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 $\min_{Ax=b, x\in K} c^\top x$ 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

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:

$$\min_{Ax=b, x_i \in K_i \text{ for } i \in [m]} c^\top x \tag{1.1}$$

where $x_i \in K_i \subset \mathbb{R}^{n_i}$ and x is the concatenation of x_i lying inside the domain $K \stackrel{\text{def}}{=} \prod_{i=1}^n K_i \subset \mathbb{R}^n$ 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 $AA^\top$ where we contract each block of matrix in the product into a single entry. Then, we define the treewidth of A, $\mathrm{tw}(A)$, 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 $AA^\top$ 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, $O(n^{2.373\cdots})$. We note that the current fastest algorithm for solving linear systems, a special case of general convex optimization problem above, invloves 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 $\widetilde{O}(n^{1.25} \cdot \mathrm{tw}(A)^3)$.

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 $\widetilde{O}(\sqrt{m})$ iteration IPM that is robust under l_∞ 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

$$x(t) = \arg\min_{Ax=b} c^\top x + t \sum_{i=1}^m w_i \phi_i(x_i) \tag{1.2}$$

where ϕ_i are self-concordant barrier functions on K_i that blows up on ∂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 ϕ_i are $O(1)$ self-concordant.

The weight $w \in \mathbb{R}^m_{>0}$ 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 $w = \mathbf{1}_m$. Since ϕ_i blows up on ∂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 $(\overline{x}, \overline{s})$. Our algorithm takes $\widetilde{O}(\sqrt{m})$ steps and each step involves solving some linear system very crudely according to $(\overline{x}, \overline{s})$. 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 $(\overline{x}, \overline{s})$ 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)

$$\begin{aligned} s/t + \nabla\phi(x) &= \mu, \\ Ax &= b, \\ A^\top y + s &= c \end{aligned} \tag{1.3}$$

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

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

and maintain the poten $\Phi^t(x, s) \leqslant O(m)$. Now the ℓ_∞ pertubation of γ translates to a small multiplicative change to Φ^t.

1.1.2 Cholesky Decomposition

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

$$P_x = H_x^{-1/2} A^\top (A H_x^{-1} A^\top)^{-1} A H_x^{-1/2},$$

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 $A H_x^{-1} A^\top$ in the expression for P_x is a symmetric, positive-definite matrix and $A H_x^{-1} A^\top$ has the same underlying graph as $A A^\top$.

Any symmetric, positive-definite matrix M admits a unique *Cholesky factorization* $M = L L^\top$, 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 $M \stackrel{\text{def}}{=} AH_x^{-1}A^\top$; 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 depdent on the column sparsity of L, which is equivalent height of corresponding vertex in elimation 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 correponding elimination tree via finding the balanced vertex separator on G_A recurisively.

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

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

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, $H_x^{-1} \succ 0$ is a block-diagonal positive definite matrix, where block i is the Hessian of ϕ_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 *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 $M := AA^\top$; 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 $PMP^\top = LL^\top$ 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 *support graph* of A is the graph $G(A) = (V, E)$ with $V = \{v_1, \ldots, v_d\}$, and $v_iv_j \in E$ 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 *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 $e = uv \in V(G)$, there is a node $t \in V(T)$ such that $u, v \in X(t)$.

The *width* of a tree-decomposition (X, T) is $\max\{|X(t)| - 1 \ : \ t \in T\}$. The *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 $\widetilde{O}(tw(A))$, where $tw(A)$ is the treewidth of $G(A)$. The permutation can be computed in time $\widetilde{O}(d \cdot tw(A)^2)$.*

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 $|E(G)| \leqslant n\tau$.* □

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

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 $\max\{nnz(A_i) : A_i$ a column of $A\} \leqslant tw(A) - 1$.* □

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 τ, 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 π*, denoted by G^+_π, is constructed as follows:

Algorithm 1 Construct G_π^+

$G_\pi^+ \leftarrow (V, E)$
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 $\pi(u) \leqslant i$ **then**
      add edge $\pi^{-1}(i)v$ to G_π^+
    end if
  end for
end for
return G_π^+

This construction of G_π^+ is also known as the elimination game on G, which intuitively models the canonical Cholesky factorization algorithm on $PAA^\top P^\top = LL^\top$, where P is the permutation matrix for π: 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 L is non-zero in the factorization algorithm. It turns out the adjacency matrix of the filled graph G_π^+ precisely gives the nonzero structure of the triangular factor L. Hence, our goal is to choose π to decrease the number of edges in G_π^+.

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* π is the tree T defined by the following parent-children relation: For a vertex $v \in V$, its parent is $\arg\min\{\pi(w) : w \in N_{G_\pi^+}(v),\ \pi(w) > \pi(v)\}$; 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 π is the *minimum etree height*.

When the rows of A are reordered according to π, 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 10×10 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.

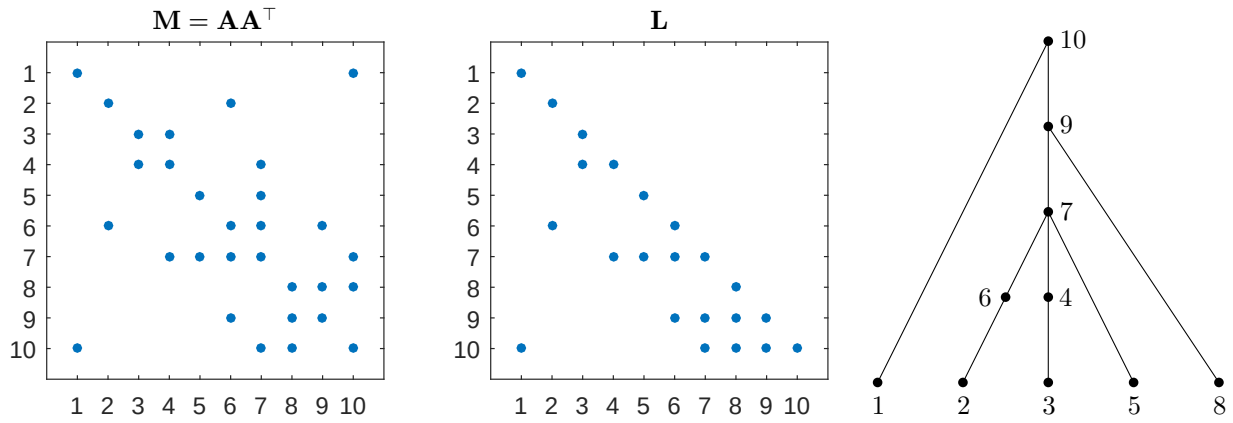

Figure 2.1: Each blue dot represents a non-zero entry in the matrix.

Lemma 2.13. *If uw 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*

$$\textit{separator number} < \textit{treewidth} \leqslant \textit{min etree height} \leqslant \textit{separator number} \cdot \log n.$$

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 π 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 τ. Suppose* `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 $|S| \leqslant O(\log k) \cdot |S^{opt}|$, where S^{opt} is the optimal $2/3$-balanced vertex separator,*
3. *the algorithm runs in time $T_{sep}(k) = \widetilde{O}(k\tau^2)$.*

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

Proof. we will use a list notation $(v_1, \ldots, v_k)$ to denote the ordering π 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 `makeElimOrder`: Let $T(k)$ denote the run-time on a graph with k vertices. Then

$$\begin{cases} T(k) = O(1) & k \leqslant \tau \\ T(k) \leqslant T(c'k) + T((1-c')k) + T_{sep}(k) & k > \tau \end{cases}$$

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

To understand the height of the elimination tree, we make the crucial observation that, at every recursive step `makeElimOrder`(H), the subroutine `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, `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 τ, all subgraphs of G has $1/2$-balanced vertex separators of size at most τ; $1/2$-balanced vertex separators are trivially $2/3$-balanced. Therefore, the minimal $2/3$-balanced vertex separator in any recursive iteration of `makeElimOrder` has size at most τ, 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. `makeElimTree` has run-time

$$\sum_{v \in V(G)} \sum_{w \in N(v)} O(\alpha(n)) \leqslant 2|E(G)|O(\log n) \leqslant \widetilde{O}(n\tau),$$

where $\alpha(n)$ is the inverse-Ackermann function. □

Algorithm 2 Constructing an Elimination Tree

```
1: procedure MAKEELIMORDER(G)
2:     if |V(G)| ⩽ τ then
3:         return arbitrary ordering of V(G)
4:     end if
5:     (G_1, S, G_2) ← approxBalancedSeparator(G)
6:     L_1 ← makeElimOrder(G_1)
7:     L_2 ← makeElimOrder(G_2)
8:     L ← arbitrary ordering of S
9:     return L_1 + L_2 + L
10: end procedure
11:
12: procedure MAKEELIMTREE(G, π)
13:     // We use union-find with an additional vertex tag on sets, as follows:
14:     // MakeSet(v, x) makes the singleton set with element v and tag x
15:     // Union(u, v, x) takes the union of the sets containing u and v, and tags the resulting set with x
16:     // Find(v) returns (v_s, x) where v_s is the representative of the set containing v with tag x
17:     for i = 1 to n do
18:         v ← π^{-1}(i)
19:         MakeSet(v, v)
20:         for w ∈ N(v) do
21:             (w_s, x) ← Find(w)
22:             (v_s, y) ← Find(v)
23:             if w_s != v_s and w_s != null then
24:                 Set x's parent to be v in T
25:                 Union(w_s, v_s, v)
26:             end if
27:         end for
28:     end for
29: 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 τ. Suppose S is an 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 $|S'| \leqslant O(\log n)|S|$ in $\widetilde{O}(n\tau^2)$ time.*

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

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 $|A|, |B| \leqslant c|V(G)|$.

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 $|S|, |\bar{S}| \leqslant c|V(G')|$. 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 set $\widetilde{S} = \{v \in G \ : \ (v, v') \in E(S, \bar{S})\}$ is an c-balanced vertex separator in G: Indeed, let $A = \{v \in G \ : \ v \in S \setminus \widetilde{S}\}$ and $B = \{v \in G \ : \ v \in \bar{S} \setminus \widetilde{S}\}$. Since $|S|, |\bar{S}| \leqslant c \cdot 2n$, we have $|A| = 1/2(|S| - |\widetilde{S}|) \leqslant c \cdot n$, 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 $\widetilde{A} = A \cup A' \cup S$ and $\widetilde{B} = B \cup B' \cup S'$, then $(\widetilde{A}, \widetilde{B})$ is a cut in G', with $E(\widetilde{A}, \widetilde{B}) = \{(v, v') \ : \ v \in S\}$. To see this cut is c-balanced, note that $|\widetilde{A}| = 2|A| + |S| \leqslant 2cn + (1 - c)n = (1 + c)n = \frac{1+c}{2} \cdot (2n) < c \cdot |V(G')|$, and similarly for $|\widetilde{B}|$.

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 $\widetilde{O}(m^{1.5})$ 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 $\mathbf{v}_i, \mathbf{w}_i$ of length $O(1)$, such that $\sum_{i,j \in S} \|\mathbf{v}_i - \mathbf{v}_j\|^2 \leqslant o(1)$. Define the directed distance $d(i, j) = \|\mathbf{v}_i - \mathbf{v}_j\|^2 - \|\mathbf{w}_i - \mathbf{v}_i\|^2 + \|\mathbf{w}_i - \mathbf{v}_j\|^2$. For any given value α, there is an algorithm which, using a single max-flow computation, either outputs a valid $O(\frac{\log(n)\alpha}{n})$-regular directed flow f_p such that $\sum_{ij} d(i, j) \geq \alpha$, or a c'-balanced cut of expansion $O(\log(n)\frac{\alpha}{n})$.*

The max-flow algorithm used in the above lemma is the Goldberg and Rao algorithm in time $\widetilde{O}(m^{1.5})$, 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 $m \leqslant O(n\tau)$, 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 τ, this is a contradiction. □

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 $\mathcal{M}_i$ 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 $nnz(A) \leqslant n\tau$.*

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:

Algorithm 3 Compute L^{-1}

```
M = [L | I_{n×n}]
for i = 1 to n do
    M^i ← M^i / l_ii
    for k = i + 1 to n do
        M^k ← M^k − l_ki M^i
    end for
end for
return right half of M
```

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

First, note that $\mathcal{L}_j^{-1}(1) = \cdots = \mathcal{L}_j^{-1}(j-1) = \{j\}$. After the j-th iteration, $\mathcal{L}_j^{-1}(j) = \mathcal{L}_j$, 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 $k \notin \mathcal{L}_j^{-1}(k-1)$, then $\mathcal{L}_j^{-1}(k) = \mathcal{L}_j^{-1}(k-1)$. On the other hand, if $k \in \mathcal{L}_j^{-1}(k-1)$, then $\mathcal{L}_j^{-1}(k) = \mathcal{L}_j^{-1}(k-1) \bigcup \mathcal{L}_k$.

This gives us a way to characterize precisely the construction of $\mathcal{L}_j^{-1}$, which is the following process:

Algorithm 4 Make $\mathcal{L}_j^{-1}$

```
L_j^{-1} ← L_j
i ← j + 1
while i ⩽ n do
    k ← min{k ∈ L_j^{-1} : k ≥ i}
    L_j^{-1} ← L_j^{-1} ∪ L_k
    i ← k + 1
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 $\mathcal{L}_j^{-1} = \mathcal{L}_j$. Next, observe that, we take some $k \neq j \in \mathcal{L}_j$ and add L_k to our set, but $k \in V(P_j)$, and $L_k \subseteq V(P_k)$. However, note that $P_k \subset P_j$. This process continues recursively. Hence, $\mathcal{L}_j^{-1} \subseteq V(P_j) \leqslant h(T) + 1$. □

Lemma 2.22. *Given the elimination tree for A with height τ, 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 $Lx = v$

1: $x \leftarrow \mathbf{0}_n$
2: **for** increasing j with $v_j \neq 0$ **do**
3: $\quad x_j \leftarrow v_j / L_{jj}$
4: $\quad v \leftarrow v - x_j L_j$
5: **end for**
6: **return** x

Since L has column density τ, each iteration of the `for` loop takes $O(\tau)$ time. In the j-th iteration, line 4 changes up to τ 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 τ, for any v, we can solve for $(L^{-\top}v)_i$ in time $O(\tau^2)$.*

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

In the i-th row, we have

$$L_{ii}^\top x_i + \cdots + L_{ik}^\top x_k = v_i, \tag{2.1}$$

where the LHS contains at most τ 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. $\|L^{-1}Ae_j\|_0 \leqslant h(T) + 1$.

