Coordinate Methods for Accelerating $\ell_\infty$ Regression
and Faster Approximate Maximum Flow

Aaron Sidford
Stanford University
sidford@stanford.edu

Kevin Tian
Stanford University
kjtian@stanford.edu

Abstract

In this paper we provide faster algorithms for approximately solving $\ell_\infty$ regression, a fundamental problem prevalent in both combinatorial and continuous optimization (for example, it was shown in [LS15a] that $\ell_\infty$ regression is equivalent to solving linear programming). 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 $\tilde{O}\left(m + \sqrt{ns}\epsilon\right)$ to compute an $\epsilon$-approximate maximum flow, for an undirected graph with $m$ edges, $n$ vertices, and where $s$ is the squared $\ell_2$ 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 $O\left(\frac{m}{\epsilon}\right)$ 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 $\ell_\infty$ 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.
1 Introduction

The classical problem of $\ell_\infty$ regression corresponds to finding a point $x^*$ such that

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

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

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

This regression problem has fundamental implications in many areas of statistics and optimization [She13, LS14, LS15a, SWY18]. In many of these settings, it is also useful to design iterative method machinery for the following more general problem of box-constrained $\ell_\infty$ regression:

$$x^* = \arg\min_{x \in S} \|Ax - b\|_\infty, A \in \mathbb{R}^{n \times m}, b \in \mathbb{R}^n, \text{ and } S = \{x \in \mathbb{R}^m : x_i \in [l_j, r_j] \forall j \in [m]\}$$

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 $\ell_\infty$ regression problem. For this work, the domain constraint will only be of the form $x \in B^c_\infty$ where we use $B^c_\infty$ 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 $\ell_\infty$ regression problem.

Many natural optimization problems which arise from combinatorial settings can be written in the form of constrained $\ell_\infty$ 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 $\epsilon$-approximate minimizers to constrained $\ell_\infty$-regression, that when specialized to the maximum flow problem, achieve faster running times.

1.1 Regression Results

In this paper we show how to apply ideas from the literature on accelerated coordinate descent methods (see Section 1.3) to provide faster algorithms for approximately solving constrained $\ell_\infty$ 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 1.1 (Accelerated constrained $\ell_\infty$ regression)** There is an algorithm initialized at $x^0$ that $\epsilon$-approximately minimizes the box-constrained $\ell_\infty$ regression problem (for any $c$)

$$\min_{x \in B^c_\infty} \|Ax - b\|_\infty$$

$\tilde{O} \left( \sqrt{\text{md}} \|A\|_\infty \|x^0 - x^*\|_2 \right)$ 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 $\ell_\infty$ 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 $\ell_2$-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 $\ell_\infty$ gradient descent for certain highly structured optimization problems, i.e. $\ell_\infty$ regression. Whereas Sherman’s work introduced an intriguing notion of area convexity and new regularizations of $\ell_\infty$ regression, our results are achieved by working with the classic smoothing of the $\ell_\infty$ 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 [AQR16, 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 $\ell_\infty$ 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 $\ell_\infty$-smooth functions.

1.2 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 specify $s$ and $t$ this problem takes as input a vector $d \in \mathbb{R}^V$ such that $d^T 1 = 0$, where $1$ 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 $\ell_\infty$ 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

$$f^* = \arg\min_{Bf = d} \|f\|_\infty$$

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
\[
f^* = \arg\min_{\|f\|_\infty \leq F} \|Bf - d\|_\infty
\]
where we note that the constraint $\|f\|_\infty \leq F$ can be decomposed as the indicator of a box so that this objective matches the form of Equation 1. The exact reduction we use has a few modifications: the box constraint is more simply replaced by $\|f\|_\infty \leq 1$, 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 4.2).

In this paper we show how to use our improved algorithms for structured $\ell_\infty$ regression in order to obtain faster algorithms for maximum flow. We do so by leveraging the tighter dependence on the domain size (in the $\ell_2$ norm rather than $\ell_\infty$) 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 1.2 ($\ell_2$ accelerated approximate maximum flow)** There is an algorithm that takes time $\tilde{O}(m + \frac{\sqrt{ms}}{\epsilon})$ to find an $\epsilon$-approximate maximum flow, where $s$ is ratio of the the $\ell_2$ 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 maximum flow, i.e. a maximum flow in which the average path length in the flow decomposition of the optimal flow is small. Leveraging this we provide several new results on exact undirected and directed maximum flow on uncapacitated graphs as well.

**Theorem 1.3 (Improved algorithms for 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 $\tilde{O}(m + \min(\sqrt{mnF^{3/4}}, m^{3/4}n^{1/4}\sqrt{F}))$.
- 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 $\tilde{O}(m + m^{1/2}s^{1/4}n^{3/4})$.
- 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 uncapacitated 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 1.4 (Exact maximum flow for directed uncapacitated graphs)** There is an algorithm which finds a maximum flow in a directed, uncapacitated graph in time $\tilde{O}(m^{5/4}n^{1/4})$. 

3
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 \( \tilde{O}(m^{5/4}n^{1/4}) \), which dominate our \( \tilde{O}(m^{5/4}n^{1/4}) \) running time, they do 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.

1.3 Previous work

Here we embark on a deeper dive into the context of the problems and tools discussed in this paper. **Solving the \( \ell_\infty \) regression problem.** For a non-differentiable function like \( f(x) = \|x\|_\infty \), it is possible to use the toolkit for linear programming (including interior point and cutting plane methods [LS14, LS15b]) to obtain iterative algorithms for approximate minimization, but these particular algorithms have a larger dependence on dimension, and it is widely believed that they are inherently super-linear. Traditionally, iterative algorithms with a better dependence on dimension for approximately solving the regression problem proceed in two stages. First, the algorithm will construct a smooth approximation to the original function, which is typically explicitly derived via regularizing the dual function using a regularizer which is both smooth and bounded in range [Nes05]. The smooth approximation is constructed such that approximately minimizing the approximate function is sufficient to approximately minimize the original function. Secondly, an iterative first-order method such as gradient descent in a particular norm, or one of its many variants, is applied to approximately minimize the smoothed function.

One of the earlier works to develop algorithms using first-order methods under this framework to solve the regression problem is [Nes05]. One regularizer used in this work for optimization over a dual variable in the simplex was the entropy regularizer, which yields the smooth approximation to the \( \ell_\infty \) norm defined by \( \text{smax}_t(x) = t \log(\sum_j \exp(x_j t)) \). Essentially, the methods presented in this work converge to an \( \epsilon \)-approximate solution in a number of iterations proportional to either \( O(\epsilon^{-2}) \) or \( O(\sqrt{m\epsilon^{-1}}) \), hiding problem-specific dependencies on smoothness and domain size. However, the cost of each iteration involves computing a whole gradient, which incurs another multiplicative loss of the dimension in the runtime.

Several other works which aimed to solve the regression problem via constructing a smooth approximation, including [Nem04] and [Nes07], incurred the same fundamental barrier in convergence rate. These works aimed to pose the (smooth) regularized regression problem as finding the saddle point of a convex-concave function via a specially-constructed first-order method. The main barrier to improving prior work up to this point has been the inability to construct regularizers of small range which are strongly convex with respect to the \( \ell_\infty \) norm. For some time, these issues posed a barrier towards finding faster algorithms for the regression problem, and many related problems.

Very recently, Sherman [She17] presented an alternative method which was able to break this barrier and obtain an \( O(\frac{1}{\epsilon}) \) iteration count for finding approximate solutions to the regression problem, where each iteration can be applied in time to compute a gradient. The algorithm used was a variation of Nesterov’s dual extrapolation method [Nes07] for approximately finding a saddle point in a convex-concave function, adapted to work for regularizers satisfying a weaker property known as area convexity, and an analysis of its convergence. As a corollary, this algorithm was able to obtain
Table 1: Dependencies of algorithms for $\ell_{\infty}$ regression in $A \in \mathbb{R}^{n \times m}$ on various problem parameters. Note that there is up to an $O(m)$ discrepancy between the $\ell_2$ and $\ell_{\infty}$ norms. Here, $d$ is the maximum number of nonzero entries in any column of $A$.

**Abbreviated history of first-order methods, emphasizing coordinate-based methods.**

First-order methods for convex optimization have a long history. Gradient descent methods converging at a $O(\frac{1}{\epsilon^2})$ rate for Lipschitz functions and at a $O(\frac{1}{\epsilon})$ rate for smooth functions has been well studied (for example, see [Nes03] or [Bub15] for a more detailed exposition), and has been applied in many important settings.

Nesterov was the first to note that there existed a first order algorithm for minimizing functions smooth in the Euclidean norm which converged at the rate $O(\frac{1}{T^2})$. The method is optimal in the sense that matching $O(\frac{1}{T^2})$ lower bounds for generic smooth functions had long been known. Unfortunately, this method does not apply generically to functions which are smooth in other norms, in the same way that unaccelerated variants do, without possibly paying an additional dependence on the dimension. In particular, the accelerated convergence rate depends on the regularizer that the mirror descent steps use, and thus the analysis incurs a loss based on the size of the regularizer, which is the barrier in the aforementioned $\ell_{\infty}$-smooth function case. More formally, it is a folklore result that any function which is strongly-convex over $[-1,1]^n$ in the $\ell_{\infty}$ norm has range at least $\frac{n^2}{2}$, which we show in Appendix A.1.

There has been much interest in applying randomized first order methods to more efficiently obtain an approximate minimizer on expectation, when the convex optimization problem has certain structure. Two examples of these randomized methods in the literature are stochastic gradient and coordinate gradient descent methods. Stochastic gradient descent is useful when the function to be minimized efficiently admits a stochastic gradient oracle (for example in finite-sum settings), which is defined as an oracle which at any iterate $x_k$ gives a noisy estimate $g(x_k)$ such that $\mathbb{E}[g(x_k)] = \nabla f(x_k)$.

Coordinate methods, on the other hand, are an alternative randomized algorithm studied first in [Nes12]. With a similar motivation as stochastic gradient methods, the idea is that using crude, computationally efficient, approximations to the full gradient, one is still able to find an approximate minimizer on expectation. One benefit is that coordinate descent admits a more fine-grained analysis of convergence rate, based on structural properties of the function, i.e. the smoothness of the function in each coordinate. There is a line of work which uses that there is a dual function of the finite-sum problem (which admits a natural stochastic oracle) that separates by coordinate, implying an interesting duality between coordinate descent and stochastic gradient descent algorithms.
Generalizations of standard coordinate descent have received much attention recently, both for their powerful theoretical and practical implications. [Nes12] provided an accelerated version of the standard coordinate descent algorithm, but the naive implementation of its steps were inefficient, taking linear time in the dimension. The study of efficient accelerated coordinate descent methods (which converge at the rate $O\left(\frac{1}{T^2}\right)$ iterations without an additional dependence on dimension) was pioneered by [LS13], and since then a flurry of other works, including [FR15, AQRY16, QR16] have improved the rate of convergence and generalized the methods to composite functions with a separable composite term, of the form $F(x) = f(x) + \sum_j \psi_j(x_j)$. We remark that our box constraint can be represented as such a separable composite term in the objective, and our constrained accelerated coordinate descent algorithm is an adaptation of such composite methods. For a more detailed history of the study of coordinate descent methods, we refer the reader to [FR15].

Accelerated coordinate based methods have proven to be useful in many ways when applied to problems in theoretical computer science. For example, the authors of [LS13] framed graph Laplacian system solvers as a coordinate descent problem to give better runtime guarantees. One particularly interesting example that highlighted the potential for using accelerated coordinate descent in minimizing entropy-based functions was the work of [AO15] in solving packing and covering LPs, where the constraint matrix is nonnegative, in which they also attained a $O\left(\frac{1}{\epsilon}\right)$ rate of convergence.

