Fast, Provable Algorithms for Isotonic Regression in all $\ell_p$-norms

Rasmus Kyng
Department of Computer Science
Yale University
New Haven, CT 06520
rasmus.kyng@yale.edu

Anup Rao
Department of Mathematics
Yale University
New Haven, CT 06520
anup.rao@yale.edu

Sushant Sachdeva
Department of Computer Science
Yale University
New Haven, CT 06520
sachdeva@cs.yale.edu

July 3, 2015

Abstract

Given a directed acyclic graph $G$, and a set of values $y$ on the vertices, the Isotonic Regression of $y$ is a vector $x$ that respects the partial order described by $G$, and minimizes $\|x - y\|$, for a specified norm. This paper gives improved algorithms for computing the Isotonic Regression for all weighted $\ell_p$-norms with rigorous performance guarantees. Our algorithms are quite practical, and their variants can be implemented to run fast in practice.

1 Introduction

A directed acyclic graph (DAG) $G(V, E)$ defines a partial order on $V$ where $u$ precedes $v$ iff there is a directed path from $u$ to $v$. We say that a vector $x \in \mathbb{R}^V$ is isotonic (with respect to $G$) if it is a weakly order-preserving mapping of $V$ into $\mathbb{R}$. Let $I_G$ denote the set of all $x$ that are isotonic with respect to $G$. It is immediate that $I_G$ can be equivalently defined as follows:

$$I_G = \{x \in \mathbb{R}^V \mid x_u \leq x_v \text{ for all } (u, v) \in E\}. \quad (1)$$

Given a DAG $G$, and a norm $\| \cdot \|$ on $\mathbb{R}^V$, the Isotonic Regression of observations $y \in \mathbb{R}^V$, is given by $x \in I_G$ that minimizes $\|x - y\|$.

Such monotonic relationships are fairly common in data. They allow one to impose only weak assumptions on the data, e.g. the typical height of a young girl child is an increasing function of her age, and the heights of her parents, rather than a more constrained parametric model.

Isotonic Regression is an important shape-constrained nonparametric regression method that has been studied since the 1950’s [1, 2, 3]. It has found applications in diverse fields such as Operations Research [4, 5] and Signal Processing [6]. In Statistics, it has found multiple applications (e.g. [7, 8]), and the statistical properties of Isotonic Regression under the $\ell_2$-norm have been well studied, particularly over linear orderings (see [9] and references therein). More recently, Isotonic regression has found several applications in Learning [10, 11, 12]. It was used by Kalai and Sastry [10] to provably learn Generalized Linear Models and Single Index Models.

*This research was partially supported by AFOSR Award FA9550-12-1-0175 and NSF grant CCF-1111257
The most common norms of interest are weighted $\ell_p$-norms, defined as

$$\|z\|_{w,p} = \begin{cases} \left(\sum_{v \in V} w_v \cdot |z_v|^p\right)^{1/p}, & p \in [1, \infty), \\ \max_{v \in V} w_v \cdot |z_v|, & p = \infty, \end{cases}$$

where $w_v > 0$ is the weight of a vertex $v \in V$. In this paper, we focus on algorithms for Isotonic Regression under weighted $\ell_p$-norms. Such algorithms have been applied to large data-sets from Microarrays \[13\], and from the web \[14, 15\].

Given a DAG $G$, and observations $y \in \mathbb{R}^V$, our regression problem can be expressed as the following convex program:

$$\min \|x - y\|_{w,p} \text{ such that } x_u \leq x_v \text{ for all } (u, v) \in E. \quad (2)$$

### 1.1 Our Results

Let $|V| = n$, and $|E| = m$. We'll assume that $G$ is connected, and hence $n \geq m - 1$.

**$\ell_p$-norms, $p < \infty$.** We give a unified, optimization-based framework for algorithms that provably solve the Isotonic Regression problem for $p \in [1, \infty)$. The following is an informal statement of our main theorem (Theorem 2.1) in this regard.

**Theorem 1.1 (Informal).** Given a DAG $G$, observations $y$, and $\delta > 0$, in time $O(m^{1.5} \log^2 n \log n/\delta)$, we can compute an isotonic $x_{\text{alg}} \in \mathcal{I}_G$ such that

$$\|x_{\text{alg}} - y\|_{w,p} \leq \min_{x \in \mathcal{I}_G} \|x - y\|_{w,p} + \delta.$$

The previous best time bound was $O(n^2 m + n^3 \log n)$ for $p \in (1, \infty)$ and $O(mn + n^2 \log n)$ for $p = 1$ \[16\].

**$\ell_\infty$-norms.** For $\ell_\infty$-norms, unlike $\ell_p$-norms for $p \in (1, \infty)$, the Isotonic Regression problem need not have a unique solution. There are several specific solutions that have been studied in the literature (see \[17\] for a detailed discussion). In this paper, we show that some of them (MAX, MIN, and AVG to be precise) can be computed in time linear in the size of $G$.

**Theorem 1.2.** There is an algorithm that, given a DAG $G(V, E)$, a set of observations $y \in \mathbb{R}^V$, and weights $w$, runs in expected time $O(m)$, and computes an isotonic $x_{\text{inf}} \in \mathcal{I}_G$ such that

$$\|x_{\text{inf}} - y\|_{w,\infty} = \min_{x \in \mathcal{I}_G} \|x - y\|_{w,\infty}.$$

The previous best running time was $O(m \log n)$ \[17\].

**Strict Isotonic Regression.** We also give improved algorithms for Strict Isotonic Regression. Given observations $y$, and weights $w$, its Strict Isotonic Regression $x_{\text{strict}}$ is defined to be the limit of $x_p$, where $x_p$ is the Isotonic Regression for $y$ under the norm $\|\cdot\|_{w,p}$. It is immediate that $x_{\text{strict}}$ is an $\ell_\infty$ Isotonic Regression for $y$. In addition, it is unique, satisfies several desirable properties (see \[18\]).

**Theorem 1.3.** There is an algorithm that, given a DAG $G(V, E)$, a set of observation $y \in \mathbb{R}^V$, and weights $w$, runs in expected time $O(mn)$, and computes $x_{\text{strict}}$, the strict Isotonic Regression of $y$.

The previous best running time was $O(mn \log n)$ \[18\].

### 1.2 Detailed Comparison to Previous Results

**$\ell_p$-norms, $p < \infty$.** There has been a lot of work for fast algorithms for special graph families, mostly for $p = 1, 2$ (see \[19\] for references). For some cases where $G$ is very simple, e.g. a directed path (corresponding to linear orders), or a rooted, directed trees (corresponding to tree orders), several works give algorithms with running times of $O(n)$ or $O(n \log n)$ (see \[19\] for references).

Theorem 1.1 not only improves on the previously best known algorithms for general DAGs, but also on several algorithms for special graph families (see Table 1). One such setting is where $V$ is a point set in $d$-dimensions, and...
Table 1: Comparison to previous best results for \( \ell_p \)-norms, \( p \neq \infty \)

|          | \( \ell_1 \)        | \( \ell_2 \)        | This paper                          |
|----------|----------------------|----------------------|------------------------------------|
| 2-dim    | \( n \log^2 n \) [16]| \( n^2 \log n \) [16]| \( n^{1.5} \log^{1.5} n \)       |
| d-dim    | \( n^2 \log^d n \) [16]| \( n^3 \log^d n \) [16]| \( n^{1.5} \log^{1.5(d+1)} n \) |
| arbitrary| \( nm + n^2 \log n \) [13]| \( n^2 m + n^3 \log n \) [16]| \( m^{1.5} \log^3 n \)        |

For sake of brevity, we have ignored the \( O(\cdot) \) notation implicit in the bounds, and \( o(\log n) \) terms. Moreover, the bounds from previous results are exact, whereas our results are reported with an error parameter \( \delta = n^{-\Omega(1)} \).

\((u, v) \in E\) whenever \( u_i \leq v_i \) for all \( i \in [d] \). This setting has applications to data analysis, as in the example given earlier, and has been studied extensively (see [20] for references). For this case, it was proved by Stout (see Prop. 2, [20]) that these partial orders can be embedded in a DAG with \( O(n \log^{d-1} n) \) vertices and edges, and that this DAG can be computed in time linear in its size. The bounds then follow by combining this result with our theorem above.

For the important special case of \( \ell_2 \)-norms, we obtain improved running times in all the above cases. For \( \ell_1 \)-norms, we obtain improved running times for \( d \)-dim point sets for \( d \geq 3 \), and for DAGs with \( m = o(n^2 / \log^6 n) \).\footnote{The work [23] was not publicly available at the time of writing this article.}

\( \ell_\infty \)-\textbf{norms.} For weighted \( \ell_\infty \)-norms on arbitrary DAGs, the previous best result was \( O(m \log n + n \log^2 n) \) due to Kaufman and Tamir [21]. A manuscript by Stout [17] gives an improved bound of \( O(m \log n) \) for arbitrary DAGs. Theorem 1.2 improves on this.

In a survey on the best known running times for Isotonic Regression [19], it is claimed that an unpublished work by Stout [22] gives \( O(n) \)-time algorithms for linear order, trees, and \( d \)-grids, and an \( O(n \log^{d-1} n) \) algorithm for arbitrary point sets in \( d \)-dimensions.\footnote{The work [23] was not publicly available at the time of writing this article.} Theorem 1.2 implies the linear-time algorithms immediately. The result for \( d \)-dimensional point sets follows after embedding the point sets into DAGs of size \( O(n \log^{d-1} n) \), as for \( \ell_p \)-norms (using Proposition 2, [20]).

\textbf{Strict Isotonic Regression.} Strict Isotonic regression was introduced and studied in [18]. It also gave the only previous algorithm for computing it, that runs in time \( O(mn \log n) \). Theorem 1.3 improves on this result.

\subsection*{1.3 Overview of the Techniques and Contribution}

\( \ell_p \)-\textbf{norms,} \( p < \infty \). It is immediate that Isotonic Regression, as formulated in Equation (2), is a convex programming problem. For weighted \( \ell_p \)-norms with \( p < \infty \), applying generic convex-programming algorithms such as Interior Point methods to this formulation leads to algorithms that are quite slow.

We obtain faster algorithms for Isotonic Regression by replacing the computationally intensive component of Interior Point methods, solving systems of linear equations, with approximate solves. This approach has been used to design fast algorithms for generalized flow problems [23][24][25].

We present a complete proof of an Interior Point method for a large class of convex programs that only requires approximate solves. Daitch and Spielman [23] had proved such a result for linear programs. We extend this to \( \ell_p \)-objectives, and provide an improved analysis that only requires linear solvers with a constant factor relative error bound, whereas the method from Daitch and Spielman required polynomially small error bounds.

The linear systems in [24][25] are Symmetric Diagonally Dominant (SDD) matrices. The seminal work of Spielman and Teng [24] gives near-linear time approximate solvers for such systems, and later research has improved these solvers further [27][28]. Daitch and Spielman [24] extended these solvers to M-matrices (generalizations of SDD). The systems we need to solve are neither SDD, nor M-matrices. We develop fast solvers for this new class of matrices using fast SDD solvers.

\( \ell_\infty \)-\textbf{norms and Strict Isotonic Regression.} Algorithms for \( \ell_\infty \)-norms and Strict Isotonic Regression are based on techniques presented in a recent paper of Kyng \textit{et al.} [29]. We reduce \( \ell_\infty \)-norm Isotonic Regression to the following problem, referred to as Lipschitz learning on directed graphs in [29] (see Section 3 for details): We have a directed graph \( H \), with edge lengths given by \( \text{len} \). Given \( x \in \mathbb{R}^{V(H)} \), for every \((u, v) \in E(H)\), define \( \text{grad}_G^x[u,v] = \).
max \( \left\{ \frac{x(u) - x(v)}{\text{len}(u, v)}, 0 \right\} \). Now, given \( y \) that assigns real values to a subset of \( V(H) \), the goal is to determine \( x \in \mathbb{R}^{V(H)} \) that agrees with \( y \) and minimizes \( \max_{(u, v) \in E(H)} \text{grad}^+_G x(u, v) \).

The above problem is solved in \( O(m + n \log n) \) time for general directed graphs in \([29]\). We give a simple linear-time reduction to the above problem with the additional property that \( H \) is a DAG. For DAGs, their algorithm can be implemented to run in \( O(m + n) \) time.

It is proved in \([13]\) that computing the Strict Isotonic Regression is equivalent to computing the isotonic vector that minimizes the error under the lexicographic ordering (see Section \(4\)). Under the same reduction as in the \( \ell_\infty \)-case, we show that this is equivalent to minimizing \( \text{grad}^+_G \) under the lexicographic ordering. It is proved in \([29]\) that the lex-minimizer can be computed with basically \( n \) calls to \( \ell_\infty \)-minimization, immediately implying our result.

### 1.4 Implementation

An important advantage of our algorithms is that they can be implemented quite efficiently. Our algorithms are based on what is known as a short-step method (see Chapter 11, \([30]\)), that leads to an \( O(\sqrt{m}) \) bound on the number of iterations. Each iteration corresponds to one linear solve in the Hessian matrix. A variant, known as the long-step method (see \([30]\)) is believed to typically require much fewer iterations, about \( \log m \), even though the only provable bound known is \( O(m) \).

For the important special case of \( \ell_2 \)-Isotonic Regression, we have implemented our algorithm in Matlab, with long step barrier method, combined with our approximate solver for the linear systems involved. A number of heuristics recommended in \([30]\) that greatly improve the running time in practice have also been incorporated. Despite the changes, our implementation is theoretically correct and also outputs an upper bound on the error by giving a solution for the dual program.

In Table 2, we give running times and iteration counts for \( \ell_2 \) Isotonic Regression on DAGs where the underlying graphs are 2-d grid graph and random regular graph (of constant degree). The edges for 2-d grid graph are all oriented towards one of the corners and for random regular graphs, the edges are oriented after picking a random permutation. The coordinates of the vector \( y \) are chosen i.i.d. from the standard normal distribution. For each graph size, the experiment is repeated multiple times and the average iteration count and the run time in seconds is reported. We produced a dual certificate to ascertain that the output was correct up to 6 decimal places.

| #vertices | 2d Grid | Random Regular |
|-----------|--------|----------------|
|           | Iterations | Time (secs) | Iterations | Time (secs) |
| 5000      | 74      | 11            | 78         | 25          |
| 10000     | 78      | 14            | 81         | 13          |
| 20000     | 91      | 31            | 90         | 51          |
| 40000     | 96      | 60            | 91         | 79          |
| 80000     | 110     | 156           | 99         | 141         |

### 2 Algorithms for \( \ell_p \)-norms, \( p < \infty \)

Without loss of generality, we assume \( y \in [0, 1]^n \). Given \( p \in [1, \infty) \), let \( p \)-ISO denote the following \( \ell_p \)-norm Isotonic Regression problem, and \( \text{OPT}_{p, \text{ISO}} \) denote its optimum:

\[
\min_{x \in \mathcal{G}} \| x - y \|_{w, p}^p.
\]

Let \( w^p \) denote the entry-wise \( p \)-th power of \( w \). We assume the minimum entry of \( w^p \) is 1, and the maximum entry is \( w^p_{\text{max}} \leq \exp(n) \). To state the following theorem cleanly, we also assume the additive error parameter \( \delta \) is lower
The algorithm requires 11 iterations and returns a point bounded by formally in Supplementary Material Section A.1. The first concept is the algorithm \( A \)

Given a DAG \( G(V, E) \), a set of observations \( y \in [0, 1]^V \), weights \( w \), and an error parameter \( \delta > 0 \), the algorithm \textsc{IsotonicIPM} runs in time \( \tilde{O}\left(m^{1.5} \log^2 n \log \left( \frac{n \cdot \text{apx}}{\theta} \right) \right) \), and with probability at least \( 1 - \frac{1}{n} \), outputs a vector \( x_{\text{ALG}} \in I_G \) with

\[
\|x_{\text{ALG}} - y\|_{w, p}^p \leq \text{OPT}_{p, \text{ISO}} + \delta.
\]

The algorithm \textsc{IsotonicIPM} is obtained by an appropriate instantiation of a general Interior Point Method (IPM) algorithm \textsc{ApproxIPM}. It is described later in this section, and we also give a proof of the above theorem. \textsc{IsotonicIPM} is also summarized in Supplementary Material Section A.5, where we also prove the following theorem. \textsc{ApproxIPM} is obtained by an appropriate instantiation of a general Interior Point Method (IPM) where we also prove the following theorem.

