞ regression proceeds by showing how to compute and sample proportional to slight overestimates to the local smoothnesses, for regression problems in a column-sparse matrix. We do so by first proving that the smooth approximation to ℓ_∞ regression admits local smoothnesses which can be bounded in a structured way, in Section III-B. Further, using a lightweight data structure, we are able to maintain these overestimates and sample by them in nearly-constant time, yielding a very efficient implementation, which we show in Lemma III.13.

Furthermore, in Section III-C and Section III-D, we use modified algorithms from the literature on accelerated proximal coordinate descent, adapted to our methods of local smoothness coordinate descent. Combining these pieces, we are able to give the full algorithm for ℓ_∞ regression.

In Section IV, we study the maximum flow problem as an example of a problem which can be reduced to ℓ_∞ regression in a column-sparse matrix. Using a careful analysis of bounds on the local smoothness values, we show that a direct application of our accelerated ℓ_∞ regression algorithm yields the fastest currently known approximate maximum flow algorithm.

III. Minimizing box-constrained $\|Ax - b\|_\infty$

We will now discuss how to turn the framework presented in the previous section into different specialized algorithms for the problem of box-constrained regression in the ℓ_∞ norm, and analyze the rates of convergence depending on the sampling probabilities associated with the (accelerated) coordinate descent method. Recall throughout that our goal is to compute an ϵ-approximate minimizer of the constrained ℓ_∞ regression problem at an $O(\frac{1}{\epsilon})$ rate.

Definition III.1 (Box-constrained ℓ_∞ regression problem)**.** *This is the problem of finding a minimizer to the function $\|Ax-b\|_\infty$, where the argument x has domain B_∞^c for some c.*

In the style of previous approaches to solving ℓ_∞ regression, because $\|x\|_\infty$ is not a smooth function, we choose to minimize a suitable smooth approximation instead. Intuitively, the $O(\frac{1}{\epsilon})$ rate comes from accelerating gradient descent for a function which is $O(\frac{1}{\epsilon})$-smooth. Therefore, the function error of the T^{th} iterate with respect to OPT is proportional to $O\left(\frac{1}{\epsilon T^2}\right)$, so if we wish for an ϵ-approximate minimizer, it suffices to pick $T = O(\frac{1}{\epsilon})$.

A. Smooth approximation to regression

In this section, we define the standard smooth approximation for ℓ_∞ regression in the literature used in the paper and provide some technical facts about it.

Definition III.2 (Soft-max)**.** *For all real valued vectors x we let* $\text{smax}_t(x) \stackrel{\text{def}}{=} t\log(\sum_j \exp(\frac{x_j}{t}))$.

Fact III.3 (smax Additive Error)**.** $\forall x \in \mathbb{R}^m$, $\max_{j\in[m]} x_j \leq \text{smax}_t(x) \leq t\log m + \max_{j\in[m]} x_j$

Definition III.4. *Inside the scope of the remainder of this section, let $f(x) \stackrel{\text{def}}{=} \text{smax}_t(Ax-b)$.*

Note that these properties say something about the quality of approximation smax provides on the maximum element of a vector, instead of its ℓ_∞ norm. However, we could have simply applied it to the vector in $\mathbb{R}^{2m}$ which has the first m coordinates the same as x and the last m the same as $-x$. For the regression problem, we could consider the minimization problem applied to f, for f defined with a proxy matrix $A' = \begin{pmatrix} A \\ -A \end{pmatrix}$ and a proxy vector $b' = \begin{pmatrix} b \\ -b \end{pmatrix}$. For notational convenience, we will focus on minimizing $f(x)$ defined above, but with $A \in \mathbb{R}^{n\times m}$ and $b \in \mathbb{R}^n$ in the original dimensionalities, which preserves all dependencies on the dimension and structural sparsity assumptions used later in this work up to a constant.

Furthermore, note that setting $t = \frac{\epsilon}{2\log m}$ and asking for an $\frac{\epsilon}{2}$-approximate minimizer of f is sufficient to finding an ϵ-approximate minimizer of the original regression problem. Thus, for notational convenience, we will fix $t = O(\frac{\epsilon}{\log m})$ for the remainder of this work, and concentrate on finding an ϵ-approximate minimizer of f, which up to constants approximately solves the original problem.

Next, we state some technical properties of our approximation. We drop the t from many definitions because the t we choose for all our methods is fixed.

Definition III.5. *For $x \in \mathbb{R}^m$ let $p(x) \in \mathbb{R}^m$ be defined as* $p_j(x) \stackrel{\text{def}}{=} \frac{\exp(x_j/t)}{\sum_{j'}\exp(x_{j'}/t)}$.

We note that as defined, the $p_j(x)$ for any x form a probability distribution. Moreover, they are defined in this way because they directly are used in the calculation of the gradient and Hessian of smax.

