A Nearly-Linear Time Algorithm for Linear Programs with Small Treewidth: A Multiscale Representation of Robust Central Path

Sally Dong
University of Washington

Yin Tat Lee
University of Washington
& Microsoft Research

Guanghao Ye
University of Washington

September 14, 2023

Abstract

Arising from structural graph theory, treewidth has become a focus of study in fixed-parameter tractable algorithms in various communities including combinatorics, integer-linear programming, and numerical analysis. Many NP-hard problems are known to be solvable in $\widetilde{O}(n \cdot 2^{O(\mathrm{tw})})$ time, where tw is the treewidth of the input graph. Analogously, many problems in P should be solvable in $\widetilde{O}(n \cdot \mathrm{tw}^{O(1)})$ time; however, due to the lack of appropriate tools, only a few such results are currently known. [FLS+18] conjectured this to hold for as broadly as all linear programs; in our paper, we show this is true:

Given a linear program of the form $\min_{Ax=b, \ell \leqslant x \leqslant u} c^\top x$, and a width-$\tau$ tree decomposition of a graph G_A related to A, we show how to solve it in time

$$\widetilde{O}(n \cdot \tau^2 \log(1/\varepsilon)),$$

where n is the number of variables and ε is the relative accuracy. Combined with recent techniques in vertex-capacitated flow [BGS21], this leads to an algorithm with $\widetilde{O}(n^{1+o(1)} \cdot \mathrm{tw}^2 \log(1/\varepsilon))$ runtime. Besides being the first of its kind, our algorithm has runtime nearly matching the fastest runtime for solving the sub-problem $Ax = b$, under the assumption that no fast matrix multiplication is used.

We obtain these results by combining recent techniques in interior-point methods (IPMs), sketching, and a novel representation of the solution under a multiscale basis similar to the wavelet basis.

Contents

1 Introduction

Linear programming is one of the most fundamental problems in computer science and optimization. General techniques for solving linear programs, such as simplex methods [Dan51], ellipsoid methods [Kha80] and interior point methods [Kar84], have been developed and continuously refined since the 1940s, and have later been found to be useful in a wide range of problems spanning optimization, combinatorics, and machine learning.

For an arbitrary linear program $\min_{Ax=b,\ell\leqslant x\leqslant u} c^\top x$ with n variables and d constraints, the current fastest algorithms take either $\widetilde{O}(n^{2.373}\log(1/\varepsilon))$ time [CLS19, JSWZ20] or $\widetilde{O}((\sqrt{nd}\cdot\mathrm{nnz}(A)+d^{2.5})\log(1/\varepsilon))$ time [vdBLSS20, BLL+20], where ε is the accuracy parameter[1]. When A is a dense matrix, these runtimes are close to optimal, as they nearly match the runtime $\widetilde{O}((\mathrm{nnz}(A)+d^\omega)\log(1/\varepsilon))$ to solve the subproblem $Ax=b$, where $\omega\approx 2.373$ is the matrix multiplication exponent. When A is sparse, as is the case in many problems arising from both theory and applications, we ask if much faster runtimes are possible.

When n and d are the same order, this problem is highly non-trivial, even for linear systems. It is only recently known how to solve a sparse linear system in slightly faster than d^ω time [PV20], and sub-quadratic time is insurmountable under the current techniques. It turns out in practice however, sparse linear systems often have low *treewidth*, a condition much stronger than mere sparsity; for example, many of the linear programs in the Netlib repository have sublinear treewidth (Appendix B). For low treewidth linear systems, a small polynomial dependence on treewidth still implies a much faster than quadratic runtime, hence making them a particularly suitable target of study.

Beyond the practical consideration, whether there is a $\widetilde{O}(n\cdot\mathrm{tw}^{O(1)})$ LP algorithm is important in parameterized complexity. Most algorithms designed for low treewidth graphs rely on dynamic programming, which naturally give algorithms with runtime exponential in treewidth even for problems in P, such as reachability and shortest paths [ASK12, CŁ13, CZ00, PdWvdK12]. There are only a few problems in P that we know how to solve in $\widetilde{O}(n\cdot\mathrm{tw}^{O(1)})$ time [FLS+18]. We refer to Sections 1.3.1 and 1.3.2 for a discussion of these problems.

Recently, [FLS+18] posed exactly this question[2]:

$$\text{Can linear programs be solved in } \widetilde{O}(n\cdot\mathrm{tw}^{O(1)}\log(1/\varepsilon)) \text{ time?}$$

We answer the question affirmatively in this paper:

Theorem 1.1. *Given a linear program* $\min_{Ax=b,\ell\leqslant x\leqslant u} c^\top x$*, where* $A\in\mathbb{R}^{d\times n}$ *is a full rank matrix with* $d\leqslant n$*, define the* dual graph G_A[3] *to be the graph with vertex set* $\{1,\ldots,d\}$*, such that* $ij\in E(G_A)$ *if there is a column* r *such that* $A_{i,r}\neq 0$ *and* $A_{j,r}\neq 0$*. Suppose that*

- *a tree decomposition of* G_A *with width* τ *is given, and*
- r *is the inner radius of the polytope, namely, there is* x *such that* $Ax=b$ *and* $\ell+r\leqslant x\leqslant u-r$*.*

[1] The current fastest exact algorithms for linear program take either $2^{O(\sqrt{d\log\frac{n-d}{d}})}$ time [HZ15], or the runtime depends on the magnitude of entries of A.

[2] We add the $\log(1/\varepsilon)$ term to their original conjecture. Without this term, the conjecture will imply the existence of strongly polynomial time algorithms for linear programs, one of Smale's 18 unsolved problems in mathematics.

[3] There are different ways of associating a graph with the matrix A (see [JK15, FLS+18]). We adopt the one used in the ILP community [JK15, EHK+19]. We choose this definition so that when applied to linear programming formulations of flow problems, in which the constraint matrix A is the incidence matrix of the input graph G, we have $G_A=G$, and hence the treewidth of the LP is most meaningfully related to the flow problem.

Let $L = \|c\|_2$ and $R = \|u-\ell\|_2$. For any $0 < \varepsilon \leqslant 1/2$, we can find x such that $Ax = b$ and $\ell \leqslant x \leqslant u$ such that

$$c^\top x \leqslant \min_{Ax=b,\, \ell \leqslant x \leqslant u} c^\top x + \varepsilon \cdot LR$$

in expected time

$$\widetilde{O}(n \cdot \tau^2 \log(R/(r\varepsilon))).$$

To keep this paper simple, we refrain from using fast matrix multiplication. Under this restriction, we note that our runtime is tight, since it nearly matches the fastest runtime for solving the subproblem $Ax = b$ (Corollary 5.8).

Our algorithm involves a pre-processing component: We need to find some suitable reordering of the rows of A, known as an *elimination order*, so that matrices in later computations will have certain desired sparsity patterns. In practice, there are various efficient algorithms for finding a good reordering, such as minimum degree orderings [GL89, ADD96, FMP+18] and nested dissection algorithms [Geo73, LRT79, KK98]. In theory, there are also different ways to compute the reordering. In the previous version of this paper, we applied techniques in [AK16] and [BW17] to give two reordering algorithms with suboptimal bounds. They are removed to shorten the paper. After our paper, [BGS21] gave an almost-linear time algorithm to compute a tree decomposition that is a polylog n factor from optimal. This implies the following:

Theorem 1.2. *Applying the algorithms in [BGS21], the runtime in Theorem 1.1 becomes*

$$\widetilde{O}\left(\left((n \cdot \mathrm{tw})^{1+o(1)} + n \cdot \mathrm{tw}^2\right) \log(1/\varepsilon)\right) = \widetilde{O}\left(\left(n^{1+o(1)} \cdot \mathrm{tw}^2\right) \log(1/\varepsilon)\right).$$

Detailed discussions can be found in literature (e.g. [Ren88] and [LS13, Sections E, F]) on converting an approximation solution to an exact solution. To summarize, for integral A, b, c, it suffices to pick $\varepsilon = 2^{-O(L)}$ to get an exact solution, where $L = \log(1 + d_{\max} + \|b\|_2 + \|c\|_2)$ is the bit complexity and $d_{\max}$ is the largest absolute value of the determinant of a square sub-matrix of A. For many combinatorial problems, $L = O(\log(n + \|b\|_2 + \|c\|_2))$.

1.1 Convex Generalization

Theorem 1.1 and Theorem 1.2 generalize to a class of convex optimization problem as follows:

Theorem 1.3. *Given a convex program*

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

where $A \in \mathbb{R}^{d \times n}$ is a full rank matrix with $d \leqslant n$ and $K_i \subset \mathbb{R}^{n_i}$ are convex sets, with $\sum_{i=1}^m n_i = n$. We identify the columns of A in blocks, such that block i contains the n_i columns corresponding to x_i. We define the generalized dual graph *G_A to be the graph with vertices set $\{1, \cdots d\}$, such that $ij \in E(G_A)$ if there is a block r such that $A_{i,r} \neq \mathbf{0}$ and $A_{j,r} \neq \mathbf{0}$. We define the product convex set $K = \Pi_{i=1}^m K_i$. Suppose that*

- *we are given a tree decomposition of G_A with width τ,*
- *R is the diameter of the set K,*
- *There exists z such that $Az = b$ and $B(z, r) \subset K$,*

- $n_i = O(1)$ *for all* $i \in [m]$,
- *we are given initial points* $x_i \in \mathbb{R}^{n_i}$ *such that* $B(x_i, r) \subset K_i$ *for each* i,
- *we can check if* $y \in K_i$ *in* $O(1)$ *time for all* $i \in [m]$.

Then, for any $0 < \varepsilon \leqslant 1/2$, *we can find* $x \in K$ *with* $Ax = b$ *such that*

$$c^\top x \leqslant \min_{Ax=b, x \in K} c^\top x + \varepsilon \cdot \|c\|_2 \cdot R$$

in expected time

$$\widetilde{O}(n \cdot \tau^2 \log(R/r) \log(R/(r\varepsilon))).$$

Remark 1.4. The proofs for the convex program and the linear program are almost identical. Any operation involving the entry $A[i, j]$ in the linear program setting is generalized to operations on the $1 \times n_j$ submatrix of A from row i and block j. Since each block has size $O(1)$, the overall runtime relating to all matrix operations is maintained. We analyze our interior point method directly using this generalized formulation in this paper; the linear programming formulation follows as a special case.

This natural convex generalization in fact captures a large number of problem formulations. We illustrate with one example from signal processing, the fused lasso model for denoising [TSR+05]: Given a 1-D input signal $u_1, u_2, \cdots, u_n$, find an output x that minimizes the potential

$$V(x) = \sum_{i=1}^{n} (x_i - u_i)^2 + \lambda \sum_{i=1}^{n-1} |x_{i+1} - x_i|,$$

where the first term restricts the output signal to be close to the input, and the second term controls the amount of irregularity, and λ is the regularization parameter. To relate it back to our problem Eq. (1.1), we consider a generalized formulation: Given a family of convex functions $\phi_1, \ldots, \phi_N$ of $x = (x_1, \ldots, x_n)$, where for each i, the function $\phi_i(x) = \phi_i(x_{S_i})$ only depends on the variables $\{x_j : j \in S_i\}$ for some subset $S_i \in [n]$, we want to solve the problem

$$\min_{x \in \mathbb{R}^n} \sum_{i=1}^{N} \phi_i(x_{S_i}). \tag{1.2}$$

By creating extra variables $y_{i,j}$ for all $i \in [n]$ and $j \in S_i$, we can write the problem as $\min \sum_i t_i$, subjected to $y_{i,j} = x_j$ and $t_i \geq \phi_i(y_{i,j})$ for all i and all $j \in S_i$. The inequality constraints is equivalent to requiring that (t, y) lie in the convex set $\{(t, y) : t_i \geq \phi_i(y_{i,j})\}$. This is exactly in the form of Eq. (1.1). The dual graph G_A of this problem is closely related to the intersection graph $G_{\mathcal{I}}$ of the set family $\{S_i\}_{i \in N}$: Specifically, each set of constraints $y_{i,j} = x_j$ corresponds to $|S_i|$ many vertices in G_A, and contracting each such set into one vertex produces $G_{\mathcal{I}}$. Hence, we have that the treewidth $\mathrm{tw}(G_A)$ of this convex program is at most the treewidth of $G_{\mathcal{I}}$. For the denoising problem above, the intersection graph is in fact close to a path and has constant treewidth. Therefore, our result shows that this problem can be solved in nearly-linear time, without relying on the specific formula or structure.

1.2 Difficulties

In this section, we discuss a few alternate approaches to our problem and why they likely prove unfruitful. We will illustrate using problems of the form Eq. (1.2) when it is more straightforward.

1.2.1 Dynamic Programming

Dynamic programming is a natural first approach, as has been applied to other low treewidth problems. To explain the difficulty of achieving fully-polynomial-time fixed-parameter tractability in the optimization setting, we consider the following simplified problem: Given a graph $G = (V, E)$ with a convex function $f_e : \mathbb{R}^2 \to \mathbb{R}$ for every edge $e \in E$, consider the objective function on $\mathbb{R}^V$ defined by

$$f_G(x) = \sum_{ij \in E} f_{ij}(x_i, x_j). \tag{1.3}$$

To divide the problem into smaller one, we consider any small balanced vertex separator $S \subset V$; namely V is partition into three sets S, L and R such that there are no edges between L and R. We can write the objective function $f(x)$ by

$$f_G(x) = f_L(x) + f_R(x) + f_{G-E(L)-E(R)}(x),$$

where $f_T(x) = \sum_{ij \in E(T)} f_{ij}(x_i, x_j)$ and $E(T)$ is the set of edges with at least one end point in T. To minimize f_G, it suffices to fix x_S and recursively minimizing x on L and R, and minimize over all fixed x_S. Namely,

$$\min_x f_G(x) = \min_{x_S} f_{G-E(L)-E(R)}(x_S) + \widetilde{f_L}(x_S) + \widetilde{f_R}(x_S).$$

where $\widetilde{f_L}(x_S) = \min_{x_L} f(x_S, x_L)$ and $\widetilde{f_R}(x_S) = \min_{x_R} f(x_S, x_R)$. Here, we crucially use the fact that $f_{G-E(L)-E(R)}(x)$ depends only on the variables in S, but not L and R; the term f_L depends only on the variables in L and S, but not R; similarly for f_R. In general, if f is convex, then both $\widetilde{f_L}$ and $\widetilde{f_R}$ are convex functions on $\mathbb{R}^S$. Hence, the formula shows that we can solve the optimization problem by first constructing the reduced problem on $G[L]$ and $G[R]$, then solve a size $|S|$ optimization problem.

If the f_{ij}'s are all quadratic functions, then both $\widetilde{f_L}$ and $\widetilde{f_R}$ are quadratic functions, and it turns out they can be stored as matrices known as Schur complements. Hence, we can solve the problem with the approach described above; in fact, algebraic manipulation gives the sparse Cholesky factorization algorithm with runtime $\widetilde{O}(n \cdot \tau^2)$.

However, for general convex function f_G, it is not known how to store the functions $\widetilde{f_L}$ and $\widetilde{f_R}$ efficiently, and this will likely require runtime exponential in treewidth. At a high level, the reason is that before we solve the outer problem f_G, we do not know at which fixed x_S we should recurse on for $\widetilde{f_L}$ and $\widetilde{f_R}$. It is known that without adaptivity, exponentially many oracle calls are needed to minimize a general convex function [Nem94, BS18, BJL+19]. This suggests we should compute $\widetilde{f_L}$ and $\widetilde{f_R}$ recursively for each different x_S. However, it is likely that we need to access at least two different points x_S, and this already leads to runtime recursion $T(n) \geq 4T(n/2) + O(1)$ which is at least n^2. Therefore, dynamic programming appears to be inefficient for general convex optimization.

1.2.2 Scanning Through Variables

When the underlying structure of the variable dependencies is simple enough, a simple scan through the variables may suffice for the problem at hand; for example, [DGRW19] successfully applies this approach for function-fitting problems on a path. To illustrate, consider a problem of the form

$$\min_x F(x) \stackrel{\text{def}}{=} f_1(x_1, x_2, \cdots, x_k) + f_2(x_2, \cdots, x_{k+1}) + f_3(x_3, \cdots, x_{k+2}) + \cdots.$$

Suppose x^* is the unique minimizer of the function and $x_1^*, x_2^*, \cdots, x_{k-1}^*$ are given. By looking at the gradient of the function above at the first coordinate, we know that

$$\frac{\partial}{\partial x_1} F(x) = \frac{\partial}{\partial x_1} f_1(x_1^*, x_2^*, \cdots, x_k^*) = 0.$$

Since $x_1^*, x_2^*, \cdots, x_{k-1}^*$ is given, this is a one variable non-linear equation on x_k^* and it has a unique solution under mild assumptions. Solving these equations, we obtain x_k^*. Now, looking at $\frac{\partial}{\partial x_2} F(x)$, we have that

$$\frac{\partial}{\partial x_2} F(x) = \frac{\partial}{\partial x_2} f_1(x_1^*, x_2^*, \cdots, x_k^*) + \frac{\partial}{\partial x_2} f_2(x_2^*, x_3^*, \cdots, x_{k+1}^*) = 0.$$

Since we already know $x_1^*, \cdots, x_k^*$, this is again a one variable non-linear equation. Therefore, we can solve this problem one variable at a time.

This approach can be modelled by an underlying graph structure in the following sense: Each variable x_i is represented by vertex i of the graph, and $i \sim j$ if there is some a term f_k dependent on both i and j. We say a vertex i is solved if we know x_i^*. In the example above, the graph is a thick path, and if the first $k-1$ vertices are solved at the beginning, then we can follow the path to solve the remaining vertices one by one.

Unfortunately, this type of scan-based algorithm cannot be generalized. Consider a convex function of the form Eq. (1.3) where the graph G is a complete binary tree with n leaves. Let i be a vertex such that the subtree rooted at i is of height two containing four leaves. Observe that we cannot solve for the children of i by case analysis, if both i and the leaves are unsolved. Since there are $n/4$ many subtree of height two in G, at least $n/4$ many variables must be known at the beginning, before we can follow the graph structure to solve for the remaining variables. As such, this approach does not produce any meaningful simplification.

1.2.3 Tightening the Iterations Bounds for Interior Point Methods

Another natural approach for attacking the conjecture is to prove that existing polynomial time methods for linear program run faster automatically for graphs with low treewidth. Currently, there are two family of polynomial time algorithms – the ellipsoid method (more generally cutting plane methods) and interior point methods. For cutting plane methods, n iterations are needed in general, since the method only obtains one hyperplane per iteration, and we need n hyperplane simply to represent the solution even for the case $\mathrm{tw}(A) = O(1)$. In general, these hyperplanes are represented by dense vectors and will probably take n^2 time in total.

For interior point methods, the iteration bound is less clear since there is no information obstruction. In general, it is known that $O(\sqrt{n}\log(1/\varepsilon))$ iterations are needed to solve a linear program, and each iteration involves solving a linear system. For the case $d = \Theta(n)$ in particular, this bound has not been improved since the '80s. In fact, it has been shown that the standard interior point method used in practice indeed takes $\Omega(\sqrt{n}\log(1/\varepsilon))$ iterations in the worst case [MT14, ABGJ18], and some of these constructions have $O(1)$ treewidth. Even for concrete problems such as maximum flow, difficult instances for iterative methods often have treewidth $O(1)$ [KLOS14]. These lower bounds suggest that obtaining an optimization method with $\widetilde{O}(\mathrm{tw}^{O(1)}(A))$ iterations requires a substantively new algorithm.

1.2.4 Faster Iterations via Inverse Maintenance

Dual to the previous approach is the idea of speeding up each iteration of interior point methods. Each iteration of these methods require some computation or maintenance involving a term $(AH^{-1}A^\top)^{-1}$; previous work on linear programming focused on inverse maintenance techniques to accomplish this either explicitly or implicitly. In [CLS19, vdBLSS20, JSWZ20], the inverse is explicitly maintained and this takes at least d^2 time in total. [vdBLN+20, BLL+20] focus on IPM for the bipartite matching problem and the maximum flow problem, where a sparsified Laplacian system $AH^{-1}A^\top x = b$ is solved directly in each iteration and hence the whole algorithm takes at least d per step and $d^{1.5}$ time in total, where d is the number of vertices. It seems that either approach cannot lead to nearly linear time (when $n = \Theta(d)$).

In our setting, one natural approach is to maintain the Cholesky factorization $LL^\top = AH^{-1}A^\top$. This can be done in nearly-linear time in total, by combining ideas from numerical methods [Dav06] and previous algorithms mentioned above. Unfortunately, in general, almost any sparse update in H leads to $\Omega(d)$ changes in L^{-1}. Hence, it seems difficult to get a runtime faster than $d^{1.5}$ by just combining inverse maintenance with current knowledge of sparse Cholesky factorization.

1.3 Related Works

1.3.1 Algorithms With Runtime at Least Exponential to Treewidth

The notion of treewidth is closely tied to vertex separators; specifically, low treewidth graphs have small vertex separators, and this structure is amenable to a dynamic-programming approach for various problems. A number of NP-hard problems such as Independent Set, Hamiltonian Circuit, Steiner Tree, and Travelling Salesman can be solved with runtimes that depend only linearly on the problem size and exponentially on treewidth [Bod94] as the result of dynamic programming. They are extensively studied as part of the class of fixed-parameter tractible problems. In general, dynamic programming style approaches based on the tree decomposition unfortunately almost always lead to an exponential dependence on treewidth, even for polynomial-time solvable problems.

We point to one particular recent result here, which is a $2^{O(k^2)} \cdot n$ time algorithm to find k disjoint paths given k vertex pairs on a planar graph by [LMP+20]; it appears to be one of the first algorithms to exploit treewidth in a completely different way from dynamic programming.

1.3.2 Algorithms with Runtime Polynomial to Treewidth

When the problem is linear algebraic, such as solving linear systems and computing rank or determinant, the dynamic programming approaches often leads to runtime polynomial in treewidth.

For linear systems $Ax = b$, George first developed the method of nested dissection in [Geo73], which leveraged the underlying graph structure of A for the case where it is a grid. This was generalized by the seminal work of Lipton, Rose and Tarjan in [LRT79], to solving systems where A is any symmetric positive-definite matrix whose underlying graph has good balanced vertex separators. This was further extended by [AY13], to apply to non-singular matrices over any field. The Cholesky factorization of A is a key part of all aforementioned results; it has a long line of study in numerical analysis [Dav06], and is used as the default sparse linear system solver in various languages such as Julia, Matlab and Python. Our algorithm heavily relies on the machineries developed in this line of work.

Recently, [FLS+18] shows several problems can be reduced to matrix factorizations efficiently, including computing determinant, computing rank, and finding maximum matching, and this leads to $O(\tau^{O(1)} \cdot n)$ time algorithms where τ is the width of the given tree decomposition of the graph. The only non-linear algebraic $O(\tau^{O(1)} \cdot n)$ time problem we are aware of is Unweighted Maximum Vertex-Flow [FLS+18], which makes use of the crucial fact that the vertex separator size is directedly connected to the flow size to achieve a $\widetilde{O}(\tau^2 \cdot n)$ runtime.

When we are not restricted to nearly linear-time algorithms, [KPSZ18] combines nested dissection with support theory to solve the class of linear systems where A can be viewed as a higher dimensional graph Laplacian. For semidefinite programming, [ZL18] shows that interior point methods can solve certain classes of sparse semidefinite programs in $O(\tau^{6.5} n^{1.5} \log(1/\varepsilon))$ time, where τ is a sparsity parameter for SDPs analogous to treewidth for LPs. Both algorithms require solving super-logarithm many linear systems.

As far as we know, there is no previous work on linear programming in direct relation to treewidth.

1.3.3 Related Works in Optimization

A long line of work in the integer-linear programming community studies solving ILPs with respect to fixed treedepth, a parameter related but more restrictive than treewidth; indeed, ILPs can be weakly NP-hard even on instances with treewidth at most two. For an ILP with treedepth denoted $\mathrm{td}(A)$, [EHK+19] gives a weakly polynomial ILPs algorithm running in time $O(g(\min\{\mathrm{td}(A), \mathrm{td}(A^\top)\}) \cdot \mathrm{poly}(n))$, where g is at least some doubly-exponential function. This is followed-up by [CEH+20], giving a strongly polynomial algorithm running in $2^{O(\mathrm{td} \cdot 2^{\mathrm{td}})} \Delta^{O(2^{\mathrm{td}})} n^{1+o(1)}$time, where Δ is an upper-bound on the absolute value of an entry of A. [EHK+19] also discusses how an algorithm for ILP may be used to solve LP, [BKO19] builds on this to show an algorithm solving mixed integer-linear programs in time $f(a, \mathrm{td}(A))\,\mathrm{poly}(n)$, where a is the largest coefficient of the constraint matrix.

The optimization work in this paper is mainly inspired by techniques for general interior point methods, where the first proof of a polynomial time algorithm was due to Karmarkar [Kar84]. After multiple running time improvements [Kar84, Ren88, Vai89, NN91, LS19, CLS19, LSZ19, vdBLN+20, vdBLSS20], the current fastest IPMs are the results of [BLL+20] and [JKL+20]. We build on this recent line of work, where ideas from interior point methods, Johnson-Lindenstrauss sketching, and linear algebraic data structures are combined. For our dynamic data structure, we inspired by ideas similar to wavelets commonly found in signal processing [RV91], where we maintain IPM information across iterations at different scales, and process updates in every level of resolution.

2 Overview of Our Approach

In this section, we provide a high-level explanation of the overall approach and the techniques used. We discuss the more general convex formulation given in Theorem 1.3, but for simplicity, *we assume each* $n_i = 1$ *and* $m = n$ in the statement of the theorem; this allows us to directly refer to coordinates of all relevant matrices and vectors, rather than blocks. We revert back to blocks for the detailed proofs in later sections.

Our algorithm is based on interior point methods [NN94]. These methods solve the convex program by alternating between taking a gradient step, and projecting back to the constraint set $Ax = b$ under a suitable norm. The movement of x follows some path $x(t)$ inside the interior of the domain

K, with t decreasing by a $1-\Theta(1/\sqrt{n})$ factor every iteration, starting at some point $x(1) \in K$ and ending at the solution $x(0)$ we want to find. We use the common central path defined by

$$x(t) = \arg\min_{Ax=b} c^\top x + t\sum_{i=1}^{m} \phi_i(x_i) \tag{2.1}$$

where ϕ_i is a self-concordant barrier function (Definition A.3) on K_i that blows up on ∂K_i, namely, $\phi_i(x_i) \to +\infty$ as $x_i \to \partial K_i$. We simultaneously require the dual central path $s(t)$ Eq. (A.2), where s is maintained similarly to x.

The main difficulty is in following the path $x(t)$ efficiently. At timestep t of the central path, the current point x is updated by $x \leftarrow x + \delta_x$, where

$$\delta_x = \left(H_x^{-1} - H_x^{-1}A^\top (AH_x^{-1}A^\top)^{-1} AH_x^{-1}\right)\delta_\mu(x,s,t) \tag{2.2}$$

for some non-negative diagonal matrix H_x dependent on x, and vector δ_μ dependent on (x,s,t).

Our work therefore focuses on how to quickly and approximately maintain Eq. (2.2), and the accumulation of δ_x over the entire central path for the final solution x (and analogously for the dual solution δ_s and s). In Section 2.1, we follow existing results and approximate $AH_x^{-1}A^\top$ by $AH_{\overline{x}}^{-1}A^\top$ where $\overline{x}$ is an approximation of x. This ensures the change in $AH_{\overline{x}}^{-1}A^\top$ is low-rank in each iteration, which allows us to update $(AH_{\overline{x}}^{-1}A^\top)^{-1}$ implicitly and efficiently using existing results in Cholesky decomposition, outlined in Section 2.2. Unfortunately, the change of δ_x is dense even under a sparse change of $\overline{x}$. In Section 2.3, we propose a novel representation of δ_x, where only $\widetilde{O}(n\tau\log(1/\varepsilon))$ "coefficients" are changed during the central path. This allows us to maintain the solution x implicitly during the whole algorithm using only $\widetilde{O}(n\tau^2\log(1/\varepsilon))$ time. Finally, to maintain $AH_{\overline{x}}^{-1}A^\top$ close to $AH_x^{-1}A^\top$, we show how to detect large coordinate changes in this new representation in Section 2.4.

The main contributions of this paper is the novel representation of the central path and the data structure to maintain and detect changes under this representation. We believe that this representation will be of independent interest beyond convex programs with low treewidth.

2.1 Robust Central Path Method

Although each entry of H_x and δ_μ is updated at every step due to the dense update of x, a *robust central path* circumvents the need to recompute them completely in every iteration, and thus lowers the cost of each step. This idea has been used since the first interior point method [Kar84], and has led to significant recent progress in convex optimization [CLS19, vdB20, vdBLSS20, vdBLN+20, JSWZ20, JLSW20, BLL+20].

In Appendix A, we give our robust central path algorithm (Algorithm 16), which is a slight variant of the one presented in [LSZ19]. The changes are needed to support some extra approximation required by our new representation. Theorem A.1 shows that to solve problem Eq. (1.1), it suffices to implement $\widetilde{O}(\sqrt{n}\log(1/\varepsilon))$ approximate steps

$$\begin{aligned} x &\leftarrow x + (H_{\overline{x}}^{-1} - H_{\overline{x}}^{-1}A^\top (AH_{\overline{x}}^{-1}A^\top)^{-1} AH_{\overline{x}}^{-1})\delta_\mu(\overline{x},\overline{s},\overline{t}) \\ s &\leftarrow s + tA^\top (AH_{\overline{x}}^{-1}A^\top)^{-1} AH_{\overline{x}}^{-1}\delta_\mu(\overline{x},\overline{s},\overline{t}) \end{aligned} \tag{2.3}$$

where $\overline{x}, \overline{s}$ are vectors close to x, s, and $\overline{t}$ is a scalar close to t.

We only need to output (x, s) at the end, and do not need their exact values during the algorithm. Instead, it suffices to detect which coordinate has changed too much and update the approximation $(\overline{x}, \overline{s})$ accordingly. For interior point methods, if updated lazily, there are only a nearly linear number of coordinate changes to $\overline{x}$ and $\overline{s}$ during the whole algorithm:

$$\sum_{\text{IPM step } k} \|\overline{x}^{(k+1)} - \overline{x}^{(k)}\|_0 + \sum_{\text{IPM step } k} \|\overline{s}^{(k+1)} - \overline{s}^{(k)}\|_0 = \widetilde{O}(n \log(1/\varepsilon)).$$

Since $\overline{x}, \overline{s}$ are n-dimensional vectors, every coordinate is updated only roughly $\log(1/\varepsilon)$ times on average, and hence it allows for very efficient updates of the approximate steps. In particular, we have the following:

Throughout the algorithm, there are only $\widetilde{O}(n \log(1/\varepsilon))$ coordinate updates to $H_{\overline{x}}$. (2.4)

2.2 Cholesky Decomposition

In recent IPM works, each iteration involves either computing or maintaining $(AH_{\overline{x}}^{-1}A^\top)^{-1}$ of the update given in Eq. (2.3). However, this is too expensive for our setting, even for the case of constant treewidth. The change of the inverse between consecutive steps is usually a dense matrix (possibly small rank) which takes at least $\Omega(d)$ space to represent. In our algorithm, we instead maintain the sparse Cholesky decomposition.

$AH_{\overline{x}}^{-1}A^\top$ is a positive-definite matrix, and therefore admits a unique *Cholesky decomposition* $AH_{\overline{x}}^{-1}A^\top = LL^\top$, where L is a lower-triangular matrix with positive diagonal entries. The diagonal matrix $H_{\overline{x}}$ changes throughout the algorithm, but crucially, this only changes the entries of L, not its non-zero pattern. In Section 4, we discuss how to compute a permutation of the rows of A (and correspondingly entries of b), and an associated elimination tree $\mathcal{T}$ of A, which reflects the non-zero pattern of L. Suppose the rows of A has been reordered, and then $AH_{\overline{x}}^{-1}A^\top$is factored into $LL^\top$. Let τ be the height of the elimination tree $\mathcal{T}$. The following properties hold (Theorem 4.1):

- $\mathcal{T}$ is a tree on d vertices $\{1, \ldots, d\}$, with vertex i representing row/column i of L.
- The columns of A, L, and L^{-1} are all τ-sparse.
- The non-zero entries of $L^{-1}e_i$ and Le_i are respectively subsets of the path from vertex i to the root of $\mathcal{T}$. Furthermore, they can be computed in $\tau^{O(1)}$ time.
- For a single coordinate change in $H_{\overline{x}}$, it takes $\tau^{O(1)}$ time to update L exactly.

Now, we can rewrite $(AH_{\overline{x}}^{-1}A^\top)^{-1}$ as $L^{-\top}L^{-1}$, and take advantage of the sparsity of L via $\mathcal{T}$ in the algorithm. In particular, by Eq. (2.4), we have the following:

Throughout the algorithm, there are only $\widetilde{O}(n\tau^{O(1)} \log(1/\varepsilon))$ coordinate updates to L. (2.5)

2.3 Multiscale Representation of the Central Path

To implement the central path steps, we want all variables to change in a sparse way, so we can update quickly between iterations. In particular, we want to represent x (similarly s) implicitly by

$$x = x_0 + Bh$$

for some vectors x_0, h and some basis matrix B.

When $H_{\overline{x}}$ and δ_μ admit only sparse changes between steps, the first term $(H_{\overline{x}}^{-1}\delta_\mu)$ of Eq. (2.3) is easy to compute explicitly, which we do and maintain as part of x_0. Part of the second term given by $h \stackrel{\text{def}}{=} L^{-1}AH_{\overline{x}}^{-1/2}\delta_\mu$ is similarly easy to maintain, due to the fact that each column of L^{-1} and A has sparsity τ and can be obtained in $\tau^{O(1)}$ time. However, computing and maintaining $H_{\overline{x}}^{-1}A^\top L^{-\top}h$ explicitly is still costly. The first key observation of this paper is that the representation

$$x = x_0 + H_{\overline{x}}^{-1}A^\top L^{-\top}h$$

has the following properties:

1. For any i, we can compute x_i in $\tau^{O(1)}$ time.

   Note that $x_i = (x_0)_i + h^\top L^{-1}AH_{\overline{x}}^{-1}e_i$. Since we know each column of A is τ-sparse, we can compute $AH_{\overline{x}}^{-1}e_i$ in $O(\tau)$ time and it is $O(\tau)$ sparse. Hence, $L^{-1}AH_{\overline{x}}^{-1}e_i$ is just a mixture of $O(\tau)$ many columns of L^{-1} and since each column of L^{-1} is $O(\tau)$ sparse, we can compute it in $\tau^{O(1)}$ time. This gives a $\tau^{O(1)}$ time algorithm to compute x_i.

2. After a sparse update to L and $H_{\overline{x}}$, we can maintain the representation in $\tau^{O(1)}$ time.

   More precisely, given $x = x_0 + H_{\overline{x}}^{-1}A^\top L^{-\top}h$, $L^{\text{new}} = L + \Delta L$, $H_{\overline{x}}^{\text{new}} = H_{\overline{x}} + \Delta H_{\overline{x}}$, then we can find x_0^{new} and h^{new} in $\tau^{O(1)}$ time such that $x = x_0^{\text{new}} + (H_{\overline{x}}^{\text{new}})^{-1}A^\top(L^{\text{new}})^{-\top}h^{\text{new}}$.

   For the change of $H_{\overline{x}}^{\text{new}}$, we can simply set $x_0^{\text{new}} = x_0 + (H_{\overline{x}}^{-1} - (H_{\overline{x}}^{\text{new}})^{-1})A^\top(L^{\text{new}})^{-\top}h$. Since $(H_{\overline{x}}^{-1} - (H_{\overline{x}}^{\text{new}})^{-1})$ is sparse, we can compute the term $(H_{\overline{x}}^{-1} - (H_{\overline{x}}^{\text{new}})^{-1})A^\top(L^{\text{new}})^{-\top}h$ by the approach from Property 1 (computing the formula from left to right).

   For the change of L^{new}, we simply need to find h^{new} such that $(L^{\text{new}})^{-\top}h^{\text{new}} = L^{-\top}h$. Rearranging, we have $h^{\text{new}} = (L^{\text{new}})^\top L^{-\top}h = h + (\Delta L)^\top L^{-\top}h$. Again, since $(\Delta L)^\top$ is sparse, we can compute it from left to right.

From now on, we call h the *multiscale coefficients*. Since there are only $\widetilde{O}(n\tau^{O(1)}\log(1/\varepsilon))$ coordinates in $H_{\overline{x}}$ and L (Eq. (2.4), Eq. (2.5)), Property 2 shows that we can maintain this representation in $\widetilde{O}(n\tau^{O(1)}\log(1/\varepsilon))$ time. Furthermore, we have:

Throughout, there are only $\widetilde{O}(n\tau^{O(1)}\log(1/\varepsilon))$ coordinates updates to the multiscale coefficients. (2.6)

Finally, Property 1 shows that these multiscale coefficients is as good as explicit representation since we can read any entry in $\tau^{O(1)}$ time. Suppose we know which coordinates of x deviated from $\overline{x}$ significantly, then we can simply use Property 1 to update $\overline{x}$.

Combining this with heavy-hitter ideas, we can easily get an algorithm of time $\widetilde{O}(n^{1.25}\tau^{O(1)}\log(1/\varepsilon))$ (See [Ye20] for an earlier draft version of this paper).

2.4 Data Structures for Maintaining Multiscale Representation

A key component of our algorithm revolves around finding which coordinates of x deviate significantly from $\overline{x}$. Specifically, we want to find large coordinate in $H_{\overline{x}}^{1/2}(x - \overline{x})$, where the term $H_{\overline{x}}^{1/2}$ is to measure the deviation in a correct norm required by the interior point method.

Similar to the discussion above, we can maintain $H_{\overline{x}}^{1/2}(x - \overline{x})$ implicitly as $x_0 + \mathcal{W}^\top h$ for some sparsely changing vectors x_0, where $\mathcal{W} \stackrel{\text{def}}{=} L^{-1}AH_{\overline{x}}^{-1/2}$ and $h \stackrel{\text{def}}{=} L^{-1}AH_{\overline{x}}^{-1/2}\delta_\mu$. Here, we focus on discussing the change of the term $\mathcal{W}^\top h$; analogous ideas are used for x_0.

First, observe that we cannot maintain $\mathcal{W}^\top h = H_{\overline{x}}^{-1/2} A^\top L^{-\top} h$, as the rows of A and L^{-1} may be dense. However, it is relatively easy to maintain $v^\top \mathcal{W}^\top h$ for any vector v, since $v^\top \mathcal{W}^\top h = h^\top \mathcal{W} v = h^\top L^{-1} A H_{\overline{x}}^{-1/2} v$, and we can exploit the column sparsity of A and L^{-1}. If we use a Johnson-Lindenstrauss sketching matrix Φ in place of v and maintain $\Phi \mathcal{W}^\top h$, then this allows us to quickly estimate $\|\mathcal{W}^\top h\|_2^2$.

We construct a data structure called the *sampling tree* $\mathcal{S}$ (Definition 6.13), based on the elimination tree $\mathcal{T}$, to store a family of sketches of the form $\Phi \mathcal{W}^\top h$. Specifically, $\mathcal{S}$ is a constant-degree tree with leaves given by the set $[n]$, where leaf i corresponds to $(\mathcal{W}^\top h)_i$. For any node $v \in V(\mathcal{S})$, let $\chi(v) \subseteq [n]$ denote the set of all leaves in the subtree rooted at v, and let $\Phi_{\chi(v)}$ denote the JL sketching matrix restricted to the indices given by $\chi(v)$. Then at node v, we maintain $\Phi_{\chi(v)} \mathcal{W}^\top h$. By JL properties, we can estimate $\|(\mathcal{W}^\top h)|_{\chi(v)}\|_2^2$ at each node v; in other words, we have the approximate ℓ_2-norm of various subvectors of $\mathcal{W}^\top h$ of different lengths. Using this information, we can apply the standard sampling technique of walking down $\mathcal{S}$ from the root to a leaf:

$$\text{We can sample for a coordinate } i \text{ proportional to } (\mathcal{W}^\top h)_i^2 \text{ in } O(\text{height}(\mathcal{S})) \leqslant \widetilde{O}(\tau) \text{ steps.} \tag{2.7}$$

A large coordinate $(\mathcal{W}^\top h)_i$ means x_i and $\overline{x}_i$ differ significantly. Then we compute x_i exactly and update $\overline{x}_i \leftarrow x_i$.

$\overline{x}$ and δ_μ are updated every iteration, hence, we must maintain the latest $\mathcal{W}$ and h to support sampling using $\mathcal{S}$. As there are $\widetilde{O}(n\tau)$ nodes in $\mathcal{S}$, we do not have enough time to update $\Phi_{\chi(v)} \mathcal{W}^\top h$ at every node v every iteration. However, observe that we only need to know the latest value of $\|(\mathcal{W}^\top h)|_{\chi(v)}\|_2^2$ during the sampling procedure, and as $\mathcal{S}$ is a constant-degree tree, sampling once only visits $O(\text{height}(\mathcal{S}))$ nodes in $\mathcal{S}$. So we may rely on a form of lazy maintenance. Here, we focus on discussing a coordinate change in $\overline{x}$; analogous ideas are used for changes in δ_μ.

For a single coordinate change in $\overline{x}_i$, we need to update $H_{\overline{x}}^{-1/2}$ and L, but crucially the update only affects $\Phi_{\chi(v)} \mathcal{W}^\top h$ at select nodes of $\mathcal{S}$. Specifically, $H_{\overline{x}}^{-1/2}$ changes by a single entry, which causes the value of $\Phi_{\chi(v)} \mathcal{W}^\top h$ to change only if $i \in \chi(v)$. Hence, for each entry update of $H_{\overline{x}}^{-1/2}$, we only need to update a path in $\mathcal{S}$. On the other hand, a change in $\overline{x}_i$ causes $O(\tau)$ columns of L to update. Each column of L has a corresponding node $u \in \mathcal{S}$, such that for any $v \in \mathcal{S}$, the value $\Phi_{\chi(v)} \mathcal{W}^\top h$ maintained at v changes only if u is an ancestor or descendant of v. Hence, for each column update of L, we split the effect into two:

1. "upwards" effect: The updates to ancestors of u. Since u has at most height$(\mathcal{S})$ many ancestors, we have sufficient time to update these sketches immediately.
2. "downwards" effect: The updates to descendants of u. We cannot afford to update the whole subtree rooted at u; hence, we delay the update. A node can have height$(\mathcal{S})$-many delayed updates, with one per ancestor. Then, we can perform all the delayed updates in $\tau^{O(1)}$ time when it is accessed during sampling.

Combined with Eq. (2.7), we have:

$$\mathcal{S} \text{ can be maintained to support sampling a coordinate } i \text{ in } \tau^{O(1)} \text{ amortized time.} \tag{2.8}$$

To further lower the cost for the case that τ is large, we present a more involved construction of a sampling tree using heavy-light decompositions with height $O(\log n)$ (Section 6.6).

2.5 Proofs of Main Theorems

We now link the various pieces of this paper together to prove Theorems 1.1 to 1.3.

All three settings require a preprocessing step to find a suitable reordering of the constraints $Ax = b$. Constructing the graph G_A from the non-zero pattern of $AH_{\overline{x}}^{-1}A^\top$ takes $O(n\tau^2)$ time. Then by Theorem 4.1, we can find a reordering of the rows of A and a binary elimination tree $\mathcal{T}$ for the corresponding Cholesky decomposition: when a width-τ tree decomposition of G_A is given as in Theorem 1.1, this takes $\widetilde{O}(n\tau)$ time and produces an elimination tree of height $\widetilde{O}(\tau)$. Otherwise, we use [BGS21] to obtain a tree of height $\widetilde{O}(\mathrm{tw}(G_A))$ which takes $\widetilde{O}(n \cdot \mathrm{tw}(G_A))$ time.

We can reduce the linear program of Theorem 1.1 to a convex program of the form Eq. (CP), before invoking Theorem A.1 for the interior point method. Specifically, for the LP given in Theorem 1.1, each convex set K_i is the interval $[u_i, l_i]$ with $n_i = 1$; we have that $\phi_i(x_i) = -\log(u_i - x_i) - \log(x_i - l_i)$ is a 1-self-concordant barrier function for K_i minimized by $x_i = (l_i + u_i)/2$; without loss of generality, we may set $w = \mathbf{1}_m$ and have $\kappa = n$.

For Theorem 1.3, we can invoke Theorem A.1 directly. When the barrier functions are not given, we use the universal barrier ϕ_i with self-concordance n_i for each i (Appendix A.8); since $n_i = O(1)$, we can find the minimizer x_i of ϕ_i as a preprocessing step in $O(1)$ time. As in the LP case, we set $w = \mathbf{1}_m$ and have $\kappa = \sum_{i=1}^m w_i n_i = n$.

Theorem A.1 shows that the robust interior point method given as Algorithm 16 produces the approximate solution as required, and terminates within $O(\sqrt{\kappa}\log(\kappa/\varepsilon \cdot R/r)) = O(\sqrt{n}\log(R/(\varepsilon r)))$ steps.

The data structure CentralPathMaintenance is used to perform one step of the central path exactly as we need. The cost of a step is analyzed in Theorem 6.1. Let τ denote the height of the elimination tree $\mathcal{T}$ computed during preprocessing, and let $N = O(\sqrt{n}\log(n/\varepsilon \cdot R/r))$ denote the number of central path steps. To begin, we initialize the data structure via Initialize in time $O(n\tau^2 \log^4(N))$. At timestep t, Algorithm 16 needs to find $\overline{x}, \overline{s}, \overline{t}$ and compute updates to x, s, t, which is all accomplished by invoking MultiplyAndMove(t). As MultiplyAndMove is called N times over the entire algorithm, the total runtime is $O(Nn^{1/2} + n\log(t_{\max}/t_{\min})) \cdot \tau^2 \operatorname{poly}\log(N) = \widetilde{O}(n\tau^2 \log(1/\varepsilon))$. At the very end, Output outputs the result (x, s) exactly in time $O(n\tau^2)$.