To state the more general IPM result, we need to introduce two important concepts. These concepts are defined formally in Supplementary Material Section A.1. The first concept is self-concordant functions. We can associate to a convex set \( S \) a self-concordant barrier function \( f \). This is a special convex function whose domain is \( S \), and it approaches infinity at the boundary of \( S \). The set of self-concordant functions is denoted \( \mathcal{SC} \). We associate to each \( f \in \mathcal{SC} \) a complexity parameter \( \theta(f) \) which is a measure of well-behaved the function is. When \( \theta(f) \) is finite, we call \( f \) a barrier function. The set of self-concordant barrier functions is denoted \( \mathcal{SCB} \).

The second important concept is symmetry of a point w.r.t. a convex set. For a convex set \( S \) and a point \( z \in S \), we define a positive scalar quantify \( \text{sym}(z, S) \) called symmetry. It is always the case that \( 0 < \text{sym}(z, S) \leq 1 \), and for our algorithms to work, we need a starting point whose symmetry is not too small. We later show that such a starting point can be constructed for the \( p \)-ISO problem.

Our IPM is a primal path following IPM. The central element of \textsc{ApproxIPM} is the following: Given a vector \( c \), a domain \( D \) and a barrier function \( f \in \mathcal{SCB} \) for \( D \), where we are interested in the program \( \min_{x \in D} \langle c, x \rangle \), we consider a function \( f_{c, \gamma}(x) = f(x) + \gamma \langle c, x \rangle \). We attempt to minimize \( f_{c, \gamma} \) for changing values of \( \gamma \). The term \( f(x) \) grows to infinity as \( x \) approaches the boundary of \( D \), and with some care, we can use this to ensure we never move to a point outside the feasible domain \( D \). As we increase \( \gamma \), the objective term \( \langle c, x \rangle \) contributes more to \( f_{c, \gamma} \). Eventually, for large enough \( \gamma \), the objective value \( \langle c, x \rangle \) of our point \( x \) will be close to the optimum of the program.

To stay near the optimum \( x \) for each new value of \( \gamma \), we use a second-order method (Newton steps) to update \( x \) when \( \gamma \) is changed. This means we locally approximate our function by a quadratic function and minimize this quadratic. To do so, we need to solve an equation of the form \( Hz = g \), where \( g \) is the gradient of \( f \) at \( x \) and \( H \) is the Hessian of \( f \) at \( x \). Solving this equation to find \( z \) is the most computationally intensive aspect of the algorithm. Crucially we ensure that crude approximate solutions to the system of equations suffices, allowing the algorithm to use fast approximate solvers for this step. The algorithm \textsc{ApproxIPM} is described in detail in Supplementary Material Section A.5, where we also prove the following theorem.

\[ \text{Theorem 2.2.} \quad \text{Suppose we are given a convex domain } D \subseteq \mathbb{R}^n \text{ and vector } c \in \mathbb{R}^n, \text{ where we are interested in the following program} \]

\[ \min_{x \in D} \langle c, x \rangle. \]  

\[ \text{(4)} \]

Let \( \text{OPT} \) denote the minimum value of the program. Let \( f \in \mathcal{SCB} \) be a self-concordant barrier function for \( D \). Given a initial point \( x_0 \in D \), a value upper bound \( K \geq \sup \{ \langle c, x \rangle : x \in D \} \), a symmetry lower bound \( s \leq \text{sym}(x_0, D) \), and an error parameter \( 0 < \epsilon < 1 \), the algorithm \textsc{ApproxIPM} runs for

\[ T_{\text{apx}} = O \left( \sqrt{\theta(f)} \log \left( \frac{\theta(f)}{\epsilon \cdot s} \right) \right) \]

iterations and returns a point \( x_{\text{apx}} \), which satisfies

\[ \frac{\langle c, x_{\text{apx}} \rangle - \text{OPT}}{K - \text{OPT}} \leq \epsilon. \]

The algorithm requires \( O(T_{\text{apx}}) \) multiplications of vectors by a matrix \( M(x) \) which satisfies \( \frac{1}{10} \cdot H(x)^{-1} \leq M(x) \leq \frac{11}{10} \cdot H(x)^{-1} \), where \( H(x) \) is the Hessian of \( f \) at various points \( x \in D \) specified by the algorithm.
We now show how to solve the $p$-ISO program using the APPROXIPM algorithm. We can express program (3) as a program that requires us to minimize a linear objective over a convex domain. We consider a domain in $x \in \mathbb{R}^n$ and an additional vector $t \in \mathbb{R}^n$. We define a set of points

$$D_G = \{ (x, t) : \text{ for all } v \in V, |x(v) - y(v)|^p - t(v) \leq 0 \}.$$ 

$D_G$ is convex because each constraint corresponds to a sublevel set of a convex function, which is a convex set, and the intersection of convex sets is a convex set. $I_G$ is also convex as it is an intersection of half-spaces. Hence, $I_G \times \mathbb{R}^V$ is convex. Thus, we can express program (3) as a minimization of a linear function over a convex domain:

$$\min_{x, t} \langle w^p, t \rangle \\
\text{s.t.} \quad (x, t) \in D_G \cap (I_G \times \mathbb{R}^V).$$

(5)

Let $OPT_{\text{lin}}$ denote the optimal value of program (4). The value $OPT_{\text{lin}}$ is attained only when $t(v) = |x(v) - y(v)|^p$ for every vertex $v$, and when this holds, the program is exactly identical to program (3). Hence $OPT_{\text{lin}} = OPT_{p, \text{ISO}}$.

Program (5) is almost in the form required Theorem 2.2. But to apply the theorem, we need four more steps: 1. Ensure that the domain is bounded. 2. Find a feasible point and lower bound its symmetry w.r.t. the domain. 3. Find a suitable barrier function for the problem. 4. Efficiently compute an approximate inverse of the Hessian of our barrier function. The first three points have fairly standard solutions, while step 4 requires the development of a new linear solver for the Hessians we encounter.

To ensure boundedness, we add a constraint $\langle w^p, t \rangle \leq K$ to program (5).

Definition 2.3. We define the domain $D$ to be the set of points $D_G \cap (I_G \times \mathbb{R}^V)$. We define the domain $D_K$ to be the set of points in $D$ where the constraint $\langle w^p, t \rangle \leq K$ is satisfied.

Both $D$ and $D_K$ are convex. We now introduce a version the program with our restricted domain $D_K$.

$$\min_{x, t} \langle w^p, t \rangle \\
\text{s.t.} \quad (x, t) \in D_K.$$ (6)

Let $OPT_{\text{bnd}}$ denote the optimal value of program (6).

Our next lemma determines a choice of $K$ which suffices to ensure that programs (3) and (6) have the same optimum. The lemma is proven in Supplementary Material Section A.4.

Lemma 2.4. The constraint $\langle w^p, t \rangle \leq K$, where $K = 3n w_{\max}^p$, as used by ISOTONICIPM ensures that the domain $D_K$ of program (6) is non-empty and bounded, and that $OPT_{\text{bnd}} = OPT_{p, \text{ISO}}$.

The following result shows that we can compute a good starting point efficiently. The algorithm GOODSTART computes a starting point in linear time by running a topological sort on the vertices of the DAG $G$ and assigning values to $x$ according to the vertex order of the sort. Combined with an appropriate choice of $t$, this suffices to give a starting point with good symmetry. The algorithm GOODSTART is specified in more detail in Supplementary Material Section A.4 together with a proof of the following lemma.

Lemma 2.5. The algorithm GOODSTART runs in time $O(m)$ and returns an initial point $(x_0, t_0)$ that is feasible, and for $K = 3n w_{\max}^p$ satisfies $\text{sym}((x_0, t_0), D_K) \geq \frac{1}{18n^2 p w_{\max}}$.

Combining standard results on self-concordant barrier functions with a barrier for $p$-norms developed by Hertog et al. [31], we can show the following properties of a function $F_K$ whose exact definition is given in Supplementary Material Section A.2.

Corollary 2.6. The function $F_K$ is a self-concordant barrier for $D_K$ and it has complexity parameter $\theta(F_K) = O(m)$. Its gradient $g_{F_K}$ is computable in $O(m)$ time, and an implicit representation of the Hessian $H_{F_K}$ can be computed in $O(m)$ time as well.
Finally, and most significantly, we show that there exists a solver that can efficiently solve linear equations in the Hessian of $F_K$. The algorithm $\text{HESSIAN}\text{SOLVE}$ solves linear systems in Hessian matrices of the barrier function $F_K$. The Hessian is composed of a structured main component plus a rank one matrix.

We develop a solver for the main component by doing a change of variables to simplify its structure, and then factoring the matrix by a block-wise $LDL^\top$-decomposition. We can solve straightforwardly in the $L$ and $L^\top$, and we show that the $D$ factor consists of blocks that are either diagonal or SDD, so we can solve in this factor approximately using a nearly-linear time SDD solver.

A secondary part of the algorithm handles the rank one term of the Hessian using the Sherman-Morrison formula update. We give a novel error analysis for this step, which ensures that only constant factor approximate solves are required by the SDD solver.

The algorithm $\text{HESSIAN}\text{SOLVE}$ is given in full in Supplementary Material Section A.3 along with a proof of the following result.

**Theorem 2.7.** For any instance of program (6) given by some $(G, y)$, at any point $z \in \mathcal{D}_K$, for any vector $a$, $\text{HESSIAN}\text{SOLVE}((G, y), z, \mu, a)$ returns a vector $b = Ma$ for a symmetric linear operator $M$ satisfying $\|H_{F_K}(z)^{-1}\| \leq M \leq 11/10 \cdot H_{F_K}(z)^{-1}$. The algorithm fails with probability $< \mu$. $\text{HESSIAN}\text{SOLVE}$ runs in time $O(m \log n \log (1/\mu))$.

These are the ingredients we need to prove our main result on solving $p$-ISO. The algorithm $\text{ISOTONIC}\text{IPM}$ is simply $\text{APPROX}\text{IPM}$ instantiated to solve program (6), with $K = 3n w_{\text{max}}^p$, using barrier function $F_K$, and using $\text{HESSIAN}\text{SOLVE}$ to approximately solve in the Hessian of $F_K$. $\text{ISOTONIC}\text{IPM}$ uses the starting point computed by $\text{GOOD}\text{START}$. By choosing error parameter $\epsilon = \delta/K$, we ensure that $\text{ISOTONIC}\text{IPM}$ achieves an additive error guarantee of $\delta$.

**Proof of Theorem 2.1** By Corollary 2.6, the barrier function $F_K$ used by $\text{ISOTONIC}\text{IPM}$ has complexity parameter $\theta(F_K) \leq O(m)$. By Lemma 2.5 the starting point $(x_0, t_0)$ computed by $\text{GOOD}\text{START}$ and used by $\text{ISOTONIC}\text{IPM}$ is feasible and has symmetry $\text{SYM}(x_0, \mathcal{D}_K) \geq \frac{1}{18n^2w_{\text{max}}^p}$. $\text{ISOTONIC}\text{IPM}$ uses a number of calls to $\text{HESSIAN}\text{SOLVE}$ bounded by

$$O(T) \leq O\left(\sqrt{\theta(F_K)} \log \left(\frac{\theta(F_K)}{\epsilon \cdot 8}\right)\right) \leq O\left(\sqrt{m} \log \left(\frac{np^5 w_{\text{max}}^p}{\delta}\right)\right).$$

Each call to $\text{HESSIAN}\text{SOLVE}$ fails with probability $< n^3$, and so by a union bound over all calls, the probability that we get one or more failed calls to $\text{HESSIAN}\text{SOLVE}$ is upper bounded by $O(n \log (np^8 w_{\text{max}}^p) / n^3) = O(1/n)$.

The point $(x_{\text{apx}}, t_{\text{apx}})$ output by $\text{ISOTONIC}\text{IPM}$ satisfies $\langle w_p, t_{\text{apx}} \rangle - \text{OPT} \leq \epsilon$, where $\text{OPT}$ is the optimum value of program (6), and $K = 3n w_{\text{max}}^p$ is the value used by $\text{ISOTONIC}\text{IPM}$ for the constraint $\langle w_p, t \rangle \leq K$, which is an upper bound on the supremum of objective values of feasible points of program (6).

By Lemma 2.4 $\text{OPT} = \text{OPT}_{p,\text{ISO}}$. Hence, $\|y - x_{\text{apx}}\|^p \leq \langle w_p, t_{\text{apx}} \rangle \leq \text{OPT} + \epsilon K = \text{OPT}_{p,\text{ISO}} + \delta$. The algorithm uses $O\left(\sqrt{m} \log \left(\frac{np^5}{\delta^2}\right)\right)$ calls to $\text{HESSIAN}\text{SOLVE}$ that each take time $\tilde{O}(m \log^2 n)$, as $\mu = n^3$. Thus the total running time is $\tilde{O}(m^{1.5} \log^2 n \log (\frac{np^5}{\delta^2}))$.

## 3 Algorithms for $\ell_\infty$ and Strict Isotonic Regression

We now reduce $\ell_\infty$ Isotonic Regression and Strict Isotonic Regression to the Lipschitz Learning problem, as defined in [29]. Let $G = (V, E, \text{len})$ be any DAG with non-negative edge lengths $\text{len} : E \to \mathbb{R}_{>0}$, and $y : V \to \mathbb{R} \cup \{\ast\}$ a partial labeling. We think of a partial labeling as a function that assigns real values to a subset of the vertex set $V$. We call such a pair $(G, y)$ a **partially-labeled DAG**. For a complete labeling $x : V \to \mathbb{R}$, define the gradient on an edge $(u, v) \in E$ due to $x$ to be $\text{grad}^\top_G [x](u, v) = \max \left\{ \frac{x(u) - x(v)}{\text{len}(u, v)}, 0 \right\}$. If $\text{len}(u, v) = 0$, then $\text{grad}^\top_G [x](u, v) = 0$ unless $x(u) > x(v)$, in which case it is defined as $+\infty$. Given a partially-labelled DAG $(G, y)$, we say that a complete assignment $x$ is a **inf-minimizer** if it extends $y$, and for all other complete assignments $x'$ that extends $y$ we have

$$\max_{(u, v) \in E} \text{grad}^\top_G [x](u, v) \leq \max_{(u, v) \in E} \text{grad}^\top_G [x'](u, v).$$

7
Note that when $G$ is a DAG with $\ell = 0$, then $\max_{(u,v) \in E} \nabla^+_{G,x}(u,v) < \infty$ if and only if $x$ is isotonic on $G$.

