Solving Second-Order Cone Programs Deterministically in Matrix Multiplication Time

Michelle Wei[*] Guanghao Ye[†]

Abstract

We propose a deterministic algorithm for solving second-order cone programs of the form

$$\min_{Ax=b, x\in\mathcal{L}_1\times\cdots\times\mathcal{L}_r} c^\top x,$$

which optimize a linear objective function over the set of $x \in \mathbb{R}^n$ contained in the intersection of an affine set and the product of r second-order cones. Our algorithm achieves a runtime of

$$\widetilde{O}((n^\omega + n^{2+o(1)}r^{1/6} + n^{2.5-\alpha/2+o(1)})\log(1/\epsilon)),$$

where ω and α are the exponents of matrix multiplication, and ϵ is the relative accuracy. For the current values of $\omega \sim 2.37$ and $\alpha \sim 0.32$, our algorithm takes $\widetilde{O}(n^\omega \log(1/\epsilon))$ time. This nearly matches the runtime for solving the sub-problem $Ax = b$. To the best of our knowledge, this is the first improvement on the computational complexity of solving second-order cone programs after the seminal work of Nesterov and Nemirovski on general convex programs. For $\omega = 2$, our algorithm takes $\widetilde{O}(n^{2+o(1)}r^{1/6}\log(1/\epsilon))$ time.

To obtain this result, we utilize several new concepts that we believe may be of independent interest:

- We introduce a novel reduction for splitting ℓ_p-cones.
- We propose a deterministic data structure to efficiently maintain the central path of interior point methods for general convex programs.

[*]michelle.wei89@gmail.com. PRIMES-USA.
[†]ghye@mit.edu. MIT.

1 Introduction

Second-order cone programming (SOCP) is a fundamental class of problems in mathematical optimization. The widespread applications of SOCP extend to various areas such as machine learning, operations research, robust optimization, and combinatorial optimization, spanning several industries including information technology, finance, energy, and transportation [LVBL98, AG03]. SOCP provides a generalized representation of several well-known convex optimization problems, including linear programs (LP), convex quadratic programs (QP), and quadratically constrained convex quadratic programs (QCQP).

SOCPs represent a class of problems that aim to optimize a linear objective within the intersection of the Cartesian product of second-order cones and an affine space. The standard definition of a SOCP includes n variables within r second-order cones and is subject to m constraints, as stated below:

Definition 1.1 (Second-order Cone Program)**.** Given the constraint matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, two vectors $\boldsymbol{b} \in \mathbb{R}^m$ and $\boldsymbol{c} \in \mathbb{R}^n$, and r second-order cones $\mathcal{L}_1, \dots, \mathcal{L}_r$, a SOCP can be expressed as:

$$\min \boldsymbol{c}^\top \boldsymbol{x} \text{ subject to } \mathbf{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{x}_i \in \mathcal{L}_i \text{ for all } i \in [r], \tag{1}$$

where $\boldsymbol{x}$ is the concatenation of $\boldsymbol{x}_i$ and lies in the domain $\mathcal{L} \overset{\text{def}}{=} \mathcal{L}_1 \times \cdots \times \mathcal{L}_r$, where $\mathcal{L}_i \in \mathbb{R}^{n_i}$ is defined as $\mathcal{L}_i := \{x = (x_{\triangleleft}, \boldsymbol{x}_{\triangleright}) \in \mathbb{R}^{n_i} : \|\boldsymbol{x}_{\triangleright}\|_2 \leq x_{\triangleleft}\}$.

1.1 Related Work

Second-order cone programming is an extremely well-studied problem with a long history. As a natural generalization of linear programming, there are numerous algorithmic frameworks developed for solving SOCPs, such as the simplex method for conic programming [Gol02, Zha19, Zha20], sequential linear programming [Siv02], and polyhedral reformulation of second-order cones [BTN01].

Interior Point Methods Karmarkar first proved interior point methods (IPM) can solve linear programs in polynomial time in 1984 [Kar84]. By introducing a barrier function in the objective and adjusting its weight, the method solves a slightly changing optimization problem at each iteration. IPMs are a prominent algorithm class for solving SOCPs to attain high-accuracy solutions. There are many efforts to generalize IPM to a more broad class of problems [BV03, NN94].

IPM with Linear Programming Recently, much attention has been given to IPM-based methods for solving linear programs. [CLS21] first demonstrated that general LPs could be solved in $O^*(n^\omega)$[1] time, where $\omega \approx 2.37$ is the exponent of matrix multiplication. For LPs with specific structures, nearly linear-time algorithms have been proposed [BLSS20, BLL$^+$21, DLY20].

IPM with Nonlinear Programming The advancements made in linear programming through IPM have been successfully extended to a broader range of optimization problems. This includes domains such as semi-definite programming [JKL$^+$20, HJS$^+$22], empirical risk minimization [LSZ19, QSZZ23], and quadratic programming [GSZ23].

[1]Throughout the report, we use O^* to hide $n^{o(1)}$ and $\operatorname{poly}\log(n/\varepsilon)$ factors, and we use $\widetilde{O}$ to hide factors polylogarithmic in n and $\log(1/\varepsilon)$.

Table 1: Previous work utilizing interior point methods in various convex programming problems

Year	Type of Convex Program	Complexity Achieved	Researchers
1984	Linear program	$O^*(n^{3.5})$	Karmarkar [Kar84]
1994	Second-order cone program	$O^*(n^{\omega+0.5})$[2]	Nesterov, Nemirovski [NN94]
2019	Constant dimension convex program	$O^*(n^{\omega})$	Lee, Song, Zhang [LSZ19]
2021	Linear program	$O^*(n^{\omega})$	Cohen, Lee, Song [CLS21]
2023	Convex quadratic program	$O^*(n^{\omega})$	Gu, Song, Zhang [GSZ23]

In Table 1, we summarize several important advancements in convex programming. As shown in the table, IPMs have been an active area of research since Karmarkar [Kar84] proved that they can solve linear programs in polynomial time. Recent years have seen substantial advances in both linear programming and semi-definite programming (SDP), especially in the theoretical computer science community [CLS21, LSZ19, LS14, LS15, LS19, LSW15, JKL+20, JLSW20, HJS+22], and the current fastest algorithms for both LP and SDP are based on an interior-point method.

The seminal paper by Nesterov and Nemirovski [NN94] in 1994 showed that their general results on self-concordant barriers can be applied to the SOCP problems, thereby yielding interior-point methods for SOCPs with run time $O^*(\sqrt{r}n^{\omega})$ time. However, progress appears to have stagnated for SOCPs in the last 30 years. To date, Nesterov and Nemirovski's work remains the most efficient for SOCPs.

