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

Sally Dong
sallyqd@uw.edu
University of Washington
USA

Yin Tat Lee
yintat@uw.edu
University of Washington
& Microsoft Research
USA

Guanghao Ye
ghye@uw.edu
University of Washington
USA

ABSTRACT

Arising from structural graph theory, treewidth has become a focus of study in fixed-parameter tractable algorithms. Many NP-hard problems are known to be solvable in $\widetilde{O}(n \cdot 2^{O(\tau)})$ time, where τ is the treewidth of the input graph. Analogously, many problems in P should be solvable in $\widetilde{O}(n \cdot \tau^{O(1)})$ time; however, due to the lack of appropriate tools, only a few such results are currently known. In our paper, we show this holds for linear programs:

Given a linear program of the form $\min_{Ax=b,\ell \leqslant x \leqslant u} c^\top x$ whose dual graph G_A has treewidth τ, and a corresponding width-τ tree decomposition, 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. When a tree decomposition is not given, we use existing techniques in vertex separators to obtain algorithms with $\widetilde{O}(n \cdot \tau^4 \log(1/\varepsilon))$ and $\widetilde{O}(n \cdot \tau^2 \log(1/\varepsilon) + n^{1.5})$ run-times. Besides being the first of its kind, our algorithm has run-time nearly matching the fastest run-time 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. This representation further yields the first IPM with $o(\text{rank}(A))$ time per iteration when the treewidth is small.

CCS CONCEPTS

• **Theory of computation** → **Linear programming**; **Fixed parameter tractability**.

KEYWORDS

linear program, interior point method, treewidth, parameterized complexity

*The full version of this paper is available as [18] at https://arxiv.org/abs/2011.05365.

STOC '21, June 21–25, 2021, Virtual, Italy

ACM ISBN 978-1-4503-8053-9/21/06…$15.00
https://doi.org/10.1145/3406325.3451056

ACM Reference Format:
Sally Dong, Yin Tat Lee, and Guanghao Ye. 2021. A Nearly-Linear Time Algorithm for Linear Programs with Small Treewidth: A Multiscale Representation of Robust Central Path. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC '21), June 21–25, 2021, Virtual, Italy*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3406325.3451056

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 [15], ellipsoid methods [32] and interior point methods [30], 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 [12, 29] or $\widetilde{O}((\sqrt{nd} \cdot \text{nnz}(A) + d^{2.5}) \log(1/\varepsilon))$ time [8, 51], 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 $\widetilde{O}((\text{nnz}(A) + d^\omega) \log(1/\varepsilon))$ 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 run-times 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 [41], 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 Matlab Netlib repository have sublinear treewidth. For low treewidth linear systems, a small polynomial dependence on treewidth still implies a much faster than quadratic run-time, hence making them a particularly suitable target of study.

Beyond the practical consideration, whether there is a $\widetilde{O}(n\tau^{O(1)})$ LP algorithm is important in parameterized complexity. Most algorithms designed for low treewidth graphs rely on dynamic programming, which naturally give algorithms with run-time exponential in treewidth, even for problems in P, such as reachability and shortest paths [1, 10, 11, 42]. There are only a few

[1]The current fastest exact algorithms for linear program take either $2^{O(\sqrt{d \log \frac{n-d}{d}})}$ time [25], or the run-time depends on the magnitude of entries of A.

problems in P that we know how to solve in $\widetilde{O}(n\tau^{O(1)})$ time [21]. We refer to sections 1.2.1 and 1.2.2 for discussion of these problems.

Recently, [21] posed exactly this question[2]:

Can linear programs be solved in $\widetilde{O}(n \cdot \mathrm{tw}^{O(1)} \log(1/\varepsilon))$ 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* $\{1,\dots,d\}$, *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 diameter of the polytope, namely, for any* $\ell \leqslant x \leqslant u$ *with* $Ax = b$, *we have* $\|x\|_2 \leqslant R$.

Then, for any $0 < \varepsilon \leqslant 1$, *we can find* $\ell \leqslant x \leqslant u$ *such that*

$$c^\top x \leqslant \min_{Ax=b,\ell\leqslant x\leqslant u} c^\top x + \varepsilon \cdot \|c\|_2 \cdot R \quad \text{and}$$
$$\|Ax-b\|_2 \leqslant \varepsilon \cdot (\|A\|_2 \cdot R + \|b\|_2)$$

in expected time

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

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

Our algorithm involves a pre-processing component: We need to find some suitable reordering of the rows of A, given by 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 [3, 20, 24] and nested dissection algorithms [22, 31, 37]. In theory, we rely on the latter, where the elimination order is computed by recursively finding small balanced vertex separators in G_A. When a tree decomposition is given, it is relatively straightforward; otherwise, we draw on two existing results on vertex separators to directly compute the ordering: One is [9] which computes a crude approximate separator. Another one is [4] which reduces the computation to the maximum flow problem. Applying these two results to theorem 1.1 implies the following:

Theorem 1.2. *Let* τ *denote the treewidth of* G_A. *Without a given tree decomposition, the runtime in theorem 1.1 becomes either*

$$\widetilde{O}(n \cdot \tau^4 \log(1/\varepsilon)),$$

or

$$\widetilde{O}(n \cdot \tau^2 \log(1/\varepsilon) + \mathcal{T}_{\text{maxflow}}),$$

where $\mathcal{T}_{\text{maxflow}}$ *is the time it takes to compute a weighted maximum flow on an auxiliary directed graph with* $O(|V(G_A)|)$ *vertices,* $O(|E(G_A)|)$ *edges, edge capacities bounded by* $O(|V(G_A)|)$, *and treewidth* $O(\tau)$.