Finally, for the setting of Theorem 1.3, since we use universal barrier functions ϕ_i for $i \in [m]$, computing $\nabla\phi_i$ and $\nabla^2\phi_i$ as part of the IPM take $O(\log(Rn/r))$ time by Remark A.2. Hence, we incur an additional $\log(R/r)$ factor in the overall runtime.

2.6 Wavelet Interpretation

Now, we explain the geometric meaning of this multiscale representation and its connection to wavelets. The rest of this subsection can be safely skipped as this view is not used in any proof.

In wavelet theory, a complex signal is represented as a linear combination of shifted and scaled versions of a simple signal. In our context, we are representing the pending change δ_x by various linear combinations of vectors $H_{\overline{x}}^{-1}A^\top L^{-\top} e_i$. In the case that A is the incidence matrix of a path, the elimination tree is simply the complete binary tree, that when flattened in a breadth-first fashion, returns the original path. Here the vertices at different levels of the elimination tree exactly correspond to dyadic intervals of different lengths, while vertices at the same level correspond to dyadic intervals of the same length but with a "time" shift. When $H = I$, the vector $H^{-1}A^\top L^{-\top} e_i$ in fact looks quite similar to the Haar basis (See Fig. 2.1).

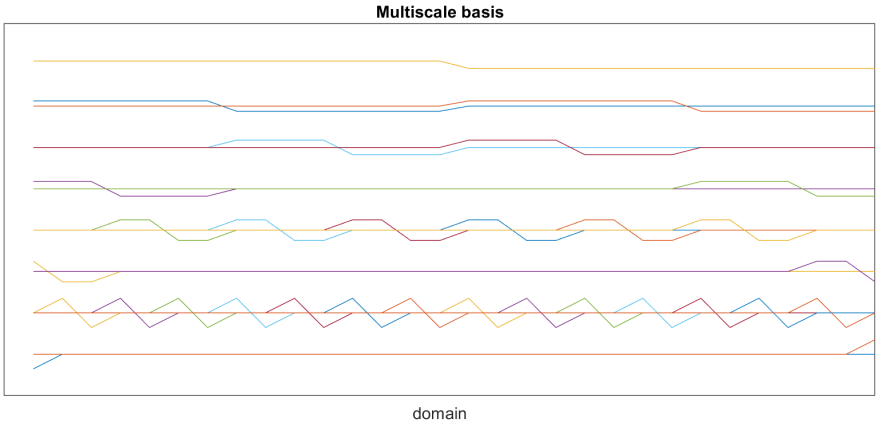

Figure 2.1: The multiscale basis $\{A^\top L^{-1} e_i\}_i$ where A is the incidence matrix of a path. We group the basis by size and shift the basis according to its size for clarity.

More precisely, we define the wavelet transform $\mathcal{W} \stackrel{\text{def}}{=} L^{-1}AH^{-1/2}$, where $\mathcal{W}$ maps the signal in the original space to the coefficient space, with the basis elements corresponding to vertices of the elimination tree. Here we list only some similarities between this and the standard wavelet transform:

1. Applying the wavelet and inverse wavelet transform recovers the signal:

$$\mathcal{W}^\top \mathcal{W} h = h \text{ for any } h \in \text{Range}(\mathcal{W}^\top).$$

2. For each point in the original space, it is only covered by a few basis elements with different scales:

$$\mathcal{W} e_i \text{ lies on } O(\tau) \text{ paths on the elimination tree.}$$

3. For each basis element, it covers the original space with different scales:

$$\text{The support of } \mathcal{W}^\top e_i \text{ is roughly a subtree.}$$

4. There is a fast wavelet transform:

$$\text{We can apply } \mathcal{W} \text{ and } \mathcal{W}^\top \text{to any vector in } n\tau^{O(1)} \text{ time.}$$

The key difference is that our wavelet basis does not represent any signal, but only the signal in the range of $H^{-1/2}A^\top$.

To illustrate the multiscale coefficient, we consider the fused lasso problem[4] in Fig. 2.2. The central path is a 1-D signal that smoothly moves from a constant at $t = 1$ to a recovered signal at $t = 10^{-3}$. Following this smooth transition is expensive, hence we consider the robust central path, which is noisier but converges to the same recovered signal. The noise comes from the approximation of x by $\overline{x}$[5]. However, maintaining this robust central path is still quite expensive, because all coordinates

[4]We pick this problem because it is easy to represent the whole central path as a surface plot.

[5]We emphasize x_t is the robust central path, not $\overline{x}_t$. We only use $\overline{x}$ to approximate the linear systems. We cannot use $\overline{x}$ as the solution because it does not satisfy the condition $Ax = b$.

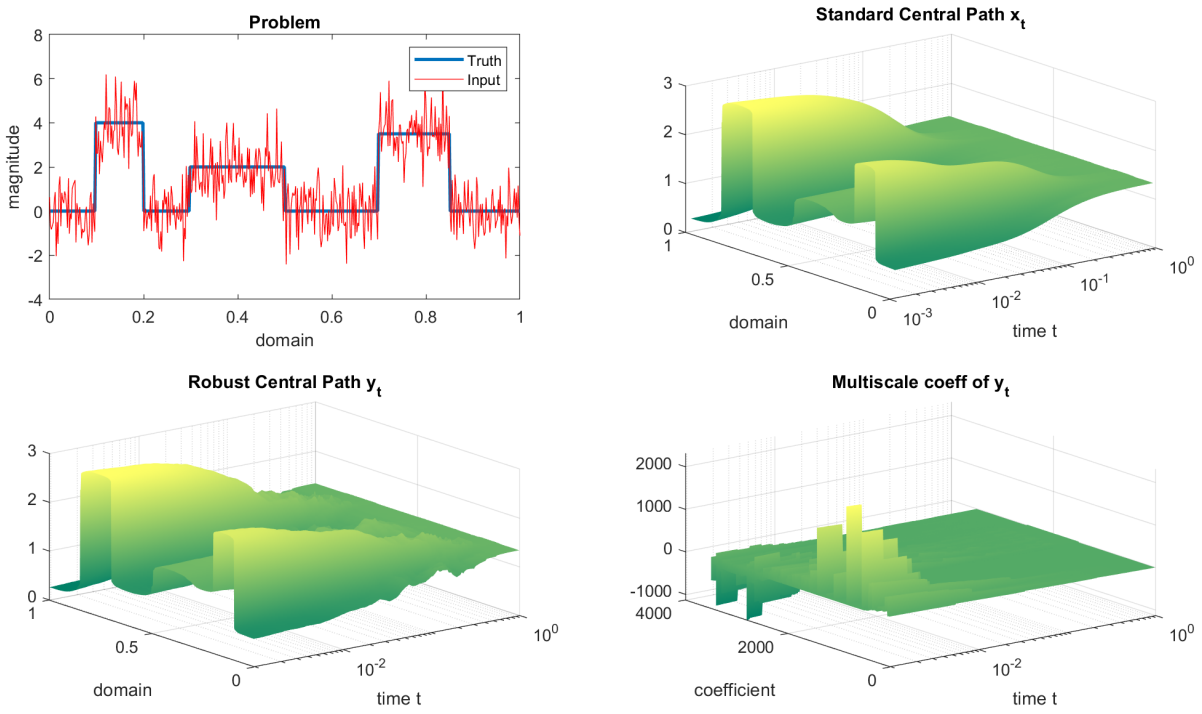

Figure 2.2: Consider the fused lasso problem, with the upper left figure showing the input and true signals. The upper right figure shows the standard central path for this problem. The lower left figure shows the robust central path implicitly maintained in our algorithm. The lower right figure shows the multiscale coefficients we explicitly maintain in the algorithm.

change at every step in the robust central path. The crux of this paper is representing the robust central path by the multiscale basis $\{A^\top L^{-1} e_i\}_i$, under which the coefficient changes sparsely.

Finally, we note that choosing this wavelet basis is quite natural from the view of physical science. Consider applying the interior point method for the maximum flow problem on a path: In this case, the linear system $AH_x^{-1}A^\top$ is simply a weighted Laplacian on a path, and the central path is simply the solution of some partial differential equations. Numerical differential equations in general face the same computation issues as us, that is, to represent the solution in a sparse way. It has been known since the '90s that both the Laplacian (more generally, elliptical differential equation), its inverse, and the solution can be represented sparsely using the wavelet basis such as [BCR91, DKO97]. The idea of using wavelets to approximate the solution has been applied to many partial differential equations [SCHO13]. Arguably, this paper shows that the idea also applies to the "partial differential equation" defined by a central path, where the self-concordance theory ensures everything is well-behave enough for this to happen.

3 Preliminaries

In this section, we introduce the notations we used throughout the paper.

We say a symmetric matrix $A \in \mathbb{R}^{n\times n}$ is positive semidefinite (PSD) if $x^\top Ax \geq 0$ for all $x \in \mathbb{R}^n$ and positive definite (PD) if $x^\top Ax > 0$ for all $x \in \mathbb{R}^n$. For symmetric matices $A, B \in \mathbb{R}^{n\times n}$, we use

$A \succeq B$ to indicate that $A - B$ is a PSD matrix. We define operators $\preceq, \succ, \prec$ analogously.

For a vector $v \in \mathbb{R}^n$, we use $\|v\|_2$ to denote its euclidean norm. We use $\|v\|_0$ to denote the number of non-zero entries in v. For a PSD matrix $A \in \mathbb{R}^{n \times n}$, we let $\|v\|_A = \sqrt{v^\top A v}$.

We use e_i to denote the standard unit vector. We use $\mathbf{0}_n, \mathbf{1}_n$ to denote all-zero and all-one vectors in $\mathbb{R}^n$. We define $\mathbf{0}_{m \times n}$ and $\mathbf{1}_{m \times n}$ analogously. We write $I_n \in \mathbb{R}^n$ to denote the identity matrix. When dimensions are clear in the context, we drop the subscripts.

We use upper case letters to denote matrices, and lower cases for vectors and scalars. We use $A \cdot B$ to denote the matrix-matrix multiplication and $A \cdot x$ to denote the matrix-vector multiplication for readability. When readability is not an issue, the operator $\cdot$ is omitted. To distinguish from the vector dot product, we always use $x^\top y$.

For any matrix $A \in \mathbb{R}^{m \times n}$, we use A_S to denote the matrix restricted to the column (block) indices given by the set S. We say a block diagonal matrix $A \in \oplus_{i=1}^{m} \mathbb{R}^{n_i \times n_i}$ if A can be written as

$$A = \begin{bmatrix} A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_m \end{bmatrix}$$

where $A_1 \in \mathbb{R}^{n_1 \times n_1}$, $A_2 \in \mathbb{R}^{n_2 \times n_2}$, $\ldots$, and $A_m \in \mathbb{R}^{n_m \times n_m}$.

For any matrix M, we use M_i to denote the i-th column or column block, and $\mathcal{M}_i$ to denote the non-zero pattern of the i-th column or column block, i.e. it is a set of row indices. For example, $j \in \mathcal{A}_i$ if row j of A is non-zero in a column in block i. We use M^i to denote the i-th row of M and $\mathcal{M}^i$ to denote the non-zero pattern of the i-th row.

We use $\widetilde{O}(\cdot)$ to hide $\log^{O(1)}(n)$ and $(\log\log(1/\varepsilon))^{O(1)}$ factors. We similarly define $\widetilde{\Omega}$ and $\widetilde{\Theta}$. For any positive integer n, we let $[n]$ denote the set $\{1, 2, \ldots, n\}$. We use $\sinh x$ to denote $\frac{e^x - e^{-x}}{2}$ and $\cosh x$ to denote $\frac{e^x + e^{-x}}{2}$.

For a tree $\mathcal{T} = (V, E)$, we write $v \in \mathcal{T}$ or $v \in V(\mathcal{T})$ interchangeably to denote $v \in V$. For a rooted tree $\mathcal{T}$, we say a set S *lies on a path of* $\mathcal{T}$ if there is a path $\mathcal{P}$ from the root of $\mathcal{T}$ to some node in $\mathcal{T}$, and $S \subseteq \mathcal{P}$. We use $\mathcal{P}(i)$ to denote the set of vertices on the path from vertex i to the root in $\mathcal{T}$, and $\mathcal{D}(i)$ to denote the set of vertices in the subtree rooted at i.

We use the convention that a tree consisting of a single vertex has height 1.

In our pseudocode, we use font to denote data structure objects, FONT to denote functions and object types, and regular math font to denote other variables stored in a data structure. Throughout our algorithms, we assume there is a basic object type LIST which gives us random access to all its elements. We write DATASTRUCTUREA extends DATASTRUCTUREB in the object-oriented programming sense: that is, DATASTRUCTUREA contains all the variables and functions from DATASTRUCTUREB, accessible either directly by name when there is no naming conflict, or with the keyword super.

4 Elimination Tree

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. In this section, we review some existing techniques [BGHK95, Dav06] for computing a permutation of the linear constraints $Ax = b$, for $A \in \mathbb{R}^{d \times n}$. Our goal is to ensure that after permuting the rows of A, the Cholesky factorization $LL^\top = AH_{\overline{x}}^{-1}A^\top$ will have certain desired sparsity patterns, which is then reflected in an associated *elimination tree*.

Let the rows of A be labelled $1, 2, \ldots, d$. Recall we are given block-diagonal structure $n = \sum_{i=1}^m n_i$ for A and $H_{\overline{x}}$. We identify A in *column blocks*, with A_i denoting the n_i columns in block i. We simply use H in the remainder of this section, as we only require its non-zero pattern which is independent of $\overline{x}$; H is an $n \times n$ block-diagonal positive-definite matrix, and without loss of generality, we may assume all entries in each block of H are non-zero. In this case, observe that the n_i columns in block i of $AH^{-1/2}$ all have the same non-zero pattern, which we denote by $\mathcal{A}_i \subseteq [d]$. We use the convention that a tree on one vertex has height 1.

The main results of this section is as follows. We give two cases for the runtime, corresponding to Theorem 1.1 with a given tree decomposition, and Theorem 1.2 without the decomposition.

Theorem 4.1. *Let A be a $d \times n$ matrix with block structure $n = \sum_{i=1}^m n_i$, and suppose we are given the generalized dual graph G_A. We can compute a permutation P of the rows of $AH^{-1/2}$ (equivalently, an ordering $\pi : [d] \mapsto [d]$), and a tree $\mathcal{T}$ on d vertices, so that in the Cholesky factorization $PAH^{-1}A^\top P^\top = LL^\top$,*

- *each vertex of $\mathcal{T}$ corresponds to a row/column of the Cholesky factor L, and*
- *the non-zero entries of $Le_i, L^{-1}e_i$ are respectively subsets of the path from vertex i to the root in $\mathcal{T}$.*

The second property implies the column sparsity of L and L^{-1} are bounded by $\text{height}(\mathcal{T})$. *The following runtimes and associated tree height are possible:*

1. $\widetilde{O}(n \cdot \tau)$ *if a tree decomposition of the dual graph G_A of width τ is given.* $\text{height}(\mathcal{T}) = O(\tau \log n)$.
2. $\widetilde{O}((n \cdot tw(G_A))^{1+o(1)})$ *without a given tree decomposition.* $\text{height}(\mathcal{T}) = O(tw(G_A)\,\text{polylog}\, n)$, *where $tw(G_A)$ is the treewidth of G_A*[6].

Proving Theorem 4.1 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.

4.1 Dual Graph and Treewidth

We begin with the necessary definitions for completion.

Definition 4.2. Recall the *generalized dual graph* of the matrix $A \in \mathbb{R}^{d \times n}$ with block structure $n = \sum_{i=1}^m n_i$ is the graph $G_A = (V, E)$ with $V = \{1, \ldots, d\}$, and $ij \in E$ if and only if $A_{i,r} \neq \mathbf{0}$ and $A_{j,r} \neq \mathbf{0}$ for some r, where we use $A_{i,r}$ to mean the submatrix of A in row i and column block r.

[6]Here, we defined the treewidth of a directed graph by simply ignoring the directions of the edges. This definition is compatible with first writing the directed max-flow as an LP, and then taking the treewidth of the dual graph of the constraint matrix.

Equivalently, G_A is the dual graph of $AH^{-1/2}$ by the definition in Theorem 1.1. In particular, the non-zero pattern of $(AH^{-1/2})(AH^{-1/2})^\top$ is precisely the adjacency matrix of G_A.

Definition 4.3. 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 $V(G)$ called *bags* labelling the vertices of T, such that

1. $\bigcup_{t \in V(T)} X(t) = V(G)$,
2. for each $v \in V(G)$, the nodes $t \in V(T)$ with $v \in X(t)$ induces a connected subgraph of T, and
3. 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 *treewidth* of G is the minimum width over all tree-decompositions of G. Intuitively, the treewidth of a graph captures how close the graph is to being a tree.

The following structural results about treewidth are elementary.

Lemma 4.4. *If G is a graph on d vertices and $tw(G) = \tau$, then $|E(G)| \leqslant d\tau$.* □

Lemma 4.5. *If G' is a subgraph of G, then $tw(G') \leqslant tw(G)$.* □

Lemma 4.6. *For the complete graph on k vertices, $tw(K_k) = k - 1$.* □

There are some basic relations between the sparsity of a matrix A and the treewidth of its dual graph:

Lemma 4.7. *Any block of A with sparsity τ induces a clique of size τ in G_A. It follows that $\max\{|\mathcal{A}_i| : A_i$ a column block of $A\} \leqslant tw(A) + 1$.* □

Treewidth is a natural structural parameter of a graph, with close connections to graph algorithms of a recursive nature. At a high level, it is generalized by the notion of well-separable graphs. We are particularly interested in its connection to vertex separators.

4.2 Balanced Vertex Separator

Definition 4.8. Let $G = (V, E)$ be a graph. For any $W \subseteq V$ and $1/2 \leqslant \alpha < 1$, an *α-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* $\alpha \cdot |W|$ vertices of W. In the particular case when $W = V$, we call the separator an *α-vertex separator of G*. The *separator number* of G is the maximum over all subsets W of V of the size of the smallest $1/2$-vertex separator of W in G.

We sometimes denote an α-vertex separator S by (G_1, S, G_2), where $V(G_1) \cup S \cup V(G_2) = V(G)$, and G_1 and G_2 are disconnected in $G \setminus S$.

Similar to treewidth, separator numbers are monotone.

Lemma 4.9. *Let G' be a subgraph of G. For any constant $1/2 \leqslant \alpha < 1$, the size of the smallest α-vertex separator of G' is at most that of G.* □

The following theorem relates the treewidth of a graph and the separator number.

Theorem 4.10 ([BGHK95], Lemma 6)**.** *If G is a graph with treewidth 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 and elimination tree.

4.3 Elimination Tree

Let $G = (V, E)$ be the generalized dual graph of A, that is, its adjacency matrix is given by the non-zero pattern of $AH^{-1}A^\top$. 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_π^+ ← (V, E)
for i from 1 to n do
    for each v ∈ V such that π(v) > i do
        if ∃ a path P from π^{-1}(i) to v in G, and all u ∈ P − v satisfies π(u) ⩽ i then
            add an edge between π^{-1}(i) and v in 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 $PAH^{-1}A^\top P^\top = LL^\top$, 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 the edge between $\pi^{-1}(i)$ and 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 $\pi(w) < \min\{\pi(u), \pi(v)\}$.

Definition 4.11 (Elimination Tree)**.** The *elimination tree corresponding to π* is the tree $\mathcal{T}$ 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 4.12 ([Sch82])**.** *Let L be the Cholesky factor for the matrix $AH^{-1}A^\top$. Let L_j denote the j-th column block. The non-zero 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 4.13. 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.

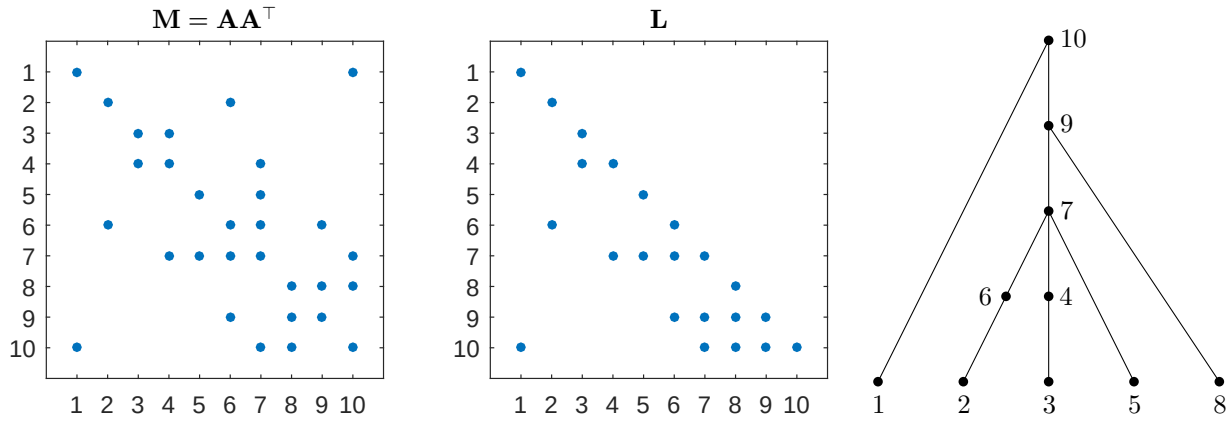

Figure 4.1: Each blue dot represents a non-zero entry in the matrix.

Lemma 4.14. *If uw is an edge in G, then in any elimination tree $\mathcal{T}$ 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 $\mathcal{T}$ of G, the vertices of K all lie on the same path from some leaf of $\mathcal{T}$ to the root.* □

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

Theorem 4.15 ([BGHK95], Theorem 12)**.** *Every graph G on n vertices satisfies*

$$\textit{separator number} - 1 \leqslant \textit{treewidth} \leqslant \textit{min elimination tree height} \leqslant \textit{separator number} \cdot \log n.$$

This structural theorem indicates that we can construct an elimination tree $\mathcal{T}$ of G_A and bound its height as a function of $tw(A)$. Specifically, we use the standard technique of recursively computing vertex separators, and using them to generate an ordering π of the vertices of $V(G_A)$. Rather than constructing the elimination tree according to the definition however, we construct a slightly taller bounded-degree tree, and show it still reflects the sparsity conditions of the Cholesky factor.

In Algorithm 2, we use a list notation $(v_1, \ldots, v_k)$ to denote the ordering π of a set of vertices $\{v_1, \ldots, v_k\}$ with $\pi(v_i) = i$. We use $+$ to denote the concatenation of two lists.

Algorithm 2 Constructing an Elimination Order and Tree

procedure MAKEELIMORDERANDTREE(G)
  if $|V(G)| \leqslant f(\tau)$ **then**
    let π be an arbitrary ordering of $V(G)$
    construct a path on $V(G)$ according to π, let u be the last vertex of the ordering/path
    return (π, u)
  end if
  $(G_1, S, G_2) \leftarrow$ APPROXBALANCEDSEPARATOR(G)
  $(\pi_1, v_1) \leftarrow$ MAKEELIMORDERANDTREE(G_1)
  $(\pi_2, v_2) \leftarrow$ MAKEELIMORDERANDTREE(G_2)
  $\pi \leftarrow$ arbitrary ordering of S
  construct a path on S according to π, let u be the first vertex of the ordering/path and v the last
  set u as the parent of v_1 and v_2
  return $(\pi_1 + \pi_2 + \pi, v)$
end procedure

Theorem 4.16. *Let $G = (V, E)$ be a graph on n vertices with treewidth τ, and let $f(\tau)$ be some function of 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 α-vertex separator of H for some universal constant $1/2 \leqslant \alpha < 1$, and $|S| \leqslant f(\tau)$,*
3. *the algorithm runs in time $T_{sep}(k)$.*

Then Algorithm 2 constructs an elimination order and a binary tree $\mathcal{T}$ for G of height at most $O(f(\tau) \cdot \log n)$, in time $\widetilde{O}(T_{sep}(n))$.

Proof. We have the following straightforward analysis of makeElimOrderAndTree: Let $T(k)$ denote the runtime on a graph with k vertices. Then

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

Solving the recurrence, we have $T(n) = \widetilde{O}(T_{sep}(n))$.

In total, makeElimOrderAndTree recurses to a depth of $O(\log n)$, and at each recursive iteration, the contribution to the elimination tree height is the size of the separator $|S| \leqslant f(\tau)$ computed in the iteration. □

A standard implementation of approxBalancedSeparator given a tree decomposition of G is as follows:

Theorem 4.17. *Let (X, T) be a width-τ tree decomposition of a graph G on n vertices. Then in $O(n\tau)$ time, we can find a 2/3-vertex separator (G_1, S, G_2) of G, and tree decompositions (X_1, T_1) of G_1 and (X_2, T_2) of G_2 each of width at most τ.*

Proof. We assume T has $O(n)$ nodes to start (a transformation can be made in $O(\tau \cdot |V(T)|)$ time in the recursive iterations, see e.g. [FLS+18] Definition 2.4). By scanning through the bags of T in $O(n\tau)$ time, we can find a node $t \in T$ such that $T \setminus t$ is two disjoint subtrees T_1, T_2, with $|\bigcup_{s \in T_1} X(s) \setminus X(t)| \leqslant 2/3n$, and similarly for T_2. Then $X(t) \subseteq V(G)$ is a 2/3-vertex separator of G. By removing the vertices $X(t)$ from all the bags in T_1 and T_2, we get the tree decompositions of G_1 and G_2 respectively, both of width at most τ. □

When the tree decomposition is not given, we can use [BGS21] to find it approximately.

Theorem 4.18 ([BGS21])**.** *Given a graph with m edges and n vertices, we can compute a width-$O(tw(G) \log^3 n)$ tree decomposition in $O(m^{1+o(1)} \operatorname{polylog} n)$ time.*

Proof of Theorem 4.1. It remains to show that the tree $\mathcal{T}$ returned by Algorithm 2 satisfies the sparsity properties specified in Theorem 4.1.

Let $\mathcal{T}'$ be the true elimination tree corresponding to the elimination order π computed by Algorithm 2. Note that in a recursive iteration makeElimOrderAndTree(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 in $\mathcal{T}'$, and vice versa. Any path in $\mathcal{T}'$ from a vertex $i \in H_1$ to the

root goes through some higher-ordered vertices in H_1 followed by a subset of the vertices S; this is contained in the path in $\mathcal{T}$ from i to the root, which includes all higher-ordered vertices in H_1 and all of S. By Lemma 4.12, after the permuting according to π, for each $j \in [d]$, the non-zero pattern of L_j is a subset of the path from j to the root in $\mathcal{T}'$; it is therefore also true in $\mathcal{T}$.

Plugging in $f(\tau) = \tau$ when a width-τ decomposition is given, and $f(\tau) = O(tw(G)\log^3 n)$ when not, and using the monotonicity property of treewidth and separator size, we get the conclusions of Theorem 4.1 immediately. □

5 Sparsity Patterns and Maintaining the Cholesky Factorization

In this section, we discuss the sparsity properties of all the matrices we work with for the central path, and the required runtime for their computations and maintenance. All of these properties are known (see textbooks [GLN94, Dav06] for more complete introductions). We include some algorithms and proofs to familiarize readers for the techniques we will use. As in the previous sections, we have the constraint matrix $A \in \mathbb{R}^{d\times n}$ whose rows are permuted according to Theorem 4.1. Let L be the Cholesky factor of $AH^{-1}A^\top$, and let $\mathcal{T} = (\{1,\ldots,d\}, E)$ be the elimination tree for L of height τ. Note we use the convention that a tree consisting of a single vertex has height 1.

Recall $\mathcal{P}(i)$ denote the set of vertices on the path from vertex i to the root in $\mathcal{T}$, and $\mathcal{D}(i)$ denotes the set of vertices in the subtree rooted at i (including i) in $\mathcal{T}$. For any matrix M, we use M_i to denote the i-th column or block, and $\mathcal{M}_i$ to denote the non-zero pattern of the i-th column or block (i.e. it is a set of row indices). For example, $j \in \mathcal{A}_i$ if row j of A is non-zero in a column in block i. We use M^i to denote the i-th row of M and $\mathcal{M}^i$ to denote the non-zero pattern of the i-th row.

We begin with basic properties of A and L:

Lemma 5.1. *If $tw(A) = \tau$, then $nnz(A_i) \leqslant \tau$ for all $i \in [n]$. In particular, $\mathcal{A}_i$ is a subset of some path from a leaf to the root of $\mathcal{T}$.*

Proof. By construction, $\mathcal{A}_i$ form a clique in the dual graph G_A, and $tw(A)$ is lower-bounded by the size of the largest clique in G_A. By construction of the elimination tree, any clique in G_A must lie on one path from a leaf to the root of $\mathcal{T}$. □

Lemma 5.2 ([Sch82, proposition 5])**.** *$\mathcal{L}_i \subseteq \mathcal{P}(i)$ for each i. In particular, the height of the elimination tree satisfies $\tau \geq \max\{|\mathcal{L}_i| : i \in [d]\}$.* □

As a corollary, this relation between the non-zero pattern of the columns of L and $\mathcal{T}$ further allow us to characterize the non-zero pattern of the rows of L:

Lemma 5.3. $\mathcal{L}^i \subseteq \mathcal{D}(i)$. □

5.1 Solving Triangular Systems

Now, we discuss the cost of solving triangular systems, in connection to the elimination tree $\mathcal{T}$.

Algorithm 3 Solving $Lx = v$

$x \leftarrow \mathbf{0}_d$
for increasing j with $v_j \neq 0$ **do**
    $x_j \leftarrow v_j / L_{jj}$
    $v \leftarrow v - x_j L_j$
end for
return x

Lemma 5.4. *Let $x = L^{-1}v$, and let $\mathcal{S}$ be the non-zero pattern of v. Then, the nonzero pattern of x is a subset of $\bigcup_{i \in \mathcal{S}} \mathcal{P}(i)$. Furthermore, we can solve for $L^{-1}v$ in $O(\|L^{-1}v\|_0 \cdot \tau)$ time. In particular, if the non-zero pattern of v is a subset of some path $\mathcal{P}$ from a leaf to the root in $\mathcal{T}$, then the non-zero pattern of $L^{-1}v$ is also a subset of $\mathcal{P}$, and we can solve for $L^{-1}v$ in $O(\tau^2)$ time.*

Proof. We prove the sparsity pattern by inspecting Algorithm 3. Note that the $x_i \neq 0$ if $v_i \neq 0$ or ($x_j \neq 0$ and $L_{ij} \neq 0$). Lemma 5.2 shows that $L_{ij} \neq 0$ implies i is an ancestor of j. Hence, the non-zeros in x can only propagate to its ancestors from the non-zeros of v. As a result, the non-zero pattern of x is a subset of $\bigcup_{i \in \mathcal{S}} \mathcal{P}(i)$.

For the runtime, we note that L_j has τ non-zero entries and hence each step takes $O(\tau)$ time. Since the number of steps is exactly $\|L^{-1}v\|_0$, we have the runtime $O(\|L^{-1}v\|_0 \cdot \tau)$. □

Lemma 5.5. *For any v, we can solve for $(L^{-\top}v)_i$ in time $O(\tau^2)$.*

Proof. Note that $(L^{-\top}v)_i = e_i^\top L^{-\top} v$. By Lemma 5.4, computing $e_i^\top L^{-\top}$ takes $O(\tau^2)$ time and the resulting vector has τ sparsity, so the subsequent multiplication with v also takes $\tau = O(\tau^2)$ time. □

Lemma 5.6. *Let $S \subseteq [d]$ be a subset of the vertices on some path $\mathcal{P}$ from a leaf to the root in $\mathcal{T}$. Then for any y, we can compute the subvector $(L^{-\top}y)|_S = y^\top L^{-1}|_S$ in $O(\tau^2)$ time, where $L^{-1}|_S$ denotes L^{-1} restricted to the columns given by S.*

Proof. Let $S' = V(\mathcal{P})$, so we have $S \subseteq S'$. Lemma 5.4 shows that $L^{-1}e_i$ is supported on S' for any $i \in S'$. It follows that for any $i \in S$, we have $e_i^\top L^{-\top} y = y^\top L^{-1} e_i = y|_{S'}^\top (L^{-1}e_i)|_{S'}$. Hence, $(L^{-\top}y)|_S$ only depends on the entries of y on S'.

This allows us to write $(L^{-\top}y)|_S = (L^{-\top})|_{S' \times S'} y|_{S'} = (L_{S' \times S'})^{-\top} y|_{S'}$. Finally, we note that $L_{S' \times S'}^\top$ is a $(\tau+1) \times (\tau+1)$ upper triangular matrix and hence we can solve it in $O(\tau^2)$ time.

□

5.2 Computing and Updating the Cholesky Factorization

Next, we study the cost of computing and updating the Cholesky factorization. The crux for efficient implementation of sparse Cholesky factorization is that both the matrix M and its Cholesky decomposition $M = LL^\top$ are sparse, and hence the operations involving 0 can be skipped. There are many different algorithms for this; the following is one of them.

Algorithm 4 Cholesky factorization of a matrix M

for $j = 1$ to d **do**
  $L_{j,j} \leftarrow \sqrt{M_{j,j} - \sum_{k=1}^{j-1} L_{j,k}^2}$
  for $i = j + 1$ to d **do**
    $L_{i,j} \leftarrow \frac{1}{L_{j,j}} \left(M_{i,j} - \sum_{k=1}^{j-1} L_{i,k} L_{j,k} \right)$
  end for
end for
return L

By analyzing the number of non-zeros operations of the above algorithm (or other Cholesky factorization algorithms), one can show the following:

Lemma 5.7 ([GLN94, Theorem 2.2.2])**.** *For a positive definite matrix M, we can compute its Cholesky factorization $M = LL^\top$ in time*

$$\Theta(\sum_{j=1}^{d} |\mathcal{L}_j|^2),$$

where $|\mathcal{L}_j|$ denotes the number of nonzero entries in the j-th column of L. □

Corollary 5.8. *The Cholesky factorization $AH_{\overline{x}}^{-1}A^\top = LL^\top$ can be computed in $O(n\tau^2)$ time.*

Proof. By the definition of tree height, for any vertex i in the tree, the length of the path from i to root is less than τ. Then Lemma 4.12 implies $|\mathcal{L}_j| \leqslant \tau$ for all j. Hence, it takes $O(n\tau^2)$ time to compute $AH_{\overline{x}}^{-1}A^\top$ explicitly. Then, we can apply Lemma 5.7 to compute Cholesky factorization and it takes time $O(d\tau^2) = O(n\tau^2)$. □

The following two lemmas involve rank-1 updates of the Cholesky factorization, one regarding the sparsity pattern and one the update time. We state a simplified version of [DH03], which makes a further sparsity assumption. We include the proof of first lemma for intuition.

Lemma 5.9 ([DH03, Section 5])**.** *Given a positive definite matrix $M \in \mathbb{R}^{d \times d}$, its elimination tree $\mathcal{T}$ of height τ, and the corresponding Cholesky factorization $M = LL^\top$. Let $(L+\Delta L)(L+\Delta L)^\top$ be the new Cholesky factorization of $M + ww^\top$. Suppose that the sparsity pattern of M and $M + ww^\top$ are same. If we let $\mathcal{S}$ be the index set of columns of L that are updated, i.e. $\mathcal{S} = \{j \in [d] \mid \Delta L_j \neq \mathbf{0}\}$, then $\mathcal{S}$ is a subset of some path from k to the root in $\mathcal{T}$ where k is the first non-zero index in w. Consequently, the row and column sparsity of ΔL are bounded by τ, and $\mathrm{nnz}(\Delta L) = O(\tau^2)$.*

The same holds for $M - ww^\top$ as long as $M - ww^\top$ is positive definite.

Proof. Since $ww^\top$ is a clique in the graph associated with non-zeros of M, it is contained in a path from k to the root in $\mathcal{T}$ where k is the first non-zeros in w. Let $\mathcal{I}$ be the set of indices in this path. It follows that M is only changed in the $\mathcal{I} \times \mathcal{I}$ block. Now, we run Algorithm 4 twice, once on M and once on $M + ww^\top$ and prove that the difference in L is in the $\mathcal{I} \times \mathcal{I}$ block. The formulas in Algorithm 4 show that the changes to M and L are propagated in the L in the next step in the following ways:

- Updating the entry M_{ij} causes L_{ij} to update.
  This case is good because we know $i, j \in \mathcal{I}$.

- Updating the entry L_{jk} causes L_{jj} to update.
  By induction, in the last step L_{jk} is updated implies $j, k \in \mathcal{I}$. Hence, $(j, j) \in \mathcal{I} \times \mathcal{I}$ (only entries in the $\mathcal{I} \times \mathcal{I}$ submatrix of L are updated).
- Updating the entry L_{ik} and $L_{jk} \neq 0$ causes L_{ij} to update.
  By induction, we know $i, k \in \mathcal{I}$. Since $L_{jk} \neq 0$, Lemma 5.2 shows j is on the path of k to the root. Since $k \in \mathcal{I}$, we have $j \in \mathcal{I}$. Hence, $(i, j) \in \mathcal{I} \times \mathcal{I}$ again.
- Updating the entry L_{jk} and $L_{ik} \neq 0$ causes L_{ij} to update.
  Same argument as above.

In all the cases, the change in L is restricted to the $\mathcal{I} \times \mathcal{I}$ submatrix. □

At a high level, since we know L is changed in a $\tau \times \tau$ sized block, we only need to update the factorization on that block. Similarly to matrix inverse, there are simple algorithms for rank-1 update for factorization in time linear to the square of the dimension.

Lemma 5.10 ([DH03, Section 5]). *Given a positive definite matrix $M \in \mathbb{R}^{d \times d}$, its elimination tree $\mathcal{T}$ of height τ, and the corresponding Cholesky factorization $M = LL^\top$. Let $(L + \Delta L)(L + \Delta L)^\top$ be the new Cholesky factorization of $M + ww^\top$. Suppose that the sparsity pattern of M and $M + vv^\top$ are same. Then, we can compute ΔL in $O(\tau^2)$ time.*

The same holds for $M - ww^\top$ as long as $M - ww^\top$ is positive definite. □

6 Robust Central Path Maintenance

In this section, we present a data structure CentralPathMaintenance to efficiently perform the robust central path step needed in Algorithm 16. Specifically, we will prove the following theorem.

Theorem 6.1 (Robust Central Path Step). *Suppose Algorithm 16 is run on the convex program Eq. (CP). Given the constraint matrix $A \in \mathbb{R}^{d \times n}$ with block-diagonal structure $n = \sum_{i=1}^{m} n_i$, its binary elimination tree $\mathcal{T}$ of height τ, and parameters $\lambda, \bar{\varepsilon}, \varepsilon_t, \alpha, w = \mathbf{1}_m$ as defined in Algorithm 16, the randomized data structure CentralPathMaintenance (Algorithms 14 and 15) implicitly maintains the central path primal-dual solution pair (x, s) (Algorithm 16 Line 31) and explicitly maintains its approximation $(\overline{x}, \overline{s})$ (Algorithm 16 Line 28) using the following functions:*

- *Initialize(x, s, t_0, k): Initializes the data structure with initial primal-dual solution pair (x, s), initial central path timestep t_0, and a runtime tuning parameter k in $O(n\tau^2 \log^4(n))$ time.*
- *MultiplyAndMove(t): It implicitly maintains*

$$\begin{aligned} x &\leftarrow x + H_{\overline{x}}^{-1/2}(I - P_{\overline{x}})H_{\overline{x}}^{-1/2}\delta_\mu(\overline{x}, \overline{s}, \overline{t}) \\ s &\leftarrow s + tH_{\overline{x}}^{1/2}P_{\overline{x}}H_{\overline{x}}^{-1/2}\delta_\mu(\overline{x}, \overline{s}, \overline{t}) \end{aligned} \tag{6.1}$$

  where $H_{\overline{x}} \stackrel{\text{def}}{=} \nabla^2\phi(\overline{x})$, $P_{\overline{x}} \stackrel{\text{def}}{=} H_{\overline{x}}^{-1/2}A^\top(AH_{\overline{x}}^{-1}A^\top)^{-1}AH_{\overline{x}}^{-1/2}$, and $\bar{t}$ is some earlier timestep satisfying $|t - \bar{t}| \leqslant \varepsilon_t \cdot \bar{t}$.

  It also explicitly maintains $(\overline{x}, \overline{s})$ such that $\|\overline{x}_i - x_i\|_{\overline{x}_i} \leqslant \bar{\varepsilon}$ and $\|\overline{s}_i - s_i\|_{\overline{x}_i}^* \leqslant t\bar{\varepsilon}w_i$ for all $i \in [m]$ with probability at least 0.9.

Assuming the function is called at most N times and t is monotonically decreasing from $t_{\max}$ to $t_{\min}$, the total running time is

$$O\left(\left(Nn^{1/2} + n\log(t_{\max}/t_{\min})\right)\tau^2 \operatorname{poly}\log(N)\right).$$

- *Output: It computes (x, s) exactly and outputs them in $O(n\tau^2)$ time.*

Remark 6.2. The N dependence in the runtime is a result of parameter tuning. If the IPM takes more than $\widetilde{O}(\sqrt{n}\log(1/\varepsilon))$ steps, the data structure can still run in $\widetilde{O}(n\tau^2\log(1/\varepsilon))$ by choosing a larger value for the parameter k in Initialize.

6.1 Multiscale Representation of the Central Path Dynamic

In any call to MultiplyAndMove, we want to update the central path primal-dual solution pair according to Eq. (6.1), as well as the approximation pair. Here, we introduce the multiscale representation used in these computations:

Definition 6.3 (Multiscale Basis)**.** At any step of the robust central path with approximate primal-dual solution pair $(\overline{x}, \overline{s})$, we define

$$\mathcal{W} \stackrel{\text{def}}{=} L_{\overline{x}}^{-1} A H_{\overline{x}}^{-1/2}$$

where $H_{\overline{x}} = \nabla^2\phi(\overline{x})$ and $L_{\overline{x}}$ is the lower Cholesky factor of $AH_{\overline{x}}^{-1}A^\top$.

Intuitively, the basis element are rows of $\mathcal{W}$, which are represented by vertices in the elimination tree $\mathcal{T}$. Note that our data structure never computes or stores $\mathcal{W}$ explicitly, as it is a costly operation.

Definition 6.4 (Multiscale Coefficients)**.** At any step of the robust central path with approximate primal-dual solution pair $(\overline{x}, \overline{s})$, we define

$$h \stackrel{\text{def}}{=} L_{\overline{x}}^{-1} A H_{\overline{x}}^{-1} \delta_\mu(\overline{x}, \overline{s}, \overline{t})$$

where $H_{\overline{x}} = \nabla^2\phi(\overline{x})$, and $L_{\overline{x}}$ is the lower Cholesky factor of $AH_{\overline{x}}^{-1}A^\top$.

Now, we can rewrite the central path update from Eq. (6.1) using the multiscale representation:

$$\begin{aligned} x &\leftarrow x + H_{\overline{x}}^{-1}\delta_\mu(\overline{x}, \overline{s}, \overline{t}) - H_{\overline{x}}^{-1/2}\mathcal{W}^\top h \\ s &\leftarrow s + tH_{\overline{x}}^{1/2}\mathcal{W}^\top h. \end{aligned} \tag{6.2}$$

6.2 Implicit Representation of (x, s)

For the first part of proof of Theorem 6.1, we demonstrate how to obtain an implicit representation of the robust central path pair (x, s), using the explicitly maintained approximation pair $(\overline{x}, \overline{s})$. Rather than directly working with the expression in Eq. (6.2), we rewrite (x, s) in terms of variables that admit sparse changes between consecutive steps in the central path, in order to more efficiently maintain them.

Theorem 6.5. *Given constraint matrix A and its binary elimination tree $\mathcal{T}$ with height τ, the data structure* MultiscaleRepresentation *(Algorithms 5 and 6) implicitly maintains the primal-dual pair (x, s) as defined by Eq. (6.2), computable via the expression*

$$\begin{aligned} x &= \widehat{x} + H_{\overline{x}}^{-1/2}\beta_x c_x - H_{\overline{x}}^{-1/2}\mathcal{W}^\top(\beta_x h + \varepsilon_x) \\ s &= \widehat{s} + H_{\overline{x}}^{1/2}\mathcal{W}^\top(\beta_s h + \varepsilon_s), \end{aligned} \tag{6.3}$$

by maintaining the variables $\widehat{x}, \beta_x, c_x, \varepsilon_x, \widehat{s}, \beta_s, \varepsilon_s, h, H_{\overline{x}}$ and $L_{\overline{x}}$. Note that the variables ε_x and ε_s here denote the accumulated error of $\beta_x h$ and $\beta_s h$; they are not necessarily small.

The data structure supports the following functions:

1. INITIALIZE$(x, s, \overline{x}, \overline{s}, \overline{t})$ *: Initializes the data structure in $O(n\tau^2)$ time, with initial value of the primal-dual pair (x, s), its initial approximation $(\overline{x}, \overline{s})$, and initial approximate timestep $\overline{t}$.*
2. MOVE()*: Moves (x, s) according to Eq. (6.2) in $O(1)$ time by updating its implicit representation.*
3. UPDATE$(\overline{x}^{\text{new}}, \overline{s}^{\text{new}})$*: Updates the approximation pair $(\overline{x}, \overline{s})$ to $(\overline{x}^{\text{new}}, \overline{s}^{\text{new}})$.*

   Let $\mathcal{S} = \{i \in [m] \mid \overline{x}_i^{\text{new}} \neq \overline{x}_i \text{ or } \overline{s}_i^{\text{new}} \neq \overline{s}_i\}$. Then each call to UPDATE takes $O(|\mathcal{S}| \cdot \tau^2)$ time, and each variable in Eq. (6.3) except $\mathcal{W}$ changes in $O(|\mathcal{S}| \cdot \tau)$ many entries.