Fact III.6 (Soft-max Calculus)**.** *The gradient and Hessian of* smax *are as follows:* $\nabla\text{smax}_t(x) = p(x)$ *and* $\nabla^2\text{smax}_t(x) = \frac{1}{t}\textbf{diag}(p(x)) - vv^\top \preceq \frac{1}{t}\textbf{diag}(p(x))$, *for some vector v.*

These facts about smax can be verified by direct calculation. Now, taking a cue from our earlier naive analysis of coordinate descent, we wish to make steps in regions where the coordinate smoothnesses do not change by much. Thus, we will show the following key stability property in providing estimates to the coordinate smoothnesses. In particular, we demonstrate that within some box around our current iterate, the function exhibits good local coordinate smoothness behavior.

Lemma III.7. *For all $x, y \in \mathbb{R}^n$ with $\|y-x\|_\infty \leq t$, the following upper bound holds:*

$$\text{smax}_t(y) \leq \text{smax}_t(x) + \nabla\text{smax}_t(x)^\top(y-x) + \frac{4}{t}\|y-x\|_{p(x)}^2 .$$

Proof: For $\alpha \in [0,1]$, let $x^\alpha = x + \alpha(y-x)$. Then, we have

$$\text{smax}_t(y) = \text{smax}_t(x) + \nabla\text{smax}_t(x)^\top(y-x) + \int_0^1\int_0^\beta (y-x)^\top\nabla^2\text{smax}_t(x^\alpha)(y-x)d\alpha d\beta$$

However, based on Fact III.6, we know that $\nabla^2\text{smax}_t(x^\alpha) \preceq \frac{1}{t}\textbf{diag}(p(x_\alpha))$. Also, note that $p_j(x^\alpha) \leq 8p_j(x)$ for all j, α for the following reason: for $\|x^\alpha - x\|_\infty \leq t$ the numerator $\exp(\frac{x_j}{t})$ underestimates $\exp(\frac{x_j^\alpha}{t})$ by at most a factor of e, and the denominator similarly is an overestimate by at most a factor of e, so the discrepancy is at most a factor of $e^2 < 8$. So, $\nabla^2\text{smax}(x^\alpha) \preceq \frac{8}{t}\textbf{diag}(p(x))$. ■

B. Local coordinate smoothness estimates of the approximation

In order to analyze convergence rates of local smoothness coordinate descent algorithms, we need to provide estimates of the progress of a single step. Correspondingly to the classical approach to bounding progress of coordinate descent, we proceed by providing overestimates of the local coordinate smoothnesses at a given iterate. The next couple of lemmas prove, respectively, a lower bound on the progress of a single step via our overestimate of the local smoothness, and an upper bound on the sum of the smoothness overestimates. This allows us to analyze our later algorithm in full.

In particular, we claim that for the function f that we have defined, at a point x, choosing $L_j(x) = \frac{8}{t}\|A_{:j}\|_\infty p(Ax-b)^\top|A_{:j}|$ suffices as a local coordinate smoothness overestimate, where $|A_{:j}|$ is the absolute value applied entrywise on the column of A. For notational convenience, throughout the rest of this section we will fix a point x and use p^x to denote

$p(Ax-b)$, which is fixed for the duration of an iteration of coordinate descent. Though this estimate looks daunting, the intuition for it comes from fairly straightforward analysis of what we have already derived.

Lemma III.8. *Let $L_j(x) \geq \frac{8}{t} p^{x\top}(A_{:j})^2$, $L_j(x) \geq \frac{1}{t}\|A_{:j}\|_\infty |p^{x\top} A_{:j}|$, where $(A_{:j})^2$ refers to the vector obtained by squaring each entry of the column of A. Then, at x, f is $L_j(x)$ locally coordinate smooth. Consequently, $f(x - \frac{1}{L_j}\nabla_j f(x)) \leq f(x) - \frac{(\nabla_j f(x))^2}{2L_j}$.*

Proof: Firstly, we know that $\nabla f(x) = A^\top p^x$ and $\nabla^2 f(x) \preceq A^\top \mathbf{diag}(p^x) A$. Furthermore, we have the following upper bound based on the analysis of Lemma III.8: $f(x+\eta e_j)$ is at most

$$f(x) + \eta \nabla_j f(x) e_j + \frac{4}{t}\eta^2 (p^{x\top}(A_{:j})^2) \ , \ \forall |\eta| \|A_{:j}\|_\infty \leq t \tag{1}$$

Let $C, D = \frac{8}{t} e_j^\top A^\top \mathbf{diag}(p^x) A e_j, \frac{1}{t}|\nabla_j f(x)| \|A_{:j}\|_\infty$. We wish to prove that for $L_j(x) \geq \max(C, D)$, the guarantee holds. Note that setting $\eta = \left|\frac{-t\nabla_j f(x)}{|\nabla_j f(x)| \|A_{:j}\|_\infty}\right| = \left|\frac{-\nabla_j f(x)}{D}\right|$ is the largest step size we can take to stay within the box where given by Equation 2 holds. So if we take a step size of smaller absolute value than $\left|-\frac{\nabla_j f(x)}{L_j(x)}\right|$, since $L_j(x) \geq D$, certainly we stay within the valid region. Also, note that since $L_j(x)$ is an overestimate of C, the quadratic upper bound for $f(x+\eta e_j)$,

$$f(x) + \eta \nabla_j f(x) e_j + \frac{L_j(x)}{2}\eta^2 \ , \ \forall |\eta| \|A_{:j}\|_\infty \leq t \tag{2}$$

certainly holds. Consequently, f is $L_j(x)$-locally coordinate smooth at x. The progress bound of the step holds as an immediate corollary of local coordinate smoothness. ∎

It will prove to be useful that we can further use any overestimates to the $L_j(x)$ defined specifically in the statement of Lemma III.9. Indeed, we show later that using overestimates may yield more efficient sampling and smoothness oracles in implementation, without affecting runtime by more than a $\tilde{O}(1)$ factor. The next lemma gives the particular overestimates we use in our paper.