Proof. We have $L^{-1}Ae_j = \sum_{i \in \mathcal{A}_j} A_{ij} \cdot L_i^{-1}$. Note that $\mathcal{A}_i$ 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 $M = LL^\top$ in time*

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

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

Theorem 2.26. *Given a positive definite matrix $M \in \mathbb{R}^{n\times n}$ and its elimination tree T of height τ, 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 τ. Then Lemma 2.11 implies $|L_j| \leqslant \tau$ for all j. By Lemma 2.25, we have the time of Cholesky factorization is bounded by $\mathcal{O}(n\tau^2)$. □

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 $\mathcal{T}$ has height τ. Then for any λ 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 τ.*

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_x^{-1} is a diagonal matrix.

Remark 2.28. Suppose H_x^{-1} is block-diagonal with m total blocks, where block i is the Hessian of ϕ_i, with constant size. To maintain $AH_x^{-1}A^\top$ 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_x^{-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

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

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 ν_i-self concordant barrier function ϕ_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 $\|x\|_2 \leqslant R$.*
2. *Lipschitz constant of the program: $\|c\|_\infty \leqslant L$.*

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

$$\begin{aligned} \min \quad & \bar{c}^\top \bar{x} \\ s.t. \quad & \bar{A}\bar{x} = \bar{b}, \quad \bar{x} \in \prod_{i=1}^{m} K_i \times \mathbb{R}_+ \end{aligned} \tag{$\overline{CP}$}$$

with

$$\bar{A} = [A \mid b - Ax^{(0)}],\ \bar{b} = b,\ \textit{and}\ \bar{c} = \begin{bmatrix} \frac{\delta}{LR} \cdot c \\ 1 \end{bmatrix},$$

satisfies the following:

1. $\bar{x} = \begin{bmatrix} x^{(0)} \\ 1 \end{bmatrix}$ *and* $\bar{y} = \mathbf{0}_d$ *are feasible primal and dual vectors, with slack* $\bar{s} = \begin{bmatrix} \frac{\delta}{LR} \cdot c \\ 1 \end{bmatrix}$, *and* $\|\bar{s} + \nabla\bar{\phi}(\bar{x})\|^*_{\bar{x}} \leqslant \delta$, *where* $\bar{\phi}(\bar{x}) = \sum_{i=1}^m \phi_i(\bar{x}_i) - \log(\bar{x}_{m+1})$.
2. *For any* $\bar{x}$ *such that* $\bar{A}\bar{x} = \bar{b}$, $\bar{x} \in \prod_{i=1}^m K_i \times \mathbb{R}_+$, *and* $\bar{c}^\top \bar{x} \leqslant \overline{OPT} + \delta^2$, *the vector* $\bar{x}_{1:n}$ *is an approximate solution to the original convex program* (CP) *in the following sense:*

$$\begin{aligned} c^\top \bar{x}_{1:n} &\leqslant \min_{Ax=b,\ x \in \prod_{i=1}^m K_i} c^\top x + LR \cdot \delta \\ \|A\bar{x}_{1:n} - b\|_1 &\leqslant 3\delta \cdot (R\|A\|_1 + \|b\|_1) \\ \bar{x}_{1:n} &\in \prod_{i=1}^m K_i. \end{aligned}$$

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

Theorem 3.2. *Let* $A, b, c, \bar{A}, \bar{b}, \bar{c}, x^{(0)}$ *be defined as in Theorem 3.1. Given the elimination tree* T *for* A, *we can construct a modified convex program*

$$\min\ c'^\top \begin{bmatrix} x \\ y \end{bmatrix} \quad s.t. \quad A' \begin{bmatrix} x \\ y \end{bmatrix} = b', \quad x \in \prod_{i=1}^m K_i, \quad y \in \mathbb{R}_+^{d'}, \tag{CP'}$$

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

1. $x' = \begin{bmatrix} x^{(0)} \\ \frac{1}{d'}\mathbf{1} \end{bmatrix}$ *and* $\mathbf{0}$ *are feasible primal and dual vectors with slack* $s' = \begin{bmatrix} \frac{\delta}{LR} \cdot c \\ \frac{1}{d'}\mathbf{1} \end{bmatrix}$, *and* $\|s' + \nabla\bar{\phi}(x')\|^*_{x'} \leqslant \delta$, *where* $\bar{\phi}(x') = \sum_{i=1}^m \phi_i(x'_i) - \sum_j \log(y_j)$.
2. *For any* $(x, y) \in \prod_{i=1}^m K_i \times \mathbb{R}_+^{d'}$ *such that* $A'[x\ y]^\top = b'$, *and* $c'^\top [x\ y]^\top \leqslant OPT' + \delta^2$, *the vector* x *is an approximate solution to the original convex program* (CP) *in the following sense:*

$$\begin{aligned} c^\top x &\leqslant \min_{Ax=b,\ x \in \prod_{i=1}^m K_i} c^\top x + LR \cdot \delta \\ \|Ax\|_1 &\leqslant 3\delta \cdot (R\|A\|_1 + \|b\|_1) \\ x &\in \prod_{i=1}^m K_i. \end{aligned}$$

3. *We can construct an elimination order and tree* T' *for* A', *such that* $height(T') \leqslant O(height(T) \cdot \log d)$. *Furthermore, if* t_v *is the height of vertex* v *in* T, *and* t'_i *is the height of vertex* i *in* T', *then*

$$\sum_{v \in V(T')} t'^2_v \leqslant O(\log d \cdot \sum_{v \in V(T)} t_v^2).$$

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 $A'[x\ y]^\top = b'$. 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: add constraints (translates to adding rows to A' and entries to b')

$$
\begin{aligned}
y_{\{i_1\}} &= y_{\{i_1,i_2\}} \\
y_{\{i_2\}} &= y_{\{i_1,i_2\}} \\
&\vdots \\
y_{\{i_{k-1}\}} &= y_{\{i_{k-1},i_k\}} \\
y_{\{i_k\}} &= y_{\{i_{k-1},i_k\}} \\
y_{\{i_1,i_2\}} &= y_{\{i_1,i_2,i_3,i_4\}} \\
y_{\{i_3,i_4\}} &= y_{\{i_1,i_2,i_3,i_4\}} \\
&\vdots \\
y_{\{i_1,\ldots,i_{k/2}\}} &= y_{\{i_1,\ldots,i_k\}} \\
y_{\{i_{k/2+1},\ldots,i_k\}} &= y_{\{i_1,\ldots,i_k\}} \\
y_{\{i_1,\ldots,i_k\}} &= y_{\{i\}}
\end{aligned}
$$

5: **end for**
6: $c' \leftarrow \begin{bmatrix} \frac{\delta}{L_I R} \cdot c \\ \frac{1}{d'} \cdot \mathbf{1} \end{bmatrix}$ ▷ $\frac{1}{d'} \cdot \mathbf{1}$ 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 $\bar{A}\bar{x} = \bar{b}$.

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 $\bar{A}$). We use $y_{\{1\}}, \ldots, y_{\{d\}} \in \mathbb{R}_+$ to denote the new variables introduced. Compared against $\bar{A}\bar{x} = \bar{b}$, we see that

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

A solution here might not yield a solution to the original $\bar{A}\bar{x} = \bar{b}$. 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 $OPT' = \overline{OPT}$ where the two are optimal values for (CP') and $(\overline{CP})$ respectively.

Second, any $(x, y) \in \prod_{i=1}^{m} K_i \times \mathbb{R}_+^{d'}$ satisfying $A'[x\ y]^\top = b'$ also satisfies $y_S = y_T$ for all valid indices $S, T \subseteq [d]$; hence, $\bar{A}[x\ y_{\{1\}}]^\top = \bar{b}$. If on top of this, we also have $c'^\top[x\ y]^\top \leqslant OPT' + \delta^2$, then $(x, y_{\{1\}}) \in \prod_{i=1}^{m} K_i \times \mathbb{R}_+$ satisfies $\bar{A}[x\ y_{\{1\}}]^\top = \bar{b}$, and $\bar{c}^\top[x\ y_{\{1\}}]^\top \leqslant \overline{OPT} + \delta^2$. 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 $v_S v_R$, 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 $v_S v_i$ and $v_S v_{\{i\}}$. 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.

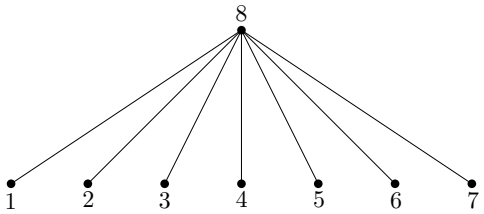

Figure 3.1: The elimination tree T at a single vertex, with seven children. For simplicity, we label a vertex v_i by its subscript i.

After new constraints are added in the current iteration, G' locally looks like the following:

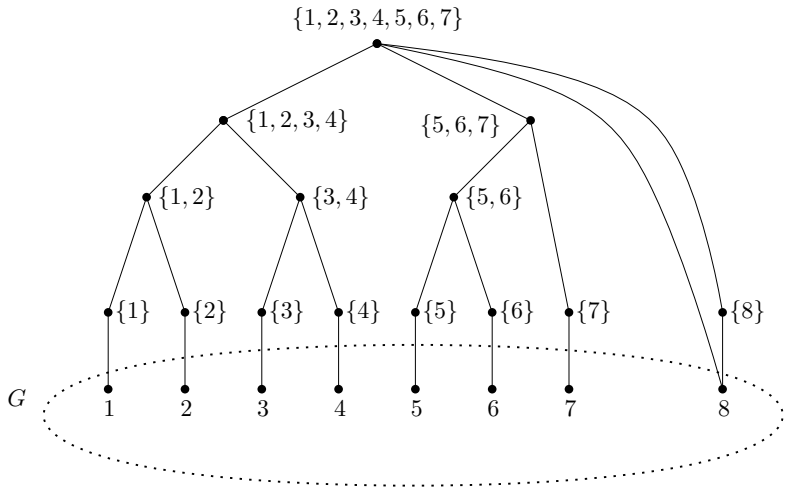

Figure 3.2: The dashed ellipse represents the original graph G. One iteration adds all vertices and edges shown outside to. For simplicity, we label a vertex by its subscript.

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. $v_{\{i\}}$ 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:

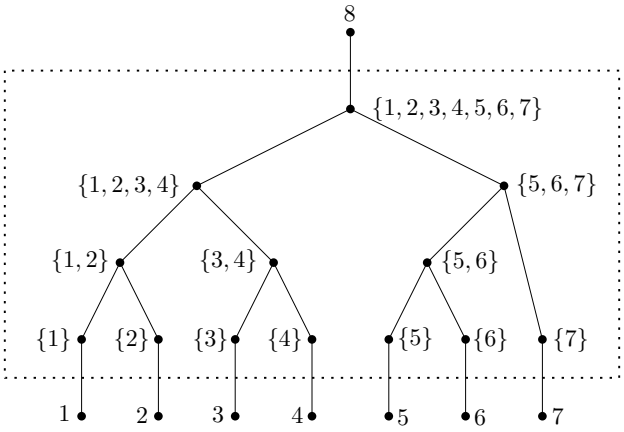

Figure 3.3: The elimination tree T' locally at the vertex v_8. Inside the dashed box are the changes compared to T, caused by the new constraints from the current iteration. For simplicity, vertices are labelled by the subscripts in their names.

If we can produce the elimination tree described above, then it is straightforward to see that $\mathit{height}(T') \leqslant O(\mathit{height}(T) \cdot \log d)$, 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

$$\begin{aligned}\sum_{v\in V(T')} t'^2_v &= \sum_{i\in\{1,\dots,d\}} \left(t'^2_{v_i} + \sum_{S\in P(v_i)} t'^2_{v_S} \right) \\ &\leqslant \sum_{i\in\{1,\dots,d\}} (t_i \log d)^2 + \log d \cdot (t_i \log d)^2 \\ &\leqslant \log^4 d \cdot \sum_{i\in\{1,\dots,d\}} t_i^2.\end{aligned}$$

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 $v_1, \dots, v_d$. 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 $\{v_1, \dots, v_8\}$, and its elimination tree T is as shown Figure 3.1. Then the ordering for G' would be

$$v_1, v_{\{1\}}, v_2, v_{\{2\}}, v_{\{1,2\}}, v_3, v_{\{3\}}, v_4, v_{\{4\}}, v_{\{3,4\}}, v_{\{1,2,3,4\}}, v_5, v_{\{5\}}, v_6, v_{\{6\}}, v_{\{5,6\}}, v_7, v_{\{7\}}, v_{\{5,6,7\}}, v_8,$$

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 $\pi : V(G') \mapsto [|V(G')|]$ be the order on the vertices.

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

Proof. Since $v_i v_{\{i\}} \in E(G')$, 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. □

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_k: 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_U 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, $P^+ = Q' \cup Q \cup v_j v_{\{j\}}$ 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 $p_T(U) = v_j$.

