Fast Algorithm for Solving Structured Convex Programs

by

Guanghao Ye

Supervised by Yin Tat Lee

A senior thesis submitted in partial fulfillment of

the requirements for the degree of

Bachelor of Science

With Departmental Honors

Computer Science & Engineering

University of Washington

March 2020

Presentation of work given on __________________________

Thesis and presentation approved by __________________________

Date ____________________________
Fast Algorithm for Solving Structured Convex Programs

Guanghao Ye

March 2020

Abstract

Treewidth is a fundamental parameter of structural graph theory which measures the closeness of a graph being a tree. It’s known that many graph problems can be solved more efficiently if the given graph has low treewidth. In this thesis, we study the general convex program \( \min_{\mathbf{Ax} = \mathbf{b}, \mathbf{x} \in K} \mathbf{c}^T \mathbf{x} \) and show this convex program can be solved more efficiently if their underlying graph structure has low treewidth. To obtain this result, we show how to construct an near-optimal cholesky factorization to utilize the low-treewidth property and how to efficiently implement the \( O(\sqrt{d}) \)-iteration interior point method given the factorization.

Contents

1 Introduction 3
   1.1 Our Approaches and Techniques ................................................. 3

2 Projection Matrix Maintenance via Cholesky Decomposition 5
   2.1 Cholesky Factorization ............................................................. 5
   2.2 Computing an Elimination Tree .................................................. 8
   2.3 Sparsity Patterns in \( \mathbf{A} \) and \( \mathbf{L} \), and Associated Matrix Operation Costs ........................................ 11
   2.4 Maintaining the Cholesky Factorization ....................................... 12

3 Initial Point Reduction 13
   3.1 Algorithm ................................................................................. 15
   3.2 Algorithm Explanation ............................................................... 15
   3.3 Correctness ................................................................................. 16
   3.4 Change from \( \mathbf{G} \) to \( \mathbf{G}' \), and from \( \mathbf{T} \) to \( \mathbf{T}' \) .................................................... 16
   3.5 Updated Elimination Order .......................................................... 18

4 Robust Interior Point Algorithm for General Convex Sets 20
   4.1 Robust Central Path Method ......................................................... 20
   4.2 Gradient Descent on \( \Psi_\lambda \) ....................................................... 21
   4.3 Implicit Step ............................................................................... 24
   4.4 Bounding \( \Phi \) under changes of \( \mathbf{x} \) and \( \mathbf{s} \) ............................ 25

5 Central Path Maintenance 32

6 Put It All Together 37

7 Acknowledgement 39

A Appendix 41
   A.1 Hyperbolic function lemmas ....................................................... 41
   A.2 Extension of Locally Strongly Convex Function ............................... 41
1 Introduction

Treewidth is a fundamental parameter of structural graph theory which measures closeness of a graph being a tree. It’s known that many graph problems can be solved more efficiently if the given graph has low treewidth. In this paper, we show that a general class of optimization problems can be solved more efficiently if their underlying graph structure has low treewidth. More specifically, we study the following convex optimization problem:

$$\min_{Ax=b, x \in K_i \text{ for } i \in \{m\}} c^T x$$

where $x_i \in K_i \subset \mathbb{R}^{n_i}$ and $x$ is the concatenation of $x_i$ lying inside the domain $K \equiv \prod_{i=1}^{n} K_i \subset \mathbb{R}^n$ with $n = \sum_{i=1}^{m} n_i$. We associate the constraint matrix $A$ with a graph $G_A$, which can be constructed as follows: Each block of variables in $A$ corresponds a vertex and the adjacency matrix is given by the non-zero pattern of $AA^T$ where we contract each block of matrix in the product into a single entry. Then, we define the treewidth of $A$, $\text{tw}(A)$, to be the treewidth of the underlying graph $G_A$. This construction ties into linear programming formulations of flow problems, in which the constraint matrix $A$ is the incidence matrix of underlying graph and $AA^T$ is precisely the adjacency graph.

In practice, a large number of graphs do have low treewidth; all planar graphs have treewidth $O(\sqrt{n})$ treewidth, where $n$ is the number of vertices. There are also a lot of optimization problems have small treewidth in their underlying graph. For example, one simple model for signal denoising in the following: Given an input signal $f \in \mathbb{R}^n$, find an output $u$ that minimizes the function

$$E(u) = \sum_{i=1}^{n} |u_i - f_i| + |u_{i+1} - u_i|,$$

where the first term restricts the output signal to be close to the input, and the second term controls the amount of irregularity, and for this kind of problem, its underlying graph has constant treewidth.

The current fastest algorithm for solving Eq. (1.1) are the IPM of Lee, Song, and Zhang [LSZ19], in which they combined ideas from interior point methods, sketching, fast matrix multiplication, and data structures to obtain an randomized algorithm in the current matrix multiplication time, $O(n^{2.373...})$. We note that the current fastest algorithm for solving linear systems, a special case of general convex optimization problem above, involves matrix multiplication; hence the result [LSZ19] achieves the current best runtime for this kind of convex optimization problem. This raises a natural question: can we obtain an even faster algorithm for this kind of convex optimization problem when the underlying graph has low treewidth?

In this paper, we resolve this question in affirmative and achieves the following guarantee:

**Theorem 1.1** (Main theorem, informal). Given a matrix $A \in \mathbb{R}^{d \times n}$, two vector $b \in \mathbb{R}^d$, $c \in \mathbb{R}^n$ and $m$ compact convex sets $K_1, \ldots, K_m$. Assume that there is no redundant constraints and $n_i = O(1)$, then there exists an randomized algorithm that solves the optimization problem Eq. (1.1) in time $\tilde{O}(n^{1.25} \cdot \text{tw}(A)^3)$.

1.1 Our Approaches and Techniques

For this paper, we split the proof of Theorem 1.1 into two parts. In the first part, we obtain a fast algorithm for computing a Cholesky decomposition with approximate minimum elimination tree height up to a polylogarithmic factor. In the second part, we show how to obtain a $O(\sqrt{m})$ iteration IPM that is robust under $l_\infty$ perturbation in the measure of Hessian norm and how to implement this IPM fast with a given Cholesky decomposition with small elimination tree height.

1.1.1 Robust Central Path Method

Our algorithm is based on interior point methods which follow some path $x(t)$ inside the the interior of the domain $K$. The path starts at some interior point of the domain $x(1)$ and ends at the solution $x(0)$ we want.
to find. One commonly used path is defined by

\[ x(t) = \arg \min_{ Ax = b } c^T x + t \sum_{i=1}^{m} w_i \phi_i(x_i) \]  

(1.2)

where \( \phi_i \) are self-concordant barrier functions on \( K_i \) that blows up on \( \partial K_i \), namely, \( \phi_i(x_i) \to +\infty \) as \( x_i \to \partial K_i \). As the same as [LSZ19], we will assume all blocks are constant size and \( \phi_i \) are \( O(1) \) self-concordant.

The weight \( w \in \mathbb{R}_0^m \) are fixed throughout the algorithm and is defined according to the cost of updating block \( i \). For the sake of simplicity, in this thesis, we will not try to tune the weight \( w \), instead we will assume \( w = 1_m \). Since \( \phi_i \) blows up on \( \partial K_i \), \( x(t) \) lies in the interior of the domain for \( t > 0 \) (if the interior is non-empty). Also, by the definition of \( x(t) \), \( x(0) \) is a minimizer of the problem Eq. (4.1). In Section 3, we show how to find the initial point \( x(1) \) quickly by reformulating the problem into an equivalent form.

The key difficulty is to follow the path \( x(t) \) efficiently. To lower the cost of each step, we maintain our \( (x, s) \) implicitly. Throughout the algorithm, we can only directly access an approximate of \( (x, s) \), which called \( (\tilde{x}, \tilde{s}) \). Our algorithm takes \( \tilde{O}(\sqrt{m}) \) steps and each step involves solving some linear system very crudely according to \( (x, s) \). Then we call data structure that implicitly maintains \( (x, s) \) (Section 5). At the same time, we call a separate data structure that maintains the approximation \( (\tilde{x}, \tilde{s}) \) via some random sampling scheme (Section 5).

Our algorithm and proof in this section is similar to [LSZ19; CLS19], which develop a robust interior point methods for linear programs and empirical risk minimization. The main difference is that we choose the weight such that the expensive-to-update block changes slower. For completeness, we will include the motivation and proofs. Following idea in [LS19], we use a soft-max-like function on the optimality condition as the potential and prove that each step we make enough decrease of the potential. To motivate our potential, we consider the optimality condition of Eq. (4.2)

\[ s/t + \nabla \phi(x) = \mu, \]
\[ Ax = b, \]
\[ A^T y + s = c \]

where \( \mu = 0 \) and \( \phi(x) = \sum_{i=1}^{m} w_i \phi_i(x) \). Then, we define our potential as

\[ \Phi^t(x, s) = \sum_{i=1}^{m} \cosh \left( \frac{\lambda}{w_i} \gamma_i^t(x) \right) \]

and maintain the poten \( \Phi^t(x, s) \leq O(m) \). Now the \( \ell_\infty \) pertubation of \( \gamma \) translates to a small multiplicative change to \( \Phi^t \).

1.1.2 Cholesky Decomposition

In the interior point method, we will need to maintain the orthogonal projection matrix

\[ P_x = H_x^{-1/2} A^T (A H_x^{-1} A^T)^{-1} A H_x^{-1/2}, \]

where \( H_x^{-1} \) is a block-diagonal positive definite matrix and its \( i \)-th block is the Hessian of the self-concordant barrier function \( \nabla^2 \phi_i(x) \). The projection matrix \( P_x \) is used in every iteration to compute the next central path parameter \( (x, s) \). However, updating \( P_x \) according to the newly computed parameter \( (x, s) \), as well as multiplying it with other matrices, are both costly operations. To improve the run-time, we exploit the fact that \( A H_x^{-1} A^T \) in the expression for \( P_x \) is a symmetric, positive-definite matrix and \( A H_x^{-1} A^T \) has the same underlying graph as \( AA^T \).

Any symmetric, positive-definite matrix \( M \) admits a unique Cholesky factorization \( M = LL^T \), where \( L \) is a lower-triangular matrix with real and positive diagonal entries. The canonical Cholesky factorization

4
algorithm can be viewed as a modified symmetric Gaussian elimination. We hope to make use of this factorization of \( M \defeq AH^{-1}A^\top \); however, this may not reduce the cost of matrix operations in the case that \( L \) is dense, since for the most of matrix operation, the cost is dependent on the column sparsity of \( L \), which is equivalent height of the corresponding vertex elimination tree. A well-known technique from numerical analysis [Dav06] is to permute the rows and columns of \( M \) so that the Cholesky factorization \( PMP^\top = PLL^\top P^\top \) has some desired sparsity property that will help speed up computations. [Bod+95] shows that the minimum elimination tree height of matrix \( A \) is equivalent to treewidth of \( A \) up to a logarithmic factor. We compute the nearly optimal ordering matrix \( P \) as well as the corresponding elimination tree via finding the balanced vertex separator on \( G_A \) recursively.

## 2 Projection Matrix Maintenance via Cholesky Decomposition

In the interior point method, we will need to maintain a stochastic version of the orthogonal projection matrix

\[
P_x = H_x^{-1/2} A^\top (AH^{-1}A^\top)^{-1} AH_x^{-1/2},
\]

which is used in every iteration to compute the next point of the central path. However, updating \( P_x \), as well as multiplying it with other matrices and vectors, are both costly operations. To improve the run-time, we may assume without loss of generality that the rows of \( A \) are linearly independent, and exploit the fact that \( AH_x^{-1}A^\top \) in the expression for \( P_x \) is a symmetric, positive-definite matrix.

In general, \( H_x^{-1} \) is a block-diagonal positive definite matrix, where block \( i \) is the Hessian of \( \phi_i \) of constant size. To simplify the presentation of the combinatorial structures, we assume \( H_x^{-1} \) is a diagonal positive definite matrix; this assumption is removed in Remark 2.28. When \( H_x^{-1} \) is diagonal, observe that \( AH_x^{-1/2} \) is simply a rescaling of the columns of \( A \), and therefore has the same non-zero pattern as \( A \). Because all the analyses depend only on the non-zero pattern, we may consider \( AA^\top \) instead of \( AH_x^{-1}A^\top \) in the remainder of this section without loss of generality.

### 2.1 Cholesky Factorization

Any positive-definite matrix \( M \) admits a unique \textit{Cholesky factorization} \( M = LL^\top \), where \( L \) is a lower-triangular matrix with real and positive diagonal entries. The canonical Cholesky factorization algorithm can be viewed as a modified symmetric Gaussian elimination. We hope to make use of this factorization of \( M \defeq AA^\top \); however, this may not reduce the cost of matrix operations in the case that \( L \) is dense. A well-known technique from numerical analysis [Dav06] is to permute the rows and columns of \( M \), so that the Cholesky factorization \( PMP^\top = PLL^\top P^\top \) has some desired sparsity property that will help speed up computations. Note that this simply corresponds to permuting the rows of the constraint matrix \( A \) in the initial convex program.

Before stating the main result of this section, we introduce the necessary definitions. The application of these concepts in numerical analysis is well-studied. Recall the initial convex program has constraints given by \( Ax = b \), for some \( A \in \mathbb{R}^{d \times n} \). Let the rows of \( A \) be labelled \( 1, 2, \ldots, d \).

**Definition 2.1.** The \textit{support graph} of \( A \) is the graph \( G(A) = (V, E) \) with \( V = \{v_1, \ldots, v_d\} \), and \( v_i v_j \in E \) if and only if row \( i \) and row \( j \) of \( A \) have non-empty intersection in their support. Equivalently, the non-zero pattern of \( AA^\top \) (ignoring numerical cancellation) is precisely the adjacency matrix of \( G(A) \).

**Definition 2.2.** A \textit{tree-decomposition} of a graph \( G \) is a pair \((X, T)\), where \( T \) is a tree, and \( X : V(T) \mapsto 2^{V(G)} \) is a family of subsets of \( V(G) \) labelling the vertices of \( T \), such that

1. for each \( v \in V(G) \), the nodes \( t \in V(T) \) with \( v \in X(t) \) induces a connected subgraph of \( T \), and
2. for each edge \( e = uv \in V(G) \), there is a node \( t \in V(T) \) such that \( u, v \in X(t) \).

The \textit{width} of a tree-decomposition \((X, T)\) is \( \max\{|X(t)| - 1 : t \in T\} \). The \textit{tree-width} of \( G \) is the minimum width over all tree-decompositions of \( G \).

The main result of this section is the following:
Theorem 2.3. Let $A$ be an $d \times n$ matrix. We can compute a permutation $P$ of the rows of $A$ (equivalently, an ordering $\pi : [d] \mapsto [d]$), so that in the Cholesky factorization $PAA^\top P^\top = LL^\top$, the column sparsity of $L$ and $L^{-1}$ are both bounded by $\tilde{O}(\tw(A))$, where $\tw(A)$ is the treewidth of $G(A)$. The permutation can be computed in time $\tilde{O}(d \cdot \tw(A)^2)$.

After reordering the rows of $A$, the Cholesky factorization can be computed using $O(d \cdot \tw(A)^2)$ time.

Proving Theorem 2.3 requires a number of concepts that may be unfamiliar to the reader. We begin by presenting them and their basic properties in the subsections below.

### 2.1.1 Support Graph and Tree-Width

The following structural results about tree-width are elementary.

**Lemma 2.4.** If $G$ is a graph on $n$ vertices and $\tw(G) = \tau$, then $|E(G)| \leq n\tau$.

**Lemma 2.5.** If $G'$ is a subgraph of $G$, then $\tw(G') \leq \tw(G)$.

**Lemma 2.6.** $\tw(K_n) = n - 1$.

There are some basic relations between the sparsity of a matrix $A$ and the tree-width of its support graph.

**Lemma 2.7.** For any matrix $A$, a column of the matrix with sparsity $t$ induces a clique of size $t$ in $G(A)$. It follows that $\max\{\nnz(A_i) : A_i \text{ a column of } A\} \leq \tw(A) - 1$.

Tree-width is a natural structural parameter of a graph, with close connections to graph algorithms of a recursive nature; we are particularly interested in its connection to vertex separators.

### 2.1.2 Balanced Vertex Separator

**Definition 2.8.** Let $G = (V,E)$ be a graph. For any $W \subseteq V$, an $c$-balanced vertex separator of $W$ is a set $S \subseteq V$ of vertices such that every connected component of the graph $G[V - S]$ contains at most $c \cdot |W|$ vertices of $W$. In the particular case when $W = V$, we call the separator an $c$-balanced vertex separator of $G$. The separator number of $G$ is the maximum over all subsets $W$ of $V$ of the size of the smallest $1/2$-balanced vertex separator of $W$ in $G$.

Similar to tree-width, separator numbers are monotone.

**Lemma 2.9.** Let $G'$ be a subgraph of $G$. For any constant $0 < c < 1$, the size of the smallest $c$-vertex separator of $G'$ is at most that of $G$.

The following theorem relates the tree-width of a graph and the separator number. Indeed, tree-decomposition is a generalized notion of vertex separator.

**Theorem 2.10** ([Bod+95]). If $G$ is a graph with tree-width $\tau$, then there exists a $1/2$-balanced separator of $G$ of size at most $\tau + 1$.

Now we return to ideas for computing the permutation matrix before Cholesky factorization.

### 2.1.3 Elimination Tree

Let $G = (V,E)$ be the support graph of $A$. Let $\pi : V \mapsto [d]$ be an ordering of the vertices of $G$, which we will call an elimination order. We say a vertex $v \in V$ is eliminated at step $\pi(v)$. The filled graph of $G$ corresponding to $\pi$, denoted by $G^+_{\pi}$, is constructed as follows:
Algorithm 1 Construct $G^+_\pi$

$G^+_\pi \leftarrow (V, E)$

for $i$ from 1 to $n$ do

for each $v \in V$ such that $\pi(v) > i$ do

if $\exists$ a path $P$ from $\pi^{-1}(i)$ to $v$ in $G$, and all $u \in P - v$ satisfies $\pi(u) \leq i$ then

add edge $\pi^{-1}(i)v$ to $G^+_\pi$

end if

end for

end for

return $G^+_\pi$

This construction of $G^+_\pi$ is also known as the elimination game on $G$, which intuitively models the canonical Cholesky factorization algorithm on $PAA^T P^T = LL^T$, where $P$ is the permutation matrix for $\pi$: Indeed, eliminating the vertex $\pi^{-1}(i)$ at the $i$-th iteration of the elimination game can be viewed as moving the $\pi^{-1}(i)$-th row of $A$ to the $i$-th row in the factorization algorithm, and adding edges $\pi^{-1}(i)v$ for the specified vertices $v \in V$ indicates that the $vi$-th entry of $A$ is non-zero in the factorization algorithm. It turns out the adjacency matrix of the filled graph $G^+_\pi$ precisely gives the nonzero structure of the triangular factor $L$. Hence, our goal is to choose $\pi$ to decrease the number of edges in $G^+_\pi$.

Formally, $u, v \in V(G^+_\pi)$ are adjacent if and only if there is a path $P$ from $u$ to $v$ in $G$, such that all interior vertices $w$ on $P$ satisfies $\pi(w) < \min\{\pi(u), \pi(v)\}$.

The elimination tree corresponding to $\pi$ is the tree $T$ defined by the following parent-children relation: For a vertex $v \in V$, its parent is $\arg \min \{\pi(w) : w \in \mathcal{N}_{G^+_\pi}(v), \pi(w) > \pi(v)\}$; in words, it is the vertex $w$ that is eliminated earliest after $v$, that is reachable from $v$ in $G$ using a path whose interior vertices are all eliminated before $v$. Different elimination orders give rise to different elimination trees. The height of the shortest elimination tree over all choices of $\pi$ is the minimum etree height.

When the rows of $A$ are reordered according to $\pi$, the elimination tree reflects the non-zero pattern in the Cholesky factor.

Lemma 2.11 ([Sch82]). Let $L$ be the Cholesky factor for the matrix $AA^\top$. Let $L_j$ denote the $j$-th column. The nonzero pattern of $L_j$ is a subset of the vertices on the path from $j$ to the root in the elimination tree corresponding to the identity permutation.

Example 2.12. The figure below shows the relationship between a matrix, its Cholesky factor, and the corresponding elimination tree. On the left is a $10 \times 10$ matrix $AA^\top$, with rows labelled $\{1, \ldots, 10\}$. In the middle is the Cholesky factor $L$ of $AA^\top$. On the right is the elimination tree, where node $i$ in the tree corresponds to row $i$ of the matrices $AA^\top$ and $L$.