Lemma III.9. *Let $L_j(x) = \frac{8}{t} p^{x\top}|A_{:j}| \cdot \|A_{:j}\|_\infty$. Then, $L_j(x) \geq \max\{\frac{8}{t}p^{x\top}(A_{:j})^2, \frac{1}{t}\|A_{:j}\|_\infty |p^{x\top}A_{:j}|\}$, and $\sum_j L_j(x) \leq \frac{8\|A\|_\infty^2}{t}$.*

Proof: $L_j(x) \geq \frac{1}{t}\|A_{:j}\|_\infty |p^{x\top} A_{:j}|$ is obvious and follows from the triangle inequality. $L_j(x) \geq \frac{8}{t}\sum_i p_i^x (A_{ij}^2)$ follows from

$$\frac{8}{t}\sum_i p_i^x (A_{ij}^2) \leq \frac{8}{t}\sum_i p_i^x |A_{ij}| \cdot \|A_{:j}\|_\infty = L_j(x)$$

Second, we show that the sum of these estimates is not too large. Indeed, for any p^x, the sum up to scaling by $\frac{8}{t}$ is upper bounded by the following:

$$\max_{p \in \Delta^n} \sum_j p^\top |A_{:j}| \cdot \|A_{:j}\|_\infty \leq \|A\|_\infty^2 \tag{3}$$

∎

For the remainder of this section, whenever we refer to $L_j(x)$, unless otherwise specified, we will mean these particular overestimates $L_j(x) = \frac{8}{t}p^{x\top}|A_{:j}| \cdot \|A_{:j}\|_\infty$.

C. Acceleration analysis under local smoothness sampling and dual initializations

We now state the following theorem, adapted from [QR16], for performing coordinate descent under local smoothness estimates. We note that this adaptation differs from its original presentation in two ways, which may be of independent interest. Firstly, it is specifically stated for the use of local coordinate smoothnesses, a key contribution of this work. Furthermore, the convergence rate depends on two different points, x^0 and z^0, which are initializations of a primal and dual variable respectively. This allows us to obtain a better dependency on the domain size of the problem, because our new algorithm pays a domain dependence only on the initial dual point, whereas the original algorithm pays a domain dependence with regards to the worst dual point, giving us a better dependence on the sparsity of the solution. The proof is very similar to its original form in [QR16]. We include the proof in the full version of our paper [ST18].

Theorem III.10 (Local smoothness acceleration)**.** *For some choice of c, let $F(x)$ be the convex function defined such that $F(x) = f(x)$ for $x \in B_\infty^c$ and is ∞ otherwise. Further assume that there is a global constant $S_{1/2}$, and values $L_j(x)$ at every point x, such that for each point x, f is $L_j(x)$ locally coordinate smooth in the j^{th} coordinate at the point, and $S_{1/2} \geq \sum_j \sqrt{L_j(x)}$. There is an algorithm,* AP-CD*, initialized at some primal point x^0 and some dual point z^0 both in B_∞^c, which in T iterations returns a point x^T such that $F(x^T) - F(x^*) \leq$*

$$O\left(\max\left\{\frac{m^2(F(x^0)-F(x^*))}{T^2}, \frac{S_{1/2}^2\|z^0 - x^*\|_2^2}{T^2}\right\}\right)$$

Each iteration of the algorithm requires (1) maintaining the sum $\sum_j \sqrt{L_j(x)}$, (2) sampling from the distribution $\{p_j \propto \sqrt{L_j(x)}\}$, (3) computing the local smoothness $\sqrt{L_j(x)}$ for the coordinate sampled, and (4) solving the constrained minimization problem $\text{argmin}_{y\in[-c,c]}\{\langle \xi, y\rangle + \eta|y_j - x_j|^2\}$ for some vector-valued ξ, x.

In particular, assuming nearly-constant time solutions to the subproblems (1), (2), (3) and (4), each iteration takes nearly-constant time to implement. In our setting of interval constraints, it is easy to see that there is a constant time implementation for (4). Thus, the bottleneck for a full implementation of this algorithm usually is the efficient sampling / smoothness calculation step. In Lemma III.13, we show that all these steps are implementable in nearly-constant time for ℓ_∞ regression in a column-sparse matrix.