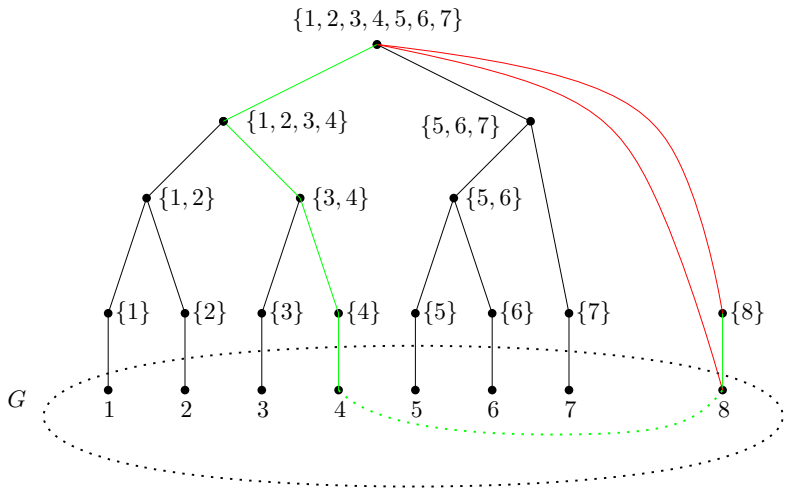

Figure 3.4: Recall the example of G' locally, where $v_1, \ldots, v_7$ are children of v_8 in G. A red edge in an ancestor-certifying path can be replaced by a path like the one in green labelled $Q \cup Q'$.

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 $v_l \in P$ that we move to must satisfy $\pi(v_l) \leqslant \pi(v_i)$. 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. □

Claim 3.5. *For a vertex v_S with $S \subseteq [d]$, if there does not exist v_R with $R \supset S$, then the parent of v_S in T' is $p_T(S)$.*

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 $G \subset G'$, 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. □

□

4 Robust Interior Point Algorithm for General Convex Sets

4.1 Robust Central Path Method

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

$$\min_{Ax=b, x_i \in K_i \text{ for } i \in [m]} c^\top x \tag{4.1}$$

where $x_i \in K_i \subset \mathbb{R}^{n_i}$ and x is the concatenation of x_i lying inside the domain $K \stackrel{\text{def}}{=} \prod_{i=1}^n K_i \subset \mathbb{R}^n$ with $n = \sum_{i=1}^m n_i$. Each step involves solving some linear system of the form $ADA^\top$ 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

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

where ϕ_i are self-concordant barrier functions on K_i. The weight $w \in \mathbb{R}^m_{>0}$ are fixed throughout the algorithm and is defined according to the cost of updating block i.

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

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

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

Since ϕ_i blows up on ∂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 $(\overline{x}, \overline{s})$. Our algorithm takes $O(\sqrt{m})$ steps and each step involves solving some linear system very crudely according to $(\overline{x}, \overline{s})$. Then we call a data structure that implicitly maintains (x, s) and the approximation $(\overline{x}, \overline{s})$ via some random sketching scheme (Section 5).

Definition 4.2 (Induced Norms)**.** For each block, we define $\|v\|_{x_i} \stackrel{\text{def}}{=} \|v\|_{\nabla^2 \phi_i(x_i)}$, $\|v\|_{x_i}^* \stackrel{\text{def}}{=} \|v\|_{(\nabla^2 \phi_i(x_i))^{-1}}$ for $v \in \mathbb{R}^{n_i}$. For the whole domain, we define $\|v\|_x \stackrel{\text{def}}{=} \|v\|_{\nabla^2 \phi(x)} = \sqrt{\sum_i w_i \|v_i\|_{x_i}^2}$ and $\|v\|_x^* \stackrel{\text{def}}{=} \|v\|_{(\nabla^2 \phi(x))^{-1}} = \sqrt{\sum_i w_i^{-1} \|v_i\|_{x_i}^{*2}}$ 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 ripidly, 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 ϕ is a self-concordant barrier and if $\|y-x\|_x < 1$,then we have*

$$(1-\|y-x\|_x)^2\nabla^2\phi(x) \preceq \nabla^2\phi(y) \preceq \frac{1}{(1-\|y-x\|_x)^2}\nabla^2\phi(x).$$

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

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

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

$$\Psi_\lambda(r) \stackrel{\text{def}}{=} \sum_{i=1}^m \cosh(\lambda\frac{r_i}{w_i})$$

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

$$\Phi^t(x,s) = \Psi_\lambda(\gamma^t(x,s)).$$

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 Φ^t and the proof simply shows the potential Φ is bounded throughout.

4.2 Gradient Descent on Ψ_λ

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

$$\delta_r^* = \arg\min_{\sum_i w_i^{-1}\delta_{r,i}^2 \leqslant \alpha^2} \langle\nabla\Psi_\lambda(r), \delta_r\rangle.$$

Using that $\Psi_\lambda(r) = \sum_{i=1}^m \cosh(\lambda\frac{r_i}{w_i})$, we have $\nabla_r\Psi_\lambda(r) = \frac{\lambda}{w_i}\sinh(\frac{\lambda}{w_i}r_i)$ 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 δ_r^* indeed decreases Ψ_λ. Furthermore, this step is robust under ℓ_∞ perturbation of r and ℓ_2 perturbation of δ_r^*.

Lemma 4.5. *Fix any $r \in \mathbb{R}^m$ and $w \in \mathbb{R}^m_{\geq 1}$. Given any $\overline{r} \in \mathbb{R}^m$ such that $|r_i - \overline{r}_i| < \frac{w_i}{8\lambda}$ for all i and*

$$\delta_r = \frac{-\alpha\cdot\sinh(\frac{\lambda}{w_i}\overline{r}_i)}{\sqrt{\sum_j w_j^{-1}\sinh^2(\frac{\lambda}{w_j}\overline{r}_j)}} + \epsilon_r \tag{4.3}$$

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

$$\Psi_\lambda(r+\delta_r) \leqslant \Psi_\lambda(r) - \frac{\alpha}{2}\|\nabla\Psi_\lambda(r)\|_w + \frac{\alpha\lambda}{2}\sqrt{\sum_i w_i^{-1}}.$$

[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

$$\Psi_\lambda(r+\delta_r) = \Psi_\lambda(r) + \langle \nabla\Psi_\lambda(r), \delta_r\rangle + \frac{1}{2}\delta_r^\top \nabla^2\Psi_\lambda(\widetilde{r})\delta_r \tag{4.4}$$

where $\widetilde{r} = r + t\delta_r$ 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

$$\langle \nabla\Psi_\lambda(r), \delta_r - \epsilon_r\rangle = -\alpha\lambda \frac{\sum_i w_i^{-1}\sinh(\frac{\lambda}{w_i}\overline{r}_i)\sinh(\frac{\lambda}{w_i}r_i)}{\sqrt{\sum_j w_j^{-1}\sinh^2(\frac{\lambda}{w_j}\overline{r}_j)}}.$$

Using Lemma A.1 and the assumption $|r_i - \overline{r}_i| < \frac{w_i}{8\lambda}$, we have

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

Hence, we have

$$\begin{aligned}
\langle \nabla\Psi_\lambda(r), \delta_r - \epsilon_r\rangle &\leqslant -\frac{6}{7}\alpha\lambda \frac{\sum_i w_i^{-1}\sinh(\frac{\lambda}{w_i}\overline{r}_i)^2}{\sqrt{\sum_j w_j^{-1}\sinh^2(\frac{\lambda}{w_j}\overline{r}_j)}} + \frac{1}{7}\alpha\lambda \frac{\sum_i w_i^{-1}\left|\sinh(\frac{\lambda}{w_i}\overline{r}_i)\right|}{\sqrt{\sum_j w_j^{-1}\sinh^2(\frac{\lambda}{w_j}\overline{r}_j)}} \\
&= -\frac{6}{7}\alpha\lambda\sqrt{\sum_i w_i^{-1}\sinh^2(\frac{\lambda}{w_i}\overline{r}_i)} + \frac{1}{7}\alpha\lambda\sqrt{\sum_i w_i^{-1}}
\end{aligned} \tag{4.5}$$

Using Lemma A.1 and the assumption $|r_i - \overline{r}_i| < \frac{w_i}{8\lambda}$ again, we have

$$\begin{aligned}
\sqrt{\sum_i w_i^{-1}\sinh^2(\frac{\lambda}{w_i}\overline{r}_i)} &\geq \sqrt{\sum_i w_i^{-1}(\frac{6}{7}|\sinh(\frac{\lambda}{w_i}r_i)| - \frac{1}{7})^2} \\
&\geq \frac{6}{7}\sqrt{\sum_i w_i^{-1}\sinh^2(\frac{\lambda}{w_i}r_i)} - \frac{1}{7}\sqrt{\sum_i w_i^{-1}}.
\end{aligned}$$

Putting this into Eq. (4.5), we have

$$\begin{aligned}
\langle \nabla\Psi_\lambda(r), \delta_r - \epsilon_r\rangle &\leqslant -\frac{36}{49}\alpha\lambda\sqrt{\sum_i w_i^{-1}\sinh^2(\frac{\lambda}{w_i}r_i)} + \frac{2}{7}\alpha\lambda\sqrt{\sum_i w_i^{-1}} \\
&\leqslant -\frac{36}{49}\alpha\|\nabla\Psi_\lambda(r)\|_w + \frac{2}{7}\alpha\lambda\sqrt{\sum_i w_i^{-1}}
\end{aligned} \tag{4.6}$$

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

$$\begin{aligned}
\langle \nabla\Psi_\lambda(r), \epsilon_r\rangle &= \sum_i \frac{\lambda}{w_i}\sinh(\frac{\lambda}{w_i}r_i)\epsilon_{r,i} \\
&\leqslant \lambda\cdot\sqrt{\sum_i w_i^{-1}\sinh^2(\frac{\lambda}{w_i}r_i)}\sqrt{\sum_i w_i^{-1}\epsilon_{r,i}^2} \\
&= \frac{1}{8}\alpha\lambda\sqrt{\sum_i w_i^{-1}\sinh^2(\frac{\lambda}{w_i}r_i)} \\
&\leqslant \frac{1}{8}\alpha\|\nabla\Psi_\lambda(r)\|_w
\end{aligned} \tag{4.7}$$

For the second order term $\delta_r^\top \nabla^2 \Psi_\lambda(\widetilde{r})\delta_r$ in Eq. (4.4), we note that

$$\delta_r^\top \nabla^2 \Psi_\lambda(\widetilde{r})\delta_r = \lambda^2 \sum_i w_i^{-2}\delta_{r,i}^2 \cosh(\lambda \frac{\widetilde{r}_i}{w_i}).$$

Note that

$$\begin{aligned}\sqrt{\sum_i w_i^{-1}\delta_{r,i}^2} &\leqslant \sqrt{\sum_i w_i^{-1}\left(\frac{\alpha \cdot \sinh(\frac{\lambda}{w_i}\overline{r}_i)}{\sqrt{\sum_j w_j^{-1}\sinh^2(\frac{\lambda}{w_j}\overline{r}_j)}}\right)^2} + \sqrt{\sum_i w_i^{-1}\epsilon_{r,i}^2} \\ &\leqslant \alpha + \frac{\alpha}{8} = \frac{9\alpha}{8}. \end{aligned} \tag{4.8}$$

In particular, this shows that $|\delta_{r,i}| \leqslant \frac{9\alpha}{8}\sqrt{w_i} \leqslant \frac{9\alpha}{8} w_i$. Using this and Eq. (4.8), we have

$$\begin{aligned}\delta_r^\top \nabla^2 \Psi_\lambda(\widetilde{r})\delta_r &= \lambda^2 \sum_i w_i^{-2}\delta_{r,i}^2 \cosh(\lambda\frac{\widetilde{r}_i}{w_i}) \\ &\leqslant \frac{9\alpha}{8}\lambda^2 \sum_i w_i^{-1}|\delta_{r,i}| \cosh(\lambda\frac{\widetilde{r}_i}{w_i}) \\ &\leqslant \frac{9\alpha}{8}\lambda^2 \sqrt{\sum_i w_i^{-1}\delta_{r,i}^2}\sqrt{\sum_i w_i^{-1}\cosh^2(\lambda\frac{\widetilde{r}_i}{w_i})} \\ &\leqslant (\frac{9\alpha}{8})^2\lambda^2\left(\sqrt{\sum_i w_i^{-1}\sinh^2(\lambda\frac{\widetilde{r}_i}{w_i})} + \sqrt{\sum_i w_i^{-1}}\right) \\ &\leqslant (\frac{9\alpha}{8})^2\lambda^2\left(\frac{8}{7}\sqrt{\sum_i w_i^{-1}\sinh^2(\lambda\frac{r_i}{w_i})} + \frac{8}{7}\sqrt{\sum_i w_i^{-1}}\right) \\ &\leqslant (\frac{9\alpha}{8})^2\frac{8}{7}\left(\lambda\|\nabla\Psi_\lambda(r)\|_w + \lambda^2\sqrt{\sum_i w_i^{-1}}\right) \end{aligned} \tag{4.9}$$

where we used $\cosh(x) \leqslant 1 + |\sinh(x)|$ 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

$$\begin{aligned}\Psi_\lambda(r+\delta) =& \Psi_\lambda(r) + \langle\nabla\Psi_\lambda(r),\delta\rangle + \frac{1}{2}\delta^\top\nabla_\lambda^2(\widetilde{r})\delta \\ \leqslant& \Psi_\lambda(r) - \frac{36}{49}\alpha\|\nabla\Psi_\lambda(r)\|_w + \frac{2}{7}\alpha\lambda\sqrt{\sum_i w_i^{-1}} + \frac{1}{8}\alpha\|\nabla\Psi_\lambda(r)\|_w \\ &+ \frac{1}{2}(\frac{9\alpha}{8})^2\frac{8}{7}\lambda\|\nabla\Psi_\lambda(r)\|_w + \frac{1}{2}(\frac{9\alpha}{8})^2\frac{8}{7}\lambda^2\sqrt{\sum_i w_i^{-1}}. \end{aligned}$$

Using $\alpha \leqslant \frac{1}{8\lambda}$, we can simplify it to

$$\begin{aligned}\Psi_\lambda(r+\delta) &\leqslant \Psi_\lambda(r) - \left(\frac{36}{49} - \frac{1}{8} - \frac{1}{2}(\frac{9}{8})^2\frac{1}{7}\right)\alpha\|\nabla\Psi_\lambda(r)\|_w + \left(\frac{2}{7} + \frac{1}{2}(\frac{9}{8})^2\frac{1}{7}\right)\alpha\lambda\sqrt{\sum_i w_i^{-1}} \\ &\leqslant \Psi_\lambda(r) - \frac{\alpha}{2}\|\nabla\Psi_\lambda(r)\|_w + \frac{\alpha\lambda}{2}\sqrt{\sum_i w_i^{-1}}. \end{aligned}$$