Algorithm 5 Multiscale Representation Data Structure - Initialize and Move

datastructure MULTISCALEREPRESENTATION
private : member
  Constraint matrix $A \in \mathbb{R}^{d \times n}$, elimination tree $\mathcal{T}$ ▷ Fixed global constants
  $\overline{x}, \overline{s} \in \mathbb{R}^n$ ▷ Approximate primal dual pair of (x, s)
  $H_{\overline{x}} \in \oplus_{i \in [m]} \mathbb{R}^{n_i \times n_i}$ ▷ Hessian matrix $H_{\overline{x}} = \nabla^2 \phi(\overline{x})$
  $L_{\overline{x}} \in \mathbb{R}^{d \times d}$ ▷ Lower Cholesky factor of $AH_{\overline{x}}A^\top$
  $\widehat{x}, \widehat{s}, c_x \in \mathbb{R}^n,\ \varepsilon_x, \varepsilon_s, h \in \mathbb{R}^d,\ \beta_x, \beta_s \in \mathbb{R}$ ▷ Implicit representation of (x, s) as in Eq. (6.3)
  $\overline{\alpha} \in \mathbb{R}, \overline{\delta_\mu} \in \mathbb{R}^n$ ▷ Implicit representation of δ_μ as in Invariant 6.6
  $\overline{t} \in \mathbb{R}_+$ ▷ Central path timestep parameter
end members
procedure INITIALIZE$(x \in \mathbb{R}^n, s \in \mathbb{R}^n, \overline{x} \in \mathbb{R}^n, \overline{s} \in \mathbb{R}^n, \overline{t} \in \mathbb{R}_+)$
  $\overline{x} \leftarrow \overline{x}, \overline{s} \leftarrow \overline{s}, \overline{t} \leftarrow \overline{t}$
  $\widehat{x} \leftarrow x, \widehat{s} \leftarrow s$
  $\varepsilon_x \leftarrow \mathbf{0}, \varepsilon_s \leftarrow \mathbf{0}$
  $\beta_x \leftarrow 0, \beta_s \leftarrow 0$
  $H_{\overline{x}} \leftarrow \nabla^2 \phi(\overline{x})$
  Find lower Cholesky factor $L_{\overline{x}}$ where $L_{\overline{x}} L_{\overline{x}}^\top = AH_{\overline{x}}^{-1}A^\top$ using $\mathcal{T}$ ▷ By Corollary 5.8
  INITIALIZE$h(\overline{x}, \overline{s}, H_{\overline{x}}, L_{\overline{x}})$
end procedure
procedure INITIALIZE$h(\overline{x}, \overline{s}, H_{\overline{x}}, L_{\overline{x}})$ ▷ Lemma 6.8
  for $i \in [m]$ **do**
    $(\overline{\delta_\mu})_i \leftarrow -\frac{\alpha \sinh(\frac{\lambda}{w_i}\gamma_i(\overline{x}, \overline{s}, \overline{t}))}{\gamma_i(\overline{x}, \overline{s}, \overline{t})} \cdot \mu_i(\overline{x}, \overline{s}, \overline{t})$
    $\overline{\alpha} \leftarrow \overline{\alpha} + \alpha^2 \cdot w_i^{-1} \cosh^2(\frac{\lambda}{w_i}\gamma_i(\overline{x}, \overline{s}, \overline{t}))$ ▷ λ, w, γ, μ as defined in Algorithm 16
  end for
  $c_x \leftarrow H_{\overline{x}}^{-1/2} \overline{\delta_\mu}$
  $h \leftarrow L_{\overline{x}}^{-1} A H_{\overline{x}}^{-1} \overline{\delta_\mu}$
end procedure
procedure MOVE
  $\beta_x \leftarrow \beta_x + (\overline{\alpha})^{-1/2}$
  $\beta_s \leftarrow \beta_s + \overline{t} \cdot (\overline{\alpha})^{-1/2}$
end procedure

Algorithm 6 Multiscale Representation Data Structure - Update

datastructure MULTISCALEREPRESENTATION
procedure UPDATE($\overline{x}^{\text{new}}$,$\overline{s}^{\text{new}}$) ▷ Lemma 6.10
  $H^{\text{new}} \leftarrow \nabla^2 \phi(\overline{x}^{\text{new}})$
  Find lower Cholesky factor L^{new} where $L^{\text{new}}(L^{\text{new}})^\top = AH^{\text{new}}A^\top$ ▷ Lemma 5.10
  UPDATE$h(\overline{x}^{\text{new}}, \overline{s}^{\text{new}}, H^{\text{new}}, L^{\text{new}})$
  UPDATE$\mathcal{W}(L^{\text{new}}, H^{\text{new}})$
  $\overline{x} \leftarrow \overline{x}^{\text{new}},\ \overline{s} \leftarrow \overline{s}^{\text{new}}$
  $H_{\overline{x}} \leftarrow H^{\text{new}},\ L_{\overline{x}} \leftarrow L^{\text{new}}$
end procedure
procedure UPDATE$h(\overline{x}^{\text{new}}, \overline{s}^{\text{new}}, H^{\text{new}}, L^{\text{new}})$ ▷ Lemma 6.11
  $S \leftarrow \{i \in [m] \mid \overline{x}_i^{\text{new}} \neq \overline{x}_i \text{ or } \overline{s}_i^{\text{new}} \neq \overline{s}_i\}$
  $\overline{\alpha}^{\text{new}} \leftarrow \overline{\alpha},\ \overline{\delta_\mu}^{\text{new}} \leftarrow \overline{\delta_\mu}$
  for $i \in S$ **do**
    ▷ λ, w, γ, μ as defined in Algorithm 16
    $\overline{\alpha}^{\text{new}} \leftarrow \overline{\alpha}^{\text{new}} - \alpha^2 \cdot w_i^{-1} \cosh^2(\frac{\lambda}{w_i}\gamma_i(\overline{x}, \overline{s}, \overline{t})) + \alpha^2 \cdot w_i^{-1} \cosh^2(\frac{\lambda}{w_i}\gamma_i(\overline{s}^{\text{new}}, \overline{x}^{\text{new}}, \overline{t}))$
    $(\overline{\delta_\mu}^{\text{new}})_i \leftarrow -\frac{\alpha \sinh(\frac{\lambda}{w_i}\gamma_i(\overline{x}^{\text{new}}, \overline{s}^{\text{new}}, \overline{t}))}{\gamma_i(\overline{x}^{\text{new}}, \overline{s}^{\text{new}}, \overline{t})} \cdot \mu_i(\overline{x}^{\text{new}}, \overline{s}^{\text{new}}, \overline{t})$
  end for
  $c_x^{\text{new}} \leftarrow (H^{\text{new}})^{-1/2}\overline{\delta_\mu}^{\text{new}}$
  $h^{\text{new}} \leftarrow (L^{\text{new}})^{-1}A(H^{\text{new}})^{-1}\overline{\delta_\mu}^{\text{new}}$
  $\varepsilon_x^{\text{new}} \leftarrow \varepsilon_x + \beta_x(h^{\text{new}} - h)$
  $\varepsilon_s^{\text{new}} \leftarrow \varepsilon_s + \beta_s(h^{\text{new}} - h)$
  $\widehat{x}^{\text{new}} \leftarrow \widehat{x} + \beta_x(H_{\overline{x}}^{-1/2}c_x - (H^{\text{new}})^{-1/2}c_x^{\text{new}}) - (H_{\overline{x}}^{-1/2} - (H^{\text{new}})^{-1/2})\mathcal{W}^\top(\beta_x h + \varepsilon_x)$
  $\widehat{s}^{\text{new}} \leftarrow \widehat{s} + (H_{\overline{x}}^{1/2} - (H^{\text{new}})^{1/2})\mathcal{W}^\top(\beta_s h + \varepsilon_s)$
  $c_x \leftarrow c_x^{\text{new}}, h \leftarrow h^{\text{new}}$
  $\varepsilon_x \leftarrow \varepsilon_x^{\text{new}}, \varepsilon_s \leftarrow \varepsilon_s^{\text{new}}$
  $\widehat{x} \leftarrow \widehat{x}^{\text{new}}, \widehat{s} \leftarrow \widehat{s}^{\text{new}}$
end procedure
procedure UPDATE$\mathcal{W}(L^{\text{new}}, H^{\text{new}})$ ▷ Lemma 6.12
  $\widehat{x}^{\text{new}} \leftarrow \widehat{x} - (H^{\text{new}})^{-1/2}((H^{\text{new}})^{-1/2} - H_{\overline{x}}^{-1/2})A^\top L^{-\top}(\beta_x h + \varepsilon_x)$
  $\widehat{s}^{\text{new}} \leftarrow \widehat{s} - (H^{\text{new}})^{1/2}((H^{\text{new}})^{-1/2} - H_{\overline{x}}^{-1/2})A^\top L^{-\top}(\beta_s h + \varepsilon_s)$
  $\varepsilon_x^{\text{new}} \leftarrow \varepsilon_x + (L^{\text{new}} - L)^\top L^{-\top}(\beta_x h + \varepsilon_x)$
  $\varepsilon_s^{\text{new}} \leftarrow \varepsilon_s + (L^{\text{new}} - L)^\top L^{-\top}(\beta_s h + \varepsilon_s)$
  $\widehat{x} \leftarrow \widehat{x}^{\text{new}}, \widehat{s} \leftarrow \widehat{s}^{\text{new}}$
  $\varepsilon_x \leftarrow \varepsilon_x^{\text{new}}, \varepsilon_s \leftarrow \varepsilon_s^{\text{new}}$
end procedure

Proof of Theorem 6.5

We prove the correctness and running time for each operation of MULTISCALEREPRESENTATION, and that they respect Invariant 6.6. The correctness of the implicit representation in Eq. (6.3) then follows immediately.

Invariant 6.6. *After the data structure MULTISCALEREPRESENTATION is initialized, the correct central path pair (x, s) is always implicitly maintained and can be computed according to Eq. (6.3). Moreover, the following additional invariants are maintained:*

$$\overline{\alpha} = \sum_{j=1}^{m} w_j^{-1} \cosh^2\left(\frac{\lambda}{w_j}\gamma_i(\overline{x}, \overline{s}, \overline{t})\right) \tag{i}$$

$$\overline{\delta_\mu} = \overline{\alpha}^{1/2} \cdot \delta_\mu(\overline{x}, \overline{s}, \overline{t}) \tag{ii}$$

$$c_x = H_{\overline{x}}^{-1/2}\overline{\delta_\mu} \tag{iii}$$

$$h = L_{\overline{x}}^{-1} A H_{\overline{x}}^{-1}\overline{\delta_\mu}. \tag{iv}$$

Lemma 6.7 (INITIALIZE)**.** *The data structure MULTISCALEREPRESENTATION takes $O(n\tau^2)$ to initialize. Moreover, Invariant 6.6 is satisfied after initialization.*

Proof. **Proof of Correctness:** We initialize $\widehat{x}$ to x and $\widehat{s}$ to s and all other terms from Eq. (6.3) to zero. Hence, this is the correct representation. Next, we call the helper function INITIALIZEh, and the remainder of Invariant 6.6 is guaranteed by Lemma 6.8.

Proof of Runtime: Since $n_i = O(1)$ for all $i \in [m]$, we can compute $\nabla^2\phi(\overline{x})$ in $O(n)$ time. By Corollary 5.8, we can find the lower Cholesky factor in $O(n\tau^2)$ time. By Lemma 6.8, INITIALIZEh takes $O(n\tau^2)$ time. Hence, the initialization takes $O(n\tau^2)$ time. □

Lemma 6.8 (INITIALIZE$h(\overline{x}, \overline{s}, H_{\overline{x}}, L_{\overline{x}})$)**.** *Given approximate central path pair $(\overline{x}, \overline{s})$, the Hessian matrix $H_{\overline{x}} = \nabla^2\phi(\overline{x})$, and lower Cholesky factor $L_{\overline{x}}$, the data structure takes $O(n\tau^2)$ time to perform INITIALIZEh. Moreover, (i)-(iv) of Invariant 6.6 are satisfied after initialization.*

Proof. **Proof of Correctness:** The invariants directly follow from the definition.

Proof of Runtime: Since $n_i = O(1)$ for all $i \in [m]$, each iteration of the for-loop takes $O(1)$ time. Then, it takes $O(n)$ time to compute $\overline{\alpha}$ and $\overline{\delta_\mu}$. Since $H_{\overline{x}}$ is a block-diagonal matrix, we can compute $c_x = H_{\overline{x}}^{-1/2}\overline{\delta_\mu}$ in $O(n)$ time. Finally, $L_{\overline{x}}^{-1} A H_{\overline{x}}^{-1}\overline{\delta_\mu}$ can be computed in time $O(n\tau^2)$ by Lemmas 5.1 and 5.4. □

Lemma 6.9 (MOVE)**.** *Under Invariant 6.6, the data structure MULTISCALEREPRESENTATION takes $O(1)$ time to move the current central path pair (x, s) by one step according to Eq. (6.2). Moreover, Invariant 6.6 is preserved afterwards.*

Proof. **Proof of Correctness:** Let $x^{\text{new}}, s^{\text{new}}$ be the updated values of x, s after MOVE is performed. We check that the implicit representation from Eq. (6.3) is indeed the correct expression

for x^{new} by comparing it to x:

$$\begin{aligned} x^{\text{new}} - x &= H_{\overline{x}}^{-1/2}\overline{\alpha}^{-1/2}c_x - H_{\overline{x}}^{-1/2}\mathcal{W}^\top(\overline{\alpha}^{-1/2}h) \\ &= H_{\overline{x}}^{-1/2}\overline{\alpha}^{-1/2}H_{\overline{x}}^{-1/2}\overline{\delta_\mu} - H^{-1/2}\mathcal{W}^\top(\overline{\alpha}^{-1/2}L_{\overline{x}}^{-1}AH_{\overline{x}}^{-1}\overline{\delta_\mu}) \\ &= H_{\overline{x}}^{-1}\delta_\mu(\overline{x},\overline{s},\overline{t}) - H_{\overline{x}}^{-1/2}\mathcal{W}^\top L_{\overline{x}}^{-1}AH^{-1}\delta_\mu(\overline{x},\overline{s},\overline{t}) \\ &= H_{\overline{x}}^{-1}\delta_\mu(\overline{x},\overline{s},\overline{t}) - H_{\overline{x}}^{-1/2}\mathcal{W}^\top h, \end{aligned}$$

where the first step follows by Line 29, the second by (3) and (4) of Invariant 6.6, the third by (2) of Invariant 6.6, and the fourth step follows by the definition of h. This difference is exactly as given in Eq. (6.2).

Similarly, for s^{new}, we have

$$\begin{aligned} s^{\text{new}} - s &= H_{\overline{x}}^{1/2}\mathcal{W}^\top(\overline{t}\cdot\beta_s) \\ &= \overline{t}H_{\overline{x}}^{1/2}\mathcal{W}^\top(\overline{\alpha}^{-1/2}L_{\overline{x}}AH_{\overline{x}}^{-1}\overline{\delta_\mu}) \\ &= \overline{t}H_{\overline{x}}^{1/2}\mathcal{W}^\top L_{\overline{x}}AH_{\overline{x}}^{-1}\delta_\mu(\overline{x},\overline{s},\overline{t}) \\ &= \overline{t}H_{\overline{x}}^{1/2}\mathcal{W}^\top h, \end{aligned}$$

exactly as given in Eq. (6.2). The first step follows from Line 30, the second and third steps from (4) and (2) of Invariant 6.6, and the fourth step from the definition of h.

Proof of Runtime: The operation only uses addition and taking square roots of real numbers. □

Lemma 6.10 (UPDATE$(\overline{x}^{\text{new}},\overline{s}^{\text{new}})$)**.** *Under Invariant 6.6, the data structure* MULTISCALEREPRESENTATION *takes $O(|S|\cdot\tau^2)$ time to move the approximation pair $(\overline{x},\overline{s})$ to $(\overline{x}^{\text{new}},\overline{s}^{\text{new}})$, where $S=\{i\in[m]\mid \overline{x}_i^{\text{new}}\neq\overline{x}_i \text{ or } \overline{s}_i^{\text{new}}\neq\overline{s}_i\}$. Invariant 6.6 is preserved at the end of the function call.*

Moreover, the total number of coordinate changes in the variables involved in the implicit representation is bounded by $O(|S|\cdot\tau)$.

Proof. We can update $\overline{x},\overline{s}$ trivially. Immediately afterwards, we must update $H_{\overline{x}}$, L and h in the data structure, so they correspond correctly to $\overline{x}^{\text{new}}$. As a result of these updates, Eq. (6.3) will no longer hold, so we must then adjust the other variables $\widehat{x},\widehat{s},c_x,\varepsilon_x,\varepsilon_s,\beta_x,\beta_s$ used in the implicit representation, in order to restore the invariant. To simplify the presentation, we accomplish this via two helper functions, UPDATEh and UPDATE$\mathcal{W}$.

By combining Lemmas 6.11 and 6.12, we show that the implicit representation expression holds after all variables are updated. Furthermore, they show the required bound on the total number of coordinate changes in all the implicit representation variables.

For the runtime, we can compute L^{new} in $O(\tau^2\cdot(\|\overline{x}^{\text{new}}-\overline{x}\|_0+\|\overline{s}^{\text{new}}-\overline{s}\|_0))$ time by Lemma 5.10. Furthermore, UPDATEh takes $O(|S|\cdot\tau^2)$ time. For UPDATE$\mathcal{W}$, we can split the update of L into $|S|$ many rank-1 updates by updating $A((H_i^{\text{new}})^{-1}-H_i^{-1})A^\top$ in time $O(\tau^2)$ for each $i\in S$, where $H_i=\nabla^2\phi_i(x_i)$. By Lemma 5.9, the non-zero columns of $\Delta L\overset{\text{def}}{=}L^{\text{new}}-L$ lie on a path of $\mathcal{T}$. Then, each call of UPDATE$\mathcal{W}$ takes $O(\tau^2)$ time by Lemma 6.12. Hence, the total runtime is $O(|S|\cdot\tau^2)$. □

Lemma 6.11 (UPDATE$h(\overline{x}^{\text{new}},\overline{s}^{\text{new}},H^{\text{new}},L^{\text{new}})$)**.** *Under Invariant 6.6, given the new approximation pair $\overline{x}^{\text{new}},\overline{s}^{\text{new}}$, $H^{\text{new}}=\nabla^2\phi(\overline{x}^{\text{new}})$, and the lower Cholesky factor L^{new} of $A(H^{\text{new}})^{-1/2}A^\top$,*

the function UPDATEh *updates the implicit representation such that (i)-(iv) of Invariant 6.6 are preserved, and at the end of the function call, the central path pair are given by*

$$x = \widehat{x} + (H^{\text{new}})^{-1/2}\beta_x c_x - (H^{\text{new}})^{-1/2}\mathcal{W}^\top(\beta_x h + \varepsilon_x),$$
$$s = \widehat{s} + (H^{\text{new}})^{1/2}\mathcal{W}^\top(\beta_s h + \varepsilon_s).$$

Moreover, it takes $O(|S| \cdot \tau^2)$ time to perform UPDATEh *where $S = \{i \in [m] \mid \overline{x}_i^{\text{new}} \neq \overline{x}_i$ or $\overline{s}_i^{\text{new}} \neq \overline{s}_i\}$, and all the variables in Eq. (6.3) change in at most $O(|S| \cdot \tau)$ many entries.*

Proof. **Proof of Correctness:** First, we check (i)-(iv) of Invariant 6.6: For (i) and (ii), note that the values of μ_i and γ_i only depend on $\overline{x}_i$, $\overline{s}_i$ and $\overline{t}$, so it suffices to update only the entries of $\overline{\alpha}$ and $\overline{\delta_\mu}$ with indices in S. For (iii) and (iv), they are trivially satisfied by definition.

After Line 23, we have computed new versions of the variables $\widehat{x}, \widehat{s}, c_x, h, \varepsilon_x, \varepsilon_s$. For the implicit representation of x, they satisfy:

$$\begin{aligned}
&\widehat{x}^{\text{new}} + (H^{\text{new}})^{-1/2}\beta_x c_x^{\text{new}} - (H^{\text{new}})^{-1/2}\mathcal{W}^\top(\beta_x h^{\text{new}} + \varepsilon_x^{\text{new}}) \\
&= \widehat{x} + \beta_x(H_{\overline{x}}^{-1/2}c_x - (H^{\text{new}})^{-1/2}c_x^{\text{new}}) - (H_{\overline{x}}^{-1/2} - (H^{\text{new}})^{-1/2})\mathcal{W}^\top(\beta_x h + \varepsilon_x) \\
&\quad + (H^{\text{new}})^{-1/2}\beta_x c_x^{\text{new}} - (H^{\text{new}})^{-1/2}\mathcal{W}^\top(\beta_x h^{\text{new}} + \varepsilon_x^{\text{new}}) \\
&= \widehat{x} + \beta_x H_{\overline{x}}^{-1/2}c_x - H_{\overline{x}}^{-1/2}\mathcal{W}^\top(\beta_x h + \varepsilon_x) \\
&= x,
\end{aligned}$$

where the first step follows by the definition of $\widehat{x}^{\text{new}}$ and the second step follows by $\beta_x h^{\text{new}} + \varepsilon_x^{\text{new}} = \beta_x h + \varepsilon_x$ from Line 20. The proof of implicit representation of s is identical; we omit it here.

The remainder of the function updates the variables to their new versions, giving the desired conclusion of the lemma.

Proof of Runtime: Since $n_i = O(1)$ for all $i \in [m]$, it takes $O(|S|)$ time to compute $\overline{\alpha}^{\text{new}}$ and $\overline{\delta_\mu}^{\text{new}}$. Since H is a block-diagonal matrix, it takes $O(|S|)$ time to compute c_x^{new}. Similarly, it takes $O(|S|)$ time to compute $(H^{\text{new}})^{-1}\overline{\delta_\mu}^{\text{new}}$. We use $\hat{\mu}$ to denote $(H)^{-1}\overline{\delta_\mu}$ and $\hat{\mu}^{\text{new}}$ to denote $(H^{\text{new}})^{-1}\overline{\delta_\mu}^{\text{new}}$. Then, we can compute h^{new} by computing $h^{\text{new}} - h$. We note that

$$\begin{aligned}
h^{\text{new}} - h &= (L^{\text{new}})^{-1}A(H^{\text{new}})^{-1}\hat{\mu}^{\text{new}} - L^{-1}AH^{-1}\hat{\mu} \\
&= \underbrace{(L^{\text{new}})^{-1}A(H^{\text{new}})^{-1}\hat{\mu}^{\text{new}} - L^{-1}A(H^{\text{new}})^{-1}\hat{\mu}^{\text{new}}}_{\delta_h^{(1)}} + \underbrace{L^{-1}A(H^{\text{new}})^{-1}\hat{\mu}^{\text{new}} - L^{-1}AH^{-1}\hat{\mu}}_{\delta_h^{(2)}}.
\end{aligned}$$

We note that $\delta_h^{(2)}$ can be computed in $O(|S| \cdot \tau^2)$ time using Lemmas 5.1 and 5.4. By definition of L^{new}, we can compute $\delta_h^{(1)}$ in $O(|S| \cdot \tau^2)$ time. Let $S_L \stackrel{\text{def}}{=} \{i \in [d] \mid L_i^{\text{new}} \neq L_i\}$. By Lemma 5.9, S_L can be covered by $|S|$ many paths on the elimination tree. This also shows $h^{\text{new}} - h$ has $O(|S| \cdot \tau)$ many non-zero entries. Hence, we can compute $\varepsilon_x^{\text{new}}$ and $\varepsilon_s^{\text{new}}$ in $O(|S| \cdot \tau)$ time. Finally, since $\text{nnz}(H - H^{\text{new}}) = O(|S|)$, we can compute $(H^{-1/2} - (H^{\text{new}})^{-1/2})\mathcal{W}^\top$ and $(H^{1/2} - (H^{\text{new}})^{1/2})\mathcal{W}^\top$ in $O(|S| \cdot \tau^2)$ time by Lemmas 5.1 and 5.4.

Number of Coordinate Changes: Recall that $h^{\text{new}} - h$ has $O(|S| \cdot \tau)$ many non-zero entries, so the number of coordinate changes in $\varepsilon_x, \varepsilon_s$ is also bounded by $O(|S| \cdot \tau)$. The number of coordinate changes in $\widehat{x}$ and $\widehat{s}$ is bounded by $O(|S| \cdot \tau)$ since $(H^{-1/2} - (H^{\text{new}})^{-1/2})\mathcal{W}^\top$ and $(H^{1/2} - (H^{\text{new}})^{1/2})\mathcal{W}^\top$ both have $O(|S| \cdot \tau)$ many non-zero entries by Lemmas 5.1 and 5.4, and $\|c_x^{\text{new}} - c_x\|_0 = O(|S|)$. □

Lemma 6.12 (UPDATE$\mathcal{W}(L^{\text{new}}, H^{\text{new}})$). *Given $H^{\text{new}} = \nabla^2\phi(\overline{x}^{\text{new}})$, the lower Cholesky factor L^{new} of $A(H^{\text{new}})^{-1/2}A^\top$, and current implicit representation of (x, s) given by*

$$x = \widehat{x} + (H^{\text{new}})^{-1/2}\beta_x c_x - (H^{\text{new}})^{-1/2}\mathcal{W}^\top(\beta_x h + \varepsilon_x)$$
$$s = \widehat{s} + (H^{\text{new}})^{1/2}\mathcal{W}^\top(\beta_s h + \varepsilon_s)$$

UPDATE$\mathcal{W}$ takes $O((|\mathcal{S}|+|\mathcal{S}_L|)\cdot\tau^2)$ time to update the variables maintained by MULTISCALEREPRESENTATION, such that at the end of the function call, the central path pair is given by

$$x = \widehat{x}^{\text{new}} + (H^{\text{new}})^{-1/2}\beta_x c_x - (H^{\text{new}})^{-1/2}(\mathcal{W}^{\text{new}})^\top(\beta_x h + \varepsilon_x^{\text{new}})$$
$$s = \widehat{s}^{\text{new}} + (H^{\text{new}})^{1/2}(\mathcal{W}^{\text{new}})^\top(\beta_s h + \varepsilon_s^{\text{new}}),$$

where $\mathcal{W}^{\text{new}} \stackrel{\text{def}}{=} (L^{\text{new}})^{-1}A(H^{\text{new}})^{-1/2}$, $\mathcal{S} \stackrel{\text{def}}{=} \{i \in [m] \mid \overline{x}_i^{\text{new}} \neq \overline{x}_i \text{ or } \overline{s}_i^{\text{new}} \neq \overline{s}_i\}$, and $\mathcal{S}_L \stackrel{\text{def}}{=} \{i \in [d] \mid L_i^{\text{new}} \neq L_i\}$.

Moreover, when viewed as a set of vertices in $\mathcal{T}$, if $\mathcal{S}_L$ can be covered by $O(|\mathcal{S}|)$ many paths in $\mathcal{T}$, then the running time of UPDATE$\mathcal{W}$ is $O(|\mathcal{S}|\cdot\tau^2)$ and the number of coordinate changes in $\widehat{x}, \widehat{s}, \varepsilon_x$ and ε_s is bounded by $O(|\mathcal{S}|\cdot\tau)$.

Proof. **Proof of Correctness:**

First, we examine the reason behind the definition of $\varepsilon_x^{\text{new}}$: We want to find $\varepsilon_x^{\text{new}}$ such that

$$(L^{\text{new}})^{-\top}(\beta_x h + \varepsilon_x^{\text{new}}) = L^{-\top}(\beta_x h + \varepsilon_x).$$

Rearrange, we get

$$\begin{aligned}\varepsilon_x^{\text{new}} &= (L^{\text{new}})^\top L^{-\top}(\beta_x h + \varepsilon_x) - \beta_x h \\ &= (L^{\text{new}} - L + L)^\top L^{-\top}(\beta_x h + \varepsilon_x) - \beta_x h \\ &= (L^{\text{new}} - L)^\top L^{-\top}(\beta_x h + \varepsilon_x) + L^\top L^{-\top}(\beta_x h + \varepsilon_x) - \beta_x h \\ &= \varepsilon_x + (L^{\text{new}} - L)^\top L^{-\top}(\beta_x h + \varepsilon_x).\end{aligned}$$

Now, we check the implicit representation of x. At the end of the function, we have

$$\begin{aligned}&\widehat{x}^{\text{new}} + (H^{\text{new}})^{-1/2}\beta_x c_x - (H^{\text{new}})^{-1/2}(\mathcal{W}^{\text{new}})^\top(\beta_x h + \varepsilon_x^{\text{new}}) \\ &= \widehat{x}^{\text{new}} + (H^{\text{new}})^{-1/2}\beta_x c_x - (H^{\text{new}})^{-1/2}(H^{\text{new}})^{-1/2}A^\top(L^{\text{new}})^{-\top}(\beta_x h + \varepsilon_x^{\text{new}}) \\ &= \widehat{x}^{\text{new}} + (H^{\text{new}})^{-1/2}\beta_x c_x - (H^{\text{new}})^{-1}A^\top L^{-\top}(\beta_x h + \varepsilon_x) \\ &= x,\end{aligned}$$

where the first step follows by definition of $\mathcal{W}^{\text{new}}$, the second step follows by the property of $\varepsilon_x^{\text{new}}$ above, and the last step follows by definition of $\widehat{x}^{\text{new}}$.

The proofs for $\varepsilon_s^{\text{new}}$ and s are identical; we omit them here.

Proof of Runtime: Note that $\Delta\varepsilon_x \stackrel{\text{def}}{=} \varepsilon_x^{\text{new}} - \varepsilon_x = ((\beta_x h + \varepsilon_x)^\top L^{-1}(L^{\text{new}} - L))^\top$. Then, we can compute $L^{-1}(L^{\text{new}} - L)$ in $O(|\mathcal{S}_L|\cdot\tau^2)$ time by Lemmas 5.2 and 5.4, and therefore compute $\Delta\varepsilon_x$ in $O(|\mathcal{S}_L|\cdot\tau^2)$ time. By Lemmas 5.1 and 5.4, we can compute $((H^{\text{new}})^{-1/2} - H^{-1/2})A^\top L^{-\top}$ in $O(|\mathcal{S}|\cdot\tau^2)$ time, and the result has sparsity $O(|\mathcal{S}|\cdot\tau)$. Thus, we can compute $\widehat{x}^{\text{new}}$ and $\widehat{s}^{\text{new}}$ in $O(|\mathcal{S}|\cdot\tau^2)$ time. In total, the function runs in $O((|\mathcal{S}| + |\mathcal{S}_L|)\cdot\tau^2)$ time.

For each path $\mathcal{P}$, we can directly compute $(L^{-\top}(\beta_x h + \varepsilon_x))|_{\mathcal{P}}$ in time $O(\tau^2)$ by Lemma 5.6. Then, it takes $O(\tau^2)$ to compute $\Delta\varepsilon_x$ for each path. Hence, the update time in this case is bounded by $O(|\mathcal{S}| \cdot \tau^2)$.

Number of Coordinate Changes: By Lemmas 5.1 and 5.4, $((H^{\text{new}})^{-1/2} - H^{-1/2})A^\top L^{-\top}$ has sparsity $O(|\mathcal{S}|\cdot\tau)$. For each path $\mathcal{P}$, the solution of $L^{-1}(L^{\text{new}} - L)$ is a $\tau \times \tau$ submatrix by Lemmas 5.2 and 5.4, leading to $\varepsilon_x^{\text{new}} - \varepsilon_x$ and $\varepsilon_s^{\text{new}} - \varepsilon_s$ having sparsity $O(|\mathcal{S}| \cdot \tau)$. □

6.3 Approximating A Sequence of Vectors

The central path maintenance involves a number of dynamic vectors, e.g. $\widehat{x}, \widehat{s}, c_x$ from Eq. (6.3). These can essentially be viewed as online sequences of vectors, where the sequence length is the number of central path steps. To work with these vector variables efficiently over the central path steps, we maintain their ℓ_∞-approximations.

In this section, we introduce the techniques for obtaining ℓ_∞-approximations of an online sequence of vectors using a *sampling tree* data structure, crucially avoiding reading the input vectors in full at all times to lower the runtime. The underlying idea is standard in sampling, heavy-hitters, and sketching, see e.g. [CM05]. We explain how it is used in the context of central path maintenance in subsequent sections.

Definition 6.13. A *sampling tree* $(\mathcal{S}, \chi)$ of $\mathbb{R}^n$ consists of a constant degree rooted tree $\mathcal{S} = (V, E)$ and a labelling of the vertices $\chi : V \to 2^{[n]}$, such that:

- $\chi(\text{root}) = [n]$,
- If v is a leaf node of $\mathcal{S}$, then $|\chi(v)| = 1$,
- For any node v of $\mathcal{S}$, the set $\{\chi(c) \mid c \text{ is a child of } v\}$ forms a partition of $\chi(v)$.

Theorem 6.14. *Given a sampling tree $(\mathcal{S}, \chi)$ with height η, some $0 < \varepsilon_{\text{apx}}, \delta_{\text{apx}} < 1$, length of input sequence k, a fixed but unknown JL-matrix $\Phi \in \mathbb{R}^{r \times n}$ where $r = \Omega(\eta^2 \log(nk/\delta_{\text{apx}}))$, and upper bound $\zeta > 0$ such that the sequence $\{y^{(\ell)}\}_{\ell=1}^k$ satisfies $\|y^{(\ell)} - y^{(\ell-1)}\|_2 \leqslant \zeta$ for all $\ell \in [k]$, the data structure ℓ_∞-*Approximates* (Algorithms 7 and 8) supports k calls to* Query*, such that:*

In the ℓ-th call to Query*, the data structure can indirectly access $\{y^{(i)}\}_{i=1}^\ell$ using the list of oracles $\{\mathcal{O}[y^{(i)}]\}_{i=1}^\ell$ as follows:*

> *$\mathcal{O}[y^{(i)}]$.*TypeI*(v): access to the vector $\Phi_{\chi(v)} y^{(i)}$ for node $v \in \mathcal{S}$,*
>
> *$\mathcal{O}[y^{(i)}]$.*TypeII*(j): access to entry $y_j^{(i)}$ for $j \in [n]$,*

and returns $z^{(\ell)}$ such that $\|z^{(\ell)} - y^{(\ell)}\|_\infty \leqslant \varepsilon_{\text{apx}}$ with probability at least $1 - \delta_{\text{apx}}/k$.

Over the entire input sequence, the data structure makes $O(\eta \cdot \zeta^2 k^2/\varepsilon_{\text{apx}}^2 \cdot \text{poly}\log(nk\zeta/(\varepsilon_{\text{apx}} \cdot \delta_{\text{apx}})))$ type-I oracle calls and $O(\zeta^2 k^2/\varepsilon_{\text{apx}}^2 \cdot \text{poly}\log(nk\zeta/(\varepsilon_{\text{apx}} \cdot \delta_{\text{apx}})))$ type-II oracle calls, with $O(\eta \cdot r \cdot \zeta^2 k^2/\varepsilon_{\text{apx}}^2 \cdot \text{poly}\log(nk\zeta/(\varepsilon_{\text{apx}} \cdot \delta_{\text{apx}})))$ additional computation time. It maintains $\{z^{(\ell)}\}_{\ell=1}^k$ such that $\|z^{(\ell)} - y^{(\ell)}\|_\infty \leqslant \varepsilon_{\text{apx}}$ for all $\ell \in [k]$ with success probability at least $1 - \delta_{\text{apx}}$.

For any vector y in the sequence, we show that $\Phi_{\chi(v)} y$ allows us to obtain a $(1 \pm \frac{1}{\eta})$-approximation of $\|y|_{\chi(v)}\|_2^2$. With such estimations, we can sample a coordinate of y with probability proportional to y_i^2 using $O(\eta)$ many oracle calls, using a random descent on the sampling tree, where we choose each child with probability proportional to their estimation. This further enables us to obtain a

Algorithm 7 ℓ_∞ Maintenance Data Structure – Initialize and Query

data structure ℓ_∞-APPROXIMATES
private : members
  Sampling tree $(\mathcal{S}, \chi)$ ▷ Fixed global constant
  $\varepsilon_{\text{apx}}, \delta_{\text{apx}} \in (0,1)$ ▷ error parameter and failure probability parameter
  $\ell, k \in \mathbb{N}$ ▷ counter and total length of sequence
  $\zeta \in \mathbb{R}_+$ ▷ upper bound of $\|y^{(\ell+1)} - y^{(\ell)}\|_2$
  list $\{\mathcal{O}\{y^{(i)}\}\}_{i=0}^{k}$ ▷ sequence of oracles of input vectors $y^{(i)}$
  list $\{z^{(i)}\}_{i=0}^{k}$ ▷ sequence of approximations
end members
procedure INITIALIZE($\mathcal{S}, \chi, \varepsilon_{\text{apx}} \in (0,1), \delta_{\text{apx}} \in (0,1), \zeta \in \mathbb{R}_+, k \in \mathbb{N}$)
  $\ell \leftarrow 0, k \leftarrow k$
  $\varepsilon_{\text{apx}} \leftarrow \varepsilon_{\text{apx}},\ \delta_{\text{apx}} \leftarrow \delta_{\text{apx}},\ \zeta \leftarrow \zeta$
  $y^{(0)} \leftarrow \mathbf{0}, z^{(0)} \leftarrow \mathbf{0}$
end procedure
procedure QUERY($\mathcal{O}[y^{\text{new}}]$)
  $\ell \leftarrow \ell + 1$
  $\mathcal{O}[y^{(\ell)}] \leftarrow \mathcal{O}[y^{\text{new}}]$ ▷ store new oracle to list
  $z^{(\ell)} \leftarrow z^{(\ell-1)}$ ▷ first set the approximation to be the same as the previous
  $\mathcal{I} \leftarrow \emptyset$ ▷ set of indices i where we may need to update $z_i^{(\ell)}$
  for $j = 0, 1, \ldots, \lfloor \log \ell \rfloor$ **do**
    $\mathcal{I}_j \leftarrow \emptyset$
    if $\ell \equiv 0 \mod 2^j$ **then**
      for $O(4^j \zeta^2 / \varepsilon_{\text{apx}}^2 \cdot \log^3 k \cdot \log(nk\zeta/(\varepsilon_{\text{apx}}\delta_{\text{apx}})))$ many times **do**
        $\mathcal{I}_j \leftarrow \mathcal{I}_j \cup \{\text{SAMPLE}(\ell - 2^j + 1, \ell)\}$
      end for
    end if
    $\mathcal{I} \leftarrow \mathcal{I} \cup \mathcal{I}_j$
  end for
  for all $i \in \mathcal{I}$ **do**
    $z_i \leftarrow \mathcal{O}[y^{(\ell)}].\text{TYPEII}(i)$
    if $|z_i - (z^{(\ell)})_i| > \varepsilon_{\text{apx}}$ **then** ▷ Set $z_i^{(\ell)} \leftarrow y_i^{(\ell)}$ when error is larger than ε_{apx}
      $(z^{(\ell)})_i \leftarrow z_i$
    end if
  end for
  return $z^{(\ell)}$
end procedure

Algorithm 8 ℓ_∞ Maintenance Data Structure – Sample and Estimate

procedure SAMPLE($a \in [k], b \in [k]$)
  repeat
    $v \leftarrow \text{root}(\mathcal{S}), p \leftarrow 1$
    while v is not a leaf node **do**
      Sample child v' of v with probability $p_{v'} \stackrel{\text{def}}{=} \dfrac{\text{ESTIMATE}(a, b, v')}{\sum_{u \text{ a child of } v} \text{ESTIMATE}(a, b, u)}$
      $p \leftarrow p \cdot p_{v'}$
      $v \leftarrow v'$
    end while
    ▷ Since v is a leaf node, $\chi(v)$ consists of a single index from $[n]$
    ▷ For notational purposes, suppose $\chi(v) = \{i\}$
    $y_i^{(a)} \leftarrow \mathcal{O}[y^{(a)}].\text{TYPEII}(i)$
    $y_i^{(b)} \leftarrow \mathcal{O}[y^{(b)}].\text{TYPEII}(i)$
    with probability $(y_i^{(a)} - y_i^{(b)})^2/(10 \cdot p \cdot \text{ESTIMATE}(a, b, \text{root}(\mathcal{S})))$, **return** i
  until false
end procedure
procedure ESTIMATE($a \in [k], b \in [k], v \in \mathcal{S}$)
  return $\|\mathcal{O}\{y^{(a)}\}.\text{TYPEI}(v) - \mathcal{O}\{y^{(b)}\}.\text{TYPEI}(v)\|_2^2$
end procedure

$(1 \pm \varepsilon)$-approximation of y in the ℓ_∞-norm using $O(\|y\|_2^2/\varepsilon^2 \log(\|y\|_2/\varepsilon))$ type-II oracle calls by the coupon collection problem. By linearity of Φ, this then allows us to approximate $y^{(b)} - y^{(a)}$.

Instead of directly estimating $y^{(\ell)}$ for each ℓ, we obtain a ℓ_∞-approximation for all $y^{(b)} - y^{(a)}$, where $[a, b] \stackrel{\text{def}}{=} \{a, a+1, \ldots, b-1, b\}$ is in the set of *dyadic intervals* of $[k]$.

Definition 6.15. Let k be a positive integer. The set of *dyadic intervals* of $[k]$ is

$$\{[i \cdot 2^j + 1, (i+1) \cdot 2^j] \mid i, j \in \mathbb{N}; (i+1) \cdot 2^j \leqslant k\}.$$

The following lemma tells us why dyadic intervals help to keep the error sub-linear to the size of the intervals.

Lemma 6.16 (folklore)**.** *Any interval $[a, b]$ in $[k]$ can be partitioned into at most $2 \log k$ dyadic intervals.*

Hence, it suffices to obtain a $(1 \pm \frac{\varepsilon}{2 \log k})$ approximation for every dyadic interval.

Before we prove Theorem 6.14, we show that the function SAMPLE(a, b) in the data structure indeed samples a coordinate i of $(y^{(b)} - y^{(a)})$ with probability proportional to $(y^{(b)} - y^{(a)})_i^2$.

Lemma 6.17 (SAMPLE)**.** *Under the same setting as Theorem 6.14, conditioned on the function ESTIMATE(a, b, v) always returns $\|\Phi_{\chi(v)}(y^{(a)} - y^{(b)})\|_2^2 = (1 \pm \frac{1}{2\eta})\|(y^{(a)} - y^{(b)})|_{\chi(v)}\|_2^2$, the function SAMPLE$(a, b)$ on Line 1 samples a coordinate i proportional to $(y^{(b)} - y^{(a)})_i^2$ with $O(\eta \cdot r)$ expected running time, and makes $O(\eta)$ many type-I oracle calls and $O(1)$ many type-II oracle calls in expectation.*

Proof. **Proof of Correctness:** Let δ_y denote $y^{(b)} - y^{(a)}$ in this proof.

Let $v_1, v_2, \ldots, v_m \in V$ be the sequence of nodes visited in the while-loop on Lines 4 to 8, where v_1 is the root node of $\mathcal{S}$ and v_m is the leaf node with $\chi(v_m) = \{i\}$, for some $i \in [n]$ and $m \leqslant \eta$. Observe that at end of the while-loop, p is exactly the probability that v_m is sampled. Hence, the algorithm outputs v_m with probability

$$\Pr[i \text{ is outputed}] = \Pr[v_m \text{ is sampled}] \cdot \Pr[i \text{ is returned} \mid v_m] = p \cdot \frac{(y_i^{(a)} - y_i^{(b)})^2}{10 \cdot p\|\Phi\delta_y\|_2^2} = \frac{\delta_{y,i}^2}{10 \cdot \|\Phi\delta_y\|_2^2}.$$

This shows SAMPLE outputs a coordinate i with probability proportional to $\delta_{y,i}^2$.

Proof of Runtime: Consider one iteration of the repeat-loop (Lines 2 to 14), where the inner while-loop visits the node sequence $v_1, \ldots, v_m$. The probability to choose child v_{j+1} from node v_j is $\|\Phi_{\chi(v_{j+1})}\delta_y\|_2^2 / \sum_{u \text{ a child of } v_j} \|\Phi_{\chi(u)}\delta_y\|_2^2$. By our condition on the function ESTIMATE(a, b, v), we have $\sum_{c \text{ a child of } v_j} \|\Phi_{\chi(u)}\delta_y\|_2^2 = (1 \pm \frac{1}{\eta})\|\Phi_{\chi(v_j)}\delta_y\|_2^2$. Hence,

$$\Pr[v_{j+1} \mid v_j] \leqslant \left(1 + \frac{1}{\eta}\right) \frac{\|\Phi_{\chi(v_{j+1})}\delta_y\|_2^2}{\|\Phi_{\chi(v_j)}\delta_y\|_2^2}.$$

Taking the telescoping product, we have

$$p \leqslant (1 + \frac{1}{\eta})^\eta \frac{\|\Phi_{\chi(v_2)}\delta_y\|_2^2}{\|\Phi_{\chi(v_1)}\delta_y\|_2^2} \times \frac{\|\Phi_{\chi(v_3)}\delta_y\|_2^2}{\|\Phi_{\chi(v_2)}\delta_y\|_2^2} \times \ldots \times \frac{\|\Phi_{\chi(v_m)}\delta_y\|_2^2}{\|\Phi_{\chi(v_{m-1})}\delta_y\|_2^2} \leqslant 3 \cdot \frac{\|\Phi_{\{i\}}\delta_y\|_2^2}{\|\Phi\delta_y\|_2^2}.$$

This iteration of the repeat-loop returns with probability

$$\frac{\delta_{y,i}^2}{10p\|\Phi\delta_y\|_2^2} \geq \frac{\delta_{y,i}^2}{10\|\Phi\delta_y\|_2^2} \cdot \frac{\|\Phi\delta_y\|_2^2}{3\|\Phi_{\{i\}}\delta_y\|_2^2} \geq \frac{1}{50},$$

where we used $\|\Phi_{\{i\}}\delta_y\|_2^2 \leqslant (1 + \frac{1}{2\eta})\delta_{y,i}^2$. Hence, the expected number of iteration is $O(1)$.