[2] We add the $\log(1/\varepsilon)$ term into their original conjecture. Without this term, this conjecture will imply the existence of strongly polynomial time algorithm 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 [21, 26]). We adopt the one used in the ILP community [19, 26]. 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.

Using $\mathcal{T}_{\text{maxflow}} = \widetilde{O}(m + n^{1.5})$ [8], our algorithm takes $\widetilde{O}(n \cdot \tau^2 \log(1/\varepsilon) + n^{1.5})$ when the treewidth τ is large. This is faster than the current fastest algorithm for linear programs for the case $\tau \leqslant n^{0.68}$ with $d = \Theta(n)$; this is common among instances in practice.

Detailed discussions can be found in literature (e.g. [43] and [34, 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 $L = \log(1 + d_{\max} + \|c\|_2)$ 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, $L = O(\log(n + \|b\|_2 + \|c\|_2))$.

Finally, we note that the notion of treewidth for a linear program is robust: Introducing a new constraint increases treewidth by at most one, and introducing a new variable increases treewidth by a factor of at most $O(\log d)$ after reformulation. This property is particularly useful for problems with a standard combinatorial structure plus a small number of additional restrictions. For example, finding a max-flow with limited flow on a subgraph would simply require an additional constraint to the standard max-flow formulation, and the runtimes would not be severely affected in our framework.

1.1 Convex Generalization

Theorems 1.1 and 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* $A_{j,r} \neq \mathbf{0}$. *Suppose that*

- *we are given a tree decomposition of* G_A *with width* τ,
- R *is the diameter of the set* $\{Ax = b\} \bigcap \prod_{i\in[m]} K_i$,
- $n_i = O(1)$ *for all* $i \in [m]$,
- *we are given initial points* $x_i \in \mathbb{R}^{n_i}$ *for each* $i \in [m]$, *and there is* $r_i \in \mathbb{R}$ *and some* $M > 0$ *such that* $B(x_i, r_i) \subset K_i \subset B(x_i, Mr_i)$ *for each* i,
- *we can check if* $y \in K_i$ *in* $O(1)$ *time for all* $i \in [m]$.

Then, for any $0 < \varepsilon \leqslant 1$, *we can find* $x \in \prod_{i\in[m]} K_i$ *such that*

$$c^\top x \leqslant \min_{Ax=b, x_i\in K_i \text{ for } i\in[m]} c^\top x + \varepsilon \cdot \|c\|_2 \cdot R$$
$$\text{and} \qquad \|Ax-b\|_2 \leqslant \varepsilon \cdot (\|A\|_2 \cdot R + \|b\|_2)$$

in expected time

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

The proof for the convex program and the linear program is almost identical. Any operation pertaining to the entry $A[i, j]$ in the linear programming case is generalized to operations pertaining to the $1 \times n_j$ submatrix of A from row i and block j. Since each block has size $O(1)$, the overall run-time 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 [47]: 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 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 treewidth $O(1)$. Therefore, our result shows that this problem can be solved in nearly-linear time, without relying on the specific formula or structure.

1.2 Related Works

1.2.1 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 run-times that depend only linearly on the problem size and exponentially on treewidth [5] 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 [38]; it appears to be one of the first algorithms to exploit treewidth in a completely different way from dynamic programming.

1.2.2 Algorithms with Runtime Polynomial to Treewidth. When the problem is linear algebraic, such as solving linear systems and computing rank/determinant, the dynamic programming approaches often leads to runtime polynomial to treewidth.

For linear systems $Ax = b$, George first developed the method of nested dissection in [22], 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 [37], to solving systems where A is any symmetric positive-definite matrix whose underlying graph has good balanced vertex separators. This was further extended by [2], 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 [16], 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, [21] 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 [21], 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, [33] 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, [53] 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.

1.2.3 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 $\mathrm{td}(A)$, [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 [14], 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. [19] also discusses how an algorithm for ILP may be used to solve LP, [7] 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 [30]. After multiple running time improvements [12, 30, 35, 36, 40, 43,

48, 50, 51], the current fastest IPMs are the results of [8] and [27]. 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 use ideas similar to wavelets commonly found in signal processing [44], 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 [39]. 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 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 $s(t)$, 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 end 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 [30], and has led to significant recent progress in convex optimization [8, 12, 28, 29, 49–51]

In section 6, we give our robust central path algorithm (algorithm 1), which is a slight variant of the one presented in [36]. The changes are needed to support some extra approximation required by our new representation. theorem 6.1 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. If the algorithm updates lazily, there are only a nearly linear number of coordinate changes to $\overline{x}$ and $\overline{s}$ during the whole algorithm:

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

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 $H_{\overline{x}}$.

2.2 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 usually is 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, however, 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 section 2.1, we have the following:

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

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 (sections 2.1 and 2.2), 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.