Conventional wisdom is that these results are specific to the structure of the particular problem, so any exploration of accelerated methods in greater generality is particularly interesting.

The maximum flow problem. The maximum flow problem is a fundamental problem in combinatorial optimization that has been studied extensively for several decades. Up until recently, the toolkit used to solve the problem has been primarily combinatorial, culminating in algorithms with runtime roughly $\tilde{O}\left(\min\{mn^{2/3}, m^{3/2}\}\right)$ for finding a maximum flow in graphs with $m$ edges and $n$ vertices and polynomially bounded capacities [GR98], and $\tilde{O}(m + nF)$ for finding a maximum flow in undirected graphs with $m$ edges, $n$ vertices, and a maximum flow value of $F$ [KL02]. Breakthroughs in the related problem of electrical flow using tools from continuous optimization and numerical linear algebra were first achieved by Spielman and Teng [ST04] who showed that solving a linear system in the Laplacian of a graph could be done in nearly linear time, which is equivalent to computing an electrical flow.

Notably, the electric flow problem corresponds to approximately solving an $\ell_2$ regression problem $\|Ax - b\|_2$, and the maximum flow problem corresponds to approximately solving an $\ell_\infty$ regression problem $\|Ax - b\|_\infty$. Accordingly, using the faster algorithms for electric flow combined with a multiplicative weights approach, the authors of [CKM+11] were able to make a breakthrough to approximately solve maximum flow with a runtime of $\tilde{O}(mn^{1/3})$, where $\tilde{O}$ hides logarithms in the runtime. Finally, using constructions presented in [Mad10], the authors of [She13] and [KLOS14] were able to reduce this runtime to nearly linear, essentially using variants of preconditioned gradient descent in the $\ell_\infty$ norm. This runtime was reduced to $\tilde{O}\left(\frac{m}{\epsilon^2}\right)$ by Peng in [Pen16] by using a recursive construction of the combinatorial preconditioner. As previously mentioned, the $\frac{1}{T^2}$ dependence in the runtime was a barrier which was typical of algorithms for minimizing $\ell_\infty$-smooth functions, and was broken in [She17], who attained a runtime of $\tilde{O}\left(\frac{mk}{\epsilon^4}\right)$ for the $k$-multicommodity flow problem.

1.4 Organization

The rest of this paper is organized as follows. Many proofs are deferred to the appendices.

- **Section 2: Overview.** We introduce the definitions and notation we use throughout the paper, and give a general framework motivating our work.
Table 2: Complexity of maximum flow algorithms since [GR98] for undirected graphs with \( n \) vertices, \( m \) edges, where \( s \) is the \( \ell_2^2 \) of the maximum flow’s congestion, and \( F \) is the maximum flow value.

- **Section 3: Regression.** We first give a framework for accelerated randomized algorithms which minimize the constrained \( \ell_\infty \) regression function based on uniform sampling, as well as a faster one 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 4: 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 (indeed, our implementation works in more generality to regression problems in a column-sparse matrix). Furthermore, we give the exact maximum flow runtimes achieved via rounding the resulting approximate flow of our method.

2 Overview

2.1 Basic Definitions

First, we define some basic objects and properties which we will deal with throughout this paper.

**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 \( \{1, 2, \ldots, m\} \). 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 which is the coordinate-wise product, i.e. its \( j^{th} \) 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 \( \text{diag}(w) \) 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.
We associate the following matrices with the graph $G$ well-defined.

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 $\| x \|_p \overset{\text{def}}{=} (\sum_j x_j^p)^{1/p}$ to denote the $\ell_p$ norm. For vector valued $w \in \mathbb{R}^m_{\geq 0}$, we use $\| x \|_w^2 \overset{\text{def}}{=} \sum_j w_j x_j^2$ to denote the weighted quadratic norm.

For a norm $\| \cdot \|$, we write the dual norm as $\| \cdot \|_*$, defined as $\| x \|_* \overset{\text{def}}{=} \max_{\| y \| \leq 1} y^\top x$. It is well known that the dual norm of $\ell_p$ is $\ell_q$ for $1 = \frac{1}{p} + \frac{1}{q}$. For a matrix $A$ and a vector norm $\| \cdot \|$, we correspondingly define the matrix norm $\| A \|_m \overset{\text{def}}{=} \max_{\| x \| = 1} \| A x \|$. For example, $\| A \|_\infty$ is the largest $\ell_1$ norm of a row of $A$.

**Functions.** We will primarily be concerned with minimizing convex functions $f(x)$ subject to the argument being restricted by a box constraint, where the domain will be $x \in B^m_\infty$ where $B^m_\infty$ is some scaled $\ell_\infty$ ball unless otherwise specified. Whenever the function is clear from context, $x^*$ will refer to any minimizing argument of the function. We use the term $\epsilon$-approximate minimizer of a function $f$ to mean any point $x$ such that $f(x^*) \leq f(x) \leq f(x^*) + \epsilon$. Furthermore, we define the OPT operator to be such that $\text{OPT}(f)$ is the optimal value of $f$, when this optimal value is well-defined.

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

**2.2 Overview of our algorithms**

Here, we give an overview of the main ideas used in our algorithms for approximately solving $\ell_\infty$ 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 $\ell_\infty$ regression in particular follows from a bound on the local coordinate smoothnesses for an $\ell_\infty$-smooth function, which is described in full detail in Section 3.4. 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 3.13.

**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 2.1 (Local coordinate smoothness)** We say a function $f$ is $L_j(x)$ locally coordinate smooth in coordinate $j$ at a point $x$, if for $|c| \leq \frac{\|\nabla f(x)\|}{L_j(x)}$, $|\nabla_j f(x + c e_j) - \nabla_j f(x)| \leq L_j(x)|c|$ $\forall x \in \text{dom}(f), c \in \mathbb{R}$. Equivalently, for differentiable convex $f$, for $y$ between $x \pm \frac{1}{L_j(x)} \nabla_j f(x)$, $f(y) \leq f(x) + \nabla_f_j(x)(y_j - x_j) + \frac{L_j(x)}{2}|y_j - x_j|^2$.

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. For a more motivating discussion of this definition, we refer the reader to an analysis of coordinate descent presented in Appendix A.4. 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 $\ell_\infty$-smooth functions.** Here, we sketch the main idea underlying our accelerated methods. Why is it possible to hope to accelerate gradient methods in the $\ell_\infty$ 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 $f$.

As we show in Appendix A, gradient descent for a $\ell_\infty$-smooth function initialized at $x^0 \in \mathbb{R}^m$ takes roughly $\frac{L_j(x)}{\epsilon} \sum_j L_j$ iterations to converge to a solution which has $\epsilon$ additive error, whereas coordinate descent with appropriate sampling probabilities $\frac{S}{\epsilon}$, for $S = \sum_j L_j$, takes $\frac{S\|x^0 - x^*\|_\infty^2}{\epsilon}$ iterations to converge to the same quality of solution.

Note that when the norm in the gradient descent method is $\|\cdot\|_\infty$, we have $\|x^0 - x^*\|_\infty^2 \leq m\|x^0 - x^*\|_\infty^2$, 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 $S \leq L$, 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 2.2** Suppose for some point $x$, $f : \mathbb{R}^m \to \mathbb{R}$ is convex and $L$-smooth with respect to $\|\cdot\|_\infty$, $\Lambda_j(x) = \nabla_j^2 f(x)$, and $S = \sum_j \Lambda_j(x)$. Then $S \leq L$.

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

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

Thus, by the probabilistic method, there exists some $y$ such that $S \leq y^\top My \leq L$, 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 for accelerated algorithms.** 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 $\ell_\infty$ 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 $\ell_\infty$ regression admits local smoothnesses which can be bounded in a structured way, in Section 3.2. 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 3.13.

Furthermore, in Section 3.3 and Section 3.4, 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 $\ell_\infty$ regression.

In Section 4, we study the maximum flow problem as an example of a problem which can be reduced to $\ell_\infty$ 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 $\ell_\infty$ regression algorithm yields the fastest currently known approximate maximum flow algorithm.

### 3 Minimizing $\|Ax - b\|_\infty$ subject to a box constraint

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 $\ell_\infty$ 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 $\epsilon$-approximate minimizer of the constrained $\ell_\infty$ regression problem at an $O(\frac{1}{\epsilon})$ rate.

**Definition 3.1 (Box-constrained $\ell_\infty$ regression problem)** This is the problem of finding a minimizer to the function $\|Ax - b\|_\infty$, where the argument $x$ has domain $B_c^\infty$ for some $c$.

In the style of previous approaches to solving $\ell_\infty$ 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 $\text{OPT}$ is proportional to

$$O\left(\frac{1}{T^2}\right)$$

so if we wish for an $\epsilon$-approximate minimizer, it suffices to pick $T = O(\frac{1}{\epsilon})$. 


3.1 Constructing the smooth approximation to regression

In this section, we define the smooth approximation for $\ell_\infty$ regression we use through the paper and provide some technical facts about this approximation. Note that these approximations are standard in the literature. First, we define the smax function which is used throughout.

**Definition 3.2 (Soft-max)** For all real valued vectors $x$ we let $\text{smax}_t(x) \overset{\text{def}}{=} t \log(\sum_j \exp(x_j/t))$.

**Fact 3.3 (smax Additive Error)** $\forall x \in \mathbb{R}^m$, $\max_{j \in [m]} x_j \leq \text{smax}_t(x) \leq t \log m + \max_{j \in [m]} x_j$

**Proof:** This follows from monotonicity of log and positivity of exp: letting $p$ be the maximal index of $x$, $\text{smax}_t(x) \geq t \log(\exp(x_p/t)) = x_p^*$, and $\text{smax}_t(x) \leq t \log(m \exp(x_p/t)) = t \log m + x_p^*$.

**Definition 3.4** Inside the scope of the remainder of this section, let $f(x) \overset{\text{def}}{=} \text{smax}_t(Ax - b)$.

Note that these properties say something about the quality of approximation smax provides on the maximum element of a vector, instead of its $\ell_\infty$ 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 $\frac{\epsilon}{t}$-approximate minimizer of $f$ is sufficient to finding an $\epsilon$-approximate minimizer of the original regression problem. Thus, for notational convenience, we will fix $t = O(\frac{1}{\log m})$ for the remainder of this work, and concentrate on finding an $\epsilon$-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 3.5** For $x \in \mathbb{R}^m$ let $p(x) \in \mathbb{R}^m$ be defined as $p_j(x) \overset{\text{def}}{=} \frac{\exp(x_j/t)}{\sum_i \exp(x_i/t)}$.

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 3.6 (Soft-max Calculus)** The gradient and Hessian of smax are as follows: $\nabla \text{smax}_t(x) = p(x)$ and $\nabla^2 \text{smax}_t(x) = \frac{1}{t} \text{diag}(p(x)) - vv^T \leq \frac{1}{t} \text{diag}(p(x))$, 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 3.7** For all $x, y \in \mathbb{R}^n$ with $\|y - x\|_\infty \leq t$, the following upper bound holds:

$$\text{smax}_t(y) \leq \text{smax}_t(x) + \nabla \text{smax}_t(x)^\top (y - x) + \frac{4}{t} \|y - x\|_2^2.$$
Proof: For $\alpha \in [0,1]$, let $x^\alpha = x + \alpha(y - x)$. Then, we have

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

(3.1)

However, based on Fact 3.6, we know that $\nabla^2 \text{smax}_t(x^\alpha) \preceq \frac{1}{t} \text{diag}(p(x^\alpha))$. Also, note that $p_j(x^\alpha) \leq 8p_j(x)$ for all $j, \alpha$ for the following reason: for $\|x^\alpha - x\|_\infty \leq t$ the numerator $\exp\left(\frac{t}{\alpha}\right)$ underestimates $\exp\left(\frac{\alpha}{t}\right)$ by at most a factor of 4, and the denominator similarly is an overestimate by at most a factor of 4, so the discrepancy is at most a factor of $4^2 < 8$. So, $\nabla^2 \text{smax}_t(x^\alpha) \preceq \frac{8}{t} \text{diag}(p(x))$. Plugging this into Equation (3.1), we have the desired conclusion.