□

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 μ. In particular, the solution x is the solution of the optimization problem $\min_{Ax=b} c^\top x + t\sum_{i=1}^m w_i\phi_i(x_i) - t\mu^\top x$. Hence, we can move μ 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:

$$\delta_\mu^* = \arg\min_{\|\delta_\mu\|_x^*=\alpha} \langle \nabla_\mu \Psi_\lambda(\|\mu_i\|_{x_i}^*), \mu + \delta_\mu\rangle \tag{4.10}$$

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

$$\nabla_{\mu_i}\Psi_\lambda(\|\mu_i\|_{x_i}^*) = \frac{\lambda\sinh(\frac{\lambda}{w_i}\|\mu_i\|_{x_i}^*)}{w_i\|\mu_i\|_{x_i}^*}\cdot\nabla\phi_i(x_i)^{-1}\mu_i$$

Solve Eq. (4.10)[2], we get

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

To move μ to $\mu+\delta_\mu$ approximately, we take Newton step (δ_x^*, δ_s^*)[3]:

$$\begin{aligned}\frac{1}{t}\delta_s^* + \nabla^2\phi(x)\delta_x^* &= \delta_\mu^*,\\ A\delta_x^* &= 0,\\ A^\top\delta_y^* + \delta_s^* &= 0.\end{aligned}$$

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

$$\begin{aligned}\delta_x^* &= H_x^{-1}\delta_\mu^* - H_x^{-1}A^\top(AH_x^{-1}A^\top)^{-1}AH_x^{-1}\delta_\mu^*,\\ \delta_s^* &= tA^\top(AH_x^{-1}A^\top)^{-1}AH_x^{-1}\delta_\mu^*.\end{aligned}$$

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

$$\begin{aligned}\delta_x^* &= H_x^{-1/2}(I-P_x)H_x^{-1/2}\delta_\mu^*,\\ \delta_s^* &= tH_x^{1/2}P_xH_x^{-1/2}\delta_\mu^*.\end{aligned}$$

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

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

$$\begin{aligned}\delta_x &= H_{\overline{x}}^{-1/2}(I-P_{\overline{x}})H_{\overline{x}}^{-1/2}\delta_\mu,\\ \delta_s &= tH_{\overline{x}}^{1/2}P_{\overline{x}}H_{\overline{x}}^{-1/2}\delta_\mu\end{aligned}$$

and

[2]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 $\min_{Ax=b, A^\top y+s=c}\Phi^t(x,s)$. 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.

[3]We use the $*$ to highlight this is the ideal step and to distinguish with the step we will take..

- $H_{\overline{x}}$ is a block diagonal matrix with the (i,i) block given by $w_i \nabla^2 \phi_i(\overline{x}_i)$,
- $P_{\overline{x}}$ is the projection matrix $H_{\overline{x}}^{-1/2} A^\top (A H_{\overline{x}}^{-1} A^\top)^{-1} A H_{\overline{x}}^{-1/2}$,
- δ_μ is the proposed step of μ defined by $-\alpha \cdot c_i^t(\overline{x}, \overline{s}) \cdot \mu_i^t(\overline{x}, \overline{s})$ with $c_i^t(\overline{x}, \overline{s}) = \frac{\sinh(\frac{\lambda}{w_i}\gamma_i^t(\overline{x}, \overline{s}))}{\gamma_i^t(\overline{x}, \overline{s}) \cdot \sqrt{\sum_{j=1}^m w_j^{-1} \sinh^2(\frac{\lambda}{w_i}\gamma_j^t(\overline{x}, \overline{s}))}}$ (See Definition 4.4 for the definition of γ_i and μ_i),
- α is the step size.

Formally, our algorithm is defined in Algorithm 9.

4.4 Bounding Φ under changes of x and s

In this section, we will use the following assumptions.

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

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

To use Lemma 4.5 to bound the potential, we need to verify Eq. (4.3) and that $|\gamma_i^t(x^{\mathrm{new}}, s^{\mathrm{new}}) - \gamma_i(x,s)| \leqslant \frac{w_i}{8\lambda}$.

4.4.1 Verifying conditions of Lemma 4.5

Recall that the ideal step we want to take is

$$\delta_{\mu,i}^* = -\alpha \cdot c_i^t(x,s) \cdot \mu_i^t(x,s).$$

A rough calculation shows

$$\begin{aligned}
\gamma_i^t(x^{\mathrm{new}}, s^{\mathrm{new}}) &= \|\mu_i + \delta_{\mu,i}^*\|_{x_i}^* \\
&\sim \|\mu_i\|_{x_i}^* - \frac{\alpha}{\|\mu_i\|_{x_i}^*} \cdot c_i^t(x,s) \cdot \mu_i^\top \nabla^2 \phi_i(x)^{-1} \mu_i \\
&= \gamma_i^t(x,s) - \alpha \cdot c_i^t(x,s) \cdot \gamma_i^t(x,s)
\end{aligned}$$

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

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

$$\sum_{i=1}^m w_i \alpha_i^2 \leqslant \alpha^2.$$

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

Proof. We have

$$\sum_{i=1}^m w_i \alpha_i^2 = \|\delta_x\|_{\overline{x}}^2 = \|(I - P_{\overline{x}}) H_{\overline{x}}^{-1/2} \delta_\mu\|_2^2 \leqslant \|H_{\overline{x}}^{-1/2} \delta_\mu\|_2^2 = \|\delta_\mu\|_{\overline{x}}^{*2} = \alpha^2,$$

where the first inequality follows by $I - P_{\overline{x}}$ is an orthogonal projection matrix and the last equality follows by the step size for δ_μ. □

To bound the change of γ, we first show that μ^{new} is close to $\mu + \delta_\mu$.

Lemma 4.9 (Change in μ). *Under Assumption 4.7, let*

$$\mu_i^t(x^{new}, s^{new}) = \mu_i^t(x, s) + \delta_{\mu,i} + w_i \epsilon_i^{(\mu)}.$$

Then, $\|\epsilon_i^{(\mu)}\|_{x_i}^* \leqslant 6\bar{\epsilon}\alpha_i$.

Proof. By definition of μ, we have

$$\begin{aligned}
\mu_i(x^{\text{new}}, s^{\text{new}}) &= \frac{s_i^{\text{new}}}{t} + w_i \nabla\phi_i(x^{\text{new}}) \\
&= \mu_i(x, s) + \frac{1}{t}\delta_s + w_i(\nabla\phi_i(x^{\text{new}}) - \nabla\phi_i(x_i)) \\
&= \mu_i(x, s) + \delta_{\mu,i} + w_i \underbrace{(\nabla\phi_i(x^{\text{new}}) - \nabla\phi_i(x_i) - \nabla^2\phi_i(\overline{x}_i)\delta_x)}_{\epsilon_i^{(\mu)}},
\end{aligned}$$

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

To bound $\epsilon_i^{(\mu)}$, let $x^{(u)} = ux^{\text{new}} + (1-u)x$, then we have

$$\begin{aligned}
\epsilon_i^{\mu} &= \nabla\phi_i(x_i^{\text{new}}) - \nabla\phi_i(x_i) - \nabla^2\phi_i(\overline{x}_i)\delta_{x,i} \\
&= \int_0^1 \left(\nabla^2\phi_i(x_i^{(u)}) - \nabla^2\phi_i(\overline{x}_i)\right)\delta_{x,i}\mathrm{d}u.
\end{aligned}$$

By Lemma 4.3, we have

$$(1 - \|x_i^{(u)} - \overline{x}_i\|_{\overline{x}_i})^2\nabla^2\phi_i(\overline{x}_i) \preceq \nabla^2\phi_i(x^{(u)}) \preceq \frac{1}{(1 - \|x_i^{(u)} - \overline{x}_i\|_{\overline{x}_i})^2}\nabla^2\phi_i(\overline{x}_i). \tag{4.11}$$

Note that

$$\|x_i^{(u)} - \overline{x}_i\|_{\overline{x}_i} \leqslant \|x_i^{(u)} - x_i\|_{\overline{x}_i} + \|x_i - \overline{x}_i\|_{\overline{x}_i} = u\|\delta_{x,i}\|_{\overline{x}_i} + \bar{\epsilon} \leqslant \alpha_i + \bar{\epsilon} \leqslant \alpha + \bar{\epsilon} \leqslant 2\bar{\epsilon},$$

where the last inequality follows by $\alpha_i \leqslant \alpha$ (Lemma 4.8) and $\alpha \leqslant \bar{\epsilon}$ (Assumption 4.7). Combine two inequalities above and using that $\bar{\epsilon} \leqslant \frac{1}{8}$, we get

$$-5\bar{\epsilon}\nabla^2\phi_i(\overline{x}_i) \preceq \nabla^2\phi_i(x^{(u)}) - \nabla^2\phi_i(\overline{x}_i) \preceq 5\bar{\epsilon}\nabla^2\phi_i(\overline{x}_i).$$

Using this, Eq. (4.11) and Assumption 4.7, we have

$$\begin{aligned}
&\left(\nabla^2\phi_i(x^{(u)}) - \nabla^2\phi_i(\overline{x}_i)\right)\left(\nabla^2\phi_i(\overline{x}_i)\right)^{-1}\left(\nabla^2\phi_i(x^{(u)}) - \nabla^2\phi_i(\overline{x}_i)\right) \\
\preceq &\frac{1}{(1-1/10)^2}\left(\nabla^2\phi_i(x^{(u)}) - \nabla^2\phi_i(\overline{x}_i)\right)\left(\nabla^2\phi_i(x_i)\right)^{-1}\left(\nabla^2\phi_i(x^{(u)}) - \nabla^2\phi_i(\overline{x}_i)\right) \\
\preceq &(6\bar{\epsilon})^2\nabla^2\phi_i(\overline{x}_i)
\end{aligned}$$

This implies

$$\begin{aligned}
\left\|\left(\nabla^2\phi_i(x^{(u)}) - \nabla^2\phi_i(x_i)\right)\delta_{x,i}\right\|_{x_i}^* &= \sqrt{\delta_{x,i}^\top\left(\nabla^2\phi_i(x^{(u)}) - \nabla^2\phi_i(\overline{x}_i)\right)^\top\left(\nabla^2\phi_i(x_i)\right)^{-1}\left(\nabla^2\phi_i(x^{(u)}) - \nabla^2\phi_i(\overline{x}_i)\right)\delta_{x,i}} \\
&\leqslant \sqrt{(6\bar{\epsilon})^2\delta_{x,i}^\top\nabla^2\phi_i(\overline{x}_i)\delta_{x,i}^\top} \\
&= 6\bar{\epsilon}\|\delta_{x,i}\|_{\overline{x}_i} \\
&= 6\bar{\epsilon}\alpha_i.
\end{aligned}$$

Hence,

$$\|\epsilon_i^{(\mu)}\|_{x_i}^* \leqslant \int_0^1 \left\|\left(\nabla^2\phi_i(x^{(u)}) - \nabla^2\phi_i(\overline{x}_i)\right)\delta_{x,i}\right\|_{x_i}^* \mathrm{d}u \leqslant 6\bar{\epsilon}\alpha_i.$$

□

To get our bound for γ, we first need two helper lemmas. One is to show $\mu(x,s)$ is close to $\mu(\overline{x},\overline{s})$.

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

$$\|\mu_i^t(x,s) - \mu_i^t(\overline{x},\overline{s})\|_{x_i}^* \leqslant 4w_i\bar{\epsilon}.$$

Furthermore, we have that $|\gamma_i^t(x,s) - \gamma_i^t(\overline{x},\overline{s})| \leqslant 3w_i\bar{\epsilon} + 2\bar{\epsilon}\gamma_i^t(x,s)$.

Proof. For the first result, note that

$$\|\mu_i^t(x,s) - \mu_i^t(\overline{x},\overline{s})\|_{\overline{x}_i}^* \leqslant \frac{1}{t}\|s_i - \overline{s}_i\|_{\overline{x}_i}^* + w_i\|\nabla\phi_i(x_i) - \nabla\phi_i(\overline{x}_i)\|_{\overline{x}_i}^*.$$

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

$$\|\nabla\phi_i(x_i) - \nabla\phi_i(\overline{x}_i)\|_{\overline{x}_i}^* = \left\|\int_0^1 \nabla^2\phi_i(x_i^{(u)})(x_i - \overline{x}_i)\mathrm{d}u\right\|_{\overline{x}_i}^* \leqslant 2\|x - \overline{x}_i\|_{\overline{x}_i} \leqslant 2\bar{\epsilon}.$$

Hence, $\|\mu_i^t(x,s) - \mu_i^t(\overline{x},\overline{s})\|_{\overline{x}_i}^* \leqslant 3w_i\bar{\epsilon}$ and hence the result.