For each iteration of the repeat-loop, the inner while-loop on Lines 4 to 8 traverses a path from the root to a leaf node in $\mathcal{S}$, so we can bound the number of iterations by η. At each node v_j along the path with c children, sampling on Line 5 requires $O(r)$ time and $2c$ many type-I queries. At the end of the descent to a leaf, we make 2 type-II queries on Lines 11 and 12 and then 2 type-I queries. Hence, each call of SAMPLE takes $O(\eta \cdot r)$ time, $O(\eta)$ type-I queries, and $O(1)$ type-II queries in expectation. □

Proof of Theorem 6.14. **Proof of Runtime:** For a fix $j \in [\lfloor \log k \rfloor]$, the data structure makes $O(4^j\zeta^2 \cdot \log^3 k \cdot \log(nk\zeta/(\varepsilon_{\text{apx}} \cdot \delta_{\text{apx}}))/\varepsilon_{\text{apx}}^2)$ many calls to SAMPLE, for each 2^j calls to QUERY. Since there are k calls to QUERY in total, the total number of SAMPLE call is

$$\sum_{j=1}^{\lfloor \log k \rfloor} \frac{k}{2^j} \cdot O\left(\frac{4^j\zeta^2}{\varepsilon_{\text{apx}}^2} \cdot \log^3 k \cdot \log(\frac{nk\zeta}{\varepsilon_{\text{apx}} \cdot \delta_{\text{apx}}})\right) = O\left(\frac{k^2\zeta^2 \cdot \log^3 k \cdot \log(nk\zeta/(\varepsilon_{\text{apx}}\delta_{\text{apx}}))}{\varepsilon_{\text{apx}}^2}\right).$$

Combining the bound above and Lemma 6.17, the total runtime is $O(\frac{\eta r k^2\zeta^2}{\varepsilon_{\text{apx}}^2} \operatorname{poly}\log(nk\zeta/(\varepsilon_{\text{apx}} \cdot \delta_{\text{apx}})))$ in expectation, with $O(k^2\zeta^2/\varepsilon_{\text{apx}}^2 \cdot \operatorname{poly}\log(nk\zeta/(\varepsilon_{\text{apx}} \cdot \delta_{\text{apx}})))$ type-II queries and $O(\eta k^2\zeta^2/\varepsilon_{\text{apx}}^2 \cdot \operatorname{poly}\log(nk\zeta/(\varepsilon_{\text{apx}} \cdot \delta_{\text{apx}})))$ type-I queries in expectation.

Proof of Correctness: By the coupon collection problem, for any vector v, we can find all coordinates i of v such that $|v_i| \geq \frac{1}{a}\|v\|_2$ with high probability, by sampling $O(a^2 \log a)$ many coordinates, each time sampling coordinate i with probability proportional to v_i^2. By our choice of $r = \Omega(\eta^2 \log(nk/\delta_{\text{apx}}))$ and union bound over all type-I queries, we have the function $\textsc{Estimate}(a, b, v)$ always return $\|\Phi_{\chi(v)}(y^{(a)} - y^{(b)})\|_2^2 = (1 \pm \frac{1}{2\eta})\|(y^{(a)} - y^{(b)})|_{\chi(v)}\|_2^2$ with probability at least $1 - \frac{\delta_{\text{apx}}}{2}$.

Fix $j \in [\lfloor \log \ell \rfloor]$. We sample $O(4^j \zeta^2 \cdot \log^3 k \cdot \log(nk\zeta/(\varepsilon_{\text{apx}}\delta_{\text{apx}}))/\varepsilon_{\text{apx}}^2)$ many coordinates for $y^{(\ell)} - y^{(\ell-2^j)}$ (Lines 22 to 26). By the triangle inequality, and the property that consecutive $y^{(\ell)}$'s change slowly, we have $\|y^{(\ell)} - y^{(\ell-2^j)}\|_2 \leqslant 2^j \cdot \zeta$. Hence, $\mathcal{I}_j$ contains all coordinates i such that $|(y^{(\ell)} - y^{(\ell-2^j)})_i| > O(\varepsilon_{\text{apx}}/\log k)$ with success probability $1 - \delta_{\text{apx}}/\operatorname{poly}(nk)$.

Taking the union bound over all j, we have that $\mathcal{I} = \bigcup \mathcal{I}_j$ contains all coordinates i such that $|(y^{(\ell)} - y^{(\ell-2^j+1)})_i| > O(\varepsilon_{\text{apx}}/\log k)$ for all $\ell \in [k]$ and $j \in [\lfloor \log k \rfloor]$ with probability at least $1 - \frac{\delta_{\text{apx}}}{2}$.

Consider the data structure at the end of iteration ℓ. Fix a coordinate i, and let ℓ_i be the iteration when the value of z_i was set by Lines 30 and 32. In other words, $z_i^{(\ell)} = y_i^{(\ell_i)}$. Let $\delta_{y,i}^{(t)} = (y^{(t)} - y^{(t-1)})_i$ for any $t \in [k]$. Then,

$$|y_i^{(\ell)} - z_i^{(\ell)}| = |y_i^{(\ell)} - y_i^{(\ell_i)}| = \left| \sum_{t=\ell_i+1}^{\ell} \delta_{y,i}^{(t)} \right| = \left| \sum_{s=1}^{2\log k} \sum_{t=a_s 2^{js}+1}^{(a_s+1)2^{js}} \delta_{y,i}^{(t)} \right| \leqslant 2\log k \cdot O(\varepsilon_{\text{apx}}/\log k) \leqslant \varepsilon_{\text{apx}},$$

where the third step follows by Lemma 6.16, and the fourth step follows by the fact that ℓ_i is the last time z_i was updated, so for any $[a \cdot 2^j + 1, (a+1)2^j] \subset [\ell_i, \ell]$, we have $|(y^{((a+1)2^j+1)} - y^{(a \cdot 2^j)})_i| < O(\varepsilon_{\text{apx}}/\log k)$.

Taking the union bound again, we have the data structure succeeds with probability at least $1 - \delta_{\text{apx}}$.

□

6.4 Sketching A Sequence of Vectors

In this section, we show how to construct an oracle used in Theorem 6.14 that supports type-I queries for a sequence of vectors.

Theorem 6.18. *Given a sampling tree $(\mathcal{S}, \chi)$ of $\mathbb{R}^n$ with height η, and a JL sketching matrix $\Phi \in \mathbb{R}^{r \times n}$, the data structure $\textsc{VectorSketch}$ maintains $y_v \stackrel{\text{def}}{=} \Phi_{\chi(v)} h$ for all nodes v in the sampling tree through the following functions:*

- *$\textsc{Initialize}(\mathcal{S}, \chi, \Phi \in \mathbb{R}^{r \times n}, h)$: Initializes the data structure in time $O(n \cdot \eta \cdot r)$, so that node $v \in \mathcal{S}$ maintains y_v.*
- *$\textsc{Update}(h^{\text{new}} \in \mathbb{R}^n)$: Maintains the data structure for $h \leftarrow h^{\text{new}}$ in $O(\eta \cdot r \cdot \|h^{\text{new}} - h\|_0)$ time.*
- *$\textsc{Query}(v \in V(\mathcal{S}))$: Outputs $\Phi_{\chi(v)} h$ in $O(r)$ time.*

Proof. **Correctness:** In $\textsc{Initialize}$, we calculate y_v directly for all v. In $\textsc{Update}$ when h is updated to h^{new}, note that y_v maintained at a node v needs to be updated if and only if $(h^{\text{new}} - h)_{\chi(v)} \neq \mathbf{0}$. Hence, for each index i with $(h^{\text{new}} - h)_i \neq 0$, we need to update at all nodes v with $i \in \chi(v)$, which is exactly the path from the leaf node u with $\chi(u) = \{i\}$ to the root of $\mathcal{S}$. Moreover, the update due to coordinate i is precisely $\Phi_{\{i\}}(h^{\text{new}} - h)$.

Algorithm 9 Vector Sketching Data Structure

datastructure VECTORSKETCH
private : members
  $\Phi \in \mathbb{R}^{r \times n}$ ▷ JL matrix
  Sampling tree $(\mathcal{S}, \chi)$
  $h \in \mathbb{R}^n$ ▷ latest input vector
  LIST $\{y_v\}_{v \in V(\mathcal{S})}$ ▷ sketches indexed by nodes of $\mathcal{S}$, where $y_v \stackrel{\text{def}}{=} \Phi_{\chi(v)} h$
end members
procedure INITIALIZE($\mathcal{S}, \chi, \Phi \in \mathbb{R}^{r \times n}, h \in \mathbb{R}^n$)
  $(\mathcal{S}, \chi) \leftarrow (\mathcal{S}, \chi)$
  $\Phi \leftarrow \Phi$
  $h \leftarrow h$
  for all $v \in \mathcal{S}$ **do**
    $y_v \leftarrow \Phi_{\chi(v)} h$
  end for
end procedure
procedure UPDATE(h^{new})
  for all i such that $h_i^{\text{new}} \neq h_i$ **do**
    Find leaf node v of $\mathcal{S}$ such that $\chi(v) = \{i\}$
    for all node $u \in \mathcal{P}^{\mathcal{S}}(v)$ **do** ▷ where $\mathcal{P}(v)$ is the path from v to the root in $\mathcal{S}$
      $y_v \leftarrow y_v - \Phi_{\{i\}} h + \Phi_{\{i\}} h^{\text{new}}$
    end for
  end for
  $h \leftarrow h^{\text{new}}$
end procedure
procedure QUERY($v \in \mathcal{S}$)
  return y_v
end procedure

Runtime: For INITIALIZE, let $\mathsf{Layer}^{\mathcal{S}}(i) \stackrel{\text{def}}{=} \{v \in V(\mathcal{S}) \mid \mathsf{depth}(v) = i\}$ be the set of nodes in the i-th layer of the sampling tree. By property (3) of the sampling tree in Definition 6.13, $\chi(v) \cap \chi(u) = \emptyset$ for any $u, v \in \mathsf{Layer}^{\mathcal{S}}(i)$. Hence, we can compute $\Phi_{\chi(v)}h$ for all $v \in \mathsf{Layer}^{\mathcal{S}}(i)$ in time $O(n \cdot r)$. Since $\mathcal{S}$ has height η, initialization takes $O(n \cdot \eta \cdot r)$ time.

For UPDATE, observe that the outer for-loop runs $\|h^{\text{new}} - h\|_0$ times. The inner for-loop iterates at most η times, as it traverses up a path from a leaf node to the root in $\mathcal{S}$. For each node on the path, we need to compute $z_v - \Phi_{\{i\}}h + \Phi_{\{i\}}h^{\text{new}}$ which takes r time. Thus, we can bound the total update time by $O(\eta \cdot r \cdot \|h^{\text{new}} - h\|_0)$.

To find the leaf node v such that $\chi(v) = \{i\}$, we note the function χ is fixed, so it can be pre-process during initialization in $O(n)$ time.

The query time follows by the fact that y_v is an r-dimensional vector. □

6.5 Sketching the Multiscale Representation via Simple Sampling Tree

The previous section shows how to construct an oracle used in Theorem 6.14 that supports type-I queries for a sequence of slowing changing vectors. However, not all vector variables in our main central path maintenance data structure change slowly across consecutive central path steps. In particular, we also want to maintain the sketches of matrix-vector products involving $\mathcal{W}^\top$, such as $\mathcal{W}^\top h, \mathcal{W}^\top \varepsilon_x$ and $\mathcal{W}^\top \varepsilon_s$ from Eq. (6.3).

Consider maintaining ℓ_∞-approximations of the sequence of $\mathcal{W}^\top h$: Using VECTORSKETCH presented in Theorem 6.18 directly yields a data structure whose update time at iteration $\ell + 1$ is a function of $\|(\mathcal{W}^\top h)^{(\ell+1)} - (\mathcal{W}^\top h)^{(\ell)}\|_0$. Recall that $\mathcal{W}$ and h change between central path steps as a function of changes in $\overline{x}$; unfortunately, even if $\overline{x}$ only changes in a single coordinate, $\mathcal{W}^\top h$ can change densely. Hence, we would like to design a modified data structure whose update time is a function of $\|\overline{x}^{(\ell+1)} - \overline{x}^{(\ell)}\|_0$ and $\|h^{(\ell+1)} - h^{(\ell)}\|_0$ instead. In this section and the next, we present sketching data structures that serve as the oracle needed in Theorem 6.14 for type-I queries, specifically for the case when the online sequence of vectors is of the form $\{(\mathcal{W}^\top h)^{(\ell)}\}_{\ell=0}^{k}$, for dynamic $\mathcal{W}$ and h.

To this end, we have to crucially utilize the structure of the lower Cholesky factor L and the elimination tree $\mathcal{T}$. In this section, we present a simple construction of a sampling tree which preserves the structural property of the elimination tree $\mathcal{T}$, and an intuitive implementation of the sketching maintenance data structure with $\widetilde{O}(\text{poly}(\tau))$ amortized runtime per update. In Section 6.6, we show a more involved data structure to lower the runtime, using many of the same ideas.

This section mainly serves to illustrate our approach to maintaining the sketches, so *we assume each $n_i = 1$ and $m = n$* in the block structure of A for simplicity of presentation; the assumption is removed in Section 6.6.

Theorem 6.19. *Given the constraint matrix A, its binary elimination tree $\mathcal{T}$ with height τ, a JL matrix $\Phi \in \mathbb{R}^{r \times n}$, and a sampling tree $(\mathcal{S}, \chi)$ with height $\eta \leqslant O(\tau + \log(n))$ constructed as in Section 6.5.1, the data structure SIMPLESKETCH (Algorithms 10 and 11) maintains the sketch $\Phi_{\chi(v)}\mathcal{W}^\top h = \Phi_{\chi(v)}H_{\overline{x}}^{-1/2}A^\top L_{\overline{x}}^{-\top}$ at every node $v \in \mathcal{S}$ through following operations:*

- *INITIALIZE$(S, \chi, \Phi, \overline{x}, h)$: Initializes the data structure in $O(n \cdot \tau \cdot \eta \cdot r)$ time, so that node $v \in \mathcal{S}$ maintains the sketch $\Phi_{\chi(v)}\mathcal{W}^\top h = \Phi_{\chi(v)}H_{\overline{x}}^{-1/2}A^\top L_{\overline{x}}^{-\top}$.*

- *UPDATE*$(\overline{x}^{\text{new}}, h^{\text{new}})$*: Updates all sketches in* $\mathcal{S}$ *to reflect* $\mathcal{W}$ *updating to* $\mathcal{W}^{\text{new}}$ *and* h *to* h^{new}*, where* $\mathcal{W}^{\text{new}}$ *is given implicitly by* $\overline{x}^{\text{new}}$*. This function runs in* $O(\|\overline{x}^{\text{new}} - \overline{x}\|_0 \cdot \tau^2 \cdot \eta \cdot r) + O(\|h^{\text{new}} - h\|_0 \cdot \tau \cdot r)$ *time.*
- *QUERY*(v)*: Outputs* $\Phi_{\chi(v)} \mathcal{W}^\top h$ *in* $O(\tau^2 \cdot r)$ *time.*

In Section 6.5.1, we give the construction of the sampling tree. In Section 6.5.2, we give the analysis of each individual function.

Symbol	Definition
$\mathcal{T}$	elimination tree with vertex set $\{1, \ldots, d\}$
$(\mathcal{S}, \chi)$	sampling tree. By convention, we call vertices of $\mathcal{S}$ *nodes*
$\mathcal{D}(v)$	set of nodes in the subtree rooted at v (inclusively)
$\mathcal{P}(v)$	set of nodes on the path from v to the root (inclusively)
$\mathsf{depth}(v)$	depth of node v in tree ($\mathsf{depth}(\text{root}) = 1$)
$\mathsf{low}(a)$	the lowest node in tree in the nonzero pattern of a vector a
$\overline{\mathsf{low}}(A)$	the lowest node in tree in the nonzero column pattern of A
$\mathsf{LCA}(u, v)$	the lowest common ancestor of u and v in tree
A_S	matrix A restricted to coordinates/blocks indexed by nodes in set S
$f^{\mathcal{S}}(v)$ or $f^{\mathcal{T}}(v)$	function $f(v)$ for the specified tree $\mathcal{S}$ or $\mathcal{T}$

Table 1: Notations in this section

6.5.1 Simple Sampling Tree Construction

We begin with the construction of a *simple sampling tree* $(\mathcal{S}, \chi)$ which has n leaf nodes, based on the elimination tree $\mathcal{T}$. Recall $\mathcal{T}$ has d vertices given by the set $\{1, 2, \ldots, d\}$, where vertex i correspond to row i of A.

First, we define the function $\mathsf{low}^{\mathcal{T}}(a) : \mathbb{R}^d \to [d]$ by

$$\mathsf{low}^{\mathcal{T}}(a) = \arg \max_{i \in \{i | a_i \neq 0\}} \mathsf{depth}^{\mathcal{T}}(i),$$

which gives the node i at the lowest level in $\mathcal{T}$ such that $a_i \neq 0$. Note that $\mathsf{low}^{\mathcal{T}}(A_j)$ is well-defined, since the non-zero pattern of A_j is a subset of a path in $\mathcal{T}$ by Lemma 5.1.

For each vertex $i \in \mathcal{T}$, we construct a balanced binary tree on all new nodes, rooted at node i' and with leaf nodes given by the set $F_i = \{c_j \mid \mathsf{low}^{\mathcal{T}}(A_j) = i\}$. Observe if i is a leaf node of $\mathcal{T}$, then F_i is non-empty. We construct a new tree $\mathcal{S}$ by beginning with $\mathcal{S} \leftarrow \mathcal{T}$, and then for each $i \in \mathcal{T}$, attaching the new subtree rooted at i' under i in $\mathcal{S}$. After this, the set of leaf nodes in $\mathcal{S}$ is given by $\bigcup_{i \in \mathcal{T}} F_i = \{c_j \mid j \in [n]\}$, with every leaf corresponding to a distinct column of A.

For each $v \in \mathcal{S}$, we recursively define its labelling $\chi(v)$ by:

$$\chi(v) = \begin{cases} i & \text{if } v = c_i \text{ is a leaf node} \\ \bigcup_{u \text{ a child of } v \text{ in } \mathcal{S}} \chi(u) & \text{else} \end{cases}$$

In particular, for $i \in \mathcal{S} \cap \mathcal{T} = \{1, \ldots, d\}$, $\chi(i)$ satisfies:

$$\chi(i) = \{j \mid \mathsf{low}^{\mathcal{T}}(A_j) = i\} \cup \bigcup_{j \text{ a child of } i \text{ in } \mathcal{T}} \chi(j). \tag{6.4}$$

Let this newly constructed $(\mathcal{S}, \chi)$ be the *simple sampling tree*. Since $\mathcal{T}$ is a binary tree, $\mathcal{S}$ is a degree-3 tree. An example is shown in Fig. 6.1.

Definition 6.20 ($\mathcal{T}(v)$)**.** Recall we have $V(\mathcal{T}) = \{1, \ldots, d\} \subset V(\mathcal{S})$. For a node $v \in \mathcal{S}$, we define its $\mathcal{T}$*-ancestor* $\mathcal{T}(v)$ to be the lowest ancestor of v in $\mathcal{S}$ that is also in $\mathcal{T}$. In particular, if $v \in \{1, \ldots, d\}$, then $\mathcal{T}(v) = v$.

For example, in Fig. 6.1, $\mathcal{T}(5) = 5$ and $\mathcal{T}(c_3) = 4$, where c_3 is the bottom left node in $\mathcal{S}$.

Theorem 6.21. *Given an elimination tree $\mathcal{T}$ with height τ, the simple sampling tree with height $\tau + O(\log n) = O(\tau)$ can be constructed in $O(n\tau + n \log n)$ time.*

Proof. Since the newly added balanced binary tree under each $v \in \mathcal{T}$ has height at most $O(\log n)$, the height of the sampling tree is bounded by $\tau + O(\log n) = O(\tau)$.

For each column A_i, we can find $\mathsf{low}(A_i)$ in time $O(\tau)$ since $\mathrm{nnz}(A_j) = O(\tau)$ by Lemma 5.1. Hence, we can find F_i for every $i \in \mathcal{T}$ in $O(n\tau)$ time in total. Constructing the balanced binary tree rooted at every i' takes at most $O(n \log n)$ total time. Finding the sets $\chi(v)$ at every $v \in \mathcal{S}$ takes $O(n \log n)$ total time. Hence, we can construct $(\mathcal{S}, \chi)$ in $O(n\tau + n \log n)$ time. □

6.5.2 Data Structure for Sketching

Now, we discuss how to maintain the sketches $\Phi_{\chi(v)} \mathcal{W}^\top h$ at every node $v \in \mathcal{S}$. Recall the non-zero pattern of the Cholesky factor L is reflected in the elimination tree. Specifically, the non-zero pattern of column L_i is a subset of the path from i to the root of $\mathcal{T}$. Since we have constructed $\mathcal{S}$ to preserve the ancestor-descendant relationships from $\mathcal{T}$, we will be able to update the sketches in $\mathcal{S}$ in a more clever way.

To better utilizing the structural relationship between the lower Cholesky factor and the sampling tree, for any $v \in \mathcal{S}$, we rewrite the sketches $\Phi_{\chi(v)} \mathcal{W}^\top h = \Phi_{\chi(v)} H^{-1/2} A^\top L^{-\top} h$ using the following notation:

Definition 6.22 (J_v, Z_v^*, y_v)**.** For each $v \in \mathcal{S}$, let

$$J_v \overset{\text{def}}{=} \Phi_{\chi(v)} H^{-1/2} A^\top, \quad Z_v^* \overset{\text{def}}{=} J_v \cdot L^{-\top}, \quad y_v \overset{\text{def}}{=} Z_v^* \cdot h = \Phi_{\chi(v)} \mathcal{W}^\top h.$$

At every node v, we will maintain J_v, and some variant of Z_v^* and y_v discussed later.

Let us first examine the sparsity pattern of J_v and Z_v^*:

Lemma 6.23 (Sparsity pattern of J_v)**.** *Let $v \in \mathcal{S}$, and suppose J_v satisfies (i) of Invariant 6.28. Let S be the non-zero column pattern of J_v, i.e. $S = \{j \in [d] \mid (J_v)_j \neq \mathbf{0}\}$. If $v \in \mathcal{S} \setminus \mathcal{T}$, then $S \subseteq \mathcal{P}^{\mathcal{T}}(\mathcal{T}(v))$. On the other hand, if $v \in \mathcal{T}$, then $S \subseteq \mathcal{D}^{\mathcal{T}}(\mathcal{T}(v)) \cup \mathcal{P}^{\mathcal{T}}(\mathcal{T}(v))$.*

Proof. First, note that for any $i \in [n]$, the non-zero column pattern of $\Phi_{\{i\}} H^{-1/2} A^\top$ is the non-zero pattern of column A_i. More generally, the non-zero column pattern of $J_v = \Phi_{\chi(v)} H^{-1/2} A^\top$ is given by the union of the non-zero pattern of columns A_j such that $j \in \chi(v)$ for any $v \in \mathcal{S}$.

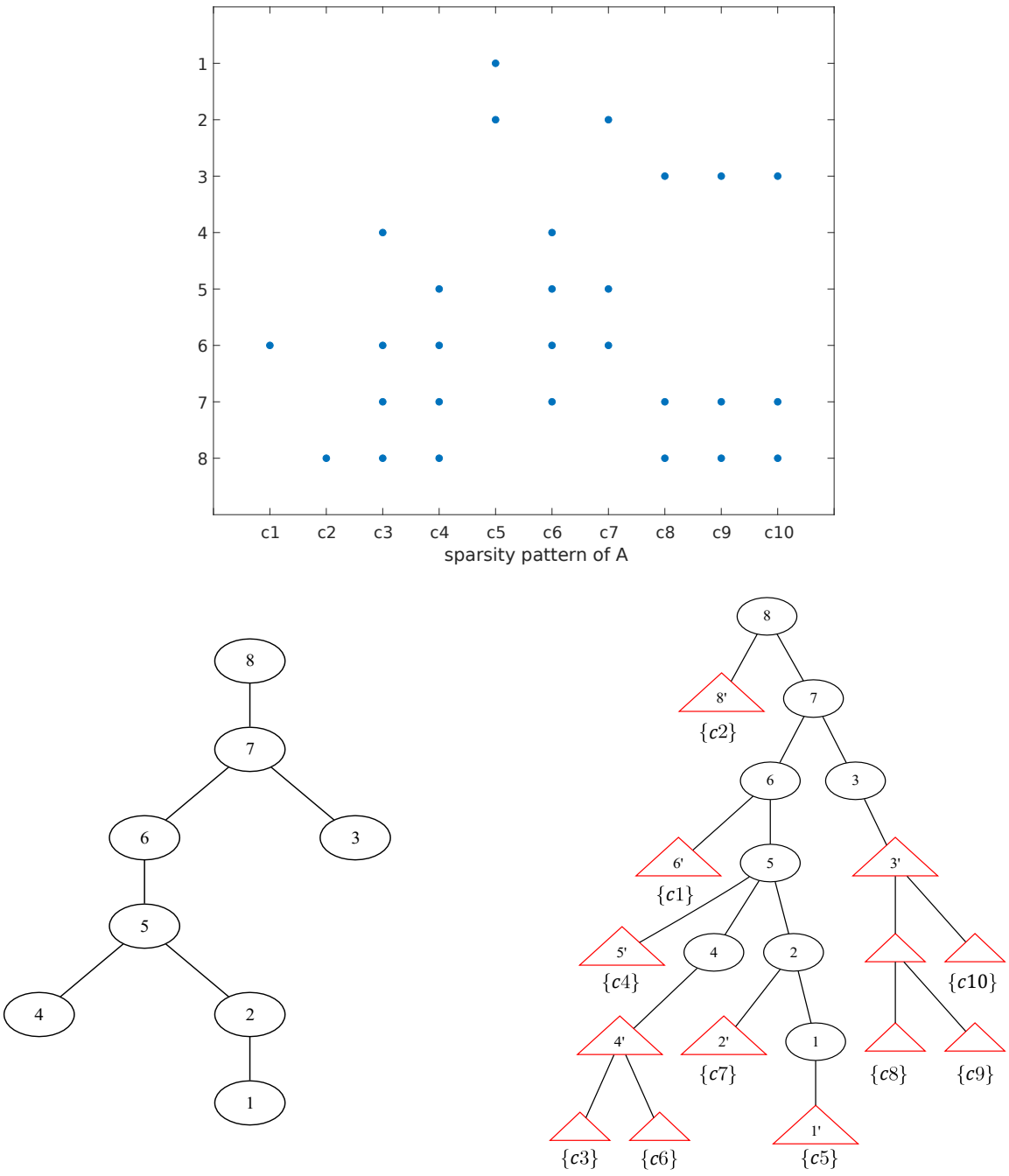

Figure 6.1: Example of simple sampling tree: The tree on the left is the elimination tree $\mathcal{T}$ for constraint matrix A, whose sparsity pattern is shown on the right ($n_i = 1$ for all i). The tree on the right is the simple sampling tree, where red triangles are newly added nodes, and the bracket under each leaf node denotes the column that the node will maintain in the data structure.

In the case that $v \in \mathcal{S} \setminus \mathcal{T}$, let $i = \mathcal{T}(v)$ be the $\mathcal{T}$-ancestor of v. By construction of $(\mathcal{S}, \chi)$, we have $\chi(v) \subseteq \{j \in [n] \mid \mathsf{low}^{\mathcal{T}}(A_j) = i\}$. Hence, the sparsity pattern of any column A_j with $j \in \chi(v)$ is a subset of $\mathcal{P}^{\mathcal{T}}(i)$ by Lemma 5.1. Since v is a descendant of i in $\mathcal{S}$, we have $\chi(v) \subset \chi(i)$ by property of χ. Therefore, $S \subseteq \mathcal{P}^{\mathcal{T}}(i)$, as required.

In the case that $v = i \in \mathcal{T}$, observe that as a consequence of Eq. (6.4), we have

$$\chi(i) = \{j \in [n] \mid \mathsf{low}^{\mathcal{T}}(A_j) = k, \text{ where } k \in \mathcal{D}^{\mathcal{T}}(i)\}.$$

By Lemma 5.1, for any $j \in \chi(i)$, the sparsity pattern of A_j is a subset of a path in $\mathcal{T}$ containing i, that is, it is contained in $\mathcal{D}^{\mathcal{T}}(i) \cup \mathcal{P}^{\mathcal{T}}(i)$. □

Lemma 6.24 (Sparsity pattern of Z_v^*)**.** *Let $v \in \mathcal{T}$, and let S be the non-zero pattern of the columns of Z_v, i.e. $S = \{i \in [d] \mid (Z_v)_i \neq \mathbf{0}\}$. Then, $S \subseteq \mathcal{D}^{\mathcal{T}}(v) \cup \mathcal{P}^{\mathcal{T}}(v)$.*

Proof. This directly follows by Lemmas 5.4 and 6.23. □

At this point, we have all the tools to answer queries for the sketch at a node $v \in \mathcal{S} \setminus \mathcal{T}$: Given J_v at $v \in \mathcal{S} \setminus \mathcal{T}$, the non-zero columns of J_v is a subset of $\mathcal{P}^{\mathcal{T}}(\mathcal{T}(v))$ by Lemma 6.23. Hence, we can compute the sketch $y_v^* = J_v \cdot L^{-\top} h$ in $\mathrm{poly}(\tau)$ time during a query. The sketch at a node $v \in \mathcal{T}$ needs to be maintained more carefully.

Updates to $\mathcal{W}$ via $\overline{x}$ causes a corresponding update to the Cholesky factor L. We will show later that if column j of L changes, then the sketches that change are at nodes of $\mathcal{S}$ in the subtree rooted at j, and the path from j to the root; we delay the updates of L at nodes on the path $\mathcal{P}^{\mathcal{T}}(j)$.

Definition 6.25 ($L[t], t_v, Z_v$)**.** Let $\{L[t]\}_{t \geq 0}$ be a list of Cholesky factors computed at different iterations during the maintenance, such that $L[\ell]$ is the Cholesky factor computed at iteration ℓ, for an internal iteration counter $\ell \geq 0$ that advances whenever L is updated.

At every node $v \in \mathcal{S} \cap \mathcal{T}$, we maintain a time stamp $t_v \geq 0$. Furthermore, we maintain a modified Z_v that depends on L from an earlier iteration given by t_v, that is,

$$Z_v = J_v \cdot L[t_v]^{-\top}.$$

Remark 6.26. For any $v \in \mathcal{T}$, note that Z_v and Z_v^* have the same non-zero pattern, as the non-zero pattern of L is constant throughout the algorithm.

Similarly, updates to h may cause sketches at many nodes of $\mathcal{S}$ to change. Again, we implement lazy updating for the part of the sketch $y_v = Z_v^* \cdot h$ involving h.

Definition 6.27 ($y_v^{\triangledown}$)**.** For $v \in \mathcal{T}$, let

$$y_v^{\triangledown} \stackrel{\text{def}}{=} Z_v \cdot (I - I_{\mathcal{P}^{\mathcal{T}}(v)})h.$$

At every node $v \in \mathcal{S} \cap \mathcal{T}$, we maintain $y_v^{\triangledown}$. When $Z_v = Z_v^*$, observe that we can write y_v as

$$y_v = Z_v^* \cdot I_{\mathcal{P}^{\mathcal{T}}(v)} h + y_v^{\triangledown}. \tag{6.5}$$

It follows that to obtain the latest sketch y_v at node v, we can update $Z_v \leftarrow Z_v^*$, update $y_v^{\triangledown}$ accordingly, and then compute y_v by Eq. (6.5), noting that the first term can be computed in $\mathrm{poly}(\tau)$ time given Z_v^* and h.

Now, we list the invariants our data structure maintains during the algorithm.

Invariant 6.28. *The variables maintained in the data structure* SimpleSketch*, as given in Algorithm 10, always preserve the following invariant before and after each function call:*

$$J_v = \Phi_{\chi(v)} H^{-1/2} A^\top \qquad v \in \mathcal{S} \tag{i}$$
$$Z_v = J_v \cdot L[t_v]^{-\top} \qquad v \in \mathcal{T} \tag{ii}$$
$$\mathbf{0} = (L[\ell] - L[t_v])_{\mathcal{D}^{\mathcal{T}}(v)\setminus\{v\}} \qquad v \in \mathcal{T} \tag{iii}$$
$$y_v^{\triangledown} = Z_v \cdot (I - I_{\mathcal{P}^{\mathcal{T}}(v)})h \qquad v \in \mathcal{T} \tag{iv}$$

where $H = \nabla^2\phi(\overline{x})$, L is the lower Cholesky factor such that $LL^\top = AH^{-1/2}A^\top$, and t_v is the time stamp of v.

Finally, suppose the current iteration counter is ℓ. the following lemma tells us how to compute the latest value $Z_v^* = J_v \cdot L[\ell]^{-\top}$, using $Z_v = J_v \cdot L[t_v]^{-\top}$ maintained by the data structure:

Lemma 6.29. *Suppose Invariant 6.28 is satisfied for node $v \in \mathcal{T}$, then*

$$Z_v^* = Z_v - \left(L[\ell]^{-1}(L[\ell] - L[t_v])_{\mathcal{P}^{\mathcal{T}}(v)} \cdot Z_v^\top\right)^\top.$$

Proof. Let us denote $\Delta L = L[\ell] - L[t_v]$. Then $(Z^*)^\top = (L[\ell] + \Delta L)^{-1} J_v^\top$, and we want to find ΔZ such that

$$Z_v^\top + (\Delta Z)^\top = (Z^*)^\top = (L[t_v] + \Delta L)^{-1} J_v^\top.$$

We have

$$\begin{aligned}(L[t_v] + \Delta L)(Z_v^\top + (\Delta Z)^\top) = J_v^\top &= L[\ell] Z_v^\top \\ (\Delta Z)^\top &= -(L[t_v] + \Delta L)^{-1}(\Delta L) Z_v^\top \\ Z_v^* - Z_v = \Delta Z &= -\left(L[\ell]^{-1}(L[\ell] - L[t_v]) Z_v^\top\right)^\top.\end{aligned}$$

We split ΔL into three parts:

$$\Delta L = (I_{\mathcal{P}^{\mathcal{T}}(v)} + I_{\mathcal{D}^{\mathcal{T}}(v)\setminus\{v\}} + I_{\mathcal{T}\setminus(\mathcal{D}^{\mathcal{T}}(v)\cup\mathcal{P}^{\mathcal{T}}(v))})\Delta L.$$

By Lemma 6.24, the non-zero columns of $Z_v = J_v \cdot L[t_v]^{-\top}$ is a subset of $\mathcal{D}^{\mathcal{T}}(v) \cup \mathcal{P}^{\mathcal{T}}(v)$. Hence, $I_{\mathcal{T}\setminus(\mathcal{D}^{\mathcal{T}}(v)\cup\mathcal{P}^{\mathcal{T}}(v))} \cdot Z_v^\top = \mathbf{0}$. By (iii) of Invariant 6.28, $(L[\ell] - L[t_v]) \cdot I_{\mathcal{D}^{\mathcal{T}}(v)\setminus\{v\}} = \mathbf{0}$, implying that

$$L[\ell]^{-1}(L[\ell] - L[t_v]) Z_v^\top = L[\ell]^{-1}(L[\ell] - L[t_v])_{\mathcal{P}^{\mathcal{T}}(v)} \cdot Z_v^\top.$$

□

Now we are ready to prove the correctness and runtime of each function in the data structure. The correctness of the overall maintenance data structure then follows immediately from the invariants.

Lemma 6.30 (Initialize)**.** *Given initial $\overline{x}$ and h, the JL matrix $\Phi \in \mathbb{R}^{r\times n}$, and the elimination tree $\mathcal{T}$ with height τ, the data structure* SimpleSketch *initializes the sketches in the sampling tree in $O(n \cdot \tau \cdot \eta \cdot r)$ time. Moreover, the internal state of the data structure satisfies Invariant 6.28 after initialization.*

Algorithm 10 Simple Multiscale Representation Sketching Data Structure – Initialize and Query

datastructure SIMPLESKETCH
private : members
  $\Phi \in \mathbb{R}^{r \times n}$ ▷ JL matrix
  sampling tree $(\mathcal{S}, \chi)$ ▷ constructed according to Section 6.5.1
  elimination tree $\mathcal{T}$
  $\ell \in \mathbb{N}$ ▷ iteration counter
  $h \in \mathbb{R}^d$
  $\overline{x} \in \mathbb{R}^n$ ▷ $\mathcal{W}$ given implicitly by $\overline{x}$
  $H \in \mathbb{R}^{n \times n}$ ▷ Hessian $H = \nabla^2 \phi(\overline{x})$
  LIST $\{L[t] \in \mathbb{R}^{d \times d}\}_{t \geq 0}$ ▷ sequence of Cholesky factors at various iterations t
  LIST $\{J_v \in \mathbb{R}^{r \times d}\}_{v \in \mathcal{T}}$ ▷ $J_v = \Phi_{\chi(v)} H^{-1/2} A^\top$
  LIST $\{Z_v \in \mathbb{R}^{r \times d}\}_{v \in \mathcal{S}}$ ▷ $Z_v = J_v \cdot L[t_v]^{-\top}$
  LIST $\{y_v^\nabla \in \mathbb{R}^r\}_{v \in \mathcal{T}}$ ▷ $y_v^\nabla = Z_v \cdot (I - I_{\mathcal{P}\mathcal{T}(v)})h$
  LIST $\{t_v \in \mathbb{N}\}_{v \in \mathcal{T}}$ ▷ t_v is the time of the last update at a node v
end members
procedure INITIALIZE($\mathcal{S}, \chi, \Phi \in \mathbb{R}^{r \times n}, \overline{x} \in \mathbb{R}^n, h \in \mathbb{R}^d$) ▷ Lemma 6.30
  $(\mathcal{S}, \chi) \leftarrow (\mathcal{S}, \chi)$
  $\Phi \leftarrow \Phi$
  $\ell \leftarrow 0, h \leftarrow h$
  Compute $H \leftarrow \nabla^2 \phi(\overline{x})$
  Find the lower Cholesky factor $L[\ell]$ of $AH^{-1}A^\top$
  for all $v \in \mathcal{S}$ **do**
    $J_v \leftarrow \Phi_{\chi(v)} H^{-1/2} A^\top$ ▷ compute J_v for all $v \in \mathcal{S}$
  end for
  for all $v \in \mathcal{T}$ **do** ▷ these nodes store additional partial computations
    $Z_v \leftarrow J_v \cdot L[\ell]^{-\top}$
    $y_v^\nabla \leftarrow Z_v \cdot (I - I_{\mathcal{P}\mathcal{T}(v)})h$
    $t_v \leftarrow \ell$ ▷ record that Z_v and y_v^∇ were last updated at time ℓ
  end for
end procedure
procedure QUERY($v \in \mathcal{S}$) ▷ Lemma 6.32
  if $v \in \mathcal{S} \setminus \mathcal{T}$ **then** ▷ directly compute and return the value of sketch
    return $J_v \cdot L[\ell]^{-\top} h$
  end if
  ▷ for $v \in \mathcal{T}$, we make use of existing partial computations
  $\Delta L \leftarrow (L[\ell] - L[t_v])_{\mathcal{P}\mathcal{T}(v)}$
  $Z_v \leftarrow Z_v - (L[\ell]^{-1} \cdot \Delta L \cdot Z_v^\top)^\top$ ▷ Update Z_v to correspond to $L[\ell]$, that is, $Z_v = Z_v^*$
  $y_v^\nabla \leftarrow Z_v \cdot (I - I_{\mathcal{P}\mathcal{T}(v)}) \cdot h$
  ▷ Z_v and y_v^∇ now correspond to the latest $L[\ell]$, so we update the time stamp of v
  $t_v \leftarrow \ell$
  return $Z_v \cdot I_{\mathcal{P}\mathcal{T}(v)} \cdot h + y_v^\nabla$
end procedure

Algorithm 11 Simple Multiscale Representation Sketching Data Structure – Updates

datastructure SIMPLESKETCH

procedure UPDATE($\overline{x}^{\text{new}} \in \mathbb{R}^n$, $h^{\text{new}} \in \mathbb{R}^n$) ▷ Lemma 6.31
  for $i \in [n]$ where $\overline{x}_i^{\text{new}} \neq \overline{x}_i$ **do**
    UPDATECOORDINATE($\overline{x}^{\text{new}}, i$) ▷ break up the update into single-coordinate updates
  end for
  for all $h_i^{\text{new}} \neq h_i$ **do**
    for all $v \in \mathcal{P}^{\mathcal{T}}(i)$ **do**
      $y_v^{\triangledown} \leftarrow y_v^{\triangledown} + Z_v \cdot I_{\{i\}} \cdot (h^{\text{new}} - h)$
    end for
  end for
  $h \leftarrow h^{\text{new}}$
end procedure
procedure UPDATECOORDINATE($\overline{x}^{\text{new}} \in \mathbb{R}^n, i \in [n]$) ▷ Lemma 6.33
  $\overline{x}_i \leftarrow \overline{x}_i^{\text{new}}$
  $H^{\text{new}} = \nabla^2 \phi(\overline{x})$
  $\ell \leftarrow \ell + 1$ ▷ increment iteration before computing a new Cholesky factor
  Find lower Cholesky factor $L[\ell]$ of $A(H^{\text{new}})^{-1}A^\top$
  UPDATE$L(\mathcal{P}^{\mathcal{T}}(\mathsf{low}^{\mathcal{T}}(A_i)))$
  UPDATE$H(H_i^{\text{new}}, i)$
end procedure
procedure UPDATE$L(S \subseteq \mathcal{T})$ ▷ S is a path in $\mathcal{T}$, Lemma 6.35
  for all $v \in S$ **do**
    ▷ We update Z_v to Z_v^* in two steps: first from $L[t_v]$ to $L[\ell-1]$, then from $L[\ell-1]$ to $L[\ell]$
    $Z_v \leftarrow Z_v - \left(L[\ell-1]^{-1} \cdot (L[\ell-1] - L[t_v])_{\mathcal{P}^{\mathcal{T}}(v)} \cdot Z_v^\top\right)^\top$
    $Z_v \leftarrow Z_v - (L[\ell]^{-1} \cdot (L[\ell] - L[\ell-1]) \cdot Z_v^\top)^\top$
    $y_v^{\triangledown} \leftarrow Z_v \cdot (I - I_{\mathcal{P}^{\mathcal{T}}(v)}) \cdot h$
    $t_v \leftarrow \ell$
  end for
end procedure
procedure UPDATE$H(H^{\text{new}})$ ▷ Lemma 6.34
  $\Delta H = H^{\text{new}} - H$
  for all $i \in [n]$ such that $(\Delta H)_i \neq \mathbf{0}$ **do**
    Find v such that $\chi(v) = \{i\}$
    for all $u \in \mathcal{P}^{\mathcal{S}}(v)$ **do**
      $J_v \leftarrow \Phi_{\chi(v)}(H + \Delta H \cdot I_{\{i\}})^{-1/2} A^\top$
      if $u \in \mathcal{T}$ **then**
        $Z_v \leftarrow J_v \cdot L[t_v]^{-\top}$
        $y_v^{\triangledown} \leftarrow Z_v \cdot (I - I_{\mathcal{P}^{\mathcal{T}}(v)}) \cdot h$
      end if
    end for
  end for
  $H \leftarrow H^{\text{new}}$
end procedure

Proof. The correctness directly follows by the setup of Invariant 6.28.

Runtime: By Corollary 5.8, we can find $L[\ell]$ in time $O(n\tau^2)$. For any non-leaf node $v \in \mathcal{S}$, we note that $J_v = \sum_{u \text{ a child of } v} J_u$. For a leaf node $v \in \mathcal{S}$, we have $\chi(v) = \{i\}$ for some $i \in [n]$, so we can compute J_v in time $O(\tau \cdot r)$ by Lemma 5.1. Then, we can compute all J_v for non-leaf nodes by summing the values of its children, iteratively up the tree. Since the height of tree $\mathcal{S}$ is η, we can compute J_v for all $v \in \mathcal{S}$ in time $O(|V(\mathcal{S})| \cdot \tau \cdot \eta \cdot r)$.

For $v \in \mathcal{T}$, by Eq. (6.4), we have

$$\begin{aligned} Z_v &= \left(\Phi_{\{i \mid \mathsf{low}^{\mathcal{T}}(A_i) = v\}} + \sum_{\text{child } u \text{ of } v \text{ in } \mathcal{T}} \Phi_{\chi(u)} \right) H^{-1/2} A^\top L[\ell]^{-\top} \\ &= \Phi_{\{i \mid \mathsf{low}^{\mathcal{T}}(A_i) = v\}} H^{-1/2} A^\top L^{-\top} + \sum_{\text{child } u \text{ of } v \text{ in } \mathcal{T}} J_v \cdot L[\ell]^{-\top} \end{aligned}$$

Thus, for each $v \in \mathcal{T}$, we only need to compute the term $\Phi_{\{i \mid \mathsf{low}^{\mathcal{T}}(A_i) = v\}} H^{-1/2} A^\top L^{-\top}$. Since the non-zero columns of $\Phi_{\{i \mid \mathsf{low}^{\mathcal{T}}(A_i) = v\}} H^{-1/2} A^\top$ lie on $\mathcal{P}^{\mathcal{T}}(v)$, this term has $O(\tau \cdot r)$ many non-zero entries, and we can compute it in $O(\tau^2 \cdot r)$ time by Lemma 5.4. Again, by iteratively computing Z_v up the tree, we can compute Z_v for all $v \in \mathcal{T}$ in $O(|\mathcal{T}| \cdot \tau^2 \cdot r)$ time.

Because h is explicitly given, we can compute $y_v^{\triangledown} = Z_v \cdot (I - I_{\mathcal{P}^{\mathcal{T}}(v)}) \cdot h$ in $\mathrm{nnz}(Z_v)$ time for each $v \in \mathcal{S}$; then we can compute $y_v^{\triangledown}$ for all $v \in \mathcal{T}$ in $O(|\mathcal{T}| \cdot \tau^2 \cdot r)$ time.