Figure 2.1: Each blue dot represents a non-zero entry in the matrix.
Lemma 2.13. If \( uv \) is an edge in \( G \), then in any elimination tree \( T \) of \( G \), there is an ancestor-descendant relationship between \( u \) and \( w \). It follows that if \( K \) is a clique in \( G \), then in any elimination tree \( T \) of \( G \), the vertices of \( K \) all lie on the same path from some leaf of \( T \) to the root.

2.2 Computing an Elimination Tree

The various parameters presented above are related by the following result:

**Theorem 2.14 ([Bod+95]).** Every graph \( G \) on \( n \) vertices satisfies

\[
\text{separator number} < \text{treewidth} \leq \min \text{ etree height} \leq \text{separator number} \cdot \log n.
\]

This structural theorem indicates that it is possible to construct an elimination tree \( T \) of \( G(A) \), and bound its height as a function of \( tw(A) \). Specifically, we will recursively compute vertex separators and use them to generate an ordering \( \pi \) of the vertices of \( V(G) \), which then uniquely determines the elimination tree \( T \).

**Theorem 2.15.** Let \( G \) be a graph on \( n \) vertices with tree-width \( \tau \). Suppose \( \text{approxBalancedSeparator} \) is an algorithm that, given a graph \( H \) on \( k \) vertices, computes \((H_1, S, H_2)\) where

1. \( H_1, H_2 \subseteq H \) are subgraphs of \( H \), and \( V(H_1) \cup S \cup V(H_2) = V(H) \),
2. \( S \) is an \( c' \)-balanced vertex separator of \( H \) for some \( c' > 1/2 \), and \( |S| \leq O(\log k \cdot |S^{opt}|) \), where \( S^{opt} \) is the optimal 2/3-balanced vertex separator,
3. the algorithm runs in time \( T_{\text{sep}}(k) = \tilde{O}(k\tau^2) \).

Then we can construct an elimination tree for \( G \) of height at most \( O(\tau \log_{1/c'} n) \), in time \( \tilde{O}(n \tau^2) \).

*Proof.* we will use a list notation \((v_1, \ldots, v_k)\) to denote the ordering \( \pi \) of a subset of vertices \( \{v_1, \ldots, v_k\} \) with \( \pi(v_i) = i \). We use \( + \) to denote the concatenation of two lists.

We have the following straightforward analysis of \( \text{makeElimOrder} \): Let \( T(k) \) denote the run-time on a graph with \( k \) vertices. Then

\[
\begin{align*}
T(k) &= O(1) & k \leq \tau \\
T(k) &\leq T(c'k) + T((1-c')k) + T_{\text{sep}}(k) & k > \tau
\end{align*}
\]

Solving the recurrence, we have \( T(n) = \tilde{O}(n\tau^2) \).

To understand the height of the elimination tree, we make the crucial observation that, at every recursive step \( \text{makeElimOrder}(H) \), the subroutine \( \text{approxBalancedSeparator}(H) \) will return \((H_1, S, H_2)\), such that in the original graph \( G \), vertices in \( H_1 \) are only connected to vertices in \( H_2 \) via a path containing vertices in \( S \). Hence, vertices in \( H_1 \) have no ancestors in \( H_2 \), and vice versa.

In total, \( \text{makeElimOrder} \) recurses to a depth of \( O(\log_{1/c'} n) \), and at each recursive iteration, the contribution to the elimination tree height is bounded by the size of the separator \( |S| \) computed in the iteration. Since \( G \) has tree-width \( \tau \), all subgraphs of \( G \) has 1/2-balanced vertex separators of size at most \( \tau \); 1/2-balanced vertex separators are trivially 2/3-balanced. Therefore, the minimal 2/3-balanced vertex separator in any recursive iteration of \( \text{makeElimOrder} \) has size at most \( \tau \), and we get the desired tree height.

After computing the elimination order, it is a fairly straightforward algorithms exercise to construct the elimination tree quickly; we include the pseudocode for completeness. The main idea here is that at iteration \( i \), each set represents a connected component in \( G(\{v: \pi(v) < i\}) \), and the tag on a set represents the unique element in the component without a parent in \( T \) yet. The full explanation is omitted. \( \text{makeElimTree} \) has run-time

\[
\sum_{v \in V(G)} \sum_{w \in N(v)} O(\alpha(n)) \leq 2|E(G)|O(\log n) \leq \tilde{O}(n\tau),
\]

where \( \alpha(n) \) is the inverse-Ackermann function. \( \square \)
Algorithm 2 Constructing an Elimination Tree

1: procedure makeElimOrder($G$)
2: if $|V(G)| \leq \tau$ then
3: return arbitrary ordering of $V(G)$
4: end if
5: $(G_1, S, G_2) \leftarrow$ approxBalancedSeparator($G$)
6: $L_1 \leftarrow$ makeElimOrder($G_1$)
7: $L_2 \leftarrow$ makeElimOrder($G_2$)
8: $L \leftarrow$ arbitrary ordering of $S$
9: return $L_1 + L_2 + L$
10: end procedure

11: procedure makeElimTree($G, \pi$)
12: // We use union-find with an additional vertex tag on sets, as follows:
13: // MakeSet($v, x$) makes the singleton set with element $v$ and tag $x$
14: // Union($u, v, x$) takes the union of the sets containing $u$ and $v$, and tags the resulting set with $x$
15: // Find($v$) returns $(v_s, x)$ where $v_s$ is the representative of the set containing $v$ with tag $x$
16: for $i = 1$ to $n$ do
17: $v \leftarrow \pi^{-1}(i)$
18: MakeSet($v, v$)
19: for $w \in N(v)$ do
20: $(w_s, x) \leftarrow$ Find($w$)
21: $(v_s, y) \leftarrow$ Find($v$)
22: if $w_s \neq v_s$ and $w_s \neq$ null then
23: Set $x$'s parent to be $v$ in $T$
24: Union($w_s, v_s, v$)
25: end if
26: end for
27: end for
28: end procedure
2.2.1 Approximating a Balanced Separator

We present approxBalancedSeparator as required by Algorithm 2.

**Theorem 2.16.** Let $G = (V, E)$ be a graph on $n$ vertices with tree-width $\tau$. Suppose $S$ is a minimum $c$-balanced vertex separator of $G$ for some $c > 1/2$. Then we can compute a $c'$-balanced vertex separator $S'$ for some constant $c'$, such that $|S'| \leq O(\log n)|S|$ in $\tilde{O}(n\tau^2)$ time.

**Proof.** To begin, note that by Lemma 2.4, $|E| \leq n\tau$.

We use a standard technique [LR99] to reduce UNDIRECTED BALANCED VERTEX SEPARATOR to DIRECTED BALANCED EDGE SEPARATOR:

Without loss of generality, we may assume a $c$-balanced vertex separator in $G$ is represented by $(A, S, B)$, where $A \cup S \cup B = V(G)$, and $S$ is the vertex separator that disconnects vertices between $A$ and $B$, where $|A|, |B| \leq c|V(G)|$.

Let $G'$ be a weighted, directed graph, where $V(G') = V(G) \cup \{v : v \in V(G)\}$, and $E(G') = \{(v, v') : v \in V(G)\} \cup \{(u', v) : (u, v) \in E(G)\} \cup \{(v', u) : (u, v) \in E(G)\}$. Note that $|V(G')| = 2n$ and $|E(G')| = O(n\tau)$. Weights are assigned to edges so that $w(v, v') = 1$, and $w(u, v) = \infty$ for all other edges.

For any set $T \subseteq V(G)$, define the notation $T' = \{v : v \in T\} \subseteq V(G')$.

For a cut $(S, \bar{S})$ in $G'$, define $E(S, \bar{S}) = \delta^-(S)$ to be the edges going from $S$ to $\bar{S}$. For any $c > 1/2$, the cut is $c$-balanced if $|S|, |\bar{S}| \leq c|V(G')|$. The minimum $c$-balanced edge separator is the $c$-balanced cut $(S, \bar{S})$ that minimizes $w(E(S, \bar{S})) = |E(S, \bar{S})|$.

Given the weight assignments, any minimum $c$-balanced edge separator $(S, \bar{S})$ in $G'$ only contains edges of the form $(v, v')$ for $v \in G$. Then the $\bar{S} = \{v \in G : (v, v') \in E(S, \bar{S})\}$ is an $c$-balanced vertex separator in $G$: Indeed, let $A = \{v \in G : v \in \bar{S} \setminus \bar{S}\}$ and $B = \{v \in G : v \in \bar{S} \setminus \bar{S}\}$. Since $|S|, |\bar{S}| \leq c \cdot 2n$, we have $|A| = 1/2(|S| - |\bar{S}|) \leq c \cdot n$, and similarly for $B$.

Conversely, any minimum $c$-balanced vertex separator $(A, S, B)$ in $G$ gives a $c$-balanced edge separator in $G'$ with the same size: Indeed, let $\bar{A} = A \cup A' \cup S$ and $\bar{B} = B \cup B' \cup S'$, then $(\bar{A}, \bar{B})$ is a cut in $G'$, with $E(\bar{A}, \bar{B}) = \{(v, v') : v \in S\}$. To see this cut is $c$-balanced, note that $|\bar{A}| = 2|A| + |S| \leq 2cn + (1-c)n = (1+c)n = \frac{1+c}{2} \cdot (2n) < c \cdot |V(G')|$, and similarly for $|\bar{B}|$.

The expansion of a cut $E(S, \bar{S})$ is defined as $\frac{E(S, \bar{S})}{\min(|S|, |\bar{S}|)}$. A $t$ pseudo-approximation to our problem is a $c'$-balanced cut for some constant $c' > c$, whose expansion is within a factor $t$ of the optimal $c$-balanced cut. In turn, the size of the cut is also within a factor $t$ of the optimal $c$-balanced cut.

**Theorem 2.17 ([AK16]).** Given a directed graph $G'$ on $n$ vertices and $m$ edges, we can compute an $O(\log n)$ pseudo-approximation to the minimum $c$-balanced edge separator in $\tilde{O}(m^{1.5})$ time using polylog($n$) single-commodity flow computations.

This is not directly applicable to our setting. We modify the algorithm in the proof of Theorem 2.17 to get the desired running time. The proof uses the following lemma:

**Lemma 2.18.** Let $S \subseteq V$ be a set of nodes of size $\Omega(n)$. Suppose we are given, for all $i \in S$, vectors $v_i, w_i$ of length $O(1)$, such that $\sum_{i,j \in S} ||v_i - v_j||^2 \leq o(1)$. Define the directed distance $d(i, j) = ||v_i - v_j||^2 - ||w_i - v_i||^2 + ||w_i - v_j||^2$. For any given value $\alpha$, there is an algorithm which, using a single max-flow computation, either outputs a valid $O(\log n)$-regular directed flow $f_p$ such that $\sum_{ij} d(i, j) \geq \alpha$, or a $c'$-balanced cut of expansion $O(\log(n)^2/n^2)$.

The max-flow algorithm used in the above lemma is the Goldberg and Rao algorithm in time $\tilde{O}(m^{1.5})$, which is the bottleneck of the overall time to compute the edge separator. Instead, we use the standard
Ford-Fulkerson max flow algorithm that runs in \(O(m \cdot F)\) time, where \(F\) is the value of the maximum flow on the graph. Note that \(m \leq O(n\tau)\), and we claim \(F = O(\tau)\).

Indeed, by max-flow-min-cut, if \(F = \omega(\tau)\), then the minimum cut in \(G'\) would also have value \(\omega(\tau)\). Recall our reduction from vertex-separator to directed edge separator: a cut in \(G'\) of value \(\omega(\tau)\) corresponds to a vertex separator in the original graph \(G\) of value \(\omega(\tau)\). However, since \(G\) has treewidth \(\tau\), this is a contradiction.

\[\square\]

2.3 Sparsity Patterns in \(A\) and \(L\), and Associated Matrix Operation Costs

Let \(A \in \mathbb{R}^{d \times n}\). Let \(L\) be the Cholesky factor of \(M := AA^\top\), and let \(T\) be the elimination tree. For any matrix \(M\), we use \(M_i\) to denote the \(i\)-th column, and \(M_i\) to denote the non-zero pattern of the \(i\)-th column (i.e. it is a set of row indices). We use \(M_i\) to denote the \(i\)-th row of \(M\).

**Lemma 2.19.** If \(tw(A) = \tau\), then \(nnz(A) \leq n\tau\).

**Lemma 2.20 (Sch82).** The height of the elimination tree satisfies \(h(T) \geq \max\{nnz(L_i) : i \in [n]\}\).

**Lemma 2.21.** The column sparsity of \(L^{-1}\) is at most \(h(T) + 1\).

*Proof.* Recall the elementary way to compute a matrix inverse:

\textbf{Algorithm 3 Compute} \(L^{-1}\)

1: \(M = [L | I_{n \times n}]\)
2: \(\text{for } i = 1 \text{ to } n \text{ do}\)
3: \(M^i \leftarrow M^i / l_{ii}\)
4: \(\text{for } k = i + 1 \text{ to } n \text{ do}\)
5: \(M^k \leftarrow M^k - l_{ki} M^i\)
6: \(\text{end for}\)
7: \(\text{return right half of } M\)

\(L_j^{-1}\) denotes the non-zero pattern of the \(j\)-th column of \(L^{-1}\). Let \(L_j^{-1}(i)\) denote the non-zero pattern of the \(j\)-th column at the end of iteration \(i\) in the above algorithm, so that \(L_j^{-1}(1) \subseteq L_j^{-1}(2) \subseteq \cdots \subseteq L_j^{-1}(n) = L_j^{-1}\).

First, note that \(L_j^{-1}(1) = \cdots = L_j^{-1}(j - 1) = \{j\}\). After the \(j\)-th iteration, \(L_j^{-1}(j) = L_j\), since every row \(M^k\) with \(k > j\) gets updated by adding a multiple of \(M^j\) (which is non-zero in the \(j\)-th column). In a subsequent iteration \(k > j\), we only update a row \(k'\) if \(k' > k\) and \(l_{k'k} \neq 0\), by adding a multiple of \(M^k\) to them. Hence, if \(k \notin L_j^{-1}(k - 1)\), then \(L_j^{-1}(k) = L_j^{-1}(k - 1)\). On the other hand, if \(k \in L_j^{-1}(k - 1)\), then \(L_j^{-1}(k) = L_j^{-1}(k - 1) \cup L_k\).

This gives us a way to characterize precisely the construction of \(L_j^{-1}\), which is the following process:

\textbf{Algorithm 4 Make} \(L_j^{-1}\)

1: \(L_j^{-1} \leftarrow L_j\)
2: \(i \leftarrow j + 1\)
3: \(\text{while } i \leq n \text{ do}\)
4: \(k \leftarrow \min\{k \in L_j^{-1} : k \geq i\}\)
5: \(L_j^{-1} \leftarrow L_j^{-1} \cup L_k\)
6: \(i \leftarrow k + 1\)
7: \(\text{end while}\)
Let $P_j$ denote the path from $j$ to the root of $T$, and recall the property that $L_j \subseteq P(j)$. We begin with $L_j^{-1} = \mathcal{L}_j$. Next, observe that, we take some $k \neq j \in L_j$ and add $L_k$ to our set, but $k \in V(P_j)$, and $L_k \subseteq V(P_k)$. However, note that $P_k \subseteq P_j$. This process continues recursively. Hence, $\mathcal{L}_j^{-1} \subseteq V(P_j) \leq h(T) + 1$.

**Lemma 2.22.** Given the elimination tree for $A$ with height $\tau$, for any $v$, we can solve for $L^{-1}v$ in $O(\tau^2 \|v\|_0)$ time.

*Proof.* The proof is similar in spirit to the previous. We solve for $x$ such that $Lx = v$:

**Algorithm 5** Elementary algorithm for solving $Lx = v$

```plaintext
1: $x \leftarrow 0$
2: for increasing $j$ with $v_j \neq 0$ do
3: $x_j \leftarrow v_j / L_{jj}$
4: $v \leftarrow v - x_jL_j$
5: end for
6: return $x$
```

Since $L$ has column density $\tau$, each iteration of the for loop takes $O(\tau)$ time. In the $j$-th iteration, line 4 changes up to $\tau$ entries of $v$ from zero to non-zero; specifically for $k > j$, the $k$-th entry is changed if and only if $k$ is an ancestor of $j$ in $T$. In this case, note that in the $k$-th iteration, $v_l$ is changed if and only if $l$ is an ancestor of $k$, but $v_l$ is already non-zero after the $j$-th iteration.

It follows that the for loop iterates through $O(\tau \cdot \|v\|_0)$ entries of $v$ in total, and the overall run-time is $O(\tau^2 \|v\|_0)$.

**Lemma 2.23.** Given the elimination tree for $A$ with height $\tau$, for any $v$, we can solve for $(L^{-T}v)_i$ in time $O(\tau^2)$.

*Proof.* Recall $L^T$ has row density $\tau$. We are solving for $L^T x = v$, and interested in $x_i$.

In the $i$-th row, we have

$$L_{ii}^Tx_i + \cdots + L_{ik}^Tx_k = v_i, \quad (2.1)$$

where the LHS contains at most $\tau$ terms; specifically, $x_j$ appears in the equation if and only if $j$ is an ancestor of $x_i$ in $T$.

We can solve for $x_i$ if the values for all other $x_j$ for $x_j$ appearing in Equation 2.1 are known; the computation takes $O(\tau)$ time. If $P$ is the path from the root of $T$ to $i$, then we solve for the $x_j$’s along the path in order, from the root to $x_i$. In total, this takes $O(\tau^2)$ time.

**Lemma 2.24.** $\|L^{-1}Ae_j\|_0 \leq h(T) + 1$.

*Proof.* We have $L^{-1}Ae_j = \sum_{i \in A_j} A_{ij} \cdot L^{-1}_i$. Note that $A_i$ identifies a clique in $G(A)$, so by Lemma 2.13, these vertices all appear in the same path from some leaf to the root in $T$. By an argument similar to that in Lemma 2.21, we conclude the number of non-zero entries is at most $h(T) + 1$.

### 2.4 Maintaining the Cholesky Factorization

**Lemma 2.25 ([Dav06]).** For a positive definite matrix $M$, we can compute its Cholesky factorization $M = LL^T$ in time

$$\sum_{j=1}^{n} |L_j|^2,$$

where $|L_j|$ denotes the number of nonzero entries in the $j$th column of $L$. 

---

12
**Theorem 2.26.** Given a positive definite matrix $M \in \mathbb{R}^{n \times n}$ and its elimination tree $T$ of height $\tau$, Cholesky factorization of $M$ can be performed using $O(n \tau^2)$ time.

**Proof.** By the definition of tree height, for any vertex $v$ in the tree, the length of the path from $v$ to root is less than $\tau$. Then Lemma 2.11 implies $|L_j| \leq \tau$ for all $j$. By Lemma 2.25, we have the time of Cholesky factorization is bounded by $O(n \tau^2)$.

**Theorem 2.27.** Let $A$ be any $d \times n$ matrix, and let $D$ be an $n \times n$ diagonal matrix with only positive entries. Suppose $ADA^\top$ has Cholesky factorization $LL^\top$, and the corresponding elimination tree $T$ has height $\tau$. Then for any $\lambda$ with $\lambda > -D_{ii}$, we can compute the Cholesky factorization and elimination tree of $A(D + \lambda e_i e_i^\top)A^\top$ in $O(\tau^2)$ time. Furthermore, the resulting elimination tree still has height $\tau$.

**Proof.** First, note that since $ADA^\top$ and $A(D + \lambda e_i e_i^\top)A^\top$ have the same non-zero pattern, their Cholesky factors will also have the same non-zero pattern, and their corresponding elimination trees will be identical. Only certain entries in the Cholesky factors will differ. By Lemma 4.2 of [DH03], the time to accomplish this is $O(\tau^2)$.

Finally in this section, we resolve the initial assumption that $H^{-1}$ is a diagonal matrix.

**Remark 2.28.** Suppose $H^{-1}$ is block-diagonal with $m$ total blocks, where block $i$ is the Hessian of $\phi_i$, with constant size. To maintain $AH^{-1}A^\top$ efficiently using the ideas presented in this section, we make the following modifications:

The columns of $A$ are partitioned into $m$ block-columns, each corresponding to one block of $H^{-1}$. For the support graph $G(A) = (V, E)$, the vertex set remains the same, with each vertex corresponding to a row of $A$. We have $ij \in E$ if and only if there exists a block-column in which one entry in the $i$-th row and one in the $j$-th row are non-zero.

$tw(A)$ is the tree-width of the resulting support graph.

In all the matrix operations and sparsity arguments, we could carry out the computations within a block-column in any elementary way; since the blocks have constant size, the run-time is not affected asymptotically. Between blocks, we follow the arguments as presented in this section, which all hold in this generalized setting.