1.2 Research Gap and Objectives

The recent breakthroughs in IPMs are underpinned by inverse maintenance techniques [Vai87, LS15, CLS21, LSZ19], taking advantage of the block-diagonal structure of the Hessian matrix to simplify per-iteration updates into solving lower-dimensional linear systems. However, it is challenging in making similar improvements to SOCP. Given the inherent nature of SOCPs, the sizes of the block matrices in the Hessian can vary significantly. Even updating a single high-dimension block will keep the time complexity unchanged at $O(\sqrt{r}n^{\omega})$. Addressing these challenges, our research objectives are as follows:

- **Develop time complexity-reducing strategies:** Design strategies to overcome the complexity barrier inherent in SOCP algorithms by exploring innovative approaches to reduce computational costs.
- **Develop a fast SOCP algorithm and prove its runtime:** Create a new SOCP solution and provide rigorous mathematical proof of its convergence, which is crucial to ensure that the solution is robust.
- **Build and test a complete SOCP solver:** Implement theoretical ideas in practice by creating a new SOCP solver, producing a resource that can be readily employed by the research community.

[2]Nesterov and Nemirovski's work covers general convex programming, but they produced an $O^*(\sqrt{r}n^{\omega})$ run time for SOCP.

- **Explore applications:** Identify and explore new applications in fields where our results can provide significant benefits. With a more efficient SOCP algorithm, we aim to open up new avenues for scientific exploration and practical problem-solving.

1.3 Main Theorems

We introduce a novel cone-splitting technique to reduce high-dimension second-order cones into smaller ones, enabling each cone to be rewritten into roughly equal-sized dimensions. This is a crucial step to overcome the high computational cost associated with high-dimension constraints.

Integrating cone-splitting along with vector approximation and inverse maintenance techniques, we present a new deterministic algorithm based on IPM with a run time of $O^*(n^\omega + n^{2.5-\alpha/2} + n^2 r^{1/6})$, where ω is the exponent of matrix multiplication, and α is the dual exponent of matrix multiplication[3]. Note that this simplifies to $\widetilde{O}((n^\omega)\log(1/\epsilon))$ for the current values of $\omega \approx 2.37$ and $\alpha \approx 0.31$. To the best of our knowledge, this is the first improvement on the computational complexity of solving SOCPs after the seminal work of [NN94] on general convex programs.

Moreover, our approach is fairly general and can be extended to produce an algorithm with runtime $O^*(n^\omega + n^{2.5-\alpha/2} + n^2 r^{1/6})$ for any convex program where each convex set in the constraints has the property that the dimension is bounded by $O(n/r)$ and there is an $O(1)$-self-concordant barrier.

More formally, we prove the following results:

Theorem 1.2 (Main Theorem)**.** *Given a second-order cone program as specified in Definition 1.1 with full-rank constraint matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $m \le n$, define the following parameters:*

1. *Outer radius R: Assume $\|\mathbf{A}\|_1 \le R$, and $\|\boldsymbol{b}\|_2 \le R$ and any feasible solution $\boldsymbol{x}$ satisfies $\|\boldsymbol{x}\|_2 \le R$.*
2. *Lipschitz constant L: $\|\boldsymbol{c}\|_2 \le L$.*

Then, for any $0 < \epsilon \le 1/2$, there is a deterministic *algorithm which runs in time*

$$\widetilde{O}((n^\omega + n^{2.5-\alpha/2+o(1)} + n^{2+o(1)} r^{1/6})\log(\tfrac{R}{\varepsilon}))$$

and outputs an approximate solution $\boldsymbol{x} \in \mathcal{L}$ such that

$$\|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|_1 \le \epsilon$$

and

$$\boldsymbol{c}^\top \boldsymbol{x} \le \min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b}, \boldsymbol{x}_i \in \mathcal{L}_i \text{ for all } i\in[r]} \boldsymbol{c}^\top \boldsymbol{x} + \epsilon L.$$

Theorem 1.3. *Given the matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, two vectors $\boldsymbol{b} \in \mathbb{R}^m$ and $\boldsymbol{c} \in \mathbb{R}^n$, and r compact convex sets $K_1, \dots, K_r$, let's consider the following convex program:*

$$\min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b}, \boldsymbol{x}_i \in K_i \text{ for all } i\in[r]} \boldsymbol{c}^\top \boldsymbol{x}.$$

Assume the following:

1. *$\mathbf{A} \in \mathbb{R}^{m\times n}$ is a full-rank matrix with $m \le n$.*
2. *$K_i \in \mathbb{R}^{n_i}$ where $n_i \le O(n/r)$ is a compact convex set with $O(1)$-self-concordant barrier such that $\phi_i(\boldsymbol{x}_i)$, $\nabla\phi_i(\boldsymbol{x}_i)$, $\nabla^2\phi_i(\boldsymbol{x}_i)$ can all be computed in $O(n_i^\omega)$ time.*
3. *Outer radius R: Assume $\|\mathbf{A}\|_1 \le R$, and $\|\boldsymbol{b}\|_2 \le R$ and any feasible solution $\boldsymbol{x}$ satisfies $\|\boldsymbol{x}\|_2 \le R$.*

[3] The dual exponent of matrix multiplication α is the supremum among all $\alpha \ge 0$ such that it takes $n^{2+o(1)}$ time to multiply an $n \times n$ matrix by an $n \times n^\alpha$ matrix.

4. *Lipschitz constant L: $\|\boldsymbol{c}\|_2 \leq L$.*

Then, for any $0 < \epsilon \leq 1/2$, there is a deterministic *algorithm which runs in time*

$$\widetilde{O}((n^\omega + n^{2.5-\alpha/2+o(1)} + n^{2+o(1)} r^{1/6}) \log(\tfrac{R}{\varepsilon}))$$

and outputs an approximate solution $\boldsymbol{x} \in \prod_{i=1}^r K_i$ such that

$$\|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|_1 \leq \epsilon$$

and

$$\boldsymbol{c}^\top \boldsymbol{x} \leq \min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b}, \boldsymbol{x}_i \in K_i \text{ for all } i \in [r]} \boldsymbol{c}^\top \boldsymbol{x} + \epsilon L.$$

Report Organization This paper is organized as follows. In Section 2, we establish the notation used throughout this report. In Section 3, we lay out our strategies for developing a faster SOCP algorithm, utilizing cone-splitting and approximation techniques. In Section 4, we discuss the details of cone-splitting. In Section 5, we detail the approximation techniques used, including vector approximation and inverse maintenance techniques, and we analyze the run time of each component of the IPM. Putting everything together, in Section 6, we utilize results from the previous sections to prove the convergence of the algorithm as stated in Theorems 1.2 and 1.3. In Section 7, we provide empirical results confirming the convergence of the SOCP algorithm in $O^*(n^\omega)$ time, as well as further explorations into factors affecting the run time. In Section 8, we explore how the new SOCP solution facilitates advancements in various fields, including machine learning, portfolio optimization, and energy management.