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 $\tilde{O}(n^{1.25}\tau^{O(1)}\log(1/\varepsilon))$ (See [52] 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 $v^\top\mathcal{W}^\top h = h^\top\mathcal{W}v = h^\top L^{-1}AH_{\overline{x}}^{-1/2}v$, 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 7.6), based on the elimination tree $\mathcal{T}$, to store a family of sketches of the form $\Phi\mathcal{W}^\top h$. In particular, $\mathcal{S}$ 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:

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

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, for a change in $\overline{x}_i$, $H_{\overline{x}}^{-1/2}$ changes by a single entry, and the value of $\Phi_{\chi(v)}\mathcal{W}^\top h$ changes 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 ever 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 section 2.4, we have:

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

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 $O(\log n)$.

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 taimes $\widetilde{O}(n \cdot \tau)$ time and produces an elimination tree of height $\widetilde{O}(\tau)$. Otherwise, it takes $\widetilde{O}(n \cdot \text{tw}(G_A)^4)$ time to obtain a tree of height $\widetilde{O}(\text{tw}(G_A)^2)$, or $\widetilde{O}(n \cdot \text{tw}(G_A)^2 + \mathcal{T}_{\text{maxflow}})$ time for a tree of height $\widetilde{O}(\text{tw}(G_A))$, as needed in the two respective cases of theorem 1.2.

We can reduce the linear program of theorem 1.1 to a convex program of the form eq. (CP), before invoking theorem 6.1 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 6.1 directly. When the barrier functions are not given, we use the universal barrier ϕ_i with self-concordance n_i for each i; 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 6.1 shows that the robust interior point method given as algorithm 1 produces the approximate solution as required, and terminates within $O(\sqrt{\kappa}\log(\kappa/\varepsilon)) = O(\sqrt{n}\log(1/\varepsilon))$ steps. To begin, algorithm 1 makes a small reduction in order to find an initial point (x, s) for the central path (line 15); it is clear this change (section 6.1) can be performed in linear time, and does not affect the order of d, m, n or the elimination tree. In the parameters of algorithm 1, the central path itself begins at timestep $t_{\max} = 1$ and terminates at some $t_{\min} \geq \varepsilon^2/(360n^4\kappa) = O(\varepsilon^2/n^5)$ (line 17).

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 7.1. Let τ denote the height of the elimination tree $\mathcal{T}$ computed during preprocessing, and let $N = O(\sqrt{n}\log(n/\varepsilon))$ 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 1 needs to find $\overline{x}, \overline{s}, \overline{t}$ and compute updates to x, s, t (lines 18 to 23), which is all accomplished invoking MultiplyAndMove(t). As MultiplyAndMove is called N times over the entire algorithm, the total run-time 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 $O(\log mM)$ time by remark 6.2. Hence, we incur an additional $\log(M)$ factor in the overall run-time.

3 NOTATIONS

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 mean 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. For a PSD matrix $A \in \mathbb{R}^{n\times n}$, we let $\|v\|_A = \sqrt{v^\top Av}$.

We use e_i to denote the standard unit vector. We use $\mathbf{0}_n, \mathbf{1}_n$ to denote the all-zero and all-one vectors in $\mathbb{R}^n$. We use $\mathbf{0}_{m\times n}$ and $\mathbf{1}_{m\times n}$ analogously. We use $I_n \in \mathbb{R}^n$ 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 columns (blocks) indexed 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 $A_1 \in \mathbb{R}^{n_1 \times n_1}$, $A_2 \in \mathbb{R}^{n_2 \times n_2}$, and $A_m \in \mathbb{R}^{n_m \times n_m}$.

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 $S \subseteq \mathcal{P}$.

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 [6, 16] 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 three cases for the run-time, corresponding to theorem 1.1 with a given tree decomposition, and the two cases in theorem 1.2 respectively.

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 run-times 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. $\text{height}(\mathcal{T}) = \widetilde{O}(\tau)$.
(2) $\widetilde{O}(n \cdot tw(G_A)^4)$ without a given tree decomposition. $\text{height}(\mathcal{T}) = \widetilde{O}(tw(G_A)^2)$, *where $tw(G_A)$ is the treewidth of G_A.*
(3) $\widetilde{O}(\mathcal{T}_{\text{maxflow}})$ without a given tree decomposition, where $\mathcal{T}_{\text{maxflow}}$ is the time required to compute the max flow on an auxilliary directed graph with $O(d)$ vertices and $\widetilde{O}(tw(G_A))$ treewidth[4]. $\text{height}(\mathcal{T}) = \widetilde{O}(tw(G_A))$.

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 Treewidth and Vertex Separator

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

Lemma 4.2. *Any block of A with sparsity τ induces a clique of size τ in G_A. It follows that* $\max\{|\mathcal{A}_i| : A_i \text{ a column block of } A\} \leqslant tw(A) + 1$. □

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

Theorem 4.3 ([6], Lemma 6)**.** *If G is a graph with treewidth τ, then there exists a 1/2-balanced separator of G of size at most $\tau + 1$.* □

4.2 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 $AH^{-1}H^\top$. 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 $\pi(v)$.

Definition 4.4 (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 4.5 ([45])**.** *Let L be the Cholesky factor for the matrix $AH^{-1}A^\top$. Let L_j denote the j-th column. 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.*

[4] 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.

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

Theorem 4.6 ([6], Theorem 12). *Every graph G on n vertices satisfies*

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

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 $V(G_A)$.

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 run-time for their computations and maintenance. All of these properties are known (see textbooks [16, 23] for more complete introductions). We include some algorithms and proofs to familiarize readers for the techniques we will use.