### 3 Initial Point Reduction

Our problem begins with a generic convex program

$$\min c^\top x \quad s.t. \quad Ax = b, \quad x \in \prod_{i=1}^m K_i. \quad (CP)$$

To generate an initial feasible point for the central path, the following reduction is required.

**Theorem 3.1** (ERM paper). Consider a convex problem $\min c^\top x \ st. \ Ax = b, \ x \in \prod_{i=1}^m K_i$ where $A \in \mathbb{R}^{d \times n}$. For each $i \in [m]$, we are given a $\nu_i$-self concordant barrier function $\phi_i$ for $K_i$. Also, we are given $x^{(0)} = \arg \min_x \sum \phi_i(x_i)$, where $x_i \in K_i$. Assume that

1. Diameter of the polytope: For any $x \in \prod_{i=1}^m K_i$, we have $\|x\| \leq R$.
2. Lipschitz constant of the program: $\|c\|_{\infty} \leq L$.

Then, for any $\delta > 0$, the modified convex program

$$\min \quad \bar{c}^\top \bar{x} \quad s.t. \quad \bar{A}x = \bar{b}, \quad \bar{x} \in \prod_{i=1}^m K_i \times \mathbb{R}_+ \quad (\bar{CP})$$
with
\[ \tilde{A} = [A | b - Ax^{(0)}], \] \[ \tilde{b} = b, \] \[ \text{and} \quad \tilde{c} = \left[ \frac{\delta}{L^R} \cdot c \right], \]

satisfies the following:

1. \( \tilde{x} = \begin{bmatrix} x^{(0)} \\ 1 \end{bmatrix} \) and \( \tilde{y} = 0_d \) are feasible primal and dual vectors, with slack \( \tilde{s} = \left[ \frac{\delta}{L^R} \cdot c \right], \) and \( \|\tilde{s} + \nabla \tilde{\phi}(\tilde{x})\|_\infty \leq \delta, \) where \( \tilde{\phi}(\tilde{x}) = \sum_{i=1}^m \phi_i(\tilde{x}_i) - \log(\tilde{x}_{m+1}). \)

2. For any \( \tilde{x} \) such that \( \tilde{A} \tilde{x} = \tilde{b}, \) \( \tilde{x} \in \prod_{i=1}^n K_i \times \mathbb{R}_+, \) and \( \tilde{c}^T \tilde{x} \leq \text{OPT} + \delta^2, \) the vector \( \tilde{x}_1:n \) is an approximate solution to the original convex program \( (CP) \) in the following sense:
\[
\begin{aligned}
    c^T \tilde{x}_1:n &\leq \min_{Ax = b, \ x \in \prod_{i=1}^n K_i} c^T x + LR \cdot \delta \\
    \|A \tilde{x}_1:n - b\|_1 &\leq 3\delta \cdot (R\|A\|_1 + \|b\|_1) \\
    \tilde{x}_1:n &\in \prod_{i=1}^m K_i.
\end{aligned}
\]

In the above reduction, note that \( tw(\tilde{A}) \) can become much greater than \( tw(A) \), depending on the non-zero pattern of the additional column \( b - Ax^{(0)} \) in \( \tilde{A} \). We make a further reduction and leverage Theorem 3.1 to resolve this.

**Theorem 3.2.** Let \( A, b, c, \tilde{A}, \tilde{b}, \tilde{c}, x^{(0)} \) be defined as in Theorem 3.1. Given the elimination tree \( T \) for \( A \), we can construct a modified convex program
\[
\min c'^T \begin{bmatrix} x \\ y \end{bmatrix} \quad \text{s.t.} \quad A' \begin{bmatrix} x \\ y \end{bmatrix} = b', \quad x \in \prod_{i=1}^m K_i, \quad y \in \mathbb{R}_{d'}^d, \quad (CP')
\]

based on \( T \), where \( A' \in \mathbb{R}^{O(d) \times (n+d')} \) and \( d' = O(d) \), such that

1. \( x' = \begin{bmatrix} x^{(0)} \\ 1/2 \cdot 1 \end{bmatrix} \) and \( 0 \) are feasible primal and dual vectors with slack \( s' = \left[ \frac{\delta}{L^R} \cdot c \right], \) and \( \|s' + \nabla \tilde{\phi}(x')\|_\infty \leq \delta, \) where \( \tilde{\phi}(x') = \sum_{i=1}^m \phi_i(x'_i) - \sum_j \log(y_j). \)

2. For any \( (x, y) \in \prod_{i=1}^n K_i \times \mathbb{R}_{d'}^d \) such that \( A' [x \ y]^T = b', \) and \( c'^T [x \ y]^T \leq \text{OPT} + \delta^2, \) the vector \( x \) is an approximate solution to the original convex program \( (CP) \) in the following sense:
\[
\begin{aligned}
    c^T x &\leq \min_{Ax = b, \ x \in \prod_{i=1}^n K_i} c^T x + LR \cdot \delta \\
    \|Ax\|_1 &\leq 3\delta \cdot (R\|A\|_1 + \|b\|_1) \\
    x &\in \prod_{i=1}^m K_i.
\end{aligned}
\]

3. We can construct an elimination order and tree \( T' \) for \( A' \), such that \( \text{height}(T') \leq O(\text{height}(T) \cdot \log d). \) Furthermore, if \( t_v \) is the height of vertex \( v \) in \( T \), and \( t'_i \) is the height of vertex \( i \) in \( T' \), then
\[
\sum_{v \in V(T')} t'_v^2 \leq O(\log d \cdot \sum_{v \in V(T)} t_v^2).
\]

4. Each column of \( A' \) has at most \( tw(A) \) non-zero entries.

**Proof.** Let \( (CP') \) denote the convex program we construct, with constraint matrix \( A'[x \ y]^T = b' \). There will be a natural correspondence between the rows of \( A \) and the first \( d \) rows of \( A' \). In the construction, \( y \) are new variables; we index the entries of \( y \) by some \( S \subset [d] \) rather than integers.
3.1 Algorithm

Let $G$ denote the support graph of $A$, where $V(G) = V(T) = \{v_1, \ldots, v_d\}$, and $v_i$ is the $i$-th constraint of $Ax = b$. We may assume without loss of generality that $T$ is connected.

Crucially, $A'$ will rely on the structure of $T$, so that we can bound the resulting height of $T'$. First, we give the algorithmic construction of $A'$. We use matrix notation and linear equations interchangeably.

**Algorithm 6** Construct $LP'$ with feasible initial point

1. $A' \leftarrow [A \mid D(b - Ax(0))]$  \>
   ($D(v)$ is a diagonal matrix with $v$ on the diagonal)
2. $b' \leftarrow \bar{b}$
3. for each $v_i \in V(T)$ with $k > 1$ children $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ do
4. \quad add constraints (translates to adding rows to $A'$ and entries to $b'$)
   \[
   y_{\{i_1\}} = y_{\{i_1, i_2\}}
   y_{\{i_2\}} = y_{\{i_1, i_2\}}
   \vdots
   y_{\{i_{k-1}\}} = y_{\{i_{k-1}, i_k\}}
   y_{\{i_k\}} = y_{\{i_{k-1}, i_k\}}
   y_{\{i_1, i_2\}} = y_{\{i_1, i_2, i_3, i_4\}}
   y_{\{i_1, i_4\}} = y_{\{i_1, i_2, i_3, i_4\}}
   \vdots
   y_{\{i_{1, \ldots, i_k/2}\}} = y_{\{i_{1, \ldots, i_k}\}}
   y_{\{i_{k/2+1, \ldots, i_k}\}} = y_{\{i_{1, \ldots, i_k}\}}
   y_{\{i_{1, \ldots, i_k}\}} = y_{\{i\}}
   \]
5. end for
6. $c' \leftarrow \left[ \frac{\delta}{\delta} \cdot \frac{\bar{c}}{\bar{c}} \cdot \mathbf{1} \right]$  \>
   ($\frac{\delta}{\delta} \cdot \mathbf{1}$ is the constant vector of length $d'$, where $d'$ is the number of new variables)

3.2 Algorithm Explanation

We construct $A'[x \ y]^{\top} = b'$ to be an extension of $\bar{A}x = \bar{b}$.

In lines 1-3, we concatenate $A$ with an $d \times d$ diagonal matrix $D(b - Ax(0))$ (rather than the column $b - Ax(0)$ for $A$). We use $y_{\{1\}}, \ldots, y_{\{d\}} \in \mathbb{R}^+$ to denote the new variables introduced. Compared against $\bar{A}x = \bar{b}$, we see that

$$
[A \mid b - Ax(0)] \begin{bmatrix} x \\ x_{n+1} \end{bmatrix} = \bar{b}
$$

becomes

$$
[A \mid D(b - Ax(0))] \begin{bmatrix} x \\ y_{\{1\}} \\ \vdots \\ y_{\{d\}} \end{bmatrix} = b', \text{ where } x \in \mathbb{R}^n.
$$

A solution here might not yield a solution to the original $\bar{A}x = \bar{b}$. To ensure that it does, we need to add constraints and intermediate variables to ensure $y_{\{1\}} = \cdots = y_{\{d\}}$, after which we can set $\bar{x}_{n+1}$ as that value, and $\bar{x}_i = x_i$ for all other $i$. How these constraints are added will affect the resulting elimination tree.

Lines 4-6 iterates through each vertex in $T$, and adds new constraints based on the arrangement of its children.
3.3 Correctness

First, observe that $OPT' = \bar{OPT}$ where the two are optimal values for $(CP')$ and $(\bar{CP})$ respectively.

Second, any $(x, y) \in \prod_{i=1}^{m} K_i \times \mathbb{R}^{d'}$ satisfying $A'[x \ y]^T = b'$ also satisfies $y_S = y_T$ for all valid indices $S, T \subseteq [d]$; hence, $A[x \ y]^T = b$. If on top of this, we also have $c'[x \ y]^T \leq OPT' + \delta^2$, then $(x, y_{\{1\}}) \in \prod_{i=1}^{m} K_i \times \mathbb{R}^d$ satisfies $A[x \ y_{\{1\}}]^T = b$, and $c'[x \ y_{\{1\}}]^T \leq \bar{OPT} + \delta^2$. Then by Theorem 3.1, $x$ satisfies part (2) in the theorem statement.

Each column of $A$ has at most $tw(A)$ non-zero entries, by definition of tree-width. In our modification, we note that the first $n$ columns of $A'$ are simply those of $A$ padded with zeros; the additional columns each has a constant number of non-zero entries, since each $y_S$ exists in a constant number of new constraints. Hence part (4) is satisfied.

It is easy to check that part (1) is satisfied as well by relating it Theorem 3.1.

The dimension of $A'$ is as follows: At the vertex $v \in V(T)$ with $k$ children, the construction introduces $O(k)$ new variables of the form $y_S$. Hence, we introduce $d' = O(d)$ new variables in total. Furthermore, since each $y_S$ appears in at most 3 constraints, there are $O(d)$ total new constraints. Therefore, $A' \in \mathbb{R}^{O(d) \times (n+d')}$. It remains to examine the elimination tree for $A'$.

3.4 Change from $G$ to $G'$, and from $T$ to $T'$

Recall $V(G) = V(T) = \{v_1, \ldots, v_d\}$, where vertex $v_i$ corresponds to the $i$-th row of $A$. Let $G'$ denote the support graph of $A'$. Let $p_T(v_i)$ denote the parent of $v_i$ in $T$.

The first $d$ rows of $A'$ is given by $[A \mid D(b - Ax^{(0)})]$; we label the corresponding vertices in $G'$ by $\{v_1, \ldots, v_d\}$ in order. Then, note that $G'[\{v_1, \ldots, v_d\}] \cong G$, since the concatenated diagonal matrix does not contribute any new edges.

The new constraints added in Algorithm 6 correspond to additional vertices in $G'$. We label such a constraint (vertex) by a subset $S$ where $S \subset [d]$ as follows: constraint $S$ is the unique constraint of the form $y_S = y_R$, where $R$ is a subset containing $S$. For example, in Figure 3.2, vertex $v_{\{3,4\}}$ is the constraint $y_{\{3,4\}} = y_{\{1,2,3,4\}}$.

In iteration $v_i$ of lines 4-6 of Algorithm 6, we modify vertex $v_i$ with children $v_{i_1}, \ldots, v_{i_k}$. The support graph changes as follows: new vertices $v_{\{i_1\}}, \ldots, v_{\{i_k\}}, v_{\{i_1,i_2\}}, \ldots, v_{\{i_1,\ldots,i_k\}}, v_{\{i\}}$ are added to $G'$ (if they haven’t been added already). Each new vertex $v_{\{i_j\}}$ is connected to $v_{i_j}$, since constraints $\{i_j\}$ and $i_j$ share the variables $y_{\{i_j\}}$. A pair of new constraints $y_S = y_R$ (labelled by $S$) and $y_R = y_U$ (labelled by $R$), where $S, R, U \subseteq \{i_1, \ldots, i_k\}$, adds the edge $v_Sv_R$, since constraints $S$ and $R$ share the variable $y_R$. For $S = \{i_1, \ldots, i_k\}$, constraint $S$ shares the variable $y_{\{i\}}$ with constraints $i$ and $\{i\}$, adding the edge $v_Sv_{\{i\}}$. One straightforward observation of this construction is that for any vertex $v_S$ with $S \subseteq [d]$, if $i, j \in S$, then $p_T(v_i) = p_T(v_j)$.

We illustrate this change to the support graph and elimination tree with the following example, which corresponds to one iteration of the loop in Algorithm 6, for the vertex $v_8$ with children $v_1, v_2, \ldots, v_7$ in $T$.

![Figure 3.1: The elimination tree $T$ at a single vertex, with seven children. For simplicity, we label a vertex $v_i$ by its subscript $i$.](image)

After new constraints are added in the current iteration, $G'$ locally looks like the following:
We will present an elimination order of the constraints, so that $T'$ has the following desired structure:

1. $p_T(v_i)$ is an ancestor of $v_i$ in $T'$ for all $i \in \{1, \ldots, d\}$,
2. $v_{i(j)}$ is the parent of $v_i$ for all $i \in \{1, \ldots, d\}$,
3. for all triples of vertices $v_S, v_T, v_{S \cup T}$, where $S, T \subseteq [d]$, we have that $v_{S \cup T}$ is the parent of $v_S$ and $v_T$,
4. for a vertex $v_S$ with $S \subseteq [d]$, if there does not exist another vertex $v_R$ such that $R \supset S$, then the parent of $v_S$ is $p_T(v_i)$ for any $i \in S$.

Intuitively, a vertex $v$ with $k$ children in $T$ is transformed into a vertex $v$ at the root of a balanced binary tree with $k$ leaves in $T'$. Below is an illustration:

If we can produce the elimination tree described above, then it is straightforward to see that $\text{height}(T') \leq O(\text{height}(T) \cdot \log d)$, since at every level of $T$, our construction expands the height by a factor of at most $\log d$. Moreover, let $P(v_i)$ be the vertices between $v_i$ and $p_T(v_i)$ in $T'$. Then, if $t_i$ is the height of vertex $v_i$
in $T$ and $t'_v$ is the height of vertex $v$ in $T'$, we have

$$\sum_{v \in V(T')} t'^2_v = \sum_{i \in \{1, \ldots, d\}} \left( t'^2_{v_i} + \sum_{S \in P(v_i)} t'^2_{v_S} \right) \leq \sum_{i \in \{1, \ldots, d\}} (t_i \log d)^2 + \log d \cdot (t_i \log d)^2 \leq \log^4 d \cdot \sum_{i \in \{1, \ldots, d\}} t'^2_i.$$ 

Hence, part (3) of the theorem statement would be satisfied.

### 3.5 Updated Elimination Order

Finally we present the elimination order for $G'$ that yields the elimination tree $T'$ as discussed above.

Without loss of generality, we may assume the elimination order for $G$ is the sequence $v_1, \ldots, v_d$. We will insert the remaining vertices into this sequence to arrive at the final ordering for $G'$. The rules are as follows:

1. vertex $v_{(i)}$ is eliminated immediately after vertex $v_i$ for each $i \in [d]$, and
2. for any three vertices $v_S, v_T$, and $v_{S \cup T}$, where $S, T \subseteq [d]$, vertex $v_{S \cup T}$ is eliminated immediately after the latter of $v_S$ and $v_T$.

As an example, suppose $G$ has eight vertices $\{v_1, \ldots, v_8\}$, and its elimination tree $T$ is as shown Figure 3.1. Then the ordering for $G'$ would be

$$v_1, v_{(1)}, v_2, v_{(2)}, v_{(1,2)}, v_3, v_{(3)}, v_4, v_{(4)}, v_{(3,4)}, v_{5}, v_{(5)}, v_{6}, v_{(5,6)}, v_7, v_{(7)}, v_{(5,6,7)}, v_8,$$

which can be verified to generate $T'$ as shown in Figure 3.3.

We prove a series of claims to show the correctness of the elimination order in producing the desired elimination tree.

For simplicity, we introduce some additional terminology: Given two vertices $u, v \in V(G)$ (resp. $G'$) with $u$ eliminated before $v$, we call a path $P$ from $u$ to $v$ an ancestor-certifying path if all internal vertices on $P$ are eliminated before $u$. The existence of such a path implies that $v$ is an ancestor of $u$ in the elimination tree $T$ (resp. $T'$).

Given a vertex $v_S$ with $S \subseteq [d]$, since $p_T(v_i) = p_T(v_j)$ for any $i, j \in S$ by construction, we denote this unique vertex by $p_T(S)$. We also use $C(S)$ to denote the subtree in $G'$ with root $v_S$ and leaves $\{v_i : i \in S\}$.

Finally, let $\pi : V(G') \to |V(G')|$ be the order on the vertices.

**Claim 3.3.** The parent of vertex $v_i$ in $T'$ is $v_{(i)}$.

**Proof.** Since $v_{(i)} \in E(G')$, this edge is an ancestor-certifying path from $v_i$ to $v_{(i)}$ in $G'$. Since $\pi(v_{(i)}) = \pi(v_i) + 1$ (i.e. $v_{(i)}$ is eliminated immediately after $v_i$), the claim follows by the definition of parent. \(\square\)

**Claim 3.4.** For any pair of vertices $v_S, v_R$, where $S \subset R \subseteq [d]$ and $R$ is the smallest subset containing $S$, $v_R$ is the parent of $v_S$ in $T'$.

**Proof.** Since $v_S v_R$ is an edge, it is also an ancestor-certifying path from $v_S$ to $v_{S \cup R}$. Suppose for a contradiction that $v \neq v_R$ is the vertex eliminated earliest after $v_S$ with an ancestor-certifying path from $v_S$ to it, implying $v$ is the parent of $v_S$ rather than $v_R$.

We can modify $P$ as follows: For every edge $v_U v_{(j)} \in P$, where $U \subseteq [d], j \in [d]$, and $p_T(U) = v_j$, we can replace it by a path $P^+$, such that the resulting path is still ancestor-certifying from $v_S$ to $v_U$: Let $i \in U$, so $p_T(v_i) = v_j$. Let $Q$ be a ancestor-certifying path from $v_i$ to $v_j$ in $G$ and therefore $G'$, and let $Q'$ be the
path from \(v_i\) to \(v_j\) in \(C(U)\). All internal vertices \(w\) on \(Q\) satisfies \(\pi(w) < \pi(v_i) < \pi(v_j)\), and all vertices \(w\) on \(Q'\) satisfies \(\pi(w) < \pi(v_U) < \pi(v_j)\). Hence, \(P' = Q' \cup Q \cup v_U v_{(j)}\) is a path from \(v_U\) to \(v_{(j)}\) that we use to replace the \(v_U v_{(j)}\) in \(P\). Similarly, we can replace edges \(v_U v_j \in P\), where \(U \subseteq [d], j \in [d]\), and \(p_T(U) = v_j\).

![Figure 3.4](image)

Figure 3.4: Recall the example of \(G'\) locally, where \(v_1, \ldots, v_7\) are children of \(v_8\) in \(G\). A red edge in an ancestor-certifying path can be replaced by a path like the one in green labelled \(Q \cup Q'\).

After this modification, \(P\) is a union of

1. A path \(Q_1\) in \(C(S)\) from \(v_S\) to some \(i \in S\),
2. Type 1 paths, which are single edges of the form \(v_a v_b\), where \(a, b \in [d]\)
3. Type 2 paths, which are of the form \(v_a, P', v_b\), where \(a, b \in [d]\) and \(p_T(a) = p_T(b)\), and \(P' \subset C(S_{ab})\), where \(S_{ab}\) is the set of smallest cardinality containing \(a\) and \(b\) such that \(v_{S_{ab}} \in V(G')\)
4. If \(v = v_{U}\) for some \(U \subseteq [d]\), a path \(Q_2\) in \(C(U)\) from some \(j \in U\) to \(v_U\). Alternatively, if \(v = v_k\) for some \(k \in [d]\), we let \(Q_2\) be the path with a single vertex \(v_k\).