2 Notation

In this section, we introduce the notation used in our paper as well as the definitions and prior known results that we rely on.

We use lowercase boldface letters to denote (column) vectors and uppercase boldface letters to denote matrices. We use $\boldsymbol{x}_i$ to denote the i^{th} block of coordinates in the vector $\boldsymbol{x}$. For a vector $\boldsymbol{v} \in \mathbb{R}^n$, we use $\|\boldsymbol{v}\|_2$ to denote its Euclidean norm. Each block of vector $\boldsymbol{x}$ usually lives in a second-order cone; we further use $x_{i,\lhd}$ and $\boldsymbol{x}_{i,\rhd}$ to denote the coordinates of that block.

For a positive semi-definite matrix (PSD) $\mathbf{A} \in \mathbb{R}^{n \times n}$, we use $\|\boldsymbol{v}\|_{\mathbf{A}}$ to denote the induced norm $\sqrt{\boldsymbol{v}^\top \mathbf{A} \boldsymbol{v}}$. For a convex function $f(\boldsymbol{x})$ clear from the context, we define the local norm $\|\boldsymbol{v}\|_{\boldsymbol{x}} = \|\boldsymbol{v}\|_{\nabla^2 f(\boldsymbol{x})}$ and $\|\boldsymbol{v}\|_{\boldsymbol{x}}^* = \|\boldsymbol{v}\|_{(\nabla^2 f(\boldsymbol{x}))^{-1}}$. We define the ℓ_1-norm for a matrix as $\|\mathbf{A}\|_1 = \sum_{ij} |A_{i,j}|$.

We use $\mathcal{L}^k$ to denote the second-order cone in $\mathbb{R}^{k+1}$.

3 Overview

In this section, we provide a high-level overview of the algorithmic framework and technique used. While our result extends to the product space of general convex sets in Theorem 1.3, for simplicity, we focus on the SOCP here.

3.1 Robust Interior Point Method

Consider the following second-order cone program in the standard form

$$\min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b}, \boldsymbol{x} \in \mathcal{L}} \boldsymbol{c}^\top \boldsymbol{x} \quad \text{(primal)} \qquad \text{and} \qquad \max_{\mathbf{A}^\top \boldsymbol{y} + \boldsymbol{s} = \boldsymbol{c}, \boldsymbol{s} \in \mathcal{L}} \boldsymbol{b}^\top \boldsymbol{y} \quad \text{(dual).} \tag{SOCP}$$

for some matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$. The interior point method follows the central path $\boldsymbol{x}(t)$ which it starts at some interior point ($t \gg 0$) to the optimal solution ($t = 0$):

$$\boldsymbol{x}(t) = \arg\min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b}} \boldsymbol{c}^\top \boldsymbol{x} + t\phi(\boldsymbol{x}) \quad \text{with } \phi(\boldsymbol{x}) \stackrel{\text{def}}{=} \sum_{i=1}^{r} \phi_i(\boldsymbol{x}_i),$$

where $\phi_i : \mathcal{L}_i \to \mathbb{R}$ are self-concordant barrier functions. The optimal condition of the path outlined above is defined by:

$$\begin{aligned} \frac{1}{t}\boldsymbol{s} + \nabla\phi(\boldsymbol{x}) &= 0, \\ \mathbf{A}\boldsymbol{x} &= \boldsymbol{b}, \\ \mathbf{A}^\top \boldsymbol{y} + \boldsymbol{s} &= \boldsymbol{c}, \\ (\boldsymbol{x}, \boldsymbol{s}) &\in \mathcal{L} \times \mathcal{L}. \end{aligned} \tag{2}$$

Due to the computational cost, following the central path precisely is prohibitive. Instead, the RIPM maintains a feasible solution pair $(\boldsymbol{x}, \boldsymbol{s})$ that serve as *approximate* minimizers. More specifically, we define $\mu_i(t, \boldsymbol{x}_i, \boldsymbol{s}_i) \stackrel{\text{def}}{=} \boldsymbol{s}_i/t + \nabla\phi_i(\boldsymbol{x}_i)$. RIPM maintains $(\boldsymbol{x}, \boldsymbol{s})$ such that

$$\|\gamma(t, \boldsymbol{x}, \boldsymbol{s})\|_\infty = \frac{1}{\text{poly}\log n} \quad \text{where} \quad \gamma(t, \boldsymbol{x}, \boldsymbol{s})_i \stackrel{\text{def}}{=} \|(\nabla^2\phi_i(\boldsymbol{x}_i))^{-1/2}\mu_i(t, \boldsymbol{x}_i, \boldsymbol{s}_i)\|_2. \tag{3}$$

In each iteration of the algorithm, t is decreased by some multiplicative factor, and then μ takes a Newton-like step. The rate of decreasing t is determined by the self-concordance of the chosen barrier function ϕ.

Definition 3.1 ([Nes89]). A function ϕ is a ν-self-concordant barrier for a non-empty open convex set K if $\operatorname{dom}\phi = K$, $\phi(x) \to +\infty$ as $x \to \partial K$, and for any $x \in K$ and for any $u \in \mathbb{R}^n$

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

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

Lemma 3.2 ([Nes03]). *The self-concordance ν is at least 1 for any self-concordant barrier function.*

Nesterov and Nemirovski [NN94] proved that one can decrease t by $1 - \frac{1}{\sqrt{\nu}}$ factor in each iteration, implying that the IPM converges in $O(\sqrt{\nu}\log(1/\epsilon))$ iterations. For any open convex set $K \in \mathbb{R}^n$, there are n-self-concordant barriers [LY21, BE14, Che21]. For linear constraint $x \in [l, u]$, one can use the log-barrier $-\log(u-x) - \log(x-l)$. Notably, the barrier $-\log(x_\triangleleft^2 - \|\boldsymbol{x}_\triangleright\|^2) - \log(x_\triangleleft)$ is $O(1)$-self-concordant for second-order cone [Nem04].