For the second result, note that

$$\begin{aligned}
|\gamma_i^t(x,s) - \gamma_i^t(\overline{x},\overline{s})| &\leqslant \|\mu_i^t(x,s) - \mu_i^t(\overline{x},\overline{s})\|_{\overline{x}_i}^* + |\|\mu_i^t(x,s)\|_{x_i}^* - \|\mu_i^t(x,s)\|_{\overline{x}_i}^*| \\
&\leqslant 3w_i\bar{\epsilon} + 2\|x_i - \overline{x}_i\|_{\overline{x}_i}\|\mu_i^t(x,s)\|_{x_i}^* \\
&= 3w_i\bar{\epsilon} + 2\bar{\epsilon}\gamma_i^t(x,s)
\end{aligned}$$

where we used Assumption 4.7 and Lemma 4.3. □

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

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

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

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

Proof. For the first inequality, we note that

$$\begin{aligned}
\sum_{i=1}^m w_i^{-1}\sinh^2(\frac{\lambda}{w_i}\gamma_i^t(\overline{x},\overline{s})) &\geq \frac{(\sum_{i=1}^m|\sinh(\frac{\lambda}{w_i}\gamma_i^t(\overline{x},\overline{s}))|)^2}{\sum_{i=1}^m w_i} \\
&\geq \frac{(\sum_{i=1}^m(\cosh(\frac{\lambda}{w_i}\gamma_i^t(\overline{x},\overline{s})) - 1))^2}{\sum_{i=1}^m w_i} \\
&= \frac{(\Phi^t(\overline{x},\overline{s}) - m)^2}{\sum_{i=1}^m w_i} \\
&\geq \frac{4(\sum_{i=1}^m w_i)^2}{\sum_{i=1}^m w_i} = 4\sum_{i=1}^m w_i
\end{aligned} \tag{4.12}$$

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

For the second inequality, we note that

$$c_i^t(\overline{x},\overline{s}) = \frac{\sinh(\frac{\lambda}{w_i}\gamma_i^t(\overline{x},\overline{s}))}{\gamma_i^t(\overline{x},\overline{s})\cdot\sqrt{\sum_{j=1}^m w_j^{-1}\sinh^2(\frac{\lambda}{w_i}\gamma_j^t(\overline{x},\overline{s}))}}.$$

Since $\gamma_i \geq 0$ (by definition), we have $c_i^t \geq 0$.

If $\gamma_i^t(\overline{x},\overline{s}) \geq \frac{w_i}{\lambda}$, we have that

$$c_i^t(\overline{x},\overline{s}) \leqslant \frac{\sqrt{w_i}\sinh(\frac{\lambda}{w_i}\gamma_i^t(\overline{x},\overline{s}))}{\gamma_i^t(\overline{x},\overline{s})\cdot\sinh(\frac{\lambda}{w_i}\gamma_j^t(\overline{x},\overline{s}))} \leqslant \frac{\lambda}{\sqrt{w_i}} \leqslant \lambda$$

where we used that $w_i \geq 1$. If $\gamma_i^t(\overline{x},\overline{s}) \leqslant \frac{w_i}{\lambda}$, we use that $|\sinh(x)| \leqslant 2|x|$ for all $|x| \leqslant 1$ and get

$$\begin{aligned} c_i^t(\overline{x},\overline{s}) &\leqslant \frac{2\frac{\lambda}{w_i}\gamma_i^t(\overline{x},\overline{s})}{\gamma_i^t(\overline{x},\overline{s})\cdot\sqrt{\sum_{j=1}^m w_j^{-1}\sinh^2(\frac{\lambda}{w_i}\gamma_j^t(\overline{x},\overline{s}))}} \\ &\leqslant \frac{2\lambda}{w_i\sqrt{4\sum_{j=1}^m w_j}} \leqslant \lambda \end{aligned}$$

where we used Eq. (4.12) at the end.

For the third inequality, we note that $\cosh(\lambda) \geq \Phi^t(x,s)$ implies that $\gamma_i^t(\overline{x},\overline{s}) \leqslant w_i$ for all i. Hence, Lemma 4.10 shows that

$$|\gamma_i^t(x,s) - \gamma_i^t(\overline{x},\overline{s})| \leqslant 3w_i\bar{\epsilon} + 2\bar{\epsilon}\gamma_i^t(x,s \leqslant 5w_i\bar{\epsilon} \leqslant \frac{5w_i}{8\lambda}.$$

Since $\cosh(x-\frac{5}{8}) \geq \frac{1}{2}\cosh(x)$ for all x, this shows

$$\begin{aligned} \Phi^t(\overline{x},\overline{s}) &= \sum_{i=1}^m \cosh(\frac{\lambda}{w_i}\gamma_i^t(\overline{x},\overline{s})) \\ &\geq \sum_{i=1}^m \cosh(\frac{\lambda}{w_i}\gamma_i^t(x,s) - \frac{5}{8}) \\ &\geq \frac{1}{2}\sum_{i=1}^m \cosh(\frac{\lambda}{w_i}\gamma_i^t(x,s)) \\ &= \frac{1}{2}\Phi^t(x,s). \end{aligned}$$

□

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

Lemma 4.12 (Change in γ)**.** *Under Assumption 4.7, for all $i \in [m]$, let*

$$\epsilon_{r,i} \stackrel{\text{def}}{=} \gamma_i^t(x^{new},s^{new}) - \gamma_i^t(x,s) + \alpha\cdot c_i^t(\overline{x},\overline{s})\cdot\gamma_i^t(\overline{x},\overline{s}).$$

Assuming $\Phi^t(\overline{x},\overline{s}) \geq 3\sum_{i=1}^m w_i$, we have

$$\sum_{i=1}^m w_i^{-1}\epsilon_{r,i}^2 \leqslant (310\bar{\epsilon}^2 + 48\max_i(w_i^{-1}\gamma_i^t(x,s))^2)\alpha^2$$

Proof. For notation simplicity, we write $\overline{c}_i = c_i^t(\overline{x},\overline{s})$. Also, we use $\gamma_i^t(x,z,s)$ to denote $\|\mu_i(x,s)\|_{z_i}^*$. Using $\delta_{\mu,i} = -\alpha\cdot\overline{c}_i\cdot\mu_i^t(\overline{x},\overline{s})$, we have

$$\begin{aligned} \gamma_i^t(x^{\text{new}},x,s^{\text{new}}) =& \|\mu_i(x,s) + \delta_{\mu,i} + w_i\epsilon_i^{(\mu)}\|_{x_i}^* \\ =& \|\mu_i(x,s) - \alpha\cdot\overline{c}_i\cdot\mu_i^t(\overline{x},\overline{s})\|_{x_i}^* \pm w_i\|\epsilon_i^{(\mu)}\|_{x_i}^* \\ =& \|\mu_i(x,s) - \alpha\cdot\overline{c}_i\cdot\mu_i^t(x,s)\|_{x_i}^* \\ & \pm \alpha\cdot\overline{c}_i\cdot\|\mu_i^t(x,s) - \mu_i^t(\overline{x},\overline{s})\|_{x_i}^* \pm w_i\|\epsilon_i^{(\mu)}\|_{x_i}^* \\ =& (1-\alpha\cdot\overline{c}_i)\gamma_i^t(x,s) \pm \alpha\cdot\overline{c}_i\cdot\|\mu_i^t(x,s) - \mu_i^t(\overline{x},\overline{s})\|_{x_i}^* \pm w_i\|\epsilon_i^{(\mu)}\|_{x_i}^* \end{aligned} \tag{4.13}$$

where we used that $0 \leqslant \alpha \cdot \bar{c}_i \leqslant 1$ at the end (Lemma 4.11).

In particular, we have that

$$\begin{aligned}\gamma_i^t(x^{\mathrm{new}}, x, s^{\mathrm{new}}) &\leqslant \gamma_i^t(x,s) + \alpha \cdot \bar{c}_i \cdot \|\mu_i^t(x,s) - \mu_i^t(\overline{x},\overline{s})\|_{x_i}^* + w_i \|\epsilon_i^{(\mu)}\|_{x_i}^* \\ &\leqslant \gamma_i^t(x,s) + \bar{\epsilon} w_i (4\alpha\bar{c}_i + 6\alpha_i) \end{aligned} \tag{4.14}$$

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

$$\begin{aligned}\left|\gamma_i^t(x^{\mathrm{new}}, x^{\mathrm{new}}, s^{\mathrm{new}}) - \gamma_i^t(x^{\mathrm{new}}, x, s^{\mathrm{new}})\right| &= \left|\|\mu_i^t(x^{\mathrm{new}}, s^{\mathrm{new}})\|_{x_i^{\mathrm{new}}} - \|\mu_i^t(x^{\mathrm{new}}, s^{\mathrm{new}})\|_{x_i}\right| \\ &\leqslant 2\|x_i^{\mathrm{new}} - x_i\|_{x_i} \|\mu_i^t(x^{\mathrm{new}}, s^{\mathrm{new}})\|_{x_i} \\ &\leqslant 4\|\delta_{x,i}\|_{\overline{x}_i} \gamma_i^t(x^{\mathrm{new}}, x, s^{\mathrm{new}}) \\ &= 4\alpha_i \gamma_i^t(x^{\mathrm{new}}, x, s^{\mathrm{new}}) \\ &\leqslant 4\alpha_i \gamma_i^t(x,s) + \bar{\epsilon}\alpha_i w_i (16\alpha\bar{c}_i + 24\alpha_i) \end{aligned} \tag{4.15}$$

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

Using Eq. (4.13), we have

$$\begin{aligned}&\left|\gamma_i^t(x^{\mathrm{new}}, x, s^{\mathrm{new}}) - \gamma_i^t(x,s) + \alpha \cdot \bar{c}_i \cdot \gamma_i^t(\overline{x},\overline{s})\right| \\ \leqslant &|(1 - \alpha\cdot\bar{c}_i)\gamma_i^t(x,s) - \gamma_i^t(x,s) + \alpha\cdot\bar{c}_i\cdot\gamma_i^t(\overline{x},\overline{s})| \\ &+ \alpha \cdot \bar{c}_i \cdot \|\mu_i^t(x,s) - \mu_i^t(\overline{x},\overline{s})\|_{x_i}^* + w_i\|\epsilon_i^{(\mu)}\|_{x_i}^* \\ \leqslant &2\alpha \cdot \bar{c}_i \cdot \|\mu_i^t(x,s) - \mu_i^t(\overline{x},\overline{s})\|_{x_i}^* + w_i\|\epsilon_i^{(\mu)}\|_{x_i}^* \\ \leqslant &8\bar{\epsilon} w_i(\alpha\bar{c}_i + \alpha_i) \end{aligned} \tag{4.16}$$

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

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

$$\begin{aligned}|\epsilon_{r,i}| &\leqslant \left|\gamma_i^t(x^{\mathrm{new}}, x, s^{\mathrm{new}}) - \gamma_i^t(x,s) - \alpha\cdot\bar{c}_i\cdot\gamma_i^t(\overline{x},\overline{s})\right| + \left|\gamma_i^t(x^{\mathrm{new}}, x^{\mathrm{new}}, s^{\mathrm{new}}) - \gamma_i^t(x^{\mathrm{new}}, x, s^{\mathrm{new}})\right| \\ &\leqslant 8\bar{\epsilon} w_i(\alpha\bar{c}_i + \alpha_i) + 4\alpha_i\gamma_i^t(x,s) + \bar{\epsilon}\alpha_i w_i(16\alpha\bar{c}_i + 24\alpha_i) \\ &\leqslant 10\bar{\epsilon} w_i(\alpha\bar{c}_i + \alpha_i) + 4\alpha_i\gamma_i^t(x,s). \end{aligned} \tag{4.17}$$

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

Now, we bound the ℓ_2 norm of v, we first note that

$$\begin{aligned}\sum_{i=1}^m w_i \bar{c}_i^2 &= \frac{\sum_{i=1}^m w_i \frac{\sinh^2(\frac{\lambda}{w_i}\gamma_i^t(\overline{x},\overline{s}))}{\gamma_i^t(\overline{x},\overline{s})^2}}{\sum_{j=1}^m w_j^{-1}\sinh^2(\frac{\lambda}{w_i}\gamma_j^t(\overline{x},\overline{s}))} \\ &= \lambda^2 \frac{\sum_{i=1}^m w_i^{-1}\frac{w_i^2}{\lambda^2\gamma_i^t(\overline{x},\overline{s})^2}\sinh^2(\frac{\lambda}{w_i}\gamma_i^t(\overline{x},\overline{s}))}{\sum_{j=1}^m w_j^{-1}\sinh^2(\frac{\lambda}{w_i}\gamma_j^t(\overline{x},\overline{s}))} \\ &\leqslant \lambda^2 \frac{\sum_{i=1}^m w_i^{-1}(1 + \sinh^2(\frac{\lambda}{w_i}\gamma_i^t(\overline{x},\overline{s})))}{\sum_{j=1}^m w_j^{-1}\sinh^2(\frac{\lambda}{w_i}\gamma_j^t(\overline{x},\overline{s}))} \\ &\leqslant 2\lambda^2 \end{aligned}$$

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

Lemma 4.11 at the last inequality. Using this and $\sum_i w_i\alpha_i^2 \leqslant \alpha^2$ (Lemma 4.8) into Eq. (4.17), we have

$$\begin{aligned}\sum_{i=1}^m w_i^{-1}\epsilon_{r,i}^2 &\leqslant 300\overline{\epsilon}^2\alpha^2\sum_{i=1}^m w_i\overline{c}_i^2 + 300\overline{\epsilon}^2\sum_{i=1}^m w_i\alpha_i^2 + 48\max_i(w_i^{-1}\gamma_i^t(x,s))^2\sum_{i=1}^m w_i\alpha_i^2\\ &\leqslant 600\overline{\epsilon}^2\alpha^2\lambda^2 + (300\overline{\epsilon}^2 + 48\max_i(w_i^{-1}\gamma_i^t(x,s))^2)\alpha^2\\ &\leqslant (310\overline{\epsilon}^2 + 48\max_i(w_i^{-1}\gamma_i^t(x,s))^2)\alpha^2\end{aligned}$$

where we used that $\alpha \leqslant \frac{1}{200\lambda}$ at the end. □

4.4.2 Bounding the Movement of Φ

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

$$\Phi^t(x^{new},s^{new}) \leqslant (1 - \frac{\alpha\lambda}{2\sqrt{\sum_{i=1}^m w_i}})\Phi^t(x,s) + \frac{\alpha\lambda}{2}\sqrt{\sum_i w_i^{-1}}.$$

Proof. Let $r_i = \gamma_i^t(x,s)$, $\overline{r}_i = \gamma_i^t(\overline{x},\overline{s})$ and $\delta_{r,i} = \gamma_i^t(x^{\mathrm{new}},s^{\mathrm{new}}) - \gamma_i^t(\overline{x},\overline{s})$. Now, we verify the conditions in Lemma 4.5 for r_i, $\overline{r}_i$ and δ_r. Lemma Lemma 4.10 shows that

$$|r_i - \overline{r}_i| \leqslant 3w_i\overline{\epsilon} + 2\overline{\epsilon}\gamma_i^t(x,s).$$

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

$$|r_i - \overline{r}_i| \leqslant 4w_i\overline{\epsilon} \leqslant \frac{w_i}{8\lambda}$$

where we used the assumption $\overline{\epsilon} \leqslant \frac{1}{200\lambda}$.