Furthermore, without loss of generality, we may assume \(v_i\) is the unique vertex on \(P\) with \(i \in S\), for otherwise we could shortcut \(P\) while maintaining all the desired properties. Similarly, in the case that \(v = v_U\), we may assume \(v_j\) is the unique vertex on \(P\) with \(j \in U\). So we can write \(P\) as \(Q_1 \cup P' \cup Q_2\), where \(P'\) is a union of type 1 and type 2 paths. We can view \(P'\) as a directed path, and consider what happens as we traverse along it:

Traversing along a Type 1 path \((v_a, v_b)\) can be viewed as moving from vertex \(v_a\) to \(v_b\) in \(T\), where \(v_b\) is an ancestor of \(v_a\) if \(a < b\), and vice versa. Traversing along a Type 2 path \((v_a, P', v_b)\) can be viewed as moving from vertex \(v_a\) to \(v_b\) in \(T\), where \(v_a\) and \(v_b\) are children of the same parent.

We begin with \(v_i\) and traverse along \(P'\) while making the corresponding moves in \(T\) as described above. Note that all internal vertices \(v_i \in P\) that we move to must satisfy \(\pi(v_i) \leq \pi(v_j)\). It follows that all internal vertices on \(P'\) we move to are contained in \(T\) in the subtree rooted at \(v_i\). If the final vertex on \(P\) is \(v = v_k\) for some \(k \in [d]\), then we must have \(i < k < j\), where \(v_j = p_T(v_i)\), because of our initial assumption for a contradiction. However, in the last valid move in \(T\), it is not possible to go from a vertex strictly in the subtree rooted at \(v_i\) to such a vertex \(v_k\). Alternatively, suppose the final vertex on \(P\) is \(v = v_U\) for some \(U \subseteq [d]\). Let \(v_j\) be the unique vertex on \(P\) with \(j \in U\). Then \(v_j\) is an internal vertex on \(P\) and therefore a descendant of \(v_i\). It follows that \(v_U\) must have been eliminated before \(v_i\) by construction. Hence we get the desired contradiction. 

\[\square\]

**Claim 3.5.** For a vertex \(v_S\) with \(S \subseteq [d]\), if there does not exist \(v_R\) with \(R \supseteq S\), then the parent of \(v_S\) in \(T'\) is \(p_T(S)\).
\textbf{Proof.} Suppose \(v_p = p_T(S)\). Take arbitrarily \(i \in S\), and note \(v_p = p_T(v_i)\). We know there is an ancestor-certifying path \(P_T\) from \(v_i\) to \(v_p\) in \(G\); since \(G \subseteq G'\), there is the same ancestor certifying path \(P_T\) from \(v_i\) to \(v_p\) in \(G'\). Furthermore, there is a path \(Q\) from \(v_S\) to \(v_i\) in \(C(S)\), and all vertices on this path are eliminated before \(v_S\) by construction. Hence \(Q \cup P_T\) is an ancestor-certifying path from \(v_S\) to \(v_p\).

To see that there does not exist a vertex eliminated before \(v_p\) with an ancestor-certifying path from \(v_S\), we use a similar argument as in Claim 3.4; the details are omitted.

\(\square\)

\section{Robust Interior Point Algorithm for General Convex Sets}

\subsection{Robust Central Path Method}

In this section, we give a robust interior point method for the optimization problem

\[
\min_{Ax=b, x \in K_i} c^T x \tag{4.1}
\]

where \(x_i \in K_i \subset \mathbb{R}^{n_i}\) and \(x\) is the concatenation of \(x_i\) lying inside the domain \(K = \prod_{i=1}^{n} K_i \subset \mathbb{R}^n\) with \(n = \sum_{i=1}^{n_i} n_i\). Each step involves solving some linear system of the form \(ADA^T\) where \(D\) is a block diagonal matrix with each block \(D_{ii}\) of size \(n_i \times n_i\).

Our algorithm is based on interior point methods which follow some path \((x(t))\) inside the the interior of the domain \(K\). The path starts at some interior point of the domain \(x(1)\) and ends at the solution \(x(0)\) we want to find. One commonly used path is defined by

\[
x(t) = \arg \min_{Ax=b} c^T x + t \sum_{i=1}^{m} w_i \phi_i(x_i) \tag{4.2}
\]

where \(\phi_i\) are self-concordant barrier functions on \(K_i\). The weight \(w \in \mathbb{R}_{\geq 0}^m\) are fixed throughout the algorithm and is defined according to the cost of updating block \(i\).

\textbf{Definition 4.1.} A function \(\phi\) is a \(\nu\) self-concordant barrier function for compact convex set \(K\) if dom \(\phi = K\) and for any \(x \in K\) and for any \(u \in \mathbb{R}^n\)

\[
|D^3 \phi(x)|u, u, u| \leq 2\|u\|^2/2 \quad \text{and} \quad \|\nabla \phi(x)\|_x^* \leq \sqrt{\nu},
\]

where \(\|\cdot\|_x\) and \(\|\cdot\|_x^*\) are induced norm defined at Definition 4.2.

Since \(\phi_i\) blows up on \(\partial K_i\), \(x(t)\) lies in the interior of the domain for \(t > 0\) (if the interior is non-empty).

Also, by the definition of \(x(t)\), \(x(0)\) is a minimizer of the problem Eq. (4.1). In Section 3, we show how to find the initial point \(x(1)\) quickly by reformulating the problem into an equivalent form.

The key difficulty is to follow the path \(x(t)\) efficiently. To lower the cost of each step, we maintain our \((x, s)\) implicitly. Throughout the algorithm, we can only directly access an approximate of \((x, s)\), which called \((\bar{x}, \bar{s})\). Our algorithm takes \(O(\sqrt{m})\) steps and each step involves solving some linear system very crudely according to \((\bar{x}, \bar{s})\). Then we call a data structure that implicitly maintains \((x, s)\) and the approximation \((\bar{x}, \bar{s})\) via some random sketching scheme (Section 5).

\textbf{Definition 4.2 (Induced Norms).} For each block, we define \(\|v\|_{x_i} \defeq \|v\|_{\nabla^2 \phi_i(x_i)}\), \(\|v\|_{x_i}^* \defeq \|v\|_{(\nabla^2 \phi_i(x_i))^{-1}}\) for \(v \in \mathbb{R}^{n_i}\). For the whole domain, we define \(\|v\|_x \defeq \|v\|_{\nabla^2 \phi(x)} = \sqrt{\sum_i w_i \|v\|_{x_i}^2}\) and \(\|v\|_x^* \defeq \|v\|_{(\nabla^2 \phi(x))^{-1}} = \sqrt{\sum_i w_i^{-1} \|v\|_{x_i}^*^2}\) for \(v \in \mathbb{R}^n\).

This norm depends on the Hessian and so changes as the parameter change. The following lemma about self-concordance implies when the central path parameter is not changed rapidly, then the approximate solution for previous iteration will not be too far from the solution of next iteration.
Lemma 4.3 (Theorem 4.1.6 in [Nes98]). If \( \phi \) is a self-concordant barrier and if \( \|y - x\|_x < 1 \), then we have

\[
(1 - \|y - x\|_x)^2 \nabla^2 \phi(x) \leq \nabla^2 \phi(y) \leq \frac{1}{(1 - \|y - x\|_x)^2} \nabla^2 \phi(x).
\]

Our potential enforce that \( s/t + \nabla \phi(x) \) is close to 0 in \( \nabla^2 \phi(x)^{-1} \) norm.

Definition 4.4 (Potential Function). For each \( i \in [m] \), we define the \( i \)-th coordinate error

\[
\mu_i^t(x, s) \overset{\text{def}}{=} \frac{s_i}{t} + w_i \nabla \phi_i(x_i)
\]

and its norm \( \gamma_i^t(x, s) \overset{\text{def}}{=} \|\mu_i^t(x, s)\|_{x_i}^* \). We define the soft-max function by

\[
\Psi_r(r) \overset{\text{def}}{=} \sum_{i=1}^m \cosh(\lambda \frac{r_i}{w_i})
\]

for some \( \lambda > 0 \) and finally the potential function is the soft-max of the norm of the error of each coordinate

\[
\Phi^t(x, s) = \Psi_r(\gamma_i^t(x, s)).
\]

When \( (x, s) \) or \( t \) is clear in the context, we may ignore them in the notation. We show that to solve Eq. (4.1) approximately, it suffices to find \( x \) such that \( \Phi^t(x, s) \) is bounded for some \( s \) and for a tiny \( t \). The algorithm alternates between decreasing \( t \) multiplicatively and a Newton-like step on \( \Phi^t \) and the proof simply shows the potential \( \Phi \) is bounded throughout.

4.2 Gradient Descent on \( \Psi_r \)

Since our goal is to bound \( \Phi(x, s) = \Psi_r(\gamma) \), we first discuss how to decreases \( \Psi_r(r) \) in general. Suppose we can make step \( r \leftarrow r + \delta \), with step size \( \sum_i w_i^{-1} \delta_{r,i}^2 \leq \alpha^2 \). Then, a natural choice is the steepest descent direction\(^1\):

\[
\delta_r^* = \arg \min_{\|\delta_r\|_{w} \leq \alpha^2} \langle \nabla \Psi_r(r), \delta_r \rangle.
\]

Using that \( \Psi_r(r) = \sum_{i=1}^m \cosh(\lambda \frac{r_i}{w_i}) \), we have \( \nabla_r \Psi_r(r) = \frac{\lambda}{w_i} \sinh(\frac{\lambda}{w_i} r_i) \) and hence

\[
\delta_r^* = \frac{-\alpha \cdot \sinh(\frac{\lambda}{w_i} r_i)}{\sqrt{\sum_j w_j^{-1} \sinh^2(\frac{\lambda}{w_j} r_j)}}.
\]

The following Lemma shows that the direction \( \delta_r^* \) indeed decreases \( \Psi_r \). Furthermore, this step is robust under \( \ell_\infty \) perturbation of \( r \) and \( \ell_2 \) perturbation of \( \delta_r^* \).

Lemma 4.5. Fix any \( r \in \mathbb{R}^m \) and \( w \in \mathbb{R}_{\geq 1}^m \). Given any \( r \in \mathbb{R}^m \) such that \( |r_i - r_i^| < \frac{w_i}{\lambda} \) for all \( i \) and

\[
\delta_r = \frac{-\alpha \cdot \sinh(\frac{\lambda}{w_i} r_i)}{\sqrt{\sum_j w_j^{-1} \sinh^2(\frac{\lambda}{w_j} r_j)}} + \epsilon_r, \quad (4.3)
\]

with \( \sqrt{\sum_i w_i^{-1} \epsilon_{r,i}^2} \leq \frac{\alpha}{8} \). Then, for any \( \alpha \leq \frac{1}{8\lambda} \), we have that

\[
\Psi_r(r + \delta_r) \leq \Psi_r(r) - \frac{\alpha}{2} \|\nabla \Psi_r(r)\|_w + \frac{\alpha \lambda}{2} \sqrt{\sum_i w_i^{-1}}.
\]

\(^1\)We use the * to highlight this is the ideal step and to distinguish with the step we will take.
Proof. By Taylor expansion, we have

$$\Psi_\lambda(r + \delta_r) = \Psi_\lambda(r) + \langle \nabla \Psi_\lambda(r), \delta_r \rangle + \frac{1}{2} \delta_r^T \nabla^2 \Psi_\lambda(\tilde{r}) \delta_r$$

(4.4)

where $\tilde{r} = r + t\delta_r$ for some $t \in [0, 1]$.

For the first order term $\langle \nabla \Psi_\lambda(r), \delta_r - \epsilon_r \rangle$ in Eq. (4.4), we have that

$$\langle \nabla \Psi_\lambda(r), \delta_r - \epsilon_r \rangle = -\alpha \sum_i w_i^{-1} \sinh\left(\frac{\lambda}{w_i} r_i \right) \sinh\left(\frac{\lambda}{w_i} \tilde{r}_i \right)$$

$$\sqrt{\sum_j w_j^{-1} \sinh^2\left(\frac{\lambda}{w_j} \tilde{r}_j \right)}.$$  

Using Lemma A.1 and the assumption $|r_i - \tilde{r}_i| < \frac{w_i}{\alpha}$, we have

$$\sinh\left(\frac{\lambda}{w_i} r_i \right) \sinh\left(\frac{\lambda}{w_i} \tilde{r}_i \right) \geq \frac{6}{7} \sinh\left(\frac{\lambda}{w_i} \tilde{r}_i \right)^2 - \frac{1}{7} \left| \sinh\left(\frac{\lambda}{w_i} r_i \right) \right|.$$  

Hence, we have

$$\langle \nabla \Psi_\lambda(r), \delta_r - \epsilon_r \rangle \leq -\frac{6}{7} \alpha \sum_i w_i^{-1} \sinh^2\left(\frac{\lambda}{w_i} \tilde{r}_i \right) + \frac{1}{7} \alpha \sum_i \left| \sinh\left(\frac{\lambda}{w_i} r_i \right) \right|$$

$$\sqrt{\sum_j w_j^{-1} \sinh^2\left(\frac{\lambda}{w_j} \tilde{r}_j \right)}$$

$$= -\frac{6}{7} \alpha \sqrt{\sum_i w_i^{-1} \sinh^2\left(\frac{\lambda}{w_i} \tilde{r}_i \right)} + \frac{1}{7} \alpha \sqrt{\sum_i w_i^{-1}}$$

(4.5)

Using Lemma A.1 and the assumption $|r_i - \tilde{r}_i| < \frac{w_i}{\alpha}$ again, we have

$$\sqrt{\sum_i w_i^{-1} \sinh^2\left(\frac{\lambda}{w_i} \tilde{r}_i \right)} \geq \sqrt{\sum_i w_i^{-1} \left( \frac{6}{7} \sinh\left(\frac{\lambda}{w_i} \tilde{r}_i \right) - \frac{1}{7} \right)^2}$$

$$\geq \frac{6}{7} \sqrt{\sum_i w_i^{-1} \sinh^2(\frac{\lambda}{w_i} \tilde{r}_i)} - \frac{1}{7} \sqrt{\sum_i w_i^{-1}}.$$  

Putting this into Eq. (4.5), we have

$$\langle \nabla \Psi_\lambda(r), \delta_r - \epsilon_r \rangle \leq -\frac{36}{49} \alpha \sqrt{\sum_i w_i^{-1} \sinh^2\left(\frac{\lambda}{w_i} \tilde{r}_i \right)} + \frac{2}{7} \alpha \sqrt{\sum_i w_i^{-1}}$$

$$\leq -\frac{36}{49} \alpha \||\nabla \Psi_\lambda(r)||_w + \frac{2}{7} \alpha \sqrt{\sum_i w_i^{-1}}$$

(4.6)

For the first order term $\langle \nabla \Psi_\lambda(r), \epsilon_r \rangle$ in Eq. (4.4), we have that

$$\langle \nabla \Psi_\lambda(r), \epsilon_r \rangle = \sum_i \frac{\lambda}{w_i} \sinh\left(\frac{\lambda}{w_i} r_i \right) \epsilon_{r,i}$$

$$\leq \lambda \sqrt{\sum_i w_i^{-1} \sinh^2\left(\frac{\lambda}{w_i} r_i \right)} \sqrt{\sum_i w_i^{-1} \epsilon_{r,i}^2}$$

$$= \frac{1}{8} \alpha \lambda \sqrt{\sum_i w_i^{-1} \sinh^2\left(\frac{\lambda}{w_i} \tilde{r}_i \right)}$$

$$\leq \frac{1}{8} \alpha \||\nabla \Psi_\lambda(r)||_w$$

(4.7)
For the second order term $\delta_r^T \nabla^2 \Psi_\lambda(\bar{r}) \delta_r$ in Eq. (4.4), we note that
\[
\delta_r^T \nabla^2 \Psi_\lambda(\bar{r}) \delta_r = \lambda^2 \sum_i w_i^{-2} \delta_{r,i}^2 \cosh(\frac{\bar{r}_i}{w_i}).
\]
Note that
\[
\sqrt{\sum_i w_i^{-1} \delta_{r,i}^2} \leq \sqrt{\sum_i w_i^{-1} \left( \frac{\alpha \cdot \sinh(\frac{\bar{r}_i}{w_i})}{\sqrt{\sum_j w_j^{-1} \sinh^2(\frac{\bar{r}_j}{w_j})}} \right)^2} + \sqrt{\sum_i w_i^{-1} \epsilon_{r,i}^2} \leq \alpha + \frac{\alpha}{8} = \frac{9\alpha}{8}.
\]
(4.8)
In particular, this shows that $|\delta_{r,i}| \leq \frac{9\alpha}{8} \sqrt{w_i} \leq \frac{9\alpha}{8} w_i$. Using this and Eq. (4.8), we have
\[
\delta_r^T \nabla^2 \Psi_\lambda(\bar{r}) \delta_r = \lambda^2 \sum_i w_i^{-2} \delta_{r,i}^2 \cosh(\frac{\bar{r}_i}{w_i}) \leq \frac{9\alpha}{8} \lambda^2 \sum_i w_i^{-1} |\delta_{r,i}| \cosh(\frac{\bar{r}_i}{w_i}) \leq \frac{9\alpha}{8} \lambda^2 \sqrt{\sum_i w_i^{-1} \delta_{r,i}^2} \sqrt{\sum_i w_i^{-1} \cosh^2(\frac{\bar{r}_i}{w_i})} \leq \left( \frac{9\alpha}{8} \right)^2 \lambda^2 \left( \sqrt{\sum_i w_i^{-1} \sinh^2(\frac{\bar{r}_i}{w_i})} + \sqrt{\sum_i w_i^{-1}} \right) \leq \left( \frac{9\alpha}{8} \right)^2 \lambda^2 \left( \frac{8}{7} \sqrt{\sum_i w_i^{-1} \sinh^2(\frac{\bar{r}_i}{w_i})} + \frac{8}{7} \sqrt{\sum_i w_i^{-1}} \right) \leq \left( \frac{9\alpha}{8} \right)^2 \lambda^2 \left( \lambda \|\nabla \Psi_\lambda(r)\|_w + \lambda^2 \sqrt{\sum_i w_i^{-1}} \right)
\]
(4.9)
where we used $\cosh(x) \leq 1 + |\sinh(x)|$ at the third last inequality and Lemma A.1 at the second last inequality.

Putting Eq. (4.6), Eq. (4.7), and Eq. (4.9) into Eq. (4.4) gives
\[
\Psi_\lambda(r + \delta) = \Psi_\lambda(r) + \langle \nabla \Psi_\lambda(r), \delta \rangle + \frac{1}{2} \delta_r^T \nabla^2 \Psi_\lambda(\bar{r}) \delta_r \leq \Psi_\lambda(r) - \frac{36}{49} \alpha \|\nabla \Psi_\lambda(r)\|_w + \frac{2}{7} \alpha \lambda \sqrt{\sum_i w_i^{-1}} + \frac{1}{8} \alpha \|\nabla \Psi_\lambda(r)\|_w + \frac{1}{2} \left( \frac{9\alpha}{8} \right)^2 \lambda^2 \left( \lambda \|\nabla \Psi_\lambda(r)\|_w + \lambda^2 \sqrt{\sum_i w_i^{-1}} \right) .
\]

Using $\alpha \leq \frac{1}{8\lambda}$, we can simplify it to
\[
\Psi_\lambda(r + \delta) \leq \Psi_\lambda(r) - \frac{36}{49} \alpha \|\nabla \Psi_\lambda(r)\|_w + \frac{2}{7} \alpha \lambda \sqrt{\sum_i w_i^{-1}} + \frac{1}{8} \alpha \|\nabla \Psi_\lambda(r)\|_w + \frac{1}{2} \left( \frac{9\alpha}{8} \right)^2 \lambda^2 \left( \lambda \|\nabla \Psi_\lambda(r)\|_w + \lambda^2 \sqrt{\sum_i w_i^{-1}} \right) \leq \Psi_\lambda(r) - \frac{\alpha}{2} \|\nabla \Psi_\lambda(r)\|_w + \frac{\alpha \lambda}{2} \sqrt{\sum_i w_i^{-1}} .
\]
\]

\[
\square
\]
4.3 Implicit Step

To design the Newton-like step for \((x, s)\), we note that the non-linear equation Eq. (1.3) has an unique solution for any vector \(\mu\). In particular, the solution \(x\) is the solution of the optimization problem \(\min_{A x = b} c^\top x + t \sum_{i=1}^m w_i \phi_i(x_i) - t \mu^\top x\). Hence, we can move \(\mu\) arbitrarily while maintaining Eq. (1.3) by moving \(x\) and \(s\).