After decreasing t by $1 - \frac{1}{\sqrt{\nu}}$ factor, the centrality measure $\|(\nabla^2\phi(\boldsymbol{x}))^{-1/2}\mu\|_\infty$ may increase. Therefore, we let μ take a Newton-like step. To move $(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{y}) \leftarrow (\boldsymbol{x} + \delta_{\boldsymbol{x}}, \boldsymbol{s} + \delta_{\boldsymbol{s}}, \boldsymbol{y} + \delta_{\boldsymbol{y}})$, it suffices to solve the following linear system:

$$\begin{aligned} \frac{1}{t}\delta_{\boldsymbol{s}} + \nabla^2\phi(\boldsymbol{x})\delta_{\boldsymbol{x}} &= \delta_\mu, \\ \mathbf{A}\delta_{\boldsymbol{x}} &= 0, \\ \mathbf{A}^\top \delta_{\boldsymbol{y}} + \delta_{\boldsymbol{s}} &= 0. \end{aligned} \tag{4}$$

Recently, it has been observed that one can use an approximation to replace $\nabla^2\phi(\boldsymbol{x})$ and δ_μ in the aforementioned linear system. Instead of computing them exactly, we maintain some block-wise approximation, denoted as $\overline{\boldsymbol{x}}$ and $\overline{\boldsymbol{s}}$, and use them to compute $\nabla^2\phi$ and δ_μ. We only update $(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}})$ when they change significantly to ensure they have sparse updates. We refer to this IPM as a robust interior point method (RIPM). In this paper, we follow the RIPM framework developed in [DLY20] in a black-box manner, maintaining $\|\overline{\boldsymbol{x}}_i - \boldsymbol{x}_i\|_{\overline{\boldsymbol{x}}_i} \leq \frac{1}{\operatorname{poly}\log(n)}$ for all $i \in [r]$.

Lemma 3.3 ([DLY20]). *Consider the convex program*

$$\min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b}, \boldsymbol{x}_i \in K_i \text{ for all } i\in[r]} \boldsymbol{c}^\top \boldsymbol{x},$$

where $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, $\boldsymbol{c} \in \mathbb{R}^n$, where $K \in \mathbb{R}^{n_i}$ with ν_i-self-concordant barrier ϕ_i. Given an initial central path parameter $(t_{\text{start}}, \boldsymbol{x}, \boldsymbol{s})$ such that $\Phi(t_{\text{start}}, \boldsymbol{x}, \boldsymbol{s}) \leq \cosh(\lambda/128)$ where $\Phi(t, \boldsymbol{x}, \boldsymbol{s}) \overset{def}{=} \sum_{i=1}^r \cosh(\lambda\gamma_i(t, \boldsymbol{x}, \boldsymbol{s}))$, and a target t_{end} such that $t_{\text{start}}/t_{\text{end}} = \Omega(1)$, the procedure Centering *in Algorithm 1 outputs a central path parameter $(t_{\text{end}}, \boldsymbol{x}', \boldsymbol{s}')$ in $O(\sqrt{\nu}\log(t_{\text{start}}/t_{\text{end}}))$ many iterations such that*

$$\boldsymbol{c}^\top \boldsymbol{x}' \leq \min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b}, \boldsymbol{x}_i \in K_i \text{ for all } i\in[r]} \boldsymbol{c}^\top \boldsymbol{x} + 4t_{\text{end}}\nu,$$

where $\nu = \sum_{i=1}^r \nu_i$.

3.2 Rewriting Second-order Cone Constraint

The RIPM itself is insufficient for obtaining an efficient algorithm. Given our goal is to get an algorithm that runs in n^ω time, amortized cost per iteration is constrained to be roughly $n^\omega r^{-1/2}$, which is sub-quadratic for moderately large r.

A key obstacle within the RIPM is our lack of control over which cone needs to be updated. Even when updating just one cone per iteration, there may be some cones with dimension $\Omega(n)$, necessitating n^2 time even to write down $\nabla^2\phi_i(\boldsymbol{x}_i)$. To handle this issue, we present a novel reduction that enables us to express second-order cone constraints in a smaller dimension. Specifically, we can transform a second-order cone constraint $\mathcal{L}^{2k} = \{(x_\triangleleft, \boldsymbol{x}_\triangleright) : x_\triangleleft \geq \|\boldsymbol{x}_\triangleright\|_2\} \subset \mathbb{R}^{2k+1}$ into the intersection of three cones with an affine space as follows:

$$\{\boldsymbol{x}^{(1)} \in \mathcal{L}^k, \boldsymbol{x}^{(2)} \in \mathcal{L}^k, (c, a, b) \in \mathcal{L}^2 : a = x^{(1)}_\triangleleft, b = x^{(2)}_\triangleleft\}.$$

This reduction directly leads to a $O^*(n^\omega + n^{2.5-\alpha/2} + n^{2+1/6})$ time algorithm, by iteratively applying the rewriting until the largest cone has constant size and then utilizing the algorithm from [LSZ19]. We refer readers to Theorem 4.3 and Appendix A for the details.

3.3 Inverse Maintenance

Rewriting cones into constant dimensions and applying the algorithm of [LSZ19] gives us a randomized algorithm with iteration complexity $O^*(\sqrt{n})$ and run time $O^*(n^\omega + n^{2.5-\alpha/2} + n^{2+1/6})$. In this subsection, we show to obtain a simple deterministic algorithm that retains $\sqrt{r}$ iteration complexity, with total run time is $O^*(n^\omega + n^{2.5-\alpha/2} + n^2 r^{1/6})$.

We observe that instead of rewriting the largest cone into constant dimension, it suffices to make sure all the cones have size $O(n/r)$. Note that we will at most introduce $O(r)$ many new cones. Since each cone has a $O(1)$-self-concordance barrier, this directly implies an IPM with $O^*(\sqrt{r})$ iterations.