Let $\mathcal{P}(i)$ denote the path from vertex i to the root 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 $\mathcal{T}$.*

Lemma 5.2 ([45, 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. *$\mathcal{L}^i \subseteq \mathcal{D}(i)$, where $\mathcal{D}(i)$ is the set of all vertices in the subtree rooted at i (including i) in $\mathcal{T}$.*

5.1 Solving Triangular Systems

Now, we discuss the cost of solving triangular systems.

Lemma 5.4. *Let $x = L^{-1}v$, and let S be the non-zero pattern of v. Then, the nonzero pattern of x is a subset of $\bigcup_{i \in S} \mathcal{P}(i)$. 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.*

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

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.*

5.2 Computing and Updating the Cholesky factorization

By analyzing the number of non-zeros operations of Cholesky factorization algorithms), one can show the following:

Lemma 5.7 ([23, 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 $AH_{\bar{x}}^{-1}A^\top = LL^\top$ can be computed in $O(n\tau^2)$ time.*

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 [17], which makes a further sparsity assumption. We include the proof of first lemma for intuition.

Lemma 5.9 ([17, 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 S be the index set of columns of L that are updated, i.e. $S = \{j \in [d] \mid \Delta L_j \neq \mathbf{0}\}$, then 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 $\text{nnz}(\Delta L) = O(\tau^2)$.*

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

Lemma 5.10 ([17, 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 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 6.1. *Consider the convex program eq. (CP). Given ν_i-self-concordant barriers $\phi_i : K_i \to \mathbb{R}$ with its minimum x_i. Assume that*

(1) Lipschitz constant of the program: $\|c\|_2 \leqslant L$.
(2) Diameter of the polytope: For any $x \in \prod_{i=1}^m K_i$, we have $\|x\|_2 \leqslant R$.

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

$$c^\top x \leqslant \min_{Ax=b,\, x\in\prod_{i=1}^m K_i} c^\top x + LR\cdot\varepsilon$$
$$\|Ax-b\|_2 \leqslant \varepsilon\cdot(R\|A\|_2 + \|b\|_2)$$
$$x \in \prod_{i=1}^m K_i.$$

Remark 6.2. If the barrier functions ϕ_i is not given, we can use $w_i = 1$ and universal barrier functions ϕ_i for K_i. In this case, the algorithm takes $O(\sqrt{n}\log(\kappa/\varepsilon))$ steps, and the cost of computing a good enough approximation of $\nabla\phi_i$ and $\nabla^2\phi_i$ both takes $n_i^{O(1)}\log(mM)^{O(1)}$ 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) $B(x_i, r_i) \subset K_i \subset B(x_i, Mr_i)$ for some given x_i and r_i.

Our algorithm and the proof is similar to [36], the main difference being that we do not use an approximate t in the formula, which is needed for our main 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 [36] 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 [35] (beyond the case $n_i = 1$).

6.1 Initial Point Reduction

To generate an initial feasible point for the central path, we follow the reduction from [36].

Lemma 6.3. *Given a convex program eq. (CP), suppose for each $i \in [m]$, we are given a ν_i-self concordant barrier function ϕ_i for K_i, and $x^{(0)} = \arg\min_x \sum \phi_i(x_i)$, where $x_i \in K_i$. Assume that*

(1) Diameter of the polytope: For any $x \in \prod_{i=1}^m K_i$, we have $\|x\|_2 \leqslant R$.
(2) Lipschitz constant of the program: $\|c\|_2 \leqslant L$.

Then, for any $0 < \varepsilon \leqslant 1$, the modified convex program

$$\min \quad \bar{c}^\top \bar{x} \qquad (\overline{CP})$$
$$s.t. \qquad \bar{A}\bar{x} = \bar{b}, \quad \bar{x} \in \prod_{i=1}^m K_i \times \mathbb{R}^d_+$$

with

$$\bar{A} = [A \mid D(b - Ax^{(0)})],\ \bar{b} = b \text{ and } \bar{c} = \begin{bmatrix} \frac{\varepsilon}{LR}\cdot c \\ \frac{1}{d}\mathbf{1} \end{bmatrix}$$

where $D(b-Ax^{(0)})$ is the diagonal matrix with $b - Ax^{(0)}$ on the diagonal, satisfies the following:

(1) $\bar{x} = \begin{bmatrix} x^{(0)} \\ \mathbf{1} \end{bmatrix}$ and $\bar{y} = \mathbf{0}_d$ are feasible primal and dual vectors, with slack $\bar{s} = \begin{bmatrix} \frac{\varepsilon}{LR}\cdot c \\ \mathbf{1} \end{bmatrix}$, and $\|\bar{s} + \nabla\bar{\phi}(\bar{x})\|^*_{\bar{x}} \leqslant \varepsilon$, where $\bar{\phi}(\bar{x}) = \sum_{i=1}^m \phi_i(\bar{x}_i) - \sum_{i=m+1}^{m+d}\log(\bar{x}_i)$.