3.2 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 in the style of Lemma A.4 and Lemma A.5. These corresponding results for bounding the convergence of gradient descent and coordinate descent operate by giving an upper bound on the function value, and then showing that the descent step minimizes the upper bound. Therefore, 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 3.8 Let $L_j(x) \geq \max\{\frac{8}{t} p^x(A_j)^2, \frac{1}{t} ||A_j||_\infty |p^x A_j|\}$, 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, $f(x - \frac{1}{L_j} \nabla f(x)) \leq f(x) - \frac{\nabla^2 f(x)}{2L_j}$.

Proof: Firstly, we know that $\nabla f(x) = A^\top p^x$ and $\nabla^2 f(x) \preceq A^\top \text{diag}(p^x) A$. Furthermore, we have the following upper bound based on the analysis of Lemma 3.8:

$$f(x + d) \leq f(x) + \nabla f(x)^\top d + \frac{4}{t} d^\top A^\top \text{diag}(p^x) A d, \forall \|Ad\|_\infty \leq t$$

Indeed, specialized to the case where $d = \eta e_j$, we have

$$f(x + \eta e_j) \leq f(x) + \eta \nabla_j f(x)e_j + \frac{4}{t} \eta^2 (p^x(A_j)^2), \forall \eta \|A_j\|_\infty \leq t$$

(3.2)

Let $C = \frac{8}{t} e_j^\top A^\top \text{diag}(p^x) A e_j$ and let $D = \frac{1}{t} |\nabla_j f(x)||A_j|_\infty$. We wish to prove that for $L_j(x) \geq \max(C, D)$, the guarantee holds.

Note that setting $\eta = \frac{\nabla^2 f(x)}{\nabla f(x)|A_j|_\infty} = \frac{|\nabla f(x)|}{D}$ is the largest step size we can take to stay within the box where the quadratic upper bound given by Equation 3.3 holds. So if we take a step size of
smaller absolute value than $\left| -\nabla f(x)_{L_j(x)} \right|$, 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

$$f(x + \eta e_j) \leq f(x) + \eta \nabla f(x)e_j + \frac{L_j(x)}{2}\eta^2, \ \forall \eta \|A_j\|_\infty \leq t$$

(3.3)
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 3.9. As we show in the application to column-sparse matrices and maximum flow, it is possible that using these overestimates yields 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 3.9** Let $L_j(x) = \frac{8}{T}p^T|A_{ij}| \cdot \|A_{ij}\|_\infty$. Then, $L_j(x) \geq \max\{\frac{8}{T}p^T(A_{ij})^2, \frac{1}{T}\|A_{ij}\|_\infty|p^T A_{ij}|\}$, and $\sum_j L_j(x) \leq \frac{8\|A\|_\infty^2}{t}.$

**Proof:** First, we show $L_j(x) \geq \max\{\frac{8}{T}p^T(A_{ij})^2, \frac{1}{T}\|A_{ij}\|_\infty|p^T A_{ij}|\}$. $L_j(x) \geq \frac{1}{T}\|A_{ij}\|_\infty|p^T A_{ij}|$ is obvious and follows from the triangle inequality. $L_j(x) = \frac{8}{T}\sum_i p_i^T(A_{ij}^2)$ follows from

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

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

$$\max_{p \in \Delta_n} \sum_j p^T|A_{ij}| \cdot \|A_{ij}\|_\infty \leq \max_{p \in \Delta_n} \|A\|_\infty \sum_i p_i \|A_i\|_1 = \|A\|_\infty^2$$

(3.4)

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

**3.3 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 the original algorithm requires reinitializing both the primal and the dual point multiple times in order to converge, whereas our algorithm restarts with a new primal point, but the same dual point. In particular, the 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 essentially the same as in its original form, where we note that due to the choice
of sampling probabilities the only norm used in the analysis is a multiple of the \( \ell_2 \) norm, and carefully change the argument to extend to a convergence for differing primal and dual initial points. Nonetheless, we include the proof for completeness in the appendices: on a first read, one may use the statement of convergence as a black box result for further analysis. We follow the presentation of [QR16].

**Theorem 3.10 (Acceleration for local coordinate smoothnesses and dual initializations)**

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 \( \infty \) 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 \( B_{\infty}^c \), which in \( T \) iterations returns a point \( x^T \) such that

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

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 \( \xi, x \).

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 3.13, we show that all these steps are implementable in nearly-constant time for \( \ell_\infty \) regression in a column-sparse matrix.

### 3.4 Accelerated constrained \( \ell_\infty \) 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. Suppose for now we have as an input \( D \defeq f(x^0) - f(x^*) \), the function distance to \( \text{OPT} \) from some initial point \( x^0 \), and \( s \defeq \|x^0 - x^*\|_2^2 \), the squared \( \ell_2 \) distance from the minimizer. We will discuss the details of this assumption at the end of this section, and why they are not restrictive. Then, we give our algorithm in full. Recall that per the guarantees in Theorem 3.10, 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

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

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 \( O(\log \frac{D}{\varepsilon}) \) phases of AP-CD for \( T \) iterations. At the end of the \( k \)-th phase, we will either \( \varepsilon \)-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.
\[ x = \text{Accel-Regress}(A, b, c, x^0, D, s, \epsilon) \]
1. Let \( f(x) = \max_{s} A^T_s (Ax - b) \).
2. Let \( F(x) = f(x) \forall x \in B^c_{2\epsilon} \), and let \( F(x) = \infty \) otherwise.
3. Let \( N = \lceil \log \frac{D}{\epsilon} \rceil + 1 \).
4. Initialize \( x^{0,0} = z^{0,0} = x^0 \).
5. Let \( T = \sqrt{8C_1m^2} \).
6. Let \( S_{1/2} = \sqrt{\frac{16}{\epsilon} \| A \|_{\infty}^2 \frac{m \log m}{\epsilon}} \).
7. Iterate for \( k = 0, 1, 2 \ldots N - 1 \):
   (a) Set \( x^{k,0} = x^{k-1,T}, z^{k,0} = z^{0,0} \).
   (b) If \( F(x^{0,0}) - F(x^{k,0}) \geq D - \epsilon \), terminate and return \( F(x^{k,0}) \).
   (c) If \( C_2 S_{1/2}^2 s \geq C_1 m^2 \frac{D}{2^{1/2}} \), then terminate and return the result of running AP-CD for \( \sqrt{\frac{2C_2 S_{1/2}^2}{s}} \) iterations on \( F \).
   (d) Else, let \( x^{k,T} \) be the result of running AP-CD on \( F \) for \( T \) iterations.
8. Return \( x^{N-1,T} \).

**Figure 1:** Solving constrained \( \ell_{\infty} \) regression with AP-CD as a subroutine

**Theorem 3.11** \( \text{Accel-Regress} \) initialized at \( x^0 \) computes an \( \epsilon \)-approximate minimizer to the regression problem \( F(x) = \| Ax - b \|_{\infty} \) constrained to \( B^c_{2\epsilon} \) for some \( c \) in

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

iterations, using AP-CD as a subroutine.

**Proof:** Throughout, we call \( x^* \) the minimizer of \( F \). First, we show that our choice of \( S_{1/2} \) is actually a valid upper bound on \( \sum_j \sqrt{L_j(x)} \) for any point \( x \) and coordinate smoothnesses \( L_j(x) \). To see this, it suffices to use Lemma 3.9, note that \( t = \tilde{O}(\epsilon) \), and use Cauchy-Schwarz to say that

\[
\left( \sum_{j \in [m]} \sqrt{L_j(x)} \right)^2 \leq \left( \sum_{j \in [m]} 1 \right) \left( \sum_{j \in [m]} L_j(x) \right) \leq mS
\]

where \( S = \frac{16 \| A \|_{\infty}^2 \log m}{\epsilon} \) is a global upper bound on \( \sum_{j \in [m]} L_j(x) \).

Next, clearly the number of iterations of \( \text{Accel-Regress} \) is bounded by

\[
NT + \sqrt{\frac{2C_2 S_{1/2}^2}{\epsilon}}
\]

where the \( NT \) comes from our runs of step 6 which don’t include terminating on \( 6c \), and step \( 6c \) takes \( \sqrt{\frac{2C_2 S_{1/2}^2}{\epsilon}} \) iterations. We have \( N = \log \frac{D}{\epsilon}, T = \tilde{O}(m) \), and \( \sqrt{\frac{2C_2 S_{1/2}^2}{\epsilon}} = \tilde{O} \left( \frac{S_{1/2} \| x^0 - x^* \|_2}{\sqrt{\epsilon}} \right) \), where \( S_{1/2} = \tilde{O} \left( \frac{\sqrt{m} \| A \|_{\infty}}{\sqrt{\epsilon}} \right) \). This proves our runtime guarantee.

Finally, we show that the point we return is an \( \epsilon \)-approximate minimizer. Clearly if step \( 6c \) is never run, it means that in every round the convergence was dominated by the function error, and after
$N$ rounds, we will have a $D^{\frac{1}{2m}}$-approximate minimizer by the guarantee that the final point of every phase halves the function error with respect to the initial point, so we are done. Furthermore, if we ever do run step 6c, then we will be in the other case of the convergence guarantee of ACCEL-REGRESS; whatever point $x$ we return will be an $\varepsilon$-approximate minimizer because the convergence was dominated by the other term, $C_2 \frac{S^2}{2}\|x^0 - x^\star\|^2_2$. We remark that we were only able to obtain this dependence on $s$ by re-initializing the dual variable to $x^0$ every time.

In order to implement this ACCEL-REGRESS in full, we need estimates on $D$ and $s$. Note that $s$ simply specifies an $\ell_2$ ball within which we are searching for $x^\star$. We can obtain $s$ via a binary search by repeatedly doubling some initial estimate, until at some point allowing $s$ to double no longer decreases our objective value by more than an $\varepsilon$ amount.

To estimate $D$, we assume that the range of our objective value is polynomially bounded (for all applications we consider, it will be). Note that if we actually initialize the algorithm with any overestimate to $D$, we can still obtain the same invariants; the number of iterations will have a dependence on $\log \frac{D}{\varepsilon}$, but if our overestimate is only polynomially worse than the true value, then this does not incur more than a constant runtime loss.

We remark that the accelerated algorithm always has a better iteration count than the unaccelerated $l_\infty$ gradient descent method, for matrices $A$ with $\|A\|_\infty \geq \Omega(1)$ and sufficiently small column sparsity. Disregarding the dependence on $\varepsilon^{-1}$, the algorithm from Theorem 3.11 has a convergence parameter that behaves at worst like $\sqrt{m}\|A\|_\infty \|x^0 - x^\star\|_2$, whereas standard gradient descent has convergence parameter roughly $m\|A\|_\infty^2 \|x^0 - x^\star\|_\infty$, because an iteration involves computing a full gradient, which takes $O(m)$ time. It suffices to observe that $\sqrt{m}\|x^0 - x^\star\|_2 \leq m\|x^0 - x^\star\|_\infty$.