D. Accelerated constrained ℓ_∞ regression: reducing the iteration count

We are now ready to put the tools we have together to provide runtime guarantees for the constrained regression problem. We assume as an input $D \stackrel{\text{def}}{=} f(x^0) - f(x^*)$, the function distance to OPT from some initial point x^0, which is polynomially bounded for all our applications, and $s \stackrel{\text{def}}{=} \left\|x^0 - x^*\right\|_2^2$, the squared ℓ_2 distance from the minimizer, which we can obtain via a binary search. Recall that per the guarantees in Theorem III.10, we can guarantee that in T iterations of AP-CD initialized at x^0, z^0, we can guarantee a point x^T such that $F(x^T) - F(x^*) \leq$

$$2\max\left(C_1\frac{m^2 D}{T^2}, C_2\frac{S_{1/2}^2\left\|z^0 - x^*\right\|_2^2}{T^2}\right)$$

for some globally computable $C_1, C_2 \in \tilde{O}(1)$, where $S_{1/2}$ is some upper bound on the sum of square roots of the local coordinate smoothnesses of f at any point. The high-level idea is that we will run $O(\log\frac{D}{\epsilon})$ phases of AP-CD for T iterations. At the end of the k^{th} phase, we will either ϵ-approximately minimize the original function, or obtain a point $x^{k,T}$ which has halved the function error with respect to $x^{k,0}$. For the next phase, we will set the primal initial point $x^{k+1,0} = x^{k,T}$ to be the point we obtained, but we will reset the dual point to be $z^{0,0}$, the very first dual point. In this way, the domain dependence we pay is only with respect to the first dual point, but the function error we obtain halves each round.

Theorem III.11. Accel-Regress *initialized at x^0 computes an ϵ-approximate minimizer to the regression problem $F(x) = \|Ax - b\|_\infty$ constrained to B_∞^c for some c in*

$$\tilde{O}\left(\max\left\{\frac{\sqrt{m}\|A\|_\infty\|x^0 - x^*\|_2}{\epsilon}, m\log\frac{1}{\epsilon}\right\}\right)$$

iterations, using AP-CD *as a subroutine.*

We prove the correctness of our accelerated algorithm in the full paper [ST18], but remark that it simply follows from the bounds on $S_{1/2}$ via Lemma III.9 and the fact that in each iteration, either the function error is halved or we obtain an ϵ-approximate minimizer directly. By re-initializing the dual starting point at every phase, we obtain tighter dependence on the sparsity of the optimal solution.

We remark that the accelerated algorithm always has a better iteration count than the unaccelerated l_∞ gradient descent method, for matrices A with $\|A\|_\infty \geq \Omega(1)$ and sufficiently small column sparsity. Disregarding the dependence on ϵ^{-1}, the algorithm from Theorem III.11 has a convergence parameter that behaves at worst like $\sqrt{m}\|A\|_\infty\|x^0 - x^*\|_2$, whereas standard gradient descent has convergence parameter roughly $m\|A\|_\infty^2\|x^0 - x^*\|_\infty$, because an iteration involves computing a full gradient, which takes $O(m)$ time. It suffices to observe that $\sqrt{m}\|x^0 - x^*\|_2 \leq m\|x^0 - x^*\|_\infty$.

Algorithm 1 Solving constrained ℓ_∞ regression with AP-CD as a subroutine

```
Let $f(x) = \mathrm{smax}_{\frac{\epsilon}{2\log m}}(Ax - b)$.
Let $F(x) = f(x) \forall x \in B_\infty^c$, and let $F(x) = \infty$ otherwise.
Let $N = \lceil \log\frac{D}{\epsilon}\rceil + 1$.
Initialize $x^{0,0} = z^{0,0} = x^0$.
Let $T = \sqrt{8C_1 m^2}$.
Let $S_{1/2} = \sqrt{\frac{16\left\|A\right\|_\infty^2 m\log m}{\epsilon}}$
for $k = 0, 1, 2, \ldots N - 1$ do
    Set $x^{k,0} = x^{k-1,T}, z^{k,0} = z^{0,0}$.
    if $F(x^{0,0}) - F(x^{k,0}) \geq D - \epsilon$ then
        Terminate and return $F(x^{k,0})$.
    else if $C_2 S_{1/2}^2 s \geq C_1 m^2\frac{D}{2^{k+1}}$ then
        Terminate and return the result of running AP-CD for $\sqrt{\frac{2C_2 S_{1/2}^2 s}{\epsilon}}$ iterations on $F$.
    else
        Let $x^{k,T}$ be the result of running AP-CD on $F$ for $T$ iterations.
    end if
end for
Return $x^{N-1,T}$.
```

We also remark that there is an unaccelerated version of our coordinate descent algorithm which can use uniform sampling and also matches the runtime guarantee of ℓ_∞ gradient descent for the regression problem, if efficient smoothness and sampling oracles are not available. We state this result here, and give the proof in the full version of our paper [ST18].