Algorithm 1 Robust Interior Point Method for SOCP

```
1: procedure SOCPSOLVE($\mathbf{A} \in \mathbb{R}^{m\times n}, \boldsymbol{b}, \boldsymbol{c}, R, \varepsilon$)
2:     Rewrite the new SOCP accoridng to Theorem 4.1 with $d = n/r$.
3:     Define $\phi_i(\boldsymbol{x}_i) \overset{\text{def}}{=} -\log(\boldsymbol{x}_{i,\triangleleft}) - \log(\boldsymbol{x}_{i,\triangleleft}^2 - \|\boldsymbol{x}_{i,\triangleright}\|_2^2) - \log(R - \boldsymbol{x}_{i,\triangleleft})$.
4:     Modify the SOCP and obtain an initial $(\boldsymbol{x}, \boldsymbol{s})$ for modified convex program according to Lemma B.1.
5:     Let $t_{\text{start}} = 1, t_{\text{end}} = \varepsilon^2/(2^{10} n^3 R^4)$.
6:     $\boldsymbol{x}, \boldsymbol{s} \leftarrow$ CENTERING$(t_{\text{start}}, t_{\text{end}}, \boldsymbol{x}, \boldsymbol{s}, \phi)$.
7:     Let $\boldsymbol{x}', \boldsymbol{s}'$ be the approximate solution of the new SOCP according to Lemma B.1.
8:     Return the approximate solution of the original SOCP using Lemma 4.2.
9: end procedure

10: procedure CENTERING($t_{\text{start}}, t_{\text{end}}, \boldsymbol{x}, \boldsymbol{s}, \phi$)          ▷ See the implementation in Algorithm 2.
11:     Let $\bar{\varepsilon} = \frac{1}{2^{12}\lambda}$ and $\lambda = 64\log(256 r^2)$.
12:     Let $t \leftarrow t_{\text{start}}, \overline{\boldsymbol{x}} \leftarrow \boldsymbol{x}, \overline{\boldsymbol{s}} \leftarrow \boldsymbol{s}, \bar{t} \leftarrow t$.
13:     while $t \geq t_{\text{end}}$ do
14:         Set $t \leftarrow \max((1 - \frac{\bar{\varepsilon}}{128\sqrt{r}})t, t_{\text{end}})$.
15:         Maintain $\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}$ such that $\|\overline{\boldsymbol{x}}_i - \boldsymbol{x}_i\|_{\overline{\boldsymbol{x}}_i} \leq \bar{\varepsilon}, \|\overline{\boldsymbol{s}}_i - \boldsymbol{s}_i\|_{\overline{\boldsymbol{x}}_i} \leq \bar{t}\bar{\varepsilon}$ for all $i \in [r]$.
16:         Compute $\delta_\mu(\bar{t}, \overline{\boldsymbol{x}}, \overline{\boldsymbol{s}})$, where
                $$\delta_\mu(t, \boldsymbol{x}, \boldsymbol{s})_i \overset{\text{def}}{=} -\frac{\bar{\varepsilon}}{2} \cdot \frac{\sinh(\lambda\gamma(t, \boldsymbol{x}, \boldsymbol{s})_i)}{\|\cosh(\lambda\gamma(t, \boldsymbol{x}, \boldsymbol{s}))\|_2} \cdot \mu_i(t, \boldsymbol{x}, \boldsymbol{s}). \quad (5)$$
17:         Find $(\delta_{\boldsymbol{x}}, \delta_{\boldsymbol{s}}, \delta_{\boldsymbol{y}})$ such that these three equations hold:
                $$\begin{aligned} \frac{1}{\bar{t}}\delta_{\boldsymbol{s}} + \nabla^2\phi(\overline{\boldsymbol{x}})\delta_{\boldsymbol{x}} &= \delta_\mu(\bar{t}, \overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}), \\ \mathbf{A}\delta_{\boldsymbol{x}} &= 0, \\ \mathbf{A}^\top\delta_{\boldsymbol{y}} + \delta_{\boldsymbol{s}} &= 0. \end{aligned} \quad (6)$$
18:         $\boldsymbol{x} \leftarrow \boldsymbol{x} + \delta_{\boldsymbol{x}}, \boldsymbol{s} \leftarrow \boldsymbol{s} + \delta_{\boldsymbol{s}}$.
19:         Update $\bar{t} \leftarrow t$ if $|\bar{t} - t| \geq \bar{\varepsilon}\bar{t}/4$.
20:     end while
21:     return $(t_{\text{end}}, \boldsymbol{x}, \boldsymbol{s})$.
22: end procedure
```

Recall from Eq. (4) that each step of the robust interior point method solves the following linear system:

$$\begin{pmatrix} \nabla^2\phi(\overline{\boldsymbol{x}}) & \mathbf{I}/\overline{t} & \mathbf{0} \\ \mathbf{A} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{A}^\top \end{pmatrix} \begin{pmatrix} \delta_{\boldsymbol{x}} \\ \delta_{\boldsymbol{s}} \\ \delta_{\boldsymbol{y}} \end{pmatrix} = \begin{pmatrix} \delta_\mu \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}. \tag{7}$$

Using the techniques from [LV21, Bra21], we can reduce the dynamic linear system problem into an inverse maintenance problem.

Lemma 3.4 ([Bra21])**.** *For any invertible matrix $\mathbf{M} \in \mathbb{R}^{n\times n}$ and any vector $v \in \mathbb{R}^n$, we have*

$$\begin{bmatrix} \mathbf{M} & \boldsymbol{v} \\ 0 & -1 \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{M}^{-1} & \mathbf{M}^{-1}\boldsymbol{v} \\ 0 & -1 \end{bmatrix}.$$

Then the question reduces to maintaining the last column of inverse of $\mathbf{M}(\overline{t}, \overline{\boldsymbol{x}}, \overline{\boldsymbol{s}})$ where

$$\mathbf{M}(t, \boldsymbol{x}, \boldsymbol{s}) \overset{\text{def}}{=} \begin{bmatrix} \nabla^2\phi(\boldsymbol{x}) & \mathbf{I}/t & \mathbf{0} & \delta_\mu(t, \boldsymbol{x}, \boldsymbol{s}) \\ \mathbf{A} & \mathbf{0} & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{I} & \mathbf{A}^\top & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}. \tag{8}$$

Finally, we use the Sherman-Morison-Woodbury matrix identity and fast rectangular matrix multiplication to maintain the inverse above.

4 Reformulating Second-Order Cone Constraints

In this section, we show how to break down large cone constraints in the SOCP into smaller ones. This is a key step to reduce computational cost.

Theorem 4.1. *Consider a SOCP in the standard form*

$$\min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b}, \boldsymbol{x}\in\mathcal{L}} \boldsymbol{c}^\top \boldsymbol{x},$$

where $\mathcal{L} = \mathcal{L}^{n_1} \times \cdots \times \mathcal{L}^{n_r}$ and $\mathbf{A} \in \mathbb{R}^{m\times n}$.

Given a dimension threshold $d > 10$, we can construct an equivalent SOCP with the dimension of the largest cone bounded by $d + 1$:

$$\min_{\overline{\mathbf{A}}\overline{\boldsymbol{x}}=\overline{\boldsymbol{b}}, \overline{\boldsymbol{x}}\in\overline{\mathcal{L}}} \overline{\boldsymbol{c}}^\top \overline{\boldsymbol{x}},$$

in $O(\operatorname{nnz}(\mathbf{A}) + n)$ time, which satisfies the following:

1. *Cone Dimension : $\overline{n}_i \le d + 1$ for all i.*
2. *Total Dimension : $\overline{n} = \sum_{i=1}^{\overline{r}} \overline{n}_i \le 2n$.*
3. *Cone Number : $\overline{r} \le 2n/d$.*
4. *Number of Rows in $\overline{\mathbf{A}}$: $\overline{m} \le m + \overline{r}$.*

The new SOCP is equivalent to the original in the following sense: given a feasible solution to either of these two SOCPs, one can construct a feasible solution for the other with the same objective function value in $O(n)$ time.