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 $l_\infty$ gradient descent for the regression problem, if efficient smoothness and sampling oracles are not available. We state this result here, and give the proof for the non-box-constrained case in the appendices (for brevity, we omit the proof of the general constrained result, but it follows in a similar way from the corresponding result for composite function minimization in [QR16]).

**Theorem 3.12 (Uniform sampling unaccelerated coordinate descent)** There is an algorithm initialized at $x^0$ which computes an $\varepsilon$-approximate minimizer restricted to some $B^\varepsilon_\infty$ to the regression problem $\|Ax - b\|_\infty$ in $\tilde{O}\left(\frac{m\|A\|_\infty^2 \|x^0 - x^\star\|^2}{\varepsilon^2}\right)$ iterations.

### 3.5 Cheap iterations for $l_\infty$ regression in column-sparse $A$

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 $L^u_j(x)$, implement maintenance of the $L^u_j(x)$ and sampling by the quantities $\sqrt{L^u_j(x)}$ 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 3.13 (Efficient implementation of iterates)** Suppose we implement ACCEL-REGRESS
for some \(d\)-column-sparse \(A\). Then, throughout the lifetime of the algorithm, for some local coordinate smoothness overestimate parameters \(L^u_j(x)\) where \(x\) is an iterate, it is possible to (1) maintain the sum \(\sum_j \sqrt{L^u_j(x)}\), (2) compute for any \(j\) the value \(L^u_j(x)\), and (3) sample from the distribution \(\{p_j \propto \sqrt{L^u_j(x)}\}\) 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:**

**Maintaining smoothness overestimates.**

In this part of the proof, we describe the overestimates \(L^u_j(x)\) 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 \(\sum_j \sqrt{L^u_j(x)}\) and (2) computing the value \(L^u_j(x)\) in \(\tilde{O}(d^2)\) time for an iterate \(x\).

Recall from the discussion in Lemma 3.9 that, for an iterate \(x\), using any smoothness estimates \(L^u_j(x)\) which overestimate \(L_j(x) = \frac{\delta}{\sqrt{d}}||A_j||_\infty (\sum_i p_i^2 |A_{ij}|)\) suffices for our algorithm’s convergence, where the convergence rate is with respect to the overestimates \(L^u_j(x)\) used. We will use the estimates \(L^u_j(x) = \frac{\delta}{\sqrt{d}}||A_j||_\infty (\sum_i p_i^2 |A_{ij}|)^2\) 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 \(L^u_j(x)\) 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 \(\sum_i p_i^2 |A_{ij}| \leq (\sum_i \sqrt{p_i^2 |A_{ij}|})^2 \leq d \sum_i p_i^2 |A_{ij}|\), so the number of iterations using the \(L^u_j(x)\) 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 \(L^u_j(x)\), so to compute one on the fly for an iterate, it suffices to compute the \(d^2\) relevant cross-terms \(\sqrt{p_i^2 p_{i'}^2 |A_{ij}| |A_{ij'}|}\). The terms \(\sqrt{|A_{ij}| |A_{ij'}|}\) can all be precomputed in time \(\tilde{O}(m)\) across the columns.

Following the notation in Appendix B, in order to efficiently compute the new \(L^u_j(y^k)\), 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, following the algorithm in Appendix B, from one iterate \(y^k\) to the next, only one coordinate of the iterate changes values. So, only \(d\) updates need to be made to numerators, and the sum can also be updated in almost-constant time.

**Sampling from the distribution.**

In this part of the proof, we describe how to implement (3) sampling from the distribution proportional to \(\sqrt{L^u_j(x)}\).

For some iterate \(x\), we will describe how to sample from the distribution proportional to \(\sqrt{L^u_j(x)} = \sum_i \sqrt{p_i^2} \sqrt{||A_j||_\infty |A_{ij}|}\) in \(\tilde{O}(d)\) time. Clearly, it suffices to first sample the rows by a distribution proportional to \(\sqrt{p_i^2}\), 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 appropriately biased 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 $\sqrt{\|A_{i,j}\|_\infty |A_{ij}|}$, which can be done in $\tilde{O}(m)$ time, because the entire matrix $A$ has $\tilde{O}(m)$ entries.

4 Accelerating Maximum Flow

The primary goal of this section is to show how to implement efficient sampling and smoothness oracles for the regression problem associated with maximum flow in the presentation of [She13], and carefully analyze the runtime guarantee to demonstrate how it yields faster algorithms. The reduction to $\ell_\infty$ regression is the same as introduced in [She13], and is included for completeness.

4.1 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 $v \neq s, t$.

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, and its dual, the maximum congested cut, can be formulated as follows:

$$\begin{align*}
\min_f & \quad \|C^{-1}f\|_\infty & \text{s.t.} & \quad Bf = d, f \geq 0. \\
\max_v & \quad d^\top v & \text{s.t.} & \quad \|CB^\top v\|_1 \leq 1.
\end{align*}$$

Let $d_S \overset{\text{def}}{=} \sum_{u \in S} d_u$ and $c(S, T)$ denote the total weight of edges from $S$ to $T$. It is well-known that for the second problem, one of the threshold cuts with respect to $v$ achieves $d_S / c(S, V - S) \geq d^\top v$. Whenever the flow problem is clear from context, we will refer to any optimal flow by $f^{\text{OPT}}$.

4.2 From maximum flow to constrained $\ell_\infty$ regression

Firstly, we show how to transform the maximum flow problem into a constrained regression problem. The key tool used here is the concept of a good congestion approximator, and associated properties.

**Definition 4.1 (Congestion Approximator)** An $\alpha$-congestion approximator for $G$ is a matrix $R$ such that for any demand vector $b$, $\|Rb\|_\infty \leq \text{OPT}_b \leq \alpha \|Rb\|_\infty$.

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 4.2 (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 $\alpha$-congestion approximator $R$, for $\alpha = \tilde{O}(1)$. Furthermore, the matrix $A \overset{\text{def}}{=} 2\alpha RBC$ 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 4.3** Let $G$ be an undirected graph and $d$ be a demand vector. Assume we are given an $\alpha$-congestion approximator $R$, and the associated matrix $A = 2\alpha RBC$. Furthermore, let $2\alpha Rd \overset{\text{def}}{=} b$.

In order to approximately solve the maximum flow problem given by Equation (4.5), it suffices to approximately solve an associated box-constrained regression problem $\|Ax - b\|_\infty$ over $x \in B_c^\infty$ a nearly-constant number of times, and pay an additional $\tilde{O}(m)$ cost, which under the change of variables $x \overset{\text{def}}{=} C^{-1}f$ 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 4.2. For completeness, we will prove Lemma 4.3 in the appendices, but on a first read one may skip the proof and use the reduction statement as a black box result for the remaining analysis.

### 4.3 Implementation details for accelerated maximum flow

Here, we provide a full description of how to implement relevant machinery for applying the tools from Section 3 for accelerating the minimization of a constrained $\ell_\infty$ function to the regression problem given in Lemma 4.3. Due to the arguments presented in Appendix C, it suffices to bound the runtime of approximately solving the initial regression problem.

**Definition 4.4 (Flow regression problem)** The maximum flow regression problem asks to approximately minimize the function $f(x) = \text{smax}_t(Ax - b)$ subject to $x \in B_c^\infty$ for some $c$ and for $A$ for $\tilde{O}(1)$-column-sparse $A$ with $\|A\|_\infty = \tilde{O}(1)$.

#### 4.3.1 A tighter bound for $S_{1/2}$

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, $\sqrt{L_j} = \sqrt{\sum_i \sqrt{p_i} \sqrt{\|A_j\|_\infty |A_{ij}|}}$.

**Lemma 4.5** $S_{1/2} \leq \tilde{O}(\sqrt{n})$.

**Proof:** Let the largest $\ell_1$ 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
Then the following holds:

\[
\left( \sum_j \sum_i \sqrt{p_i |A_{ij}| A_j^*} \right)^2 = \left( \sum_j \sqrt{A_j^*} \cdot \left[ \sqrt{C_2} \cdot \left( \sum_i p_i |A_{ij}| \right) \right] \right)^2 \\
\leq C_2 \left( \sum_j A_j^* \right) \left( \sum_j \sum_i p_i |A_{ij}| \right) \\
\leq C_2 (nC_1)(\sum_i p_i ||A_i||_1) \leq C_1^2 C_2 n
\]

(4.6)

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 \( \tilde{O}(1) \).

\[\square\]

4.4 Putting it all together: faster approximate undirected maximum flow

We have all the tools we need to prove the following theorem.

**Theorem 4.6** There exists an algorithm which computes an \( \epsilon \)-approximate maximum flow of an \( m \)-edge, \( n \)-node graph \( G \) in time \( \tilde{O}(m + \sqrt{mC^{-1}f^{\text{OPT}}_2}) \), where \( f^{\text{OPT}} \) 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 4.3, the runtime is \( \tilde{O}(m) \) plus the cost of approximately solving the initial regression problem to \( \epsilon \) 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 \( x \in B_\infty^c \) for some \( c \), initializing at \( f^0 = 0 \).

Following the analysis of Theorem 3.11, 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 \( ||x^*||_2 \leq \sqrt{s} \). Per Lemma 4.5, we have \( S_{1/2} \leq \tilde{O}(\sqrt{n}) \). Finally, per Lemma 3.13, the time to implement each of these iterations is \( \tilde{O}(1) \). Putting all these pieces together, we have the desired result.

\[\square\]

4.5 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 4.7 (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 \( \lceil F \rceil \).

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 4.8 (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 $O(\epsilon F m)$.

4.5.1 Undirected uncapacitated graphs

We state several corollaries of Theorem 4.6 which apply to finding exact maximum flows in various types of undirected uncapacitated graphs. All of these results only hold with high probability.

Corollary 4.9 (Undirected graphs) There is an algorithm which finds a maximum flow in an undirected, uncapacitated graph in time $\tilde{O}(m^{5/4}n^{1/4})$.

Proof: We run the algorithm from Theorem 4.6 for $\epsilon = \frac{n^{1/4}}{m^{3/4}}$, and then run augmenting paths for $O(\epsilon m^2)$ iterations. Note that the maximum flow value and sparsity is certainly bounded by $m$, and thus this will yield a maximum flow. Furthermore the runtime of the approximate algorithm is bounded by $\sqrt{nm}$. Putting together these two runtimes yields the result. ■

Corollary 4.10 (Undirected graphs with small maximum flow value) There is an algorithm which finds a maximum flow in an undirected, uncapacitated graph with maximum flow value $F$ in time $\tilde{O}(m + \min(\sqrt{mnF^{3/4}}, m^{3/4}n^{1/4}\sqrt{F}))$.

Proof: The analysis here is the same as in Corollary 4.9, but instead we note that the bound on $s$ is $\min(m, Fn)$. If the better bound is $Fn$, our runtime is bounded by $\tilde{O}(m + \sqrt{mnF} + \epsilon F m)$, and choosing $\epsilon = \frac{n^{1/2}}{F^{1/4}m^{1/2}}$ yields the result. If the better bound is $m$, our runtime is bounded by $\tilde{O}(m + \sqrt{nm} + \epsilon F m)$, and choosing $\epsilon = \frac{n^{1/4}}{m^{1/4}\sqrt{F}}$ yields the result. ■

Corollary 4.11 (Undirected graphs with 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 $\tilde{O}(m + m^{1/2}n^{1/4}s^{3/4})$.