Theorem III.12 (Uniform sampling regression)**.** *There is an algorithm initialized at x^0 which computes an ϵ-approximate minimizer restricted to some B_∞^c to the regression problem $\left\|Ax - b\right\|_\infty$ in $\tilde{O}\left(\frac{m\left\|A\right\|_\infty^2\left\|x^0 - x^*\right\|_\infty^2}{\epsilon^2}\right)$ iterations.*

E. Cheap iterations for ℓ_∞ regression

In this section, we show how to attain cheap iterations for A whose columns have bounded sparsity. In particular, suppose A is d-column-sparse. We show how to, for some local smoothness overestimates $L_j^u(x)$, implement maintenance of the $L_j^u(x)$ and sampling by the quantities $\sqrt{L_j^u(x)}$ for each iteration, in time $\tilde{O}(d^2)$, while not affecting the number of iterations by more than a $\sqrt{d}$ multiplicative factor. This shows that the runtime of the efficient implementation of our algorithm is, up to a $\tilde{O}(d^{2.5})$ multiplicative factor, the same as the iteration count for the naive algorithm without efficient iterations. In particular, for $d = \tilde{O}(1)$, we are able to implement each step in $\tilde{O}(1)$ time, without affecting the number of iterations by more than a $\tilde{O}(1)$ factor. More formally, in this section we show the following.

Lemma III.13 (Efficient implementation of iterates)**.** *Suppose we implement* Accel-Regress *for some d-column-sparse A. Then, for some local coordinate smoothness overestimate parameters $L_j^u(x)$ where x is an iterate, it is possible to (1) maintain the sum $\sum_j \sqrt{L_j^u(x)}$, (2) compute for any j the value $L_j^u(x)$, and (3) sample from the distribution $\{p_j \propto \sqrt{L_j^u(x)}\}$ in time $\tilde{O}(d^2)$, without affecting the number of iterations with respect to the original $L_j(x)$ by more than a $\sqrt{d}$ multiplicative factor.*

Proof:

First, we describe the overestimates $L_j^u(x)$ we use for the algorithm, show that the number of iterations is not affected by more than a $\sqrt{d}$ factor, and describe how to implement (1) maintaining the sum of $\sum_j \sqrt{L_j^u(x)}$ and (2) computing the value $L_j^u(x)$ in $\tilde{O}(d^2)$ time for an iterate x.

Recall from the discussion in Lemma III.9 that, for an iterate x, using any smoothness estimates $L_j^u(x)$ which overestimate $L_j(x) = \frac{8}{t}\|A_{:j}\|_\infty(\sum_i p_i^x|A_{ij}|)$ suffices for our algorithm's convergence, where the convergence rate is with respect to the overestimates $L_j^u(x)$ used. We will use the estimates $L_j^u(x) = \frac{8}{t}\|A_{:j}\|_\infty(\sum_i \sqrt{p_i^x|A_{ij}|})^2$ throughout the rest of the discussion in this section and of maximum flow, because it is clear they suffice as an overestimate to the $L_j(x)$ we defined earlier, but using the $L_j^u(x)$ simplify sampling and efficient estimate maintenance.

Let d be the maximum number of nonzero elements in any column in the matrix A. We claim first that using our new overestimates suffices for all runtime guarantees. Indeed, that the new estimates are still overestimates but their sum is only off by a factor of at most d results from Cauchy-Schwarz implying $\sum_i p_i^x|A_{ij}| \le (\sum_i \sqrt{p_i^x|A_{ij}|})^2 \le d\sum_i p_i^x|A_{ij}|$, so the number of iterations using the $L_j^u(x)$ grows by no more than a factor of $\sqrt{d}$ compared to the original $L_j(x)$ we defined. Furthermore, there are only at most d^2 elements in the sum associated with $L_j^u(x)$, so to compute one on the fly for an iterate, it suffices to compute the d^2 relevant cross-terms $\sqrt{p_i^x p_{i'}^x|A_{ij}||A_{i'j}|}$. The terms $\sqrt{|A_{ij}||A_{i'j}|}$ can all be precomputed in time $\tilde{O}(m)$ across the columns.

In order to efficiently compute the new $L_j^u(y^k)$, we will need to maintain at any current iterate y^k the values for $p_i(Ay^k - b) = \frac{\exp((Ay^k-b)_i)}{\sum_{i'}\exp((Ay^k-b)_{i'})}$. Note that taking a coordinate step from y^k to y^{k+1} can only change at most d coordinate values of $Ay^k - b$ in becoming $Ay^{k+1} - b$, because of the column sparsity of A. This is because in our accelerated algorithm, only one coordinate of the iterate changes values per iteration. So, only d updates need to be made to numerators, and the sum can also be updated in almost-constant time.

Now, we describe how to implement (3) sampling from the distribution proportional to $\sqrt{L_j^u(x)}$. For some iterate x, we will describe how to sample from the distribution proportional to $\sqrt{L_j^u(x)} = \sum_i \sqrt{p_i^x}\sqrt{\|A_{:j}\|_\infty|A_{ij}|}$ in $\tilde{O}(d)$ time. Clearly, it suffices to first sample the rows by a distribution proportional to $\sqrt{p_i^x}$, and then sample the indices of that row proportional to $\sqrt{\|A_{:j}\|_\infty|A_{ij}|}$.