Next, Lemma 4.12 shows that

$$\delta_{r,i} = -\alpha\cdot c_i^t(\overline{x},\overline{s})\cdot\gamma_i^t(\overline{x},\overline{s}) + \epsilon_{r,i}$$

with

$$\begin{aligned}\sum_{i=1}^m w_i^{-1}\epsilon_{r,i}^2 &\leqslant (310\overline{\epsilon}^2 + 48\max_i(w_i^{-1}\gamma_i^t(x,s))^2)\alpha^2\\ &\leqslant \left(\frac{310}{(200)^2} + \frac{48}{(200)^2}\right)\alpha^2 \leqslant (\frac{\alpha}{8})^2\end{aligned}$$

where we used $|\gamma_i^t(x,s)| \leqslant \frac{w_i}{200}$ and $\overline{\epsilon} \leqslant \frac{1}{200\lambda}$. Using the formula of $c_i^t(\overline{x},\overline{s})$, we have

$$\begin{aligned}\delta_{r,i} &= -\alpha\cdot\frac{\sinh(\frac{\lambda}{w_i}\gamma_i^t(\overline{x},\overline{s}))}{\sqrt{\sum_{j=1}^m w_j^{-1}\sinh^2(\frac{\lambda}{w_i}\gamma_j^t(\overline{x},\overline{s}))}} + \epsilon_{r,i}\\ &= \frac{-\alpha\sinh(\frac{\lambda}{w_i}\overline{r}_i)}{\sqrt{\sum_{j=1}^m w_j^{-1}\sinh^2(\frac{\lambda}{w_i}\overline{r}_i)}} + \epsilon_{r,i}\end{aligned}$$

and this exactly satisfies the conditions in Lemma 4.5.

Now, Lemma 4.5 shows that

$$\begin{aligned}\Phi^t(x^{\mathrm{new}},s^{\mathrm{new}}) &\leqslant \Phi^t(x,s) - \frac{\alpha\lambda}{2}\sqrt{\sum_{i=1}^m w_i^{-1}\sinh^2(\frac{\lambda}{w_i}\gamma_i^t(x,s))} + \frac{\alpha\lambda}{2}\sqrt{\sum_i w_i^{-1}}\\ &\leqslant \Phi^t(x,s) - \frac{\alpha\lambda}{2}\frac{\sum_{i=1}^m|\sinh(\frac{\lambda}{w_i}\gamma_i^t(x,s))|}{\sqrt{\sum_{i=1}^m w_i}} + \frac{\alpha\lambda}{2}\sqrt{\sum_i w_i^{-1}}\\ &\leqslant (1 - \frac{\alpha\lambda}{2\sqrt{\sum_{i=1}^m w_i}})\Phi^t(x,s) + \frac{\alpha\lambda}{2}\sqrt{\sum_i w_i^{-1}}.\end{aligned}$$

□

Lemma 4.14 (Change in Φ by t). *Let $t^{new} \leftarrow (1-h)t$, for $h \leqslant \frac{1}{8\lambda\sqrt{\max \nu_i}}$ and $h \leqslant \frac{1}{2\lambda\|\gamma/w\|_\infty}$, then*

$$\Phi^{t^{new}}(x,s) \leqslant (1+4h\lambda(\|\sqrt{\nu}\|_\infty + \|\gamma_i^t/w_i\|_\infty)\Phi^t(x,s).$$

For all $i \in [m]$, let $t^{new} \leftarrow (1-h)t$, for $h \leqslant \frac{1}{8\lambda\sqrt{\max \nu_i}}$ and $h \leqslant \frac{1}{2\lambda\|\gamma/w\|_\infty}$,

$$\Phi^{t^{new}}(x,s) \leqslant \Phi^t(x,s) + (1+4h\lambda(\|\sqrt{\nu}\|_\infty + \|\gamma_i^t/w_i\|_\infty))\Phi^t(x,s).$$

Proof. By definition of γ, we have

$$\begin{aligned}
\gamma_i^{t^{\text{new}}}(x,s) &= \|\frac{s_i}{t^{\text{new}}} + w_i\nabla\phi_i(x_i)\|_{x_i}^* \\
&= \|\frac{s}{t(1-h)} + w_i\nabla\phi_i(x_i)\|_{x_i}^* \\
&\leqslant (1+2h)\gamma_i^t + 2hw_i\sqrt{\nu_i} \\
&= \gamma_i^t + 2h(\gamma_i^t + w_i\nu_i),
\end{aligned}$$

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

Then, for $\Phi^{t^{\text{new}}}$,we have

$$\begin{aligned}
\Phi^{t^{\text{new}}}(x,s) &= \sum_{i=1}^m \cosh(\lambda\gamma_i^{t^{\text{new}}}/w_i) \\
&\leqslant \sum_{i=1}^m \cosh(\lambda\gamma_i^t/w_i + 2h\lambda(\gamma_i^t/w_i + \sqrt{\nu_i}) \\
&= \sum_{i=1}^m \cosh(\lambda\gamma_i^t/w_i)(1+4h\lambda(\gamma_i^t/w_i + \sqrt{\nu_i})) \\
&\leqslant (1+4h\lambda(\|\sqrt{\nu}\|_\infty + \|\gamma_i^t/w_i\|_\infty))\Phi^t(x,s),
\end{aligned}$$

where the last inequality follows by Lemma A.2. □

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

$$\Phi^{t^{\text{new}}}(x^{\text{new}},s^{\text{new}}) \leqslant (1-\frac{\alpha\lambda}{4\sqrt{\sum_{i=1}^m w_i}})\Phi^t(x,s) + \frac{\alpha\lambda}{2}\sqrt{\sum_i w_i^{-1}} \leqslant (1-\frac{\alpha\lambda}{4\sqrt{\sum_{i=1}^m w_i}})\Phi^t(x,s) + \frac{\alpha\lambda}{2}\sqrt{m}.$$

Proof. By Lemma 4.14, we have

$$\begin{aligned}
\Phi^{t^{\text{new}}}(x^{\text{new}},s^{\text{new}}) &\leqslant (1-\frac{\alpha\lambda}{2\sqrt{\sum_{i=1}^m w_i}})\Phi^{t^{\text{new}}}(x,s) + \frac{\alpha\lambda}{2}\sqrt{\sum_i w_i^{-1}} \\
&\leqslant (1+4h\lambda(\|\sqrt{\nu}\|_\infty + \|\gamma_i^t/w_i\|_\infty))(1-\frac{\alpha\lambda}{2\sqrt{\sum_{i=1}^m w_i}})\Phi^t(x,s) + \frac{\alpha\lambda}{2}\sqrt{\sum_i w_i^{-1}}
\end{aligned}$$

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

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

this implies $\|\gamma_i^t(x,s)/w_i\|_\infty \leqslant \frac{\operatorname{arcosh}(10m^3/\alpha)}{\lambda} \leqslant \frac{3\log(10m^3/\alpha)}{\lambda}$. By our choice of λ and α, we have $\|\gamma_i^t(x,s)/w_i\|_\infty \leqslant \|\sqrt{\nu}\|_\infty$.

Then, we have

$$\begin{aligned}\Phi^{t^{\mathrm{new}}}(x^{\mathrm{new}}, s^{\mathrm{new}}) &\leqslant (1+8h\lambda\|\sqrt{\nu}\|_\infty)(1-\frac{\alpha\lambda}{2\sqrt{\sum_{i=1}^m w_i}})\Phi^t(x,s)+\frac{\alpha\lambda}{2}\sqrt{\sum_i w_i^{-1}}\\ &\leqslant (1+8h\lambda\|\sqrt{\nu}\|_\infty-\frac{\alpha\lambda}{2\sqrt{\sum_{i=1}^m w_i}})\Phi^t(x,s)+\frac{\alpha\lambda}{2}\sqrt{\sum_i w_i^{-1}}\\ &\leqslant (1-\frac{\alpha\lambda}{4\sqrt{\sum_{i=1}^m w_i}})\Phi^t(x,s)+\frac{\alpha\lambda}{2}\sqrt{\sum_i w_i^{-1}}\\ &\leqslant (1-\frac{\alpha\lambda}{4\sqrt{\sum_{i=1}^m w_i}})\Phi^t(x,s)+\alpha\lambda\sqrt{m}.\end{aligned}$$

where the second step follows by our choice of $h \leqslant \frac{\alpha}{32\sqrt{w}\|\sqrt{\nu}\|_\infty}$ and the last step follows by $w_i \geq 1$. □

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 $\mathrm{tw}(A) = \tau$. 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($\mathbf{A}, x, s, \epsilon, k$): Given matrix $\mathbf{A} \in \mathbb{R}^{n\times d}$, initial central path parameter (x, s), target accuracy ϵ, and phase length k, the data-structure preprocesses in $O(\mathrm{nnz}(A) + n\log^3 n)$ time.
2. MultiplyAndMove(δ_μ, t): It maintains implicitly

   $$x = x + H_{\overline{x}}^{-1/2}(I - P_{\overline{x}})H_{\overline{x}}^{-1/2}\delta_\mu, s = s + tH_{\overline{x}}^{1/2}P_{\overline{x}}H_{\overline{x}}^{-1/2}\delta_\mu,$$

   where $P_{\overline{x}} = H_{\overline{x}}^{-1/2}A^\top(AH_{\overline{x}}^{-1}A^\top)^{-1}AH_{\overline{x}}^{-1/2}$ and, with probability at $1-\frac{1}{n^3}$, outputs $(\overline{x}, \overline{s})$ such that

   $$\|\overline{x}_j - x_j\|_{\overline{x}_j} \leqslant \epsilon \text{ and } t^{-1}\cdot\|\overline{s}_j - s_j\| \leqslant \epsilon \quad \text{for all } j \in [m].$$

   Moreover, for the query sequence $\delta_\mu^{(1)}, \delta_\mu^{(2)}, \ldots, \delta_\mu^{(l)}$, let $C_i = \|\delta_\mu^{(i)} - \delta_\mu^{(i-1)}\|_0$, then the time per call of MultiplyAndMove takes $\widetilde{O}(C_l\cdot\tau^3 + k^3\tau^3 + C_{l+1}k^2\tau^2)$.
3. RestartPhase: It compute (x, s) to $\frac{\epsilon}{n}$ accuracy explictly and reset $(\overline{x}, \overline{s})$ to current (x, s) and l to 0 to control the ℓ_2 norm that heavy-hitter maintains in $\widetilde{O}(n\tau^2)$ time.

We remark that since in the RestartPhase, we compute (x, s) to ϵ/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 serveral ingredients. The first one is the maintenance of $L^{-1}x$.

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

Proof. We can rewrite $L' = \sum_{i=0}^{\mathrm{nnz}(L')} u^{(i)}(v^{(i)})^\top$ where $u^{(i)}$ and $v^{(i)}$ are $O(1)$-sparse for any i. Then, using Sherman-Morrison formula, we have

$$(L + uv^\top)^{-1}x = L^{-1}x - \frac{L^{-1}uv^\top L^{-1}x}{1 + v^\top L^{-1}u}.$$

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^{new} in $O(\mathrm{nnz}(L')\cdot\tau^2)$. □

However, we cannot hope to use the same technique to maintain the product of $L^{-\top}$, since the column of $L^{-\top}$ can have $\Omega(n)$ many nonzero entries. Then, we use heavy-hitters to find all the coordinates that changes too much and use Lemma 2.23 to compute it exactly.

Lemma 5.3 (ℓ_2-heavy hitter [Kan+11; Pag12])**.** *There exists a function* Sketch(ϵ,n) *that given* $\epsilon > 0$ *explicitly returns a matrix* $\mathbf{\Phi} \in \mathbb{R}^{m_\Phi \times n}$ *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* $\mathbf{\Phi}$*. There further exists a function* Decode(y) *that given a vector* $y = \mathbf{\Phi}x$ *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* $1 - \frac{1}{n^5}$ *includes all* i *with*

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

Algorithm 7 Central Path Maintainance Data Structure Part 1

data structure: CentralPathMaintenance

private : members
- $\mathbf{A} \in \mathbb{R}^{d\times n}$
- $k \in \mathbb{Z}$
- $\epsilon_\Phi \leftarrow \frac{\epsilon}{2^{10}\cdot k\cdot \alpha \cdot \max_i n_i}$
- $\mathbf{\Phi} \in \mathbb{R}^{m_s \times n}$
- $l \in \mathbb{Z}$
- $l_x, l_s \in \mathbb{Z}^m$

procedure Initialize$(\mathbf{A}, x, s, \epsilon, k)$
- Let L_x to be Cholesky factorization of $AH_x^{-1}A^\top$ ▷ Theorem 2.3
- $\mathbf{\Phi} \leftarrow$ Sketch(ϵ_Φ, n)
- $x^{\text{old}} \leftarrow x, s^{\text{old}} \leftarrow s, k \leftarrow k, \epsilon \leftarrow \epsilon$
- Compute $\mathbf{\Phi}H_{\overline{x}}^{-1}A^\top L_{\overline{x}}^{-\top}, \mathbf{\Phi}A^\top L_{\overline{x}}^{-\top}$
- $l \leftarrow 1$
- $p \leftarrow \mathbf{1}$
- $\overline{x}^{(1)} \leftarrow x, \overline{s}^{(1)} \leftarrow s$

end procedure

procedure ResartPhase$(\overline{x}, \overline{s})$
- Recompute x and s using gradient descent
- Let L_x to be Cholesky factorization of $AH_x^{-1}A^\top$ ▷ Theorem 2.3
- $l \leftarrow 0$
- $p \leftarrow \mathbf{0}$

end procedure

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

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

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

Lemma 5.5. *The function* ComputeExact(j) *where* j *is indice of block move the approximate central path parameter of block* j $(\overline{x}_j, \overline{s}_j)$ *to* (x_j, s_j)*, and maintains* $L_{\overline{x}}$ *,* $h = L_{\overline{x}}^{-1}AH_{\overline{x}}\delta_\mu$*, and* $\mathbf{\Phi}A^\top L_{\overline{x}^{(l)}}^{-\top}h^{(l)}$*. Assuming* $l \leqslant k$*, each call of* ComputeExact *takes* $\widetilde{O}(k^2\tau^2)$ *time.*