Algorithm 1 A Robust Interior Point Method for eq. (CP)

▷ Constants
$A \in \mathbb{R}^{d\times n}$, $w \in \mathbb{R}^n_{\geq 1}$
ν_i self-concordant barrier functions $\phi_i : \mathbb{R}^{n_i} \to \mathbb{R}$
$\lambda = \log(256m\sum_{i=1}^m w_i)$, $\bar{\varepsilon} = \frac{1}{1440\lambda}$, $\alpha = \frac{\bar{\varepsilon}}{2}$, $\varepsilon_t = \frac{\bar{\varepsilon}}{4}(\min_i \frac{w_i}{w_i+\nu_i})$, $h = \frac{\alpha}{64\sqrt{\sum_{i=1}^m w_i\nu_i}}$
procedure InteriorPointMethod($\varepsilon > 0$)
  ▷ Definitions
  $\mu_i^t(x,s) \stackrel{\text{def}}{=} s_i/t + w_i\nabla\phi_i(x_i)$, $\gamma_i^t(x,s) \stackrel{\text{def}}{=} \|\mu_i^t(x,s)\|^*_{x_i}$
  $c_i^t(x,s) \stackrel{\text{def}}{=} \frac{\sinh(\frac{\lambda}{w_i}\gamma_i^t(x,s))}{\gamma_i^t(x,s)\cdot\sqrt{\sum_{j=1}^m w_j^{-1}\cosh^2(\frac{\lambda}{w_i}\gamma_j^t(x,s))}}$
  $\Psi_\lambda(r) \stackrel{\text{def}}{=} \sum_{i=1}^m \cosh(\lambda r_i/w_i)$, $\Phi^t(x,s) \stackrel{\text{def}}{=} \Psi_\lambda(\gamma^t(x,s))$
  $\phi(x) \stackrel{\text{def}}{=} \sum_{i=1}^m w_i\phi_i(x_i)$, $H_x \stackrel{\text{def}}{=} \nabla^2\phi(x)$
  $P_x \stackrel{\text{def}}{=} H_x^{-1/2}A^\top(AH_x^{-1}A^\top)^{-1}AH_x^{-1/2}$

  ▷ Initialization
  $t \leftarrow 1$
  $\varepsilon \leftarrow \min(\varepsilon, \frac{1}{6\log(m)})$
  Modify the convex program and obtain an initial x, s according to lemma 6.3

  ▷ Main Loop
  while $t > \frac{\varepsilon^2}{360n^4\sum_i \nu_i w_i}$ **do**
    Find $\overline{x}, \overline{s}, \overline{t}$ such that $\|\overline{x}_i - x_i\|_{\overline{x}_i} \leqslant \bar{\varepsilon}$, $\|\overline{s}_i - s_i\|^*_{\overline{x}_i} \leqslant t\bar{\varepsilon}w_i$ and $|t-\overline{t}| \leqslant \varepsilon_t\overline{t}$
    for $i \in [m]$ **do**
      $\delta_{\mu,i} \leftarrow -\alpha\cdot c_i^{\overline{t}}(\overline{x},\overline{s})\cdot\mu_i^{\overline{t}}(\overline{x},\overline{s})$
    end for
    Find δ_x, δ_s with $\|\varepsilon_1\|^*_{\overline{x}} \leqslant \bar{\varepsilon}\alpha$ and $\|\varepsilon_2\|^*_{\overline{x}} \leqslant \bar{\varepsilon}\alpha$ such that
    $$\delta_x = H_{\overline{x}}^{-1/2}(I - P_{\overline{x}})H_{\overline{x}}^{-1/2}(\delta_\mu + \varepsilon_1),$$
    $$\delta_s = tH_{\overline{x}}^{1/2}P_{\overline{x}}H_{\overline{x}}^{-1/2}(\delta_\mu + \varepsilon_2),$$
    $t \leftarrow (1-h)t$, $x \leftarrow x + \delta_x$, $s \leftarrow s + \delta_s$
  end while
  Return an approximate solution of the original convex program according to lemma 6.3
end procedure

(2) For any $\bar{x}$ such that $\bar{A}\bar{x} = \bar{b}$, $\bar{x} \in \prod_{i=1}^m K_i \times \mathbb{R}_+$, and $\bar{c}^\top\bar{x} \leqslant \overline{OPT} + \frac{\varepsilon^2}{10n^4}$, the vector $\bar{x}_{1:n}$ is an approximate solution to the original convex program (CP) in the following sense:

$$c^\top\bar{x}_{1:n} \leqslant \min_{Ax=b,\, x\in\prod_{i=1}^m K_i} c^\top x + LR\cdot\varepsilon$$
$$\|A\bar{x}_{1:n} - b\|_2 \leqslant \varepsilon\cdot(R\|A\|_2 + \|b\|_2)$$
$$\bar{x}_{1:n} \in \prod_{i=1}^m K_i.$$

The proof follows directly from Lemma D.2 of [36], with the exception that they have one additional variable x_{m+1} in $\bar{x}$ and a dense column $b - Ax^{(0)}$ concatenated to A, whereas we have d

additional variables $x_{m+1}, \ldots, x_{m+d}$ in $\bar{x}$, and correspondingly the diagonal matrix $D(b - Ax^{(0)})$.

Finally, observe that $\mathrm{tw}(A) = \mathrm{tw}(\bar{A})$, since concatenating a diagonal matrix to A does not change the adjacency relations in the graph.