Since our goal is to decrease \(\Phi(x, s) = \Psi_\lambda(\gamma)\), similar to Section 4.2, a natural choice is the steepest descent direction:

\[
\delta_\mu^* = \arg \min_{\|\delta_\mu\|_2^2 \leq \alpha} \langle \nabla_\mu \Psi_\lambda(\|\mu_i\|_{z_i}), \mu + \delta_\mu \rangle
\]

with step size \(\alpha\). We can view this is a gradient descent step on \(\Phi\) for \(\mu\) with step size \(\alpha\). Recall that \(\Psi_\lambda(r) = \sum_{i=1}^m \cosh(\lambda \frac{r_i}{w_i})\). Hence, \(\nabla_{\|\mu_i\|_{z_i}} \Psi_\lambda(\|\mu_i\|_{z_i}) = \frac{\lambda}{w_i} \sinh(\frac{\lambda}{w_i} \|\mu_i\|_{z_i})\) and

\[
\nabla_\mu \Psi_\lambda(\|\mu_i\|_{z_i})^* = \frac{\lambda \sinh(\frac{\lambda}{w_i} \|\mu_i\|_{z_i})}{w_i \|\mu_i\|_{z_i}^2} \cdot \nabla \phi_i(x_i)^{-1} \mu_i
\]

Solve Eq. (4.10), we get

\[
\delta_{\mu, i}^* = -\frac{\alpha \sinh(\frac{\lambda}{w_i} \gamma_i^t(x, s))}{\gamma_i^t(x, s) \cdot \sqrt{\sum_{j=1}^m w_j^{-1} \sinh^2(\frac{\lambda}{w_i} \gamma_j(x, s))}} \cdot \mu_i^t(x, s).
\]

To move \(\mu\) to \(\mu + \delta_\mu\) approximately, we take Newton step \((\delta_x^*, \delta_s^*)\):

\[
\frac{1}{t} \delta_x^* + \nabla^2 \phi(x) \delta_x^* = \delta_\mu^*,
\]

\[
A \delta_s^* = 0,
\]

\[
A^\top \delta_y^* + \delta_s^* = 0.
\]

Using \(H_x\) to denote \(\nabla^2 \phi(x)\), we solve the system above, and get

\[
\delta_x^* = H_x^{-1} \delta_\mu^* - H_x^{-1} A^\top (AH_x^{-1} A^\top)^{-1} AH_x^{-1} \delta_\mu^*,
\]

\[
\delta_s^* = t A^\top (AH_x^{-1} A^\top)^{-1} AH_x^{-1} \delta_\mu^*.
\]

Let the orthogonal projection matrix \(P_x = H_x^{-1/2} A^\top (AH_x^{-1} A^\top)^{-1} AH_x^{-1/2}\), then we can rewrite it as

\[
\delta_x^* = H_x^{-1/2} (I - P_x) H_x^{-1/2} \delta_\mu^*,
\]

\[
\delta_s^* = t H_x^{1/2} P_x H_x^{-1/2} \delta_\mu^*.
\]

Our robust algorithm only assume we have \((\tilde{x}, \tilde{s})\), which is an approximation of \((x, s)\). In summary, our step on \(x\) is given by the following:

**Definition 4.6 (Implicit step).** Given \((\tilde{x}, \tilde{s})\), and step size \(\alpha\), the new pair is given by \(x^{\text{new}} = x + \delta_x\) and \(s^{\text{new}} = s + \delta_s\) where

\[
\delta_x = H_x^{-1/2} (I - P_x) H_x^{-1/2} \delta_\mu,
\]

\[
\delta_s = t H_x^{1/2} P_x H_x^{-1/2} \delta_\mu.
\]

\footnote{The derivation of the formula is not used in the main proof as this is just a motivation for the choice of the step. Therefore, we skip the proof of this. An alternative choice is the gradient step on \(\min_{A x = b, A^\top y + \mu = \Phi(x, s)} \Phi(x, s)\). This step will be very similar to the step we use in this paper. But it contains few more terms and may make the proof longer.}

\footnote{We use the \(\ast\) to highlight this is the ideal step and to distinguish with the step we will take.}
**•** $H_x$ is a block diagonal matrix with the $(i, i)$ block given by $w_i \nabla^2 \phi_i(x_i)$,

**•** $P_x$ is the projection matrix $H_x^{-1/2} A^T (AH_x^{-1} A^T)^{-1} AH_x^{-1/2}$,

**•** $\delta_{\mu}$ is the proposed step of $\mu$ defined by $-\alpha \cdot c^t_i(x, s) \cdot \mu^t_i(x, s)$ with $c^t_i(x, s) = \frac{\sinh(\frac{\lambda}{w_i} \gamma^t_i(x, s))}{\gamma^t_i(x, s) \sqrt{\sum_{j=1}^m w_j^{-1} \sinh(\frac{\lambda}{w_j} \gamma^t_j(x, s))}}$

(See Definition 4.4 for the definition of $\gamma_i$ and $\mu_i$),

**•** $\alpha$ is the step size.

Formally, our algorithm is defined in Algorithm 9.

### 4.4 Bounding $\Phi$ under changes of $x$ and $s$

In this section, we will use the following assumptions.

**Assumption 4.7.** We assume the following invariants hold during Algorithm 9.

1. $\|x_i - x_i\|_{\pi_i} \leq \varepsilon$ and $t^{-1} \|x_i - s_i\|_{\pi_i}^* \leq \varepsilon$.
2. $\alpha \leq \varepsilon \leq \frac{1}{200\lambda}$, $\lambda \geq 1$.
3. $w_i \geq 1$ for all $i$.
4. $\Phi(x, s) \leq 10 m^3 \alpha$.

To use Lemma 4.5 to bound the potential, we need to verify Eq. (4.3) and that $|\gamma^t_i(x_{\text{new}}, s_{\text{new}}) - \gamma^t_i(x, s)| \leq \frac{w_i}{\alpha}$.

#### 4.4.1 Verifying conditions of Lemma 4.5

Recall that the ideal step we want to take is $\delta_{\mu,i}^* = -\alpha \cdot c^t_i(x, s) \cdot \mu^t_i(x, s)$.

A rough calculation shows

\[
\gamma^t_i(x_{\text{new}}, s_{\text{new}}) = \|\mu_i + \delta_{\mu,i}^*\|^t_{x_i}
\sim \|\mu_i\|^t_{x_i} - \frac{\alpha}{\|\mu_i\|^t_{x_i}} \cdot c^t_i(x, s) \cdot \mu^t_i \nabla^2 \phi_i(x)^{-1} \mu_i
= \gamma^t_i(x, s) - \alpha \cdot c^t_i(x, s) \cdot \gamma^t_i(x, s)
\]

This shows that Eq. (4.3) should roughly hold. Formally, we prove this on Lemma 4.12. First, we bound the step size for each block $\delta_{x,i}$.

**Lemma 4.8.** Let $\alpha_i = \|\delta_{x,i}\|_{\pi_i}$, then

\[
\sum_{i=1}^m w_i \alpha_i^2 \leq \alpha^2.
\]

This directly implies $\alpha_i \leq \alpha$.

**Proof.** We have

\[
\sum_{i=1}^m w_i \alpha_i^2 = \|\delta_x\|^2_{\pi_x} = \|(I - P_x) H_x^{-1/2} \delta_{\mu}\|^2_{\pi_x} \leq \|H_x^{-1/2} \delta_{\mu}\|^2_{\pi_x} = \|\delta_{\mu}\|^2_{\pi_x} = \alpha^2,
\]

where the first inequality follows by $I - P_x$ is an orthogonal projection matrix and the last equality follows by the step size for $\delta_{\mu}$. 

To bound the change of $\gamma_i$, we first show that $\mu_{\text{new}}$ is close to $\mu + \delta_{\mu}$.
Lemma 4.9 (Change in $\mu$). Under Assumption 4.7, let
\[
\mu_i'(x^{\text{new}}, s^{\text{new}}) = \mu_i'(x, s) + \delta_{\mu,i} + w_i \epsilon_i^{(u)}.
\]
Then, $\|\epsilon_i^{(u)}\|_{x,i}^2 \leq 6\pi \alpha_i$.

Proof. By definition of $\mu$, we have
\[
\mu_i(x^{\text{new}}, s^{\text{new}}) = \frac{s_i^{\text{new}}}{t} + w_i \nabla \phi_i(x^{\text{new}}) \\
= \mu_i(x, s) + \frac{1}{t} \delta_s + w_i(\nabla \phi_i(x^{\text{new}}) - \nabla \phi_i(x_i)) \\
= \mu_i(x, s) + \delta_{\mu,i} + w_i(\nabla \phi_i(x^{\text{new}}) - \nabla \phi_i(x_i) - \nabla^2 \phi_i(\overline{x}_i) \delta_{x,i}),
\]

where the last step follows by $\delta_{\mu,i} = \frac{1}{t} \delta_s + w_i \nabla^2 \phi_i(\overline{x}_i) \delta_{x,i}$.

To bound $\epsilon_i^{(u)}$, let $x^{(u)} = u x^{\text{new}} + (1 - u)x$, then we have
\[
\epsilon_i^{(u)} = \nabla \phi_i(x_i^{(u)}) - \nabla \phi_i(x_i) - \nabla^2 \phi_i(\overline{x}_i) \delta_{x,i} \\
= \int_0^1 (\nabla^2 \phi_i(x_i^{(u)}) - \nabla^2 \phi_i(\overline{x}_i)) \delta_{x,i} \, du.
\]

By Lemma 4.3, we have
\[
(1 - \|x_i^{(u)} - \overline{x}_i\|_{\overline{x}_i})^2 \nabla^2 \phi_i(\overline{x}_i) \leq \nabla^2 \phi_i(x_i^{(u)}) \leq \frac{1}{1 - \|x_i^{(u)} - \overline{x}_i\|_{\overline{x}_i}} \nabla^2 \phi_i(\overline{x}_i).
\]

Note that
\[
\|x_i^{(u)} - \overline{x}_i\|_{\overline{x}_i} \leq \|x_i^{(u)} - x_i\|_{\overline{x}_i} + \|x_i - \overline{x}_i\|_{\overline{x}_i} = u \|\delta_{x,i}\|_{\overline{x}_i} + \overline{\epsilon} \leq \alpha_i + \overline{\epsilon} \leq \alpha + \overline{\epsilon} \leq 2\pi,
\]

where the last inequality follows by $\alpha_i \leq \alpha$ (Lemma 4.8) and $\overline{\epsilon} \leq \overline{\epsilon}$ (Assumption 4.7). Combine two inequalities above and using that $\overline{\epsilon} \leq \frac{1}{2}$, we get
\[
-5\overline{\epsilon} \nabla^2 \phi_i(\overline{x}_i) \leq \nabla^2 \phi_i(x_i^{(u)}) - \nabla^2 \phi_i(\overline{x}_i) \leq 5\overline{\epsilon} \nabla^2 \phi_i(\overline{x}_i).
\]

Using this, Eq. (4.11) and Assumption 4.7, we have
\[
\left(\nabla^2 \phi_i(x_i^{(u)}) - \nabla^2 \phi_i(\overline{x}_i)\right) \left(\nabla^2 \phi_i(\overline{x}_i)\right)^{-1} \left(\nabla^2 \phi_i(x_i^{(u)}) - \nabla^2 \phi_i(\overline{x}_i)\right) \\
\leq \frac{1}{(1 - 1/10)^2} \left(\nabla^2 \phi_i(x_i^{(u)}) - \nabla^2 \phi_i(\overline{x}_i)\right) \left(\nabla^2 \phi_i(x_i^{(u)}) - \nabla^2 \phi_i(\overline{x}_i)\right)^{-1} \\
\leq (6\overline{\epsilon})^2 \nabla^2 \phi_i(\overline{x}_i)
\]

This implies
\[
\left\|\left(\nabla^2 \phi_i(x_i^{(u)}) - \nabla^2 \phi_i(x_i)\right) \delta_{x,i}\right\|_{x_i}^2 = \sqrt{\delta_{x,i}^T \left(\nabla^2 \phi_i(x_i^{(u)}) - \nabla^2 \phi_i(\overline{x}_i)\right)^T \left(\nabla^2 \phi_i(x_i^{(u)}) - \nabla^2 \phi_i(\overline{x}_i)\right)^{-1} \left(\nabla^2 \phi_i(x_i^{(u)}) - \nabla^2 \phi_i(\overline{x}_i)\right) \delta_{x,i}} \\
\leq \sqrt{\left(6\overline{\epsilon}\right)^2 \delta_{x,i}^T \nabla^2 \phi_i(\overline{x}_i) \delta_{x,i}^T} \\
= 6\overline{\epsilon} \delta_{x,i} \|\overline{x}_i\|_{\overline{x}_i} \\
= 6\overline{\epsilon} \alpha_i.
\]

Hence,
\[
\|\epsilon_i^{(u)}\|_{x,i}^2 \leq \int_0^1 \left\|\left(\nabla^2 \phi_i(x_i^{(u)}) - \nabla^2 \phi_i(\overline{x}_i)\right) \delta_{x,i}\right\|_{x_i}^2 \, du \leq 6\overline{\epsilon} \alpha_i.
\]
To get our bound for $\gamma$, we first need two helper lemmas. One is to show $\mu(x, s)$ is close to $\mu(\mathbf{x}, \mathbf{s})$.

**Lemma 4.10.** For all $i \in [m]$, we have

$$\|\mu_i^t(x, s) - \mu_i^t(\mathbf{x}, \mathbf{s})\|_{\mathbf{x}_i} \leq 4w_i \varepsilon.$$

Furthermore, we have that $|\gamma_i^t(x, s) - \gamma_i^t(\mathbf{x}, \mathbf{s})| \leq 3w_i \varepsilon + 2\varepsilon \gamma_i^t(x, s)$.

**Proof.** For the first result, note that

$$\|\mu_i^t(x, s) - \mu_i^t(\mathbf{x}, \mathbf{s})\|_{\mathbf{x}_i} \leq \frac{1}{2} \|s_i - i\|_{\mathbf{x}_i} + w_i \|\nabla \phi_i(x_i) - \nabla \phi_i(\mathbf{x})\|_{\mathbf{x}_i}.$$ 

By Assumption 4.7 and Lemma 4.3, we have $\nabla^2 \phi_i(x_i) \leq (1 + 3\varepsilon) \nabla^2 \phi_i(\mathbf{x}) \leq 2\nabla^2 \phi_i(\mathbf{x})$. Let $x^{(u)} = ux_i + (1-u)\mathbf{x}_i$, we get

$$\|\nabla \phi_i(x_i) - \nabla \phi_i(\mathbf{x})\|_{\mathbf{x}_i} = \int_0^1 \nabla^2 \phi_i(x_i^{(u)}) (x_i - \mathbf{x}_i) du \leq 2\|x - \mathbf{x}_i\|_{\mathbf{x}_i} \leq 2\varepsilon.$$

Hence, $\|\mu_i^t(x, s) - \mu_i^t(\mathbf{x}, \mathbf{s})\|_{\mathbf{x}_i} \leq 3w_i \varepsilon$ and hence the result.

For the second result, note that

$$|\gamma_i^t(x, s) - \gamma_i^t(\mathbf{x}, \mathbf{s})| \leq \|\mu_i^t(x, s) - \mu_i^t(\mathbf{x}, \mathbf{s})\|_{\mathbf{x}_i} + \|\mu_i^t(x, s)\|_{\mathbf{x}_i} - \|\mu_i^t(x, s)\|_{\mathbf{x}_i} \leq 3w_i \varepsilon + 2\|x - \mathbf{x}_i\|_{\mathbf{x}_i} \|\mu_i^t(x, s)\|_{\mathbf{x}_i} = 3w_i \varepsilon + 2\varepsilon \gamma_i^t(x, s)$$

where we used Assumption 4.7 and Lemma 4.3.

The second lemma is to bound $c_i(\mathbf{x}, \mathbf{s})$ and $\sum_{i=1}^{m} w_i^{-1} \sin^2(\frac{\lambda}{w_i} \gamma_i^t(\mathbf{x}, \mathbf{s}))$.

**Lemma 4.11.** Suppose that $\Phi^t(\mathbf{x}, \mathbf{s}) \geq 3 \sum_{i=1}^{m} w_i$, then we have

- $\sum_{i=1}^{m} w_i^{-1} \sin^2(\frac{\lambda}{w_i} \gamma_i^t(\mathbf{x}, \mathbf{s})) \geq 4 \sum_{i=1}^{m} w_i$.
- $0 \leq c_i^t(\mathbf{x}, \mathbf{s}) \leq \lambda$.

Furthermore, if $\cosh(\lambda) \geq \Phi^t(x, s)$, we have $\Phi^t(\mathbf{x}, \mathbf{s}) \geq \frac{1}{2} \Phi^t(x, s)$.

**Proof.** For the first inequality, we note that

$$\sum_{i=1}^{m} w_i^{-1} \sin^2(\frac{\lambda}{w_i} \gamma_i^t(\mathbf{x}, \mathbf{s})) \geq \left( \sum_{i=1}^{m} |\sinh(\frac{\lambda}{w_i} \gamma_i^t(\mathbf{x}, \mathbf{s}))| \right)^2 \left( \sum_{i=1}^{m} w_i \right)^{-1} \geq \left( \sum_{i=1}^{m} (\cosh(\frac{\lambda}{w_i} \gamma_i^t(\mathbf{x}, \mathbf{s})) - 1) \right)^2 \sum_{i=1}^{m} w_i \geq \left( \Phi^t(\mathbf{x}, \mathbf{s}) - m \right)^2 \sum_{i=1}^{m} w_i \geq \frac{4(\sum_{i=1}^{m} w_i)^2}{\sum_{i=1}^{m} w_i} = 4 \sum_{i=1}^{m} w_i$$

where we used that $|\sinh(x)| \geq \cosh(x) - 1$ and the assumption $\Phi^t(\mathbf{x}, \mathbf{s}) \geq 3 \sum_{i=1}^{m} w_i \geq m + 2 \sum_{i=1}^{m} w_i$.

For the second inequality, we note that

$$c_i^t(\mathbf{x}, \mathbf{s}) = \frac{\frac{\lambda}{w_i} \gamma_i^t(\mathbf{x}, \mathbf{s})}{\gamma_i^t(\mathbf{x}, \mathbf{s})} \cdot \sqrt{\sum_{j=1}^{m} w_j^{-1} \sin^2(\frac{\lambda}{w_j} \gamma_j^t(\mathbf{x}, \mathbf{s}))}$$
Finally, we can bound the distance between $\gamma_i\in\bar{\Gamma}_i$ and $\gamma_i^\dagger\in\Gamma_i$. If $\gamma_i^\dagger(\bar{x},\bar{s}) \geq \frac{w_i}{\lambda}$, we have that
\[
\gamma_i^\dagger(\bar{x},\bar{s}) \leq \frac{\lambda}{w_i} \leq \lambda
\]
where we used that $w_i \geq 1$. If $\gamma_i^\dagger(\bar{x},\bar{s}) \leq \frac{w_i}{\lambda}$, we use that $|\sinh(x)| \leq |x|$ for all $|x| \leq 1$ and get
\[
\gamma_i^\dagger(\bar{x},\bar{s}) \leq \frac{2\lambda}{w_i} \sqrt{\sum_{j=1}^{m} w_j^{-1} \sinh^2 \left( \frac{\lambda}{w_i} \gamma_j^\dagger(\bar{x},\bar{s}) \right)}
\]
where we used Eq. (4.12) at the end.

For the third inequality, we note that $\cosh(\lambda) \geq \Phi'\lambda(x, s)$ implies that $\gamma_i^\dagger(\bar{x},\bar{s}) \leq w_i$ for all $i$. Hence, Lemma 4.10 shows that
\[
|\gamma_i^\dagger(x, s) - \gamma_i^\dagger(\bar{x},\bar{s})| \leq 3w_i\bar{\tau} + 2\tau\gamma_i^\dagger(x, s) \leq 5w_i\bar{\tau} \leq \frac{5w_i}{8\lambda}.
\]
Since $\cosh(x - \frac{3}{8}) \geq \frac{1}{2} \cosh(x)$ for all $x$, this shows
\[
\Phi'(\bar{x}, \bar{s}) = \sum_{i=1}^{m} \cosh \left( \frac{\lambda}{w_i} \gamma_i^\dagger(\bar{x},\bar{s}) \right)
\geq \sum_{i=1}^{m} \cosh \left( \frac{\lambda}{w_i} \gamma_i^\dagger(x, s) - \frac{5}{8} \right)
\geq \frac{1}{2} \sum_{i=1}^{m} \cosh \left( \frac{\lambda}{w_i} \gamma_i^\dagger(x, s) \right)
= \frac{1}{2} \Phi'(x, s).
\]

Finally, we can bound the distance between $\gamma^{\text{new}}$ and $\gamma - \alpha c\gamma$.