Proof. It suffices to show how to rewrite a single second-order cone constraint $\boldsymbol{x}_i \in \mathcal{L}_i$ for some i such that $n_i > d$. For the simplicity of the proof, we drop the subscript and consider $\boldsymbol{x} \in \mathcal{L} \subset \mathbb{R}^n$ instead. Moreover, we assume $n = kd + 1$ for some integer k; note one can just pad dummy variables if $n-1$ is not a multiple of d.

We split $\mathcal{L}$ into product of $k+1$ many second-order cones, where the j-th cone $\mathcal{L}^{(j)} \in \mathbb{R}^{d+1}$ for $j \leq k$ and $\mathcal{L}^{(k+1)} \in \mathbb{R}^{k+1}$. Denote this product of new cones as $\mathcal{L}'$. We note that $\mathcal{L}' \subset \mathbb{R}^{kd+2k+1}$. To make the new cone equivalent to the original one, we need to add some equality constraints: for $\boldsymbol{x}' \in \mathcal{L}'$, we add $x'_{i,\triangleleft} = x'_{k+1,i}$ for all $i \in [k]$.

Let $\boldsymbol{c}$ be the corresponding cost vector. We define the new cost vector $\boldsymbol{c}'$ as follows: set $c'_{i,\triangleleft} = 0$ and $c'_{i,j} = c_{(i-1)d+j}$ for $i \in [k]$. We set $\boldsymbol{c}'_{k+1} = (c_\triangleleft, \mathbf{0})$.

Now, we demonstrate how to convert the solution from the modified SOCP to the original one. Given $\boldsymbol{x}' \in \mathcal{L}'$, we set $\boldsymbol{x} = (x'_{k+1,\triangleleft}, \boldsymbol{x}'_{1,\triangleright}, \ldots, \boldsymbol{x}'_{k,\triangleright})$. First, we check $\boldsymbol{x} \in \mathcal{L}$. Since $\boldsymbol{x}' \in \mathcal{L}'$, we have

$$(x'_{k+1,\triangleleft})^2 \geq \|\boldsymbol{x}'_{k+1,\triangleright}\|_2^2 = \sum_{i=1}^{k} (\boldsymbol{x}'_{i,\triangleleft})^2 \geq \sum_{i=1}^{k} \|\boldsymbol{x}'_{i,\triangleleft}\|_2^2,$$

where the first inequality follows by $\boldsymbol{x}'_{k+1} \in \mathcal{L}^{(k+1)}$, the second step follows by the equality constraints above, and the last inequality follows by $\boldsymbol{x}_i \in \mathcal{L}^{(i)}$. This shows $x \in \mathcal{L}$. Then, for the objective function value, we have

$$\begin{aligned}\langle \boldsymbol{c}', \boldsymbol{x}' \rangle &= \sum_{i=1}^{k+1} \langle \boldsymbol{c}'_i, \boldsymbol{x}'_i \rangle \\ &= c_\triangleleft \cdot x_\triangleleft + \sum_{i=1}^{k} \langle \boldsymbol{c}'_i, \boldsymbol{x}'_i \rangle \\ &= c_\triangleleft \cdot x_\triangleleft + \langle \boldsymbol{c}_\triangleright, \boldsymbol{x}_\triangleright \rangle = \langle \boldsymbol{c}, \boldsymbol{x} \rangle,\end{aligned}$$

where the first step follows by $\boldsymbol{c}'_{k+1} = (c_\triangleleft, \mathbf{0})$ and $x_\triangleleft = x'_{k+1,\triangleleft}$, and the second step follows by the definition of $\boldsymbol{c}'$ and $\boldsymbol{x}_\triangleright = (\boldsymbol{x}'_{1,\triangleright}, \ldots, \boldsymbol{x}'_{k,\triangleright})$.

Similarly, one can convert the solution from the original SOCP to the new one. Given $\boldsymbol{x} \in \mathcal{L}$, we construct x' by setting $x'_{i,j} = x_{(i-1)d+j}$ for $i \in [k], j \in [d]$. Then, we set $x'_{i,\triangleleft} = x'_{k+1,i} = \|x'_{i,\triangleright}\|_2$. Finally, let $x'_{k+1,\triangleleft} = x_\triangleleft$. It is easy to verify $x' \in \mathcal{L}'$ and $\langle c', x' \rangle = \langle c, x \rangle$.

If $k \leq d$, then we are done. Otherwise, we repeat the procedure above on $\mathcal{L}^{(k+1)}$.

Note that each time we repeat the procedure above, the dimension of the largest cone is decreased by a factor of d. Then, using the sum of a geometric series, this procedure at most introduces $2n/d$ many variables, and the number of cones can also be bounded by $2n/d$. Moreover, we notice that for $\mathcal{L}^{(j)}$ for $j \in [k]$, there will be at most 1 corresponding constraint.

Since we only need to add $O(\overline{r})$ many constraints, and each of them touches two variables, this gives the bound on the running time. □

However, the result obtained by RIPM may not satisfy $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ exactly; we need the following lemma to bound the error.

Lemma 4.2. *Consider an original SOCP with standard form* $\min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b}, \boldsymbol{x}\in\mathcal{L}} \boldsymbol{c}^\top \boldsymbol{x}$ *with a modified SOCP* $\min_{\overline{\mathbf{A}}\overline{\boldsymbol{x}}=\overline{\boldsymbol{b}}, \overline{\boldsymbol{x}}\in\overline{\mathcal{L}}} \overline{\boldsymbol{c}}^\top \overline{\boldsymbol{x}}$, *obtained by Theorem 4.1. Given an* $\overline{\boldsymbol{x}}$ *such that* $\overline{\boldsymbol{x}} \in \overline{\mathcal{L}}$, *and* $\|\overline{\mathbf{A}}\overline{\boldsymbol{x}} - \overline{\boldsymbol{b}}\|_1 < \delta$, *we can find a* $\boldsymbol{x} \in \mathcal{L}$ *such that* $\|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|_1 \leq \delta(1 + \|\mathbf{A}\|_1)$ *and* $\boldsymbol{c}^\top \boldsymbol{x} + \delta\|\boldsymbol{c}\|_2 \leq \overline{\boldsymbol{c}}^\top \overline{\boldsymbol{x}}$.