Suppose we are interested in Isotonic Regression on a DAG $G(V, E)$ under $\|\cdot\|_{w,\infty}$. To reduce this problem to that of finding an inf-minimizer, we add some auxiliary nodes and edges to $G$. Let $V_L, V_R$ be two copies of $V$. That is, for every vertex $u \in V$, add a vertex $u_L$ to $V_L$ and a vertex $u_R$ to $V_R$. Let $E_L = \{(u_L, u) \mid u \in V\}$ and $E_R = \{(u_R, u) \mid u \in V\}$. All other edge lengths are set to 0. Finally, let $G' = (V \cup V_L \cup V_R, E \cup E_L \cup E_R, \nabla')$. The partial assignment $y'$ takes real values only on the vertices in $V_L \cup V_R$. For all $u \in V$, $y'(u_L) := y(u)$ and $y'(u_R) := y(u)$. We then let $G'$ be the lex-minimizer for the partially-labeled DAG $G$. When

Algorithm COMPINFMIN from [29] is proved to compute the inf-minimizer, and is claimed to work for directed graphs (Section 5, [29]). We exploit the fact that Dijkstra’s algorithm in COMPINFMIN can be implemented in $O(m)$ time on DAGs using a topological sorting of the vertices, giving a linear time algorithm for computing the inf-minimizer, and hence Theorem 1.3. For more details, see Section B.

Lemma 3.1. Given a DAG $G(V, E)$, a set of observations $y \in \mathbb{R}^V$, and weights $w$, construct $G'$ and $y'$ as above. Let $x$ be an inf-minimizer for the partially-labeled DAG $(G', y')$. Then, $x |_{V}$ is the Isotonic Regression of $y$ with respect to $G$ under the norm $\|\cdot\|_{w,\infty}$.

Proof. We note that since the vertices corresponding to $V$ in $(G', y')$ are connected to each other by zero length edges, $\max_{(u,v) \in E} \nabla^+_{G,x}(u,v) < \infty$ if $x$ is isotonic on those edges. Since $G$ is a DAG, we know that there are isotonic labelings on $G$. When $x$ is isotonic on vertices corresponding to $V$, gradient is zero on all the edges going in between vertices in $V$. Also, note that every vertex $x$ corresponding to $V$ in $G'$ is attached to two auxiliary nodes $x_L \in V_L, x_R \in V_R$. We also have $y'(x_L) = y'(x_R) = y(x)$. Thus, for any $x$ that extends $y$ and is isotonic on $G'$, the only non-zero entries in $\nabla^+_{G,x}$ correspond to edges in $E_R$ and $E_L$, and we have

$$\max_{(u,v) \in E} \nabla^+_{G,x}(u,v) = \max_{u \in V} w_u \cdot |y(u) - x(u)| = \|x - y\|_{w,\infty},$$

proving our claim.

Algorithm COMPINFMIN from [29] is proved to compute the inf-minimizer, and is claimed to work for directed graphs (Section 5, [29]). We exploit the fact that Dijkstra’s algorithm in COMPINFMIN can be implemented in $O(m)$ time on DAGs using a topological sorting of the vertices, giving a linear time algorithm for computing the inf-minimizer, and hence Theorem 1.3. We remark that the solution the $\ell_\infty$-Isotonic Regression that we obtain has been referred to as AVG $\ell_\infty$ Isotonic Regression in the literature [17]. It is easy to modify the algorithm to compute the MAX, MIN $\ell_\infty$ Isotonic Regressions. Details are given in Section B.

For Strict Isotonic Regression, we define the lexicographic ordering. Given $r \in \mathbb{R}^m$, let $\pi_r$ denote a permutation that sorts $r$ in non-increasing order by absolute value, i.e., $\forall i \in [m - 1], |r(\pi_r(i))| \geq |r(\pi_r(i + 1))|$. Given two vectors $r, s \in \mathbb{R}^m$, we write $r \preceq_{\text{lex}} s$ to indicate that $r$ is smaller than $s$ in the lexicographic ordering on sorted absolute values, i.e.

$$\exists j \in [m], |r(\pi_r(j))| < |s(\pi_s(j))| \text{ and } \forall i \in [j - 1], |r(\pi_r(i))| = |s(\pi_s(i))| \text{ or } \forall i \in [m], |r(\pi_r(i))| = |s(\pi_s(i))|. $$

Note that it is possible that $r \preceq_{\text{lex}} s$ and $s \preceq_{\text{lex}} r$ while $r \neq s$. It is a total relation: for every $r$ and $s$ at least one of $r \preceq_{\text{lex}} s$ or $s \preceq_{\text{lex}} r$ is true.

Given a partially-labelled DAG $(G, y)$, we say that a complete assignment $x$ is a lex-minimizer if it extends $y$ and for all other complete assignments $x'$ that extend $y$ we have $\nabla^+_{G,x} \preceq_{\text{lex}} \nabla^+_{G,x'}$.

Stout [18] proves that computing the Strict Isotonic Regression is equivalent to finding an Isotonic $x$ that minimizes $z_u = w_u \cdot (x_u - y_u)$ in the lexicographic ordering. With the same reduction as above, it is immediate that this is equivalent to minimizing $\nabla^+_{G,x}$ in the lex-ordering.

Lemma 3.2. Given a DAG $G(V, E)$, a set of observations $y \in \mathbb{R}^V$, and weights $w$, construct $G'$ and $y'$ as above. Let $x$ be the lex-minimizer for the partially-labeled DAG $(G', y')$. Then, $x |_{V}$ is the Strict Isotonic Regression of $y$ with respect to $G$ with weights $w$.

It is proved in [29] that the lex-minimizer can be computed with $n$ calls to inf-minimization, which given our observation above, implies Theorem 1.3. For more details, see Section B.
Acknowledgements

We would like to thank Sabyasachi Chatterjee for introducing the problem to us and Daniel Spielman for his advice, comments and discussions.

References

[1] M. Ayer, H. D. Brunk, G. M. Ewing, W. T. Reid, and E. Silverman. An empirical distribution function for sampling with incomplete information. *The Annals of Mathematical Statistics*, 26(4):pp. 641–647, 1955.

[2] Richard E Barlow, David J Bartholomew, JM Bremner, and H Daniel Brunk. *Statistical inference under order restrictions: the theory and application of isotonic regression*. Wiley New York, 1972.

[3] Friedrich Gebhardt. An algorithm for monotone regression with one or more independent variables. *Biometrika*, 57(2):263–271, 1970.

[4] W. L. Maxwell and J. A. Muckstadt. Establishing consistent and realistic reorder intervals in production-distribution systems. *Operations Research*, 33(6):pp. 1316–1341, 1985.

[5] R. Roundy. A 98%-effective lot-sizing rule for a multi-product, multi-stage production / inventory system. *Mathematics of Operations Research*, 11(4):pp. 699–727, 1986.

[6] S.T. Acton and A.C. Bovik. Nonlinear image estimation using piecewise and local image models. *Image Processing, IEEE Transactions on*, 7(7):979–991, Jul 1998.

[7] Chu-In Charles Lee. The min-max algorithm and isotonic regression. *The Annals of Statistics*, 11(2):pp. 467–477, 1983.

[8] Richard L. Dykstra and Tim Robertson. An algorithm for isotonic regression for two or more independent variables. *The Annals of Statistics*, 10(3):pp. 708–716, 1982.

[9] S. Chatterjee, A. Guntuboyina, and B. Sen. On Risk Bounds in Isotonic and Other Shape Restricted Regression Problems. *The Annals of Statistics*, to appear.

[10] A. T. Kalai and R. Sastry. The isotron algorithm: High-dimensional isotonic regression. In *COLT*, 2009.

[11] T. Moon, A. Smola, Y. Chang, and Z. Zheng. Intervalrank: Isotonic regression with listwise and pairwise constraints. In *Proceedings of WSDM*, WSDM ’10, pages 151–160. ACM, 2010.

[12] S. M Kakade, V. Kanade, O. Shamir, and A. Kalai. Efficient learning of generalized linear and single index models with isotonic regression. In *NIPS*, 2011.

[13] S. Angelov, B. Harb, S. Kannan, and L. Wang. Weighted isotonic regression under the $l_1$ norm. In *Proceedings of SODA*. SIAM, 2006.

[14] K. Punera and J. Ghosh. Enhanced hierarchical classification via isotonic smoothing. In *Proceedings of WWW’08*, pages 151–160. ACM, 2008.

[15] Z. Zheng, H. Zha, and G. Sun. Query-level learning to rank using isotonic regression. In *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*, pages 1108–1115, Sept 2008.

[16] Q. F. Stout. Isotonic regression via partitioning. *Algorithmica*, 66(1):93–112, 2013.

[17] Q. F. Stout. Weighted $l_\infty$ isotonic regression. *Manuscript*, 2011.

[18] Q. F. Stout. Strict $l_\infty$ Isotonic Regression. *Journal of Optimization Theory and Applications*, 152(1):121–135, 2012.

[19] Q. F Stout. Fastest isotonic regression algorithms. [http://web.eecs.umich.edu/~qstout/IsoRegAlg_140812.pdf](http://web.eecs.umich.edu/~qstout/IsoRegAlg_140812.pdf)

[20] Q. F. Stout. Isotonic regression for multiple independent variables. *Algorithmica*, 71(2):450–470, 2015.

[21] Y. Kaufman and A. Tamir. Locating service centers with precedence constraints. *Discrete Applied Mathematics*, 47(3):251 – 261, 1993.

[22] Q. F. Stout. $l_\infty$ isotonic regression for linear, multidimensional, and tree orders. In *preparation*.

[23] Samuel I. Daitch and Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. *STOC ’08*, pages 451–460. ACM, 2008.

[24] A. Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *FOCS*, 2013.
[25] Y. T. Lee and A. Sidford. Path finding methods for linear programming: Solving linear programs in $\tilde{O}(\text{vrank})$ iterations and faster algorithms for maximum flow. In FOCS, 2014.

[26] D. A. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. STOC ’04, pages 81–90. ACM, 2004.

[27] Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for SDD linear systems. In Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS ’11, pages 590–598, Washington, DC, USA, 2011. IEEE Computer Society.

[28] Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup B. Rao, and Shen Chen Xu. Solving sdd linear systems in nearly $\log \log n$ time. In Proceedings of the 46th Annual ACM Symposium on Theory of Computing, STOC ’14, pages 343–352, New York, NY, USA, 2014. ACM.

[29] R. Kyng, A. Rao, S. Sachdeva, and D. A. Spielman. Algorithms for lipschitz learning on graphs. CoRR, abs/1505.00290. To appear at COLT, 2015.

[30] Stephen Boyd and Lieven Vandenberghe. Convex Optimization. Cambridge University Press, New York, NY, USA, 2004.

[31] D. den Hertog, F. Jarre, C. Roos, and T. Terlaky. A sufficient condition for self-concordance, with application to some classes of structured convex programming problems. Math. Program., 69(1):75–88, July 1995.

[32] James Renegar. A mathematical view of interior-point methods in convex optimization. SIAM, 2001.

[33] A. Nemirovski. Lecture notes: Interior point polynomial time methods in convex programming. 2004.

[34] E. J. McShane. Extension of range of functions. Bull. Amer. Math. Soc., 40(12):837–842, 12 1934.

[35] Hassler Whitney. Analytic extensions of differentiable functions defined in closed sets. Transactions of the American Mathematical Society, 36(1):pp. 63–89, 1934.
A  IPM Definitions and Proofs

A.1 Definitions

Given a positive definite $n \times n$ matrix $A$, we define the norm $\|\cdot\|_A$ by
\[
\|x\|_A = \sqrt{x^T A x}.
\]

Given a twice differentiable function $f$ with domain $D_f$, which has positive definite Hessian $H(x)$ at some $x \in D_f$, we define
\[
\|y\|_x = \|y\|_{H(x)},
\]
and when $M$ is a matrix, let $\|M\|_x$ denote the corresponding induced matrix norm.

We let $B_x(y, r)$ denote the open ball centered at $y$ of radius $r$ in the $\|\cdot\|_x$ norm.

Again, suppose $f$ is a twice differentiable convex function with Hessian $H$, defined on a domain $D_f$. If for all $x \in D_f$ we have $B_x(x, 1) \subseteq D_f$, and for all $y \in B_x(x, 1)$ and all $v \neq 0$ we have
\[
1 - \|y - x\|_x \leq \frac{\|v\|_y}{\|v\|_x} \leq \frac{1}{1 - \|y - x\|_x},
\]
then we say the function is self-concordant. We denote the set of self-concordant functions by $SC$. A key theorem about self-concordant functions is the following (Theorem 2.2.1 of Renegar [32]).

**Theorem A.1**. Suppose $f$ is a twice differentiable function with Hessian $H$, defined on a domain $D_f$, and for all $x \in D_f$ we have $B_x(x, 1) \subseteq D_f$.

Then $f \in SC$ iff
\[
\|H(x)^{-1} H(y)\|_x, \|H(x)^{-1} H(y)\|_x \leq \frac{1}{(1 - \|y - x\|_x)^2}.
\]

Also $f \in SC$ iff
\[
\|I - H(x)^{-1} H(y)\|_x, \|I - H(x)^{-1} H(y)\|_x \leq \frac{1}{(1 - \|y - x\|_x)^2} - 1.
\]

If $f \in SC$ also satisfies $\sup_{x \in D_f} \|g_x(x)\|^2 < \infty$, we say that $f$ is a self-concordant barrier function. Given any $\theta(f) \geq \sup_{x \in D_f} \|g_x(x)\|^2$, we say $\theta(f)$ is a complexity parameter for $f$. We denote the set of self-concordant barrier functions by $SCB$.

We need the following notion of symmetry. We state a definition that is equivalent to the definition used by Renegar (Section 2.3.4 of [32]).

**Definition A.2**. Given a convex set $S$ and a point $x \in S$, the symmetry of $x$ w.r.t. $S$ is defined as
\[
sym(x, S) = \inf_{z \in \partial S} \inf \left\{ t > 0 : x + \frac{(x - z)}{t} \in S \right\}.
\]

A.2 A Barrier Function for $D_K$

Hertog et al. [31] proved the existence of self-concordant barrier functions for a class of domains including ones capable of expressing program (3). The exact statement we wish to employ can be found in lecture notes by Nemirovski [33].
Theorem A.3. For every pair of variables \((x, t) \in \mathbb{R}^2\), and for every constant \(p \geq 1\), a self-concordant barrier function \(f \in \text{SCB}\) exists for the domain \(\{(x, t) \in \mathbb{R}^2 : |x|^p \leq t\}\).

This barrier function is given by
\[
f(t, x) = -\log\left(\frac{t^{2/p}}{x^2} - x^2\right) - 2\log t,
\]
and has complexity parameter \(\theta(f) \leq 4\).