**Lemma 4.12** (Change in $\gamma$). Under Assumption 4.7, for all $i \in [m]$, let
\[
\epsilon_{\tau,i} \stackrel{\text{def}}{=} \gamma_i^\dagger(x^{\text{new}}, s^{\text{new}}) - \gamma_i^\dagger(x, s) + \alpha \cdot c_i^\dagger(\bar{x},\bar{s}) \cdot \gamma_i^\dagger(\bar{x},\bar{s}).
\]
Assuming $\Phi'(\bar{x}, \bar{s}) \geq 3 \sum_{i=1}^{m} w_i$, we have
\[
\sum_{i=1}^{m} w_i^{-1} \epsilon_{\tau,i}^2 \leq (31\pi^2 + 48 \max_i (w_i^{-1} \gamma_i^\dagger(x, s))^2)\alpha^2
\]

**Proof.** For notation simplicity, we write $\bar{\tau}_i = c_i^\dagger(\bar{x},\bar{s})$. Also, we use $\gamma_i^\dagger(x, z, s)$ to denote $\|\mu_i(x, s)\|_{z_i}^*$. Using $\delta_{\mu,i} = -\alpha \cdot \bar{\tau}_i \cdot \mu_i(\bar{x},\bar{s})$, we have
\[
\gamma_i^\dagger(x^{\text{new}}, s^{\text{new}}) = \|\mu_i(x, s)\|_{x_i}^* + \delta_{\mu,i} + w_i \epsilon_i^\dagger(\mu)_{x_i}^*
= \|\mu_i(x, s) - \alpha \cdot \bar{\tau}_i \cdot \mu_i(\bar{x},\bar{s})\|_{x_i}^* + w_i \epsilon_i^\dagger(\mu)_{x_i}^*
= \|\mu_i(x, s) - \alpha \cdot \bar{\tau}_i \cdot \mu_i(\bar{x},\bar{s})\|_{x_i}^*
\pm \alpha \cdot \bar{\tau}_i \cdot \|\mu_i(\bar{x},\bar{s}) - \mu_i^\dagger(\bar{x},\bar{s})\|_{x_i}^* + w_i \epsilon_i^\dagger(\mu)_{x_i}^*
= (1 - \alpha \cdot \bar{\tau}_i) \gamma_i^\dagger(x, s) \pm \alpha \cdot \bar{\tau}_i \cdot \|\mu_i(\bar{x},\bar{s}) - \mu_i^\dagger(\bar{x},\bar{s})\|_{x_i}^* + w_i \epsilon_i^\dagger(\mu)_{x_i}^*
\tag{4.13}
\]

28
where we used that $0 \leq \alpha \cdot \tau_i \leq 1$ at the end (Lemma 4.11).

In particular, we have that

$$
\gamma_i(x, s) + \alpha \cdot \tau_i \cdot \left( \|\mu_i(x, s) - \mu_i(\bar{x}, \bar{s})\|_{x_i} + w_i \|\epsilon^{(\mu)}_{i}\|_{x_i} \right) \leq \gamma_i(x, s) + \epsilon w_i (4\alpha \tau_i + 6\alpha_i)
$$

(4.14)

where we used Lemma 4.10 and Lemma 4.9 at the end. Hence, we have

$$
|\gamma_i(x, s) - \gamma_i(x, s) + \alpha \cdot \tau_i \cdot \left( \|\mu_i(x, s) - \mu_i(\bar{x}, \bar{s})\|_{x_i} + w_i \|\epsilon^{(\mu)}_{i}\|_{x_i} \right) |

\leq 2\|\alpha - x_i\|_{x_i} \|\mu_i(x, s) - \mu_i(\bar{x}, \bar{s})\|_{x_i} + 4\|\epsilon^{(\mu)}_{i}\|_{x_i}

\leq 2\alpha \cdot \tau_i \cdot \left( \|\mu_i(x, s) - \mu_i(\bar{x}, \bar{s})\|_{x_i} + w_i \|\epsilon^{(\mu)}_{i}\|_{x_i} \right) + 4\alpha \gamma_i(x, s) + \epsilon w_i (16\alpha \tau_i + 24\alpha_i)

\leq 4\alpha \gamma_i(x, s) + \epsilon w_i (16\alpha \tau_i + 24\alpha_i)

(4.15)

where we used Lemma 4.3 on the first inequality, $x_i^{\text{new}} - x_i = \delta_{x_i}$ on the second inequality, the definition of $\alpha_i$ on the second equality, and Eq. (4.14) on the last inequality.

Using Eq. (4.13), we have

$$
|\gamma_i(x, s) - \gamma_i(x, s) + \alpha \cdot \tau_i \cdot \left( \|\mu_i(x, s) - \mu_i(\bar{x}, \bar{s})\|_{x_i} + w_i \|\epsilon^{(\mu)}_{i}\|_{x_i} \right) |

\leq |(1 - \alpha \cdot \tau_i) \gamma_i(x, s) - \gamma_i(x, s) + \alpha \cdot \tau_i \cdot \gamma_i(x, s) + \alpha \cdot \tau_i \cdot \left( \|\mu_i(x, s) - \mu_i(\bar{x}, \bar{s})\|_{x_i} + w_i \|\epsilon^{(\mu)}_{i}\|_{x_i} \right) |

\leq 2\alpha \cdot \tau_i \cdot \left( \|\mu_i(x, s) - \mu_i(\bar{x}, \bar{s})\|_{x_i} + w_i \|\epsilon^{(\mu)}_{i}\|_{x_i} \right) + 8\epsilon w_i (\alpha \tau_i + \alpha_i)

\leq 8\epsilon w_i (\alpha \tau_i + \alpha_i) + 4\alpha_i \gamma_i(x, s) + \epsilon w_i (16\alpha \tau_i + 24\alpha_i)

\leq 10\epsilon w_i (\alpha \tau_i + \alpha_i) + 4\alpha_i \gamma_i(x, s) + \epsilon w_i (16\alpha \tau_i + 24\alpha_i)

(4.16)

where we used Lemma 4.10 and Lemma 4.9 at the end.

Combining Eq. (4.15) and Eq. (4.16), we have

$$
|\gamma_i(x, s) - \gamma_i(x, s) - \alpha \cdot \tau_i \cdot \gamma_i(\bar{x}, \bar{s})| + |\gamma_i(x, s) - \gamma_i(x, s) - \gamma_i(x, s)| + 10\epsilon w_i (\alpha \tau_i + \alpha_i) + 4\alpha_i \gamma_i(x, s) + \epsilon w_i (16\alpha \tau_i + 24\alpha_i)

\leq 10\epsilon w_i (\alpha \tau_i + \alpha_i) + 4\alpha_i \gamma_i(x, s) + \epsilon w_i (16\alpha \tau_i + 24\alpha_i)

(4.17)

where we used $\alpha_i \leq \alpha \leq \frac{1}{200}$ at the end.

Now, we bound the $\ell_2$ norm of $v$, we first note that

$$
\sum_{i=1}^{m} w_i \tau_i^2 = \frac{\sum_{i=1}^{m} w_i \sinh^2 \left( \frac{\lambda \gamma_i(\bar{x}, \bar{s})}{\gamma_i(\bar{x}, \bar{s})^2} \right)}{\sum_{j=1}^{m} \sinh^2 \left( \frac{\lambda \gamma_j(\bar{x}, \bar{s})}{\gamma_j(\bar{x}, \bar{s})^2} \right)}

= \lambda^2 \frac{\sum_{i=1}^{m} w_i \sinh^2 \left( \frac{\lambda \gamma_i(\bar{x}, \bar{s})}{\gamma_i(\bar{x}, \bar{s})^2} \right)}{\sum_{j=1}^{m} \sinh^2 \left( \frac{\lambda \gamma_j(\bar{x}, \bar{s})}{\gamma_j(\bar{x}, \bar{s})^2} \right)}

\leq \lambda^2 \frac{\sum_{i=1}^{m} w_i \left( 1 + \sinh^2 \left( \frac{\lambda \gamma_i(\bar{x}, \bar{s})}{\gamma_i(\bar{x}, \bar{s})^2} \right) \right)}{\sum_{j=1}^{m} \sinh^2 \left( \frac{\lambda \gamma_j(\bar{x}, \bar{s})}{\gamma_j(\bar{x}, \bar{s})^2} \right)}

\leq 2\lambda^2

\text{where we used that } \frac{\sinh^2(x)}{x^2} \leq \cosh^2(x) = \sinh^2(x) + 1 \text{ for all } x \text{ at the second last inequality, we used}

\text{at the second last inequality, we used}

\text{at the second last inequality, we used}
Lemma 4.11 at the last inequality. Using this and \( \sum_i w_i \alpha_i^2 \leq \alpha^2 \) (Lemma 4.8) into Eq. (4.17), we have

\[
\sum_{i=1}^{m} w_i^{-1} \epsilon_{r,i}^2 < 300^2 \alpha^2 \sum_{i=1}^{m} w_i \tau_i^2 + 300^2 \sum_{i=1}^{m} w_i \alpha_i^2 + 48 \max_i (w_i^{-1} \gamma_i^f(x,s))^2 \sum_{i=1}^{m} w_i \alpha_i^2
\]

\[
\leq 600^2 \alpha^2 \lambda^2 + (300^2 + 48 \max_i (w_i^{-1} \gamma_i^f(x,s))^2) \alpha^2
\]

\[
\leq (310 \alpha^2 + 48 \max_i (w_i^{-1} \gamma_i^f(x,s))^2) \alpha^2
\]

where we used that \( \alpha \leq \frac{1}{200} \) at the end.

4.4.2 Bounding the Movement of \( \Phi \)

**Lemma 4.13** (Move from \( \Phi(x,s) \) to \( \Phi(x^{new}, s^{new}) \)). Under Assumption 4.7 and that \( \Phi^f(x,s) \leq \cosh(\frac{\lambda}{200}) \), we have

\[
\Phi^f(x^{new}, s^{new}) \leq (1 - \frac{\alpha \lambda}{2 \sqrt{\sum_{i=1}^{m} w_i}}) \Phi^f(x,s) + \frac{\alpha \lambda}{2} \sqrt{\sum_i w_i^{-1}}.
\]

**Proof.** Let \( r_i = \gamma_i^f(x,s), \tau_i = \gamma_i^f(\tau, \tau) \) and \( \delta_{r,i} = \gamma_i^f(x^{new}, s^{new}) - \gamma_i^f(\tau, \tau) \). Now, we verify the conditions in Lemma 4.5 for \( r_i, \tau_i \) and \( \delta_r \). Lemma 4.10 shows that

\[
|r_i - \tau_i| \leq 3w_i \tau + 2\tau_i \gamma_i^f(x,s).
\]

Since \( \Phi^f(x,s) \leq \cosh(\frac{\lambda}{200}) \), we have \( |\gamma_i^f(x,s)| \leq \frac{w_i}{200} \) and hence

\[
|r_i - \tau_i| \leq 4w_i \tau \leq \frac{w_i}{8 \lambda}
\]

where we used the assumption \( \tau \leq \frac{1}{200 \lambda} \).

Next, Lemma 4.12 shows that

\[
\delta_{r,i} = -\alpha \cdot c_i^f(\tau, \tau) \cdot \gamma_i^f(\tau, \tau) + \epsilon_{r,i}
\]

with

\[
\sum_{i=1}^{m} w_i^{-1} \epsilon_{r,i}^2 \leq (310 \alpha^2 + 48 \max_i (w_i^{-1} \gamma_i^f(x,s))^2) \alpha^2
\]

\[
\leq \left( \frac{310}{(200)^2} + \frac{48}{(200)^2} \right) \alpha^2 \leq \left( \frac{\alpha}{8} \right)^2
\]

where we used \( |\gamma_i^f(x,s)| \leq \frac{w_i}{200} \) and \( \tau \leq \frac{1}{200 \lambda} \). Using the formula of \( c_i^f(\tau, \tau) \), we have

\[
\delta_{r,i} = -\alpha \cdot \frac{\sinh(\frac{\lambda}{w_i} \gamma_i^f(\tau, \tau))}{\sqrt{\sum_{j=1}^{m} w_j^{-1} \sinh^2(\frac{\lambda}{w_j} \gamma_j^f(\tau, \tau))}} + \epsilon_{r,i}
\]

\[
\leq \frac{\alpha}{2} \sqrt{\sum_{j=1}^{m} w_j^{-1} \sinh^2(\frac{\lambda}{w_j} \gamma_j^f(\tau, \tau))} + \epsilon_{r,i}
\]

and this exactly satisfies the conditions in Lemma 4.5.

Now, Lemma 4.5 shows that

\[
\Phi^f(x^{new}, s^{new}) \leq \Phi^f(x,s) - \frac{\alpha \lambda}{2} \sqrt{\sum_{i=1}^{m} w_i^{-1} \sinh^2(\frac{\lambda}{w_i} \gamma_i^f(x,s))} + \frac{\alpha \lambda}{2} \sqrt{\sum_i w_i^{-1}}
\]

\[
\leq \Phi^f(x,s) - \frac{\alpha \lambda}{2} \frac{\sum_{i=1}^{m} |\sinh(\frac{\lambda}{w_i} \gamma_i^f(x,s))|}{\sqrt{\sum_{i=1}^{m} w_i}} + \frac{\alpha \lambda}{2} \sqrt{\sum_i w_i^{-1}}
\]

\[
\leq (1 - \frac{\alpha \lambda}{2 \sqrt{\sum_{i=1}^{m} w_i}}) \Phi^f(x,s) + \frac{\alpha \lambda}{2} \sqrt{\sum_i w_i^{-1}}.
\]
Lemma 4.14 (Change in $\Phi$ by $t$). Let $t_{\text{new}} \leftarrow (1 - h)t$, for $h \leq \frac{1}{8\lambda \sqrt{\max v_i}}$ and $h \leq \frac{1}{2\lambda \|\gamma\|_\infty}$, then

$$
\Phi_{t_{\text{new}}}^r(x, s) \leq (1 + 4h\lambda(\|\sqrt{v}\|_\infty + \|\gamma_i^t/w_i\|_\infty))\Phi^t(x, s).
$$

For all $i \in [m]$, let $t_{\text{new}} \leftarrow (1 - h)t$, for $h \leq \frac{1}{8\lambda \sqrt{\max v_i}}$ and $h \leq \frac{1}{2\lambda \|\gamma\|_\infty}$, then

$$
\Phi_{t_{\text{new}}}^r(x, s) \leq \Phi^t(x, s) + (1 + 4h\lambda(\|\sqrt{v}\|_\infty + \|\gamma_i^t/w_i\|_\infty))\Phi^t(x, s).
$$

Proof. By definition of $\gamma$, we have

$$
\gamma_i^{t_{\text{new}}}(x, s) = \|\frac{s_i}{t_{\text{new}}} + w_i \nabla \phi_i(x_i)\|_{x_i}^* = \|\frac{s}{t(1 - h)} + w_i \nabla \phi_i(x_i)\|_{x_i}^* \\
\leq (1 + 2h)\gamma_i^t + 2hw_i\sqrt{v_i} \\
= \gamma_i^t + 2h(\gamma_i^t + w_iu_i),
$$

where the first inequality follows by definition of self-concordance and $h \leq 0.1$.

Then, for $\Phi_{t_{\text{new}}^r}$, we have

$$
\Phi_{t_{\text{new}}^r}(x, s) = \sum_{i=1}^m \cosh(\lambda\gamma_i^{t_{\text{new}}}/w_i) \\
\leq \sum_{i=1}^m \cosh(\lambda\gamma_i^t/w_i + 2h\lambda(\gamma_i^t/w_i + \sqrt{v_i})) \\
= \sum_{i=1}^m \cosh(\lambda\gamma_i^t/w_i)(1 + 4h\lambda(\gamma_i^t/w_i + \sqrt{v_i})) \\
\leq (1 + 4h\lambda(\|\sqrt{v}\|_\infty + \|\gamma_i^t/w_i\|_\infty))\Phi^t(x, s),
$$

where the last inequality follows by Lemma A.2.

Theorem 4.15. If $\Phi^t(x, s) \leq 10\frac{m^3}{\alpha}$, then

$$
\Phi_{t_{\text{new}}^r}(x_{\text{new}}, s_{\text{new}}) \leq (1 - \frac{\alpha\lambda}{4\sqrt{\sum_{i=1}^m w_i}})\Phi^t(x, s) + \frac{\alpha\lambda}{2} \sqrt{\sum_{i=1}^m w_i} \leq (1 - \frac{\alpha\lambda}{4\sqrt{\sum_{i=1}^m w_i}})\Phi^t(x, s) + \frac{\alpha\lambda}{2} \sqrt{m}.
$$

Proof. By Lemma 4.14, we have

$$
\Phi_{t_{\text{new}}^r}(x_{\text{new}}, s_{\text{new}}) \leq (1 - \frac{\alpha\lambda}{2\sqrt{\sum_{i=1}^m w_i}})\Phi_{t_{\text{new}}}^t(x, s) + \frac{\alpha\lambda}{2} \sqrt{\sum_{i=1}^m w_i} \\
\leq (1 + 4h\lambda(\|\sqrt{v}\|_\infty + \|\gamma_i^t/w_i\|_\infty))(1 - \frac{\alpha\lambda}{2\sqrt{\sum_{i=1}^m w_i}})\Phi^t(x, s) + \frac{\alpha\lambda}{2} \sqrt{\sum_{i=1}^m w_i}.
$$

First, we need to bound $\|\gamma_i^t(x, s)/w_i\|_\infty$. Note that

$$
\Phi^t(x, s) \leq 10\frac{m^3}{\alpha},
$$

this implies $\|\gamma_i^t(x, s)/w_i\|_\infty \leq \frac{\text{arcosh}(10m^3/\alpha)}{\lambda} \leq \frac{3\log(10m^3/\alpha)}{\lambda}$. By our choice of $\lambda$ and $\alpha$, we have $\|\gamma_i^t(x, s)/w_i\|_\infty \leq \|\sqrt{v}\|_\infty$.
Then, we have
\[
\Phi^{i,\text{new}}(x, y, s, t) \leq (1 + 8 + \alpha \lambda \|x\|_\infty)(1 + \frac{\alpha \lambda}{2 \sum_{i=1}^m u_i})\Phi^i(x, s) + \frac{\alpha \lambda}{2} \sqrt{\sum_{i=1}^m w_i^{-1}}
\]
\[
\leq (1 + 8 + \alpha \lambda \|x\|_\infty)(1 + \frac{\alpha \lambda}{2 \sum_{i=1}^m u_i})\Phi^i(x, s) + \frac{\alpha \lambda}{2} \sqrt{\sum_{i=1}^m w_i^{-1}}
\]
\[
\leq (1 - \frac{\alpha \lambda}{4 \sum_{i=1}^m w_i})\Phi^i(x, s) + \frac{\alpha \lambda}{2} \sqrt{\sum_{i=1}^m w_i^{-1}}
\]
where the second step follows by our choice of \( h \leq \frac{\alpha}{2 \sqrt{\|x\|_\infty}} \) and the last step follows by \( w_i \geq 1 \).

\[\square\]

5 Central Path Maintenance

The goal of this section is to present a data-structure to perform the efficient implementation of robust central path step.

**Theorem 5.1** (Robust central path maintenance). Given a matrix \( A \in \mathbb{R}^{d \times n} \) and a block diagonal structure \( n = \sum_{i=1}^m n_i \), such that \( tw(A) = \tau \). There is a randomized data structure CENTRALPATHMAINTENANCE Algorithm 7 that exactly implicitly maintain the central path parameters \( (x, s) \) and supports the following operations:

1. **Initialize**(\( A, x, s, \epsilon, k \)): Given matrix \( A \in \mathbb{R}^{d \times n} \), initial central path parameter \( (x, s) \), target accuracy \( \epsilon \), and phase length \( k \), the data-structure preprocesses in \( O(n \log n) \) time.