Proof: The analysis here is the same as in Corollary 4.9, but instead we note that the bound on the maximum flow value is also $s$. Thus, our runtime is bounded by $\tilde{O}(m + \sqrt{ns} + \epsilon s m)$, and choosing $\epsilon = \frac{n^{1/2}}{s^{1/4}m^{1/2}}$ yields the result. ■

4.5.2 Directed graphs

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 4.6 at a flow which is off from the true maximum flow by $s$ in $\ell_2^2$ distance. We give a summary of this reduction in Appendix C.2, and refer the reader to [Lin09] for a more detailed exposition.

Thus, after applying this reduction, the only difference in the runtimes given by the previous section are 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 4.12 (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 4.13 (Directed graphs with a sparse optimal flow) There is an algorithm which finds a maximum flow in a directed, uncapacitated graph in time $\tilde{O}(mn^{1/4}s^{1/4})$.

Acknowledgments

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. 1.1, 1.3

[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. 1.1, 1.3

[Bub15] Sébastien Bubeck. Convex optimization: Algorithms and complexity. Foundations and Trends in Machine Learning, 8(3-4):231–357, 2015. 1.3

[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. 1.3

[FR15] Olivier Fercoq and Peter Richtárik. Accelerated, parallel, and proximal coordinate descent. SIAM Journal on Optimization, 25(4):1997–2023, 2015. 1.3

[GR98] Andrew V. Goldberg and Satish Rao. Beyond the flow decomposition barrier. J. ACM, 45(5):783–797, 1998. 1.2, 1.2, 1.3, 2

[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. 1.3

[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. 1.2, 1.3

[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 multi-commodity 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. 1.2, 1.3, 4.2

[Lin09] Henry Lin. Reducing directed max flow to undirected max flow. Unpublished Manuscript, 2009. 4.5.2, C.2, C.4

[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. 1.3, 4.7
[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. 1.3

[LS14] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in $\tilde{O}(\text{rank})$ 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. 1, 1.2, 1.3, 1.3

[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. 1.3

[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. 1.3, 4.2

[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. 1.2, 1.3

[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. 1.3

[Nes03] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*, volume I. 2003. 1.3

[Nes05] Yurii Nesterov. Smooth minimization of non-smooth functions. *Math. Program.*, 103(1):127–152, 2005. 1.3

[Nes07] Yurii Nesterov. Dual extrapolation and its applications to solving variational inequalities and related problems. *Math. Program.*, 109(2-3):319–344, 2007. 1.3

[Nes12] Yurii Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012. 1.3

[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. 1.1

[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. 1.3, 4.2, 4.2

[QR16] Zheng Qu and Peter Richtáírik. Coordinate descent with arbitrary sampling I: algorithms and complexity. *Optimization Methods and Software*, 31(5):829–857, 2016. 1.1, 1.3, 3.3, 3.4, B, B
A Missing proofs from Section 1 and Section 2

A.1 Folklore bound on size of $\ell_\infty$-strongly-convex functions

In this section, we prove the following claim which occurs in the literature, but does not seem to usually be formally shown:

**Lemma A.1** Suppose $\psi$ is 1-strongly convex with respect to the $\ell_\infty$ norm on $[-1,1]^n$. Then,

$$\max_{x \in [-1,1]^n} \psi(x) - \min_{x \in [-1,1]^n} \psi(x) \geq \frac{n}{2}$$

Furthermore, this lower bound is tight, i.e. there is a 1-strongly convex function in the $\ell_\infty$ norm for which equality holds.

**Proof:** We will prove this by iteratively constructing a set of points $x_0, x_1, \ldots, x_n \in [-1,1]^n$ such that for all $i$ with $0 \leq i \leq n - 1$, we have

$$\psi(x_i) \leq \psi(x_{i+1}) - \frac{1}{2}$$

and consequently,

$$\psi(x_0) \leq \psi(x_n) - \frac{n}{2}$$

Let $e_i$ be the $i^{th}$ standard basis vector, namely the $n$-dimensional vector which is 1 in the $i^{th}$ coordinate and 0 elsewhere. Let $x_0 = (0,0,\ldots,0)$, the $n$-dimensional point which is 0 in every coordinate. Let $x_1^+ = x_0 + e_1$ and let $x_1^- = x_0 - e_1$, such that $x_0 = \frac{1}{2}x_1^+ + \frac{1}{2}x_1^-$. By strong convexity,

$$\psi(x_0) \leq \frac{1}{2}\psi(x_1^+) + \frac{1}{2}\psi(x_1^-) - \frac{1}{8}\|x_1^+ - x_1^-\|_\infty^2 = \frac{1}{2}\psi(x_1^+) + \frac{1}{2}\psi(x_1^-) - \frac{1}{2}$$
Consequently, it must be the case that at least one of

\[ \psi(x_0) \leq \psi(x_1^+) - \frac{1}{2} \]
\[ \psi(x_0) \leq \psi(x_1^-) - \frac{1}{2} \]

holds. Let \( x_1 \) be the point \( x_1^+ \) or \( x_1^- \) for which this holds.

More generally, suppose we have constructed \( x_0, x_1, \ldots, x_i \) in this fashion, such that \( x_i \) is 0 in the coordinates \( i + 1, i + 2, \ldots, n \). Then, let \( x_{i+1}^+ = x_i + e_{i+1} \) and let \( x_{i+1}^- = x_i - e_{i+1} \), such that \( x_i = \frac{1}{2}x_{i+1}^+ + \frac{1}{2}x_{i+1}^- \). Again by strong convexity, we have that at least one of

\[ \psi(x_i) \leq \psi(x_{i+1}^+) - \frac{1}{2} \]
\[ \psi(x_i) \leq \psi(x_{i+1}^-) - \frac{1}{2} \]

holds, and therefore we can pick one of the points \( x_{i+1}^+, x_{i+1}^- \) to be the point \( x_{i+1} \). We can clearly iteratively construct a point \( x_n \) in this fashion, proving the claim.

To show that the lower bound is tight, consider \( \psi(x) = \frac{1}{2}\|x\|^2 \). Clearly this function has range \( \frac{1}{2} \) over \([-1, 1]^n \). Furthermore, for all \( x \in [-1, 1]^n \), and arbitrary vector \( z \), we have

\[ z^T \nabla^2 \psi(x) z = z^T I z = \|z\|^2 \geq \|z\|_\infty^2 \]

where this second-order condition is well-known to be equivalent to 1-strong convexity, for twice-differentiable functions.

\[ \blacksquare \]

A.2 Convergence rates of first-order methods

In this section, we give guarantees for the convergence rates of the classical unaccelerated first-order methods of gradient descent in general norms and coordinate descent. We will use these results in proving the runtime of our unaccelerated uniform sampling algorithm for the regression problem in Appendix B.4.

A.2.1 Gradient descent in general norms

We briefly review the basic guarantees of gradient descent applied to a convex function \( f \) which is \( L \)-smooth in an arbitrary norm \( \| \cdot \| \). The general framework of gradient descent initializes at some point \( x^0 \) and iteratively maximizes the primal progress using the upper bound guaranteed by the smoothness. In particular, we perform the following update:

\[ x^{k+1} \leftarrow \arg\min_y \left\{ f(x^k) + \nabla f(x^k)^\top (y - x^k) + \frac{L}{2}\|y - x^k\|^2 \right\} \]

The \( O(\frac{1}{T}) \) convergence rate of gradient descent is well-known in the literature. We state the convergence guarantee here.

**Lemma A.2** Let \( x^T \) be the result of running gradient descent for \( T \) iterations. Then for the global minimizer \( x^* \), we have \( f(x^T) - f(x^*) \leq \frac{2L^2R^2}{T} \), where \( R = \max_{y} f(y) - f(x^0) \| y - x^* \| \).
A.2.2 Coordinate descent

Next, we briefly review the basic guarantees of randomized coordinate descent when applied to a convex function \( f \) which is \( L_j \)-smooth in the \( j \)th coordinate. Here, we analyze the convergence rate of the simple unaccelerated variant of coordinate descent where coordinate \( j \) is sampled with probability \( \frac{L_j}{S} \), where \( S \overset{\text{def}}{=} \sum_j L_j \). In particular, we perform the following update after sampling a coordinate \( j \):

\[
x_{k+1} = \arg\min_y \left\{ f(x^k) + \nabla_j f(x^k)^\top (y - x^k) + \frac{L_j}{2} |y_j - x_j^k|^2 \right\} = x^k - \frac{1}{L_j} \nabla_j f(x^k)
\]

Here, we give the convergence rate of this simple coordinate descent algorithm.

**Lemma A.3** Let \( x^T \) be the result of running gradient descent for \( T \) iterations. Then for the global minimizer \( x^* \), we have \( f(x^T) - f(x^*) \leq \frac{2S^2 R^2}{T^2} \), where \( R = \max_{y: f(y) \leq f(x^*)} \|y - x^*\|_2 \).

We remark that for any randomized iterative method for minimizing a convex function which converges in expectation, it is easy to use Markov’s inequality to bound the convergence with constant probability. For example, if an algorithm terminates with a \( \epsilon \)-approximate minimizer on expectation, with probability at least \( \frac{1}{2} \) it terminates with a \( 2\epsilon \)-approximate minimizer. Thus, if one desires a high probability result for the approximate minimization, the runtime only incurs a logarithmic multiplicative loss in the failure probability.

A.3 Proof of Lemma A.2

First we give an intermediate progress bound which will be useful in the final proof.

**Lemma A.4** \( f(x^k) - f(x^{k+1}) \geq \frac{1}{2L} \|\nabla f(x^k)\|_2^2 \)

**Proof:** We will prove that \( \min_y \left\{ \nabla f(x)^\top (y - x) + \frac{L}{2} \|y - x\|^2 \right\} \leq -\frac{1}{2L} \|\nabla f(x)\|_2^2 \); clearly this yields the desired claim. Let \( z \) be such that \( \|z\| = 1 \) and \( z^\top \nabla f(x) = \|\nabla f(x)\|_1 \), by the definition of dual norm; let \( y = x - \frac{\|\nabla f(x)\|_2}{L} z \). Then,

\[
\nabla f(x)^\top (y - x) + \frac{L}{2} \|y - x\|^2 = -(\frac{\|\nabla f(x)\|_2}{L}) z^\top \nabla f(x) + \frac{L}{2} \|\nabla f(x)\|_2^2 \|z\|^2 = -\frac{1}{2L} \|\nabla f(x)\|_2^2
\]

Thus, the minimizer of the upper bound yields the desired progress result. 

Next, we prove Lemma A.2.

**Proof:** Let \( \epsilon_k \overset{\text{def}}{=} f(x^k) - f(x^*) \). Note that by convexity and Cauchy-Schwarz, we have

\[
f(x^k) - f(x^*) \leq \nabla (f(x^k))^\top (f(x^k) - f(x^*)) \leq \|f(x^k)\|_1^\top \|x^k - x^*\|
\]

Thus, we have the two equations \( \epsilon_k - \epsilon_{k+1} \geq \frac{1}{2L} \|\nabla f(x^k)\|_2^2 \) and \( \epsilon_k \leq R \|\nabla f(x^k)\|_2^2 \). Combining the two, it’s easy to see that

\[
\epsilon_k^2 \leq 2LR^2 (\epsilon_k - \epsilon_{k+1}) \leftrightarrow \left( \frac{1}{\epsilon_{k+1}} - \frac{1}{\epsilon_k} \right) \geq \frac{\epsilon_k}{2LR^2 \epsilon_{k+1}} \geq \frac{1}{2LR^2}
\]

Thus, telescoping we have \( \frac{1}{\epsilon_T} \geq \frac{T}{2LR^2} \), which yields the desired rate of convergence. 

\[\tag*{\blacksquare}\]
A.4 Proof of Lemma A.3

The progress of a step in the \( j^{th} \) coordinate is thus lower bounded by \(-\frac{1}{2L_j}|\nabla_j f(x^k)|^2\), which can be verified by computing the upper bound on \( f(x^{k+1}) \). The analysis of convergence follows directly from the following result on the expected progress of a single step.