To sample the rows, we maintain a complete binary search tree data structure on $\tilde{O}(n)$ leaf nodes over the indices, maintaining the values of $\sqrt{\exp((Ay^k-b)_i)}$ at the leaf nodes, and also maintaining the sum of the values of the leaf nodes in the subtree rooted at each internal node. As we already argued, updating these values can be done in $\tilde{O}(d)$ time in each coordinate step, because the depth of the tree is $O(\log m)$. Finally, the sampling procedure itself simply consists of flipping coins at each layer of the tree, proportional to the weights on the children of the node. To sample the columns within a row, it suffices to pre-compute the values $\sqrt{\|A_{:j}\|_\infty|A_{ij}|}$, which can be done in $\tilde{O}(m)$ time, because the entire matrix A has $\tilde{O}(m)$ entries.

■

IV. Accelerating Maximum Flow

A. Maximum flow preliminaries

The maximum flow problem is defined as follows: given a graph, and two of its vertices s and t labeled as source and sink, find a flow $f \in \mathbb{R}^m$ which satisfies the capacity constraints such that the discrete divergence at the sink, $(Bf)_t$, is as large as possible, and $(Bf)_s = -(BF)_t$, $(Bf)_v = 0$ for $v \ne s, t$.

Following the framework of [She13], we consider instead the equivalent problem of finding a minimum congestion flow; intuitively, if we route 1 unit of flow from s to t and congest edges as little as possible, we can find the maximum flow by just taking the multiple of the minimum congestion flow which just saturates edges. The congestion incurred by a flow f is $\|C^{-1}f\|_\infty$, and we say f routes demands d if $Bf = d$. The problem of finding a minimum congestion flow for a given demand vector can be formulated as follows:

$$\min_f. \quad \|C^{-1}f\|_\infty \quad \text{s.t.} \quad Bf = d, f \ge 0. \tag{4}$$

Whenever the flow problem is clear from context, we will refer to any optimal flow by f^{OPT}.

B. From maximum flow to constrained ℓ_∞ regression

Our paper uses the fact that the maximum flow problem can be transformed into a constrained regression problem, where the key tool used is the concept of a good *congestion approximator*, and its associated properties.

Definition IV.1 (Congestion Approximator)**.** *An α-congestion approximator for G is a matrix R such that for any demand vector b, $\left\|Rb\right\|_\infty \leq OPT_b \leq \alpha\left\|Rb\right\|_\infty$.*

For undirected graphs, it is known that $\tilde{O}(1)$-congestion approximators can be computed in nearly linear time [Mad10], [She13], [KLOS14], [Pen16]. Furthermore, the congestion approximator has certain nice properties, via the construction in [Pen16].

Theorem IV.2 (Summary of results in [Pen16])**.** *There is an algorithm which given an m-edge n-vertex undirected graph runs in time $\tilde{O}(m)$ and with high probability produces an α-congestion approximator R, for $\alpha = \tilde{O}(1)$. Furthermore, the matrix $A \stackrel{\text{def}}{=} 2\alpha RBC$ has the following properties: (1) each column of A has at most $\tilde{O}(1)$ nonzero entries, (2) $\|A\|_\infty = \tilde{O}(1)$, (3) A has $O(n)$ rows, and (4) A can be computed in time $\tilde{O}(m)$.*

The above theorem is the result of a construction in [Pen16]. Properties 2, 3, and 4 are direct results of the construction given in the paper (where 2 follows from the fact that the congestion approximator comes from routing on a graph which is a tree). Property 1 also results from the way in which the tree is constructed, such that the depth of the congestion-approximating tree is only $\tilde{O}(1)$, so that each edge in the original graph G is only routed onto a polylogarithmic number of edges in the tree.

For the remainder of this section, we will for simplicity refer to the number of rows of A with $O(n)$. Our analysis of reducing the flow problem to the regression problem follows that of [She13]. In particular, the reduction is given as follows:

Lemma IV.3. *Let G be an undirected graph and d be a demand vector. Assume we are given an α-congestion approximator R, and the associated matrix $A = 2\alpha RBC$. Furthermore, let $2\alpha Rd \stackrel{\text{def}}{=} b$. In order to approximately solve the maximum flow problem given by Equation* (4)*, it suffices to approximately solve an associated box-constrained regression problem $\|Ax - b\|_\infty$ over $x \in B_\infty^c$ a nearly-constant number of times, and pay an additional $\tilde{O}(m)$ cost, which under the change of variables $x \stackrel{\text{def}}{=} C^{-1}f$ recovers a corresponding flow. We call the full algorithm* FLOW-TO-REGRESS.

In particular, we are able to use R from the statement of Theorem IV.2.

C. Implementation details for accelerated maximum flow

Here, we provide a full description of how to implement relevant machinery for applying the tools from Section III for accelerating the minimization of a constrained ℓ_∞ function to the regression problem given in Lemma IV.3. it suffices to bound the runtime of approximately solving the initial regression problem.

Definition IV.4 (Flow regression problem)**.** *The maximum flow regression problem asks to approximately minimize the function $f(x) = \text{smax}_t(Ax-b)$ subject to $x \in B_\infty^c$ for some c and for A for $\tilde{O}(1)$-column-sparse A with $\left\|A\right\|_\infty = \tilde{O}(1)$.*

We give a simple proof that shows we can control the runtime in a more fine-grained way via bounding the sum $S_{1/2} = \sum_j \sqrt{L_j}$, where as above, $\sqrt{L_j} = \sum_i \sqrt{p_i}\sqrt{\|A_{:j}\|_\infty |A_{ij}|}$.