We are now ready to introduce a number of barrier functions:
\[
F(x, t) = \left(\sum_{v \in V} -\log \left(t(v)^{2/p} - (x(v) - y(v))^2\right) - 2\log t(v)\right) + \left(\sum_{(a, b) \in E} -\log(x(b) - x(a))\right).
\]
\[
f_K(x, t) = -\log(K - \langle w^p, t \rangle).
\]
\[
F_K(x, t) = F(x, t) + f_K(x, t).
\]

Proof of Corollary 2.6. To prove the corollary, we need the standard fact that \(-\log x\) is a self-concordant barrier for the domain \(x \geq 0\) with complexity parameter 1, as shown in Renegar’s section 2.3.1 [32]. We also need standard results on composition of barrier functions (adding barriers and composition with an affine function), as given by Renegar’s Theorems 2.2.6, 2.2.7, 2.3.1, and 2.3.2 [32]. Given these and Theorem A.3, the corollary follows immediately. □

A.3 Fast Solver for Approximate Hessian Inverse

Algorithm 1: Algorithm HESSIAN\texttt{SOVL}\texttt{E} \((G, y), (x, t), \mu, a)\): Given a p-ISO instance \((G, y)\), a feasible point \((x, t)\) of program, a vector \(a\), outputs vector \(b\).

1. \(u \leftarrow 1_{(K - \langle 1, 0 \rangle)}\).
2. \(\tau \leftarrow 1/50\).
3. \(M \leftarrow \text{BLOCK\texttt{SOVL}\texttt{E} \((G, y), (x, t), \mu, \tau\)}\).
4. return RANK\texttt{ONE}\texttt{MO}\texttt{RE} \((M, u, a)\).

Algorithm 2: Algorithm BLOCK\texttt{SOVL}\texttt{E} \((G, y), (x, t), \mu, \tau\)\):

1. Let \(r \leftarrow B^T(x \oplus y)\).
2. For each \(v \in V\), identify \(t(\hat{v}, v) = t(v)\).
3. Compute \(R, T\) and \(C\) as given by equations (9), (8), and (10).
4. \(S \leftarrow Q^T B (R - CT^{-1}C^T) B^T Q\).
5. \(M_S \leftarrow \text{SDDS\texttt{OLVE}(S, \mu, \tau)}\).
6. \(Z \leftarrow \begin{bmatrix} I & 0 \\ -Q^T CBT^{-1} & I \end{bmatrix}\).
7. Return a procedure that given vector \(a\) returns vector
\[
b \leftarrow Z^T \begin{bmatrix} T^{-1} & 0 \\ 0 & M_S \end{bmatrix} Z a.
\]

Algorithm 3: Algorithm RANK\texttt{ONE}\texttt{MO}\texttt{RE}(M, u, a)\): Given a linear operator \(M\), a vector \(u\), and a vector \(a\), outputs vector \(b\).

1. \(w = Mu\).
We associate with \( \bigcup V \)

Given vectors and while vertex set \( t \)

the cross-terms. In fact, the only thing we will need about the explicit forms of these matrices is that they are efficiently computable. For completeness, we state them:

Now, we define a vector \( \hat{v} \in \mathbb{R}^{\hat{E}} \), and columns indexed by the set \( E \cup \hat{E} \). It is given by

We introduce an extended graph \( \hat{G} = (V \cup \hat{V}, E \cup \hat{E}) \), which includes our original vertex set \( V \), and a second vertex set

We define an additional set of edges

Given vectors \( t \in \mathbb{R}^{\hat{E}} \) and \( r \in \mathbb{R}^{E \cup \hat{E}} \), we define a function

We associate with \( \hat{G} \) a matrix \( B \) known as the signed edge-vertex incidence matrix. \( B \) has rows indexed by the set \( V \cup \hat{V} \), and columns indexed by the set \( E \cup \hat{E} \). It is given by

Now, we define a vector \( x \oplus y \in \mathbb{R}^{V \cup \hat{V}} \) by

Note that \( |\hat{E}| = |V| \). Abusing notation, we identify the vector \( t \in \mathbb{R}^{\hat{E}} \) with the vector \( t \in \mathbb{R}^{V} \) by equating \( t(\hat{v}, v) = t(v) \). We then get

We compute the Hessian \( H_h \) of \( h(r, t) \) in variables \( r \) and \( t \). The Hessian can be expressed as a block matrix

where \( T \) contains derivatives in two coordinates of \( t \), while \( R \) contains derivatives in two coordinates in \( r \), and \( C \) has the cross-terms. \( T \) and \( R \) are square diagonal matrices, and \( C \) is not generally square, but has non-zero entries on the principal diagonal. In fact, the only thing we will need about the explicit forms of these matrices is that they are efficiently computable. For completeness, we state them:

and

while

\[ C(e, e) = -\frac{4}{p} \frac{t(e)^{-1+2/p} r(e)}{t(e)^{2/p} - r(e)^2} \text{ where } e \in \hat{E}. \]
Finally, let $Q$ denote the projection matrix which maps $x$ to $(x \oplus 0)$. It is a matrix with non-zeroes only on the principal diagonal:

$$ Q(v, v) = \begin{cases} 1 & \text{for } v \in V \\ 0 & \text{otherwise}. \end{cases} $$

To prove Theorem 2.7 we will need three results: The first is a theorem on fast SDD solvers proven by Koutis et al. [27].

**Theorem A.4.** Given an $n \times n$ SDD matrix $X$ with $m$ non-zero entries, an error probability $\mu$, and an error parameter $\tau$, with probability $\geq 1 - \mu$ the procedure $\operatorname{SDDSolve}(X, \mu, \tau)$ returns an (implicitly represented) symmetric linear operator $M$ satisfying

$$(1 - \tau)X^{-1} \preceq M \preceq (1 + \tau)X^{-1}.$$ 

$\operatorname{SDDSolve}(X, \mu, \tau)$ runs in time $\tilde{O}(m \log n \log(1/\mu) \log(1/\tau))$, and $M$ can be applied to a vector in time $\tilde{O}(m \log n \log(1/\mu) \log(1/\tau))$ as well.

**Lemma A.5.** Suppose $X$ is a positive definite matrix, and $\tau \in [0, 1/5)$ is an error parameter, and we are given a symmetric linear operator $M$ satisfying

$$(1 - \tau)X^{-1} \preceq M \preceq (1 + \tau)X^{-1},$$

and suppose we are given a vector $u \in \mathbb{R}^n$. Then $\operatorname{RankOneMore}(M, u, a)$ acts as a linear operator on $a$ and returns a vector $b = Za$ for a symmetric matrix $Z$ satisfying

$$(1 - 5\tau)(X + uu^T)^{-1} \preceq Z \preceq (1 + 5\tau)(X + uu^T)^{-1}.$$

If $M$ can be applied in time $T_M$, then $\operatorname{RankOneMore}$ runs in time $O(T_M + n)$.

**Lemma A.6.** For any instance of program (5) given by some $(G, y)$, at any point $z \in D_K$, given an error probability $\mu$, and an error parameter $\tau$, with probability $\geq 1 - \mu$ the procedure $\operatorname{BlockSolve}(X, \mu, \tau)$ returns an (implicitly represented) symmetric linear operator $M$ satisfying

$$(1 - \tau)H_F(z)^{-1} \preceq M \preceq (1 + \tau)H_F(z)^{-1}.$$ 

$\operatorname{BlockSolve}(X, \mu, \tau)$ runs in time $\tilde{O}(m \log n \log(1/\mu) \log(1/\tau))$, and $M$ can be applied to a vector in time $\tilde{O}(m \log n \log(1/\mu) \log(1/\tau))$ as well.

We prove Lemmas A.5 and A.6 at the end of this section.

**Proof of Theorem 2.7:** By Lemma A.6 $\operatorname{BlockSolve}((G, y), (x, t), \mu, 1/50)$ returns an implicitly represented linear operator $M$ satisfying

$$\left(1 - \frac{1}{50}\right)H_F((x, t))^{-1} \preceq M \preceq \left(1 + \frac{1}{50}\right)H_F((x, t))^{-1}.$$ 

This procedure satisfies the requirements of a solver $\operatorname{Solve}_{H_F}$ in Lemma A.5 With $u = \frac{1}{(x \cdot 1, t)} 1$, where $H_F(x, t) + uu^T = H_{F_K}(x, t)$, we get that $\operatorname{RankOneMore}(M, u, a)$ returns a vector $Za$, for a symmetric matrix $Z$ satisfying

$$\frac{9}{10}H_{F_K}(x, t)^{-1} \preceq Z \preceq \frac{11}{10}H_{F_K}(x, t)^{-1}.$$ 

The total running time is $\tilde{O}(m \log n \log(1/\mu))$, as the running time of $\operatorname{BlockSolve}$ dominates. The algorithms fails only if $\operatorname{BlockSolve}$ fails, which happens with probability $< \mu$. $\square$
Proof of Lemma A.5: Note \( T \) is a diagonal matrix, so that its inverse can be computed in linear time. Using standard facts about the Hessian under function composition, we can express the Hessian of \( F \) as

\[
H_F = \begin{bmatrix}
1 & 0 \\
0 & QT B
\end{bmatrix} \begin{bmatrix}
T & C^T \\
C & R
\end{bmatrix} \begin{bmatrix}
1 & 0 \\
0 & B^T Q
\end{bmatrix} = \begin{bmatrix}
T & QT B^T Q \\
QT BC & QT BRB^T Q
\end{bmatrix}.
\]

A blockwise LDU decomposition of \( H_F \) gives

\[
H_F = \begin{bmatrix}
I & 0 \\
QT B C T^{-1} & I
\end{bmatrix} \begin{bmatrix}
T & 0 \\
0 & S
\end{bmatrix} \begin{bmatrix}
I & T^{-1} C^T B^T Q \\
0 & I
\end{bmatrix}.
\]

Where the matrix

\[
S = QT B R B^T Q - QT B C T^{-1} C^T B^T Q = QT B (R - CT^{-1} C^T) B^T Q
\]

is the Schur-complement of \( T \) in \( H_F \). Now, \( R - CT^{-1} C^T \) is the Schur-complement of \( T \) in \( H \). A standard result about Schur complements states that \( H \) is positive definite if and only if both \( T \) and \( R - CT^{-1} C^T \) are positive definite. We know that \( H \) is positive definite, and consequently \( R - CT^{-1} C^T \) is too. But \( R - CT^{-1} C^T \) is a diagonal matrix, and so every entry must be strictly positive. This implies that \( B(R - CT^{-1} C^T) B^T \) is a Laplacian matrix. The matrix has \( O(m) \) non-zero entries. Since \( S = QT B (R - CT^{-1} C^T) B^T Q \) is a principal minor of a Laplacian matrix, it is positive definite and SDD. Because \( S \) is PD and SDD, by Theorem [A.4] using SDDSOLVE we can compute an (implicitly represented) approximate inverse matrix \( M_S \) that satisfies

\[(1 - \tau) S^{-1} \preceq M_S \preceq (1 + \tau) S^{-1}. \tag{11}\]

in time \( \tilde{O}(m \log n \log \frac{1}{\mu} \log \frac{1}{\tau}) \). This call may fail with a probability \( < \mu \). The matrix \( M_S \) can be applied in time \( \tilde{O}(m \log n \log \frac{1}{\mu} \log \frac{1}{\tau}) \).

A block-wise inversion of the Hessian gives

\[
H_F^{-1} = \begin{bmatrix}
I & -T^{-1} C^T B^T Q \\
0 & I
\end{bmatrix} \begin{bmatrix}
T^{-1} & 0 \\
0 & S^{-1}
\end{bmatrix} \begin{bmatrix}
I & 0 \\
-Q^T C B T^{-1} & I
\end{bmatrix}.
\]

By equations (11) and (12), and the fact that for all matrices \( W, X \preceq Y \) implies \( W X W^T \preceq W Y W^T \), it follows that

\[
(1 - \tau) H_F^{-1} \preceq M \preceq (1 + \tau) H_F^{-1}.
\]

We observe that the output of BLOCKSOLVE \( ((G, y), (x, t), \mu, \tau) \) is a procedure which applies \( M \). We require a constant number of linear time matrix operations (inversion of a diagonal matrix, multiplication of a vector with matrix), and one call to SDDSOLVE, which runs in time \( \tilde{O}(m \log n \log \frac{1}{\mu} \log \frac{1}{\tau}) \). This call dominates the running time of BLOCKSOLVE. The call to SDDSOLVE may fail with a probability \( < \mu \), and in that case BLOCKSOLVE also fails. \( \square \)

Proof of Lemma A.6: From our assumptions about \textsc{solve} and the computation in \textsc{rankonemore}, it follows that \( b = Za \).

for some

\[
Z = M - \frac{M uu^T M}{1 + a^T Mu T},
\]

\( z \) where \( \tau = \frac{\delta}{5} < 1/5 \) and

\[
(1 - \tau) X^{-1} \preceq M \preceq (1 + \tau) X^{-1}.
\]
Thus, \textsc{RankOneMore} acts as a linear operator on $a$, and it is symmetric. Suppose $Y = X + uu^T$, then by the Sherman-Morrison formula,

$$Y^{-1} = X^{-1} - \frac{X^{-1}uu^TX^{-1}}{1 + u^TX^{-1}u}.$$ 

We can restate the spectral inequalities for $M$ as $M = X^{-1} + E$, for some symmetric matrix $E$ with 

$$-\tau X^{-1} \preceq E \preceq \tau X^{-1}.$$  

We want to show that 

$$(1 - \delta)Y^{-1} \preceq Z \preceq (1 + \delta)Y^{-1},$$  

where $\delta = 5\tau$.

First observe that for any vector $v$,

$$v^TY^{-1}v = v^TX^{-1}v - \frac{v^TX^{-1}vu^TX^{-1}v}{1 + u^TX^{-1}u} = \frac{v^TX^{-1}v}{1 + u^TX^{-1}u} + \frac{(v^TX^{-1}v)(u^TX^{-1}u) - (u^TX^{-1}v)^2}{1 + u^TX^{-1}u},$$

where in the latter expression, both terms are non-negative. Similarly

$$v^TZv = v^TMv - \frac{v^Tmu^Tu^TMv}{1 + u^TMu} = \frac{v^TMv}{1 + u^TMu} + \frac{(v^TMv)(u^TMu) - (u^TMv)^2}{1 + u^TMu},$$

and again in the final expression, both terms are non-negative. We state two claims that help prove the main lemma.