**Lemma A.5** \( f(x^k) - \mathbb{E}_k[f(x^{k+1})] \geq \frac{1}{2S}||\nabla f(x^k)||_2^2 \)

**Proof:** We directly compute the expectation. We have

\[
\mathbb{E}[f(x^{k+1})] = \sum_j \frac{L_j}{S} \left( f(x^k) - \frac{1}{2L_j}|\nabla_j f(x^k)|^2 \right) = f(x^k) - \frac{1}{2S} \sum_j |\nabla_j f(x^k)|^2
\]

Thus, we can immediately plug in this expected progress result into the convergence rate proof of gradient descent, and obtain the desired result.

B Missing proofs from Section 3

In this section, we prove Theorem 3.10, giving the algorithm AP-CD and analyzing its convergence. Throughout, fix some constant \( c \), and define the function \( F \) so that \( F(x) = f(x) \) for \( x \in B_\infty^c \), and \( F(x) = \infty \) otherwise. The following analysis is largely copied from [QR16] with certain parts adapted for our setting. The two major changes in our algorithm and analysis as compared with [QR16] are that we sample coordinates based on local coordinate smoothnesses, and support the ability to initialize with different primal and dual points (which correspondingly gives a different error guarantee, where the function error is based on the initial primal point and the domain dependence is based on the initial dual point).

Note that in the assumptions of the algorithm, we maintain a global upper bound \( S_{1/2} \) for the sum \( S_{loc} = \sum_j \sqrt{L_j(y^k)} \) for iterates \( y^k \) where \( L_j(y^k) \) are the local smoothness parameters. We assume that we can maintain the local sum \( S_{loc} \), thus we can scale up all the local smoothnesses in calculations by \( \frac{S_{1/2}^2}{S_{loc}^2} \) so that the local sum is always \( S_{1/2} \). This clearly does not affect any of the sampling probabilities, but will be required in the convergence analysis. Formally, for the local coordinate smoothness parameters \( L_j(y^k) \) we maintain in our algorithm, for just the \( k \)th iteration, we will instead use the local coordinate smoothness overestimates \( \frac{S_{1/2}^2}{S_{loc}^2} L_j(y^k) \).

First we state the algorithm in full, which requires as input a global upper bound constant \( S_{1/2} \) on all values \( S_{loc} = \sum_j \sqrt{L_j(y^k)} \) throughout the algorithm, for iterates \( y^k \), a choice of primal point \( x^0 \) and a dual point \( z^0 \).

For the rest of this section, we will use \( L_j \) as shorthand to denote \( L_j(y^k) \) for a particular iterate \( y^k \), when the iterate is clear from context. Before we begin, we claim that we can assume that without loss of generality that \( \theta_0 = \Theta(\frac{1}{m}) \). To see this, instead of sampling by the distribution given by the \( \sqrt{\sum_j} \), we can instead use the average of the \( \sqrt{L_j} \) for each \( j \). Then, we treat the smoothness estimates as upper estimates \( \hat{L}_j = (\sqrt{L_j} + L_{1/2})^2 \) instead. Now, if we sample by the \( \sqrt{\hat{L}_j} \) instead, note that the sum does not go up by more than a factor of two, but each individual sampling probability is lower bounded by \( \frac{1}{2m} \), as desired. Also, this sampling is easy to do; we simply sample the original probabilities with half probability, and uniformly with half probability. Computing \( \hat{L}_{1/2} \) is easy since we maintain the sum.
\[ x^T = \text{AP-CD}(f,c,x^0,\gamma^0,S_{1/2},T) \]

1. Initialize \( \theta_0 = \min_j p_j \).
2. Iterate for \( k = 0,1,\ldots,T-1: \)
   (a) Define \( S_{\text{loc}} = \sum_j \sqrt{L_j(y^k)} \), reweight the \( L_j(y^k) \) by \( \frac{S_{1/2}^2}{S_{\text{loc}}} \), and fix sampling probabilities \( p_j = \frac{\sqrt{L_j(y^k)}}{S_{1/2}} \).
   (b) For \( k > 0 \), let \( \theta_k = \sqrt{\frac{\theta_{k-1}^4 + 4\theta_{k-1}^2 - \theta_{k-1}^2}{2}} \).
   (c) Let \( y^k = (1 - \theta_k)x^k + \theta_k z^k \).
   (d) Let \( z^{k+1} = z^k \).
   (e) Sample a coordinate \( j \) from the probability distribution \( \{ p_j \} \).
   (f) Set \( z_j^{k+1} = \arg\min_{z \in [-c,c]} \{ (\nabla_j f(y^k), z) + \frac{\theta_k L_j(y^k)}{2p_j} |z - z_j^k|^2 \} \).
   (g) Let \( x^{k+1} = y^k + \theta_k p_j^{-1} (z^{k+1} - z^k) \).
3. Return \( x^T \).

Figure 2: The accelerated algorithm for minimizing a composite function with separable proximal term.

Now, in the style of the presentation in [QR16], we give the convergence guarantee. As an overview of the organization of the following sections, Appendix B.1 gives some technical lemmas which are used in the final analysis, the first of which shows that all of the primal iterates \( \{x^k\} \) and the dual iterates \( \{z^k\} \) lie in the feasible region \( B_{\infty}^c \). Therefore, for all iterates, the composite function \( F \) is actually identically just \( f \), which greatly simplifies the analysis. Finally, we give a one-step analysis of a potential function which leads to a final convergence rate in Appendix B.2.

B.1 Technical lemmas

Lemma B.1 shows that each individual coordinate \( x_j^k \) of the variable \( x^k \) is a convex combination of all the history blocks \( z_j^0, \ldots, z_j^k \), and the initial point coordinate \( x_j^0 \). Correspondingly, assuming the initial point \( x^0 \) is in \( B_{\infty}^c \), it is easy to see that \( x^k \) is also feasible and thus \( F(x^k) = f(x^k) \ \forall k \).

**Lemma B.1** For all \( k \in \mathbb{N} \) and \( j \in [m] \) we have

\[ x_j^k = \sum_{l=0}^{k} \gamma_{j,l}^k z_j^l + \beta_j^k x_j^0, \]

where for each \( j \), the coefficients \( \{ \gamma_{j,l}^k \}_{l=0,\ldots,k}, \beta_j^k \) are defined recursively by setting \( \beta_j^0 = 1 \), \( \gamma_{j,0}^0 = 0 \), and for \( k \geq 1 \),

\[ \beta_j^{k+1} = (1 - \theta_k) \beta_j^k, \quad \gamma_{j,k+1}^l = \begin{cases} (1 - \theta_k) \gamma_{j,l}^k & l = 0,\ldots,k - 1 \\ (1 - \theta_k) \gamma_{j,k}^k + \theta_k - \theta_k p_j^{-1} & l = k \\ \theta_k p_j^{-1} & l = k + 1 \end{cases} \quad (2.7) \]

Moreover, for all \( k \in \mathbb{N} \) and \( j \in [m] \), the coefficients \( \{ \gamma_{j,l}^k \}_{l=0,\ldots,k} \) and \( \beta_j^k \) are all nonnegative and sum to 1. Consequently, \( x_j^k \) is a convex combination of \( x_j^0, \{ z_j^l \}_{l=0,\ldots,k} \), and \( F(x_j^k) = f(x_j^k) \) for all iterations \( k \) and coordinates \( j \).

**Proof:** Fix any \( j \in [m] \). We proceed by induction on \( k \). It is clear by definition that \( \beta_j^0 = 1 \) and
\[ \gamma_j^{0,0} = 0. \] Now, let’s assume that (2.7) holds for some \( k \geq 1 \). Then,

\[
x_j^{k+1} = y_j^k + \theta_k p_j^{-1}(z_j^{k+1} - z_j^k) = (1 - \theta_k) x_j^k + \theta_k z_j^k - \theta_k p_j^{-1} z_j^k + \theta_k p_j^{-1} z_j^{k+1}
\]

From the above facts, we deduce that the coefficients \( \beta_j \)

\[
\gamma \sum_{l=0}^{k-1} \gamma_j^{k+1,l} = (1 - \theta_k) \left( \beta_j^k + \sum_{l=0}^{k-1} \gamma_j^{k+1,l} \right) + (1 - \theta_k) \gamma_j^{k+1} + \theta_k p_j^{-1} + \theta_k p_j^{-1} = (1 - \theta_k) + \theta_k
\]

Therefore the recursive equation (2.7) holds for \( k + 1 \) as well. Next we show that the linear combination in (2.7) is a convex combination. Clearly, it is for \( k = 0 \). Let \( k \geq 1 \). We proceed by induction. By the recursive relationship we have derived in (2.7), we have:

\[
\beta_j^{k+1} + \sum_{l=0}^{k+1} \gamma_j^{k+1,l} = \beta_j^k + \sum_{l=0}^{k-1} \gamma_j^{k+1,l} + \gamma_j^{k+1} + \gamma_j^{k+1,k+1}
\]

We deduce from the above facts that the coefficients \( \beta_j^{k+1}, \{\gamma_j^{k+1,l}\}_{l=0}^{k+1} \) sum to 1. Moreover, it is clear from induction and \( 0 \leq \theta_k \leq \min_j p_j \) that all the coefficients are nonnegative if the same holds for \( \beta_j^k, \{\gamma_j^k\}_{l=0}^{k} \). Since \( \theta_0 = \min_j p_j \), and the \( \theta_k \) are decreasing, we conclude that the coefficients do indeed form a convex combination for all iterations \( k \) and coordinates \( j \).

Finally, we show that \( F(x^k) = f(x^k) \) \( \forall k \). Indeed, by the definition of \( z_j^k \), it must be the case that \( z_j^k \in B_{\infty}^k \), and only this coordinate changed from \( z_j^{k-1} \), so \( z_j^k \) is feasible. Furthermore, note that because all of the \( y_j^k \) are convex combinations of \( \{z_j^k\} \) and \( x_j^0 \) for some feasible \( x^0 \), it must be the case that they are also feasible (in \( B_{\infty}^k \) because the feasible region is a convex set, as desired.

For the rest of this section, we define point

\[
z_j^{k+1} = \arg\min_{z \in B_{\infty}^k} \{ \langle \nabla f(y^k), z \rangle + \|z - z^k\|_{\theta_k L^2}^2 \}
\]

i.e. the full “mirror descent” step in the algorithm over all coordinates. In particular, with probability \( p_j \), \( z_j^{k+1} = z_j^k \).

**Lemma B.2** Let

\[
\xi(z) \overset{\text{def}}{=} f(y^k) + \langle \nabla f(y^k), z - y^k \rangle + \frac{\theta_k}{2} \|z - z^k\|^2_{p^{-1} L}, \ z \in \mathbb{R}^m.
\]

Then we have:

\[
\xi(z^{k+1}) \leq \xi(y) - \frac{\theta_k}{2} \|z^{k+1} - y\|^2_{p^{-1} L}, \ y \in \mathbb{R}^m.
\]

**Proof:** First, note that by the first-order optimality condition,

\[
0 \leq \langle \nabla \xi(z^{k+1}), y - z^{k+1} \rangle
\]

29
Thus, clearly it suffices to prove that
\[ \xi(y) - \xi(\tilde{z}^{k+1}) - \frac{\theta_k}{2} \|\tilde{z}^{k+1} - y\|_{p^{-1} \circ L}^2 = \langle \nabla \xi(\tilde{z}^{k+1}), y - \tilde{z}^{k+1} \rangle \]
This can be verified up to direct calculation. \hfill \blacksquare

### B.2 Recursion

By the arguments in the previous section, we have shown that \( F(x^k) = f(x^k) \) for all iterates \( x^k \), because the iterates are feasible and thus the composite term is identically 0. For the following lemma, fix an iterate \( k \) and let \( \{L_j\} \) be the local smoothness estimates at the iterate, after correction such that \( \sum_j \sqrt{L_j} = S_{1/2} \).