2. **MultiplyAndMove**(\( \delta, t \)): It maintains implicitly
   \[
x = x + H_x^{-1/2} (I - P_x) H_x^{-1/2} \delta, s = s + t H_x^{-1/2} P_x H_x^{-1/2} \delta,
   \]
   where \( P_x = H_x^{-1/2} A^T (AH_x^{-1/2} A^T)^{-1} AH_x^{-1/2} \) and, with probability at least \( 1 - \frac{1}{\tau} \), outputs \((\pi, \tilde{\pi})\) such that
   \[
   \|\pi_j - x_j\|_\pi, \leq \epsilon \quad \text{and} \quad t^{-1} \cdot \|\pi_j - s_j\|_\pi \leq \epsilon \quad \text{for all} \; j \in [m].
   \]
   Moreover, for the query sequence \( \delta^{(1)}_\mu, \delta^{(2)}_\mu, \ldots, \delta^{(i)}_\mu \), let \( C_i = \|\delta^{(i)}_\mu - \delta^{(i-1)}_\mu\|_0 \), then the time per call of **MultiplyAndMove** takes \( \tilde{O}(C_i \cdot \tau^3 + k^3 \tau^3 + C_{i+1} k^2 \tau^2) \).

3. **RestartPhase**: It compute \((x, s)\) to \( \epsilon/n \) accuracy explicitly and reset \((\pi, \tilde{\pi})\) to current \((x, s)\) and \( l \) to 0 to control the \( \ell_2 \) norm that heavy-hitter maintains in \( \tilde{O}(n \tau^2) \) time.

We remark that since in the **RestartPhase**, we compute \((x, s)\) to \( \epsilon/n \) accuracy, then we can assume its equivalent to the real feasible solution \((x, s)\). Before we describe how to efficiently implement the data structure above, we need several ingredients. The first one is the maintenance of \( L^{-1} \).

**Lemma 5.2** (Maintain \( L^{-1}x \) informal). Given a vector \( y = L^{-1}x \), we can compute \( y^{\text{new}} = (L + L')^{-1}x \) in time \( O(n \log n \cdot \tau^2) \) time.

**Proof.** We can rewrite \( L' = \sum_{i=0}^{n \log(L')} u^{(i)}(v^{(i)})^T \) where \( u^{(i)} \) and \( v^{(i)} \) are \( O(1) \)-sparse for any \( i \). Then, using Sherman-Morrison formula, we have
\[
(L + uv^T)^{-1}x = L^{-1}x - \frac{L^{-1}uv^T L^{-1}x}{1 + v^T L^{-1}u}.
\]
Since \( u \) is \( O(1) \)-sparse, then we can compute \( L^{-1}u \) in \( O(\tau^2) \) time by **Lemma 2.22**. Hence, we can compute \( y^{\text{new}} \) in \( O(n \log n \cdot \tau^2) \). \[\square\]
However, we cannot hope to use the same technique to maintain the product of $L^{-T}$, since the column of $L^{-T}$ can have $\Omega(n)$ many nonzero entries. Then, we use heavy-hitters to find all the coordinates that change too much and use Lemma 2.23 to compute it exactly.

Lemma 5.3 ($\ell_2$-heavy hitter [Kan+11; Pag12]). There exists a function $\text{Sketch}(\epsilon,n)$ that given $\epsilon > 0$ explicitly returns a matrix $\Phi \in \mathbb{R}^{m \times n}$ with $m_\Phi = O(\epsilon^{-2} \log^3 n)$ and column sparsity $c = O(\log^3 n)$ in $O(nc + \text{poly log } n)$ time, and uses $O(nc)$ spaces to store the matrix $\Phi$. There further exists a function $\text{Decode}(y)$ that given a vector $y = \Phi x$ in time $O(\epsilon^{-2} \log^3 n)$ reports a list $L \subset [n]$ of size $O(\epsilon^{-2} \cdot \log n)$ that with probability at least $1 - \frac{1}{n^\tau}$ includes all $i$ with

$$|x_i| \geq \epsilon \min_{\|y\|_0 \leq \epsilon^{-2}} \|y - x\|_2.$$ 

Algorithm 7 Central Path Maintenance Data Structure Part 1

1: **data structure**: CENTRALPATHMAINTENANCE
2: 3: **private**: members
4: 5: $A \in \mathbb{R}^{d \times n}$
6: 7: $k \in \mathbb{Z}$
8: 9: $\epsilon_\Phi \leftarrow \max_{\epsilon_\Phi} \epsilon$
10: 11: $\Phi \in \mathbb{R}^{m \times n}$
12: 13: $l \in \mathbb{Z}$
14: 15: $l_x, l_s \in \mathbb{Z}^n$
16: 17: **procedure** INITIALIZE($A$, $s$, $\epsilon$, $k$)
18: 19: Let $L_x$ to be Cholesky factorization of $A H_x^{-1} A^T$
20: 21: $\Phi \leftarrow \text{Sketch}(\epsilon_\Phi, n)$
22: 23: $x_{\text{old}} \leftarrow x$, $s_{\text{old}} \leftarrow s$, $k \leftarrow k$, $\epsilon \leftarrow \epsilon$
24: 25: Compute $\Phi H_x^{-1} A^T L_x^{-T}$, $\Phi A^T L_x^{-T}$
26: 27: $l \leftarrow 1$
28: 29: $p \leftarrow 1$
30: 31: $x^{(1)} \leftarrow x$, $s^{(1)} \leftarrow s$
32: 33: **end procedure**
34: 35: **procedure** RESARTPHASE($\pi, \tau$)
36: 37: Recompute $x$ and $s$ using gradient descent
38: 39: Let $L_x$ to be Cholesky factorization of $A H_x^{-1} A^T$
40: 41: **end procedure**

To decouple the proof of Theorem 5.1, we prove each function separately.

Lemma 5.4. The function INITIALIZE($A$, $s$, $\epsilon$, $k$) takes $O(n \log^3 n \cdot \tau^3)$ time to preprocess the data-structure.

Proof. First, we can construct $\Phi$ in $O(n \log^3 n)$ using Lemma 5.3. Now, we consider the time for computing $\Phi H_x^{-1} A^T L_x^{-T}$ and $\Phi A^T L_x^{-T}$. We rewrite $\Phi A L_x^{-T}$ into $(L_x^{-1} A \Phi)^T$. By using the fact that total sparsity of $\Phi$ is $O(n \log^3 n)$ and the column sparsity of $A$ is bounded $O(\tau)$, then $\text{nnz}(A \Phi)^T = O(n \log^3 n \cdot \tau)$. Thus, we can compute $(L_x^{-1} A \Phi^T)^T$ in $O(n \log^3 n \cdot \tau^3)$ time using Lemma 2.22. The proof of $\Phi H_x^{-1} A^T L_x^{-T}$ is the same since $H_x^{-1}$ is a block diagonal matrix with constant size blocks.

Lemma 5.5. The function COMPUTEEXACT($j$) where $j$ is index of block move the approximate central path parameter of block $j$ ($x_j$, $s_j$) to ($x_j$, $s_j$), and maintains $L_x$, $h = L_x^{-1} A H_x^{-1},$ and $\Phi A^T L_x^{-T} h^{(l)}$. Assuming $l \leq k$, each call of COMPUTEEXACT takes $O(k^2 \tau^2)$ time.
Proof. Note that the approximate central path parameter of block \( j \), \((\bar{x}_j, \bar{s}_j)\) was last updated on iteration \( p_j \), which implies \((\bar{x}_j^{(p_j)}, \bar{s}_j^{(p_j)}) = (x_j^{(p_j)}, s_j^{(p_j)})\). Using the invariant of \( x^{(l)} = x^{old} + \sum_{i=1}^{l} H_{\bar{x}_i}^{-1} \delta^{(i)} - H_{\bar{x}_i}^{-1} A^T L_{\bar{x}_j}^{-T} h^{(i)} \), we have

\[
x_j^{(l+1)} = x_j^{(p_j)} + \left( \sum_{i=p_j}^{l} H_{\bar{x}_i}^{-1} \delta^{(i)} - H_{\bar{x}_i}^{-1} A^T L_{\bar{x}_j}^{-T} h^{(i+1)} \right)_j = \bar{x}_j^{(l)} + \left( \sum_{i=p_j}^{l} H_{\bar{x}_i}^{-1} \delta^{(i)} - H_{\bar{x}_i}^{-1} A^T L_{\bar{x}_j}^{-T} h^{(i+1)} \right)_j,
\]

where the last step follows by \( p_j \) is the last time when block \( j \) get updated, then we have \( \bar{x}_j^{(l)} = \bar{x}_j^{(p_j)} = x_j^{(p_j)} \).

Hence, we know that \( x_j^{(l+1)} = x_j^{(l+1)} \). The proof of \( s^{(l+1)} = s^{(l+1)} \) is similar, we omitted the details here.

Now, we show we can perform this function in \( O(k^2 \tau^2) \) time. We can update the cholesky decomposition of \( AH \sigma A^T \) in \( O(\tau^2) \) time by Theorem 2.27. Finally, we show how to efficiently maintain \( h \). The fact that we can update \( L \) in \( O(\tau^2) \) time, implies \( mnz(L_{\bar{x}_i} - L_{\bar{x}_j}) = O(\tau^2) \), then we can compute \( L_{\bar{x}_i}^{-1} A H_{\bar{x}_i} \delta^{(i)} \) in \( O(\tau^2) \) time by Lemma 5.2. Splitting \( L_{\bar{x}_i}^{-1} A H_{\bar{x}_i} \delta^{(i)} \), we get

\[
L_{\bar{x}_i}^{-1} A H_{\bar{x}_i} \delta^{(i)} = L_{\bar{x}_i}^{-1} A H_{\bar{x}_i} \delta^{(i)} + L_{\bar{x}_i}^{-1} A (H_{\bar{x}_i} - H_{\bar{x}_i}) \delta^{(i)}.
\]

We complete the proof by noticing that \( \| (H_{\bar{x}_i} - H_{\bar{x}_i}) \delta^{(i)} \|_0 = O(1) \) and Lemma 2.22.

Similarly, we can maintain \( \Phi H_{\bar{x}_i}^{-1} A^T L_{\bar{x}_i}^{-T} \) and \( \Phi A^T L_{\bar{x}_i}^{-T} \) using \( O(1/\epsilon^2 \delta \tau^2) = \tilde{O}(k^2 \tau^2) \) time using Lemma 5.2, which implies we can also maintain \( \Phi H_{\bar{x}_i}^{-1} A^T L_{\bar{x}_i}^{-T} h^{(i)} \) and \( \Phi A^T L_{\bar{x}_i}^{-T} h^{(i)} \) in the same time by using sparse matrix-vector multiplication.

\[
\text{Lemma 5.6.} \quad \text{The function MULTIPLYANDMOVE(} \delta^{(i)}, t \text{) maintains implicitly} \quad x = x + H_{\bar{x}_i}^{-1/2} (I - P_{\bar{x}_i}) H_{\bar{x}_i}^{-1/2} \delta, \quad s = s + t H_{\bar{x}_i}^{-1/2} P_{\bar{x}_i} H_{\bar{x}_i}^{-1/2} \delta,
\]

where \( P_{\bar{x}_i} = H_{\bar{x}_i}^{-1/2} A^T (AH_{\bar{x}_i}^{-1} A^T)^{-1} AH_{\bar{x}_i}^{-1/2} \) and, with probability at least \( 1 - 1/n^2 \), outputs \((\bar{x}, \bar{s})\) such that

\[
\| \bar{x}_j - x_j \|_{\bar{x}_j} \leq \epsilon \quad \text{and} \quad t \geq \| \bar{s}_j - s_j \|_s \quad \text{for all} \ j \in [m].
\]

Moreover, for the query sequence \( \delta^{(1)}, \delta^{(2)}, \ldots, \delta^{(i)} \), let \( C_i = \| \delta^{(i)} - \delta^{(i-1)} \|_0 \), each call of \text{MULTIPLYANDMOVE} takes \( \tilde{O}(C_{1} \cdot \tau^3 + k^3 \tau^3 + C_{t+1} k^2 \tau^2) \).

Proof. Implicit Maintanance: For the implicit maintainance, it suffices to show we maintain the following invariant:

\[
x^{(l+1)} = x^{old} + \sum_{i=1}^{l} H_{\bar{x}_i}^{-1} \delta^{(i)} - H_{\bar{x}_i}^{-1} A^T L_{\bar{x}_j}^{-T} h^{(i+1)} \quad \text{and} \quad s^{(l+1)} = s^{old} + \sum_{i=1}^{l} t^{(i)} A^T L_{\bar{x}_j}^{-T} h^{(i+1)}. \tag{5.1}
\]

Assume we have \( x^{(l)} = x^{old} + \sum_{i=1}^{l-1} H_{\bar{x}_i}^{-1} \delta^{(i)} + H_{\bar{x}_i}^{-1} A^T L_{\bar{x}_j}^{-T} h^{(i+1)} \), then we have

\[
x^{old} + \sum_{i=1}^{l} H_{\bar{x}_i}^{-1} \delta^{(i)} + H_{\bar{x}_i}^{-1} A^T L_{\bar{x}_j}^{-T} h^{(i+1)} = x^{old} + \left( \sum_{i=1}^{l-1} H_{\bar{x}_i}^{-1} \delta^{(i)} - H_{\bar{x}_i}^{-1} A^T L_{\bar{x}_j}^{-T} h^{(i+1)} \right) + \left( H_{\bar{x}_i}^{-1} \delta^{(i)} - H_{\bar{x}_i}^{-1} A^T L_{\bar{x}_j}^{-T} h^{(i+1)} \right)
\]

\[
= x^{(l)} + H_{\bar{x}_i}^{-1} \delta^{(i)} - H_{\bar{x}_i}^{-1} A^T L_{\bar{x}_j}^{-T} h^{(i+1)} + H_{\bar{x}_i}^{-1} A^T L_{\bar{x}_j}^{-T} A H_{\bar{x}_i}^{-1} \delta^{(i)}
\]

\[
= x^{(l+1)}.
\]

where the second step follows by definition of \( h \), and the last step follows by definition of \( x^{(l)} \). The proof of \( s \) is similar, we omitted details here.
Algorithm 8 Central Path Maintenance Data Structure Part 2

1: procedure MULTIPLYANDMOVE($\delta_\mu$, $t$)  \Comment{Find $(\bar{x}^{(l+1)}, \bar{s}^{(l+1)})$ using $(\bar{x}^{(l)}, \bar{s}^{(l)})$ and move counter $l$}
2: \hspace{1em} $\delta_\mu^{(l)} \leftarrow \delta_\mu$
3: \hspace{1em} $h^{(l+1)} \leftarrow L_x^{-1} A \delta_\mu^{(l)}$
4: \hspace{1em} $v_x = \Phi \left[ \sum_{i=1}^{l} H_{\bar{x}^{(i)}}^{-1} \delta_\mu^{(i)} - H_{\bar{x}^{(i)}}^{-1} A^\top L_{\bar{x}^{(i)}}^{-\top} h^{(i+1)} \right] - (\bar{x}^{(l)} - \bar{x}^{\text{old}})$ \Comment{Compute $h$ using Lemma 5.2}
5: \hspace{1em} $v_x = \Phi \left[ \sum_{i=1}^{l} t(i) A^\top L_{\bar{x}^{(i)}}^{-\top} h^{(i+1)} \right] - (\bar{s}^{(l)} - \bar{s}^{\text{old}})$
6: \hspace{1em} $\bar{x}^{(l+1)} \leftarrow \bar{x}^{(l)}, \bar{s}^{(l+1)} \leftarrow \bar{s}^{(l)}, L_{\bar{x}^{(l+1)}} \leftarrow L_{\bar{x}^{(l)}}$
7: \hspace{1em} for all block $j \cap (\text{Decode}(v_x) \cup \text{Decode}(v_s))$ do
8: \hspace{2em} $\bar{x}_j' \leftarrow \bar{x}_j^{(l)} + \sum_{i=p_j}^{l} H_{\bar{x}^{(i)}}^{-1} \delta_\mu^{(i)} - H_{\bar{x}^{(i)}}^{-1} A^\top L_{\bar{x}^{(i)}}^{-\top} h^{(i+1)}$
9: \hspace{2em} $\bar{s}_j' \leftarrow \bar{s}_j^{(l)} + \sum_{i=p_j}^{l} t(i) A^\top L_{\bar{x}^{(i)}}^{-\top} h^{(i)}$
10: \hspace{2em} if $\|\bar{x}_j' - \bar{x}_j^{(l)}\|_{\bar{x}_j} > \epsilon_\Phi$ or $t^{-1} \|\bar{s}_j' - \bar{s}_j^{(l)}\|_{\bar{s}_j} > \epsilon_\Phi$ then
11: \hspace{3em} UpdateBlock($j, \bar{x}_j', \bar{s}_j'$)
12: \hspace{2em} end if
13: \hspace{1em} end for
14: \hspace{1em} $l \leftarrow l + 1$
15: \hspace{1em} end if
16: \hspace{1em} if $l > k$ then
17: \hspace{2em} RestartPhase($\bar{x}, \bar{s}$)
18: \hspace{1em} end if
19: \hspace{1em} end procedure
20:
21: procedure COMPUTEEXACT($j, \bar{x}_j', \bar{s}_j'$) \Comment{Lemma 5.5}
22: \hspace{1em} $\bar{x}_j^{(l+1)} \leftarrow \bar{x}_j', \bar{s}_j^{(l+1)} \leftarrow \bar{s}_j'$
23: \hspace{1em} $p_j \leftarrow l$
24: \hspace{1em} $L_{\bar{x}^{(l+1)}} \leftarrow \text{UPDATE}(L_{\bar{x}^{(l+1)}}, i)$ \Comment{Theorem 2.27}
25: \hspace{1em} $h^{\text{old}} \leftarrow L_{\bar{x}^{(l+1)}}^{-1} A H_{\bar{x}^{(l+1)}}^{-1} \delta_\mu^{(l)}$
26: \hspace{1em} Compute $\Phi H_{\bar{x}^{(l)}}^{-1} A^\top L_{\bar{x}^{(l)}}^{-\top} h^{(l)}$ and $\Phi A^\top L_{\bar{x}^{(l)}}^{-\top} h^{(l)}$
27: \hspace{1em} end procedure
Explicit Maintainance: Recall that $p_j$ be the last iteration that we set $x_j$. We define $\delta_x^{(i)} \overset{\text{def}}{=} H_{\pi^{(i)}(t)}^{-1} \delta^{(i)} - H_{\pi^{(i)}(t)}^{-1} A^T L_{\pi^{(i)}(t)} h^{(i+1)}$. Then, we have

$$\|x_j - x_j\|_{\pi_j} = \left\| \sum_{i=p_j}^l \delta^{(i)} \right\|_{\pi_j}.$$

By guarantee of Lemma 5.3, we have $\|H_{\pi^{(i)}(t)}^{-1/2}(\pi^{(i)} - x^{(i)})\|_\infty \leq \epsilon_s \|H_{\pi^{(i)}(t)}^{-1/2}(\pi^{(i)} - x^{(i)})\|_2 = \epsilon_s \|\pi^{(i)} - x^{(i)}\|_{\pi^{(i)}}$. Now, it’s suffices to show $\|x^{(i)} - x^{(i)}\|_{\pi^{(i)}}$ is bounded. WLOG, we assume $\delta^{(i)} = 0$ for all $i < s_j$. Then, we have

$$\|x^{(i)} - x^{(i)}\|_{\pi^{(i)}} = \sqrt{\sum_{j=1}^m \sum_{i=1}^l \delta^{(i)}_j}^2.$$

Now, We proceed with strong induction: assuming $\|x^{(l')} - x^{(l')}\|_{\pi^{(l')}} \leq c \cdot l' \alpha$ for some universal constant for any $l' < l \leq k$, and we want to show that $\|x^{(l)} - x^{(l)}\|_{\pi^{(l)}} \leq c \cdot l \alpha$. Since $\|x^{(l)} - x^{(l-1)}\|_{\pi^{(l)}} \leq c \cdot (l-1) \alpha$, we have $\|x^{(l-1)} - x^{(l-1)}\|_{\pi^{(l-1)}} \leq 2c \cdot (l-1) \alpha + \alpha$. By guarantee of algorithm, we have $\|x^{(l)} - x^{(l)}\|_{\pi^{(l)}} \leq n_j \cdot 2c \alpha \cdot \epsilon_s \leq 0.1$. This implies

$$\|x^{(l)} - x^{(l')}\|_{\pi^{(l')}} \leq 0.1$$

for all $l' \leq l$. By Lemma 4.3, we have $0.8 H_{\pi^{(l)}} \preceq H_{\pi^{(l')}} \preceq 1.2 H_{\pi^{(l)}}$. Then, we have $\|x^{(l)} - x^{(l)}\|_{\pi^{(l)}} = \|\sum_{i=1}^l \delta^{(i)}\|_{\pi^{(l)}} \leq 1.2 \cdot l \cdot \|\delta^{(i)}\|_{\pi^{(l)}} = 1.2l \alpha$

Since $l \leq k$, then we have $\|x_j - x_j\|_{\pi_j} \leq n_j \cdot \epsilon_s \cdot c \alpha = O(\epsilon)$ by our choice of $\epsilon_s$. The failure probability directly follows by the failure probability of Lemma 5.3.