Combined with the fact $\tau \leqslant \eta$ and $|\mathcal{T}| = d \leqslant n$, the total time is bounded by $O(n \cdot \tau \cdot \eta \cdot r)$. □

Lemma 6.31 (Update)**.** *Suppose the current state of data structure satisfied the Invariant 6.28. Given $\mathcal{W}^{\mathrm{new}}$ implicitly by $\overline{x}^{\mathrm{new}}$, and h^{new}, the function* UPDATE *of* SimpleSketch *updates the sketches in $\mathcal{S}$ implicitly in time $O(\|\overline{x}^{\mathrm{new}} - \overline{x}\|_0 \cdot \tau^2 \cdot \eta \cdot r) + O(\|h^{\mathrm{new}} - h\|_0 \cdot \tau \cdot r)$ Moreover, the function also updates the internal states correspondingly so that Invariant 6.28 is still preserved.*

Proof. Note that we can process the updates to $\mathcal{W}$ and h consecutively; hence the runtime is a sum of the runtimes of the two steps.

To update $\overline{x}$ to $\overline{x}^{\mathrm{new}}$ and thus $\mathcal{W}$ to $\mathcal{W}^{\mathrm{new}}$, we again view it as a sequence of updates, where each update correspond to a single coordinate change in $\overline{x}$, processed by the helper function Update-Coordinate. The associated proof is given in Lemma 6.33.

Similarly, we update h to h^{new} by a sequence of single-coordinate updates. By Lemma 6.24, the nonzero columns of $Z_v \cdot (I - I_{\mathcal{P}^{\mathcal{T}}(v)})$ lies on $\mathcal{D}^{\mathcal{T}}(v)$. Therefore, when h_i changes, $y_v^{\triangledown}$ changes only if $i \in \mathcal{D}^{\mathcal{T}}(v)$, so it suffices to update only the sketches at nodes v where $v \in \mathcal{P}^{\mathcal{T}}(i)$, and the update is given by $Z_v \cdot I_{\{i\}} \cdot (h^{\mathrm{new}} - h)$, computable in $O(r)$ time. Thus, updating a coordinate h_i takes $|\mathcal{P}^{\mathcal{T}}(v)| O(r) = O(\tau \cdot r)$ time. Summing over all changed coordinates, we can update h in $O(\|h^{\mathrm{new}} - h\|_0 \cdot \tau \cdot r)$ time. □

Lemma 6.32 (Query)**.** *Suppose Invariant 6.28 is satisfied. The function Query(v) of* SimpleSketch *outputs $\Phi_{\chi(v)} \mathcal{W}^\top h$ in $O(\tau^2 r)$ time. Moreover, Invariant 6.28 is preserved after the function call.*

Proof. **Correctness:** For the case $v \in \mathcal{S} \setminus \mathcal{T}$, the correctness directly follows by definition of J_v. Now, we consider the case that $v \in \mathcal{T}$. The invariant maintenance of moving t_v to ℓ directly follows

by Lemma 6.29. To output $\Phi_{\chi(v)}\mathcal{W}^\top h = y_v$, the function computes the expression as given by Eq. (6.5).

Runtime: For the case $v \in \mathcal{S} \setminus \mathcal{T}$, the non-zero columns of J_v lies on $\mathcal{P}^{\mathcal{T}}(\mathcal{T}(v))$ by Lemma 6.23. By Lemma 5.4, the term $J_v \cdot L[\ell]^{-\top}$ has $O(\tau \cdot r)$ many nonzero entries and can be computed in time $O(\tau^2 r)$ time. Thus, we can compute $J_v \cdot L[\ell]^{-\top} h$ in time $O(\tau^2 \cdot r)$.

For the case $v \in \mathcal{T}$, we first note that we can find ΔL in $O(\tau^2)$ time since $|\mathcal{P}^{\mathcal{T}}(v)| \leqslant \tau$, and the column sparsity of L is also bounded by τ by Lemma 5.2. By the sparsity pattern of ΔL, we can compute $(\Delta L) \cdot (Z \triangledown_v)^\top$ in $O(\tau^2 r)$ time. By sparsity pattern of L, we can update $Z_v \leftarrow Z_v^*$ and $y_v^\triangledown$ in $O(\tau^2 r)$ time by solving $O(r)$ lower triangular system using Lemma 5.4. In Eq. (6.5), we can compute $Z_v \cdot I_{\mathcal{P}^{\mathcal{T}}(v)} \cdot h$ in $O(\tau \cdot r)$ time since $|\mathcal{P}^{\mathcal{T}}(v)| \leqslant \tau$. Hence, the function takes $O(\tau^2 \cdot r)$ time in total. □

Lemma 6.33 (UpdateCoordinate)**.** *Suppose the current state of the data structure satisfies Invariant 6.28. The function UpdateCoordinate of SimpleSketch updates the implicit representation of $\mathcal{W}$ by updating the i-th coordinate of $\overline{x}$ from $\overline{x}_i$ to $\overline{x}_i^{\text{new}}$ in $O(\tau^2 \cdot \eta \cdot r)$ time. Moreover, the function UpdateCoordinate also updates the internal states correspondingly such that Invariant 6.28 is preserved after the function call.*

Proof. **Correctness:** First, we show that for the change on L, it suffices to updates all nodes on the path $\mathcal{P}^{\mathcal{T}}(\mathsf{low}^{\mathcal{T}}(A_i))$. We note that only (iii) of Invariant 6.28 depends on the value of $L[\ell]$, so we need to update the sketch only if $(L[\ell+1] - L[t_v]) \cdot I_{\mathcal{D}^{\mathcal{T}}(v)\setminus\{v\}} \neq \mathbf{0}$. Since the data structure satisfies the invariants for ℓ, we have $(L[\ell] - L[t_v]) \cdot I_{\mathcal{D}^{\mathcal{T}}(v)\setminus\{v\}} = \mathbf{0}$ for all v. Therefore, we need to update the sketch only if

$$(L[\ell+1] - L[\ell]) \cdot I_{\mathcal{D}^{\mathcal{T}}(v)\setminus\{v\}} \neq \mathbf{0}.$$

We use L^{new} to denote $L[\ell+1]$ and use L to denote $L[\ell]$, where $L^{\text{new}}(L^{\text{new}})^\top = AH^{-1}A^\top + cA_iA_i^\top$ for some c and i. Let $\Delta L = L^{\text{new}} - L$. By Lemma 5.9, the non-zero columns of ΔL lies on $\mathcal{P}^{\mathcal{T}}(\mathsf{low}^{\mathcal{T}}(A_i))$. We denote $\mathsf{low}^{\mathcal{T}}(A_i)$ by u, and rewrite ΔL as $\sum_{w \in \mathcal{P}^{\mathcal{T}}(u)} (\Delta L)_w e_w^\top$. For each $w \in \mathcal{P}^{\mathcal{T}}(u)$, we note that $(\Delta L)_w e_w^\top \cdot I_{\mathcal{D}^{\mathcal{T}}(v)\setminus\{v\}} \neq \mathbf{0}$ only if $v \in \mathcal{P}^{\mathcal{T}}(w) \setminus w$. Hence, it suffices to update

$$\bigcup_{w \in \mathcal{P}^{\mathcal{T}}(u)} \mathcal{P}^{\mathcal{T}}(w) \setminus w \subseteq \mathcal{P}^{\mathcal{T}}(u).$$

The function then uses two helper functions UpdateL and UpdateH, whose correctness and runtime are given in Lemma 6.34 and Lemma 6.35.

Runtime: By Lemma 5.10, we can find L^{new} in $O(\tau^2)$ time and L changes in τ columns. By Lemma 6.34, H changes in coordinate i, so we can update it in $O(\tau^2 \cdot \eta \cdot r)$ time. Since L changes in τ columns, we can update L in the data structure in $O(\tau^3 r)$ time by Lemma 6.35. Hence, the function takes $O(\tau^2 \cdot \eta \cdot r)$ time in total. □

Lemma 6.34 (UpdateH)**.** *Suppose Invariant 6.28 is satisfied. UpdateH updates H to H^{new}, and implicitly adjusts the sketches in $\mathcal{S}$ to preserve Invariant 6.28 in $O(\text{nnz}(\Delta H) \cdot \tau^2 \cdot \eta \cdot r)$ time.*

Proof. **Correctness:** We observe that Z_v changes only if $I_{\chi(v)} \cdot \Delta H \neq \mathbf{0}$. Suppose the i-th column of H changes, and let v be the node of $\mathcal{S}$ with $\chi(v) = \{i\}$. Then observe that for a change in H_i, it suffices to update $\mathcal{P}^{\mathcal{S}}(v)$.

Runtime: For a change in H_i, let $\widetilde{H} \stackrel{\text{def}}{=} (H + \Delta H \cdot I_{\{i\}})^{-1/2} - H^{-1/2}$. Then we can find $\Phi_{\chi(v)}\widetilde{H}A^\top$ by computing an outer product of a column of Φ with row of $A^\top$, which takes $O(\tau \cdot r)$ time by the sparsity pattern of A (Lemma 5.1). Then for a node v, we can update Z_v by compute $\Phi_{\chi(v)}\widetilde{H}A^\top L[t_v]^{-\top}$, which takes $O(\tau^2 \cdot r)$ time by Lemma 5.4. We can then update y_v^∇ in $O(\tau^2 \cdot r)$ time. As height$(\mathcal{S}) = \eta$, and we only update along a path to the root, this function takes $O(\tau^2 \cdot \eta \cdot r)$ time for the update to H_i. □

Lemma 6.35 (UpdateL). *Given a set $S \subset V(\mathcal{T})$, the function UpdateL updates t_v to the latest time at each $v \in S$, and adjusts the implicit representation of the sketch at v to preserve Invariant 6.28. If the number of non-zero columns of ΔL is bounded by $O(\tau)$, then the function takes $O(|S| \cdot \tau^2 \cdot r)$ time.*

Proof. **Correctness:** The correctness directly follows by Lemma 6.29.

Runtime: By the sparsity pattern of L (Lemma 5.2), we can compute $(L[\ell] - L[t_v]) \cdot I_{\mathcal{P}^{\mathcal{T}}(\mathcal{T}(v))} \cdot (Z_v)^\top$ and $\Delta L \cdot (Z_v)^\top$ in $O(\tau^2 r)$ time, and the column sparsity pattern of the result is on a path in $\mathcal{T}$. Then, we can update Z_v and y_v^∇ in $O(\tau^2 r)$ time by solving $O(r)$ many lower triangular systems using Lemma 5.4. Hence, the total time is bounded by $O(|S| \cdot \tau^2 r)$. □

6.6 Sketching the Multiscale Representation via Balanced Sampling Tree

	Simple Sampling Tree	Balanced Sampling Tree
Sampling tree height	$\widetilde{O}(\tau)$	$\widetilde{O}(1)$
JL dimension	$\widetilde{O}(\tau^2)$	$\widetilde{O}(1)$
Initialization time	$\widetilde{O}(n\tau^4)$	$\widetilde{O}(n\tau^2)$
Update $\mathcal{W}$ time	$\widetilde{O}(\|\overline{x}^{\text{new}} - \overline{x}\|_0 \cdot \tau^5)$	$\widetilde{O}(\|\overline{x}^{\text{new}} - \overline{x}\|_0 \cdot \tau^2)$
Update h time	$\widetilde{O}(\|h^{\text{new}} - h\|_0 \cdot \tau^3)$	$\widetilde{O}(\|h^{\text{new}} - h\|_0)$
Query time	$\widetilde{O}(\tau^4)$	$\widetilde{O}(\tau^2)$
Query time × tree height	$\widetilde{O}(\tau^5)$	$\widetilde{O}(\tau^2)$

Table 2: Comparison between two sampling trees.

Combining the simple sampling tree data structure with our IPM algorithm will give us a $\widetilde{O}(n\tau^5 \log(1/\varepsilon))$ algorithm for solving LPs. To make our algorithm competitive when τ is large, we demonstrate how to further speed up (Algorithms 10 and 11) to $\widetilde{O}(\tau^2)$ per step in this section. More specifically, we have the following theorem:

Theorem 6.36. *Given the constraint matrix A, its elimination tree $\mathcal{T}$ with height τ, a JL matrix $\Phi \in \mathbb{R}^{r \times n}$, and a sampling tree $(\mathcal{S}, \chi)$ constructed as in Section 6.6.1 with height $O(\log n)$, the data structure BalancedSketch (Algorithms 12 and 13) maintains $\Phi_{\chi(v)}\mathcal{W}^\top h$ for each $v \in V(\mathcal{S})$ through the following operations:*

- Initialize$(\mathcal{S}, \chi, \Phi, \overline{x}, h)$*: Initializes the data structure in $O(n\tau^2 r \log n)$ time, so that each node $v \in \mathcal{S}$ maintains the sketch $\Phi_{\chi(v)}\mathcal{W}^\top h = \Phi_{\chi(v)} H_{\overline{x}}^{-1/2} A^\top L_{\overline{x}}^\top$.*

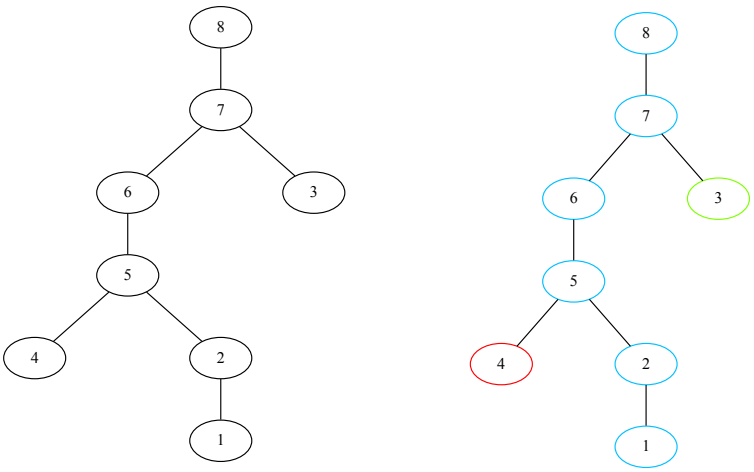

Figure 6.2: A rooted tree is given on the left, its heavy-light decomposition is shown on the right; the ordering is $[8, 7, 6, 5, 2, 1, 4, 3]$.

- $\textsc{Update}(\overline{x}^{\text{new}}, h^{\text{new}})$*: Updates all sketches in* $\mathcal{S}$ *implicitly to reflect* $\mathcal{W}$ *updating to* $\mathcal{W}^{\text{new}}$ *and* h *updating to* h^{new} *in* $O(\|\overline{x}^{\text{new}} - \overline{x}\|_0 \cdot \tau^2 r \log n) + O(\|h^{\text{new}} - h\|_0 \cdot r \log n)$ *time, where* $\mathcal{W}^{\text{new}}$ *is given implicitly by* $\overline{x}^{\text{new}}$*.*
- $\textsc{Query}(v)$*: Outputs* $\Phi_{\chi(v)}\mathcal{W}^\top h$ *in* $O(\tau^2 \cdot r)$ *time.*

We observe that the data structures in Section 6.5 has the same $\Omega(\tau^3)$ bottleneck for both updating the multiscale representation and sampling: The data structures are always operating on some path of the sampling tree $\mathcal{S}$, where we need to solve the lower triangular systems for each node on that path. In this section, we show how to obtain a balanced sampling tree with $O(\log n)$ height and thus speeds up each operation to $\widetilde{O}(\tau^2)$.

6.6.1 Balanced Sampling Tree Construction

As a first ingredient in our construction is the following lemma from Sleator and Tarjan's heavy-light decomposition.

Lemma 6.37 (Heavy-Light Decomposition [ST83])**.** *Given a rooted tree* $\mathcal{T}$*, there exists an ordering of vertices of* $V(\mathcal{T})$ *such that the path between any two vertices consists of at most* $O(\log n)$ *many contiguous subsequences of the ordering, and for any vertices* v*, the subtree rooted at* v *corresponds to a single contiguous subsequence of the order. Moreover, such an ordering can be found in* $O(n)$ *time.* □

First, we construct a sampling tree of $\mathbb{R}^d$, denoted by $(\mathcal{B}, \overline{\chi})$, as follows: We perform heavy-light decomposition on the elimination tree $\mathcal{T}$ with vertex set $[d]$. Let $\sigma_1, \ldots, \sigma_d$ denote the vertices ordered according to Lemma 6.37. Let $\mathcal{B}$ be a complete binary tree containing d leaves, where the i-th leaf is $\sigma_i \in [d]$. We set $\overline{\chi}(\sigma_i) = \{\sigma_i\}$. For each non-leaf node $v \in \mathcal{B}$, we let $\overline{\chi}(v) = \overline{\chi}(\text{left child of } v) \cup \overline{\chi}(\text{right child of } v)$. It is easy to check $(\mathcal{B}, \overline{\chi})$ is a sampling tree of $\mathbb{R}^d$ by Definition 6.13.

Now, we extend the sampling tree $(\mathcal{B}, \overline{\chi})$ on $\mathbb{R}^d$ to $\mathbb{R}^n$ to obtain the *balanced sampling tree* $(\mathcal{S}, \chi)$: At each leaf node $v \in \mathcal{B}$, we add a complete binary tree with vertex set

$$\{i \in [n] \mid \text{coordinate } i \text{ in } j\text{-th block and } \overline{\mathsf{low}}^{\mathcal{T}}(A_j) = \overline{\chi}(v)\},$$

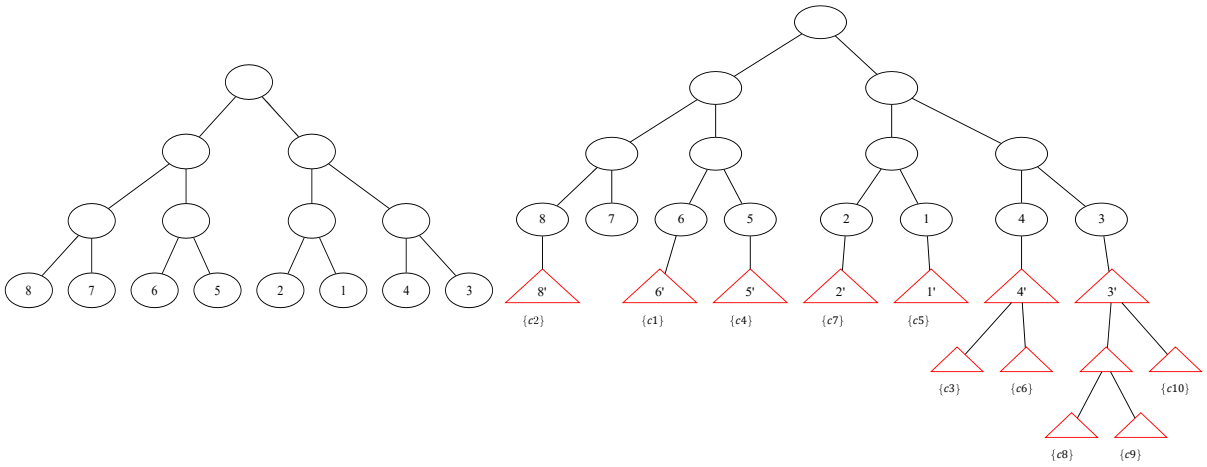

Figure 6.3: Examples of tree $\mathcal{B}$ (left) and $\mathcal{S}$ (right) constructed from the elimination tree in Fig. 6.2. The red triangle in the right graph denotes the newly added nodes in the tree. The bracket under each leaf node denotes the corresponding column maintained by that node.

where $\overline{\mathsf{low}}(A_j)$ is defined by

$$\overline{\mathsf{low}}(A_j) = \arg \max_{i \in \{i | A_{ij} \neq \mathbf{0}\}} \mathsf{depth}(i).$$

We denote this modified binary tree as $\mathcal{S}$. Then, each leaf node u of $\mathcal{S}$ corresponds to some $i \in [n]$, and we set $\chi(u) = \{i\}$. We can check that for any leaf node $v \in \mathcal{B}$,

$$\chi(v) = \{i \in [n] \mid \text{coordinate } i \text{ in } j\text{-th block and } \overline{\mathsf{low}}^{\mathcal{T}}(A_j) = \overline{\chi}(v)\}.$$

We define this $(\mathcal{S}, \chi)$ to be our *balanced sampling tree*. An example is shown in Fig. 6.3.

Since the height of $\mathcal{B}$ is $\log d$, and the height of the newly added binary trees are at most $\log n$, the height of $\mathcal{S}$ is $O(\log n)$.

Theorem 6.38. *Given an elimination tree $\mathcal{T}$ with height τ, the balanced sampling tree can be constructed in $O(n\tau + n \log n)$ time.*

Proof. By Lemma 6.37, we find the heavy-light-decomposition order in linear time. Since $\mathcal{B}$ is a binary tree on this order, we can construct $(\mathcal{B}, \overline{\chi})$ in $O(d \log d)$ time. For each block A_j where $j \in [m]$, we can find $\overline{\mathsf{low}}(A_j)$ in time $O(\tau)$ since $\mathrm{nnz}(A_j) = O(\tau)$ (Lemma 5.1). Hence, we can find $\overline{\mathsf{low}}(A_j)$ for all $j \in [m]$ in time $O(n\tau)$. This gives us $\chi(v)$ for all $v \in \mathcal{B}$. Finally, we can construct $(\mathcal{S}, \chi)$ in time $O(n \log n)$. Hence, we can construct $(\mathcal{S}, \chi)$ in $O(n\tau + n \log n)$ time. □

The balanced sampling tree does not preserve ancestor-descendant relationship of the vertices of $\mathcal{T}$. However, the following lemma about the Heavy-Light Decomposition will help us get something close.

Lemma 6.39. *Let the sequence $a_1, a_2, \ldots, a_n$ be the order produced by Heavy-Light Decomposition on tree $\mathcal{T}$. For any contiguous subsequence $a_l, a_{l+1}, \ldots, a_r$, we have*

$$\left| \left(\bigcup_{i \in [l,r]} \mathcal{P}^{\mathcal{T}}(a_i) \right) \cap \left(\bigcup_{i \in [1,n] \setminus [l,r]} \mathcal{P}^{\mathcal{T}}(a_i) \right) \right| \leqslant 2 \cdot \mathsf{height}(\mathcal{T}).$$

Proof. It suffices to show that for any four numbers l_1, l_2, r_1, r_2 where $l_1 \leqslant l_2 \leqslant r_2 \leqslant r_1$, we have $\mathcal{P}(a_{l_1}) \cap \mathcal{P}(a_{r_1}) \subseteq \mathcal{P}(a_{l_2}) \cap \mathcal{P}(a_{r_2})$. Indeed, when this is true, we have

$$\left|\left(\bigcup_{i\in[l,r]} \mathcal{P}(a_i)\right) \cap \left(\bigcup_{i\in[n]\setminus[l,r]} \mathcal{P}(a_i)\right)\right| \leqslant \left|\left(\bigcup_{i\in[1,l-1]} \mathcal{P}(a_i)\right) \cap \left(\bigcup_{i\in[l,r]} \mathcal{P}(a_i)\right)\right| + \left|\left(\bigcup_{i\in[l,r]} \mathcal{P}(a_i)\right) \cap \left(\bigcup_{i\in[r+1,n]} \mathcal{P}(a_i)\right)\right|,$$

and the two terms on the right-hand side can be bounded by $|\mathcal{P}(a_{l-1}) \cap \mathcal{P}(a_l)|$ and $|\mathcal{P}(a_r) \cap \mathcal{P}(a_{r+1})|$ respectively.

Note that $\mathcal{P}(x) \cap \mathcal{P}(y) = \mathcal{P}(\mathsf{LCA}(x, y))$, where $\mathsf{LCA}(x, y)$ denotes the *lowest common ancestor* of x and y in tree $\mathcal{T}$. Since the ordering $a_1, a_2, \ldots, a_n$ is produced by a depth-first traversal on $\mathcal{T}$, we observe that the subtree rooted at $\mathsf{LCA}(a_{l_1}, a_{r_1})$ contains a_{l_2}, a_{r_2}, since they are both discovered during the DFS after $a_{l_1}$and before a_{r_1}; consequently it contains $\mathsf{LCA}(a_{l_2}, a_{r_2})$. It also by definition contains $a_{l_1}$and a_{r_1}. Therefore, $\mathsf{LCA}(a_{l_1}, a_{r_1})$ is an ancestor of $\mathsf{LCA}(a_{l_2}, a_{r_2})$, and it follows that $\mathcal{P}(\mathsf{LCA}(a_{l_1}, a_{r_1})) \subseteq \mathcal{P}(\mathsf{LCA}(a_{l_2}, a_{r_2}))$. □

For the sampling tree $(\mathcal{B}, \overline{\chi})$, we have the following lemma:

Lemma 6.40 (See e.g. [dBCvKO08])**.** *Given the complete binary sampling tree $(\mathcal{B}, \overline{\chi})$, let $a_k = \overline{\chi}(v)$, where v is the k-th leaf of $\mathcal{B}$. For any contiguous subsequence $a_l, a_{l+1}, \ldots, a_r$ of the sequence $a_1, \ldots, a_d$, we can find a node set $S \subseteq \mathcal{B}$ of size $O(\log d)$ such that*

$$\bigcup_{u\in S} \overline{\chi}(u) = \{a_i \mid i \in [l, r]\}.$$

Moreover, this set S can be found in $O(\log d)$ time.

6.6.2 Data Structure for Sketching

As our balanced sampling tree $\mathcal{S}$ does not totally preserve the ancestor-descendant relationships in $\mathcal{T}$, we need a more complex maintenance scheme. We first observe that for any node $v \in \mathcal{S} \setminus \mathcal{B}$, the nonzero columns of $\Phi_{\chi(v)} H^{-1/2} A^\top$ lies on a path of $\mathcal{T}$. Therefore, given $J_v = \Phi_{\chi(v)} H^{-1/2} A^\top$, the term $\Phi_{\chi(v)} \mathcal{W}^\top$ can be computed in $\widetilde{O}(\tau^2)$ time. In the last section, for each node $v \in \mathcal{T}$, we delay L's updates in the columns that lie on $\mathcal{P}^{\mathcal{T}}(v)$. Here, we define its analogy on $\mathcal{B}$:

Definition 6.41 $(\Lambda(v), \overline{\Lambda}(v))$**.** Let $\Lambda : \mathcal{B} \to 2^{\mathcal{T}}$ be the function

$$\Lambda(v) \stackrel{\text{def}}{=} \left(\bigcup_{i\in\overline{\chi}(v)} \mathcal{P}^{\mathcal{T}}(a_i)\right) \cap \left(\bigcup_{i\in\mathcal{T}\setminus\overline{\chi}(v)} \mathcal{P}^{\mathcal{T}}(a_i)\right).$$

We also define $\overline{\Lambda}(v) : \mathcal{B} \to 2^{\mathcal{T}}$ to be the set of columns that are maintained up-to-date for each v:

$$\overline{\Lambda}(v) \stackrel{\text{def}}{=} \left(\bigcup_{i\in\overline{\chi}(v)} \mathcal{P}^{\mathcal{T}}(a_i)\right) \setminus \Lambda(v).$$

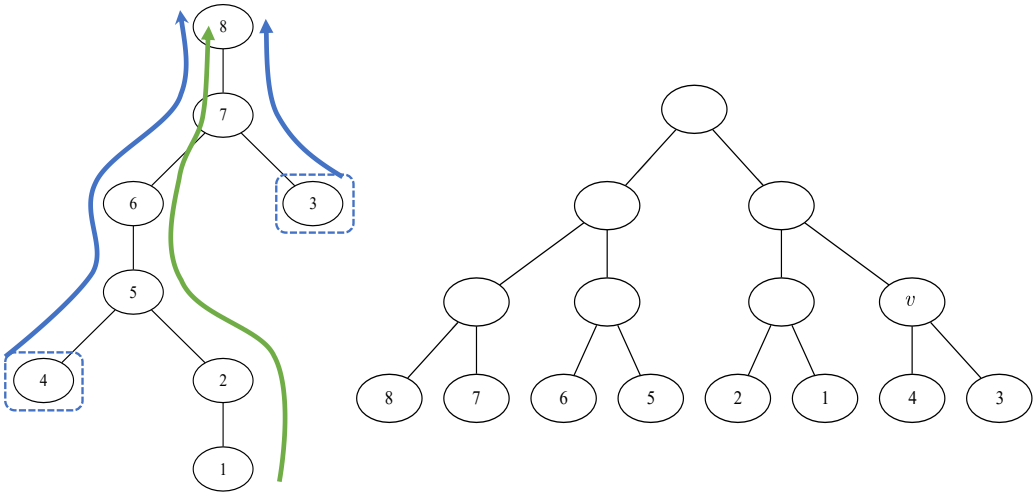

Figure 6.4: Example of $\Lambda(v)$ and $\overline{\Lambda}(v)$: on the left is $\mathcal{B}$, where $\Lambda(v) = \{5, 6, 7, 8\}$ is the set of nodes crossed by both the green path and blue path. $\overline{\Lambda}(v) = \{3, 4\}$ is the set of nodes contained by blue boxes.

Lemma 6.42. *For any nodes $u, v \in \mathcal{B}$, if $u \in \mathcal{P}^{\mathcal{B}}(v)$, then $\overline{\Lambda}(v) \subseteq \overline{\Lambda}(u)$.*

Proof. By definition of $\overline{\Lambda}(\cdot)$ and $\Lambda(\cdot)$, we have

$$\overline{\Lambda}(v) = \left(\bigcup_{i \in \overline{\chi}(v)} \mathcal{P}^{\mathcal{T}}(a_i) \right) \setminus \left(\bigcup_{i \in \mathcal{T} \setminus \overline{\chi}(v)} \mathcal{P}^{\mathcal{T}}(a_i) \right).$$

Since $(\mathcal{B}, \overline{\chi})$ is a sampling tree, we have $\overline{\chi}(v) \subseteq \overline{\chi}(u)$. Hence, we complete the proof by noting $\left(\bigcup_{i \in \overline{\chi}(v)} \mathcal{P}^{\mathcal{T}}(a_i)\right) \subseteq \left(\bigcup_{i \in \overline{\chi}(u)} \mathcal{P}^{\mathcal{T}}(a_i)\right)$ and $\left(\bigcup_{i \in \mathcal{T} \setminus \overline{\chi}(u)} \mathcal{P}^{\mathcal{T}}(a_i)\right) \subseteq \left(\bigcup_{i \in \mathcal{T} \setminus \overline{\chi}(v)} \mathcal{P}^{\mathcal{T}}(a_i)\right)$. □

Definition 6.43 ($\Lambda^{\clubsuit}$)**.** Finally, we define the function $\Lambda^{\clubsuit} : \mathcal{T} \to \mathcal{B}$.

$$\Lambda^{\clubsuit}(u) \stackrel{\text{def}}{=} \mathsf{low}^{\mathcal{B}}(\{v \in \mathcal{B} \mid u \in \overline{\Lambda}(v)\}).$$

Equivalently, we have

$$\Lambda^{\clubsuit}(u) = \mathsf{low}^{\mathcal{B}}(\{v \in \mathcal{B} \mid \mathcal{D}^{\mathcal{T}}(u) \subseteq \overline{\chi}(v)\}).$$

Using this equivalent definition, we show that $\Lambda^{\clubsuit}(\cdot)$ is well-defined.

Lemma 6.44. *For any $u \in \mathcal{T}$, the set $\{v \in \mathcal{B} \mid \mathcal{D}^{\mathcal{T}}(u) \subseteq \overline{\chi}(v)\}$ is a path on $\mathcal{B}$.*

Proof. Recall that $\{\overline{\chi}(v) \mid v \in \mathcal{B} \text{ and } \mathsf{depth}(v) = k\}$ forms a partition of $\mathcal{T}$ for any $k \leqslant \text{height}(\mathcal{B})$. Then, for any $k \leqslant \text{height}(\mathcal{B})$, there is at most one node satisfying both $\mathcal{D}^{\mathcal{T}}(u) \subseteq \overline{\chi}(v)$ and $\mathsf{depth}(v) = k$. We complete the proof by note that if $\mathcal{D}^{\mathcal{T}}(u) \subseteq \overline{\chi}(v)$, then $\mathcal{D}^{\mathcal{T}}(u) \subseteq \overline{\chi}(w)$ for any $w \in \mathcal{P}^{\mathcal{B}}(v)$ since $(\mathcal{B}, \overline{\chi})$ is a sampling tree. □

Now, we show the equivalence of two definition:

Lemma 6.45. *For any $u \in \mathcal{T}$ and $v \in \mathcal{B}$, we have $u \in \overline{\Lambda}(v)$ if and only if $\mathcal{D}^{\mathcal{T}}(u) \subseteq \overline{\chi}(v)$.*

Proof. The only if direction: Suppose $u \in \overline{\Lambda}(v)$ and there is a $w \in \mathcal{D}^{\mathcal{T}}(u)$ but $w \notin \overline{\chi}(v)$. We note that $w \notin \overline{\chi}(v)$ implies $w \in \mathcal{T} \setminus \overline{\chi}(v)$. Then, we have $u \in \Lambda(v)$ since $u \in \mathcal{P}^{\mathcal{T}}(w)$. This contradicts with our assumption that $u \in \overline{\Lambda}(v)$.

The if direction: Since $\mathcal{D}^{\mathcal{T}}(u) \subseteq \overline{\chi}(v)$, we have $u \in \overline{\chi}(v) \subseteq \left(\bigcup_{i \in \overline{\chi}(v)} \mathcal{P}^{\mathcal{T}}(a_i)\right)$. Then, it suffices to show $u \notin \Lambda(v)$. Suppose $u \in \Lambda(v)$, then there is a node $w \in \mathcal{D}^{\mathcal{T}}(v)$ such that $w \notin \overline{\chi}(v)$. This contradicts with our assumption that $\mathcal{D}^{\mathcal{T}}(u) \subseteq \overline{\chi}(v)$. □

Intuitively, for each node $v \in \mathcal{B}$, we need to maintain the union of paths on the interval $[l_v, r_v] = \overline{\chi}(v)$, i.e. $\bigcup_{i \in [l_v, r_v]} \mathcal{P}^{\mathcal{T}}(a_i)$. The set $\Lambda(v)$ to denote the set of nodes in $\mathcal{T}$ shared by other nodes the same level of binary tree $\mathcal{B}$. Lemma 6.39 shows that $|\Lambda(v)| = O(\tau)$. Hence, we never maintain them exactly in the sampling tre, but rather compute them as needed. On the other hand, for each node $v \in \mathcal{B}$, we maintain the node $u \in \mathcal{T}$ exactly only if $a_i \in \mathcal{D}^{\mathcal{T}}(u)$ for $i \in [l_v, r_v]$. Thus, for each node $v \in \mathcal{T}$, it is only been explicitly maintained on a path of $\mathcal{B}$, and $\Lambda^{\clubsuit}(v)$ denotes the lower end of that path. In particular, we have the following lemma about $\Lambda^{\clubsuit}(\cdot)$:

Lemma 6.46. *Let $u, v \in \mathcal{T}$. If $u \in \mathcal{P}^{\mathcal{T}}(v)$, then $\Lambda^{\clubsuit}(u) \in \mathcal{P}^{\mathcal{B}}(\Lambda^{\clubsuit}(v))$.*

Proof. We note that $\mathcal{D}^{\mathcal{T}}(u) \subseteq \overline{\chi}(w)$ implies $\mathcal{D}^{\mathcal{T}}(v) \subseteq \overline{\chi}(w)$ for any $w \in \mathcal{B}$. Then, we have $\{w \in \mathcal{B} \mid \mathcal{D}^{\mathcal{T}}(u) \subseteq \overline{\chi}(w)\} \subseteq \{w \in \mathcal{B} \mid \mathcal{D}^{\mathcal{T}}(v) \subseteq \overline{\chi}(w)\}$. Hence, by the equivalent definition of $\Lambda^{\clubsuit}$ and Lemma 6.44, we have $\Lambda^{\clubsuit}(u) \in \mathcal{P}^{\mathcal{B}}(\Lambda^{\clubsuit}(v))$. □

Similar to the proof of Section 6.5, ideally, we want to maintain

$$Z_v^* \stackrel{\text{def}}{=} \Phi_{\chi(v)} H^{-1/2} A^\top L[\ell]^{-\top} \quad \text{and } y_v^* \stackrel{\text{def}}{=} Z_v^* \cdot h.$$

To do so, we make use of the following properties:

Invariant 6.47. *The variables maintained in the data structure BalancedSketch, as given in Algorithm 12, preserve the following invariants before and after each function call:*

$$J_v = \Phi_{\chi(v)} H^{-1/2} A^\top \qquad v \in \mathcal{S} \tag{i}$$
$$Z_v = J_v \cdot L[t_v]^{-\top} \qquad v \in \mathcal{B} \tag{ii}$$
$$\mathbf{0} = (L[\ell] - L[t_v]) \cdot I_{\overline{\Lambda}(v)} \qquad v \in \mathcal{B} \tag{iii}$$
$$y_v^{\triangledown} = Z_v (I - I_{\Lambda(v)}) h^{(\ell)} \qquad v \in \mathcal{B} \tag{iv}$$

Lemma 6.48 (Sparsity Pattern of Z_v)**.** *Suppose J_v and Z_v satisfies (i) and (ii) of Invariant 6.47 for some $v \in \mathcal{B}$. Let S be the index set of the non-zero columns of Z_v, e.g. $S = \{i \in [d] \mid (Z_v)_i \neq \mathbf{0}\}$, then we have*

$$S \subseteq \bigcup_{i \in \overline{\chi}(v)} \mathcal{P}^{\mathcal{T}}(a_i).$$

Proof. By our construction of J_v, we note that nonzero columns of J_v lies on $\bigcup_{i \in \overline{\chi}(v)} \mathcal{P}^{\mathcal{T}}(a_i)$. By Lemma 5.4, we have $S \subseteq \bigcup_{i \in \overline{\chi}(v)} \mathcal{P}^{\mathcal{T}}(a_i)$. □

We have the following relationship between Z_v and Z_v^* for any $v \in \mathcal{T}$:

Lemma 6.49. *Suppose Invariant 6.47 is satisfied for v, then we have*

$$Z_v^* = Z_v - \left(L[\ell]^{-1}(L[\ell] - L[t_v]) \cdot I_{\Lambda(v)} Z_v^\top\right)^\top.$$

Proof. Similar to the proof of Lemma 6.29, we have

$$Z_v^* = Z_v - \left(L[\ell]^{-1}(L[\ell] - L[t_v]) Z_v^\top\right)^\top.$$

Let $\Delta L \stackrel{\text{def}}{=} L[\ell] - L[t_v]$. Then, we can split ΔL into three parts:

$$\Delta L = (I_{\Lambda(v)} + I_{\overline{\Lambda}(v)} + I_{\mathcal{T}\setminus(\Lambda(v)\cup\overline{\Lambda}(v))})\Delta L.$$

We first note that $I_{\mathcal{T}\setminus(\Lambda(v)\cup\overline{\Lambda}(v))} \cdot Z_v^\top = \mathbf{0}$. By Lemma 6.48, the nonzero columns of Z_v lies on

$$\bigcup_{i\in\overline{\chi}(v)} \mathcal{P}^{\mathcal{T}}(a_i) = (\Lambda(v) \cup \overline{\Lambda}(v)).$$

Hence, $I_{\mathcal{T}\setminus(\Lambda(v)\cup\overline{\Lambda}(v))} \cdot Z_v^\top = \mathbf{0}$. By (iii) of Invariant 6.47, we have $(L[\ell] - L[t_v]) \cdot I_{\overline{\Lambda}(v)} = \mathbf{0}$. Thus, we have

$$L[\ell]^{-1}(L[\ell] - L[t_v]) Z_v^\top = L[\ell]^{-1}(L[\ell] - L[t_v]) \cdot I_{\Lambda(v)} Z_v^\top.$$

□

Lemma 6.50 (Initialize)**.** *Given initial $\overline{x}$ and h, the JL matrix $\Phi \in \mathbb{R}^{r\times n}$, and the elimination tree $\mathcal{T}$, the data structure BalancedSketch initializes in time $O(n\tau^2 r \log n)$. Moreover, the internal state of the data structure satisfies the Invariant 6.47 after initialization.*

Proof. The correctness directly follows by Invariant 6.47.

By Corollary 5.8, we can find the initial Cholesky decomposition $L[0]$ in time $O(n\tau^2)$ time. For computing J_v, we note that $J_v = \sum_{\text{child } c \text{ of } v} J_c$. When $\chi(v) = \{i\}$ for some i, we can compute J_v in time $O(\tau r)$ by column sparsity of A. Since the height of tree is bounded by $O(\log n)$, we can compute all J_v in time $O(|\mathcal{S}| \cdot \tau r \log n)$. For Z_v, we note that

$$Z_v = \sum_{\text{child } c \text{ of } v} J_c \cdot L[\ell]^{-\top}$$

Hence, it suffices to compute $J_v \cdot L[\ell]^{-\top}$ for all leaf node $v \in \mathcal{B}$, which takes $O(n\tau^2 r)$ time. By Lemma 5.4, the solution of $J_v \cdot L[\ell]^{-\top}$ has $O(\tau r)$ nonzero entries. Hence, we can compute Z_v for all $v \in \mathcal{B}$ in time $O(n\tau^2 r \log n)$ time. Compute y_v^∇ takes time $O(n\tau^2 r \log n)$ time. □

Lemma 6.51 (Update)**.** *Suppose the current state of data structure satisfies Invariant 6.47. Given $\mathcal{W}^{\text{new}}$ implicitly by $\overline{x}^{\text{new}}$, and h^{new}, the function* Update *of* BalancedSketch *updates the sketches in $\mathcal{S}$ implicitly by in time $O(\|\overline{x}^{\text{new}} - \overline{x}\|_0 \cdot \tau^2 r \log n) + O(\|h^{\text{new}} - h\|_0 r \log n)$. Moreover, the function also updates the internal states correspondingly so that Invariant 6.47 is still preserved.*

Algorithm 12 Balanced Multiscale Representation Sketching Data Structure – Initialize and Query

datastructure BALANCEDSKETCH
private : members
  $\Phi \in \mathbb{R}^{r \times n}$ ▷ JL matrix
  sampling tree $(\mathcal{S}, \chi)$ with balanced binary tree $\mathcal{B}$ ▷ constructed as in Section 6.6.1
  $\ell \in \mathbb{N}$ ▷ iteration counter
  $h \in \mathbb{R}^d$
  $\overline{x} \in \mathbb{R}^n$ ▷ $\mathcal{W}$ given implicitly by $\overline{x}$
  $H \in \mathbb{R}^{n \times n}$ ▷ Hessian $H = \nabla^2 \phi(\overline{x})$
  LIST$\{L[t] \in \mathbb{R}^{d \times d}\}_{t \geq 0}$ ▷ sequence of Cholesky factor L at various iterations t
  LIST $\{J_v \in \mathbb{R}^{r \times d}\}_{v \in \mathcal{S}}$ ▷ $J_v = \Phi_{\chi(v)} H^{-1/2} A^\top$
  LIST $\{Z_v \in \mathbb{R}^{r \times d}\}_{v \in \mathcal{B}}$ ▷ $Z_v = \Phi_{\chi(v)} H^{-1/2} A^\top L[t_v]^{-\top}$
  LIST $\{y_v^\nabla \in \mathbb{R}^r\}_{v \in \mathcal{B}}$ ▷ $y_v^\nabla = Z_v (I - I_{\Lambda(v)}) h$
  LIST $\{t_v \in \mathbb{N}\}_{v \in \mathcal{B}}$ ▷ Last iterations during which node v was updated
end members
procedure INITIALIZE$(\mathcal{S}, \chi, \Phi \in \mathbb{R}^{r \times n}, \overline{x} \in \mathbb{R}^n, h \in \mathbb{R}^d)$ ▷ Lemma 6.50
  $(\mathcal{S}, \chi) \leftarrow (\mathcal{S}, \chi)$
  $\Phi \leftarrow \Phi$
  $\ell \leftarrow 0, h \leftarrow h$
  Find $\Lambda(v)$ for all $v \in \mathcal{B}$
  Compute $H \leftarrow \nabla^2 \phi(\overline{x})$
  Find lower Cholesky factor $L[\ell]$ of $AH^{-1}A^\top$
  for all $v \in \mathcal{S}$ **do**
    $J_v \leftarrow \Phi_{\chi(v)} H^{-1/2} A^\top$ ▷ compute J_v for all $v \in \mathcal{S}$
  end for
  for all $v \in \mathcal{B}$ **do**
    $Z_v \leftarrow J_v L[\ell]^{-\top}$
    $y_v^\nabla \leftarrow Z_v (I - I_{\Lambda(v)}) h$
    $t_v \leftarrow \ell$
  end for
end procedure
procedure QUERY$(v \in \mathcal{S})$ ▷ Lemma 6.52
  if $v \in \mathcal{S} \setminus \mathcal{B}$ **then**
    return $J_v \cdot L[\ell]^{-\top} h$ ▷ directly compute the value of sketch
  end if
  ▷ For $v \in \mathcal{T}$, we make use of existing partial computations
  $\Delta L \leftarrow (L[\ell] - L[t_v]) \cdot I_{\Lambda(v)}$
  $Z_v \leftarrow Z_v - (L[\ell]^{-1} \cdot \Delta L \cdot Z_v^\top)^\top$
  $y_v^\nabla \leftarrow Z_v \cdot (I - I_{\Lambda(v)}) h$
  $t_v \leftarrow \ell$ ▷ update the time stamp for node v
  $y_v^\triangle \leftarrow Z_v \cdot I_{\Lambda(v)} \cdot h$
  return $y_v^\triangle + y_v^\nabla$
end procedure

Algorithm 13 Balanced Multiscale Representation Sketching Data Structure – Updates

datastructure BALANCEDSKETCH
procedure UPDATE($\overline{x}^{\text{new}} \in \mathbb{R}^n, h^{\text{new}} \in \mathbb{R}^d$) ▷ Lemma 6.51
  for $i \in [m]$ where $x_i^{\text{new}} \neq x_i$ **do**
    UPDATEBLOCK(x_i^{new})
  end for
  for all $h_i^{\text{new}} \neq h_i$ **do**
    $v \leftarrow \Lambda^{\clubsuit}(i)$
    for all $u \in \mathcal{P}^{\mathcal{B}}(v)$ **do**
      $y_u^{\triangledown} \leftarrow Z_u \cdot I_{\{i\}} \cdot (h^{\text{new}} - h)$
    end for
  end for
  $h \leftarrow h^{\text{new}}$
end procedure
procedure UPDATEBLOCK($\overline{x}_i^{\text{new}} \in \mathbb{R}^{n_i}$) ▷ Lemma 6.53
  $\ell \leftarrow \ell + 1$
  $\overline{x}_i \leftarrow \overline{x}_i^{\text{new}}$
  $H^{\text{new}} = \nabla^2 \phi(\overline{x})$
  Find lower Cholesky factor $L[\ell]$ of $A(H^{\text{new}})^{-1} A^\top$
  $S \leftarrow \mathcal{P}^{\mathcal{B}}(\Lambda^{\clubsuit}(\mathsf{low}(A_i)))$
  UPDATE$L(S)$
  UPDATE$H(H^{\text{new}})$
end procedure
procedure UPDATE$L(S \subset \mathcal{B})$ ▷ Lemma 6.54
  for all $v \in S$ **do**
    ▷ We update Z_v in two steps: first from $L[t_v]$ to $L[\ell - 1]$, then from $L[\ell - 1]$ to $L[\ell]$
    $Z_v \leftarrow Z_v - \left(L[\ell-1]^{-1} (L[\ell-1] - L[t_v])_{\Lambda(v)} \cdot Z_v^\top \right)^\top$
    $Z_v \leftarrow Z_v - (L[\ell]^{-1} \cdot (L[\ell] - L[\ell-1]) \cdot Z_v^\top)^\top$
    $y_v^{\triangledown} \leftarrow Z_v (I - I_{\Lambda(v)}) h$
    $t_v \leftarrow \ell$
  end for
end procedure
procedure UPDATE$H(H^{\text{new}})$ ▷ Lemma 6.55
  $\Delta H = H^{\text{new}} - H$
  for all $i \in [m]$ such that $(\Delta H)_i \neq \mathbf{0}$ **do**
    Find set S such that $S = \{v \in \mathcal{S} \mid \chi(v) = \{j\}$ and coordinate j in i-th block$\}$
    for all $u \in \bigcup_{v \in S} \mathcal{P}^{\mathcal{S}}(v)$ **do**
      $J_v \leftarrow \Phi_{\chi(v)} (H + \Delta H \cdot I_{\{i\}})^{-1/2} A^\top$
      if $v \in \mathcal{B}$ **then**
        $Z_v \leftarrow J_v \cdot L[t_v]^{-\top}$
        $y_v^{\triangledown} \leftarrow Z_v \cdot (I - I_{\Lambda(v)}) h$
      end if
    end for
  end for
  $H \leftarrow H^{\text{new}}$
end procedure

Proof. To update $\overline{x}$ to $\overline{x}^{\text{new}}$, we view it as a sequence of updates, where each update corresponding to a single block change in $\overline{x}$, and use the helper function UpdateBlock. The proof of correctness and runtime are given in Lemma 6.53.