\textbf{Claim A.7.}

$$\left| \frac{1}{1 + u^TMu} - \frac{1}{1 + u^TX^{-1}u} \right| \leq \frac{\tau}{1 - \tau} \cdot \frac{1}{1 + u^TX^{-1}u}.$$ 

\textbf{Claim A.8.}

$$\left| (v^TX^{-1}v)(u^TX^{-1}u) - (v^TX^{-1}u)^2 - (v^TMv)(u^TMu) - (u^TMv)^2 \right| \leq 2(\tau + \tau^2) \left( (v^TX^{-1}v)(u^TX^{-1}u) - (u^TX^{-1}v)^2 \right).$$ 

We also make frequent use of the fact that $1 + u^TMu \geq 1 + (1 - \tau)u^TX^{-1}u \geq (1 - \tau)(1 + u^TX^{-1}u)$. Thus

$$\left| v^TY^{-1}v - v^TY^{-1}v \right| \leq \left| v^TMv - \frac{v^TX^{-1}v}{1 + u^TX^{-1}u} \right| + v^TX^{-1}v \cdot \left| \frac{1}{1 + u^TX^{-1}u} - \frac{1}{1 + u^TMu} \right| + \left( (v^TX^{-1}v)(u^TX^{-1}u) - (v^TX^{-1}u)^2 \right) \left| \frac{1}{1 + u^TMu} - \frac{1}{1 + u^TA^{-1}u} \right| \leq 2\tau \cdot \frac{v^TX^{-1}v}{1 + u^TX^{-1}u} + 3\tau + 2\tau^2 \cdot \frac{v^TX^{-1}v)(u^TX^{-1}u) - (u^TX^{-1}v)^2}{1 + u^TX^{-1}u} \leq \frac{3\tau + 2\tau^2}{1 - \tau} v^TY^{-1}v.$$ 

\square
Proof of Claim A.7

\[
\frac{1}{1 + u^T Mu} - \frac{1}{1 + u^T X^{-1} u} = \frac{u^T E u}{(1 + u^T Mu)(1 + u^T X^{-1} u)} \\
\leq \frac{1}{1 + u^T Mu} \cdot \frac{u^T X^{-1} u}{1 + u^T X^{-1} u} \\
\leq \frac{\tau}{1 - \tau} \cdot \frac{1}{1 + u^T X^{-1} u}.
\]

\[\square\]

Proof of Claim A.8

Let

\[v = \alpha \hat{v} \text{ where } \hat{v} X^{-1} \hat{v} = 1,\]
\[u = \beta \hat{u} \text{ where } \hat{u} X^{-1} \hat{u} = 1.\]

Also let \(\bar{u} = \gamma \hat{v} + \sqrt{1 - \gamma^2} \hat{w}, \) where \(\hat{w} X^{-1} \hat{w} = 0.\) Now

\[1 = \bar{u} X^{-1} \bar{u} = \gamma^2 + (1 - \gamma^2) \hat{w} X^{-1} \hat{w},\]

so \(\hat{w} X^{-1} \hat{w} = 1.\) Thus

\[(v^T X^{-1} v)(u^T X^{-1} u) - (v^T X^{-1} u)^2 = \alpha^2 \beta^2 (1 - \gamma^2).\]

(14)

And

\[(v^T M v)(u^T M u) - (v^T M u)^2 = \alpha^2 \beta^2 \left[\hat{v}^T M \hat{v} \left(\gamma \hat{v} + \sqrt{1 - \gamma^2} \hat{w}\right)^T \left(\gamma \hat{v} + \sqrt{1 - \gamma^2} \hat{w}\right) - \left(\hat{v}^T M \hat{v} + \sqrt{1 - \gamma^2} \hat{w} \right)^2 \right] \\
= \alpha^2 \beta^2 (1 - \gamma^2) \left[\left(\hat{v}^T M \hat{v} \hat{w} - (\hat{v}^T M \hat{w})^2 \right] \\
= \alpha^2 \beta^2 (1 - \gamma^2) \left[\left(1 + \hat{v}^T E \hat{v} + \hat{w}^T E \hat{w} \right) \right].
\]

Thus

\[\left|(v^T X^{-1} v)(u^T X^{-1} u) - (v^T X^{-1} u)^2\right| = \alpha^2 \beta^2 (1 - \gamma^2) \left|1 - \left(\hat{v}^T E \hat{v} + \hat{w}^T E \hat{w} \right) - (\hat{v}^T E \hat{w})^2\right| \\
= \alpha^2 \beta^2 (1 - \gamma^2) \left|\hat{v}^T E \hat{v} + \hat{w}^T E \hat{w} + (\hat{v}^T E \hat{w})^2\right| \leq \alpha^2 \beta^2 (1 - \gamma^2) 2(\tau + \tau^2).
\]

To establish the final inequality, we used that \(\|X^{1/2} E X^{1/2}\| \leq \tau,\) and hence

\[|\hat{v}^T E \hat{w}| \leq \tau |\hat{v}^T X^{-1} \hat{w}| \leq \tau.
\]

Combined with Equation (14), this proves the claim.  \[\square\]

A.4 Starting Point

Algorithm 4: Algorithm GoodStart: Given an instance \((G, y),\) outputs feasible starting point \((x_0, t_0)\).

1. Use a linear time DFS to compute a topological sort on \(G\) to order vertices in a sequence \((v_1, \ldots, v_n),\) s.t.

   for every edge \((v_i, v_j), \) \(i < j.\)

2. for \(i \leftarrow 1, \ldots, n:\)

   \(x_0(v_i) \leftarrow i/n.\)

3. for \(i \leftarrow 1, \ldots, n:\)
\[ t_0(v_i) \leftarrow |x_0(v_i) - y(v_i)|^p + 1. \]

We prove the following claim, which in turn will help us prove Lemmas \ref{lem:2.4} and \ref{lem:2.5}.

**Claim A.9.** Let \((x_0, t_0)\) be the point returned by \textsc{GoodStart}. For every vertex \(v\),
\[
0 \leq x_0(v) \leq 1.
\]

**Proof.** Follows immediately from the \textsc{GoodStart} algorithm. \hfill \Box

**Proof of Lemma \ref{lem:2.4}.** We start by observing that the point \((x_0, t_0)\) computed \textsc{GoodStart} is feasible for program (5). This is true because the topological sort ensures that for every edge \((a, b)\), the indices \(i_a\) and \(i_b\) assigned to vertices \(a\) and \(b\) satisfy \(i_a < i_b\) and hence \(x(b) - x(a) = \frac{1}{n}(i_a - i_b) > 0\). Meanwhile, the assignment \(t_0(v_i) = |x_0(v_i) - y(v_i)|^p + 1\) ensures that constraints on \(t\) are not violated. By Claim A.9, \(\langle w^p, t_0 \rangle \leq 2n w^p_{\text{max}} < K = 3n w^p_{\text{max}}\). Hence \((x_0, t_0)\) is also feasible for program (6). Thus, the domain \(D_K\) is non-empty, as \((x_0, t_0)\) is contained in it. Let \((x^*, t^*)\) be a feasible, optimal point for program (5), then clearly \(\langle w^p, t^* \rangle \leq \langle w^p, t_0 \rangle < K\), so this point is feasible for program (6), and thus \(\text{OPT}_{\text{bdd}} \leq \text{OPT}_{\text{lin}} = \text{OPT}_{p, \text{ISO}}\). And, as program (5) is a relaxation of program (6), it follows that \(\text{OPT}_{\text{bdd}} \geq \text{OPT}_{\text{lin}} = \text{OPT}_{p, \text{ISO}}\).

Finally, \(D_K\) is bounded, because for each vertex \(v\), \(0 \leq t(v) \leq K\), and \(y(v) - K^{1/p} \leq x(v) \leq y(v) + K^{1/p}\). \hfill \Box

**Proof of Lemma \ref{lem:2.5}.** Recall that
\[
sym(z, D_K) = \inf_{q \in \partial D_K} \inf \left\{ s > 0 : z + \frac{(z - q)}{s} \in D_K \right\}.
\]

Hence for any norm \(\|\cdot\|\)
\[
sym(p, D_K) \geq \inf_{q \in \partial D_K} \frac{\|q - p\|}{\sup_{r \in \partial D_K} \|r - p\|}.
\]

We use a norm given by \(\|(x, t)\| = \|x\|_{\infty} + \|t\|_{\infty}\). By giving upper and lower bounds on the distance from \((x_0, t_0)\) to the boundary of \(D_K\) in this norm, we can lower bound the symmetry of this point.

\[
\max_{(t, x) \in \partial D_K} \|(x - x_0, t - t_0)\| = \max_{(t, x) \in \partial D_K} \|x - x_0\|_{\infty} + \|t - t_0\|_{\infty} \\
\leq 2 \cdot K^{1/p} + K \leq 6n w^p_{\text{max}}.
\]

because for each vertex \(v\), we have \(0 \leq t(v) \leq K\), and \(y(v) - K^{1/p} \leq x(v) \leq y(v) + K^{1/p}\).

For every point \((x, t)\) on the boundary of \(D_K\), we lower bound the minimum distance to \(\|(x - x_0, t - t_0)\|\) by considering several conditions:

1. The value constraint \(\langle 1, t \rangle \leq K\) is active, i.e. \(\langle 1, t \rangle = K\).
2. \(x(a) = x(b)\) for some edge \((a, b) \in E\).
3. \(|x(a) - y(a)|^p = t(a)\) for some \(v \in V\).

At least one of the above conditions must hold for \((x, t)\) to be on the boundary of \(D_K\). We will show that each condition individually is sufficient to lower bound the distance to \((x_0, t_0)\).

**Condition** \[ (1, t) = K. \] Then
\[
\|(x - x_0, t - t_0)\| \geq \|t - t_0\|_{\infty} \geq \frac{1}{n} \|t - t_0\|_1 \geq \frac{1}{n} (\|t\|_1 - \|t_0\|_1) \geq \frac{1}{n} (K - 2n) \geq w^p_{\text{max}}.
\]
**Condition** 2: \( x(a) = x(b) = \gamma \) for some edge \((a, b) \in E\). Then
\[
\| (x - x_0, t - t_0) \| \geq \| x - x_0 \|_\infty \\
\geq \frac{1}{2} (\| x(b) - x_0(b) \| + \| x(a) - x_0(a) \|) \\
= \frac{1}{2} (\| \gamma - x_0(b) \| + \| \gamma - x_0(a) \|) \\
\geq \frac{1}{2} (\| x_0(b) - x_0(a) \|) \geq \frac{1}{2n}.
\]

**Condition** 3: \( |x(a) - y(a)| = t(a)^{1/p} \) for some \( a \in V \). We consider two cases. First case is when \( \| t - t_0 \|_\infty \geq 1/2 \).

In the second case is when \( \| t - t_0 \|_\infty < 1/2 \). We write \( x(a) = x_0(a) + \Delta \).
\[
|\Delta + x_0(a) - y(a)|^p = t(a) \geq t_0(v) - \| t - t_0 \|_\infty \\
\geq 1/2 + |x_0(a) - y(a)|^p
\]
As \( p \geq 1 \), the growth rate of \( |\Delta + x_0(a) - y(a)|^p \) is extremized when \( |x_0(a) - y(a)| \) is maximized and as \( x_0, y \in [0, 1] \), we get \( |x_0(a) - y(a)| = 1 \), and hence \(|\Delta|\) is minimized in this case. Thus \( |\Delta| + 1 |^p \geq 1/2 + 1 = 3/2 \). Consequently,
\[
|\Delta| \geq \left( \frac{3}{2} \right)^{1/p} - 1 = \exp \left[ \frac{\log(3/2)}{p} \right] - 1 \geq \frac{\log(3/2)}{p} \geq \frac{1}{3p}.
\]
Thus,
\[
\text{sym}((x_0, t_0), D_K) \geq \min \left( \frac{1}{3p}, \frac{1}{6n} \right) \geq \frac{1}{18n^2 \mu_{\max}}.
\]

### A.5 Primal Path Following IPM with Approximate Hessian Inverse

Algorithm 5: Algorithm **ISOTONICIPM**:
Run **APPROXIPM** with:
Objective vector \( c = (0, w^p)^T \) s.t. \( (0, w^p)^T (x, t) = \sum_{v \in V} w^p(v)t(v) \).
Gradient function \( g = g_{FK} \).
Hessian function \( M = \text{HESSIAN}\text{SOLVE} \) with \( \mu = 1/n^3 \).
Complexity parameter \( \theta(f) = \theta(F_K) = O(m) \).
Symmetry lower bound \( s = \frac{18n^2 \mu_{\max}}{\delta K} \).
Value upper bound \( K = 3n w^p_{\max} \).
Error parameter \( \epsilon = \frac{\delta}{K} \).
Starting point \((x_0, t_0)\) given by **GOODSTART**(\(G, y\)).
**APPROXIPM** outputs \((x_{apx}, t_{apx})\).
Return \( x_{apx} \).

Algorithm 6: Algorithm **APPROXIPM**: Given an objective vector \( c \in \mathbb{R}^n \), a gradient function \( g : \mathbb{R}^n \to \mathbb{R}^n \), a Hessian function \( M : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n \), a complexity parameter \( \theta(f) \), a feasible starting point \( x_0 \), a symmetry lower bound \( s > 0 \), a value upper bound \( K \geq 0 \), and an error parameter \( \epsilon > 0 \), outputs a vector \( x_{apx} \).

1. \( x \leftarrow x_0 \).
   \( \rho \leftarrow 1 \).
\[ T_1 \leftarrow 20 \sqrt{\theta(f)} \log (30 \beta(f)(1 + 1/s)). \]

2. for \( i \leftarrow 1, \ldots, T_1 \):
   \[
   \begin{align*}
   \rho & \leftarrow \rho \cdot \left(1 - \frac{1}{20 \sqrt{\theta(f)}}\right) \\
   z & \leftarrow -\rho g(x_0) + g(x) \\
   x & \leftarrow x - M(x, z)
   \end{align*}
   \]

3. \( \alpha \leftarrow \sqrt{c^2 M(x, c)} \)
   \[
   \eta \leftarrow \frac{\alpha}{500}. \\
   z \leftarrow \eta c + g(x) \\
   x \leftarrow x - M(x, z) \\
   T_2 \leftarrow 20 \sqrt{\theta(f)} \log \left(\frac{66 \beta(f)}{\epsilon}\right).
   \]

4. for \( i \leftarrow 1, \ldots, T_2 \):
   \[
   \begin{align*}
   \eta & \leftarrow \eta \cdot \left(1 + \frac{1}{20 \sqrt{\theta(f)}}\right) \\
   z & \leftarrow \eta c + g(x) \\
   x & \leftarrow x - M(x, z)
   \end{align*}
   \]

5. return \( x_{\text{apx}} \leftarrow x. \)

In this section we prove Theorem 2.2. We start by proving a central lemma shows that approximate Newton steps are sufficient to ensure convergence of our primal path following IPM.

The rest of this section is a matter of connecting this statement with Renegar’s primal following machinery.