7 ROBUST CENTRAL PATH MAINTENANCE

We present a data structure CentralPathMaintenance to efficiently perform the robust central path step needed in lines 18 to 23 of algorithm 1:

Theorem 7.1 (Robust Central Path Step)**.** *Suppose algorithm 1 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 $\bar{\lambda}, \bar{\varepsilon}, \varepsilon_t, \alpha, w = \mathbf{1}_m$ as defined in algorithm 1, there is a randomized data structure* Central-PathMaintenance *that implicitly maintains the central path primal-dual solution pair (x, s) (algorithm 1 line 23) and explicitly maintains its approximation $(\overline{x}, \overline{s})$ (algorithm 1 line 22) 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 run-time tuning parameter k in $O(n\tau^2 \log^4(n))$ time.*
- *MultiplyAndMove(t): It implicitly maintains*

$$\begin{aligned} x &\leftarrow x + H_{\overline{x}}^{-1/2}(I - P_{\overline{x}})H_{\overline{x}}^{-1/2}\delta_\mu(\overline{x}, \overline{s}, \overline{t}) \\ s &\leftarrow s + tH_{\overline{x}}^{1/2}P_{\overline{x}}H_{\overline{x}}^{-1/2}\delta_\mu(\overline{x}, \overline{s}, \overline{t}) \end{aligned} \tag{7.1}$$

  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 $\overline{t}$ is some earlier timestep satisfying $|t - \overline{t}| \leqslant \varepsilon_t \cdot \overline{t}$.
  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 7.2. The N dependence in the run-time 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.

7.1 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. (7.1), as well as the approximation pair. Here, we introduce the multiscale representation used in these computations:

Definition 7.3 (Multiscale Basis)**.** At any step of the robust central path with approximate primal-dual solution pair $(\overline{x}, \overline{s})$, we define

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

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

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

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. (7.1) using the multiscale representation:

$$\begin{aligned} x &\leftarrow x + H_{\overline{x}}^{-1}\delta_\mu(\overline{x}, \overline{s}, \overline{t}) - H_{\overline{x}}^{-1/2}\mathcal{W}^\top h \\ s &\leftarrow s + tH_{\overline{x}}^{1/2}\mathcal{W}^\top h. \end{aligned} \tag{7.2}$$

7.2 Implicit Representation of (x, s)

For the first part of proof of theorem 7.1, 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. (7.2), 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 7.5. *Given constraint matrix A and its binary elimination tree $\mathcal{T}$ with height τ, the data structure* MultiscaleRepresentation *(algorithms 2 and 3) implicitly maintains the primal-dual pair (x, s) as defined by eq. (7.2), computable via the expression*

$$\begin{aligned} x &= \widehat{x} + H_{\overline{x}}^{-1/2}\beta_x c_x - H_{\overline{x}}^{-1/2}\mathcal{W}^\top(\beta_x h + \varepsilon_x) \\ s &= \widehat{s} + H_{\overline{x}}^{1/2}\mathcal{W}^\top(\beta_s h + \varepsilon_s), \end{aligned} \tag{7.3}$$

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. (7.2) 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 $S = \{i \in [m] \mid \overline{x}_i^{\text{new}} \neq \overline{x}_i \text{ or } \overline{s}_i^{\text{new}} \neq \overline{s}_i\}$. Then each call to Update takes $O(|S| \cdot \tau^2)$ time, and each variable in eq. (7.3) except $\mathcal{W}$ changes in $O(|S| \cdot \tau)$ many entries.

7.3 Approximating A Sequence of Vectors

The central path maintenance involves a number of dynamic vectors from eq. (7.3). These can essentially be viewed as online sequences of vectors, where the sequence length is the number of central

Algorithm 2 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. (7.3)
  $\overline{\alpha} \in \mathbb{R}, \overline{\delta_\mu} \in \mathbb{R}^n$ ▷ Implicit representation of δ_μ
  $\overline{t} \in \mathbb{R}_+$ ▷ 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 $\mathcal{T}$
  INITIALIZE$h(\overline{x}, \overline{s}, H_{\overline{x}}, L_{\overline{x}})$
end procedure
procedure INITIALIZE$h(\overline{x}, \overline{s}, H_{\overline{x}}, L_{\overline{x}})$
  for $i \in [m]$ **do**
    $(\overline{\delta_\mu})_i \leftarrow -\frac{\alpha \sinh(\frac{\lambda}{w_i}\gamma_i(\overline{x},\overline{s},\overline{t}))}{\gamma_i(\overline{x},\overline{s},\overline{t})} \cdot \mu_i(\overline{x}, \overline{s}, \overline{t})$
    ▷ λ, w, γ, μ as defined in algorithm 1
    $\overline{\alpha} \leftarrow \overline{\alpha} + \alpha^2 \cdot w_i^{-1}\cosh^2(\frac{\lambda}{w_i}\gamma_i(\overline{x}, \overline{s}, \overline{t}))$
  end for
  $c_x \leftarrow H_{\overline{x}}^{-1/2}\overline{\delta_\mu}$
  $h \leftarrow L_{\overline{x}}^{-1}AH_{\overline{x}}^{-1}\overline{\delta_\mu}$
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 3 Multiscale Representation Data Structure - Update

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

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 run-time. The underlying idea is standard in sampling, heavy-hitters, and sketching, see e.g. [13]. We explain how it is used in the context of central path maintenance in subsequent sections.

Definition 7.6. 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 7.7. *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 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* $v \in \mathcal{S}$,
$\mathcal{O}[y^{(i)}]$.TypeII(j): *access to entry* $y^{(i)}_j$ *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 $O(\eta \cdot \zeta^2 k^2/\varepsilon^2_{\text{apx}} \cdot \text{poly}\log(nk\zeta/(\varepsilon_{\text{apx}} \cdot \delta_{\text{apx}})))$ *type-I oracle calls and* $O(\zeta^2 k^2/\varepsilon^2_{\text{apx}} \cdot \text{poly}\log(nk\zeta/(\varepsilon_{\text{apx}} \cdot \delta_{\text{apx}})))$ *type-II oracle calls, with* $O(\eta \cdot r \cdot \zeta^2 k^2/\varepsilon^2_{\text{apx}} \cdot \text{poly}\log(nk\zeta/(\varepsilon_{\text{apx}} \cdot \delta_{\text{apx}})))$ *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* $1 - \delta_{\text{apx}}$.