Similarly, we update h to h^{new} block-wise. Suppose h changes in coordinate j. We note that $(Z_v)_j \neq \mathbf{0}$ only if $j \in \bigcup_{i\in[l_v,r_v]} \mathcal{P}^{\mathcal{T}}(i)$. Then $Z_v(I - I_{\Lambda(v)})e_j \neq \mathbf{0}$ only if $j \in \bigcup_{i\in\overline{\chi}(v)} \mathcal{P}^{\mathcal{T}}(i)$ and $j \notin \Lambda(v)$. Hence, all such v lies on the path $\mathcal{P}^{\mathcal{B}}(\Lambda^{\clubsuit}(j))$.

For each coordinate j of h that changes, it suffices to compute $Z_v \cdot I_{\{j\}} \cdot (h^{\text{new}} - h)$. We note this can be done in $O(r)$ time. Thus, for each h_j, it takes $O(r \log n)$ time since $|\mathcal{P}^{\mathcal{B}}(\Lambda^{\clubsuit}(j))| = O(\log n)$. Hence, we can update h in $O(\|h^{\text{new}} - h\|_0 \cdot r \log n)$ time in total. □

Lemma 6.52 (Query)**.** *Suppose the state of the data structure satisfies Invariant 6.47 immediately before a call to Query. Then calling Query(v) of BalancedSketch returns $\Phi_{\chi(v)}\mathcal{W}^\top h$ in $O(\tau^2 r)$ time. Moreover, Invariant 6.28 is preserved at the end of the function call.*

Proof. When $v \in \mathcal{S} \setminus \mathcal{B}$, we directly compute $\Phi_{\chi(v)}H^{-1/2}A^\top L[\ell]^{-\top}h = J_v \cdot L[\ell]^{-\top}h^{(\ell)}$. Let u be the lowest ancestor node of v in $\mathcal{B}$. Note the row sparsity pattern of J_v lies on $\mathcal{P}^{\mathcal{T}}(u)$ by the construction of $\mathcal{S}$. Hence, we can solve $L[\ell]^{-1}J_v^\top$ in $O(\tau^2 r)$ time by Lemma 5.4. Since $h^{(\ell)}$ is given explicitly, we compute $J_v \cdot L[\ell]^{-\top}h^{(\ell)}$ in $O(\tau^2 r)$ time.

In the other case where $v \in \mathcal{B}$, the correctness follows by Invariant 6.47. For the runtime, by Lemma 6.39, we have $|\Lambda(v)| = O(\tau)$. By the sparsity pattern of L, we can compute $(L[\ell] - L[t_v]) \cdot I_{\Lambda(v)} \cdot Z_\triangledown[v]^\top$ in $O(\tau^2 r)$ time, and the column sparsity pattern of the result is on two paths of $\mathcal{T}$. Hence, we can update Z_v and $y_v^\triangledown$ in $O(\tau^2 r)$ time. Since $|\Lambda(v)| = O(\tau)$, computing $y_\triangle$ takes $O(r \cdot \tau)$ time. In total, we can compute $\Phi_{\chi(v)}\mathcal{W}^\top h$ in $O(\tau^2 r)$ time. □

Lemma 6.53 (UpdateBlock)**.** *Suppose the current state of the data structure satisfies Invariant 6.47. The function UpdateBlock of BalancedSketch updates the implicit representation of $\mathcal{W}$ by updating i-th block coordinate of $\overline{x}$, $\overline{x}_i$ to $\overline{x}_i^{\text{new}}$, in time $O(\tau^2 r \log n)$. Moreover, Invariant 6.47 is preserved after the function call.*

Proof. **Correctness:** To update L, it suffices to update the nodes in the set $S = \mathcal{P}^{\mathcal{B}}(\Lambda^{\clubsuit}(u))$: Indeed, note that only (iii) of Invariant 6.47 depends on $L[\ell]$. Thus, we need to update the sketch only if

$$(L[\ell+1] - L[\ell]) \cdot I_{\overline{\Lambda}(v)} \neq \mathbf{0}.$$

We use ΔL to denote $L[\ell+1] - L[\ell]$. By Lemma 5.9, the non-zero columns of ΔL lies on $\mathcal{P}^{\mathcal{T}}(\mathsf{low}^{\mathcal{T}}(A_i))$. Let $u = \mathsf{low}^{\mathcal{T}}(A_i)$, then we have

$$\Delta L = \sum_{w\in\mathcal{P}^{\mathcal{T}}(u)} (\Delta L)_w e_w^\top.$$

We note that $(\Delta L)_w e_w^\top \cdot I_{\overline{\Lambda}(v)} \neq \mathbf{0}$ only if $w \in \overline{\Lambda}(v)$. By definition of $\Lambda^{\clubsuit}$ and Lemma 6.44, we have $(\Delta L)_w e_w^\top \cdot I_{\overline{\Lambda}(v)} \neq \mathbf{0}$ only if $v \in \mathcal{P}^{\mathcal{B}}(\Lambda^{\clubsuit}(w))$. Thus, we need to update the set

$$\bigcup_{w\in\mathcal{P}^{\mathcal{T}}(u)} \mathcal{P}^{\mathcal{B}}(\Lambda^{\clubsuit}(w)).$$

By Lemma 6.46, we have $\Lambda^{\clubsuit}(w) \in \mathcal{P}^{\mathcal{B}}(\Lambda^{\clubsuit}(u))$. Hence, $\bigcup_{w\in\mathcal{P}^{\mathcal{T}}(u)} \mathcal{P}^{\mathcal{B}}(\Lambda^{\clubsuit}(w)) = \mathcal{P}^{\mathcal{B}}(\Lambda^{\clubsuit}(u))$.

Runtime: By Lemma 5.10, we can perform the rank-1 updates on L in $O(\tau^2)$ time. Since $S = \mathcal{P}^{\mathcal{B}}(\Lambda^{\clubsuit}(u))$, we have $|S| = O(\log n)$. We note when x_i updates, we only need to update one diagonal block of H, hence $\mathrm{nnz}(\Delta H) = O(1)$. By Lemmas 6.54 and 6.55, we can update H and L for the data structure in time $O(\tau^2 r \log n)$. Hence, the function takes $O(\tau^2 r \log n)$ time in total. □

Lemma 6.54 (UpdateL). *Suppose Invariant 6.47 is satisfied. Given the set $S \subset \mathcal{B}$, then the function UpdateL of BalancedSketch updates the sketches implicitly and t_v to the current time at each $v \in S$. If the number of non-zero column of ΔL is bounded by $O(\tau)$, then the function UpdateL takes $O(|S| \cdot \tau^2 \cdot r)$ time.*

Proof. **Correctness:** The correctness directly follows by Lemma 6.49.

Runtime: By Lemma 6.39, we have $|\Lambda(v)| = O(\tau)$. By Lemmas 5.2 and 5.9, we can compute $(L[\ell] - L[t_v]) \cdot I_{\Lambda(v)} \cdot Z_\nabla[v]^\top$ and $\Delta L \cdot Z_\nabla[v]^\top$ in time $O(\tau^2 r)$ and the column sparsity pattern of the result is on a path of $\mathcal{T}$. Hence, we can update Z_v and y_v^∇ in time $O(\tau^2 r)$. Hence, the total time is bounded by $O(|S| \cdot \tau^2 r)$. □

Lemma 6.55 (UpdateH). *Suppose Invariant 6.47 is satisfied for ℓ. Given ΔH such that $H[\ell{+}1] = H[\ell] + \Delta H$, then the function UpdateH updates H and the internal state of the data structure in $O(\mathrm{nnz}(\Delta H) \cdot \tau^2 r \log n)$ time.*

Proof. **Correctness:** We observe that Z_v changes only if $I_{\chi(v)} \cdot \Delta H \neq \mathbf{0}$. Suppose the i-th block of H changes, and let $v \in \mathcal{S}$ where $\chi(v) = \{i\}$. For changes on H_i, it suffices to update $\mathcal{P}^{\mathcal{S}}(v)$.

Runtime: Since H is a block-diagonal matrix, let $\widetilde{H} \stackrel{\text{def}}{=} (H[\ell] + \Delta H \cdot I_{\{i\}})^{-1/2} - H[\ell]^{-1/2}$, then the nonzero pattern of $\widetilde{H}$ is an $n_i \times n_i$ submatrix on the block diagonal. Recall $n_i = O(1)$, hence, we can find $\Phi_{\chi(v)} \widetilde{H} A^\top$ by computing $O(1)$ many outer products of columns of Φ and row of $A^\top$, which takes $O(\tau r)$ time by sparsity pattern of A (Lemma 5.1). Then, we can update Z_v by compute $\Phi_{\chi(v)} \widetilde{H} A^\top L[t_v]^{-\top}$, which takes $O(\tau^2 r)$ time by Lemma 5.4. Thus, we can update Z_v and y_v^∇ in time $O(\tau^2 r)$ time for each v. As the height of $\mathcal{S}$ is bounded by $O(\log n)$, this function takes time $O(\mathrm{nnz}(\Delta H) \cdot \tau^2 r \log n)$. □

6.7 Proof of Theorem 6.1

Proof of Correctness:[7]

At every iteration of MultiplyAndMove, we call super.Move followed by super.Update with the updated $\overline{x}^{\mathrm{new}}, \overline{s}^{\mathrm{new}}$ values. Therefore, we correctly maintain the implicit representation (x, s) directly as a result of Theorem 6.5.

Now, we show that

$$\|\overline{x}_i - x_i\|_{\overline{x}_i} \leqslant \overline{\varepsilon} \quad \text{and} \quad \|\overline{s}_i - s_i\|^*_{\overline{x}_i} \leqslant t\overline{\varepsilon} \quad \text{for all } i \in [m].$$

[7] [GS22] has noted an error pertaining to the maintenance of the approximate solutions (see their remark 4.19). Specifically, the guarantee of Eq. (6.6) is insufficient to satisfy the conditions of Theorem 6.14 for using the ℓ_∞-Approximates data structure. A fix is provided by [GS22], where the sampling procedure along the sampling tree is replaced by a top-down BFS. An alternative fix is to modify the ℓ_∞-Approximates data structure, so that rather than approximating $y = H_{\overline{x}}^{1/2} x$, where y is treated as a black-box vector, the data structure should maintain the term $H_{\overline{x}}^{1/2}$ separately from x. The correct implementation of this approach can be found in [DGG+22, Section 6]. In this manuscript, we have left the error as is.

Algorithm 14 Robust Central Path Maintenance – Initialize, MultiplyAndMove, Output

data structure CentralPathMaintenance **extends** MultiscaleRepresentation
private : member
  BalancedSketch $\mathsf{Sketch}\mathcal{W}^\top \varepsilon_x$, $\mathsf{Sketch}\mathcal{W}^\top \varepsilon_s$, $\mathsf{Sketch}\mathcal{W}^\top h$
    ▷ maintains $\mathcal{W}^\top \varepsilon_x, \mathcal{W}^\top \varepsilon_s$, and $\mathcal{W}^\top h$, Theorem 6.36
  VectorSketch $\mathsf{Sketch}H^{1/2}\widehat{x}$, $\mathsf{Sketch}H^{-1/2}\widehat{s}$, $\mathsf{Sketch}H^{-1/2}c_x$
    ▷ maintains $H_{\overline{x}}^{1/2}\widehat{x}$, $H_{\overline{x}}^{-1/2}\widehat{s}$, and $H_{\overline{x}}^{-1/2}c_x$, Theorem 6.18
  ℓ_∞−Approximates $\mathsf{Approx}H_{\overline{x}}^{1/2}x$, $\mathsf{Approx}H_{\overline{x}}^{-1/2}s$
    ▷ maintains ℓ_∞-approximations of $H_{\overline{x}}^{1/2}x$ and $H_{\overline{x}}^{-1/2}s$, Theorem 6.14
  Sampling tree $(\mathcal{S}, \chi)$
  $\ell \in \mathbb{N}$ ▷ central path step counter
  $N \in \mathbb{N}$ ▷ upper bound on total number of steps
  $k \leftarrow \sqrt{n}$ ▷ number of steps supported before a restart
  $r \leftarrow \Theta(\log^3(N))$
  $\Phi \in \mathbb{R}^{r \times n}$ ▷ JL matrix
end members
procedure Initialize($x \in \mathbb{R}^n, s \in \mathbb{R}^n, t \in \mathbb{R}_+, \overline{\varepsilon} \in (0,1)$)
  super.Initialize(x, s, x, s, t)
  Initialize $\Phi \in \mathbb{R}^{r \times n}$ by letting each entry be i.i.d. samples from $\mathcal{N}(0, \frac{1}{\sqrt{r}})$
  Construct sampling tree $(\mathcal{S}, \chi)$ as in Section 6.6.1.
  InitializeSketch()
  $\ell \leftarrow 0$
  $\varepsilon_{\text{apx}}^{(x)} \leftarrow \frac{\overline{\varepsilon}}{\max_i n_i}, \zeta^{(x)} \leftarrow 2\alpha, \delta_{\text{apx}} \leftarrow \frac{N}{20k}$ ▷ setting the appropriate approximation tolerances
  $\varepsilon_{\text{apx}}^{(s)} \leftarrow \frac{\overline{\varepsilon} \cdot \overline{t}}{2 \max_i n_i}, \zeta^{(s)} \leftarrow 2\alpha\overline{t}$ ▷ α, n_i as in Algorithm 16
  $\mathsf{Approx}H_{\overline{x}}^{1/2}x$.Initialize$(\mathcal{S}, \chi, \varepsilon_{\text{apx}}^{(x)}, \delta_{\text{apx}}, \zeta^{(x)}, k)$
  $\mathsf{Approx}H_{\overline{x}}^{-1/2}s$.Initialize$(\mathcal{S}, \chi, \varepsilon_{\text{apx}}^{(s)}, \delta_{\text{apx}}, \zeta^{(s)}, k)$
end procedure
procedure MultiplyAndMove($t \in \mathbb{R}_+$)
  $\ell \leftarrow \ell + 1$
  if $|\overline{t} - t| > \overline{t} \cdot \varepsilon_t$ or $\ell > k$ **then** ▷ restarts entire data structure
    Initialize$(x, s, t, \overline{\varepsilon})$
  end if
  super.Move()
    ▷ Oracle $\mathcal{O}_x, \mathcal{O}_s$ for the ℓ_∞-Approximates data structures, Lemma 6.56
  $\overline{x}^{\text{new}} \leftarrow H_{\overline{x}}^{-1/2} \cdot \mathsf{Approx}H_{\overline{x}}^{1/2}x$.Query$(\mathcal{O}_x\{H_{\overline{x}}^{1/2}\widehat{x} + \beta_x \cdot c_x - \mathcal{W}^\top(\beta_x h + \varepsilon_x)\})$
  $\overline{s}^{\text{new}} \leftarrow H_{\overline{x}}^{1/2} \cdot \mathsf{Approx}H_{\overline{x}}^{-1/2}s$.Query$(\mathcal{O}_s\{H_{\overline{x}}^{-1/2}\widehat{s} + \mathcal{W}^\top(\beta_s h + \varepsilon_s)\})$
  super.Update$(\overline{x}^{\text{new}}, \overline{s}^{\text{new}})$
  UpdateSketch()
end procedure
procedure Output
  return $\widehat{x} + H_{\overline{x}}^{-1/2}\beta_x c_x - H_{\overline{x}}^{-1/2}\mathcal{W}^\top(\beta_x h + \varepsilon_x),\ \ \widehat{s} + H_{\overline{x}}^{1/2}\mathcal{W}^\top(\beta_s h + \varepsilon_s)$
end procedure

Algorithm 15 Robust Central Path Maintenance – Helper Functions

datastructure CentralPathMaintenance **extends** MultiscaleRepresentation
procedure InitializeSketch
    $\mathsf{Sketch}\mathcal{W}^\top\varepsilon_x$.Initialize$(\mathcal{S}, \chi, \Phi, x, \varepsilon_x)$
    $\mathsf{Sketch}\mathcal{W}^\top\varepsilon_s$.Initialize$(\mathcal{S}, \chi, \Phi, x, \varepsilon_s)$
    $\mathsf{Sketch}\mathcal{W}^\top h$.Initialize$(\mathcal{S}, \chi, \Phi, x, h)$
    $\mathsf{Sketch}H^{-1/2}c_x$.Initialize$(\mathcal{S}, \chi, \Phi, H^{-1/2}c_x)$
    $\mathsf{Sketch}H^{1/2}\widehat{x}$.Initialize$(\mathcal{S}, \chi, \Phi, H_{\overline{x}}^{1/2}\widehat{x})$
    $\mathsf{Sketch}H^{-1/2}\widehat{s}$.Initialize$(\mathcal{S}, \chi, \Phi, H_{\overline{x}}^{-1/2}\widehat{s})$
end procedure
procedure UpdateSketch
    $\mathsf{Sketch}\mathcal{W}^\top\varepsilon_x$.Update$(\overline{x}, \varepsilon_x)$
    $\mathsf{Sketch}\mathcal{W}^\top\varepsilon_s$.Update$(\overline{x}, \varepsilon_s)$
    $\mathsf{Sketch}\mathcal{W}^\top h$.Update$(\overline{x}, h)$
    $\mathsf{Sketch}H^{-1/2}c_x$.Update$(H_{\overline{x}}^{-1/2}c_x)$
    $\mathsf{Sketch}H^{1/2}\widehat{x}$.Update$(H_{\overline{x}}^{1/2}\widehat{x})$
    $\mathsf{Sketch}H^{-1/2}\widehat{s}$.Update$(H_{\overline{x}}^{-1/2}\widehat{s})$
end procedure

Oracle $\mathcal{O}_x\{H_{\overline{x}}^{1/2}\widehat{x} + \beta_x \cdot c_x - \mathcal{W}^\top(\beta_x h + \varepsilon_x)\}$
procedure TypeI$(v \in \mathcal{S})$
    return $\mathsf{Sketch}H^{1/2}\widehat{x}$.Query$(v) + \beta_x \cdot \mathsf{Sketch}H^{-1/2}c_x$.Query$(v)-$
        $\beta_x \cdot \mathsf{Sketch}\mathcal{W}^\top h$.Query$(v) - \mathsf{Sketch}\mathcal{W}^\top\varepsilon_x$.Query$(v)$
end procedure
procedure TypeII$(i \in [n])$
        return $e_i^\top(H_{\overline{x}}^{1/2}\widehat{x} + \beta_x \cdot c_x - \mathcal{W}^\top(\beta_x h + \varepsilon_x))$
end procedure

Oracle $\mathcal{O}_s\{H_{\overline{x}}^{-1/2}\widehat{s} + \mathcal{W}^\top(\beta_s h + \varepsilon_s)\}$
procedure TypeI$(v \in \mathcal{S})$
    return $\mathsf{Sketch}H^{-1/2}\widehat{s}$.Query$(v) + \beta_s \cdot \mathsf{Sketch}\mathcal{W}^\top h$.Query$(v) + \mathsf{Sketch}\mathcal{W}^\top\varepsilon_s$.Query$(v)$
end procedure
procedure TypeII$(i \in [n])$
        return $e_i^\top(H_{\overline{x}}^{-1/2}\widehat{s} + \mathcal{W}^\top(\beta_s h + \varepsilon_s))$
end procedure

By construction of ℓ_∞-APPROXIMATES data structure, $\mathsf{Approx}H_{\overline{x}}^{1/2}x$ maintains an ℓ_∞-approximation of

$$H_{\overline{x}}^{1/2}\widehat{x} + \beta_x \cdot c_x - \mathcal{W}^\top(\beta_x h + \varepsilon_x) = H_{\overline{x}}^{1/2}x.$$

For any non-negative integer $\ell \leqslant k$, we have the guarantee

$$\|H_{\overline{x}}^{1/2}x^{(\ell+1)} - H_{\overline{x}}^{1/2}x^{(\ell)}\|_2 = \|H_{\overline{x}}^{1/2}(x^{(\ell+1)} - x^{(\ell)})\|_2 = \|\delta_x^{(\ell)}\|_{\overline{x}} \leqslant \frac{9}{8}\alpha \leqslant \zeta^{(x)}, \tag{6.6}$$

where we used Lemma A.9 for the first inequality. Since $\eta = O(\log n)$, $k = \sqrt{n}$, and $\delta_{\mathrm{apx}} = \frac{N}{2k}$, we can choose $r = \Theta(\log^3 N)$.

By Theorem 6.14, if $\widetilde{x}$ denotes the output of $\mathsf{Approx}\overline{x}$.QUERY, then it satisfies

$$\|H_{\overline{x}}^{1/2}x - \widetilde{x}\|_\infty \leqslant \varepsilon_{\mathrm{apx}}^{(x)} = \frac{\overline{\varepsilon}}{\max_i n_i}.$$

In Line 32, we set $\overline{x} = H^{-1/2}\widetilde{x}$, so

$$\|H^{1/2}(\overline{x} - x)\|_\infty \leqslant \frac{\overline{\varepsilon}}{\max_i n_i}.$$

Therefore, we have the desired error bound

$$\|\overline{x}_i - x_i\|_{\overline{x}_i} = \|H_{\overline{x}_i}^{1/2}(\overline{x}_i - x_i)\|_2 \leqslant \sqrt{n_i} \cdot \|H_{\overline{x}_i}^{1/2}(\overline{x}_i - x_i)\|_\infty \leqslant \overline{\varepsilon}.$$

Similarly, $\mathsf{Approx}H_{\overline{x}}^{-1/2}s$ maintains an ℓ_∞-approximation of

$$H_{\overline{x}}^{-1/2}\widehat{s} + \mathcal{W}^\top(\beta_s h + \varepsilon_s) = H_{\overline{x}}^{-1/2}s.$$

By Lemma A.9, for any non-negative integer $\ell \leqslant k$ we have

$$\|H_{\overline{x}}^{-1/2}\delta_s^{(\ell)}\| \leqslant \frac{9}{8}\alpha \cdot t \leqslant \frac{9}{8}\alpha\overline{t} \leqslant \zeta^{(s)},$$

where we used $t \leqslant \overline{t}$ at every step of the algorithm. By Theorem 6.14, if $\widetilde{s}$ denotes the output of $\mathsf{Approx}\overline{s}$.QUERY, then in Line 33, $\overline{s} = H_{\overline{x}}^{1/2}\widetilde{s}$, and so

$$\|H_{\overline{x}}^{-1/2}(s - \overline{s})\|_\infty \leqslant \varepsilon_{\mathrm{apx}}^{(s)} = \frac{\overline{t} \cdot \overline{\varepsilon}}{2\max_i n_i},$$

Therefore, we have the desired error bound

$$\|\overline{s}_i - s_i\|_{\overline{x}_i}^* = \|H_{\overline{x}_i}^{-1/2}(s_i - \overline{s}_i)\|_2 \leqslant \sqrt{n_i} \cdot \frac{\overline{t} \cdot \overline{\varepsilon}}{2\max_i n_i} \leqslant t\overline{\varepsilon},$$

where the last step follows by $0 < \varepsilon_t < \frac{1}{2}$ and hence $t \in (\overline{t}/2, \overline{t}]$.

By our choice of δ_{apx}, $\mathsf{Approx}H_{\overline{x}}^{1/2}x$ and $\mathsf{Approx}H_{\overline{x}}^{-1/2}s$ succeed with probability at least $1 - \frac{N}{10k}$. Taking the union bound over $\frac{N}{k}$ many restarts, the data structure succeeds with probability at least 0.9 after N total steps of central path.

Lastly, we ensure our oracle implementations are correct. For simplicity, we check $\mathcal{O}_x$:

Lemma 6.56. *Oracles $\mathcal{O}_x$ and $\mathcal{O}_s$ are implemented correctly on Lines 19 to 34 of Algorithm 15 for the latest query to the ℓ_∞-*ApproximateS *data structure $\mathsf{Approx}H_{\overline{x}}^{1/2}x$ and $\mathsf{Approx}H_{\overline{x}}^{-1/2}s$.*

Proof. The input vector is $H_{\overline{x}}^{1/2}\widehat{x}+\beta_x\cdot c_x-\mathcal{W}^\top(\beta_x h+\varepsilon_x)$. A type-I access at $v\in\mathcal{S}$ should return $\Phi\chi(v)(H_{\overline{x}}^{1/2}\widehat{x}+\beta_x\cdot c_x-\mathcal{W}^\top(\beta_x h+\varepsilon_x))$. By linearity of Φ, and by construction of the sketching data structures, this is precisely what the oracle implements. A type-II access should return coordinate i of the input vector, which the oracle does correctly.

The proof for $\mathcal{O}_s$ is identical, we omit it here. □

Proof of Runtime: We split this proof into Lemmas 6.57 to 6.59.

Lemma 6.57 (Initialization time)**.** *The initialization time of* CentralPathMaintenance *is $O(n\tau^2\log^4 N)$.*

Proof. By Theorem 6.5, initializing MultiscaleRepresentation takes $O(n\tau^2)$ time. We can construct the balanced sampling tree in time $O(n\tau+n\log n)$ by Theorem 6.38. By Theorems 6.18 and 6.36 and our choice of r, the initialization of each sketch takes $O(n\tau^2\log^4 N)$ time. Hence, the total initialization time is bounded by $O(n\tau^2\log^4 N)$. □

Lemma 6.58 (MultiplyAndMove time)**.** *Suppose that the function is called at most N times and t is monotonic decreasing, the total running time of* MultiplyAndMove *is*

$$O\left(\left(\frac{Nn^{1/2}}{\overline{\varepsilon}^4}+n\frac{\log(t_{\max}/t_{\min})}{\varepsilon_t}\right)\tau^2\operatorname{poly}\log(n/\overline{\varepsilon})\right).$$

Proof. By Theorem 6.5, Move takes time $O(1)$ time for each call. By Theorem 6.36, the sampling tree has height $\eta=O(\log n)$. Then, each Update$h$ takes time $O(\log^4 N)$ per coordinate change by Theorems 6.18 and 6.36. By Lemmas 6.60 and 6.61, each type-I query takes $O(\tau^2\log^3 N)$ time and type-II query takes $O(\tau^2)$ time. Thus, the running time of $\mathsf{Approx}H_{\overline{x}}^{1/2}x$ and $\mathsf{Approx}H_{\overline{x}}^{-1/2}s$ is bounded by $O(n\tau^2\cdot\operatorname{poly}\log(N))$ for every $k:=\sqrt{n}$ steps by Theorem 6.14 and our choice of α and $\overline{\varepsilon}$ in Algorithm 16. This also implies

$$\sum_{\ell=\ell_0}^{\ell_0+k}\|\overline{x}^{(\ell+1)}-\overline{x}^{(\ell)}\|_0+\|\overline{s}^{(\ell+1)}-\overline{s}^{(\ell)}\|_0=O(n\cdot\operatorname{poly}\log(N)).$$

Hence, by Update time in Theorem 6.5, the running time for this function during the algorithm is

$$\frac{N}{k}\cdot O(n\cdot\tau^2\cdot\operatorname{poly}\log(N))=O(Nn^{1/2}\tau^2\cdot\operatorname{poly}\log(N)).$$

and the total number of entries change during algorithm for each variable in Eq. (6.3) is

$$O\left(Nn^{1/2}\tau\cdot\operatorname{poly}\log(N)\right).$$

We note that $\mathsf{Approx}H_{\overline{x}}^{1/2}x$ (resp. $\mathsf{Approx}H_{\overline{x}}^{-1/2}s$) requires oracle queries to previous versions of variables maintained CentralPathMaintenance, including all the sketching data structures and the variables maintained in MultiscaleRepresentation. We resolve this by using persistent data structures throughout, costing an $O(\log N)$ multiplicative factor in all run-times, see e.g. [DSST89].

Hence, by Theorems 6.18 and 6.36, the total running time of UpdateSketch during algorithm is bounded by $O(Nn^{1/2}\tau^2 \cdot \operatorname{poly}\log(N))$.

Note that the algorithm will restart whenever $|\bar{t} - t| > \bar{t} \cdot \varepsilon_t$ or $\ell > k = \sqrt{n}$. Hence, we can bound the total number of restart by $\log_{1-\varepsilon_t}(t_{\min}/t_{\max}) + \frac{N}{k} = O(\frac{N}{k} + \log(t_{\max}/t_{\min})/\varepsilon_t)$. Each restart takes $O(n\tau^2 \log^4 N)$ time by Lemma 6.57.

Thus, the runtime of MultiplyAndMove is bounded by

$$\begin{aligned}
&O\left(Nn^{1/2}\tau^2 \operatorname{poly}\log(N) + n\tau^2 \log^4 N \left(\frac{\log(t_{\max}/t_{\min})}{\varepsilon_t} + \frac{N}{k}\right)\right) \\
=&O\left(\left(Nn^{1/2} + n\frac{\log(t_{\max}/t_{\min})}{\varepsilon_t}\right)\tau^2 \operatorname{poly}\log(N)\right) \\
=&O\left(\left(Nn^{1/2} + n\log(t_{\max}/t_{\min})\right)\tau^2 \operatorname{poly}\log(N)\right).
\end{aligned}$$

where the last step follows by the choice of ε_t in Algorithm 16. □

Lemma 6.59 (Output time)**.** *Output runs in $O(n\tau^2)$ time.*

Proof. We note that we compute $\beta_x h + \varepsilon_x$ exactly in time $O(n)$. Recall that $\mathcal{W} = L_{\overline{x}}^{-1} A H_{\overline{x}}^{-1/2}$, we can compute $\mathcal{W}^\top(\beta_x h + \varepsilon_x)$ in time $O(n\tau^2)$ by Lemma 5.6. Hence, we can compute x in time $O(n\tau^2)$. The analysis for s is identical, we omit it here. □

Lemma 6.60 (Query time)**.** *Type-I queries to the oracles $\mathcal{O}_x$ and $\mathcal{O}_s$ run in $O(\tau^2 \cdot \log^3 N)$ time.*

Proof. By the runtime of Query in Theorems 6.18 and 6.36 and $r = \Theta(\log^3 N)$, the total query time is bounded by $O(\tau^2 \cdot \log^3 N)$. □

Lemma 6.61 (Compute time)**.** *Type-II queries to the oracles $\mathcal{O}_x$ and $\mathcal{O}_s$ run in $O(\tau^2)$ time.*

Proof. We show the claim for $\mathcal{O}_x$: Since $H_{\overline{x}}$ is a block-diagonal matrix and $n_i = O(1)$, we can compute $e_i^\top(H_{\overline{x}}^{1/2}\widehat{x} + \beta_x \cdot c_x)$ in $O(1)$ time. Now, it suffices to show we can compute $e_i^\top \mathcal{W}^\top(\beta_x h + \varepsilon_x)$ in $O(\tau^2)$ time. By the definition of $\mathcal{W}$, we have

$$e_i^\top \mathcal{W}^\top(\beta_x h + \varepsilon_x) = (\beta_x h + \varepsilon_x)^\top L_{\overline{x}}^{-1} A H_{\overline{x}}^{-1/2} e_i.$$

By Lemmas 5.1 and 5.4, we can compute $y = L_{\overline{x}}^{-1} A H_{\overline{x}}^{-1/2} e_i$ in $O(\tau^2)$ time and y has $O(\tau)$ many non-zero entries. Then, we can compute the product $(\beta_x h + \varepsilon_x)^\top y$ in $O(\tau)$ time. Hence, the total runtime for a type-II query to $\mathcal{O}_x$ is $O(\tau^2)$. The proof for $\mathcal{O}_s$ is identical; we omit it here. □

7 Acknowledgment

We thank Aaron Sidford for discussing the optimization on thick path and Anup B. Rao for discussing the convex regression problem. The authors are supported by NSF awards CCF-1749609, CCF-1740551, DMS-1839116, DMS-2023166, Microsoft Research Faculty Fellowship, Sloan Research Fellowship, and Packard Fellowships.

References

[ABGJ18] Xavier Allamigeon, Pascal Benchimol, Stéphane Gaubert, and Michael Joswig. Log-barrier interior point methods are not strongly polynomial. *SIAM Journal on Applied Algebra and Geometry*, 2(1):140–178, 2018.

[ADD96] Patrick R Amestoy, Timothy A Davis, and Iain S Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.

[AK16] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. *Journal of the ACM (JACM)*, 63(2):1–35, 2016.

[ASK12] Takuya Akiba, Christian Sommer, and Ken-ichi Kawarabayashi. Shortest-path queries for complex networks: exploiting low tree-width outside the core. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 144–155, 2012.

[AY13] Noga Alon and Raphael Yuster. Matrix sparsification and nested dissection over arbitrary fields. *Journal of the ACM (JACM)*, 60(4):1–18, 2013.

[BCR91] Gregory Beylkin, Ronald Coifman, and Vladimir Rokhlin. Fast wavelet transforms and numerical algorithms i. *Communications on pure and applied mathematics*, 44(2):141–183, 1991.

[BGHK95] Hans L Bodlaender, John R Gilbert, Hjálmtyr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18(2):238–255, 1995.

[BGS21] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental sssp and approximate min-cost flow in almost-linear time. *arXiv preprint arXiv:2101.07149*, 2021.

[BJL+19] Sébastien Bubeck, Qijia Jiang, Yin-Tat Lee, Yuanzhi Li, and Aaron Sidford. Complexity of highly parallel non-smooth convex optimization. In *Advances in Neural Information Processing Systems*, pages 13900–13909, 2019.

[BKO19] Cornelius Brand, Martin Kouteckỳ, and Sebastian Ordyniak. Parameterized algorithms for milps with small treedepth. *arXiv preprint arXiv:1912.03501*, 2019.

[BLL+20] Jan van den Brand, Yin-Tat Lee, Yang P. Liu, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Maximum flow in nearly-linear time on moderately dense graphs. *Personal communication*, 2020.

[Bod94] Hans L Bodlaender. A tourist guide through treewidth. 1994.

[BS18] Eric Balkanski and Yaron Singer. Parallelization does not accelerate convex optimization: Adaptivity lower bounds for non-smooth convex minimization. *arXiv preprint arXiv:1808.03880*, 2018.

[BW17] Sebastian Brandt and Roger Wattenhofer. Approximating small balanced vertex separators in almost linear time. In *Workshop on Algorithms and Data Structures*, pages 229–240. Springer, 2017.

[CEH+20] Jana Cslovjecsek, Friedrich Eisenbrand, Christoph Hunkenschröder, Lars Rohwedder, and Robert Weismantel. Block-structured integer and linear programming in strongly polynomial and near linear time, 2020.

[CŁ13] Krishnendu Chatterjee and Jakub Łącki. Faster algorithms for markov decision processes with low treewidth. In *International Conference on Computer Aided Verification*, pages 543–558. Springer, 2013.

[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.*, pages 938–942, 2019.

[CM05] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, 2005.

[CZ00] Shiva Chaudhuri and Christos D Zaroliagis. Shortest paths in digraphs of small treewidth. part i: Sequential algorithms. *Algorithmica*, 27(3-4):212–226, 2000.

[Dan51] George B Dantzig. Maximization of a linear function of variables subject to linear inequalities. *Activity analysis of production and allocation*, 13:339–347, 1951.

[Dav06] Timothy A. Davis. *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, jan 2006.

[dBCvKO08] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008.

[DFK91] Martin Dyer, Alan Frieze, and Ravi Kannan. A random polynomial-time algorithm for approximating the volume of convex bodies. *Journal of the ACM (JACM)*, 38(1):1–17, 1991.

[DGG+22] Sally Dong, Yu Gao, Gramoz Goranci, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Guanghao Ye. Nested dissection meets ipms: Planar min-cost flow in nearly-linear time. In *Proceedings of the 2022 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 124–153. SIAM, 2022.

[DGRW19] David Durfee, Yu Gao, Anup B Rao, and Sebastian Wild. Efficient second-order shape-constrained function fitting. In *Workshop on Algorithms and Data Structures*, pages 395–408. Springer, 2019.

[DH03] Timothy A. Davis and William W. Hager. Modifying a Sparse Cholesky Factorization. *SIAM Journal on Matrix Analysis and Applications*, 20(3):606–627, 2003.

[DKO97] Wolfgang Dahmen, Andrew Kurdila, and Peter Oswald. *Multiscale wavelet methods for partial differential equations*. Elsevier, 1997.

[DSST89] James R. Driscoll, Neil Sarnak, Daniel Dominic Sleator, and Robert Endre Tarjan. Making data structures persistent. *J. Comput. Syst. Sci.*, 38(1):86–124, 1989.

[EHK+19] Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. An algorithmic theory of integer programming. *arXiv preprint arXiv:1904.01361*, 2019.

[FLS+18] Fedor V Fomin, Daniel Lokshtanov, Saket Saurabh, Michał Pilipczuk, and Marcin Wrochna. Fully polynomial-time parameterized computations for graphs and matrices of low treewidth. *ACM Transactions on Algorithms (TALG)*, 14(3):1–45, 2018.

[FMP+18] Matthew Fahrbach, Gary L Miller, Richard Peng, Saurabh Sawlani, Junxing Wang, and Shen Chen Xu. Graph sketching against adaptive adversaries applied to the minimum degree algorithm. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 101–112. IEEE, 2018.

[Geo73] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973.

[GL89] Alan George and Joseph WH Liu. The evolution of the minimum degree ordering algorithm. *Siam review*, 31(1):1–19, 1989.

[GLN94] Alan George, Joseph Liu, and Esmond Ng. Computer solution of sparse linear systems. *Oak Ridge National Laboratory*, 1994.

[GS22] Yuzhou Gu and Zhao Song. A faster small treewidth sdp solver. *arXiv preprint arXiv:2211.06033*, 2022.

[HZ15] Thomas Dueholm Hansen and Uri Zwick. An improved version of the random-facet pivoting rule for the simplex algorithm. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 209–218, 2015.

[JK15] Bart MP Jansen and Stefan Kratsch. A structural approach to kernels for ilps: Treewidth and total unimodularity. In *Algorithms-ESA 2015*, pages 779–791. Springer, 2015.

[JKL+20] Haotian Jiang, Tarun Kathuria, Yin Tat Lee, Swati Padmanabhan, and Zhao Song. A faster interior point method for semidefinite programming. *arXiv preprint arXiv:2009.10217*, 2020.

[JLSW20] Haotian Jiang, Yin Tat Lee, Zhao Song, and Sam Chiu-wai Wong. An improved cutting plane method for convex optimization, convex-concave games, and its applications. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 944–953, 2020.

[JSWZ20] Shunhua Jiang, Zhao Song, Omri Weinstein, and Hengjie Zhang. Faster dynamic matrix inverse for faster lps. *arXiv preprint arXiv:2004.07470*, 2020.

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

[Kha80] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[KK98] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[KLOS14] Jonathan A Kelner, Yin Tat Lee, Lorenzo Orecchia, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its mul-

ticommodity generalizations. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 217–226. SIAM, 2014.

[KPSZ18] Rasmus Kyng, Richard Peng, Robert Schwieterman, and Peng Zhang. Incomplete nested dissection. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 404–417, 2018.

[LMP+20] Daniel Lokshtanov, Pranabendu Misra, Michał Pilipczuk, Saket Saurabh, and Meirav Zehavi. An exponential time parameterized algorithm for planar disjoint paths. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 1307–1316, 2020.

[LRT79] Richard J Lipton, Donald J Rose, and Robert Endre Tarjan. Generalized nested dissection. *SIAM journal on numerical analysis*, 16(2):346–358, 1979.

[LS13] Yin Tat Lee and Aaron Sidford. Path finding i: Solving linear programs with\˜ o (sqrt (rank)) linear system solves. *arXiv preprint arXiv:1312.6677*, 2013.

[LS19] Yin Tat Lee and Aaron Sidford. Solving linear programs with sqrt(rank) linear system solves. *CoRR*, abs/1910.08033, 2019.

[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*, pages 2140–2157, 2019.

[LV18] Yin Tat Lee and Santosh S Vempala. The kannan-lovász-simonovits conjecture. *arXiv preprint arXiv:1807.03465*, 2018.

[LY18] Yin Tat Lee and Man-Chung Yue. Universal barrier is n-self-concordant. *arXiv preprint arXiv:1809.03011*, 2018.

[MT14] Murat Mut and Tamás Terlaky. A tight iteration-complexity upper bound for the mty predictor-corrector algorithm via redundant klee-minty cubes. 2014.

[Nem94] Arkadi Nemirovski. On parallel complexity of nonsmooth convex optimization. *Journal of Complexity*, 10(4):451–463, 1994.

[Nes98] Yurii Nesterov. Introductory lectures on convex optimization - a basic course. In *Lecture Notes*, 1998.

[NN91] Yurii Nesterov and Arkadi Nemirovsky. Acceleration and parallelization of the path-following interior point method for a linearly constrained convex quadratic problem. *SIAM Journal on Optimization*, 1(4):548–564, 1991.

[NN94] Yurii Nesterov and Arkadii Nemirovskii. *Interior-point polynomial algorithms in convex programming*. SIAM, 1994.

[PdWvdK12] Leon R Planken, Mathijs M de Weerdt, and Roman PJ van der Krogt. Computing all-pairs shortest paths by leveraging low treewidth. *Journal of artificial intelligence research*, 43:353–388, 2012.

[PV20] Richard Peng and Santosh Vempala. Solving sparse linear systems faster than matrix multiplication, 2020.

[Ren88] James Renegar. A polynomial-time algorithm, based on newton's method, for linear programming. *Mathematical programming*, 40(1-3):59–93, 1988.

[RV91] Olivier Rioul and Martin Vetterli. Wavelets and signal processing. *IEEE signal processing magazine*, 8(4):14–38, 1991.

[Sch82] R. Schreiber. A new implementation of sparse gaussian elimination. *ACM Trans. Math. Softw.*, 8:256–276, 1982.

[SCHO13] Hayden Schaeffer, Russel Caflisch, Cory D Hauck, and Stanley Osher. Sparse dynamics for partial differential equations. *Proceedings of the National Academy of Sciences*, 110(17):6634–6639, 2013.

[ST83] Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

[SV13] Nikhil Srivastava and Roman Vershynin. Covariance estimation for distributions with $2 + \varepsilon$ moments. *The Annals of Probability*, 41(5):3081–3111, 2013.

[TSR+05] Robert Tibshirani, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight. Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108, 2005.

[Vai89] Pravin M Vaidya. A new algorithm for minimizing convex functions over convex sets. In *30th Annual Symposium on Foundations of Computer Science*, pages 338–343. IEEE Computer Society, 1989.

[vdB20] Jan van den Brand. A deterministic linear program solver in current matrix multiplication time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 259–278. SIAM, 2020.

[vdBLN+20] Jan van den Brand, Yin-Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs, 2020.

[vdBLSS20] Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 775–788, 2020.

[Ye20] Guanghao Ye. Fast algorithm for solving structured convex programs, 2020.

[ZL18] Richard Y Zhang and Javad Lavaei. Sparse semidefinite programs with near-linear time complexity. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 1624–1631. IEEE, 2018.