**Lemma A.10.** Assume \( f \in \text{SC} \) and is defined on a domain \( D \). If \( \delta = \|H(x)^{-1}g(x)\|_{H(x)} \leq \frac{1}{\tau}, \tau < 1, \) and

\[
(1 - \tau)H(x)^{-1} \preceq M \preceq (1 + \tau)H(x)^{-1},
\]

then taking \( x_+ = x - M g(x) \) will ensure both that \( x_+ \in D \) and

\[
\|H(x_+)^{-1}g(x_+)\|_{H(x_+)} \leq \frac{1}{1 - (1 + \tau)\delta} \left(\tau\delta + \frac{(1 + \tau)\delta^2}{1 - (1 + \tau)\delta}\right).
\]

**Proof.** For brevity write \( H_x = H(x) \). Firstly,

\[
\|x_+ - x\|_{H_x} = \|M g(x)\|_{H_x} \leq (1 + \tau) \|H_x^{-1}g(x)\|_{H_x} = (1 + \tau)\delta < 1,
\]

which guarantees feasibility of \( x_+ \). Further,

\[
\|I - MH_x\|_{H_x}^2 = \max_{\|y\|_{H_x} = 1} y^T (I - H_x M) H_x (I - MH_x) y
\]

\[
= \max_{\|y\|_{H_x} = 1} y^T H_x^{1/2} (I - H_x^{1/2} M H_x^{1/2}) (I - H_x^{1/2} M H_x^{1/2}) H_x^{1/2} y
\]

\[
= \max_{\|y\|_{H_x} = 1} y^T H_x^{1/2} (I - H_x^{1/2} M H_x^{1/2})^2 H_x^{1/2} y
\]

\[
\leq \max_{\|y\|_{H_x} = 1} \tau^2 y^T H_x y = \tau^2
\]

Then

\[
\|H_x^{-1}g(x) - M g(x)\|_{H_x} = \|(I - MH_x)H_x^{-1}g(x)\|_{H_x} \leq \|I - MH_x\|_{H_x} \|H_x^{-1}g(x)\|_{H_x} \leq \tau \|H_x^{-1}g(x)\|_{H_x}.
\]

20
Now,
\[
H_x^{-1}g(x+) = H_x^{-1}g(x) + \int_0^1 H_x^{-1}H(x + t(x_+ - x))(x_+ - x) \, dt
\]
\[
= (H_x^{-1}g(x) - Mg(x)) + Mg(x) + \int_0^1 H_x^{-1}H(x + t(x_+ - x))(x_+ - x) \, dt
\]
\[
= (H_x^{-1}g(x) - Mg(x)) + \int_0^1 [I - H_x^{-1}H(x + t(x_+ - x))] M g(x) \, dt
\]

Thus, using Theorem A.1
\[
\|H_x^{-1}g(x+)|_{H_x} \leq \|H_x^{-1}g(x) - Mg(x)|_{H_x} + \left\| \int_0^1 [I - H_x^{-1}H(x + t(x_+ - x))] M g(x) \, dt \right\|_{H_x}
\]
\[
\leq \tau \|H_x^{-1}g(x)|_{H_x} + \int_0^1 \|I - H_x^{-1}H(x + t(x_+ - x))|_{H_x} \, dt \|M g(x)|_{H_x}
\]
\[
\leq \tau \delta + (1 + \tau)\delta \int_0^1 \frac{1}{1 - t(1 + \tau)\delta} - 1 \, dt
\]
\[
\leq \tau \delta + \frac{(1 + \tau)^2}{1 - (1 + \tau)\delta}.
\]

Finally, we can use the self-concordance of \( f \) to get
\[
\|H(x_+)^{-1}g(x_+)|_{H(x_+)} \leq \frac{1}{1 - \|x_+ - x|_{H_x}} \|H_x^{-1}g(x_+)|_{H_x}
\]
\[
\leq \frac{1}{1 - (1 + \tau)\delta} \left( \tau \delta + \frac{(1 + \tau)^2}{1 - (1 + \tau)\delta} \right).
\]

For completeness, we now restate several results from a textbook by Renegar [32].

**Definition A.11.** Consider a function \( f \in SC \) with bounded domain \( D_f \). Let \( \overline{D_f} \) be the closure of the domain. Given an objective vector \( c \), we define the associated minimization problem as
\[
\min \quad \langle c, x \rangle
\]
subject to \( x \in \overline{D_f} \),
(15)

and, we define the associated \( \eta \)-minimization problem as
\[
\min \quad \eta \langle c, x \rangle + f(x)
\]
subject to \( x \in D_f \).
(16)

For each \( \eta \), let \( z(\eta) \in D_f \) denote an optimum of the \( \eta \)-minimization problem.

Using this definition, we can state two lemmas, which are proven by Renegar, and appear equations (2.13) and (2.14) in [32].
Lemma A.12. Given a function \( f \in SC \) with bounded domain \( D_f \) and an objective vector \( c \), let \( \text{OPT} \) denote the value of the associated minimization problem. Then for any \( \eta > 0 \) and any \( x \in D_f \)
\[
\|H(x)^{-1}c\|_x \leq \langle c, x \rangle - \text{OPT}.
\]

Lemma A.13. Given a function \( f \in SCB \) with bounded domain \( D_f \) and an objective vector \( c \), let \( \text{OPT} \) denote the value of the associated minimization problem. Then for any \( \eta > 0 \) and any \( x \in D_f \)
\[
\langle c, x \rangle - \text{OPT} \leq \frac{1}{\eta} \theta(f) (1 + \|x - z(\eta)\|_{z(\eta)}),
\]
where \( z(\eta) \) is an optimum of the associated \( \eta \)-minimization problem.

The following is a restricted form of Renegar’s Theorem 2.2.5 \([32]\).

Lemma A.14. Assume \( f \in SC \). If \( \delta = \|H(x)^{-1}g(x)\|_x \leq 1/4 \) for some \( x \in D_f \), then \( f \) has a minimizer \( z \) and
\[
\|z - x\|_x \leq \delta + \frac{3\delta^2}{(1 - \delta)^3}.
\]

The next lemma appears in Renegar \([32]\) as Proposition 2.3.7:

Lemma A.15. Assume \( f \in SCB \). For all \( x, y \in D_f \),
\[
\|H(y)^{-1}g(x)\|_y \leq \left(1 + \frac{1}{\text{sym}(x, D_f)}\right) \theta(f).
\]

Proof of Theorem 2.2: Given a vector \( v \), and \( \gamma > 0 \), let \( f_{v,\gamma}(x) = f(x) + \gamma \langle v, x \rangle \). Let
\[
n_{v,\gamma}(x) = H(x)^{-1} (g(x) + \gamma v) = g_x(x) + \gamma H(x)^{-1} v.
\]
Now, for any \( \gamma_1 \) and \( \gamma_2 \)
\[
n_{v,\gamma_2}(x) = \frac{\gamma_2}{\gamma_1} n_{v,\gamma_1}(x) + \left(\frac{\gamma_2}{\gamma_1} - 1\right) g_x(x).
\]
Thus
\[
\|n_{v,\gamma_2}(x)\|_x \leq \frac{\gamma_2}{\gamma_1} \|n_{v,\gamma_1}(x)\|_x + \left|\frac{\gamma_2}{\gamma_1} - 1\right| \sqrt{\theta(f)}.
\]
Observe that for any \( \gamma \), the Hessian \( H(x) \) of \( f \) is also the Hessian of \( f_{\gamma} \). Consequently, we have \( f_{\gamma} \in SC \) because \( f \in SCB \). Thus by Lemma A.10 applied to the function \( f_{v,\gamma} \), if \( \delta = \|n_{v,\gamma}(x)\|_{H(x)} \leq \frac{1}{2}, \tau < 1 \), and
\[
(1 - \tau) H(x)^{-1} \leq M \leq (1 + \tau) H(x)^{-1},
\]
then for \( x_+ = x - M (g(x) + \gamma v) \), we have \( x_+ \in D_{f_{v,\gamma}} = D_f \) and
\[
\|n_{v,\gamma}(x_+))\|_{x_+} = \|H(x_+)^{-1}(g(x_+) + \gamma_2 v)\|_{H(x_+)} \leq \frac{1}{1 - (1 + \tau)\delta} \left(\tau\delta + \frac{((1 + \tau)\delta)^2}{1 - (1 + \tau)\delta}\right). \tag{17}
\]
Suppose we start with
\[
\|n_{v,\gamma_1}(x)\|_x \leq 1/9,
\]
And take
\[ \gamma_2 = \left( 1 + \frac{1}{20 \sqrt{\theta(f)}} \right) \gamma_1. \]

Then using \( \theta(f) \geq 1 \), we find
\[ \| n_{v, \gamma_2}(x) \|_x \leq 1/6. \]

For \( \tau = 1/10 \), letting \( x_+ = x - M \left( g(x) + \gamma_2 v \right) \), we get
\[ \| n_{v, \gamma_2}(x_+) \|_{x_+} = \| H(x_+)^{-1}(g(x_+) + \gamma_2 v) \|_{H(x_+)} \leq \frac{1}{1 - 11/60} \left( \frac{1/60 + (11/60)^2}{1 - 11/60} \right) < 1/9. \]

Similarly, if we take
\[ \gamma_2 = \left( 1 - \frac{1}{20 \sqrt{\theta(f)}} \right) \gamma_1. \]

then
\[ \| n_{v, \gamma_2}(x) \|_x \leq 1/6. \]

So again, taking \( x_+ = x - M \left( g(x) + \gamma_2 v \right) \) gives
\[ \| n_{v, \gamma_2}(x_+) \|_{x_+} = \| H(x_+)^{-1}(g(x_+) + \gamma_2 v) \|_{H(x_+)} \leq \frac{1}{1 - 11/60} \left( \frac{1/60 + (11/60)^2}{1 - 11/60} \right) < 1/9. \]

With these observations in mind, we are ready to prove the correctness of the APPROXIPM algorithm.

We refer to the for loop in step 2 as phase 1 of the algorithm. In phase 1, we take \( v_1 = -g(x_0) \), so
\[ n_{v_1, \rho}(x) = H(x)^{-1} \left( g(x) - \rho g(x_0) \right). \]

Intially, as \( x = x_0 \), so as \( \rho = 1 \), we \( \| n_{v_1, \rho}(x) \|_x = 0 \leq 1/9 \). Thus, by our observations on decreasing \( \gamma \), we find that after each iteration of the for loop, we get \( \| n_{v_1, \rho}(x) \|_x \leq 1/9 \), and after the \( i \)th iteration of the for loop, we get
\[ \rho \leq \left( 1 - \frac{1}{20 \sqrt{\theta(f)}} \right)^i. \]

When the for loop completes, we thus have
\[ \rho \leq \left( 1 - \frac{1}{20 \sqrt{\theta(f)}} \right)^{20 \sqrt{\theta(f)} \log(30\theta(f)(1+1/s))} \leq \frac{1}{30\theta(f)(1+1/s)}. \]

Hence, for the \( x \) obtained at the end of phase 1, by applying Lemma A.15 and our symmetry lower bound \( s \), we get
\[ \| H(x)^{-1} g(x) \|_x = \| \rho H(x)^{-1} g(x_0) + n_{v_1, \rho}(x) \|_x \leq \rho \| H(x)^{-1} g(x_0) \|_x + \| n_{v_1, \rho}(x) \|_x \leq \rho \theta(f)(1 + 1/s) + 1/9 \leq 1/30 + 1/9 = 13/90. \]

We refer to steps 3 and 4 as phase 2. In phase 2, we consider
\[ n_{c, \eta}(x) = H(x)^{-1} \left( g(x) + \eta c \right). \]

Using \( \sqrt{c^T M c} \geq \sqrt{9 \tau \theta} H(x)^{-1} c \geq \frac{9}{10} \| H(x)^{-1} c \|_x \), we get that at the start of step
\[ n_{c, \eta}(x) = \| \eta H(x)^{-1} c + H(x)^{-1} g(x) \|_x \leq \eta \| H(x)^{-1} c \|_x + \| H(x)^{-1} g(x) \|_x \leq \frac{1}{45} + 13/90 = 1/6. \]
Hence, at the end of step 3 we get $\|n_{c,\eta}(x)\|_x \leq 1/9$. Thus, at the end of each iteration of the for loop in step 4 we also get $\|n_{c,\eta}(x)\|_x \leq 1/9$.

So once the loop completes, using $\sqrt{\epsilon/m}c \leq \frac{1}{10}\|H(x)^{-1}c\|_x$, and that by Lemma A.12 $\|H(x)^{-1}c\|_x \leq K - \text{OPT}$, we have

$$\eta \geq \frac{1}{55} \|H(x)^{-1}c\|_x \left(1 + \frac{1}{20\sqrt{\theta(f)}}\right)^{20\sqrt{\theta(f)} \log \left(\frac{6\theta(f)}{\epsilon}\right)} \geq \frac{6\theta(f)}{5\epsilon(K - \text{OPT})}.$$ 

Now from $\|n_{c,\eta}(x)\|_x \leq 1/9$ and Lemma A.14 applied to $f,\eta$, we get that $\|x - z(\eta)\|_x \leq 1/9 + 3(1/9)^2/(1 - 1/9)^3 \leq 1/6$, and by the self-concordance of $f$, $\|x - z(\eta)\|_{z(\eta)} \leq (1/6)/(1 - 1/6) = 1/5$. Then by Lemma A.13 applied to $f$, we have

$$\langle c, x \rangle - \text{OPT} \leq \frac{\theta(f)}{\eta} (1 + \|x - z(\eta)\|_{z(\eta)}) \leq \epsilon \cdot (K - \text{OPT}).$$

□

B Inf and Lex minimization on DAGs

In this section, we show that given a partially labeled DAG $(G, v_0)$, we can find an inf-minimizer in $O(m)$ time and a lex-minimizer in $O(mn)$ time.

Notations and Convention. We assume that $G = (V, E, \text{len})$ is a DAG and the vertex set is denoted by $V = \{1, 2, ..., n\}$. We further assume that the vertices are topologically sorted. This is a standard routine and is known to take $O(m)$ time. This means that if $(i, j) \in E$, then $i < j$. $\text{len} : E \rightarrow \mathbb{R}_{\geq 0}$ denotes non-negative edge lengths. For all $x, y \in V$, by $\text{dist}(x, y)$, we mean the length of the shortest directed path from $x$ to $y$. It is set to $\infty$ when no such path exists.

A path $P$ in $G$ is an ordered sequence of (distinct) vertices $P = (x_0, x_1, \ldots, x_k)$, such that $(x_{i-1}, x_i) \in E$ for $i \in [k]$. For notational convenience, we also call repeated pair $(x, x)$ as a path. The endpoints of $P$ are denoted by $\partial_0 P = x_0, \partial_1 P = x_k$. The set of interior vertices of $P$ is defined to be $\text{int}(P) = \{x_i : 0 < i < k\}$. For $0 \leq i < j \leq k$, we use the notation $P[x_i : x_j]$ to denote the subpath $(x_i, \ldots, x_j)$. The length of $P$ is $\text{len}(P) = \sum_{i=1}^k \text{len}(x_{i-1}, x_i)$.

A function $v_0 : V \rightarrow \mathbb{R} \cup \{\ast\}$ is called a labeling (of $G$). A vertex $x \in V$ is a terminal with respect to $v_0$ iff $v_0(x) \neq \ast$. The other vertices, for which $v_0(x) = \ast$, are non-terminals. We let $T(v_0)$ denote the set of terminals with respect to $v_0$. If $T(v_0) = V$, we call $v_0$ a complete labeling (of $G$). We say that an assignment $v : V \rightarrow \mathbb{R} \cup \{\ast\}$ extends $v_0$ if $v(x) = v_0(x)$ for all $x$ such that $v_0(x) \neq \ast$.

Given a labeling $v_0 : V \rightarrow \mathbb{R} \cup \{\ast\}$, and two terminals $x, y \in T(v_0)$ for which $(x, y) \in E$, we define the gradient on $(x, y)$ due to $v_0$ to be

$$\text{grad}^+_{G, v_0}[v_0](x, y) = \max \left\{ \frac{v_0(x) - v_0(y)}{\text{len}(x, y)}, 0 \right\}.$$ 

Here and wherever applicable, we follow the convention $0 / 0 = 0, 0 \cdot \infty = 0$ and $\frac{\text{finite number}}{\infty} = 0$. When $v_0$ is a complete labeling, we interpret $\text{grad}^+_{G, v_0}[v_0]$ as a vector in $\mathbb{R}^m$, with one entry for each edge.