Proof. Let $\boldsymbol{x}'$ be the solution converted from $\overline{\boldsymbol{x}}$, then $\boldsymbol{x}' = (\overline{x}_{k+1,\lhd}, \overline{\boldsymbol{x}}_{1,\rhd}, \ldots, \overline{\boldsymbol{x}}_{k,\rhd})$. Since $\overline{\boldsymbol{x}} \in \overline{L}$, we have

$$\overline{x}_{k+1,\lhd}^2 \geq \|\overline{\boldsymbol{x}}_{k+1,\rhd}\|_2^2 = \sum_{i=1}^{k} (\overline{x}_{i,\lhd} + \xi_i)^2 \quad \text{and} \quad \sum_{i=1}^{k} (\overline{x}_{i,\lhd})^2 \geq \sum_{i=1}^{k} \|\overline{x}_{i,\lhd}\|_2^2$$

for some ξ such that $\|\xi\|_1 \leq \delta$. We define

$$\boldsymbol{x} = (\overline{x}_{k+1,\lhd} + \delta, \overline{\boldsymbol{x}}_{1,\rhd}, \ldots, \overline{\boldsymbol{x}}_{k,\rhd}).$$

Because $|\|a\|_2 - \|b\|_2| \leq \|a - b\|_2$ for any a, b and $\|\xi\|_2 \leq \|\xi\|_1$, one can check that $\boldsymbol{x} \in \mathcal{L}$. The bound on $\|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|$ and $\boldsymbol{c}^\top \boldsymbol{x}$ follows directly from $\boldsymbol{x} - \boldsymbol{x}' = \delta \boldsymbol{e}_1$.

□

A direct corollary of the theorems above is that we get an $O^*(n^\omega + n^{2.5-\alpha} + n^{2+1/6})$ time algorithm using [LSZ19].

Theorem 4.3. *Consider a second-order cone program as specified in Definition 1.1 with no redundant constraints. Define the following parameters:*

1. *Radius R: Assume $\|\mathbf{A}\|_F \leq R$, and $\|\boldsymbol{b}\|_2 \leq R$ and any feasible solution $\boldsymbol{x}$ satisfies $\|\boldsymbol{x}\|_2 \leq R$.*
2. *Lipschitz constant L: $\|\boldsymbol{c}\|_2 \leq L$.*

Then, for any $0 < \epsilon \leq 1/2$, there is a deterministic algorithm which runs in time

$$\widetilde{O}((n^{\omega+o(1)} + n^{2.5-\alpha/2+o(1)} + n^{2+1/6+o(1)}) \log(\tfrac{R}{\varepsilon}))$$

and outputs an approximate solution $\boldsymbol{x} \in \mathcal{L}$ such that

$$\boldsymbol{c}^\top \boldsymbol{x} \leq \min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b}, \boldsymbol{x} \in \mathcal{L}} \boldsymbol{c}^\top \boldsymbol{x} + \varepsilon L$$

and

$$\|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|_1 \leq \varepsilon.$$

We defer the proof to Appendix A.

5 Robust IPM Made Efficient

In this section, we present our implementation for the RIPM. Since the modified convex program is not necessarily a SOCP, we prove a slightly more general version here.

Theorem 5.1. *Consider the convex program*

$$\min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b}, \boldsymbol{x}_i \in K_i \text{ for all } i \in [r]} \boldsymbol{c}^\top \boldsymbol{x},$$

where $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, $\boldsymbol{c} \in \mathbb{R}^n$, where $K \in \mathbb{R}^{n_i}$ and $n_i = O(n/r)$. Given $O(1)$-self-concordant barrier functions ϕ_i for each K_i, a feasible central path parameter $(t_{\text{start}}, \boldsymbol{x}, \boldsymbol{s})$ such that $\Phi(t_{\text{start}}, \boldsymbol{x}, \boldsymbol{s}) \leq \cosh(\lambda/128)$, where $\Phi(t, \boldsymbol{x}, \boldsymbol{s}) = \sum_{i=1}^r \cosh(\lambda\gamma_i(t, \boldsymbol{x}, \boldsymbol{s}))$, and a target t_{end} such that $t/t_{\text{end}} = \Omega(1)$, the Algorithm 2 outputs a feasible central path parameter $(t_{\text{end}}, \boldsymbol{x}', \boldsymbol{s}')$ such that

$$\boldsymbol{c}^\top \boldsymbol{x} \leq \min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b}, \boldsymbol{x}_i \in K_i \text{ for all } i \in [r]} \boldsymbol{c}^\top \boldsymbol{x} + 4\nu t_{\text{end}},$$

where ν is the self-concordance parameter of $\phi = \sum_{i=1}^r \phi_i$.

Assuming $\phi_i(\boldsymbol{x}_i), \nabla\phi_i(\boldsymbol{x}_i), \nabla^2\phi_i(\boldsymbol{x}_i)$ can be computed in $O(n_i^\omega)$ time for any $\boldsymbol{x}_i \in \mathbb{R}^{n_i}$ and $i \in [r]$, the algorithm runs in time $\widetilde{O}((n^\omega + n^{2+o(1)} r^{1/6} + n^{2.5-\alpha/2+o(1)}) \log(t_{\text{start}}/t_{\text{end}}))$.

We split the proof into two parts: vector approximation maintenance (Theorem 5.2) and matrix inverse maintenance (Theorem 5.8), and we combine them together in Section 5.3.