A Robust Interior Point Algorithm for General Convex Sets

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

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

where A is a $d \times n$ matrix, $x_i \in K_i \subset \mathbb{R}^{n_i}$, and x is the concatenation of x_i lying inside the domain $K \stackrel{\text{def}}{=} \prod_{i=1}^m K_i \subset \mathbb{R}^n$ with $n = \sum_{i=1}^m n_i$. The main result of this section is the following:

Theorem A.1. *Consider the convex program Eq. (CP). Given ν_i-self-concordant barriers $\phi_i : K_i \to \mathbb{R}$ with its minimum x_i. Define the following parameters of the convex problem:*

1. *Inner radius r: There exists a z such that $Az = b$ and $B(z, r) \subset K$.*
2. *Outer radius R: We have $K \subset B(x, R)$ for some $x \in \mathbb{R}^n$.*
3. *Lipschitz constant L: $\|c\|_2 \leqslant L$.*

Let $w \in \mathbb{R}^m_{\geq 1}$ be any weight vector, and $\kappa = \sum_{i=1}^m w_i \nu_i$. For any $0 < \varepsilon \leqslant 1/2$, Algorithm 16 outputs an approximate solution x in $O(\sqrt{\kappa} \log(m) \log(\frac{n\kappa R}{\varepsilon r}))$ steps, such that $Ax = b$, $x \in K$ and

$$c^\top x \leqslant \min_{Ax=b,\ x \in K} c^\top x + \varepsilon L R.$$

Remark A.2. If the barrier functions ϕ_i is not given, we can use $w_i = 1$ and universal barrier functions ϕ_i for K_i [NN94, LY18]. In this case, the algorithm takes $O(\sqrt{n} \log n \log(\frac{n\kappa R}{\varepsilon r}))$ steps, and the cost of computing a good enough approximation of $\nabla \phi_i$ and $\nabla^2 \phi_i$ both takes $n_i^{O(1)} \log(\frac{nR}{r})$ time for each i, assuming the following mild conditions:

1. We can check if x_i is in K_i in time $n_i^{O(1)}$.
2. We are given x_i such that $B(x_i, r) \subset K$.

Our algorithm and the proof is a simplified but strengthen version of [LSZ19]. We introduce approximate t in the algorithm to simplify our main data structure. We introduce a new reduction for finding initial point, which allows us to output x exactly satisfying $Ax = b$. We used the potential $\cosh(\|\cdots\|)$ instead of $\exp(\|\cdots\|)$ as in [LSZ19] and this simplifies the proof and the algorithm for the data structure.

Although we will simply use $w_i = 1$ for all i in this paper, we support the use of other weights in case it is useful in the future. Another improvement over [LSZ19] is that our bound is tight even for the case some ν_i is much larger than other ν_i. We note that it is an interesting open question to extend it to dynamic weighted barriers such as the Lee-Sidford barrier [LS19] (beyond the case $n_i = 1$).

A.1 Overview

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

$$x(t) = \arg\min_{Ax=b} c^\top x + t\phi(x) \quad \text{with } \phi(x) \stackrel{\text{def}}{=} \sum_{i=1}^m w_i \phi_i(x_i) \tag{A.1}$$

where ϕ_i are self-concordant barrier functions on K_i. The weights $w \in \mathbb{R}^m_{>0}$ are fixed throughout the algorithm.

Definition A.3 ([Nes98])**.** A function ϕ is a ν-self-concordant barrier for a non-empty open convex set K if $\operatorname{dom}\phi = K$, $\phi(x) \to +\infty$ as $x \to \partial K$, and for any $x \in K$ and for any $u \in \mathbb{R}^n$

$$D^3\phi(x)[u,u,u] \leqslant 2\|u\|_{\nabla^2\phi(x)} \text{ and } \|\nabla\phi(x)\|_{(\nabla^2\phi(x))^{-1}} \leqslant \sqrt{\nu}.$$

A function ϕ is a self-concordant barrier if the first condition holds.

For many convex sets, we have an explicit barrier with $\nu = O(n)$. For the case of linear programs, the convex set $K_i = [\ell_i, u_i]$ and one can use the log barrier $-\log(u_i - x) - \log(x - \ell_i)$. It has self-concordance 1. Throughout this section, we only use the fact that $\nu \geq 1$ to simplify formulas.

Lemma A.4 ([Nes98, Corollary 4.3.1])**.** *The self-concordance ν is larger than 1 for any barrier function.*

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). By the definition of $x(t)$, $x(0)$ is a minimizer of the problem Eq. (CP). In Appendix A.2 to Appendix A.4, we explain how to follow the path from $x(t)$ to $x(0)$ assuming $x(t)$ is given for some t. In Appendix A.6, we show how to find the initial point $x(t)$ (for some t) quickly by reformulating the problem into an equivalent form.

A.2 Interior Point Algorithm

In this section, we discuss how to follow the path $x(t)$ efficiently. To lower the cost of each step, we maintain our (x, s) implicitly. Throughout the algorithm, we only access an approximation of (x, s), which called $(\overline{x}, \overline{s})$. Our algorithm takes $O(\sqrt{\sum_i w_i\nu_i}\log(1/\varepsilon))$ steps and each step involves solving some linear system according to $(\overline{x}, \overline{s})$.

To analyze the central path, we use the norm induced by the Hessian of Φ throughout this paper.

Definition A.5 (Induced Norms)**.** For each block K_i, 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 $K = \prod_{i=1}^m K_i$, we define $\|v\|_x \stackrel{\text{def}}{=} \|v\|_{\nabla^2\phi(x)} = \sqrt{\sum_i w_i\|v_i\|^2_{x_i}}$ 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 it changes as the parameter x changes. The following lemma about self-concordance implies when the parameter x is not changed rapidly, then the approximate solution for previous iteration will not be too far from the solution of next iteration.

Lemma A.6 ([Nes98, Theorem 4.1.6])**.** *Given a self-concordant barrier ϕ. For any $x \in \operatorname{dom}\phi$ and any y such that $\|y - x\|_x < 1$, we have $y \in \operatorname{dom}\phi$ and that*

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

Instead of following the path $x(t)$ exactly, we follow the path

$$\begin{aligned} s/t + w\nabla\phi(x) &= \mu, \\ Ax &= b, \\ A^\top y + s &= c \end{aligned} \tag{A.2}$$

Algorithm 16 A Robust Interior Point Method for Eq. (CP)

procedure InteriorPointMethod
  Input: linear program $A \in \mathbb{R}^{d\times n}, b \in \mathbb{R}^d, c \in \mathbb{R}^n$ with inner radius r and outer radius R
  Input: ν_i self-concordant barrier functions $\phi_i : \mathbb{R}^{n_i} \to \mathbb{R}$ for $i \in [m]$ and its weight $w \in \mathbb{R}^m_{\geq 1}$
  Let $\phi(x) \stackrel{\text{def}}{=} \sum_{i=1}^m w_i \phi_i(x_i)$, $L = \|c\|_2, \kappa = \sum_{i=1}^m w_i \nu_i$
  ▷ Modify the convex program and obtain an initial (x, s) according to Theorem A.18
  Let $t = 2^{16}(n+\kappa)^5 \cdot \frac{LR}{\delta} \cdot \frac{R}{r}$ with $\delta = 1/128$
  Compute $x_c = \arg\min_{x\in K} c^\top x + t\phi(x)$ and $x_\circ = \arg\min_{Ax=b} \|x - x_c\|_2$
  Let $x = (x_c, 3R + x_\circ - x_c, 3R)$ and $s = (-t\nabla\phi(x_c), \frac{t}{3R+x_\circ - x_c}, \frac{t}{3R})$
  Let the new matrix $A^{\text{new}} = [A, A, -A]$, the new barrier and new weight

$$\phi_i^{\text{new}} = \begin{cases} \phi_i & \text{if } i \in [m] \\ -\log x & \text{elses} \end{cases} \quad \text{and} \quad w_i^{\text{new}} = \begin{cases} w_i & \text{if } i \in [m] \\ 1 & \text{elses} \end{cases}$$

  ▷ Find an initial (x, s) for the original linear program.
  $((x^{(1)}, x^{(2)}, x^{(3)}), (s^{(1)}, s^{(2)}, s^{(3)})) \leftarrow \textsc{Centering}(A^{\text{new}}, \phi^{\text{new}}, w^{\text{new}}, x, s, t, LR)$
  $(x, s) \leftarrow (x^{(1)} + x^{(2)} - x^{(3)}, s^{(1)})$
  ▷ Optimize the original linear program.
  $(x, s) \leftarrow \textsc{Centering}(A, \phi, w, x, s, LR, \frac{\varepsilon}{4\sum_i w_i \nu_i})$
  Return x
end procedure
procedure Centering$(A, \phi, w, x, s, t_{\text{start}}, t_{\text{end}})$
  ▷ Definitions
  $\lambda = 64\log(256m\sum_{i=1}^m w_i)$, $\overline{\varepsilon} = \frac{1}{1440\lambda}$, $\alpha = \frac{\overline{\varepsilon}}{2}$
  $\varepsilon_t = \frac{\overline{\varepsilon}}{4}(\min_i \frac{w_i}{w_i+\nu_i})$, $h = \frac{\alpha}{64\sqrt{\sum_{i=1}^m w_i\nu_i}}$ where ν_i is the self-concordance of ϕ_i
  $\mu_i^t(x, s) \stackrel{\text{def}}{=} s_i/t + w_i\nabla\phi_i(x_i)$, $\gamma_i^t(x, s) \stackrel{\text{def}}{=} \|\mu_i^t(x, s)\|^*_{x_i}$
  $c_i^t(x, s) \stackrel{\text{def}}{=} \frac{\sinh(\frac{\lambda}{w_i}\gamma_i^t(x,s))}{\gamma_i^t(x,s)\cdot\sqrt{\sum_{j=1}^m w_j^{-1}\cosh^2(\frac{\lambda}{w_i}\gamma_j^t(x,s))}}$
  $\Psi_\lambda(r) \stackrel{\text{def}}{=} \sum_{i=1}^m \cosh(\lambda r_i/w_i)$, $\Phi^t(x, s) \stackrel{\text{def}}{=} \Psi_\lambda(\gamma^t(x, s))$
  $P_x \stackrel{\text{def}}{=} H_x^{-1/2}A^\top(AH_x^{-1}A^\top)^{-1}AH_x^{-1/2}$ and $H_x \stackrel{\text{def}}{=} \nabla^2\phi(x)$
  ▷ Main Loop
  $\overline{t} = t = t_{\text{start}}, \overline{x} = x, \overline{s} = s, k = 0$.
  while $t \geq t_{\text{end}}$ **do**
    Maintain $\overline{x}, \overline{s}, \overline{t}$ such that $\|\overline{x}_i - x_i\|_{\overline{x}_i} \leqslant \overline{\varepsilon}$, $\|\overline{s}_i - s_i\|^*_{\overline{x}_i} \leqslant \overline{t}\overline{\varepsilon}w_i$ and $|\overline{t} - t| \leqslant \varepsilon_t\overline{t}$
    $\delta_{\mu,i} \leftarrow -\alpha \cdot c_i^{\overline{t}}(\overline{x}, \overline{s}) \cdot \mu_i^{\overline{t}}(\overline{x}, \overline{s})$ for all $i \in [m]$
    Pick δ_x and δ_s such that $A\delta_x = 0$, $\delta_s \in \text{Range}(A^\top)$ and

$$\|H_{\overline{x}}^{1/2}\delta_x - (I - P_{\overline{x}})H_{\overline{x}}^{-1/2}\delta_\mu\|_2 \leqslant \overline{\varepsilon}\alpha$$
$$\|\overline{t}^{-1}H_{\overline{x}}^{-1/2}\delta_s - P_{\overline{x}}H_{\overline{x}}^{-1/2}\delta_\mu\|_2 \leqslant \overline{\varepsilon}\alpha$$

    $k \leftarrow k + 1$, $t \leftarrow \max((1-h)t, t_{\text{end}})$, $x \leftarrow x + \delta_x$, $s \leftarrow s + \delta_s$
  end while
  Return (x, s)
end procedure

where μ is close to 0 in $(\nabla^2\phi(x))^{-1}$ norm. We enforce μ close to 0 using the following potential.

Definition A.7 (Potential Function)**.** For each $i \in [m]$, we define the i-th coordinate error

$$\mu_i^t(x,s) \stackrel{\text{def}}{=} \frac{s_i}{t} + w_i \nabla\phi_i(x_i) \tag{A.3}$$

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. The algorithm alternates between decreasing t multiplicatively and a Newton-like step on Eq. (A.2) and the proof simply shows the potential Φ is bounded throughout. In Appendix A.3 and Appendix A.4, we explain how we design our Newton step. In Appendix A.5, we bound how Φ changes under our Newton step. Finally, we give the proof of Theorem A.1 in Appendix A.7.

A.3 Gradient Descent on Ψ_λ

Since our goal is to bound $\Phi(x,s) = \Psi_\lambda(\gamma)$, we first discuss how to decreases $\Psi_\lambda(r)$ by directly controlling r. 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[8]:

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

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^*. To avoid the extra difficulties arising from 0 divided by 0, we replace the sinh by cosh in the denominator.

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

$$\delta_r = \frac{-\alpha \cdot \sinh(\frac{\lambda}{w_i}\overline{r}_i)}{\sqrt{\sum_j w_j^{-1}\cosh^2(\frac{\lambda}{w_j}\overline{r}_j)}} + \varepsilon_r \tag{A.4}$$

with $\sqrt{\sum_i w_i^{-1}\varepsilon_{r,i}^2} \leqslant \frac{\alpha}{8}$. For any step size $0 \leqslant \alpha \leqslant \frac{1}{8\lambda}$, we have that

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

[8] 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

$$\Psi_\lambda(r+\delta_r) = \Psi_\lambda(r) + \langle \nabla\Psi_\lambda(r), \delta_r\rangle + \frac{1}{2}\delta_r^\top \nabla^2\Psi_\lambda(\widetilde{r})\delta_r \tag{A.5}$$

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

For the first order term $\langle \nabla\Psi_\lambda(r), \delta_r - \varepsilon_r\rangle$ in Eq. (A.5), we have that

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

Using Lemma A.31 and the assumption $|r_i - \overline{r}_i| < \frac{w_i}{8\lambda}$, we have

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

Hence, we have

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

Using Lemma A.31 and the assumption $|r_i - \overline{r}_i| < \frac{w_i}{8\lambda}$ again, we have

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

Putting this into Eq. (A.6), we have

$$\langle \nabla\Psi_\lambda(r), \delta_r - \varepsilon_r\rangle \leqslant -\frac{36}{49}\alpha\lambda\sqrt{\sum_i w_i^{-1}\cosh^2(\frac{\lambda}{w_i}r_i)} + \alpha\lambda\sqrt{\sum_i w_i^{-1}}. \tag{A.7}$$

For the first order term $\langle \nabla\Psi_\lambda(r), \varepsilon_r\rangle$ in Eq. (A.5), we have that

$$\begin{aligned}
\langle \nabla\Psi_\lambda(r), \varepsilon_r\rangle &= \sum_i \frac{\lambda}{w_i}\sinh(\frac{\lambda}{w_i}r_i)\varepsilon_{r,i} \\
&\leqslant \lambda\cdot\sqrt{\sum_i w_i^{-1}\sinh^2(\frac{\lambda}{w_i}r_i)}\sqrt{\sum_i w_i^{-1}\varepsilon_{r,i}^2} \\
&\leqslant \frac{1}{8}\alpha\lambda\sqrt{\sum_i w_i^{-1}\cosh^2(\frac{\lambda}{w_i}r_i)}.
\end{aligned} \tag{A.8}$$

For the second order term $\delta_r^\top \nabla^2 \Psi_\lambda(\widetilde{r})\delta_r$ in Eq. (A.5), 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

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

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. (A.9), we have

$$\begin{aligned}\delta_r^\top \nabla^2 \Psi_\lambda(\widetilde{r})\delta_r &= \lambda^2 \sum_i w_i^{-2}\delta_{r,i}^2 \cosh(\lambda \frac{\widetilde{r}_i}{w_i}) \\ &\leqslant \frac{9\alpha}{8}\lambda^2 \sum_i w_i^{-1}|\delta_{r,i}| \cosh(\lambda \frac{\widetilde{r}_i}{w_i}) \\ &\leqslant \frac{9\alpha}{8}\lambda^2 \sqrt{\sum_i w_i^{-1}\delta_{r,i}^2}\sqrt{\sum_i w_i^{-1}\cosh^2(\lambda \frac{\widetilde{r}_i}{w_i})} \\ &\leqslant (\frac{9\alpha}{8})^2\lambda^2 \left(\sqrt{\sum_i w_i^{-1}\cosh^2(\lambda \frac{\widetilde{r}_i}{w_i})}\right) \\ &\leqslant (\frac{9\alpha}{8})^2\lambda^2 \left(\frac{8}{7}\sqrt{\sum_i w_i^{-1}\cosh^2(\lambda \frac{r_i}{w_i})}\right) \end{aligned} \tag{A.10}$$

where we used Eq. (A.9) at the third last inequality and Lemma A.31 at the second last inequality. Putting Eq. (A.7), Eq. (A.8), and Eq. (A.10) into Eq. (A.5) gives

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

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

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

□

A.4 Gradient Descent on Φ

In the last section, we discussed how to decrease Ψ_λ by changing the input r directly. But our real potential $\Phi^t(x,s) = \Psi_\lambda(\gamma^t(x,s))$ is defined indirectly using (x,s). In this section, we discuss how to design the Newton-like step for (x,s). Note that the non-linear equation Eq. (A.2) 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. (A.2) by moving x and s.

Since our goal is to decrease $\Phi(x,s) = \Psi_\lambda(\gamma)$, similar to Appendix A.3, a natural choice is the steepest descent direction:

$$\delta_\mu^* = \arg\min_{\|\delta_\mu\|_x^*=\alpha} \langle \nabla_\mu \Psi_\lambda(\|\mu_i\|_{x_i}^*), \mu + \delta_\mu\rangle \tag{A.11}$$

with step size α. We can view this as 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

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

Solve Eq. (A.11)[9], we get

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

To move μ to $\mu+\delta_\mu$ approximately, we take Newton step (δ_x^*,δ_s^*)[10]:

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

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

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

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

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

Our robust algorithm only uses $H_{\overline{x}}$, $P_{\overline{x}}$ and $\delta_\mu^*(\overline{x},\overline{s})$ where $(\overline{x},\overline{s})$ is some approximation of (x,s). Formally, our step on x and s is defined in Line 30. Note that we allow for an extra error for (δ_x,δ_s) on top of the error due to $\overline{x}$ and $\overline{s}$. Also, we replace sinh by cosh in the denominator as in Lemma A.8.

[9]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.

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

A.5 Bounding Φ under Changes of x and s

To use Lemma A.8 to bound the potential, we need to verify $|\gamma_i^t(x^{\mathrm{new}}, s^{\mathrm{new}}) - \gamma_i^t(x,s)| \leqslant \frac{w_i}{8\lambda}$ and Eq. (A.4).

A.5.1 Verifying conditions of Lemma A.8

Recall that the ideal step we want to take is

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

where α is the step size. 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. (A.4) should roughly holds. Formally, in Lemma A.13, we prove this holds for the step we take in Algorithm 16. First, we bound the step size for each block $\delta_{x,i}$.

Lemma A.9 (Step size of δ_x)**.** *Let $\alpha_i \stackrel{\mathrm{def}}{=} \|\delta_{x,i}\|_{\overline{x}_i}$, then*

$$\sqrt{\sum_{i=1}^m w_i \alpha_i^2} \leqslant \frac{9}{8}\alpha.$$

In particular, we have $\alpha_i \leqslant \frac{9}{8}\alpha$. Similarly, we have $\sqrt{\sum_{i=1}^m w_i^{-1}(\|\delta_{s,i}\|_{\overline{x}_i}^*)^2} \leqslant \frac{9}{8}\alpha \cdot t$.

Proof. We have

$$\sqrt{\sum_{i=1}^m w_i \alpha_i^2} = \|\delta_x\|_{\overline{x}} \leqslant \|(I - P_{\overline{x}}) H_{\overline{x}}^{-1/2} \delta_\mu\|_2 + \overline{\varepsilon}\alpha \leqslant \|H_{\overline{x}}^{-1/2}\delta_\mu\|_2 + \overline{\varepsilon}\alpha \leqslant \alpha + \overline{\varepsilon}\alpha \leqslant \frac{9}{8}\alpha,$$

where the first inequality follows by the choice that $\delta_x \approx (I - P_{\overline{x}}) H_{\overline{x}}^{-1/2}\delta_\mu$, the second inequality follows by $I - P_{\overline{x}}$ is an orthogonal projection matrix and, second last equality follows by the step size for δ_μ and the last equality follows by $\overline{\varepsilon} \leqslant \frac{1}{8}$.

For δ_s, we note that

$$\sqrt{\sum_{i=1}^m w_i^{-1}(\|\delta_{s,i}\|_{\overline{x}_i}^*)^2} = \|\delta_s\|_{\overline{x}}^* \leqslant \bar{t}\|P_{\overline{x}} H_{\overline{x}}^{-1/2}\delta_\mu\|_2 + \overline{\varepsilon}\alpha\bar{t} \leqslant \bar{t}\|H_x^{-1/2}\delta_\mu\|_2 + \overline{\varepsilon}\alpha\bar{t} \leqslant \frac{9}{8}\alpha t$$

where we used $\bar{t} \leqslant \frac{33}{32}t$ and $\overline{\varepsilon} \leqslant \frac{1}{32}$. □

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

Lemma A.10 (Change in μ)**.** *Let $\mu_i^t(x^{new}, s^{new}) = \mu_i^t(x,s) + \delta_{\mu,i} + \varepsilon_i^{(\mu)}$ with $\beta_i \stackrel{\mathrm{def}}{=} \|\varepsilon_i^{(\mu)}\|_{x_i}^*$, we have $\sqrt{\sum_{i=1}^m w_i^{-1}\beta_i^2} \leqslant 15\overline{\varepsilon}\alpha$.*

Proof. Let $\varepsilon_1 = H_{\overline{x}}^{1/2}\delta_x - (I - P_{\overline{x}})H_{\overline{x}}^{-1/2}\delta_\mu$ and $\varepsilon_2 = \bar{t}^{-1}H_{\overline{x}}^{-1/2}\delta_s - P_{\overline{x}}H_{\overline{x}}^{-1/2}\delta_\mu$. By definition of μ, we have

$$\begin{aligned}
\mu_i^t(x^{\text{new}}, s^{\text{new}}) =& \frac{s_i^{\text{new}}}{t} + w_i\nabla\phi_i(x^{\text{new}}) \\
=& \mu_i^t(x,s) + \frac{1}{t}\delta_s + w_i(\nabla\phi_i(x^{\text{new}}) - \nabla\phi_i(x_i)) \\
=& \mu_i^t(x,s) + \delta_{\mu,i} + \underbrace{w_i(\nabla\phi_i(x^{\text{new}}) - \nabla\phi_i(x_i) - \nabla^2\phi_i(\overline{x}_i)\delta_x)}_{\varepsilon_i^{(\mu,1)}} \\
&+ \underbrace{\left(H_{\overline{x}}^{1/2}(\varepsilon_1+\varepsilon_2)\right)_i}_{\varepsilon_i^{(\mu,2)}} + \underbrace{(\frac{1}{t} - \frac{1}{\bar{t}})\delta_s}_{\varepsilon_i^{(\mu,3)}}
\end{aligned} \tag{A.12}$$

where the last step follows by $\delta_{\mu,i} = \frac{1}{t}\delta_{s,i} + w_i\nabla^2\phi_i(\overline{x}_i)\delta_{x,i} - (w_i\nabla^2\phi_i(\overline{x}_i))^{1/2}(\varepsilon_1+\varepsilon_2)$.

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

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

By Lemma A.6, we have

$$(1 - \|x_i^{(u)} - \overline{x}_i\|_{\overline{x}_i})^2\nabla^2\phi_i(\overline{x}_i) \preceq \nabla^2\phi_i(x^{(u)}) \preceq \frac{1}{(1 - \|x_i^{(u)} - \overline{x}_i\|_{\overline{x}_i})^2}\nabla^2\phi_i(\overline{x}_i). \tag{A.13}$$

Note that

$$\|x_i^{(u)} - \overline{x}_i\|_{\overline{x}_i} \leqslant \|x_i^{(u)} - x_i\|_{\overline{x}_i} + \|x_i - \overline{x}_i\|_{\overline{x}_i} \leqslant u\|\delta_{x,i}\|_{\overline{x}_i} + \bar{\varepsilon} \leqslant \alpha_i + \bar{\varepsilon} \leqslant \frac{9}{8}\alpha + \bar{\varepsilon} \leqslant \frac{25}{16}\bar{\varepsilon},$$

where we used $\|x_i - \overline{x}_i\|_{\overline{x}_i} \leqslant \bar{\varepsilon}$, $\alpha_i \leqslant \frac{9}{8}\alpha$ (Lemma A.9) and $2\alpha \leqslant \bar{\varepsilon}$ (by the algorithm description). Combine two inequalities above and using that $\bar{\varepsilon} \leqslant \frac{1}{8}$, we get

$$-5\bar{\varepsilon}\nabla^2\phi_i(\overline{x}_i) \preceq \nabla^2\phi_i(x^{(u)}) - \nabla^2\phi_i(\overline{x}_i) \preceq 5\bar{\varepsilon}\nabla^2\phi_i(\overline{x}_i). \tag{A.14}$$

Using this, Eq. (A.13) and the algorithm description, we have

$$\begin{aligned}
&\left(\nabla^2\phi_i(x^{(u)}) - \nabla^2\phi_i(\overline{x}_i)\right)\left(\nabla^2\phi_i(x_i)\right)^{-1}\left(\nabla^2\phi_i(x^{(u)}) - \nabla^2\phi_i(\overline{x}_i)\right) \\
\preceq& \frac{1}{(1 - \frac{25}{16}\frac{1}{8})^2}\left(\nabla^2\phi_i(x^{(u)}) - \nabla^2\phi_i(\overline{x}_i)\right)\left(\nabla^2\phi_i(\overline{x}_i)\right)^{-1}\left(\nabla^2\phi_i(x^{(u)}) - \nabla^2\phi_i(\overline{x}_i)\right) \\
\preceq& \frac{(5\bar{\varepsilon})^2}{(1 - \frac{25}{16}\frac{1}{8})^2}\nabla^2\phi_i(\overline{x}_i) \preceq 40\bar{\varepsilon}^2\nabla^2\phi_i(\overline{x}_i).
\end{aligned}$$

This implies

$$\begin{aligned}
&\left\| \left(\nabla^2 \phi_i(x^{(u)}) - \nabla^2 \phi_i(x_i) \right) \delta_{x,i} \right\|_{x_i}^* \\
=&\sqrt{\delta_{x,i}^\top \left(\nabla^2 \phi_i(x^{(u)}) - \nabla^2 \phi_i(\overline{x}_i) \right)^\top \left(\nabla^2 \phi_i(x_i) \right)^{-1} \left(\nabla^2 \phi_i(x^{(u)}) - \nabla^2 \phi_i(\overline{x}_i) \right) \delta_{x,i}} \\
\leqslant&\sqrt{40 \overline{\varepsilon}^2 \delta_{x,i}^\top \nabla^2 \phi_i(\overline{x}_i) \delta_{x,i}^\top} \\
\leqslant&\sqrt{40} \cdot \overline{\varepsilon} \|\delta_{x,i}\|_{\overline{x}_i} \\
=&\frac{9\sqrt{40}}{8} \cdot \overline{\varepsilon} \alpha_i.
\end{aligned}$$

Hence,

$$\|\varepsilon_i^{(\mu,1)}\|_{x_i}^* \leqslant w_i \int_0^1 \left\| \left(\nabla^2 \phi_i(x^{(u)}) - \nabla^2 \phi_i(\overline{x}_i) \right) \delta_{x,i} \right\|_{x_i}^* \mathrm{d}u \leqslant 7.2 \overline{\varepsilon} w_i \alpha_i. \tag{A.15}$$

To bound the term $\varepsilon_i^{(\mu,2)}$ in Eq. (A.12), we use the definition of induced norm (Definition A.5) and Eq. (A.13) to get

$$\begin{aligned}
\sqrt{\sum_i w_i^{-1} (\|\varepsilon_i^{(\mu,2)}\|_{x_i}^*)^2} = \|\varepsilon^{(\mu,2)}\|_x^* = \|H_{\overline{x}}^{1/2} (\varepsilon_1 + \varepsilon_2)\|_x^* \\
\leqslant \frac{1}{1 - \frac{25}{16} \frac{1}{8}} \|H_{\overline{x}}^{1/2} (\varepsilon_1 + \varepsilon_2)\|_{\overline{x}}^* \\
\leqslant 2 \|\varepsilon_1 + \varepsilon_2\|_2 \leqslant 4 \overline{\varepsilon} \alpha.
\end{aligned} \tag{A.16}$$

where we used $\|\varepsilon_1\|_2 \leqslant \overline{\varepsilon}\alpha$ and $\|\varepsilon_2\|_2 \leqslant \overline{\varepsilon}\alpha$ at the end according to the algorithm description.

To bound the term $\varepsilon_i^{(\mu,3)}$ in Eq. (A.12), we note that

$$\begin{aligned}
\sqrt{\sum_i w_i^{-1} (\|(\frac{1}{t} - \frac{1}{\overline{t}}) \delta_{s,i}\|_{x_i}^*)^2} = \frac{1}{t} |\frac{\overline{t} - t}{\overline{t}}| \sqrt{\sum_i w_i^{-1} (\|\delta_{s,i}\|_{x_i}^*)^2} \\
\leqslant \frac{3}{2t} |\frac{\overline{t} - t}{\overline{t}}| \sqrt{\sum_i w_i^{-1} (\|\delta_{s,i}\|_{\overline{x}_i}^*)^2} \\
\leqslant 2 \alpha \varepsilon_t t
\end{aligned} \tag{A.17}$$

where we used Eq. (A.14) on the second inequality, Lemma A.9 $|t - \overline{t}| \leqslant \varepsilon_t \overline{t}$ at the end.

Using $\varepsilon^{(\mu)} = \varepsilon^{(\mu,1)} + \varepsilon^{(\mu,2)} + \varepsilon^{(\mu,3)}$, Eq. (A.15) and Eq. (A.16), we have

$$\sqrt{\sum_i w_i^{-1} (\|\varepsilon_i^{(\mu)}\|_{x_i}^*)^2} \leqslant 7.2 \overline{\varepsilon} \sqrt{\sum_i w_i \alpha_i^2} + 4 \overline{\varepsilon} \alpha + 2 \alpha \varepsilon_t t \leqslant 15 \overline{\varepsilon} \alpha.$$

□

Now, we can check the condition $|r_i - \overline{r}_i| \leqslant \frac{w_i}{8\lambda}$ in Lemma A.8. The following Lemma shows that it is true when $\gamma_i^t(x, s) \leqslant w_i$ for all i, which holds when Φ is small enough.

Lemma A.11. *Assume $\gamma_i^t(x,s) \leqslant w_i$ for all i. For all $i \in [m]$, we have*

$$\|\mu_i^t(x,s) - \mu_i^{\bar{t}}(\overline{x},\overline{s})\|_{x_i}^* \leqslant 3\bar{\varepsilon} w_i.$$

Furthermore, we have that $|\gamma_i^t(x,s) - \gamma_i^{\bar{t}}(\overline{x},\overline{s})| \leqslant 5\bar{\varepsilon} w_i$.

Proof. For the first result, note that

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

Let $x^{(u)} = ux_i + (1-u)\overline{x}_i$. By Eq. (A.14), we have $\nabla^2\phi_i(x_i^{(u)}) \preceq (1+5\bar{\varepsilon})\nabla^2\phi_i(\overline{x}_i) \preceq \frac{5}{8}\nabla^2\phi_i(\overline{x}_i)$ and hence

$$\nabla^2\phi_i(x_i^{(u)})(\nabla^2\phi_i(\overline{x}_i))^{-1}\nabla^2\phi_i(x_i^{(u)}) \preceq \frac{25}{64}\nabla^2\phi_i(\overline{x}_i).$$

Therefore, we have

$$\|\nabla\phi_i(x_i) - \nabla\phi_i(\overline{x}_i)\|_{\overline{x}_i}^* = \left\|\int_0^1 \nabla^2\phi_i(x_i^{(u)})(x_i - \overline{x}_i)\mathrm{d}u\right\|_{\overline{x}_i}^* \leqslant \frac{5}{8}\|x_i - \overline{x}_i\|_{\overline{x}_i}. \tag{A.18}$$

Using $\|s_i - \overline{s}_i\|_{\overline{x}_i}^* \leqslant \bar{t}\bar{\varepsilon} w_i$, we have $\|\mu_i^t(x,s) - \mu_i^t(\overline{x},\overline{s})\|_{\overline{x}_i}^* \leqslant 2\bar{\varepsilon} w_i$.

Finally, we note that $\gamma_i^t(x,s) \leqslant w_i$ and $\|\nabla\phi_i(x_i)\|_{\overline{x}_i}^* \leqslant 2\|\nabla\phi_i(x_i)\|_{x_i}^* \leqslant 2\nu_i$. This implies that

$$\|\frac{s_i}{t} - \frac{s_i}{\bar{t}}\|_{\overline{x}_i} \leqslant (1 - \frac{t}{\bar{t}})\|\frac{s_i}{t}\|_{\overline{x}_i} \leqslant 2(\frac{t-\bar{t}}{\bar{t}})(w_i + \nu_i) \leqslant \frac{1}{2}\bar{\varepsilon} w_i.$$

and hence the result.

For the second result, note that

$$\begin{aligned}
|\gamma_i^t(x,s) - \gamma_i^{\bar{t}}(\overline{x},\overline{s})| &\leqslant \|\mu_i^t(x,s) - \mu_i^{\bar{t}}(\overline{x},\overline{s})\|_{\overline{x}_i}^* + |\|\mu_i^t(x,s)\|_{x_i}^* - \|\mu_i^t(x,s)\|_{\overline{x}_i}^*| \\
&\leqslant 3\bar{\varepsilon} w_i + 2\|x_i - \overline{x}_i\|_{\overline{x}_i}\|\mu_i^t(x,s)\|_{x_i}^* \\
&= 3\bar{\varepsilon} w_i + 2\bar{\varepsilon}\gamma_i^t(x,s) \leqslant 5\bar{\varepsilon} w_i
\end{aligned}$$

where we used the algorithm description and Lemma A.6 □

A.5.2 First Order Approximation of γ

In this subsection, we will show that γ_i is close to $\gamma_i^t(x,s) - \alpha \cdot c_i^t(\overline{x},\overline{s}) \cdot \gamma_i^t(\overline{x},\overline{s})$. First, we need the following helper lemma to bound γ_i, $\sum_{i=1}^m w_i^{-1}\sinh^2(\frac{\lambda}{w_i}\gamma_i^t(\overline{x},\overline{s}))$ and $c(\overline{x},\overline{s})$. In this helper lemma, we assume that Φ is not too large, which is the invariant maintained throughout the algorithm.

Lemma A.12. *Suppose that $\Phi^t(x,s) \leqslant \cosh(\lambda)$, then we have*

- *$\gamma_i^t(x,s) \leqslant w_i$. and $\gamma_i^{\bar{t}}(\overline{x},\overline{s}) \leqslant 2w_i$.*
- *$0 \leqslant c_i^{\bar{t}}(\overline{x},\overline{s}) \leqslant \lambda$.*

Proof. For the first inequality, note that $\Phi^t(x,s) \leqslant \cosh(\lambda)$ implies that $\gamma_i^t(x,s) \leqslant w_i$ for all i. Hence, Lemma A.11 shows that

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

Hence, we have $\gamma_i^{\bar{t}}(\overline{x},\overline{s}) \leqslant 2w_i$.

For the second inequality, we note that

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

Since $\gamma_i \geq 0$ (by definition), we have $c_i^{\bar{t}} \geq 0$. If $\gamma_i^{\bar{t}}(\overline{x},\overline{s}) \geq \frac{w_i}{\lambda}$, we have that

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

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

$$c_i^{\bar{t}}(\overline{x},\overline{s}) \leqslant \frac{2\frac{\lambda}{w_i}\gamma_i^{\bar{t}}(\overline{x},\overline{s})}{\gamma_i^{\bar{t}}(\overline{x},\overline{s}) \cdot \sqrt{\sum_{j=1}^m w_j^{-1}\cosh^2(\frac{\lambda}{w_j}\gamma_j^{\bar{t}}(\overline{x},\overline{s}))}} \leqslant \frac{2\lambda}{w_i\sqrt{4\sum_{j=1}^m w_j^{-1}}} \leqslant \lambda.$$

□

Finally, we can bound the distance between γ^{new} and $\gamma - \alpha c\gamma$. Here we crucially use the fact that $\sinh(x)/x$ is bounded at $x = 0$ and it makes our argument slightly simpler than [LSZ19].

Lemma A.13 (Change in γ)**.** *Assume $\Phi^t(x,s) \leqslant \cosh(\lambda)$. For all $i \in [m]$, let*

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

Then, we have

$$\sqrt{\sum_{i=1}^m w_i^{-1}\varepsilon_{r,i}^2} \leqslant 90\overline{\varepsilon}\lambda\alpha + 4\max_i\left(\frac{\gamma_i^t(x,s)}{w_i}\right)\alpha$$

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

$$\begin{aligned}
\gamma_i^t(x^{\text{new}}, x, s^{\text{new}}) =& \|\mu_i^t(x,s) + \delta_{\mu,i} + \varepsilon_i^{(\mu)}\|_{x_i}^* \\
=& \|\mu_i^t(x,s) - \alpha\overline{c}_i\mu_i^{\bar{t}}(\overline{x},\overline{s})\|_{x_i}^* \pm \|\varepsilon_i^{(\mu)}\|_{x_i}^* \\
=& \|\mu_i^t(x,s) - \alpha\overline{c}_i\mu_i^t(x,s)\|_{x_i}^* \pm \alpha\overline{c}_i \cdot \|\mu_i^t(x,s) - \mu_i^{\bar{t}}(\overline{x},\overline{s})\|_{x_i}^* \pm \|\varepsilon_i^{(\mu)}\|_{x_i}^* \\
=& (1 - \alpha\overline{c}_i)\gamma_i^t(x,s) \pm \alpha\overline{c}_i \cdot \|\mu_i^t(x,s) - \mu_i^{\bar{t}}(\overline{x},\overline{s})\|_{x_i}^* \pm \|\varepsilon_i^{(\mu)}\|_{x_i}^* \qquad \text{(A.19)}
\end{aligned}$$

where we used that $0 \leqslant \alpha\overline{c}_i \leqslant \alpha\lambda \leqslant 1$ at the end Lemma A.12).

In particular, we have that

$$\begin{aligned}\gamma_i^t(x^{\text{new}}, x, s^{\text{new}}) &\leqslant \gamma_i^t(x,s) + \alpha\bar{c}_i\|\mu_i^t(x,s) - \mu_i^{\bar{t}}(\overline{x},\overline{s})\|_{x_i}^* + \|\varepsilon_i^{(\mu)}\|_{x_i}^* \\ &\leqslant \gamma_i^t(x,s) + 4\alpha\bar{c}_i\bar{\varepsilon}w_i + \beta_i \end{aligned} \tag{A.20}$$

where we used Lemma A.11 and Lemma A.10 at the end. Hence, we have

$$\begin{aligned}\left|\gamma_i^t(x^{\text{new}}, x^{\text{new}}, s^{\text{new}}) - \gamma_i^t(x^{\text{new}}, x, s^{\text{new}})\right| &= \left|\|\mu_i^t(x^{\text{new}}, s^{\text{new}})\|_{x_i^{\text{new}}} - \|\mu_i^t(x^{\text{new}}, s^{\text{new}})\|_{x_i}\right| \\ &\leqslant 2\|x_i^{\text{new}} - x_i\|_{x_i}\|\mu_i^t(x^{\text{new}}, s^{\text{new}})\|_{x_i} \\ &\leqslant 3\|\delta_{x,i}\|_{\overline{x}_i}\gamma_i^t(x^{\text{new}}, x, s^{\text{new}}) = 3\alpha_i\gamma_i^t(x^{\text{new}}, x, s^{\text{new}}) \\ &\leqslant 3\alpha_i\gamma_i^t(x,s) + 12\alpha\bar{c}_i\bar{\varepsilon}w_i + 3\beta_i \end{aligned} \tag{A.21}$$

where we used Lemma A.6 on the first inequality, $x_i^{\text{new}} - x_i = \delta_{x,i}$ on the second inequality, the definition of α_i on the second equality, Eq. (A.20) and $\alpha_i \leqslant 1$ on the last inequality.

Using Eq. (A.19), we have

$$\begin{aligned}&\left|\gamma_i^t(x^{\text{new}}, x, s^{\text{new}}) - \gamma_i^t(x,s) + \alpha\bar{c}_i\gamma_i^{\bar{t}}(\overline{x},\overline{s})\right| \\ \leqslant&|(1-\alpha\bar{c}_i)\gamma_i^t(x,s) - \gamma_i^t(x,s) + \alpha\bar{c}_i\gamma_i^{\bar{t}}(\overline{x},\overline{s})| \\ &+ \alpha\bar{c}_i\|\mu_i^t(x,s) - \mu_i^{\bar{t}}(\overline{x},\overline{s})\|_{x_i}^* + \|\varepsilon_i^{(\mu)}\|_{x_i}^* \\ \leqslant&\alpha\bar{c}_i|\gamma_i^t(x,s) - \gamma_i^{\bar{t}}(\overline{x},\overline{s})| + \alpha\bar{c}_i\|\mu_i^t(x,s) - \mu_i^{\bar{t}}(\overline{x},\overline{s})\|_{x_i}^* + \|\varepsilon_i^{(\mu)}\|_{x_i}^* \\ \leqslant&\alpha\bar{c}_i(5\bar{\varepsilon}w_i) + \alpha\bar{c}_i(4\bar{\varepsilon}w_i) + \beta_i \\ \leqslant&9\alpha\bar{c}_i\bar{\varepsilon}w_i + \beta_i \end{aligned} \tag{A.22}$$

where we used Lemma A.11, Lemma A.10 and $\gamma_i^t(x,s) \leqslant w_i$ at the second last inequality.

Combining Eq. (A.21) and Eq. (A.22), we have

$$\begin{aligned}|\varepsilon_{r,i}| &\leqslant \left|\gamma_i^t(x^{\text{new}}, x, s^{\text{new}}) - \gamma_i^t(x,s) + \alpha\bar{c}_i\gamma_i^{\bar{t}}(\overline{x},\overline{s})\right| + \left|\gamma_i^t(x^{\text{new}}, x^{\text{new}}, s^{\text{new}}) - \gamma_i^t(x^{\text{new}}, x, s^{\text{new}})\right| \\ &\leqslant 9\alpha\bar{c}_i\bar{\varepsilon}w_i + \beta_i + 3\alpha_i\gamma_i^t(x,s) + 12\alpha\bar{c}_i\bar{\varepsilon}w_i + 3\beta_i \\ &\leqslant 21\alpha\bar{c}_i\bar{\varepsilon}w_i + 3\alpha_i\gamma_i^t(x,s) + 4\beta_i \end{aligned} \tag{A.23}$$

where we used Lemma A.11 at the end.

Now, we bound the $\|\varepsilon_r\|_{w^{-1}}$. We first note that

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

where we used that $\frac{\sinh^2(x)}{x^2} \leqslant \cosh^2(x)$ for all x at the second last inequality. Using this and $\sum_i w_i \alpha_i^2 \leqslant \frac{9}{8}\alpha^2$ (Lemma A.9) into Eq. (A.23), we have

$$\begin{aligned}\sqrt{\sum_{i=1}^m w_i^{-1}\varepsilon_{r,i}^2} \leqslant & 21\sqrt{2}\alpha\lambda\overline{\varepsilon} + 3\max\left(\frac{\gamma_i^t(x,s)}{w_i}\right)\cdot\sqrt{\sum_{i=1}^m w_i\alpha_i^2} + 4\sqrt{\sum_{i=1}^m w_i^{-1}\beta_i^2} \\ \leqslant & 21\sqrt{2}\alpha\lambda\overline{\varepsilon} + 4\max\left(\frac{\gamma_i^t(x,s)}{w_i}\right)\alpha + 60\alpha\overline{\varepsilon}\end{aligned}$$

where we used $\sqrt{\sum_{i=1}^m w_i\alpha_i^2} \leqslant \frac{9}{8}\alpha$ (Lemma A.9) and $\sqrt{\sum_{i=1}^m w_i^{-1}\beta_i^2} \leqslant 15\overline{\varepsilon}\alpha$ (Lemma A.10) at the end.

□

A.5.3 Bounding the Movement of Φ

After verifying conditions in Lemma A.8, we are ready to bound the change of Φ in one step of (x, s).

Lemma A.14 (Change of Φ after (x, s) step)**.** *Assume $\Phi^t(x,s) \leqslant \cosh(\lambda/64)$. We have*

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

Proof. Let $r_i = \gamma_i^t(x,s)$, $\overline{r}_i = \gamma_i^{\overline{t}}(\overline{x},\overline{s})$ and $\delta_{r,i} = \gamma_i^t(x^{\mathrm{new}}, s^{\mathrm{new}}) - \gamma_i^t(x,s)$. Now, we verify the conditions in Lemma A.8 for r_i, $\overline{r}_i$ and δ_r. Lemma A.11 shows that

$$|r_i - \overline{r}_i| \leqslant 5w_i\overline{\varepsilon} \leqslant \frac{w_i}{8\lambda}$$

where we used the assumption $\overline{\varepsilon} \leqslant \frac{1}{40\lambda}$.

$$\sqrt{\sum_{i=1}^m w_i^{-1}\varepsilon_{r,i}^2} \leqslant 90\alpha\lambda\overline{\varepsilon} + 4\max\left(\frac{\gamma_i^t(x,s)}{w_i}\right)$$

Next, Lemma A.13 shows that

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

with

$$\sqrt{\sum_{i=1}^m w_i^{-1}\varepsilon_{r,i}^2} \leqslant 90\alpha\lambda\overline{\varepsilon} + 4\max\left(\frac{\gamma_i^t(x,s)}{w_i}\right)\alpha \leqslant \frac{90\alpha\lambda}{1440\lambda} + \frac{4}{64}\alpha \leqslant \frac{\alpha}{8}$$

where we used $|\gamma_i^t(x,s)| \leqslant \frac{w_i}{64}$ (due to $\Phi^t(x,s) \leqslant \cosh(\lambda/64)$), $\overline{\varepsilon} \leqslant \frac{1}{1440\lambda}$. Using the formula of $c_i^{\overline{t}}(\overline{x},\overline{s})$, we have

$$\delta_{r,i} = -\frac{\alpha\sinh(\frac{\lambda}{w_i}\gamma_i^{\overline{t}}(\overline{x},\overline{s}))}{\sqrt{\sum_{j=1}^m w_j^{-1}\cosh^2(\frac{\lambda}{w_j}\gamma_j^{\overline{t}}(\overline{x},\overline{s}))}} + \varepsilon_{r,i} = \frac{-\alpha\sinh(\frac{\lambda}{w_i}\overline{r}_i)}{\sqrt{\sum_{j=1}^m w_j^{-1}\cosh^2(\frac{\lambda}{w_j}\overline{r}_j)}} + \varepsilon_{r,i}$$

and this exactly satisfies the conditions in Lemma A.8.