A graph $G$ along with a labeling $v$ of $G$ is called a partially-labeled graph, denoted $(G, v)$. We say that a partially-labeled graph $(G, v_0)$ is a well-posed instance if for every vertex $x \in V$, either there is a path from $x$ to a terminal $t \in T(v_0)$ or there is a path from a terminal $t \in T(v_0)$ to $x$. We note that instances arising from isotonic regression problem are well-posed instances and in fact satisfy a stronger condition. Every vertex lies on a terminal-terminal path.

A path $P$ in a partially-labeled graph $(G, v_0)$ is called a terminal path if both endpoints are terminals. We define $\nabla^+ P(v_0)$ to be its gradient:

$$\nabla^+ P(v_0) = \max \left\{ \frac{v_0(\partial_0 P) - v_0(\partial_1 P)}{\text{len}(P)}, 0 \right\}.$$ 

If $P$ contains no terminal-terminal edges (and hence, contains at least one non-terminal), it is a free terminal path.
Lex-Minimization. An instance of the LEX-MINIMIZATION problem is described by a partially-labeled graph \((G, v_0)\). The objective is to compute a complete labeling \(v : V_G \to \mathbb{R}\) extending \(v_0\) that lex-minimizes \(\text{grad}(v)\).

Definition B.1 (Lex-minimizer). Given a partially-labeled graph \((G, v_0)\), we define \(\text{lex}_G[v_0]\) to be a complete labeling of \(V\) that extends \(v_0\), and such that for every other complete labeling \(v' : V_G \to \mathbb{R}\) that extends \(v_0\), we have \(\text{grad}^+_G(\text{lex}_G[v_0]) \preceq_{\text{lex}} \text{grad}^+_G[v']\). That is, \(\text{lex}_G[v_0]\) achieves a lexicographically-minimal gradient assignment to the edges.

We call \(\text{lex}_G[v_0]\) the lex-minimizer for \((G, v_0)\). Note that if \(T(v_0) = V_G\), then trivially, \(\text{lex}_G[v_0] = v_0\).

Definition B.2. A steepest fixable path in an instance \((G, v_0)\) is a free terminal path \(P\) that has the largest gradient \(\nabla^+ P(v_0)\) amongst such paths.

Observe that if \(\nabla^+ P(v_0) \neq 0\) then the path \(P\) must be a simple path by definition.

Definition B.3. Given a steepest fixable path \(P\) in an instance \((G, v_0)\), we define \(\text{fix}_G[v_0, P] : V_G \to \mathbb{R} \cup \{\ast\}\) to be the labeling defined as follows

\[
\text{fix}_G[v_0, P](x) = \begin{cases} 
  v_0(\partial_0 P) - \nabla^+ P(v_0) \cdot \text{len}_G(P[\partial_0 P : x]) & x \in \text{int}(P) \setminus T(v_0), \\
  v_0(x) & \text{otherwise}.
\end{cases}
\]

We say that the vertices \(x \in \text{int}(P)\) are fixed by the operation \(\text{fix}[v_0, P]\). If we define \(v_1 = \text{fix}_G[v_0, P]\), where \(P = (x_0, \ldots, x_r)\) is the steepest fixable path in \((G, v_0)\), then it is easy to argue that for every \(i \in [r]\), we have \(\text{grad}(v_t)(x_{i-1}, x_i) = \nabla^+ P\).

B.1 Sketch of the Algorithms

We now sketch the ideas behind our algorithms and give precise statements of our results. A full description of all the algorithms is included in the appendix.

We define the pressure of a vertex to be the gradient of the steepest terminal path through it:

\[ \text{pressure}[v_0](x) = \max\{\nabla^+ P(v_0) \mid P\text{ is a terminal path in } (G, v_0) \text{ and } x \in P\}. \]

Observe that in a graph with no terminal-terminal edges, a free terminal path is a steepest fixable path iff its gradient is equal to the highest pressure amongst all vertices. Moreover, vertices that lie on steepest fixable paths are exactly the vertices with the highest pressure. For a given \(\alpha \geq 0\), in order to identify vertices with pressure exceeding \(\alpha\), we compute vectors \(\nu_{\text{High}}[\alpha](x)\) and \(\nu_{\text{Low}}[\alpha](x)\) defined as follows in terms of dist, the metric on \(V\) induced by \(\ell\):

\[ \nu_{\text{Low}}[\alpha](x) = \min_{t \in T(v_0)} \{v_0(t) + \alpha \cdot \text{dist}(x, t)\} \quad \nu_{\text{High}}[\alpha](x) = \max_{t \in T(v_0)} \{v_0(t) - \alpha \cdot \text{dist}(t, x)\}. \]

B.1.1 Lex-minimization on Star Graphs

We first consider the problem of computing the lex-minimizer on a star graph in which every vertex but the center is a terminal. This special case is a subroutine in the general algorithm, and also motivates some of our techniques.

Let \(x_0\) be the center vertex, \(T = L \cup R\) be the set of terminals, and all edges be of the form \((x_i, t)\) if \(t \in R\) and \((t, x)\) if \(t \in L\). The initial labeling is given by \(v : T \to \mathbb{R}\), and we abbreviate \(d(x, t)\) by \(d(t) = \text{len}(x, t)\) if \(t \in R\) and \(\text{dist}(t, x)\) by \(d(t) = \text{len}(t, x)\) if \(t \in L\). We state Corollary 3.4 from [29]

Lemma B.4. Given a well-posed instance \((G, v_0)\) such that \(T(v_0) \neq V_G\), let \(P\) be a steepest fixable path in \((G, v_0)\). Then, \((G, \text{fix}[v_0, P])\) is also a well-posed instance, and \(\text{lex}_G[\text{fix}[v_0, P]] = \text{lex}_G[v_0]\).

From Lemma B.4 we know that we can determine the value of the lex minimizer at \(x\) by finding a steepest fixable path. By definition, we need to find \(t_1 \in L, t_2 \in R\) that maximize the gradient of the path from \(t_1\) to \(t_2\):

\[ \nabla^+(t_1, t_2) = \max \left\{ \frac{v(t_1) - v(t_2)}{d(t_2) + d(t_1)} \right\}. \]

As observed above, this is equivalent to finding a terminal with the highest pressure. We now present a simple randomized algorithm for this problem that runs in expected linear time.
Given a terminal \( t_1 \in L \) (or \( t_2 \in R \)), we can compute its pressure \( \alpha \) along with the terminal \( t_2 \) such that either \( \nabla^+(t_1, t_2) = \alpha \) in time \( O(|T|) \) by scanning over the terminals in \( R \) (or terminals in \( L \)). Now sample a random terminal \( t_1 \in L \) and a random terminal \( t_2 \in R \). Let \( \alpha_1 \) be the pressure of \( t_1 \) and \( \alpha_2 \) be the pressure of \( t_2 \), and set \( \alpha = \max\{\alpha_1, \alpha_2\} \). We will show that in linear time one can then find the subset of terminals \( T' = L' \cup R' \) such that \( L' \subset L \), \( R' \subset R \) whose pressure is greater than \( \alpha \). Assuming this, we complete the analysis of the algorithm. If \( L' = \emptyset \) (or \( R' = \emptyset \)), \( t_1 \) (or \( t_2 \)) is a vertex with highest pressure. Hence the path from \( t_1 \) to \( t_3 \) (or \( t_3 \) to \( t_2 \)) is a steepest fixable path, and we return \( (t_1, t_3) \) (or \( (t_4, t_2) \)). If neither \( L' \neq \emptyset \) nor \( R' \neq \emptyset \) the terminal with the highest pressure must be in \( T' \), and we recur by picking a new random \( t_1 \in L' \) and \( t_2 \in R' \). As the size of \( T' \) will halve in expectation at each iteration, the expected time of the algorithm on the star is \( O(|T|) \).

To determine which terminals have pressure exceeding \( \alpha \), we observe that the condition \( \exists t_2 \in R : \alpha < \nabla^+(t_1, t_2) = \frac{v(t_2) - v(t_1)}{d(t_2) - d(t_1)} \) is equivalent to \( \exists t_2 \in R : v(t_2) + \alpha d(t_2) < v(t_1) - \alpha d(t_1) \). This, in turn, is equivalent to \( v_{\text{Low}}[\alpha](x) < v(t_1) - \alpha d(t_1) \). We can compute \( v_{\text{Low}}[\alpha](x) \) in deterministic \( O(|T|) \) time. Similarly, we can check if \( \exists t_2 \in L : \alpha < \nabla^+(t_2, t_1) \) by checking if \( v_{\text{High}}[\alpha](x) > v(t_1) + \alpha d(t_1) \). Thus, in linear time, we can compute the set \( T' \) of terminals with pressure exceeding \( \alpha \). The above algorithm is described in Algorithm \([16]\) (named StarSteepestPath).

**Theorem B.5.** Given a pair of set of terminals \( (L, R) \), initial labeling \( v : (L \cup R) \rightarrow \mathbb{R} \), and distances \( d : L \cup R \rightarrow \mathbb{R}_{\geq 0} \), StarSteepestPath\((T, v, d)\) returns \((t_1, t_2)\) with \( t_1 \in L, t_2 \in R \) maximizing \( \frac{v(t_1) - v(t_2)}{d(t_1) + d(t_2)} \), and runs in expected time \( O(|L \cup R|) \).

### B.1.2 Lex-minimization on General Graphs

Theorem 3.3 in [29] gives an algorithm MetaLex that computes lex-minimizers given an algorithm for finding a steepest fixable path in \((G, v_0)\). Though the theorem is proven for undirected graphs, the same holds for directed graphs as long as the steepest path has gradient > 0. When the gradient of the steepest path is equal to 0, the labelings are no more unique. But one can still label all the vertices in \( O(m) \) time by a two stage algorithm so that all the new gradients are zero. In the first stage, we label all the vertices \( x \) such that there is a path from some terminal \( t \in T \) to \( x \). We label \( x \) with the label of the highest labeled terminal from which there is a path to \( x \). This is the least possible label we can assign to \( x \) in order to not create any positive gradient edges. If this procedure creates any positive gradient edges, then it would imply that the the steepest path gradient was not 0 to begin with, which we know, is false. Hence, this creates only 0 gradient edges. The steepest fixable path has zero gradient since after stage one, none of the unlabeled vertices lie on a terminal-terminal path. In the second stage, we label all the remaining vertices. An unlabeled vertex \( x \) is now labeled with the label of the least labeled terminal to which there is a path from \( x \). It is again easy to see that this doesn’t create any zero gradient edges. The routine AssignWithZeroGradient (Algorithm \([14]\)) achieves this in \( O(m) \) time. Recall that finding a steepest fixable path is equivalent to finding a path with gradient equal to the highest pressure amongst all vertices. In this section, we show how to do this in expected time \( O(m) \) for DAGs.

We now describe an algorithm VertexSteepestPath that finds a terminal path \( P \) through any vertex \( x \) such that \( \nabla^+ P(v_0) = \text{pressure}[v_0](x) \) in expected \( O(m) \) time. Using topological ordering we compute \( \text{dist}(x, t), \text{dist}(t, x) \) for all \( t \in T \) in \( O(m) \) time. This is similar to Dijkstra’s algorithm, but instead of using a heap, we can go through the vertices in order (from \( x \) to \( n \)), and when we are at vertex \( i \), we update the distance for all \( j : (i, j) \in E \). If \( x \in T(v_0) \), then there must be a terminal path \( P \) that starts or ends at \( x \) that has \( \nabla^+ P(v_0) = \text{pressure}[v_0](x) \). To compute such a \( P \) we examine all \( t \in T(v_0) \) in \( O(|T|) \) time to find the \( t \) that maximizes \( \nabla^+(x, t) = \frac{v(t) - v(x)}{\text{dist}(x, t)} \) and then return a shortest path between \( t \) and \( x \) depending on which is higher. If \( x \not\in T(v_0) \), then the steepest path through \( x \) between terminals \( t_1 \) and \( t_2 \) must consist of shortest paths between \( t_1 \) and \( x \) and between \( x \) and \( t_2 \). If there is no terminal path that contains \( x \), then we return \((x, x)\) as the steepest path, which by our convention has gradient 0. We assume this is not case. Thus, we can reduce the problem to that of finding the steepest path in a star graph where \( x \) is the only non-terminal and is connected to each terminal \( t \) by an edge of length \( \text{dist}(x, t) \). By Theorem B.5 we can find this steepest path in \( O(|T|) \) expected time. The above algorithm is formally described as Algorithm \([15]\).

**Theorem B.6.** Given a well-posed instance \((G, v_0)\), and a vertex \( x \in V_G \), VertexSteepestPath\((G, v_0, x)\) returns a terminal path \( P \) through \( x \) such that \( \nabla^+ P(v_0) = \text{pressure}[v_0](x) \) in \( O(m) \) expected time.
As in the algorithm for the star graph, we need to identify the vertices whose pressure exceeds a given $\alpha$. For a fixed $\alpha$, we can compute $v_{\text{Low}}[\alpha](x)$ and $v_{\text{High}}[\alpha](x)$ for all $x \in V_G$ using topological ordering in $O(m)$ time. We describe the algorithms COMPVHIGH, COMPVLOW for these tasks in Algorithms 8 and 9. The following lemma encapsulates the usefulness of $v_{\text{Low}}$ and $v_{\text{High}}$.

**Lemma B.7.** For every $x \in V_G$, $\text{pressure}[v_0](x) > \alpha$ iff $v_{\text{High}}[\alpha](x) > v_{\text{Low}}[\alpha](x)$.

It immediately follows that the algorithm COMPHIGHPRESSGRAPH($G$, $v_0$, $\alpha$) described in Algorithm 11 computes the vertex induced subgraph on the vertex set $\{x \in V_G | \text{pressure}[v_0](x) > \alpha\}$.

We can combine these algorithms into an algorithm STEEEPSTPATH that finds the steepest fixable path in ($G$, $v_0$) in $O(m)$ expected time. We may assume that there are no terminal-terminal edges in $G$. We sample an edge ($x_1$, $x_2$) uniformly at random from $E_G$, and a terminal $x_3$ uniformly at random from $V_G$. For $i = 1, 2, 3$, we compute the steepest terminal path $P_i$ containing $x_i$. By Theorem B.6, this can be done in $O(m)$ expected time. Let $\alpha$ be the largest gradient $\max_i \nabla^\top P_i$. As mentioned above, we can identify $G'$, the induced subgraph on vertices $x$ with pressure exceeding $\alpha$, in $O(m)$ time. If $G'$ is empty, we know that the path $P_i$ with largest gradient is a steepest fixable path. If not, a steepest fixable path in ($G$, $v_0$) must be in $G'$, and hence we can recurse on $G'$. Since we picked a uniformly random edge, and a uniformly random vertex, the expected size of $G'$ is at most half that of $G$. Thus, we obtain an expected running time of $O(m)$. This algorithm is described in detail in Algorithm 12.

### B.1.3 Linear-time Algorithm for Inf-minimization

Given the algorithms in the previous section, it is straightforward to construct an infinity minimizer. Let $\alpha^*$ be the gradient of the steepest terminal path. From Lemma 3.5 in [29], we know that the norm of the inf minimizer is $\alpha^*$. Considering all trivial terminal paths (terminal-terminal edges), and using STEEEPSTPATH, we can compute $\alpha^*$ in randomized $O(m)$ time. It is well known ([34, 35]) that $v_1 = v_{\text{Low}}[\alpha^*]$ and $v_2 = v_{\text{High}}[\alpha^*]$ are inf-minimizers. One slight issue occurs when a vertex $x$ does not lie on a terminal-terminal path. In such a case, one of $v_{\text{Low}}[\alpha^*](x)$ or $v_{\text{Low}}[\alpha^*](x)$ will not be finite. But the routine AssignWithZeroGradient described earlier can be used to fix the values of such vertices. It is also known that $\frac{1}{2}(v_1 + v_2)$ is the inf-minimizer that minimizes the maximum $\ell_\infty$-norm distance to all inf-minimizers. For completeness, the algorithm is presented as Algorithm 10 and we have the following result.