Complexity: Since $\|\delta^{(l)} - \delta^{(l)}\|_{\pi^{(l)}} \leq C_l$, we have $\|AH_{\pi^{(l)}}(\delta^{(l)} - \delta^{(l)})\|_{\pi^{(l)}} \leq C(l \cdot \tau)$. Then, we can compute $h^{(l+1)}$ in $O(C_l \cdot \tau^3)$ using Lemma 2.22. For compute of $x_j, x_j$, we show that we can compute $(A^T L_{\pi^{(l)}} h^{(l+1)})_j$ in $O(\tau^2)$ time, since all blocks are constant size and $l \leq k$. Rewrite $(A^T L_{\pi^{(l)}} h^{(l+1)})_j$ to $(A^T L_{\pi^{(l)}} h^{(l+1)})_j e_j$, we have

$$(A^T L_{\pi^{(l)}} h^{(l+1)})_j = h^{(l+1)} e_j$$

where the right hand side can be compute in $O(\tau^2)$ time by Lemma 2.22. By Lemma 5.3, we need to calculate $x_j, x_j$ for at most $O(\epsilon^{-2} \log n)$ many $j$. Since the function ComputeExactly maintains the invariant that $h^{(l+1)} = L_{\pi^{(l)}}^{-1} A H_{\pi^{(l)}}(\delta^{(l)} - \delta^{(l)})$, we have $h^{(l+1)} = h^{(l)} + L_{\pi^{(l)}}^{-1} A H_{\pi^{(l)}}(\delta^{(l)} - \delta^{(l)})$. For compute $v_x$ and $v_x$, assuming that we have $\Phi H_{\pi^{(l)}} A^T L_{\pi^{(l)}} h^{(l)}$ and $\Phi A^T L_{\pi^{(l)}} h^{(l)}$, we can compute $v_x$ and $v_x O(m \Phi \cdot C_l \cdot \tau) = \tilde{O}(k^2 \cdot C_l \cdot \tau)$ time, since $\|h^{(l+1)} - h\| = O(C_l \cdot \tau)$ and $\Phi A^T L_{\pi^{(l)}} e_j \in \mathbb{R}^{m \times n}$ is maintained explicitly. Note that the number of we call ComputeExact($j$) is exactly $C_{l+1}$, and by Lemma 5.5, then the cost is $O(C_{l+1} \cdot k^2 \cdot \tau^2)$. Summing all the cost up, we have the total cost is

$\tilde{O}(C_l \cdot \tau^3 + \epsilon^{-2} \log n \cdot k \cdot \tau^3 + C_{l+1} \cdot k^2 \cdot \tau^2) = \tilde{O}(C_l \cdot \tau^3 + k^3 \cdot \tau^3 + C_{l+1} k^2 \cdot \tau^2)$.

\[\square\]

Lemma 5.7. Suppose the function MoveAndMultiply($\delta, t$) is called for $T$ times during the algorithm, then, ComputeExact($j$), is at most called for $T \cdot k / \epsilon$ many times.

Proof: For a single phase, we have $k$ many iteration and for each iteration, we move our central path parameter by a constant in the corresponding norm, that is $\sum_{x_j^{(i+1)} - x_j^{(i)} \|_{x_j^{(i)}} \leq 1$ which directly implies

$$\sum_{i=1}^k \sum_{j=1}^m \|x_j^{(i+1)} - x_j^{(i)}\|_{x_j^{(i)}}^2 \leq k \cdot \alpha.$$
Then we have except for $\frac{4}{\epsilon} \cdot k^2$ many coordinates, we have $\sum_{i=1}^{k} \|x_j^{(i)}(i+1) - x_j^{(i)}\|_2 \leq \frac{\epsilon}{4}$, by Cauchy-Schwarz, we have
\[
\sum_{i=1}^{k} \|x_j^{(i+1)} - x_j^{(i)}\|_2 \leq \frac{\epsilon \alpha}{4}.
\]
Let $\alpha_j^{(i)} \overset{\text{def}}{=} \|x_j^{(i+1)} - x_j^{(i)}\|_2$, by Lemma 4.3, we have $H_{x_j^{(i)}} \leq e^{2\alpha_j^{(i)}} H_{x_j^{(i+1)}}$. Thus, we have $H_{x_j^{(i)}} \leq e^{\sum_{i=1}^{k} \alpha_j^{(i)}} H_{x_j^{(k)}} \leq \frac{4}{\epsilon} H_{x_j^{(k)}}$ by our choice of $\alpha$, which shows all Hessians are $\frac{4}{\epsilon}$-approximation of each other. This implies for all the coordinate that does not belong to $\frac{4}{\epsilon} \cdot k^2$, we have $\|x_j^{(l)} - x_j^{(1)}\|_2 \leq \epsilon$ for all the $l \leq k$. Thus, for each phase, we will update $\frac{4}{\epsilon} \cdot k^2$ many coordinates, then the total number of updates can be bounded by $O(\frac{T}{\epsilon} \cdot k^2/\epsilon) = O(T \cdot k/\epsilon)$. \hfill \Box

Before we prove the statement about RESTART PHASE, we need to show that we can find an approximate solution in a small ball.

**Lemma 5.8.** Given an radius $R$ and an approximate central path parameter $(\bar{x}, \bar{z})$ such that there exists a feasible central path parameter in $\ell_\infty$ ball with radius $R$, then there exists algorithm that runs in $O(n \log^2 R/\epsilon)$ time and find an optimal feasible central path parameter $(x, s)$ to target accuracy $\epsilon$.

For our propose, it’s suffice to use gradient descent. The following lemma shows that we can achieve linear convergence by taking projected gradient steps repeatedly.

**Lemma 5.9** (Theorem 52 in [LS19]). Let $H$ be a positive definite matrix and $Q \subseteq \mathbb{R}^n$ be a convex set. Let $f : Q \to \mathbb{R}$ be a twice differentiable function. Suppose that there are constant $0 \leq \mu \leq L$ such that for all $x \in Q$ we have $\mu \cdot H \leq \nabla^2 f(x) \leq L \cdot H$. For any $x^{(0)} \in Q$ and any $k \geq 0$ if we apply the update rule
\[
x^{(k+1)} = \arg\min_{x \in Q} f(x^{(k)}) + \nabla f(x^{(k)})^\top (x - x^{(k)}) + \frac{L}{2} \|x - x^{(k)}\|_H^2
\]
then it follows that
\[
\|x^{(k)} - x^*\|_H^2 \leq \left(1 - \frac{\mu}{L}\right)^k \|x^{(0)} - x^*\|_H^2.
\]

**Proof of Lemma 5.8.** Note that, we have $\nabla^2 \phi_j(x_j) \approx H_{x_j}$ then by Theorem A.5, we can find the corresponding $\phi_j^*$ and we can compute $\phi_j^*(x)$ in $O(n_1^{O(1)} \log(1/\epsilon)) = O(\log(1/\epsilon))$ time. Let $f(x) = e^\top x + t \sum_{j=1}^m \phi_j^*(x_i)$. Since $\|x^* - x^{(0)}\|_H^2 \leq R$ by assumption, we can find the optimal $(x, s)$ in $O(n \cdot \log^2 (R/\epsilon))$ time using Theorem A.5. \hfill \Box

### 6 Put It All Together

In this section, we combine the central path method in Section 4 with the data structure and prove the main result.

**Lemma 6.1.** During the Main algorithm, Assumption 4.7 is always satisfied.

**Proof.** The first assumption follows by the choice of $\epsilon_{mp}$ and the guarantee of MULTIPLY AND MOVE in Algorithm 8. The second and third assumptions directly follows by our choice of $\lambda$, $\alpha$, and $\epsilon_{mp}$. Now, we the potential $\Phi^*(x, s)$ is always bounded by $10^{\frac{2n}{\alpha}}$. At the beginning of algorithm, we use Theorem 3.2 to modify the convex program with parameter $\min_{\alpha} \left(\frac{1}{2} \lambda \right)$. Then, the initial point $x$ and $s$ satisfies $\gamma_2(x, s) = \|s_i + w_i \nabla \phi_i(x_i)\|_2 \leq \frac{1}{\lambda}$ and hence $\Phi^*(x, s) \leq \epsilon \cdot m$. Hence, we have $\Phi^*(x, s) \leq O(m)$ during the whole algorithm by Theorem 4.15. \hfill \Box
Algorithm 9 Main Algorithm

1: procedure CentralPathStep($\overline{x}, \overline{s}$)
2:   for $i = 1 \rightarrow m$ do
3:     $\mu_i^t \leftarrow \frac{s_i}{\lambda} + w_i \nabla \phi_i(\overline{x})$
4:     $\gamma_i^t \leftarrow \|\mu_i^t\|_{\overline{x}}^*$
5:     $c_i^t \leftarrow \frac{\sinh(\frac{1}{\lambda} \gamma_i^t(\overline{x}, \overline{s}))}{\gamma_i^t(\overline{x}, \overline{s}) \sqrt{\sum_{j=1}^m w_j^{-1} \sinh^2(\frac{1}{\lambda} \gamma_j^t(\overline{x}, \overline{s}))}}$
6:     $\delta_{\mu,i} \leftarrow -\alpha \cdot c_i^t \cdot \mu_i^t$
7:   end for
8: return $\delta_{\mu}$
9: end procedure

10: procedure Main($A, b, c, \phi, \delta$)
11:   $\lambda \leftarrow 100 \log^2 n$, $\alpha \leftarrow (2^{10} \lambda^2)^{-1}$, $\kappa \leftarrow \frac{1}{100} \alpha$
12:   $\delta \leftarrow \min(\frac{\delta^2}{4}, \frac{1}{\lambda})$
13:   $\epsilon_{mp} \leftarrow 2^{-10} \lambda$
14:   $k \leftarrow n^{0.25}$
15:   $w \leftarrow 1_m$
16: Modify the convex program and obtain an initial $x$ and $s$ by Theorem 3.2
17:   CentralPathMaintenance mp. 
18:   mp.INITIALIZE($A, x, s, \epsilon_{mp}, k$) 
19:   $t \leftarrow 1$
20: while $t > \delta^2 / (4\nu)$ do
21:     $t_{\text{new}} \leftarrow (1 - \frac{\kappa}{\sqrt{\nu}}) t$
22:     $\delta_{\mu} \leftarrow \text{CentralPathStep}(\overline{x}, \overline{s})$
23:     $(\overline{x}, \overline{s}) \leftarrow \text{mp.MultiplyAndMove}(\delta_{\mu}, t)$
24:     $t \leftarrow t_{\text{new}}$
25: end while
26: Recover $x, s$ using Theorem 3.2.
27: end procedure


Theorem 6.2 (Main Result). Given a convex program
\[ \min_{Ax=b, x \in \prod_{i=1}^{m} K_i} c^\top x \]
where \( K_i \) are compact convex set. For each \( i \in [m] \), we are given a \( \nu_i \)-self concordant barrier function \( \phi_i \) for \( K_i \). Also, we are given \( x^{(0)} = \arg \min_x \sum_i \phi_i(x_i) \) where \( x_i \in K_i \). Assume that
1. Diameter of the polytope: For any \( x \in \prod_{i=1}^{m} K_i \), we have that \( \|x\|_2 \leq R \).
2. Lipschits constant of program: \( \|c\|_{\infty} \leq L \).
Then, for any \( \delta > 0 \), the randomized algorithm MAIN runs in time \( \tilde{O}(n^{1.25} \cdot \tau^3 \cdot \log(1/\delta)) \) and finds a vector \( x \) such that
\[ c^\top x \leq \min_{Ax=b, x \in \prod_{i=1}^{m} K_i} c^\top x + LR \cdot \delta \]
\[ \|Ax\|_1 \leq 3\delta \cdot (R\|A\|_1 + \|b\|_1) \]
\[ x \in \prod_{i=1}^{m} K_i. \]

Proof. For each iteration, we decrease \( t \) by \( \frac{\kappa}{\sqrt{\nu}} \) factor, the algorithm takes \( T = \tilde{O}(\sqrt{n} \log(\nu/\delta)) \) many iterations. By our choice of \( \epsilon_{mp} \) and Lemma 5.7, we have \( \sum_t C_t = \tilde{O}(\frac{T}{k}) \).

Thus, the total runtime of \( \text{MULTIPLYANDMOVE} \) is
\[ \tilde{O}(\sum_{t=1}^{T} C_t \cdot \tau^3 + k^3 \tau^3 + C_{t+1}k^2 \tau^2) = \tilde{O}(T \cdot k^3 \tau^3 + \tilde{O}(\frac{T}{k} \cdot \log(\nu/\delta)) \cdot \tilde{O}(\tau^3 + k^2 \tau^2) \]
\[ = \tilde{O}(T \cdot k^3 \tau^3 + Tk^2 \tau^2 + Tk^{-1} \tau^3) \]
\[ = \tilde{O}(n^{1.25} \tau^3 \log(1/\delta)). \]

For the function \( \text{CENTRALPATHSTEP} \), it suffices to maintain the scalar and the direction separately, then we can calculate the \( \delta_{\mu} \) in \( O(C_{t+1}) \) time. Thus, the total runtime of the algorithm is bounded by \( \tilde{O}(n^{1.25} \tau^3 \log(1/\delta)) \).

The accuracy guarantee of solution directly follows by Theorem 3.2.

7 Acknowledgement

I would like to thank my advisor Yin Tat Lee, for his guidance on this project and mentorship over the past years. I would also like to thank my collaborator Sally Dong on many helpful and inspirational discussions. Finally, I want to thank my parents Ronglin and Mei for their encouragement and always being supportive.
References


A Appendix

A.1 Hyperbolic function lemmas

Lemma A.1. For any $x, y \in \mathbb{R}$ with $|y| \leq \frac{1}{8}$, we have

$$|\sinh(x + y) - \sinh(x)| \leq \frac{1}{7} |\sinh(x)| + \frac{1}{7}.$$

Note that $\sinh(x + y) = \sinh(x) \cosh(y) + \cosh(x) \sinh(y)$. Using that $|\cosh(x) - \sinh(x)| \leq 1$, we have

$$|\sinh(x + y) - \sinh(x)| \leq |\sinh(x)||\cosh(y) - 1| + \cosh(x) |\sinh(y)| \leq |\sinh(x)|(|\cosh(y) - 1| + |\sinh(y)|) + |\sinh(y)|$$

The result follows from this and $|\cosh(y) - 1| + |\sinh(y)| \leq \frac{1}{7}$ for $|y| \leq \frac{1}{8}$.

Lemma A.2. For any $x \geq 0$ and $0 \leq y \leq 1$, we have

$$\cosh(x + y) \leq \cosh(x) + 2y \cosh(x)$$

Proof. Note that $\cosh(x + y) = \cosh(x) \cosh(y) + \sinh(x) \sinh(y)$ and $\exp(x) = \sinh(x) + \cosh(x)$, then we have

$$\cosh(x + y) = \cosh(x) [\exp(y) - \sinh(y)] + \sinh(x) \sinh(y)$$

$$\leq \cosh(x) [\exp(y) - \sinh(y)] + \cosh(x) \sinh(y)$$

$$\leq \cosh(x) \exp(y)$$

$$\leq \cosh(x) + 2y \cosh(x),$$

where we use $\exp(y) \leq 1 + 2y$ for $0 \leq y \leq 1$. \qed

A.2 Extension of Locally Strongly Convex Function

In this section, we show that any locally strongly convex and with Lipschitz gradient function can be “extended” to $\mathbb{R}^n$. The main tool we are using for this is the convex conjugate:

Definition A.3 (Convex Conjugate). Given a convex function $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$, we define its convex conjugate $f^* : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ by

$$f^*(\theta) = \sup_{x \in \mathbb{R}^n} x^\top \theta - f(x).$$

One key property about convex conjugate is that it swaps the strong convexity and the Lipschitz constant of gradient.

Lemma A.4. Given a convex function $f : \mathbb{R}^n \to \mathbb{R}$. If $f$ is $\mu$-strongly convex, then $\nabla f^*$ is $\frac{1}{\mu}$-Lipschitz. If $\nabla f$ is $L$-Lipschitz, then $f^*$ is $\frac{1}{L}$-strongly convex. Furthermore, let $x_\theta \overset{\text{def}}{=} \arg \max_{x} \theta^\top x - f(x)$. Then, we have that

$$\nabla^2 f^*(\theta) = (\nabla^2 f(x_\theta))^{-1}. \quad (A.1)$$

Theorem A.5. Given a convex function $f : \mathbb{R}^n \to \mathbb{R}$. Suppose that there is a matrix $H \succ 0$ such that

$$\frac{1}{2} H \preceq \nabla^2 f(x) \preceq 2H$$

for all $x$ with $\|x\|_H < \lambda$ for some $\lambda > 0$. Then, there is a convex function $g : \mathbb{R}^n \to \mathbb{R}$ such that

- $\frac{1}{2} H \preceq \nabla^2 g(x) \preceq 2H$ for all $x$. 

41
• $g(x) = f(x)$ for all $x$ with $\|x\|_H < \frac{1}{8}$.
• For any $x$, we can compute $g(x)$ up to $\epsilon$ additive accuracy in time $n^{O(1)} \log(\frac{1}{\epsilon})$.
• $g(x)$ does not depend on $f(x)$ for $\|x\|_H \geq \lambda$.

**Proof.** By considering $\overline{f}(x) = f(H^{-1/2}x)$, we can assume $H = I$ without loss of generality. We construct $g$ as follows:

$$f|_B(x) = \begin{cases} f(x) & \text{if } \|x\|_2 \leq \lambda \\ +\infty & \text{otherwise} \end{cases},$$

$$h(\theta) = (f|_B)^*(\theta) + r(\|\theta - \nabla f(0)\|_2),$$

$$r(t) = \frac{1}{2}((t - \frac{\lambda}{4})^+)^2,$$

$$g = h^*.$$

First, we note that $f$ is $\frac{1}{4}$-strongly convex on $\|x\|_2 \leq \lambda$ and hence $f|_B$ is $\frac{1}{2}$-strongly convex. Therefore, Lemma A.4 shows that $\nabla (f|_B)^*$ is $2$-Lipschitz. By construction, the gradient of $r(\|\theta - \nabla f(0)\|_2)$ is $1$-Lipschitz. Hence, we have that $\nabla h$ is $3$-Lipschitz.

Next, for any $\theta \in \mathbb{R}^n$, we define $x_\theta$ by

$$x_\theta \overset{\text{def}}{=} \arg \max_x \theta^\top x - f|_B(x).$$

By the optimality condition, we have

$$\theta = \nabla f|_B(x_\theta).$$

Taking derivatives on $\theta$ on both sides, we have $I = \nabla^2 f(x_\theta) Dx_\theta$ where $Dx_\theta$ is the Jacobian of $x_\theta$. Hence, for any $\|x_\theta\|_2 < \lambda$, we have

$$\|Dx_\theta\|_2 = \|\nabla^2 f(x_\theta)^{-1}\|_2 \leq 2.$$

This shows that $\|x_\theta - x_{\theta'}\|_2 \leq 2 \|\theta - \theta'\|_2$. Using this and $x_{\nabla f(0)} = 0$, we have

$$\|x_\theta\|_2 < \lambda$$

for any $\theta$ with $\|\theta - \nabla f(0)\|_2 < \frac{1}{4}$. Hence, for those $\theta$, (Eq. (A.1)) shows that

$$\nabla^2 (f|_B)^*(\theta) = \nabla^2 f^*(\theta) = (\nabla^2 f(x_\theta))^{-1} \geq \frac{1}{2} I.$$

For $\|\theta - \nabla f(0)\|_2 > \frac{1}{4}$, we note that $r$ is $\frac{1}{2}$-strongly convex. Hence, $h$ is $\frac{1}{2}$-strongly convex.

By Lemma A.4 again, we have that $g \overset{\text{def}}{=} h^*$ is $\frac{1}{3}$-strongly convex and $\nabla g$ is $2$-Lipschitz.

Finally, we need to prove that $g = f$ on a small ball. We define $\theta_x = \arg \max_\theta x^\top \theta - h(\theta)$. By similar argument as above, we have that

$$\|\theta_x - \theta_y\|_2 \leq 2 \|x - y\|_2$$

and hence $\|\theta_x - \nabla f(0)\|_2 < \frac{1}{4}$ for all $\|x\|_2 < \frac{1}{8}$. Since on the ball $\{\theta$ such that $\|\theta_x - \nabla f(0)\|_2 < \frac{1}{4}\}$, $h$ agrees with $(f|_B)^*$, we have that

$$g(x) = x^\top \theta_x - h(\theta_x) = x^\top \theta_x - (f|_B)^*(\theta_x) = f|_B(x) = f(x).$$

\[\square\]