7.4 Sketching A Sequence of Vectors

In this section, we show how to construct an oracle used in theorem 7.7 that supports type-I queries for a sequence of vectors.

Theorem 7.8. *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* VectorSketch *maintains* $y_v \overset{\text{def}}{=} \Phi_{\chi(v)}h$ *for all nodes* v *in the sampling tree through the following functions:*

- 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.
- 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.*
- Query($v \in V(\mathcal{S})$): *Outputs* $\Phi_{\chi(v)}h$ *in* $O(r)$ *time.*

7.5 Sketching the Multiscale Representation

The previous section shows how to construct an oracle used in theorem 7.7 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. (7.3).

Consider maintaining ℓ_∞-approximations of the sequence of $\mathcal{W}^\top h$: Using VectorSketch presented in theorem 7.8 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, we present sketching data structures that serve as the oracle needed in theorem 7.7 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.

Theorem 7.9. *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 7.5 with height* $O(\log n)$, *there is a data structure* BalancedSketch *that maintains* $\Phi_{\chi(v)}\mathcal{W}^\top h$ *for each* $v \in V(\mathcal{S})$ *through the following operations:*

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

Algorithm 4 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
  ▷ sketches indexed by nodes of $\mathcal{S}$, where $y_v \overset{\text{def}}{=} \Phi_{\chi(v)}h$
  List $\{y_v\}_{v\in V(\mathcal{S})}$
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

- 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* $\overline{x}^{\text{new}}$.
- Query(v): *Outputs* $\Phi_{\chi(v)}\mathcal{W}^\top h$ *in* $O(\tau^2 \cdot r)$ *time.*

Balanced Sampling Tree Construction. First, we construct a sampling tree of $\mathbb{R}^d$, denoted by $(\mathcal{B}, \overline{\chi})$, as follows: We perform heavy-light decomposition [46] on the elimination tree $\mathcal{T}$ with vertex set $[d]$. Let $\sigma_1, \ldots, \sigma_d$ denote the vertices ordered. 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 7.6.

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

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

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

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

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

Algorithm 5 Robust Central Path Maintenance – Initialize, MultiplyAndMove, Output

data structure CentralPathMaintenance **extends** MultiscaleRepresentation
private : member
  ▷ maintains $\mathcal{W}^\top \varepsilon_x$, $\mathcal{W}^\top \varepsilon_s$, and $\mathcal{W}^\top h$, theorem 7.9
  BalancedSketch Sketch$\mathcal{W}^\top \varepsilon_x$, Sketch$\mathcal{W}^\top \varepsilon_s$, Sketch$\mathcal{W}^\top h$
  ▷ maintains $H_{\overline{x}}^{1/2}\widehat{x}$, $H_{\overline{x}}^{-1/2}\widehat{s}$, and $H_{\overline{x}}^{-1/2}c_x$, theorem 7.8
  VectorSketch Sketch$H^{1/2}\widehat{x}$, Sketch$H^{-1/2}\widehat{s}$, Sketch$H^{-1/2}c_x$
  ▷ maintains ℓ_∞-approx of $H_{\overline{x}}^{1/2}x$ and $H_{\overline{x}}^{-1/2}s$, theorem 7.7
  ℓ_∞–Approximates Approx$H_{\overline{x}}^{1/2}x$, Approx$H_{\overline{x}}^{-1/2}s$
  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 7.5.
  InitializeSketch()
  $\ell \leftarrow 0$
  ▷ setting the appropriate approximation tolerances
  $\varepsilon_{\text{apx}}^{(x)} \leftarrow \frac{\overline{\varepsilon}}{\max_i n_i}, \zeta^{(x)} \leftarrow 2\alpha, \delta_{\text{apx}} \leftarrow \frac{N}{20k}$
  $\varepsilon_{\text{apx}}^{(s)} \leftarrow \frac{\overline{\varepsilon}\cdot\overline{t}}{2\max_i n_i}, \zeta^{(s)} \leftarrow 2\alpha\overline{t}$ ▷ α, n_i as in algorithm 1
  Approx$H_{\overline{x}}^{1/2}x$.Initialize$(\mathcal{S}, \chi, \varepsilon_{\text{apx}}^{(x)}, \delta_{\text{apx}}, \zeta^{(x)}, k)$
  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 O_x, O_s for the ℓ_∞-Approximates data structures
  $\overline{x}^{\text{new}} \leftarrow H_{\overline{x}}^{-1/2}\cdot$Approx$H_{\overline{x}}^{1/2}x$.Query$(O_x\{H_{\overline{x}}^{1/2}\widehat{x} + \beta_x\cdot c_x - \mathcal{W}^\top(\beta_x h + \varepsilon_x)\})$
  $\overline{s}^{\text{new}} \leftarrow H_{\overline{x}}^{1/2}\cdot$Approx$H_{\overline{x}}^{-1/2}s$.Query$(O_s\{H_{\overline{x}}^{-1/2}\widehat{s} + \mathcal{W}^\top(\beta_s h + \varepsilon_s)\})$
  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 6 Robust Central Path Maintenance – Helper Functions