**Lemma B.3** For the sequence of iterates produced by AP-CD and all \( k \geq 0 \), and any point \( y \in \mathbb{R}^m \), the following recursion holds:

\[
\mathbb{E}_k \left[ F(x^{k+1}) + \frac{\theta_k^2 S_j^2}{2} \| z^{k+1} - y \|_2^2 \right] \leq \left[ F(x^k) + \frac{\theta_k^2 S_j^2}{2} \| z^k - y \|_2^2 \right] - \theta_k (F(x^k) - F(y)) \quad (2.9)
\]

**Proof:** Based on our sampling procedure and smoothness assumptions, and choice of \( y^k \):

\[
\mathbb{E}_k[f(x^{k+1})] = \mathbb{E}_k[f(y^k + \theta_k p^{-1} \cdot (\tilde{z}_j^{k+1} - z_j^k))]
\]

\[
\leq f(y^k) + \theta_k (\langle \nabla f(y^k), \tilde{z}_j^{k+1} - z_j^k \rangle + \frac{\theta_k^2}{2} \| \tilde{z}_j^{k+1} - z_j^k \|_{p^{-1} \circ L}^2)
\]

\[
= (1 - \theta_k) (f(y^k) + \langle \nabla f(y^k), z^k - y^k \rangle) + \theta_k (f(y^k) + \langle \nabla f(y^k), \tilde{z}_j^{k+1} - y^k \rangle + \frac{\theta_k}{2} \| \tilde{z}_j^{k+1} - z_j^k \|_{p^{-1} \circ L}^2).
\]

where in the last line we used that by definition, \((1 - \theta_k)(x^k - y^k) = -\theta_k (z^k - y^k)\). We then bound the expectation of \( F(x^{k+1}) \) as follows, using that for all iterates, \( F = f \):

\[
\mathbb{E}_k[F(x^{k+1})]
\]

\[
\leq (1 - \theta_k) (f(y^k) + \langle \nabla f(y^k), z^k - y^k \rangle) + \theta_k (f(y^k) + \langle \nabla f(y^k), \tilde{z}_j^{k+1} - y^k \rangle + \frac{\theta_k}{2} \| \tilde{z}_j^{k+1} - z_j^k \|_{p^{-1} \circ L}^2)
\]

\[
(2.8) \leq (1 - \theta_k) (f(y^k) + \langle \nabla f(y^k), x^k - y^k \rangle) + \theta_k (f(y^k) + \langle \nabla f(y^k), y - y^k \rangle + \frac{\theta_k}{2} \| z^k - y \|_{p^{-1} \circ L}^2 - \theta_k \| \tilde{z}_j^{k+1} - y \|_{p^{-1} \circ L}^2)
\]

\[
\leq (1 - \theta_k) F(x^k) + \theta_k F(y) + \frac{\theta_k^2}{2} (\| z^k - y \|_{p^{-1} \circ L}^2 - \| \tilde{z}_j^{k+1} - y \|_{p^{-1} \circ L}^2)
\]

\[
= (1 - \theta_k) F(x^k) + \frac{\theta_k^2}{2} \mathbb{E}_k \left[ \| z^k - y \|_{p^{-1} \circ L}^2 - \| y - z^{k+1} \|_{p^{-1} \circ L}^2 \right], \forall y \in \mathbb{R}^m.
\]
The last equality can be verified by explicitly writing out the expectation. Furthermore, we use that the squared norms \( \| \cdot \|^2_{p-2L} \) and \( S^2_{1/2} \) are equal. Therefore, for all \( y \in \mathbb{R}^m \),
\[
\mathbb{E}_k \left[ F(x^{k+1}) - F(y) + \frac{\theta_k^2 S^2_{1/2}}{2} \| y - z_k^{1+1} \|^2_2 \right] \leq (1 - \theta_k)(F(x^k) - F(y)) + \frac{\theta_k^2 S^2_{1/2}}{2} \| z_k^k - y \|^2_2 .
\]

\[\Box\]

**B.3 Proof of Theorem 3.10**

We prove the following intermediate statement first:

\[
\mathbb{E} \left[ F(x^k) - F(y) \right] \leq \frac{4 \left( (1 - \theta_0)(F(x^0) - F(y)) + \frac{\theta_k^2 S^2_{1/2}}{2} \| z^0 - y \|^2_2 \right)}{(\theta_0(k-1) + 2)^2} \tag{2.10}
\]

This follows the standard acceleration analysis. Let \( \phi_k \overset{\Delta}{=} \mathbb{E}[F(x^k)] - F(y) \), and let \( r_k \overset{\Delta}{=} \frac{S^2_{1/2}}{2} \| z^k - y \|^2_2 \). The statement of Lemma B.3 can be restated as \( \phi_{k+1} + \theta_k^2 r_k \leq (1 - \theta_k) \phi_k + \theta_k^2 r_k \). Further, note that our choice of the \( \theta_k \) has \( \frac{1 - \theta_{k+1}}{\theta_{k+1}} = \frac{1}{\theta_k^2} \), and \( \theta_k \leq \frac{2}{k+2}/\theta_0 \). Thus,
\[
\frac{1 - \theta_{k+1}}{\theta_{k+1}} \phi_{k+1} + r_{k+1} \leq \frac{1 - \theta_k}{\theta_k^2} \phi_k + r_k \leq \frac{1 - \theta_0}{\theta_0^2} \phi_0 + r_0
\]

Now, we wish to bound \( \phi_k \). As such, note that
\[
\phi_k \leq \frac{(1 - \theta_0)\theta_k^2 \phi_{k-1}}{\theta_k^2} + \theta_k^2 r_k \leq \frac{(1 - \theta_0)\theta_k^2 \phi_{k-1}}{\theta_0^2} + \theta_k^2 r_k = \frac{\theta_k^2}{\theta_0^2}((1 - \theta_0)\phi_0 + \theta_0^2 r_0)
\]

Using our bound on \( \theta_k \) yields the desired Equation (2.10). Hence, Theorem 3.10 holds for \( y \) the minimizer of the function, by the fact that we fixed the smoothness estimates so that \( \theta_0 = \Theta(\frac{1}{m}) \).

**B.4 Proof of Theorem 3.12, unconstrained case**

\[ x^T = \text{UNIFORM-CD}(f, x^0, S, T) \]

1. Initialize the vector \( p^0 \) such that \( p^0_j = \frac{\exp(x^j_k/t)}{\exp(x^j_k/t)} \).
2. Iterate for \( k = 0, 1, \ldots, T - 1 \):
   
   (a) Sample a coordinate \( j \in [m] \) uniformly at random.
   (b) Compute \( L_j(x^k) = \frac{\delta}{T}(p^k)^\top [A_j]\| A_j \|_\infty \).
   (c) Let \( x^{k+1} = x^k - \frac{1}{2L_j(x^k)} \nabla_j f(x^k) \).
   (d) Let \( p^{k+1} \) be \( p^k \) with an updated \( j^{th} \) coordinate.
3. Return \( x^T \).

![Figure 3: Uniform sampling local smoothness coordinate descent algorithm for \( \ell_\infty \) regression.](image)

In this section, we give a proof that uniformly sampling coordinates suffices to match the guarantees of the standard \( \ell_\infty \) gradient descent algorithm on the (smoothed) \( \ell_\infty \) regression problem. The algorithm is the standard unaccelerated coordinate descent algorithm using local coordinate...
smoothnesses. Here, assume \( f \) is the smoothed function \( \max_t(Ax - b) \), for \( A \in \mathbb{R}^{n \times m}, b \in \mathbb{R}^n \), and \( t = \frac{1}{2\log m} \). We remark that it is easy to maintain the value of \( p^k \) for the iterates \( k \) in nearly-constant time by simply maintaining a binary tree data structure with the values of \( \exp(x_j^k/t) \) at the leaves, and updating one leaf and the sum after each iterate.

First, we show the useful helper lemma.

**Lemma B.4** For nonnegative vector-valued \( a, b \), we have

\[
\sum_j \frac{a_j^2}{b_j} \geq \frac{\|a\|_1^2}{\|b\|_1}.
\]

**Proof:** This follows directly from Cauchy-Schwarz, whence

\[
\|a\|_1^2 = \left( \sum_j a_j \right)^2 \leq \left( \sum_j \frac{a_j}{\sqrt{b_j}} \cdot \sqrt{b_j} \right)^2 = \left( \sum_j \frac{a_j^2}{b_j} \right) \cdot \|b\|_1.
\]

Finally, the convergence given by Theorem 3.10 follows from lower bounding the progress of a single step. From the definition of local coordinate smoothness, Lemma A.5, and Lemma B.4, we have

\[
\mathbb{E}[f(x^{k+1})] - f(x^k) \geq \frac{1}{m} \sum_j \nabla_j f(x^k)^2 \geq \frac{1}{m} \frac{\|\nabla f(x^k)\|_1^2}{2L_j(x^k)}.
\]

Thus, the expected progress of a single step is lower bounded by

\[
O \left( \frac{\|\nabla f(x^k)\|_1^2}{2m \epsilon \|A\|_\infty^2} \right)
\]

where we use the trace bound from Lemma 3.9 to upper bound \( \sum_j L_j(x^k) \) by \( O \left( \frac{\|A\|_\infty^2}{\epsilon} \right) \). The result follows by applying Lemma A.2. In conclusion, an unaccelerated coordinate descent method which samples coordinates uniformly at random matches the runtime of naive gradient descent in the \( \ell_\infty \) norm for the regression problem, up to a \( O(1) \) factor.

C Missing proofs from Section 4

C.1 Reducing undirected maximum flow to \( \ell_\infty \) regression

In this section, we prove Lemma 4.3, via giving the reduction and analyzing its convergence. First, suppose we have a subroutine, \( \text{Almost-Route} \), which takes in matrices \( R \) (an \( \alpha = \tilde{O}(1) \)-congestion approximator), \( B \) (an edge-incidence matrix), \( C \) (the capacities of edges), \( \alpha \), an error tolerance \( \epsilon \), and a demand vector \( d \), and returns some \( x \) such that

\[
2\alpha \|BCx - Rd\|_\infty + \|x\|_\infty \leq (1 + \epsilon)(2\alpha \|BCx^* - Rd\|_\infty + \|x^*\|_\infty) \quad \text{def} = (1 + \epsilon)\text{OPT}(d)
\]
Here, under a change of variables we have that $x = C^{-1}f$. Note that we are writing with an $\epsilon$-multiplicative approximation to OPT instead of an additive one. We do this without loss of generality: assume we have scaled the problem appropriately so that the optimal value is 1, which will be when we find the true maximum flow instead of the minimum congestion flow. We can find this optimal value via a binary search, as we argued before, losing a $\tilde{O}(1)$ factor in the runtime.

Now, we show a key property of the function we try to minimize. Intuitively, the next lemma says that if we are able to $\epsilon$-approximately minimize our regression problem, the cost of routing the residual demands $d - Bf$ is only an $\epsilon$ fraction of routing the original demands, allowing us to quickly recurse. This is a restatement of Lemma 2.2 in [She13].