Proof. Note that the approximate central path parameter of block j, $(\overline{x}_j, \overline{s}_j)$ was last updated on iteration p_j, which implies $(\overline{x}_j^{(p_j)}, \overline{s}_j^{(p_j)}) = (x_j^{(p_j)}, s_j^{(p_j)})$. Using the invariant of $x^{(l)} = x^{\mathrm{old}} + \sum_{i=1}^{l} H_{\overline{x}^{(i)}}^{-1} \delta_\mu^{(i)} - H_{\overline{x}^{(i)}}^{-1} A^\top L_{\overline{x}^{(i)}}^{-\top} h^{(i)}$, we have

$$x_j^{(l+1)} = x_j^{(p_j)} + (\sum_{i=p_j}^{l} H_{\overline{x}^{(i)}}^{-1} \delta_\mu^{(i)} - H_{\overline{x}^{(i)}}^{-1} A^\top L_{\overline{x}^{(i)}}^{-\top} h^{(i+1)})_j = \overline{x}_j^{(l)} + (\sum_{i=p_j}^{l} H_{\overline{x}^{(i)}}^{-1} \delta_\mu^{(i)} - H_{\overline{x}^{(i)}}^{-1} A^\top L_{\overline{x}^{(i)}}^{-\top} h^{(i+1)})_j,$$

where the last step follows by p_j is the last time when block j get updated, then we have $\overline{x}_j^{(l)} = \overline{x}^{(p_j)} = x_j^{(p_j)}$. Hence, we know that $\overline{x}_j^{(l+1)} = x_j^{(l+1)}$. The proof of $\overline{s}^{(l+1)} = s^{(l+1)}$ 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 $AH_{\overline{x}}A^\top$ in $O(\tau^2)$ time by Theorem 2.27. Finally, we show how to efficiently maintain h. The fact that wecan update L in $O(\tau^2)$ time, implies $\mathrm{nnz}(L_{\overline{x}^{(l)}} - L_{\overline{x}^{(l+1)}}) = O(\tau^2)$, then we can compute $L_{\overline{x}^{(l+1)}}^{-1} AH_{\overline{x}^{(l)}} \delta_\mu$ in $O(\tau^2)$ time by Lemma 5.2. Splitting $L_{\overline{x}^{(l+1)}}^{-1} AH_{\overline{x}^{(l+1)}} \delta_\mu^{(l)}$, we get

$$L_{\overline{x}^{(l+1)}}^{-1} AH_{\overline{x}^{(l+1)}} \delta_\mu^{(l)} = L_{\overline{x}^{(l+1)}}^{-1} AH_{\overline{x}^{(l)}} \delta_\mu^{(l)} + L_{\overline{x}^{(l+1)}}^{-1} A(H_{\overline{x}^{(l+1)}} - H_{\overline{x}^{(l)}}) \delta_\mu^{(l)}.$$

We complete the proof by noticing that $\|(H_{\overline{x}^{(l+1)}} - H_{\overline{x}^{(l)}}) \delta_\mu^{(l)}\|_0 = O(1)$ and Lemma 2.22.

Similarly, we can maintain $\mathbf{\Phi} H_{\overline{x}}^{-1} A^\top L_{\overline{x}}^{-\top}$ and $\mathbf{\Phi} A^\top L_{\overline{x}}^{-\top}$ using $O(1/\epsilon_\Phi^2 \cdot \tau^2) = \widetilde{O}(k^2 \cdot \tau^2)$ time using Lemma 5.2, which implies we can also maintain $\mathbf{\Phi} H_{\overline{x}^{(l)}}^{-1} A^\top L_{\overline{x}^{(l)}}^{-\top} h^{(l)}$ and $\mathbf{\Phi} A^\top L_{\overline{x}^{(l)}}^{-\top} h^{(l)}$ in the same time by using sparse matrix-vector multiplication.

□

Lemma 5.6. *The function* MultiplyAndMove(δ_μ, t) *maintains implicitly*

$$x = x + H_{\overline{x}}^{-1/2} (I - P_{\overline{x}}) H_{\overline{x}}^{-1/2} \delta_\mu, s = s + t H_{\overline{x}}^{1/2} P_{\overline{x}} H_{\overline{x}}^{-1/2} \delta_\mu,$$

where $P_{\overline{x}} = H_{\overline{x}}^{-1/2} A^\top (AH_{\overline{x}}^{-1} A^\top)^{-1} AH_{\overline{x}}^{-1/2}$ *and, with probability at* $1 - \frac{1}{n^3}$*, outputs* $(\overline{x}, \overline{s})$ *such that*

$$\|\overline{x}_j - x_j\|_{\overline{x}_j} \leqslant \epsilon \text{ and } t^{-1} \cdot \|\overline{s}_j - s_j\| \leqslant \epsilon \quad \text{for all } j \in [m].$$

Moreover, for the query sequence $\delta_\mu^{(1)}, \delta_\mu^{(2)}, \ldots, \delta_\mu^{(l)}$*, let* $C_i = \|\delta_\mu^{(i)} - \delta_\mu^{(i-1)}\|_0$*, each call of* MultiplyAndMove *takes* $\widetilde{O}(C_l \cdot \tau^3 + k^3\tau^3 + C_{l+1} k^2 \tau^2)$*.*

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

$$x^{(l+1)} = x^{\mathrm{old}} + \sum_{i=1}^{l} H_{\overline{x}^{(i)}}^{-1} \delta_\mu^{(i)} - H_{\overline{x}^{(i)}}^{-1} A^\top L_{\overline{x}^{(i)}}^{-\top} h^{(i+1)} \text{ and } s^{(l+1)} = s^{\mathrm{old}} + \sum_{i=1}^{l} t^{(i)} A^\top L^{-\top} h^{(i+1)}. \tag{5.1}$$

Assume we have $x^{(l)} = x^{\mathrm{old}} + \sum_{i=1}^{l-1} H_{\overline{x}^{(i)}}^{-1} \delta_\mu^{(i)} + H_{\overline{x}^{(i)}}^{-1} A^\top L_{\overline{x}^{(i)}}^{-\top} h^{(i+1)}$, then we have

$$\begin{aligned} x^{\mathrm{old}} + \sum_{i=1}^{l} H_{\overline{x}^{(i)}}^{-1} \delta_\mu^{(i)} + H_{\overline{x}^{(i)}}^{-1} A^\top L_{\overline{x}^{(i)}}^{-\top} h^{(i+1)} &= x^{\mathrm{old}} + \left(\sum_{i=1}^{l-1} H_{\overline{x}^{(i)}}^{-1} \delta_\mu^{(i)} - H_{\overline{x}^{(i)}}^{-1} A^\top L_{\overline{x}^{(i)}}^{-\top} h^{(i+1)} \right) + \left(H_{\overline{x}^{(l)}}^{-1} \delta_\mu - H_{\overline{x}^{(l)}}^{-1} A^\top L_{\overline{x}^{(l)}}^{-\top} h^{(i+1)} \right) \\ &= x^{(l)} + H_{\overline{x}^{(l)}}^{-1} \delta_\mu - H_{\overline{x}^{(l)}}^{-1} A^\top L_{\overline{x}^{(l)}}^{-\top} L_{\overline{x}^{(l)}}^{-1} AH_{\overline{x}^{(l)}}^{-1} \delta_\mu^{(l)} \\ &= x^{(l+1)}. \end{aligned}$$

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 Maintainance Data Structure Part 2

procedure MULTIPLYANDMOVE(δ_μ, t) ▷ Find $(\overline{x}^{(l+1)}, \overline{s}^{(l+1)})$ using $(\overline{x}^{(l)}, \overline{s}^{(l)})$ and move counter l
  $\delta_\mu^{(l)} \leftarrow \delta_\mu$
  $h^{(l+1)} \leftarrow L_{\overline{x}^{(l)}}^{-1} A H_{\overline{x}^{(l)}} \delta_\mu^{(l)}$ ▷ Compute h using Lemma 5.2
  $v_x = \mathbf{\Phi}\left[\left(\sum_{i=1}^{l} H_{\overline{x}^{(i)}}^{-1} \delta_\mu^{(i)} - H_{\overline{x}^{(i)}}^{-1} A^\top L_{\overline{x}^{(i)}}^{-\top} h^{(i+1)}\right) - (\overline{x}^{(l)} - x^{\text{old}})\right]$
  $v_s = \mathbf{\Phi}\left[\left(\sum_{i=1}^{l} t^{(i)} A^\top L_{\overline{x}^{(i)}}^{-\top} h^{(i+1)}\right) - (\overline{s}^{(l)} - s^{\text{old}})\right]$
  $\overline{x}^{(l+1)} \leftarrow \overline{x}^{(l)}, \overline{s}^{(l+1)} \leftarrow \overline{s}^{(l)}, L_{\overline{x}^{(l+1)}} \leftarrow L_{\overline{x}^{(l)}}$
  for all block $j \cap (\text{DECODE}(v_x) \cup \text{DECODE}(v_s))$ **do**
    $\overline{x}_j' \leftarrow \overline{x}_j^{(l)} + (\sum_{i=p_j}^{l} H_{\overline{x}^{(i)}}^{-1} \delta_\mu^{(i)} - H_{\overline{x}^{(i)}}^{-1} A^\top L_{\overline{x}^{(i)}}^{-\top} h^{(i+1)})_j$
    $\overline{s}_j' \leftarrow \overline{s}_j^{(l)} + (\sum_{i=p_j}^{l} t^{(i)} A^\top L_{\overline{x}^{(i)}}^{-\top} h^{(i)})_j$
    if $\|\overline{x}_j' - \overline{x}_j^{(l)}\|_{\overline{x}_j'} > \epsilon_\Phi$ or $t^{-1}\|\overline{s}_j' - \overline{s}_j^{(l)}\|_{\overline{x}_j'}^* > \epsilon_\Phi$ **then**
      UPDATEBLOCK($j, \overline{x}_j', \overline{s}_j'$)
    end if
  end for
  $l \leftarrow l + 1$

  if $l > k$ **then**
    RESTARTPHASE($\overline{x}, \overline{s}$)

  end if
  return $(\overline{x}, \overline{s})$
end procedure

procedure COMPUTEEXACT($j, \overline{x}_j', \overline{s}_j'$) ▷ Lemma 5.5
  $\overline{x}_j^{(l+1)} \leftarrow \overline{x}_j'$, $\overline{s}_j^{(l+1)} \leftarrow \overline{s}_j'$
  $p_j \leftarrow l$
  $L_{\overline{x}^{(l+1)}} \leftarrow \text{UPDATE}(L_{\overline{x}^{(l+1)}}, i)$ ▷ Theorem 2.27
  $h^{\text{old}} \leftarrow L_{\overline{x}^{(l+1)}}^{-1} A H_{\overline{x}^{(l+1)}} \delta_\mu^{(l)}$
  Compute $\mathbf{\Phi} H_{\overline{x}^{(l)}}^{-1} A^\top L_{\overline{x}^{(l)}}^{-\top} h^{(l)}$ and $\mathbf{\Phi} A^\top L_{\overline{x}^{(l)}}^{-\top} h^{(l)}$
end procedure

Explicit Maintainance: Recall that p_j be the last iteration that we set $\overline{x}_j = x_j$. We define $\delta_x^{(i)} \stackrel{\text{def}}{=} H_{\overline{x}^{(i)}}^{-1}\delta_\mu^{(i)} - H_{\overline{x}^{(i)}}^{-1}A^\top L_{\overline{x}^{(i)}}^{-\top}h^{(i+1)}$. Then, we have

$$\|\overline{x}_j - x_j\|_{\overline{x}_j} = \Big\| \sum_{i=p_j}^{l} \delta_{\overline{x}_j^{(i)}} \Big\|_{\overline{x}_j}.$$

By guarantee of Lemma 5.3, we have $\|H_{\overline{x}^{(l)}}^{1/2}(\overline{x}^{(l)} - x^{(l)})\|_\infty \leqslant \epsilon_s \|H_{\overline{x}^{(l)}}^{1/2}(\overline{x}^{(l)} - x^{(l)})\|_2 = \epsilon_s \|\overline{x}^{(l)} - x^{(l)}\|_{\overline{x}^{(l)}}$. Now, it's suffices to show $\|\overline{x}^{(l)} - x^{(l)}\|_{\overline{x}^{(l)}}$ is bounded. WLOG, we assume $\delta_{\overline{x}_j}^{(i)} = 0$ for all $i < s_j$. Then, we have

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

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

$$\|\overline{x}_j^{(l)} - \overline{x}_j^{(l')}\|_{\overline{x}^{(l)}} \leqslant 0.1$$

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

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

Complexity: Since $\|\delta_\mu^{(l)} - \delta_\mu^{(l-1)}\|_0 \leqslant C_l$, we have $\|AH_{\overline{x}^{(l)}}(\delta_\mu^{(l)} - \delta_\mu^{(l-1)})\|_0 \leqslant O(C_l \cdot \tau)$. Then,we can compute $h^{(l+1)}$ in $O(C_l \cdot \tau^3)$ using Lemma 2.22. For compute of $\overline{x}_j, \overline{s}_j$, we show that we can compute $(A^\top L_{\overline{x}^{(i)}}^{-\top}h^{(i+1)})_j$ in $O(\tau^2)$ time, since all blocks are constant size and $l \leqslant k$. Rewrite $(A^\top L_{\overline{x}^{(i)}}^{-\top}h^{(i+1)})_j$ to $(A^\top L_{\overline{x}^{(i)}}^{-\top}h^{(i+1)})^\top e_j$, we have

$$(A^\top L_{\overline{x}^{(i)}}^{-\top}h^{(i+1)})_j = h^{(i+1)\top}L_{\overline{x}^{(i)}}^{-1}Ae_j,$$