Lemma IV.5. $S_{1/2} \leq \tilde{O}(\sqrt{n})$.

Proof: Let the largest ℓ_1 norm of a row of A be C_1, the largest number of nonzero entries in a column of A be C_2, and let the largest absolute value of an entry of the j^{th} column be denoted by A_j^*. Then the following holds:

$$\begin{aligned}
&\left(\sum_j\sum_i \sqrt{p_i|A_{ij}|A_j^*}\right)^2 \\
&= \left(\sum_j \sqrt{A_j^*}\cdot\left[\sqrt{C_2}\cdot\sqrt{\sum_i p_i|A_{ij}|}\right]\right)^2 \\
&\leq C_2\left(\sum_j A_j^*\right)\left(\sum_j\sum_i p_i|A_{ij}|\right) \\
&\leq C_2(nC_1)(\sum_i p_i\|A_i\|_1) \leq C_1^2C_2n
\end{aligned} \tag{5}$$

Here, the first line follows from Cauchy-Schwarz and using the fact that the sum $\sum_i \sqrt{p_i|A_{ij}|}$ is C_2-sparse, the second line follows from Cauchy-Schwarz again, and the third line follows from the bounds we assumed. This shows what we desired as C_1, C_2 are both bounded by $\tilde{O}(1)$. ■

D. Putting it all together: faster approximate undirected maximum flow

We have all the tools we need to prove the following:

Theorem IV.6. *There exists an algorithm which computes an ϵ-approximate maximum flow of an m-edge, n-node graph G in time $\tilde{O}\left(m + \frac{\sqrt{n}\|C^{-1}f^{\text{OPT}}\|_2}{\epsilon}\right)$, where f^{OPT} is any maximum flow.*

Proof: The algorithm is straightforward: we simply run the routine FLOW-TO-REGRESS on the demands. Per the guarantees of Lemma IV.3, the runtime is $\tilde{O}(m)$ plus the cost of approximately solving the initial regression problem to ϵ accuracy.

Now, we bound the runtime of the solving the initial regression problem. We run the algorithm ACCEL-REGRESS on the function $f(x)$ subject to $x \in B_c^\infty$ for some c, initializing at $f^0 = 0$.

Following the analysis of Theorem III.11, the number of iterations is bounded by the maximum of $\tilde{O}(m)$ and $\tilde{O}\left(\frac{S_{1/2}\|x^*\|_2}{\epsilon}\right)$, where x^* is the optimal value of F. Let us consider the second of these two values. By definition, we have $\left\|x^*\right\|_2 \leq \sqrt{s}$. Per Lemma IV.5, we have $S_{1/2} \leq \tilde{O}(\sqrt{n})$. Finally, per Lemma III.13, the time to implement each of these iterations is $\tilde{O}(1)$. Putting all these pieces together, we have the desired result. ■

E. Exact maximum flows in uncapacitated graphs

Here, we describe several corollaries of our approach, for rounding to an exact maximum flow for several types of uncapacitated graphs. In an uncapacitated graph, $s = \|f^*\|_2^2 \leq Fn$ where F is the maximum flow value, because the maximum flow is a 0-1 flow, and thus can be decomposed into F $s-t$ paths with length at most n. We assume here that all the graphs are simple, and thus $m \leq n^2$; it is not difficult to generalize these results to non-simple graphs. As preliminaries, we state the following standard techniques for rounding to exact maximum flows.

Lemma IV.7 (Theorem 5 in [LRS13])**.** *There is a randomized algorithm that runs in expected time $\tilde{O}(m)$ which takes a fractional flow of value F on an uncapacitated graph, and returns an integral flow of value $\lfloor F \rfloor$.*

We will thus always assume that we have applied the rounding to an integral flow as a pre-processing step, as it will not affect our asymptotic runtime.

Lemma IV.8 (Depth-First Search for Augmenting Paths)**.** *There is an algorithm that runs in time $O(m)$ which takes a non-maximal integral flow of value F on an uncapacitated graph, and returns an integral flow of value $F+1$.*

Suppose we have a flow with value $(1-\epsilon)F$, where the maximum flow value is F. The two lemmas for rounding and augmenting a flow therefore imply that the additional runtime required to attain an exact maximum flow is $O(\epsilon F m)$.

We state several corollaries of Theorem IV.6 which apply to finding exact maximum flows in various types of undirected uncapacitated graphs. All of these results only hold with high probability.