**Theorem B.8.** Given a well-posed instance ($G$, $v_0$), COMPINFMIN($G$, $v_0$) returns a complete labeling $v$ of $G$ extending $v_0$ that minimizes $\|\text{grad}^\top [v]\|_\infty$, and runs in randomized $O(m)$ time.

### B.2 Algorithms

**Algorithm 7: MODDIJKSTRA($G$, $v_0$, $\alpha$):** Given a well-posed instance ($G$, $v_0$), a gradient value $\alpha \geq 0$, outputs a complete labeling $v$ of $G$, and an array parent : $V \to V \cup \{\text{null}\}$.

1. for $i = 1$ to $n$
2. if $v_0(i) \neq \ast$ then set $v(i) = +\infty$ else set $v(i) = v_0(i)$
3. parent($i$) $\leftarrow$ null.
4. for $i = 1$ to $n$
5. for $j > i : (i, j) \in E_G$
6. if $v(j) > v(i) + \alpha \cdot \text{len}(i, j)$
7. Decrease $v(j)$ to $v(i) + \alpha \cdot \text{len}(i, j)$.
8. parent($j$) $\leftarrow i$.
9. return ($v$, parent)

**Theorem B.9.** For a well-posed instance ($G$, $v_0$) and a gradient value $\alpha \geq 0$, MODDIJKSTRA computes in time $O(m)$ a complete labeling $v$ and an array parent : $V \to V \cup \{\text{null}\}$ such that, $\forall x \in V_G$, $v(x) = \min_{t \in T(v_0)} \{v_0(t) + \alpha \text{dist}(t, x)\}$. Moreover, the pointer array parent satisfies $\forall x \not\in T(v_0)$ such that parent($x$) $\neq$ null, $v(x) = v(\text{parent}(x)) + \alpha \cdot \text{len}(\text{parent}(x), x)$.

27
Algorithm 8: Algorithm COMPVLOW\((G, v_0, \alpha)\): Given a well-posed instance \((G, v_0)\), a gradient value \(\alpha \geq 0\), outputs \(v_{\text{low}}\), a complete labeling for \(G\), and an array \(L\text{Parent} : V \rightarrow V \cup \{\text{null}\}\).

1. \((v_{\text{low}}, L\text{Parent}) \leftarrow \text{MODDIJKSTRA}(G, v_0, \alpha)\)
2. \textbf{return} \((v_{\text{low}}, L\text{Parent})\)

Algorithm 9: Algorithm COMPVHIGH\((G, v_0, \alpha)\): Given a well-posed instance \((G, v_0)\), a gradient value \(\alpha \geq 0\), outputs \(v_{\text{high}}\), a complete labeling for \(G\), and an array \(H\text{Parent} : V \rightarrow V \cup \{\text{null}\}\).

1. Let \(G_1\) denote the graph \(G\) with all edges reversed in direction.
2. for \(x \in V_G\) do 
3. if \(x \in \mathcal{T}(v_0)\) then \(v_1(x) \leftarrow -v_0(x)\) else \(v_1(x) \leftarrow v_1(x)\).
4. \((\text{temp}, H\text{Parent}) \leftarrow \text{MODDIJKSTRA}(G_1, v_1, \alpha)\)
5. for \(x \in V_{G_1} : v_{\text{high}}(x) \leftarrow -\text{temp}(x)\)
6. \textbf{return} \((v_{\text{high}}, H\text{Parent})\)

\textbf{Corollary B.10.} For a well-posed instance \((G, v_0)\) and a gradient value \(\alpha \geq 0\), let \((v_{\text{low}}[\alpha], L\text{Parent}) \leftarrow \text{COMPVLOW}(G, v_0, \alpha)\) and \((v_{\text{high}}[\alpha], H\text{Parent}) \leftarrow \text{COMPVHIGH}(G, v_0, \alpha)\). Then, \(v_{\text{low}}[\alpha], v_{\text{high}}[\alpha]\) are complete labeling of \(G\) such that, \(\forall x \in V_G\),

\[
    v_{\text{low}}[\alpha]|x) = \min_{t \in \mathcal{T}(v_0)} \{v_0(t) + \alpha \cdot \text{dist}(x, t)\} \quad v_{\text{high}}[\alpha]|x) = \max_{t \in \mathcal{T}(v_0)} \{v_0(t) - \alpha \cdot \text{dist}(t, x)\}.
\]

Moreover, the pointer arrays \(L\text{Parent}, H\text{Parent}\) satisfy \(\forall x \notin \mathcal{T}(v_0), L\text{Parent}(x), H\text{Parent}(x) \neq \text{null and}

\[
    v_{\text{low}}[\alpha]|x) = v_{\text{low}}[\alpha]|(L\text{Parent}(x)) + \alpha \cdot \text{dist}(x, L\text{Parent}(x)),
    v_{\text{high}}[\alpha]|x) = v_{\text{high}}[\alpha]|(H\text{Parent}(x)) - \alpha \cdot \text{dist}(H\text{Parent}(x), x).
\]

Algorithm 10: Algorithm COMPINFMIN\((G, v_0)\): Given a well-posed instance \((G, v_0)\), outputs a complete labeling \(v\) for \(G\), extending \(v_0\) that minimizes \(\|\text{grad}^+ v\|_{\infty}\).

1. \(\alpha \leftarrow \max\{\text{grad}^+ [v_0]|e| \mid e \in E_G \cap (\mathcal{T}(v_0) \times T(v_0))\}\).
2. \(E_G \leftarrow E_G \setminus (\mathcal{T}(v_0) \times T(v_0))\)
3. \(P \leftarrow \text{STEEPESTPATH}(G, v_0)\).
4. \(\alpha \leftarrow \max\{\alpha, \nabla^+ P(v_0)\}\)
5. \((v_{\text{low}}, L\text{Parent}) \leftarrow \text{COMPVLOW}(G, v_0, \alpha)\)
6. \((v_{\text{high}}, H\text{Parent}) \leftarrow \text{COMPVHIGH}(G, v_0, \alpha)\)
7. for \(x \in V_G\) do 
8. if \(x \in \mathcal{T}(v_0)\) then \(v(x) \leftarrow v_0(x)\)
9. else if \(\{v_{\text{low}}(x), v_{\text{high}}(x)\} \cap \{\infty, -\infty\} = \emptyset\) then \(v(x) \leftarrow \frac{1}{2} \cdot (v_{\text{low}}(x) + v_{\text{high}}(x))\).
10. else \(v(x) \leftarrow \ast\)
11. \(v \leftarrow \text{ASSIGNWITHZEROGRADIENT}(G, v)\)
12. \textbf{return} \(v\)

28
Algorithm 11: Algorithm COMPHIGHPRESSGRAPH\((G, v_0, \alpha)\): Given a well-posed instance \((G, v_0)\), a gradient value \(\alpha \geq 0\), outputs a minimal induced subgraph \(G'\) of \(G\) where every vertex has pressure\(v_0(\cdot) > \alpha\).

1. \((\text{vLow, LParent}) \leftarrow \text{COMPVLOW}(G, v_0, \alpha)\)
2. \((\text{vHigh, HParent}) \leftarrow \text{COMPVHIGH}(G, v_0, \alpha)\)
3. \(V_{G'} \leftarrow \{x \in V_G \mid \text{vHigh}(x) > \text{vLow}(x)\}\)
4. \(E_{G'} \leftarrow \{(x, y) \in E_G \mid x, y \in V_{G'}\}\)
5. \(G' \leftarrow (V', E', \text{len})\)
6. \text{return} \(G'\)

**Proof of Lemma B.7:**
\(\text{vHigh}[\alpha](x) > \text{vLow}[\alpha](x)\)
is equivalent to
\[
\max_{t \in T(v_0)} \{v_0(t) - \alpha \cdot \text{dist}(t, x)\} > \min_{t \in T(v_0)} \{v_0(t) + \alpha \cdot \text{dist}(x, t)\},
\]
which implies that there exists terminals \(s, t \in T(v_0)\) such that
\[
v_0(t) - \alpha \cdot \text{dist}(t, x) > v_0(s) + \alpha \cdot \text{dist}(x, s)
\]
thus,
\[
\text{pressure}[v_0](x) \geq \frac{v_0(t) - v_0(s)}{\text{dist}(t, x) + \text{dist}(x, s)} > \alpha.
\]
So the inequality on \(\text{vHigh}\) and \(\text{vLow}\) implies that pressure is strictly greater than \(\alpha\). On the other hand, if \(\text{pressure}[v_0](x) > \alpha\), there exists terminals \(s, t \in T(v_0)\) such that
\[
\frac{v_0(t) - v_0(s)}{\text{dist}(t, x) + \text{dist}(x, s)} = \text{pressure}[v_0](x) > \alpha.
\]
Hence,
\[
v_0(t) - \alpha \cdot \text{dist}(t, x) > v_0(s) + \alpha \cdot \text{dist}(x, s)
\]
which implies \(\text{vHigh}[\alpha](x) > \text{vLow}[\alpha](x)\).

Algorithm 12: Algorithm STEEPESTPATH\((G, v_0)\): Given a well-posed instance \((G, v_0)\), with \(T(v_0) \neq V_G\), outputs a steepest free terminal path \(P\) in \((G, v_0)\).

1. Sample uniformly random \(e \in E_G\). Let \(e = (x_1, x_2)\).
2. Sample uniformly random \(x_3 \in V_G\).
3. \text{for} \(i = 1\ \text{to} \ 3\)
4. \(P \leftarrow \text{VERTEXSTEEPESTPATH}(G, v_0, x_i)\)
5. Let \(j \in \arg\max_{i \in \{1, 2, 3\}} \nabla^+ P_j(v_0)\)
6. \(G' \leftarrow \text{COMPHIGHPRESSGRAPH}(G, v_0, \nabla^+ P_j(v_0))\)
7. \text{if} \(E_{G'} = \emptyset\)
8. \text{then return} \(P_j\)
9. \text{else return} \(\text{STEEPESTPATH}(G', v_0|_{V_{G'}})\)

Algorithm 13: Algorithm COMPLEXMIN\((G, v_0)\): Given a well-posed instance \((G, v_0)\), with \(T(v_0) \neq V_G\), outputs \(\text{lex}_G[v_0]\).

1. \text{while} \(T(v_0) \neq V_G\)
Algorithm 14: Algorithm AssignWithZeroGradient\((G, v_0)\): Given a well-posed instance \((G, v_0)\), with \(T(v_0) \neq V_G\), outputs a complete labeling \(v_0\).

1. \(T \leftarrow T(v_0)\)
2. \(\text{for } i = 1 \text{ to } n : v_0(i) \neq *\)
3. \(\text{for } j > i : (i, j) \in E_G\)
4. \(\text{if } v_0(j) < v_0(i) \text{ or } v_0(j) = *\)
5. \(v_0(j) \leftarrow v_0(i)\)
6. \(T \leftarrow T(v_0)\)
7. \(\text{for } i = n \text{ to } 1 : v_0(i) \neq *\)
8. \(\text{for } j < i : (j, i) \in E_G \text{ and } j \notin T\)
9. \(\text{if } v_0(j) > v_0(i) \text{ or } v_0(j) = *\)
10. \(v_0(j) \leftarrow v_0(i)\)
11. \(\text{return } v_0\)

Algorithm 15: Algorithm VERTEXSTEEPESTPath\((G, v_0, x)\): Given a well-posed instance \((G, v_0)\), and a vertex \(x \in V_G\), outputs a steepest terminal path in \((G, v_0)\) through \(x\).

1. Let \(L := \{i \in T(v_0)\mid \text{there is a path from } i \text{ to } x\}\) and \(R := \{i \in T(v_0)\mid \text{there is a path from } x \text{ to } i\}\)
2. \(\text{if } L = \emptyset \text{ or } R = \emptyset \text{ then return } (x, x)\)
3. Compute \(\text{dist}(t, x)\) for all \(t \in L\) and \(\text{dist}(x, t)\) for all \(t \in R\)
4. \(\text{if } x \in T(v_0)\)
5. \(y_1 \leftarrow \arg \max_{y \in R} \frac{v_0(x) - v_0(y)}{\text{dist}(x, y)}\)
6. \(y_2 \leftarrow \arg \max_{y \in L} \frac{v_0(y) - v_0(x)}{\text{dist}(y, x)}\)
7. \(\text{if } \frac{v_0(x) - v_0(y_1)}{\text{dist}(x, y_1)} \geq \frac{v_0(y_2) - v_0(x)}{\text{dist}(y_2, x)}\)
8. \(\text{return a shortest path from } x \text{ to } y_1\)
9. \(\text{else return a shortest path from } y_2 \text{ to } x\)
10. \(\text{else}\)
11. \(\text{if } t \in L \cup R,\)
12. \((t_1, t_2) \leftarrow \text{STARSTEEPESTPath}(L, R, v_0|L \cup R, d)\)
13. \(L' \leftarrow \{t \in L | \text{dist}(t, x) \geq \text{dist}(x, t)\}\)
14. \(\{P_{t_1, t_2}, P_{t_2, t_1}\} \leftarrow \text{STARSTEEPESTPath}(L, R, v_0|L \cup R, d)\)
15. \(\text{return } P_{t_1, t_2}\)

Algorithm 16: \(\text{STARSTEEPESTPath}(L, R, v, d)\): Returns the steepest path in a star graph, with a single non-terminal connected to terminals in \(T\), with lengths given by \(d\), and labels given by \(v\).

1. \(\text{Sample } t_1 \text{ uniformly and randomly from } L \text{ and } t_2 \text{ uniformly and randomly from } R\)
2. \(\text{Compute } t_3 \in \arg \max_{t \in R} \frac{v(t_1) - v(t)}{d(t_1) + d(t)} \text{ and } t_4 \in \arg \max_{t \in L} \frac{v(t) - v(t_2)}{d(t_1) + d(t)}\)
3. \(\alpha \leftarrow \max \left\{ \frac{v(t_1) - v(t_2)}{d(t_1) + d(t_2)}, \frac{v(t_4) - v(t_2)}{d(t_1) + d(t_2)} \right\}\)
4. \(\text{Compute } v_{\text{low}} \leftarrow \min_{t \in R} (v(t) + \alpha \cdot d(t))\)
5. \(L' \leftarrow \{t \in L \mid v(t) > v_{\text{low}} + \alpha \cdot d(t)\}\)
6. Compute \( v_{\text{high}} \leftarrow \max_{t \in L} (v(t) - \alpha \cdot d(t)) \)
7. \( R' \leftarrow \{ t \in R \mid v(t) < v_{\text{high}} - \alpha \cdot d(t) \} \)
8. if \( L' \cup R' = \emptyset \) then return \((t_1, t_2)\)
9. else return \( \text{STARSTEEPESTPath}(L', R', v|_{L' \cup R'}, d_{L' \cup R'}) \)