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 $\overline{x}'_j, \overline{s}'_j$ for at most $O(\epsilon_\Phi^{-2}\log n)$ many j. Since the function ComputeExactly maintains the invariant that $h^{\text{old}} = L_{\overline{x}^{(l)}}^{-1}AH_{\overline{x}^{(l)}}\delta_\mu^{(l-1)}$, we have $h^{(l+1)} = h^{\text{old}} + L_{\overline{x}^{(l)}}^{-1}AH_{\overline{x}^{(l)}}(\delta_\mu^{(l)} - \delta_\mu^{(l-1)})$.For compute v_x and v_s, assuming that we have $\mathbf{\Phi}H_{\overline{x}^{(l)}}^{-1}A^\top L_{\overline{x}^{(l)}}^{-\top}h^{(l)}$ and $\mathbf{\Phi}A^\top L_{\overline{x}^{(l)}}^{-\top}h^{(l)}$, we can compute v_x and v_s $O(m_\Phi \cdot C_1 \cdot \tau) = \widetilde{O}(k^2 \cdot C_l \cdot \tau)$ time, since $\|h^{(l+1)} - h\| = O(C_1 \cdot \tau)$ and $\mathbf{\Phi}A^\top L_{\overline{x}^{(l)}}^{-\top} \in \mathbb{R}^{m_\Phi \times n}$ 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 $\widetilde{O}(C_{l+1} \cdot k^2 \cdot \tau^2)$. Summing all the cost up, we have the total cost is

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

□

Lemma 5.7. *Suppose the function* MoveAndMultiply(δ_μ, 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 $\sum_{j=1}^{m} \|x_j^{(l+1)} - x_j^{(l)}\|_{x_j^{(l)}} \leqslant 1$ which directly implies

$$\sum_{i=1}^{k}\sum_{j=1}^{m} \|x_j^{(i+1)} - x_j^{(i)}\|_{x_j^{(i)}}^2 \leqslant k \cdot \alpha.$$

Then we have except for $\frac{4}{\epsilon}\cdot k^2$ many coordinates, we have $\sum_{i=1}^k \|x_j^{(i+1)} - x_j^{(i)}\|^2_{x_j^{(i)}} \leqslant \frac{\epsilon}{4k}$, by Cauchy-Schwarz, we have

$$\sum_{i=1}^k \|x_j^{(i+1)} - x_j^{(i)}\|_{x_j^{(i)}} \leqslant \frac{\epsilon\alpha}{4}.$$

Let $\alpha_j^{(i)} \stackrel{\text{def}}{=} \|x_j^{(i+1)} - x_j^{(i)}\|_{x_j^{(i)}}$, by Lemma 4.3, we have $H_{x_j^{(i)}} \preceq e^{2\alpha_j^{(i)}} H_{x_j^{(i+1)}}$. Thus, we have $H_{x_j^{(1)}} \preceq e^{\sum_{i=1}^k a_j^{(i)}} H_{x_j^{(k)}} \preceq (1+\frac{\epsilon}{2}) H_{x_j^{(k)}}$ by our choice of α, which shows all hessians are $\frac{\epsilon}{2}$-approximation of each other. This implies for all the coordinate that does not belongs to $\frac{4}{\epsilon}\cdot k^2$, we have $\|x_j^{(l)} - x_j^{(1)}\|_{x_j^{(l)}} \leqslant \epsilon$ for all the $l \leqslant k$. 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 $O(\frac{T}{k}\cdot k^2/\epsilon) = O(T\cdot k/\epsilon)$. □

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

Lemma 5.8. *Given an radius R and an approximate central path parameter $(\overline{x}, \overline{s})$ such that there exists a feasible central path parameter in ℓ_∞ 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 central parameter (x, s) to target accuracy ϵ.*

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 postive definte 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 $0 \leqslant \mu \leqslant L$ such that for all $x \in Q$ we have $\mu\cdot H \preceq \nabla^2 f(x) \preceq L\cdot H$. 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

$$\|x^{(k)} - x^*\|_H^2 \leqslant \left(1 - \frac{\mu}{L}\right)^k \|x^{(0)} - x^*\|_H^2.$$

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

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 ϵ_{mp} and the guarantee of MultiplyAndMove in Algorithm 8. The second and thrid assumptions directly follows by our choice of λ, α, and ϵ_{mp}. Now, we the potential $\Phi^t(x, s)$ is always bounded by $10\frac{m^3}{\alpha}$. At the beginning of algorithm, we use Theorem 3.2 to modify the convex program with parameter $\min(\frac{\delta}{2}, \frac{1}{\lambda})$. Then, the initial point x and s satisfies $\gamma_i(x, s) = \|s_i + w_i\nabla\phi_i(x_i)\|^*_{x_i} \leqslant \frac{1}{\lambda}$ and hence $\Phi^1(x, s) \leqslant e\cdot m$. Hence, we have $\Phi^t(x, s) \leqslant O(m)$ during the whole algorithm by Theorem 4.15. □

Algorithm 9 Main Algorithm

procedure CentralPathStep($\overline{x}, \overline{s}$)
  for $i = 1 \to m$ **do**
    $\mu_i^t \leftarrow \frac{\overline{s}_i}{t} + w_i \nabla \phi_i(\overline{x}_i)$
    $\gamma_i^t \leftarrow \|\mu_i^t\|_{\overline{x}_i}^*$
    $c_i^t \leftarrow \frac{\sinh(\frac{\lambda}{w_i}\gamma_i^t(\overline{x},\overline{s}))}{\gamma_i^t(\overline{x},\overline{s}) \cdot \sqrt{\sum_{j=1}^m w_j^{-1} \sinh^2(\frac{\lambda}{w_i}\gamma_j^t(\overline{x},\overline{s}))}}$
    $\delta_{\mu,i} \leftarrow -\alpha \cdot c_i^t \cdot \mu_i^t$
  end for
  return δ_μ
end procedure

procedure Main(A, b, c, ϕ, δ)
  $\lambda \leftarrow 100 \log^2 n$, $\alpha \leftarrow (2^{10}\lambda^2)^{-1}$, $\kappa \leftarrow \frac{1}{100}\alpha$
  $\delta \leftarrow \min(\frac{\delta}{2}, \frac{1}{\lambda})$
  $\epsilon_{\text{mp}} \leftarrow 2^{-10}\lambda$
  $k \leftarrow n^{0.25}$
  $w \leftarrow \mathbf{1}_m$
  Modify the convex program and obtain an initial x and s by Theorem 3.2
  CentralPathMaintenance mp. ▷ Algorithm 7
  mp.Initialize($\mathbf{A}, x, s, \epsilon_{\text{mp}}, k$)
  $t \leftarrow 1$ ▷ Initialize t
  while $t > \delta^2/(4\nu)$ **do**
    $t^{\text{new}} \leftarrow (1 - \frac{\kappa}{\sqrt{\nu}})t$
    $\delta_\mu \leftarrow$ CentralPathStep($\overline{x}, \overline{s}$)
    $(\overline{x}, \overline{s}) \leftarrow$ mp.MultiplyAndMove(δ_μ, t)
    $t \leftarrow t^{\text{new}}$
  end while
  Recover x, s using Theorem 3.2.
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 ν_i-self concordant barrier function ϕ_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 $x \in \sum_{i=1}^m K_i$, we have that $\|x\|_2 \leqslant R$.

2. Lipschits constant of program: $\|c\|_\infty \leqslant L$.

Then, for any $\delta > 0$, the randomized algorithm Main runs in time $\widetilde{O}(n^{1.25} \cdot \tau^3 \cdot \log(1/\delta))$ and finds a vector x such that

$$\begin{aligned} c^\top x &\leqslant \min_{Ax=b, x \in \prod_{i=1}^m K_i} c^\top x + LR \cdot \delta \\ \|Ax\|_1 &\leqslant 3\delta \cdot (R\|A\|_1 + \|b\|_1) \\ x &\in \prod_{i=1}^m K_i. \end{aligned}$$

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

Thus, the total runtime of MultiplyAndMove is

$$\begin{aligned} \widetilde{O}(\sum_{l=1}^T C_l \cdot \tau^3 + k^3\tau^3 + C_{l+1}k^2\tau^2) &= \widetilde{O}(T \cdot k^3\tau^3) + \widetilde{O}(\frac{T}{k} \cdot \log(\nu/\delta)) \cdot \widetilde{O}(\tau^3 + k^2\tau^2) \\ &= \widetilde{O}(T \cdot k^3\tau^3 + Tk\tau^2 + Tk^{-1}\tau^3) \\ &= \widetilde{O}(n^{1.25}\tau^3 \log(1/\delta)). \end{aligned}$$

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

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. Finaly, I want to thank my parents Ronglin and Mei for their encouragement and always being supportive.

References

[AK16] Sanjeev Arora and Satyen Kale. "A combinatorial, primal-dual approach to semidefinite programs". In: *Journal of the ACM (JACM)* 63.2 (2016), pp. 1–35.

[Bod+95] Hans L Bodlaender, John R Gilbert, Hjálmtyr Hafsteinsson, and Ton Kloks. "Approximating treewidth, pathwidth, frontsize, and shortest elimination tree". In: *Journal of Algorithms* 18.2 (1995), pp. 238–255.

[CLS19] Michael B. Cohen, Yin Tat Lee, and Zhao Song. "Solving linear programs in the current matrix multiplication time". In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019.* 2019, pp. 938–942. DOI: 10.1145/3313276.3316303.

[Dav06] Timothy A. Davis. *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Jan. 2006. ISBN: 978-0-89871-613-9. DOI: 10.1137/1.9780898718881.

[DH03] Timothy A. Davis and William W. Hager. "Modifying a Sparse Cholesky Factorization". In: *SIAM Journal on Matrix Analysis and Applications* 20.3 (2003), pp. 606–627. ISSN: 0895-4798. DOI: 10.1137/s0895479897321076.

[Kan+11] Daniel M. Kane, Jelani Nelson, Ely Porat, and David P. Woodruff. "Fast Moment Estimation in Data Streams in Optimal Space". In: *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*. STOC '11. San Jose, California, USA: Association for Computing Machinery, 2011, pp. 745–754. ISBN: 9781450306911. DOI: 10.1145/1993636.1993735.

[LR99] Tom Leighton and Satish Rao. "Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms". In: *Journal of the ACM (JACM)* 46.6 (1999), pp. 787–832.

[LS19] Yin Tat Lee and Aaron Sidford. "Solving Linear Programs with Sqrt(rank) Linear System Solves". In: *CoRR* abs/1910.08033 (2019). arXiv: 1910.08033.

[LSZ19] Yin Tat Lee, Zhao Song, and Qiuyi Zhang. "Solving Empirical Risk Minimization in the Current Matrix Multiplication Time". In: *Conference on Learning Theory, COLT 2019, 25-28 June 2019, Phoenix, AZ, USA*. 2019, pp. 2140–2157.

[Nes98] Yurii Nesterov. "Introductory Lectures on Convex Optimization - A Basic Course". In: *Lecture Notes*. 1998.

[Pag12] Rasmus Pagh. "Compressed Matrix Multiplication". In: *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*. ITCS '12. Cambridge, Massachusetts: Association for Computing Machinery, 2012, pp. 442–451. ISBN: 9781450311151. DOI: 10.1145/2090236.2090271.

[Sch82] R. Schreiber. "A New Implementation of Sparse Gaussian Elimination". In: *ACM Trans. Math. Softw.* 8 (1982), pp. 256–276.

A Appendix

A.1 Hyperbolic function lemmas

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

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

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

$$\begin{aligned}|\sinh(x+y) - \sinh(x)| &\leqslant |\sinh(x)||\cosh(y) - 1| + \cosh(x)\sinh(y) \\ &\leqslant |\sinh(x)|(|\cosh(y) - 1| + |\sinh(y)|) + |\sinh(y)|\end{aligned}$$

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

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

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

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

$$\begin{aligned}\cosh(x+y) &= \cosh(x)\left[\exp(y) - \sinh(y)\right] + \sinh(x)\sinh(y) \\ &\leqslant \cosh(x)\left[\exp(y) - \sinh(y)\right] + \cosh(x)\sinh(y) \\ &\leqslant \cosh(x)\exp(y) \\ &\leqslant \cosh(x) + 2y\cosh(x),\end{aligned}$$

where we use $\exp(y) \leqslant 1 + 2y$ for $0 \leqslant y \leqslant 1$. □

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 μ-strongly convex, then ∇f^* is $\frac{1}{L}$-Lipschitz. If ∇f is L-Lipschitz, then f^* is $\frac{1}{L}$-strongly convex. Furthermore, let $x_\theta \stackrel{\text{def}}{=} \arg\max_x \theta^\top x - f(x)$. Then, we have that*

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

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

- $\frac{1}{3}H \preceq \nabla^2 g(x) \preceq 2H$ for all x.

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

$$\begin{aligned} f|_B(x) &= \begin{cases} f(x) & \text{if } \|x\|_2 \leqslant \lambda \\ +\infty & \text{otherwises} \end{cases}, \\ h(\theta) &= (f|_B)^*(\theta) + r(\|\theta - \nabla f(0)\|_2), \\ r(t) &= \frac{1}{2}((t - \frac{\lambda}{4})^+)^2, \\ g &= h^*. \end{aligned}$$

First, we note that f is $\frac{1}{2}$-strongly convex on $\|x\|_2 \leqslant \lambda$ and hence $f|_B$ is $\frac{1}{2}$-strongly convex. Therefore, Lemma 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 ∇h is 3-Lipschitz.

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

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

By the optimality condition, we have

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

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

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

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

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

for any θ with $\|\theta - \nabla f(0)\|_2 < \frac{\lambda}{2}$. Hence, for those θ, (Eq. (A.1)) shows that

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

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

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

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

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

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

□