Corollary IV.9 (Undirected graphs)**.** *There is an algorithm which finds a maximum flow in an undirected, uncapacitated graph in time $\tilde{O}(m^{5/4}n^{1/4})$.*

Corollary IV.10 (Undirected small maximum flow)**.** *There is an algorithm which finds a maximum flow in an undirected, uncapacitated graph with maximum flow value F in time $\tilde{O}(m + \min(\sqrt{mn}F^{3/4}, m^{3/4}n^{1/4}\sqrt{F}))$.*

Corollary IV.11 (Undirected sparse optimal flow)**.** *There is an algorithm which finds a maximum flow in an undirected, uncapacitated graph with a maximum flow that uses at most s edges in time $\tilde{O}(m + m^{1/2}n^{1/4}s^{3/4})$.*

We follow the standard reduction of finding a maximum flow in a directed graph to finding a maximum flow in an undirected graph described in, for example, [Lin09]. In short, an undirected graph with maximum flow value $O(m)$ is created, such that we can initialize the algorithm in Theorem IV.6 at a flow which is off from the true maximum flow by s in ℓ_2^2 distance. We refer the reader to [Lin09] for a more detailed exposition of this reduction.

Thus, after applying this reduction, the only difference compared to the runtimes for exact flows in undirected graphs is that the rounding algorithm will always take time $O(\epsilon m^2)$ instead of $O(\epsilon F m)$. This immediately yields the following runtimes for exact maximum flows in directed graphs.

Corollary IV.12 (Directed graphs)**.** *There is an algorithm which finds a maximum flow in a directed, uncapacitated graph in time $\tilde{O}(m^{5/4}n^{1/4})$.*

Corollary IV.13 (Directed sparse optimal flow)**.** *There is an algorithm which finds a maximum flow in a directed, uncapacitated graph in time $\tilde{O}(mn^{1/4}s^{1/4})$.*

ACKNOWLEDGMENT

This work was supported by NSF Graduate Fellowship DGE-1656518.

REFERENCES

[AO15] Zeyuan Allen Zhu and Lorenzo Orecchia. Nearly-linear time positive LP solver with faster convergence rate. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 229–236, 2015.

[AQRY16] Zeyuan Allen Zhu, Zheng Qu, Peter Richtárik, and Yang Yuan. Even faster accelerated coordinate descent using non-uniform sampling. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 1110–1119, 2016.

[Bub15] Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, 8(3-4):231–357, 2015.

[CKM+11] Paul Christiano, Jonathan A. Kelner, Aleksander Madry, Daniel A. Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 273–282, 2011.

[FR15] Olivier Fercoq and Peter Richtárik. Accelerated, parallel, and proximal coordinate descent. *SIAM Journal on Optimization*, 25(4):1997–2023, 2015.

[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 *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 25-27 January 1998, San Francisco, California.*, pages 490–499, 1998.

[KL02] David R. Karger and Matthew S. Levine. Random sampling in residual graphs. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 63–66, 2002.

[KLOS14] Jonathan A. Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 217–226, 2014.

[Lin09] Henry Lin. Reducing directed max flow to undirected max flow. Unpublished Manuscript, 2009.

[LRS13] Yin Tat Lee, Satish Rao, and Nikhil Srivastava. A new approach to computing maximum flows using electrical flows. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 755–764, 2013.

[LS13] Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 147–156, 2013.

[LS14] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in õ(vrank) iterations and faster algorithms for maximum flow. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 424–433, 2014.

[LS15a] Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 230–249, 2015.

[LS15b] Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 230–249, 2015.

[Mad10] Aleksander Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 245–254, 2010.

[Mad13] Aleksander Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 253–262, 2013.

[Nem04] Arkadi Nemirovski. Prox-method with rate of convergence o(1/t) for variational inequalities with lipschitz continuous monotone operators and smooth convex-concave saddle point problems. *SIAM Journal on Optimization*, 15(1):229–251, 2004.

[Nes03] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course, volume I.* 2003.

[Nes05] Yurii Nesterov. Smooth minimization of non-smooth functions. *Math. Program.*, 103(1):127–152, 2005.

[Nes07] Yurii Nesterov. Dual extrapolation and its applications to solving variational inequalities and related problems. *Math. Program.*, 109(2-3):319–344, 2007.

[Nes12] Yurii Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[NS17] Yurii Nesterov and Sebastian U. Stich. Efficiency of the accelerated coordinate descent method on structured optimization problems. *SIAM Journal on Optimization*, 27(1):110–123, 2017.

[Pen16] Richard Peng. Approximate undirected maximum flows in $O(m\text{polylog}(n))$ time. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1862–1867, 2016.

[QR16] Zheng Qu and Peter Richtárik. Coordinate descent with arbitrary sampling I: algorithms and complexity. *Optimization Methods and Software*, 31(5):829–857, 2016.

[She13] Jonah Sherman. Nearly maximum flows in nearly linear time. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 263–269, 2013.

[She17] Jonah Sherman. Area-convexity, l_∞ regularization, and undirected multicommodity flow. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 452–460, 2017.

[ST04] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 81–90, 2004.

[ST18] Aaron Sidford and Kevin Tian. Coordinate Methods for Accelerating ℓ_∞ Regression and Faster Approximate Maximum Flow. *arXiv preprint arXiv:1808.01278*, 2018.

[SWWY18] Aaron Sidford, Mengdi Wang, Xian Wu, and Yinyu Ye. Variance reduced value iteration and faster algorithms for solving markov decision processes. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 770–787, 2018.