**Lemma C.1** Define the change of variables $Cx = f$. Suppose $2\alpha\|RBCx - Rd\|_\infty + \|x\|_\infty = 2\alpha\|R(d - Bf)\|_\infty + \|C^{-1}f\|_\infty \leq (1 + \epsilon)\text{OPT}(d)$. Then, $\|R(d - Bf)\|_\infty \leq \epsilon\|Rd\|_\infty$.

**Proof:** Let $f'$ be the optimal routing of the residual demands $d - Bf$, namely the argument which achieves $\text{OPT}(d - Bf)$. Then, $Bf' = d - Bf$, and by the definition of a congestion approximator,

$$\text{OPT}(d - Bf) = \|C^{-1}f'\|_\infty + 2\alpha\|R((d - Bf) - Bf')\|_\infty = \|C^{-1}f'\|_\infty \leq \alpha\|R(d - Bf)\|_\infty$$

(3.12)

For simplicity we write $d' \overset{\text{def}}{=} d - Bf$. Furthermore, we have by assumption of the quality of the initial solution $f$,

$$\text{OPT}(d) + \alpha\|Rd'\|_\infty \leq \|C^{-1}(f + f')\|_\infty + 2\alpha\|R(d - B(f + f'))\|_\infty + \alpha\|Rd'\|_\infty$$

(3.13)

$$\leq \|C^{-1}f\|_\infty + \|C^{-1}f'\|_\infty + \alpha\|Rd'\|_\infty$$

(3.14)

$$\leq \|C^{-1}f\|_\infty + 2\alpha\|Rd'\|_\infty \leq (1 + \epsilon)\text{OPT}(d)$$

(3.15)

Here, we used that $d = B(f + f')$ and our bound $\|C^{-1}f'\|_\infty \leq \alpha\|Rd'\|_\infty$. Subtracting $\text{OPT}(d)$, and noting that $\text{OPT}(d) \leq \alpha\|Rd\|_\infty$, we have the desired claim.

Now, we give the full reduction to calling ALMOST-Routing. Note that it was shown in [She13] that routing through a maximal spanning tree yields an $O(m)$-congestion approximator.

$ff_{\text{final}} = \text{FLOW-TO-REGRESS}(G, d, \epsilon)$

1. Let $T = \log 2m$.
2. Initialize $d^0 = d$. Initialize $f^0 = \text{CALMOST-Routing}(R, B, C, d^0, \alpha, \epsilon)$.
3. Let $f_{\text{final}} = f^0$.
4. Iterate for $k = 1, 2, \ldots T$:
   (a) Let $d^k = d^{k-1} - Bf^{k-1}$.
   (b) Let $f^k = \text{CALMOST-Routing}(R, B, C, D^k, \alpha, \frac{1}{2})$.
   (c) Let $f_{\text{final}} = f_{\text{final}} + f^k$.
5. Let $f^{T+1}$ be an (exact) routing of $d^k - Bf^k$ in a maximal spanning tree. Let $f_{\text{final}} = f_{\text{final}} + f^{T+1}$.
6. Return $f_{\text{final}}$

Figure 4: The reduction from solving the approximate maximum flow problem to solving $\tilde{O}(1)$ approximate regression problems.

We now need to prove the correctness of our algorithm. This is a restatement of ideas presented in [She13].
Lemma C.2 The output of Flow-To-Regress is an $\epsilon$-approximate solution to the minimum congestion flow problem.

Proof: By the guarantees of Almost-Route, we have the following guarantees:

$$\|C^{-1}f^0\|_{\infty} + 2\alpha\|Rd\|_{\infty} \leq (1 + \epsilon)\text{OPT}(d) \tag{3.16}$$

$$\|C^{-1}f^k\|_{\infty} + 2\alpha\|Rd^{k+1}\|_{\infty} \leq \frac{3}{2}\text{OPT}(d^k) \leq \frac{3}{2}\alpha\|Rd^k\|_{\infty}, k \geq 1 \tag{3.17}$$

Now, using the second inequality and repeatedly applying it to the first, we have the following guarantee:

$$\frac{1}{2}\alpha\|Rd^1\|_{\infty} + \|C^{-1}f^0\|_{\infty} + \ldots + \|C^{-1}f^T\|_{\infty} \leq (1 + \epsilon)\text{OPT}(d) \tag{3.18}$$

It suffices to note that by our choice of $T$ and seeing that by applying Lemma C.1 $T$ times, we have $\alpha\|Rd^T\|_{\infty} \leq \frac{1}{2m}\alpha\|Rd\|_{\infty}$. Thus because we routed $d^{T+1}$ exactly through a $m$-congestion approximator, we have $\|C^{-1}f^{T+1}\|_{\infty} \leq \frac{1}{2}\alpha\|Rd^{T+1}\|_{\infty}$. Finally, $Bf^{\text{final}} = d$, and

$$\|C^{-1}f^{\text{final}}\|_{\infty} \leq \|C^{-1}f^T\|_{\infty} + \|C^{-1}f^0\|_{\infty} + \ldots + \|C^{-1}f^T\|_{\infty} \tag{3.19}$$

$$\leq \frac{1}{2}\alpha\|Rd^1\|_{\infty} + \|C^{-1}f^0\|_{\infty} + \ldots + \|C^{-1}f^T\|_{\infty} \tag{3.20}$$

$$\leq (1 + \epsilon)\text{OPT}(d) \tag{3.21}$$

Lemma C.3 The runtime of our routine Flow-To-Regress is the cost of solving the first associated regression problem, $2\alpha\|RBCx - Rd\|_{\infty} + \|x\|_{\infty}$, to an $\epsilon$-approximation, plus an additional $\tilde{O}(m)$ additive overhead.

Proof: We analyze the time of each of the calls to Almost-Route. Clearly, the first call is the cost of solving the first associated regression problem.

Note that we have flexibility in terms of how to implement Almost-Route; for all remaining calls, we consider the implementation in the form of unaccelerated gradient descent in the $\ell_\infty$ norm. The runtime as we demonstrated in Appendix A.2.1 for each round $k$ is

$$\frac{m\|f^k\|_2^2\|\alpha RBC\|_2^2}{(\frac{\epsilon}{2})^2} = \tilde{O}(m) \tag{3.22}$$

where $f^k$ is the optimal solution to the $k^{th}$ regression problem. Here, we used the known properties of $\alpha RBC$, as well as the fact that the implications of Lemma C.1 allow us to bound the $\ell_\infty$ norm of the optimal solution by $O(1)$ as well.

As a final remark in the proof of Lemma 4.3, we note that to optimize the first objective $2\alpha\|Ax - b\|_{\infty} + \|x\|_{\infty}$ it suffices to binary search over values $c \geq \|C^{-1}f\|_{\infty} = \|x\|_{\infty}$, and solve the associated regression problem $\|Ax - b\|_{\infty}$ over $x \in B_{c^\infty}$. This only incurs a multiplicative loss in the runtime by a factor of $\tilde{O}(1)$, due to the binary search.
C.2 Reducing directed maximum flow to undirected maximum flow

In this section, we give an overview of the main result in [Lin09]. In particular, we prove the following statement, which is used in our algorithms for finding exact maximum flows in unit-capacity graphs.

**Lemma C.4 (Summary of results in [Lin09])** Suppose we wish to find an \( s-t \) maximum flow in a unit-capacity directed (multi)graph \( G \) with \( m \) edges and maximum flow value \( F \). Then, it suffices to find the \( s-t \) maximum flow \( f_{\text{max}} \) in an undirected (multi)graph \( G' \) with \( O(m) \) edges, such that edges of \( G' \) have capacity \( \frac{1}{2} \), and the maximum flow in \( G' \) has value \( F + \frac{m}{2} \). Furthermore, we are able to initialize the undirected maximum flow algorithm in \( G' \) with some \( f_{\text{init}} \) such that \( \| f_{\text{init}} - f_{\text{max}} \|^2 = F \).

**Proof:** First, we give the construction of the undirected graph \( G' \). For every directed edge \((u,v)\) of weight 1 in \( G \), \( G' \) has the undirected edges \((s,v),(v,u)\), and \((u,t)\) of weight \( \frac{1}{2} \). Clearly, \( G' \) has \( O(m) \) edges, since each edge in \( G \) is replaced with 3 edges in \( G' \).

Next, we give the (algorithmic) proof that one can recover a maximum flow in \( G \) from a maximum flow in \( G' \) and that the maximum flow in \( G' \) has value \( F + \frac{m}{2} \). Consider the following algorithm.

\[
\begin{align*}
  f &= \text{UMF-to-DMF}(G) \\
  1. & \text{Let } G' \text{ be the undirected graph with edges } (s,v),(v,u),(u,t) \text{ of weight } \frac{1}{2} \text{ for every directed edge } (u,v) \text{ in } G. \\
  2. & \text{Let } f_{\text{init}} \text{ be the flow which puts } \frac{1}{2} \text{ units of flow on each of the } (s,v),(v,u),(u,t). \\
  3. & \text{Compute } f_{\text{final}}, \text{ the maximum flow of } G'. \\
  4. & \text{Return } f_{\text{final}} - f_{\text{init}}.
\end{align*}
\]

**Figure 5:** Recovering a maximum flow in directed graph \( G \) via computing a maximum flow in undirected graph \( G' \).

We will now prove correctness of the algorithm UMF-to-DMF, namely that \( f_{\text{final}} - f_{\text{init}} \) is a maximum flow in graph \( G \). To do so, we show that \( f_{\text{final}} \) has value \( \frac{m}{2} + F \), and that \( f_{\text{final}} - f_{\text{init}} \) puts flow only in the \((u,v)\) direction and does not put any flow on any new edges \((s,v)\) or \((u,t)\).

Note that this immediately implies the statement \( \| f_{\text{init}} - f_{\text{max}} \|^2 = F \).

We begin by showing that \( f_{\text{final}} \) has value \( \frac{m}{2} + F \). The residual graph of \( G' \) with respect to the flow \( f_{\text{init}} \) is the directed graph \( G \). Thus, the maximum flow in the residual graph has value \( F \) by assumption, and the flow \( f_{\text{init}} \) has value \( \frac{m}{2} \), yielding the conclusion.

Next, we show that for every edge \((u,v)\) in \( G' \) which resulted from a directed edge \((u,v)\) in \( G \), \( f_{\text{final}} - f_{\text{init}} \) puts flow only in the \((u,v)\) direction, and does not violate the capacity constraint. This is simple to see because \( f_{\text{final}} \) puts a flow with value in \( \{-\frac{1}{2},0,\frac{1}{2}\} \) in the \((u,v)\) direction, and \(-f_{\text{init}}\) puts a flow with value \( \frac{1}{2} \) in the \((u,v)\) direction; adding yields the result.

Finally, we show that \( f_{\text{final}} - f_{\text{init}} \) puts no flow on any of the new edges \((s,v)\) (the same statement holds for edges \((u,t)\) by a similar argument). Again, \( f_{\text{final}} \) puts a flow with value in \( \{-\frac{1}{2},0,\frac{1}{2}\} \) in the \((v,s)\) direction, and \(-f_{\text{init}}\) puts a flow with value \( \frac{1}{2} \) in the \((v,s)\) direction, thus \( f_{\text{final}} - f_{\text{init}} \) puts a flow with nonnegative value in the \((v,s)\) direction. If this value was strictly positive, it would be part of a path in the flow decomposition that sends flow into \( s \), contradicting the maximality of \( f_{\text{final}} \).

\[\blacksquare\]