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

Oracle $O_x\{H_{\overline{x}}^{1/2}\widehat{x} + \beta_x\cdot c_x - \mathcal{W}^\top(\beta_x h + \varepsilon_x)\}$
procedure TypeI($v \in \mathcal{S}$)
  return
  Sketch$H^{1/2}\widehat{x}$.Query(v) + $\beta_x\cdot$ Sketch$H^{-1/2}c_x$.Query(v) − $\beta_x\cdot$ Sketch$\mathcal{W}^\top h$.Query(v) − 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 $O_s\{H_{\overline{x}}^{-1/2}\widehat{s} + \mathcal{W}^\top(\beta_s h + \varepsilon_s)\}$
procedure TypeI($v \in \mathcal{S}$)
  return Sketch$H^{-1/2}\widehat{s}$.Query(v)+
    $\beta_s\cdot$ Sketch$\mathcal{W}^\top h$.Query(v) + 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

check that for any leaf node $v \in \mathcal{B}$,

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

Let this $(\mathcal{S}, \chi)$ be our *balanced sampling tree.*

ACKNOWLEDGMENTS

We thank Aaron Sidford for discussing the optimization on thick path. We thank Anup B. Rao for discussing the convex regression problem. We thank Haotian Jiang and the anonymous reviewers for their helpful suggestions. 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

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

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

[3] Patrick R Amestoy, Timothy A Davis, and Iain S Duff. 1996. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.* 17, 4 (1996), 886–905.

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

[5] Hans L Bodlaender. 1994. A tourist guide through treewidth. (1994).

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

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

[8] Jan van den Brand, Yin-Tat Lee, Yang P. Liu, Danupon Nanongkai, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. 2020. Maximum Flow in Nearly-linear Time on Moderately Dense Graphs. *Personal communication* (2020).

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

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

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

[12] Michael B. Cohen, Yin Tat Lee, and Zhao Song. 2019. 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*. 938–942. https://doi.org/10.1145/3313276.3316303

[13] Graham Cormode and S. Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *J. Algorithms* 55, 1 (2005), 58–75. https://doi.org/10.1016/j.jalgor.2003.12.001

[14] Jana Cslovjecsek, Friedrich Eisenbrand, Christoph Hunkenschröder, Lars Rohwedder, and Robert Weismantel. 2020. Block-Structured Integer and Linear Programming in Strongly Polynomial and Near Linear Time. arXiv:2002.07745 [cs.CC]

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

[16] Timothy A. Davis. 2006. *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898718881

[17] Timothy A. Davis and William W. Hager. 2003. Modifying a Sparse Cholesky Factorization. *SIAM J. Matrix Anal. Appl.* 20, 3 (2003), 606–627. https://doi.org/10.1137/s0895479897321076

[18] Sally Dong, Yin Tat Lee, and Guanghao Ye. 2020. A Nearly-Linear Time Algorithm for Linear Programs with Small Treewidth: A Multiscale Representation of Robust Central Path. *CoRR* abs/2011.05365 (2020). arXiv:2011.05365 https://arxiv.org/abs/2011.05365

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

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

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

[22] Alan George. 1973. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.* 10, 2 (1973), 345–363.

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

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

[25] Thomas Dueholm Hansen and Uri Zwick. 2015. 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*. 209–218.

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

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

[28] Haotian Jiang, Yin Tat Lee, Zhao Song, and Sam Chiu-wai Wong. 2020. 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*. 944–953.

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

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

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

[32] Leonid G Khachiyan. 1980. Polynomial algorithms in linear programming. *U. S. S. R. Comput. Math. and Math. Phys.* 20, 1 (1980), 53–72.

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

[34] Yin Tat Lee and Aaron Sidford. 2013. Path Finding I: Solving Linear Programs with\~O (sqrt (rank)) Linear System Solves. *arXiv preprint arXiv:1312.6677* (2013).

[35] Yin Tat Lee and Aaron Sidford. 2019. Solving Linear Programs with Sqrt(rank) Linear System Solves. *CoRR* abs/1910.08033 (2019). arXiv:1910.08033 http://arxiv.org/abs/1910.08033

[36] Yin Tat Lee, Zhao Song, and Qiuyi Zhang. 2019. Solving Empirical Risk Minimization in the Current Matrix Multiplication Time. In *Conference on Learning Theory, COLT 2019, 25-28 June 2019, Phoenix, AZ, USA*. 2140–2157. http://proceedings.mlr.press/v99/lee19a.html

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

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

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

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

[41] Richard Peng and Santosh Vempala. 2020. Solving Sparse Linear Systems Faster than Matrix Multiplication. arXiv:2007.10254 [cs.DS]

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

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

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

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

[46] Daniel Dominic Sleator and Robert Endre Tarjan. 1983. A Data Structure for Dynamic Trees. *J. Comput. Syst. Sci.* 26, 3 (1983), 362–391. https://doi.org/10.1016/0022-0000(83)90006-5

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

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

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

[50] Jan van den Brand, Yin-Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. 2020. Bipartite Matching in Nearly-linear Time on Moderately Dense Graphs. arXiv:2009.01802 [cs.DS]

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

[52] Guanghao Ye. 2020. Fast Algorithm for Solving Structured Convex Programs.

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