Now, Lemma A.8 shows that

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

□

Now, we bound the change of Φ after changing t.

Lemma A.15 (Change of Φ after t step)**.** *Assume that $\Phi^t(x,s) \leqslant \cosh(\lambda)$. Let $t^{\mathrm{new}} \leftarrow (1-h)t$ for $h \leqslant \frac{1}{8\lambda\sqrt{\max_i \nu_i}}$. We have*

$$\Phi^{t^{\mathrm{new}}}(x,s) \leqslant \Phi^t(x,s) + 16h\lambda\sum_{i=1}^m \sqrt{\nu_i}\cosh(\lambda\gamma_i^{t^{\mathrm{new}}}(x,s)/w_i).$$

Proof. By definition of γ, we have

$$\begin{aligned}\gamma_i^{t^{\mathrm{new}}}(x,s) &= \|\frac{s_i}{t^{\mathrm{new}}} + w_i\nabla\phi_i(x_i)\|_{x_i}^* = \|\frac{s}{t(1-h)} + w_i\nabla\phi_i(x_i)\|_{x_i}^* \\ &\leqslant \frac{1}{1-h}\gamma_i^t(x,s) + \left(\frac{1}{1-h} - 1\right)w_i\|\nabla\phi_i(x_i)\|_{x_i}^* \\ &\leqslant (1+2h)\gamma_i^t(x,s) + 2h\sqrt{\nu_i}w_i \end{aligned} \tag{A.24}$$

where the last inequality follows by the definition of self-concordance, $h \leqslant \frac{1}{8\lambda\sqrt{\max_i \nu_i}}$ and $\nu_i \geq 1$.

For $\Phi^{t^{\mathrm{new}}}$,we have

$$\Phi^{t^{\mathrm{new}}}(x,s) = \sum_{i=1}^m \cosh(\lambda\gamma_i^{t^{\mathrm{new}}}/w_i) \leqslant \sum_{i=1}^m \cosh(\lambda\gamma_i^t/w_i + 2h\lambda(\gamma_i^t/w_i + \sqrt{\nu_i})).$$

Since $\Phi^t(x,s) \leqslant \cosh(\lambda)$, we have $\gamma_i^t/w_i \leqslant 1$. Since we have self-concordance $\nu_i \geq 1$, we have

$$\begin{aligned}\Phi^{t^{\mathrm{new}}}(x,s) &\leqslant \sum_{i=1}^m \cosh(\lambda\gamma_i^t/w_i + 4h\lambda\sqrt{\nu_i}) \\ &\leqslant \Phi^t(x,s) + 8h\lambda\sum_{i=1}^m \sqrt{\nu_i}\cosh(\lambda\gamma_i^t/w_i)\end{aligned}$$

where the last inequality follows by Lemma A.32.

Similar to the argument in Eq. (A.24), we have

$$\gamma_i^{t^{\mathrm{new}}}(x,s) \geq (1+h)\gamma_i^t(x,s) - 2h\sqrt{\nu_i}w_i.$$

Hence, we have $\gamma_i^t(x,s) \leqslant \gamma_i^{t^{\mathrm{new}}}(x,s) + 2h\sqrt{\nu_i}w_i$. By Lemma A.32 again, we have

$$\cosh(\lambda\gamma_i^t/w_i) \leqslant 2\cosh(\lambda\gamma_i^{t^{\mathrm{new}}}/w_i). \tag{A.25}$$

This gives the result.

□

Combining the bound of Φ under (x,s) change (Lemma A.14) and the bound of Φ under t change (Lemma A.15), we get the bound on Φ after 1 step.

Theorem A.16. *Assume $\Phi^t(x,s) \leqslant \cosh(\lambda/64)$. Then for any $0 \leqslant h \leqslant \frac{\alpha}{64\sqrt{\sum_{i=1}^m w_i\nu_i}}$, we have*

$$\Phi^{t^{\mathrm{new}}}(x^{\mathrm{new}}, s^{\mathrm{new}}) \leqslant (1 - \frac{\alpha\lambda}{8\sqrt{\sum_i w_i}})\Phi^t(x,s) + \alpha\lambda\sqrt{\sum_i w_i^{-1}}.$$

In particular, for any $\cosh(\lambda/128) \leqslant \Phi^t(x,s) \leqslant \cosh(\lambda/64)$, we have that

$$\Phi^{t^{\mathrm{new}}}(x^{\mathrm{new}}, s^{\mathrm{new}}) \leqslant \Phi^t(x,s).$$

Proof. By Lemma A.15 and Lemma A.14, we have

$$\begin{aligned}
&\Phi^{t^{\mathrm{new}}}(x^{\mathrm{new}}, s^{\mathrm{new}}) \\
\leqslant&\Phi^{t^{\mathrm{new}}}(x,s) - \frac{\alpha\lambda}{2}\sqrt{\sum_{i=1}^m w_i^{-1}\cosh^2(\frac{\lambda}{w_i}\gamma_i^{t^{\mathrm{new}}}(x,s))} + \alpha\lambda\sqrt{\sum_i w_i^{-1}} \\
\leqslant&\Phi^t(x,s) + 16h\lambda\sum_{i=1}^m \sqrt{\nu_i}\cosh(\frac{\lambda}{w_i}\gamma_i^{t^{\mathrm{new}}}(x,s)) - \frac{\alpha\lambda}{2}\sqrt{\sum_{i=1}^m w_i^{-1}\cosh^2(\frac{\lambda}{w_i}\gamma_i^{t^{\mathrm{new}}}(x,s))} + \alpha\lambda\sqrt{\sum_i w_i^{-1}}.
\end{aligned}$$

By Cauchy Schwarz inequality, we have

$$\begin{aligned}
\frac{\alpha}{4}\sqrt{\sum_{i=1}^m w_i^{-1}\cosh^2(\frac{\lambda}{w_i}\gamma_i^{t^{\mathrm{new}}}(x,s))} \geq&\frac{\alpha}{4}\frac{\sum_{i=1}^m \sqrt{\nu_i}\cosh(\frac{\lambda}{w_i}\gamma_i^{t^{\mathrm{new}}}(x,s))}{\sqrt{\sum_{i=1}^m w_i\nu_i}} \\
\geq&16h\sum_{i=1}^m \sqrt{\nu_i}\cosh(\frac{\lambda}{w_i}\gamma_i^{t^{\mathrm{new}}}(x,s)).
\end{aligned}$$

Hence, we have that

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

where we used Eq. (A.25) at the second inequality.

If $\Phi^t(x,s) \geq \cosh(\lambda/128)$, we have

$$\begin{aligned}
\frac{\Phi^t(x,s)}{8\sqrt{\sum_i w_i}} \geq \frac{\cosh(\lambda/128)}{8\sqrt{\sum_i w_i}} &= \frac{\exp(\lambda/128)}{16\sqrt{\sum_i w_i}} \\
&= \frac{\exp(64\log(256m\sum_i w_i)/128)}{16\sqrt{\sum_i w_i}} \\
&= \frac{16\sqrt{m\sum w_i}}{16\sqrt{\sum_i w_i}} = \sqrt{m} \geq \sqrt{\sum_i w_i^{-1}}.
\end{aligned}$$

Hence, we have $\Phi^{t^{\text{new}}}(x^{\text{new}}, s^{\text{new}}) \leqslant \Phi^t(x, s)$.

□

A.6 Initial Point Reduction

Since Algorithm 16 requires a point on the central path, we modify the convex program to make it happen. To satisfy the constraint $x \in K$, we start the algorithm by solving $\min_{x \in K} c^\top x + t\phi(x)$ for some parameter t. Since x may not satisfy the constraint $Ax = b$, we write $x^{\text{new}} = x^{(1)} + x^{(2)} - x^{(3)}$ where $x^{(1)}$ acts as the original variable and $x^{(2)}, x^{(3)} \in \mathbb{R}^n_{\geq 0}$ are the extra variables. We put a large cost vector on $x^{(2)}$ and $x^{(3)}$ to ensure the solution is roughly the same. The proof shows that if we optimize this new program well enough, we will have $x^{\text{new}} = x^{(1)} + x^{(2)} - x^{(3)} \in K$ and hence x^{new} gives a starting point of the original program. The precise formulation of the modified linear program is as follows:

Definition A.17. Given a convex program $\min_{Ax=b, x \in K} c^\top x$ with inner radius r, outer radius R and Lipschitz constant L. For any t, we define the modified convex program by

$$\min_{(x^{(1)}, x^{(2)}, x^{(3)}) \in \mathcal{P}_t} \left\langle c^{(1)}, x^{(1)} \right\rangle + \left\langle c^{(2)}, x^{(2)} \right\rangle + \left\langle c^{(3)}, x^{(3)} \right\rangle$$

where $\mathcal{P}_t = \{x^{(1)} \in K, (x^{(2)}, x^{(3)}) \in \mathbb{R}^{2n}_{\geq 0} : A(x^{(1)} + x^{(2)} - x^{(3)}) = b\}$, $c^{(1)} = c$, $c^{(2)} = \frac{t}{3R + x_\circ - x_c}$, $c^{(3)} = \frac{t}{3R}$, $x_c = \arg\min_{x \in K} c^\top x + t\phi(x)$ and $x_\circ = \arg\min_{Ax=b} \|x - x_c\|_2$. We define the corresponding dual set by

$$\mathcal{D}_t = \{s^{(1)} \in K^*, (s^{(2)}, s^{(3)}) \in \mathbb{R}^{2n}_{\geq 0} : A^\top y + s^{(1)} = c^{(1)}, A^\top y + s^{(2)} = c^{(2)}, -A^\top y + s^{(3)} = c^{(3)} \text{ for } y \in \mathbb{R}^d\}.$$

We define the corresponding central path problem

$$\min_{(x^{(1)}, x^{(2)}, x^{(3)}) \in \mathcal{P}_t} f_t(x^{(1)}, x^{(2)}, x^{(3)}) \tag{A.26}$$

where $f_t(x^{(1)}, x^{(2)}, x^{(3)}) = \langle c^{(1)}, x^{(1)} \rangle + \langle c^{(2)}, x^{(2)} \rangle + \langle c^{(3)}, x^{(3)} \rangle + t\phi(x^{(1)}) - t\sum_{i=1}^n \log x_i^{(2)} - t\sum_{i=1}^n \log x_i^{(3)}$.

The main result about the modified program is the following.

Theorem A.18. *Given a convex program $\min_{Ax=b, x \in K} c^\top x$ with inner radius r, outer radius R and Lipschitz constant L. For any $0 \leqslant \delta \leqslant \frac{1}{2}$, the modified linear program (Definition A.17) with $t \geq 2^{16}(n + \kappa)^5 \cdot \frac{LR}{\delta} \cdot \frac{R}{r}$ has the following properties:*

- *The point $(x_c, 3R + x_\circ - x_c, 3R)$ is the minimizer of Eq. (A.26). The corresponding s variables are $(-t\nabla\phi(x_c), \frac{t}{3R + x_\circ - x_c}, \frac{t}{3R})$.*
- *Given any primal $(x^{(1)}, x^{(2)}, x^{(3)}) \in \mathcal{P}_t$ and dual $(s^{(1)}, s^{(2)}, s^{(3)}) \in \mathcal{D}_t$ that approximately minimizes $f_{t'}$ at $t' = LR$ as promised by Algorithm 16:*

$$\|s_i^{(1)}/t' + w_i \nabla\phi_i(x_i^{(1)})\|^*_{x_i^{(1)}} \leqslant \frac{w_i}{16} \text{ for all } i \in [m], \tag{A.27}$$

$$x_i^{(j)} s_i^{(j)} \in [1 \pm \frac{1}{16}]t' \text{ for all } i \in [n], j \in \{2, 3\}.$$

Let $x^{\text{new}} = x^{(1)} + x^{(2)} - x^{(3)}$ and $s^{\text{new}} = s^{(1)}$, then we have that $Ax^{\text{new}} = b$, $x^{\text{new}} \in K$, $A^\top y + s^{\text{new}} = c$ for some y and

$$\|s_i^{\text{new}}/t' + w_i \nabla\phi_i(x_i^{\text{new}})\|^*_{x^{\text{new}}} \leqslant \|s_i^{(1)}/t' + w_i \nabla\phi_i(x_i^{(1)})\|^*_{x_i^{(1)}} + \delta.$$

Proof. The proof is separated into Lemma A.19 and Lemma A.26 □

First, we prove the first conclusion in Theorem A.18.

Lemma A.19. *The point* $x \stackrel{\text{def}}{=} (x_c, 3R + x_\circ - x_c, 3R)$ *is the minimizer of* f_t *over* $\mathcal{P}_t$ *(Eq. (A.26)). The corresponding* s *variables are* $(-t\nabla\phi(x_c), \frac{t}{3R+x_\circ-x_c}, \frac{t}{3R})$.

Proof. We will show that $x \in \mathcal{P}_t$ and that it minimizes f_t over $\mathbb{R}^{3n}$, not just $\mathcal{P}_t$.

For the set constraints, we note that $x^{(1)} = x_c \in K$ by the definition of x_c and $x^{(3)} = 3R \geq 0$ by the definition. For $x^{(2)}$, we note that $z \in K$ with $Az = b$ and hence $\|x_\circ - x_c\|_2 \leqslant \|z - x_c\|_2 \leqslant 2R$ (since K has radius R). Hence, $x_i^{(2)} \geq R$ for all $i \in [n]$. Hence, $(x^{(1)}, x^{(2)}, x^{(3)}) \in \mathcal{P}_t$.

For the optimality, we note that

$$\begin{aligned}\nabla_{x^{(1)}} f_t(x) &= c^{(1)} + t\nabla\phi(x^{(1)}) \\ &= c + t\nabla\phi(x_c) = 0\end{aligned}$$

where we used that $x_c = \arg\min_{x\in K} c^\top x + t\phi(x)$. We note that

$$\nabla_{x^{(2)}} f_t(x) = c^{(2)} - \frac{t}{x^{(2)}} = 0$$

and similarly $\nabla_{x^{(3)}} f_t(x) = 0$. Hence x is the minimizer of f_t.

□

Next, we show that the minimizer of $f_{t'}(x)$ for $t' = LR$ is far from the boundary of K for $x^{(1)}$ and has small $x^{(2)}$ and $x^{(3)}$. The proof for both involves the same idea: use the optimality condition of $f_{t'}$. Throughout the rest of the section, we are given $(x^{(1)}, x^{(2)}, x^{(3)}) \in \mathcal{P}_t$ and $(s^{(1)}, s^{(2)}, s^{(3)}) \in \mathcal{D}_t$ satisfying Eq. (A.27). The following lemma shows that $(x^{(1)}, x^{(2)}, x^{(3)})$ is the minimizer of some function g and we use it to prove the properties of x.

Lemma A.20. $(x^{(1)}, x^{(2)}, x^{(3)})$ *is the minimizer of the function*

$$g(x^{(1)}, x^{(2)}, x^{(3)}) \stackrel{\text{def}}{=} \left\langle \widetilde{c}, x^{(1)} \right\rangle + \left\langle c^{(2)}, x^{(2)} \right\rangle + \left\langle c^{(3)}, x^{(3)} \right\rangle + t'\phi(x^{(1)}) - \sum_{i=1}^n \mu_i^{(2)} \log x_i^{(2)} - \sum_{i=1}^n \mu_i^{(3)} \log x_i^{(3)}$$

over the domain $\mathcal{P}_t$ *for some* $\widetilde{c} = c^{(1)} - t'(s^{(1)}/t' + \nabla\phi(x^{(1)}))$, $\frac{15}{16}t' \leqslant \mu_i^{(2)} \leqslant \frac{17}{16}t'$, $\frac{15}{16}t' \leqslant \mu_i^{(3)} \leqslant \frac{17}{16}t'$

Proof. Let $\mu^{(2)} = x^{(2)}s^{(2)}$ and $\mu^{(3)} = x^{(3)}s^{(3)}$. By the definition of $\mathcal{P}_t \times \mathcal{D}_t$, we have that

$$\begin{aligned}\nabla_{x^{(2)}} g(x) &= c^{(2)} - \frac{\mu^{(2)}}{x^{(2)}} = c^{(2)} - s^{(2)} = A^\top y, \\ \nabla_{x^{(3)}} g(x) &= c^{(3)} - \frac{\mu^{(3)}}{x^{(3)}} = c^{(3)} - s^{(3)} = -A^\top y\end{aligned}$$

for some y. For the gradient with respect to $x^{(1)}$, we note that

$$\nabla_{x^{(1)}} g(x) = \widetilde{c} + t'\nabla\phi(x^{(1)}) = c^{(1)} - s^{(1)} = A^\top y.$$

This shows that x satisfies the optimality condition for g, namely $\nabla g(x) = [A, -A, A]^\top y$.

□

The gradient of g is a bit complicated. We avoid it by considering the directional derivative at x on the direction "$z - x$" for some $z \in K$ promised by the definition of inner radius. Since our domain is in $\mathcal{P}_t \subset \mathbb{R}^{3n}$, we need to lift z to higher dimension. Now, we define the point

$$\begin{aligned} z^{(1)} =& z, \\ z^{(2)} =& z^{(3)} = \frac{t'}{t} R. \end{aligned}$$

By construction, we have that $z \in \mathcal{P}_t$. Now, we define the path $p(\beta) = (1-\beta) \cdot (x^{(1)}, x^{(2)}, x^{(3)}) + \beta \cdot (z^{(1)}, z^{(2)}, z^{(3)})$. Since $p(0)$ minimizes g, we have that $\frac{d}{d\beta} g(p(\beta))|_{\beta=0} \geq 0$. In particular, we have

$$\begin{aligned} 0 \leqslant & \frac{d}{d\beta} g(p(\beta))|_{\beta=0} \\ =& (\widetilde{c} + t' \nabla \phi(x^{(1)}))^\top (z^{(1)} - x^{(1)}) \\ & + \sum_{i=1}^n (c_i^{(2)} - \frac{\mu_i^{(2)}}{x_i^{(2)}})(z_i^{(2)} - x_i^{(2)}) + \sum_{i=1}^n (c_i^{(3)} - \frac{\mu_i^{(3)}}{x_i^{(3)}})(z_i^{(3)} - x_i^{(3)}). \end{aligned} \tag{A.28}$$

Now, we bound the terms one by one. To bound the first term in (A.28), we need following lemmas relating the self-concordance barrier and the distance to the boundary.

Lemma A.21 ([Nes98, Theorem 4.1.6, Theorem 4.2.6])**.** *Given a ν-self-concordant barrier ϕ. For any $x, y \in \operatorname{dim} \phi$ such that $\nabla \phi(x)^\top (y - x) \geq 0$, we have $\|y - x\|_x \leqslant \nu + 2\sqrt{\nu}$. In particular, for $x^* = \arg\min_x \phi(x)$, we have*

$$\{x : \|x - x^*\|_{x^*} \leqslant 1\} \subset \operatorname{dom} \phi \subset \{x : \|x - x^*\|_{x^*} \leqslant \nu + 2\sqrt{\nu}\}.$$

Lemma A.22. *Given a ν-self-concordant barrier ϕ for the interval $[\alpha, \beta]$. For any $x, z \in (\alpha, \beta)$, we have that*

$$\sqrt{\phi''(x)} \leqslant \frac{3\nu}{\min(x - \alpha, \beta - x)}$$

and

$$\phi'(x)(z - x) + \frac{1}{16}\sqrt{\phi''(x)}|z - x| \leqslant 4\nu^2 - \frac{1}{16} \max(\frac{z - \alpha}{x - \alpha}, \frac{\beta - z}{\beta - x}).$$

Proof. For the first result, we bound ϕ'' in two case. If $\phi'(x) \geq 0$, then $\phi'(x)(x - \alpha) \geq 0$ and Lemma A.21 shows that $|\alpha - x|\sqrt{\phi''(x)} \leqslant \nu + 2\sqrt{\nu} \leqslant 3\nu$. Hence, we have $\sqrt{\phi''(x)} \leqslant \frac{3\nu}{x - \alpha}$. If $\phi'(x) \leqslant 0$, similar argument shows that $\sqrt{\phi''(x)} \leqslant \frac{3\nu}{\beta - x}$.

For the second result, we split into four cases. First, we note that both sides on the equation is invariant under affine transformation. Hence, we can assume $\alpha = 0$ and $\beta = 1$.

Case 1) $\phi'(x)(z - x) \geq 0$.

Lemma A.21 shows that

$$\sqrt{\phi''(x)}|z - x| \leqslant \nu + 2\sqrt{\nu} \leqslant 3\nu.$$

Together with the fact that $|\phi'(x)| \leqslant \sqrt{\nu \phi''(x)}$, we have

$$\phi'(x)(z - x) + \frac{1}{16}\sqrt{\phi''(x)}|z - x| \leqslant 2\nu^2.$$

Case 2) $x \in [\frac{1}{12\nu}, 1 - \frac{1}{12\nu}]$.

Since $\phi'(x)(z-x) \leqslant 0$ and $z, x \in [0,1]$, we have

$$\phi'(x)(z-x) + \frac{1}{16}\sqrt{\phi''(x)}|z-x| \leqslant \frac{1}{16}\sqrt{\phi''(x)} \leqslant \frac{1}{16} \cdot 36\nu^2 = 3\nu^2$$

where we used the first result at the end.

Case 3) $x \leqslant \frac{1}{12\nu}$

Let $x^* = \arg\min_{x\in[0,1]} \phi(x)$. Lemma A.21 shows that there is an interval $I = [-\gamma, \gamma]$ such that

$$x^* + I \subset [0,1] \subset x^* + (\nu + 2\sqrt{\nu})I \subset x^* + 3\nu I.$$

In particular, this implies that $x^* \in [\frac{1}{6\nu}, 1 - \frac{1}{6\nu}]$. Since $x \leqslant \frac{1}{12\nu}$, we have that $x \leqslant x^* - x$.

Now we use this to show $\phi'(x) \leqslant -\frac{1}{8}\sqrt{\phi''(x)}$. Note that

$$\phi'(x) = \phi'(x^*) - \int_x^{x^*} \phi''(t)dt = -\int_x^{x^*} \phi''(t)dt.$$

Lemma A.6 shows that $[x - \frac{1}{\sqrt{\phi''(x)}}, x + \frac{1}{\sqrt{\phi''(x)}}]$ lies in $\operatorname{dom}\phi$. In particular, this implies that

$$\frac{1}{\sqrt{\phi''(x)}} \leqslant x \leqslant x^* - x \tag{A.29}$$

and hence $x^* \geq x + \frac{1}{\sqrt{\phi''(x)}}$. Hence, we have

$$\begin{aligned}
\phi'(x) &\leqslant -\int_x^{x+(\phi''(x))^{-1/2}/2} \phi''(t)dt \\
&\leqslant -\frac{1}{4}\phi''(x) \cdot \frac{(\phi''(x))^{-1/2}}{2} \\
&= -\frac{1}{8}\sqrt{\phi''(x)}.
\end{aligned}$$

where we used $\phi''(t) \geq \frac{1}{4}\phi''(x)$ for all $|t-x| \leqslant \frac{1}{2\sqrt{\phi''(x)}}$ (Lemma A.6).

Since $\phi'(x)(z-x) \leqslant 0$ and $\phi'(x) \leqslant 0$, we have $z \geq x$ and

$$\begin{aligned}
\phi'(x)(z-x) + \frac{1}{16}\sqrt{\phi''(x)}(z-x) &\leqslant -\frac{1}{16}\sqrt{\phi''(x)}(z-x) \\
&\leqslant -\frac{z-x}{16x} = \frac{1}{16} - \frac{z}{16x}
\end{aligned}$$

where we used $x \geq \frac{1}{\sqrt{\phi''(x)}}$ at the end (Eq. (A.29)).

Case 4) $x \geq 1 - \frac{1}{12\nu}$

By the same argument as case 3, we have $\phi'(x)(z-x) + \frac{1}{16}\sqrt{\phi''(x)}(z-x) \leqslant \frac{1}{16} - \frac{1-z}{16(1-x)}$.

Combining all the cases, we have the result.

□

Now, we can bound the first term in (A.28).

Lemma A.23. *We have that* $(\widetilde{c}+t'\nabla\phi(x^{(1)}))^\top(z^{(1)}-x^{(1)}) \leqslant (6\kappa^2-\frac{r}{16\eta})LR$ *where* η *is the minimum distance between* $x^{(1)}$ *to the boundary of some* K_i*, i.e.* $\eta = \min_i \min_{q\in\partial K_i}\|q-x_i^{(1)}\|_2$.

Proof. Recall that $\widetilde{c} = c^{(1)} - t'\alpha$ with $\alpha = s^{(1)}/t' + \nabla\phi(x^{(1)})$. By the assumption on (x,s), we have that

$$\|\alpha_i\|^*_{x_i^{(1)}} \leqslant \frac{w_i}{16} \text{ for all } i\in[m].$$

Hence, we have

$$\begin{aligned}
&(\widetilde{c}+t'\nabla\phi(x^{(1)}))^\top(z^{(1)}-x^{(1)})\\
=&c^{(1)\top}(z^{(1)}-x^{(1)}) - t'\sum_{i=1}^m \alpha_i^\top(z_i^{(1)}-x_i^{(1)}) + t'\sum_{i=1}^m w_i\nabla\phi_i(x^{(1)})^\top(z_i^{(1)}-x_i^{(1)}).\\
\leqslant&2LR + \frac{t'}{16}\sum_{i=1}^m w_i\|z_i^{(1)}-x_i^{(1)}\|_{x_i^{(1)}} + t'\sum_{i=1}^m w_i\nabla\phi_i(x^{(1)})^\top(z_i^{(1)}-x_i^{(1)}) \qquad \text{(A.30)}
\end{aligned}$$

where we used $\|c^{(1)}\|_2 \leqslant L$ and $\|z^{(1)}-x^{(1)}\|_2 \leqslant 2R$ (the radius of K is bounded by R).

To bound the last two terms, we define $\widetilde{\phi}$ be the ϕ_i restricted on the line between $z_i^{(1)}$ and $x_i^{(1)}$. Note that $\widetilde{\phi}$ is a ν_i-self-concordant barrier function on some interval $[\alpha,\beta]$. Let z and x be the scalar such that $\widetilde{\phi}(z)$ and $\widetilde{\phi}(x)$ corresponding to $\phi_i(z_i^{(1)})$ and $\phi_i(x_i^{(1)})$. Then, we have that

$$\begin{aligned}
u_i \stackrel{\text{def}}{=} \nabla\phi_i(x^{(1)})^\top(z_i^{(1)}-x_i^{(1)}) + \frac{1}{16}\|z_i^{(1)}-x_i^{(1)}\|_{x_i^{(1)}} &= \widetilde{\phi}'(x)(z-x) + \frac{1}{16}\sqrt{\widetilde{\phi}''(x)}|z-x|\\
&\leqslant 4\nu_i^2 - \frac{1}{16}\max(\frac{z-\alpha}{x-\alpha}, \frac{\beta-z}{\beta-x}).
\end{aligned}$$

Let $\eta_i = \min_{q\in\partial K_i}\|q-x_i^{(1)}\|_2$ and q_i be a minimizing q. Suppose $\alpha \leqslant x \leqslant z$ (the other case is similar). Since K_i is convex, there is a hyperplane separating q_i and K_i. Let h be the ℓ_2 distance to the hyperplane. Note that h is linear on K and that $h(\alpha)\geq 0$, $h(z)\geq r$ (because $B(z,r)\subset K_i$). Hence,

$$\eta_i = h(x) = \frac{x-\alpha}{z-\alpha}h(z) + \frac{z-x}{z-\alpha}h(\alpha) \geq \frac{x-\alpha}{z-\alpha}r.$$

Hence, we have

$$\frac{z-\alpha}{x-\alpha} \geq \frac{r}{\eta_i}.$$

This shows $u_i \leqslant 4\nu_i^2 - \frac{r}{16\eta_i}$. In particular, we know that $u_i \leqslant 4\nu_i^2 - \frac{r}{16\eta}$ for one of the i. For other terms, we can simply by it by $4\nu_i^2$. Putting these into Eq. (A.30), we have

$$\begin{aligned}
(\widetilde{c}+t'\nabla\phi(x^{(1)}))^\top(z^{(1)}-x^{(1)}) &\leqslant 2LR + 4t'\sum_{i=1}^m w_i\nu_i^2 - \frac{rt'}{16\eta}\\
&\leqslant 2LR + 4t'\kappa^2 - \frac{rt'}{16\eta}.
\end{aligned}$$

Using $t' = LR$ and $\kappa\geq 1$, we have the result.

□

For the second term and the third term in (A.28), we have the following

Lemma A.24. *We have that*

$$\sum_{i=1}^{n}(c_i^{(j)} - \frac{\mu_i^{(j)}}{x_i^{(j)}})(z_i^{(j)} - x_i^{(j)}) \leqslant 3LRn - \frac{t}{5R}\sum_{i=1}^{n} x_i^{(j)}$$

for both $j = 2$ and 3.

Proof. We only prove the case $j = 2$. The proof for $j = 3$ is similar. As proved in Lemma A.19, $\|x_\circ - x_c\|_2 \leqslant 2R$. Hence $c_i^{(2)} = \frac{t}{3R + x_{\circ,i} - x_{c,i}} \in [\frac{t}{5R}, \frac{t}{R}]$. Hence, we have

$$\begin{aligned}(c_i^{(2)} - \frac{\mu_i^{(2)}}{x_i^{(2)}})(z_i^{(2)} - x_i^{(2)}) =& c_i^{(2)} z_i^{(2)} - \frac{\mu_i^{(2)}}{x_i^{(2)}} z_i^{(2)} - c_i^{(2)} x_i^{(2)} + \mu_i^{(2)} \\ \leqslant& \frac{t}{R}\cdot\frac{t'}{t}R - \frac{t}{5R}\cdot x_i^{(2)} + 2t' \\ \leqslant& 3t' - \frac{t}{5R}\cdot x_i^{(2)}.\end{aligned}$$

Summing over all i and using $t' = LR$ gives the result.

□

Combining (A.28), Lemma A.23 and Lemma A.24, we have

$$0 \leqslant (6\kappa^2 - \frac{r}{16\eta})LR + 6LRn - \frac{t}{5R}\sum_{i=1}^{n}(x_i^{(2)} + x_i^{(3)}).$$

Hence, this shows that $(x^{(1)}, x^{(2)}, x^{(3)})$ satisfies Eq. (A.27) implies that it is far from ∂K (i.e. η is large) and $x^{(2)}, x^{(3)}$ are small:

$$\frac{r}{16\eta} + \frac{t}{5LR^2}\sum_{i=1}^{n}(x_i^{(2)} + x_i^{(3)}) \leqslant 6n + 6\kappa^2.$$

In particular, this shows the following:

Lemma A.25. *We have that $\eta \geq \frac{r}{96(n+\kappa^2)}$ and $\sum_{i=1}^{n}(x_i^{(2)} + x_i^{(3)}) \leqslant 30(n+\kappa^2)\cdot\frac{LR}{t}\cdot R$.*

Now, we are ready to prove the second conclusion of Theorem Theorem A.18.

Lemma A.26. *Let $x^{\text{new}} = x^{(1)} + x^{(2)} - x^{(3)}$ and $s^{\text{new}} = s^{(1)}$, then we have that $Ax^{\text{new}} = b$, $x^{\text{new}} \in K$, $A^\top y + s^{\text{new}} = c$ for some y and*

$$\|s_i^{\text{new}}/t' + w_i\nabla\phi_i(x_i^{\text{new}})\|^*_{x_i^{\text{new}}} \leqslant \|s_i^{(1)}/t' + w_i\nabla\phi_i(x_i^{(1)})\|^*_{x_i^{(1)}} + \delta.$$

Proof. Note that $Ax^{\text{new}} = b$ by definition. Lemma A.25 shows that $x^{(1)}$ is $\eta \geq \frac{r}{96(n+\kappa^2)}$ far from ∂K. Since $x^{\text{new}} = x^{(1)} + x^{(2)} - x^{(3)}$, we have

$$\|x^{\text{new}} - x^{(1)}\|_2 \leqslant \|x^{(2)}\|_1 + \|x^{(3)}\|_1 \leqslant 30(n+\kappa^2)\cdot\frac{LR}{t}\cdot R$$

where we used $x^{(2)}, x^{(3)} \geq 0$ and Lemma A.25. Hence, we have that

$$\frac{\|x^{\text{new}} - x^{(1)}\|_2}{\eta} \leqslant 2^{12}(n+\kappa)^4 \cdot \frac{R}{r} \cdot \frac{LR}{t} < 1$$

by the choice of t. In particular, this shows that $x^{\text{new}} \in K$.

Next, $A^\top y + s^{\text{new}} = A^\top y + s^{(1)} = c$ by construction.

Finally, to bound $s/t + w\nabla\phi$, we note that Lemma A.22 shows that $\nabla^2\phi_i(x^{(1)}) \preceq \frac{9\nu^2}{\eta^2}$. This gives

$$\begin{aligned}
\|x_i^{\text{new}} - x_i^{(1)}\|_{x_i^{(1)}} &\leqslant \frac{3\nu}{\eta} \cdot \|x_i^{\text{new}} - x_i^{(1)}\|_2 \\
&\leqslant 3\nu \cdot (\frac{r}{96(n+\kappa^2)})^{-1} \cdot 30(n+\kappa^2) \cdot \frac{LR}{t} \cdot R \\
&\leqslant 2^{14}(n+\kappa)^5 \cdot \frac{LR}{t} \cdot \frac{R}{r} \leqslant \frac{\delta}{4}
\end{aligned}$$

where we used our choice of δ. Using this and Lemma A.6 gives $\|v\|_{x_i^{\text{new}}} \leqslant (1+\frac{\delta}{2})\|v\|_{x_i^{(1)}}$ and $\|\nabla\phi_i(x_i^{\text{new}}) - \nabla\phi_i(x_i^{(1)})\|^*_{x_i^{(1)}} \leqslant \frac{\delta}{2}$. Hence

$$\begin{aligned}
&\|s_i^{\text{new}}/t' + w_i\nabla\phi_i(x_i^{\text{new}})\|^*_{x_i^{\text{new}}} \\
\leqslant&(1+\frac{\delta}{2})\|s_i^{\text{new}}/t' + w_i\nabla\phi_i(x_i^{\text{new}})\|^*_{x_i^{(1)}} \\
=&(1+\frac{\delta}{2})\|s_i^{(1)}/t' + w_i\nabla\phi_i(x_i^{(1)}) + (\nabla\phi_i(x_i^{\text{new}}) - \nabla\phi_i(x_i^{(1)}))\|^*_{x_i^{(1)}} \\
\leqslant&(1+\frac{\delta}{2})\|s_i^{(1)}/t' + w_i\nabla\phi_i(x_i^{(1)})\|^*_{x_i^{(1)}} + (1+\frac{\delta}{2})\|\nabla\phi_i(x_i^{\text{new}}) - \nabla\phi_i(x_i^{(1)})\|^*_{x_i^{(1)}} \\
\leqslant&\|s_i^{(1)}/t' + w_i\nabla\phi_i(x_i^{(1)})\|^*_{x_i^{(1)}} + \delta.
\end{aligned}$$

□

A.7 Main Result

To prove Theorem A.1, we first need the following lemma showing that the iterate x is a good solution when t is small enough.

Lemma A.27 ([LSZ19, Lemma D.3]). *Let $\phi_i(x)$ be a ν_i-self-concordant barrier for K_i. Suppose we have $\|\frac{s_i}{t} + \nabla\phi_i(x_i)\|^*_{x_i} \leqslant 1$ for $i \in [m]$, $A^\top y + s = c$ and $Ax = b$. Then, we have*

$$c^\top x \leqslant \min_{Ax=b, x\in\prod_{i=1}^m K_i} c^\top x + 4t\sum \nu_i.$$

Theorem A.1. *Consider the convex program Eq. (CP). Given ν_i-self-concordant barriers $\phi_i : K_i \to \mathbb{R}$ with its minimum x_i. Define the following parameters of the convex problem:*

1. *Inner radius r: There exists a z such that $Az = b$ and $B(z, r) \subset K$.*
2. *Outer radius R: We have $K \subset B(x, R)$ for some $x \in \mathbb{R}^n$.*
3. *Lipschitz constant L: $\|c\|_2 \leqslant L$.*

Let $w \in \mathbb{R}^m_{\geq 1}$ be any weight vector, and $\kappa = \sum_{i=1}^m w_i \nu_i$. For any $0 < \varepsilon \leqslant 1/2$, Algorithm 16 outputs an approximate solution x in $O(\sqrt{\kappa}\log(m)\log(\frac{n\kappa R}{\varepsilon r}))$ steps, such that $Ax = b$, $x \in K$ and

$$c^\top x \leqslant \min_{Ax=b,\, x\in K} c^\top x + \varepsilon LR.$$

Proof. Theorem A.18 gives an explicit point on the central path. Hence, we have $\Phi^t(x,s) = m \leqslant \cosh(\lambda/128)$ initially. Theorem A.16 shows that $\Phi^t(x,s) \leqslant \cosh(\lambda/128)$ throughout the first call of Centering. After we obtain the approximate central path point $((x^{(1)}, x^{(2)}, x^{(3)}), (s^{(1)}, s^{(2)}, s^{(3)}))$ at $t = LR$ for the modified convex program, Theorem A.18 shows that $(x^{(1)} + x^{(2)} - x^{(3)}, s^{(1)})$ is an approximate central path point at $t = LR$ for the original convex program. Furthermore, γ_i^t is increased by $\delta = \frac{1}{128}$ for all i. Hence, Φ^{LR} is increased by at most $\exp(\frac{\lambda}{128})$ factor. Hence, we have $\Phi^{LR}(x,s) \leqslant \cosh(\lambda/64)$. Now, Theorem A.16 shows that $\Phi^t(x,s) \leqslant \cosh(\lambda/64)$ throughout the second call of Centering.

Now, we verify the output. Note that $A\delta_x = 0$ and $\delta_s \in \mathrm{Im}A^\top$. Hence, throughout the algorithm, we have $Ax = b$ and $c - s \in \mathrm{Im}A^\top$. Finally, for the optimality, we note that $w_i\phi_i$ are $w_i\nu_i$ self-concordant. Lemma A.27 shows that

$$c^\top x' \leqslant \min_{Ax=b, x\in\prod_{i=1}^m K_i} c^\top x + 4t_{\mathrm{end}} \sum_{i=1}^m w_i\nu_i.$$

Since the algorithm terminates at $t_{\mathrm{end}} = \varepsilon/(4\sum_{i=1}^m w_i\nu_i)$, we have the error bounded.

□

A.8 Using the Universal Barrier

In this subsection, we discuss the case if the barriers $\phi_i : K_i \to \mathbb{R}$ is not given. In this case, we can use the universal barrier, which has self concordance n_i.

Theorem A.28 ([NN94, LY18])**.** *For any convex set K, the universal barrier function $\phi(x) = \log \mathrm{Vol}(K^\circ(x))$ is a n self-concordant barrier where $K^\circ(x) = \{y \in \mathbb{R}^n : y^\top(z - x) \leqslant 1, \forall z \in K\}$.*

The gradient and Hessian of the universal barrier function ϕ can be computed using the center of gravity and the covariance of $K^\circ(x)$.

Lemma A.29 ([LY18, Lemma1])**.** *For any convex set $K \subset \mathbb{R}^n$ and any $x \in \mathrm{int}(K)$, we have*

$$\nabla\phi(x) = -(n+1)\mu(K^\circ(x)),$$
$$\nabla^2\phi(x) = (n+1)(n+2)\mathrm{Cov}(K^\circ(x)) + (n+1)\mu(K^\circ(x))\mu(K^\circ(x))^\top.$$

where $\mu(K^\circ(x))$ is the center of gravity of $K^\circ(x)$ and $\mathrm{Cov}(K^\circ(x))$ is the covariance matrix of $K^\circ(x)$.

Computing center of gravity and covariance takes polynomial time. See for example [LV18] for a survey.

Theorem A.30 ([DFK91, SV13])**.** *Given a membership oracle for a convex set $K \subset \mathbb{R}^n$ with cost $\mathcal{T}$. Assuming $B(0,r) \subset K \subset B(0,R)$, we can compute x and A such that*

$$\|x - \mu(K)\|_{\mathrm{Cov}(K)^{-1}} \leqslant \varepsilon \qquad \text{and} \qquad (1-\varepsilon)A \preceq \mathrm{Cov}(K) \preceq (1+\varepsilon)A$$

in time $O(n^{O(1)}\mathcal{T}\log(R/r)/\varepsilon^2)$.

Next, note that the membership oracle of $K^\circ(x)$ involves optimizing one linear function over the convex set k and it can be done using membership oracle of K and the ellipsoid method. Therefore, for any x, we can compute an approximate gradient g and the Hessian H of the universal barrier function such that

$$\|g - \nabla\phi(x)\|_{\nabla^2\phi(x)^{-1}} \leqslant \varepsilon \qquad \text{and} \qquad (1-\varepsilon)H \preceq \nabla^2\phi(x) \leqslant (1+\varepsilon)H$$

in time $O(n^{O(1)}\mathcal{T}\log(R/r)/\varepsilon^2)$ where $\mathcal{T}$ is the cost of the membership oracle of K.

Finally, we note that as long as $\varepsilon \leqslant \frac{1}{\log^c m}$ for some large enough c, our robust interior point method works with those approximate gradient and the Hessian with the same guarantee. Since the proof is essentially same, we skip the analysis here. We note there are known explicit barrier functions with good self-concordance for most commonly used convex sets and in this case, we do not need heavy machinery like the above to compute them.

A.9 Hyperbolic Function Lemmas

Lemma A.31. *For any $x, y \in \mathbb{R}$ with $|y| \leqslant \frac{1}{8}$, we have*

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

Similarly, we have $|\cosh(x+y) - \cosh(x)| \leqslant \frac{1}{7}\cosh(x)$.

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

For the second result, note that $\cosh(x+y) = \cosh(x)\cosh(y) + \sinh(x)\sinh(y)$. Hence,

$$\begin{aligned}|\cosh(x+y) - \cosh(x)| &\leqslant (\cosh(y) - 1)\cosh(x) + \sinh(x)\sinh(y) \\ &\leqslant (\cosh(y) - 1 + |\sinh(y)|)\cosh(x) \\ &\leqslant \frac{1}{7}\cosh(x).\end{aligned}$$

□

Lemma A.32. *For any $x \geq 0$ and $0 \leqslant y \leqslant 1$, we have*

$$\cosh(x+y) \leqslant (1+2y)\cosh(x)$$

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

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

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

B Treewidth vs. Problem Size in Netlib Instances

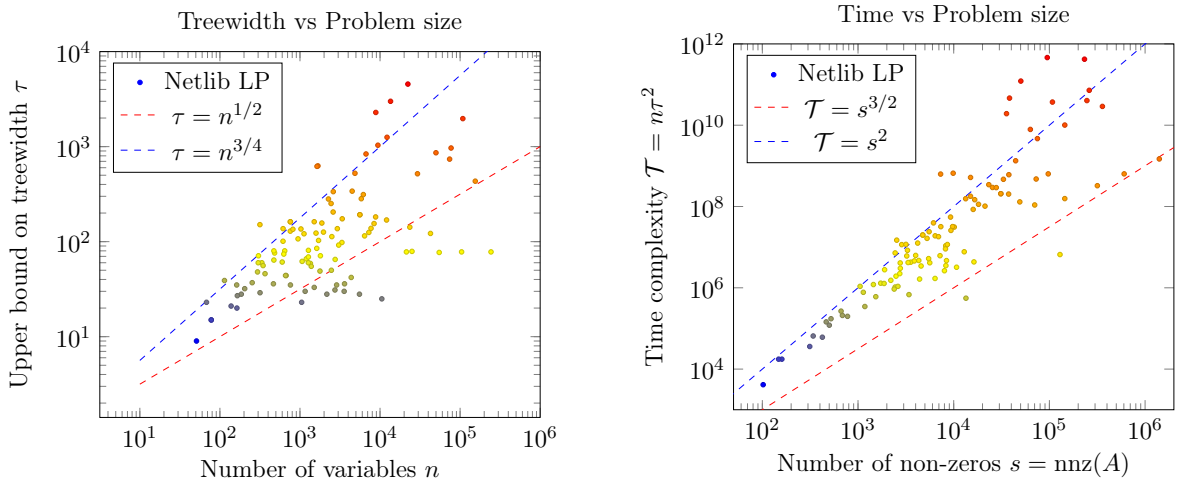

Figure B.1: The left plot shows some upper bound of treewidth vs d for all 109 feasible linear program instances in Netlib repository. We compute a upper bound of treewidth using [KK98]. This shows that treewidth is between $n^{1/2}$ and $n^{3/4}$ for many linear programs in this data set. The right plot shows that the runtime $n\tau^2$ is sub-quadratic in the input size $\text{nnz}(A)$ for many linear programs in this data set.