Algorithm 2 implementation of CENTERING in Algorithm 1

```
1: procedure CENTERINGIMPL($t_{\text{start}}, t_{\text{end}}, \boldsymbol{x}, \boldsymbol{s}$)
2:     Let $\bar{\varepsilon} = \frac{1}{2^{12}\lambda}$ and $\lambda = 64\log(256n^2)$.
3:     Let $\ell_*$ be the smallest integer such that $2^{2\ell_*} \geq \min\{r^{2/3}, \max(1, r \cdot n^{\alpha-1})\}$.
4:     Let approxSolution be an instance of APPROXSOLUTION in Algorithm 3.
5:     Let $k \leftarrow 0, t \leftarrow t_{\text{start}}, \overline{\boldsymbol{x}} \leftarrow \boldsymbol{x}, \overline{\boldsymbol{s}} \leftarrow \boldsymbol{s}, \bar{t} \leftarrow t$.
6:     Let $\mathbf{T} \leftarrow (\mathbf{M}(\bar{t}, \overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}))^{-1}$ and $u \leftarrow \mathbf{T}\boldsymbol{e}_{2n+m+1}$.
7:     approxSolution.INITIALIZE($\boldsymbol{x}, \boldsymbol{s}, \bar{t}, \alpha$)
8:     while $t \geq t_{\text{end}}$ do
9:         Set $t \leftarrow \max((1 - \frac{\bar{\varepsilon}}{128\sqrt{r}})t, t_{\text{end}})$.
10:        Set $k \leftarrow k + 1$.
11:        $(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}) \leftarrow$ approxSolution.APPROXIMATE($\boldsymbol{x}, \boldsymbol{s}, \bar{t}, \bar{\varepsilon}$).
12:        Compute $\delta_{\mu}(\bar{t}, \overline{\boldsymbol{x}}, \overline{\boldsymbol{s}})$, where
               $$\delta_{\mu}(t, \boldsymbol{x}, \boldsymbol{s})_i \overset{\text{def}}{=} -\frac{\bar{\varepsilon}}{2} \cdot \frac{\sinh(\lambda\gamma(t, \boldsymbol{x}, \boldsymbol{s})_i)}{\|\cosh(\lambda\gamma(t, \boldsymbol{x}, \boldsymbol{s}))\|_2} \cdot \mu_i(t, \boldsymbol{x}, \boldsymbol{s}). \quad (9)$$
13:        if $k \bmod 2^{\ell_*} \equiv 0$ then
14:            Update $\mathbf{T}$ to $(\mathbf{M}(\bar{t}, \overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}))^{-1}$ using Theorem 5.8.
15:            $\boldsymbol{u} \leftarrow \mathbf{T}\boldsymbol{e}_{2n+m+1}, \boldsymbol{v} \leftarrow \boldsymbol{u}$.
16:        else
17:            Update $\boldsymbol{v}$ to $(\mathbf{M}(\bar{t}, \overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}))^{-1}\boldsymbol{e}_{2n+m+1}$ using Theorem 5.8.
18:        end if
19:        Let $(\delta_{\boldsymbol{x}}, \delta_{\boldsymbol{s}})$ be the first $2n$ coordinates of $\boldsymbol{v}$.
20:        $\boldsymbol{x} \leftarrow \boldsymbol{x} + \delta_{\boldsymbol{x}}, \boldsymbol{s} \leftarrow \boldsymbol{s} + \delta_{\boldsymbol{s}}$.
21:        if $|\bar{t} - t| \geq \bar{\varepsilon}\bar{t}/4$ then                                  ▷ Restart
22:            Update $\bar{t} \leftarrow t, \overline{\boldsymbol{x}} \leftarrow \boldsymbol{x}, \overline{\boldsymbol{s}} \leftarrow \boldsymbol{s}, \mathbf{T} \leftarrow (\mathbf{M}(\bar{t}, \overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}))^{-1}$.
23:            approxSolution.INITIALIZE($\boldsymbol{x}, \boldsymbol{s}, \bar{t}, \bar{\varepsilon}$).
24:        end if
25:    end while
26:    return $(t_{\text{end}}, \boldsymbol{x}, \boldsymbol{s})$.
27: end procedure
```

5.1 Vector Approximation Maintenance

In this subsection, we present a data structure to maintain $\overline{\boldsymbol{x}}$ and $\overline{\boldsymbol{s}}$. The algorithm is similar to the corresponding algorithm in [DLY20, Section 6.3] and [DGG+22, Section 6.1]. Here, we present a slightly simpler version, since we have explicit access to $(\boldsymbol{x}, \boldsymbol{s})$ throughout the algorithm.

Theorem 5.2. *The data structure* APPROXSOLUTION *in Algorithm 3 maintains approximate solution $(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}})$ of $(\boldsymbol{x}, \boldsymbol{s})$ with the following procedures:*

- INITIALIZE($\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{s} \in \mathbb{R}^n, \phi, \bar{t}, \delta > 0$): *Initialize the data structure at step 0 with initial solution pair $(\boldsymbol{x}, \boldsymbol{s})$, a barrier function $\phi : \mathbb{R}^n \to \mathbb{R}$, and the target additive approximation error δ.*

- APPROXIMATE$(\boldsymbol{x}^{(\text{new})}, \boldsymbol{s}^{(\text{new})})$: *Increment the step counter, and update solution pair* $(\boldsymbol{x}, \boldsymbol{s})$. *Output an approximation* $(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}})$ *such that* $\|\overline{\boldsymbol{x}}_i - \boldsymbol{x}_i\|_{\overline{\boldsymbol{x}}_i} \leq \delta$, $\|\overline{\boldsymbol{s}}_i - \boldsymbol{s}_i\|_{\overline{\boldsymbol{x}}_i} \leq \bar{t}\delta$ *for all* $i \in [r]$.

Moreover, if $\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^{(k-1)}\|_{\overline{\boldsymbol{x}}} \leq \beta$ *and* $\|\boldsymbol{s}^{(k)} - \boldsymbol{s}^{(k-1)}\|^*_{\overline{\boldsymbol{x}}} \leq \bar{t}\beta$ *for all* k*, then at the* k*-th step, the data structure updates* $(\overline{\boldsymbol{x}}_i, \overline{\boldsymbol{s}}_i) \leftarrow (\boldsymbol{x}_i, \boldsymbol{s}_i)$ *for* $O(2^{2\ell_k}(\beta/\delta)^2 \log^2 n)$ *many blocks, where* ℓ_k *is the largest integer* ℓ *with* $k \equiv 0 \bmod 2^\ell$*. Moreover, if* $n_i \leq d$ *for all* $i \in [r]$*, and suppose we can compute* $\nabla^2\phi_i(\boldsymbol{x}_i)$ *in time* T*, then the total run time for this algorithm after* K *steps is*

$$\widetilde{O}\left(K^2(\beta/\delta)^2(d^\omega + T) + Krd^2\right).$$

Remark 5.3. *For our usage,* $(\beta/\delta) = \operatorname{poly}\log n$, $k = \widetilde{O}(\sqrt{r})$, $T = O(d^2)$*, and* $r \cdot d = O(n)$*, so we have*

$$\widetilde{O}\left(K^2(\beta/\delta)^2(d^\omega + T) + Krd^2\right) = \widetilde{O}(n^2 r^{-1/2} + n^\omega r^{1-\omega}).$$

Proof of Theorem 5.2. We first show the correctness of the algorithm by using dyadic intervals. Suppose the last time $\overline{\boldsymbol{x}}_i$ get updated at step k'. Then, we note that the corresponding induced norm did not change for all $k \geq k'$. Since we reset all the coordinates every $2^{\lceil \log n \rceil}$ steps, we have $k - 2^{\lceil \log n \rceil} < k' \leq k$. Using dyadic intervals, we can $k' = k_0 < k_1 < \cdots < k_s = k$ where $k_{j+1} - k_j$ is a power of 2, $k_{j+1} - k_j$ divides k_{j+1}, and $|s| \leq 2\lceil \log n \rceil$. Then, we have

$$\|\overline{\boldsymbol{x}}_i^{(k)} - \boldsymbol{x}_i^{(k)}\|_{\overline{\boldsymbol{x}}_i^{(k)}} = \|\boldsymbol{x}_i^{(k_0)} - \boldsymbol{x}_i^{(k)}\|_{\boldsymbol{x}_i^{(k_0)}} \leq \sum_{i=1}^{s} \|\boldsymbol{x}_i^{(k_i)} - \boldsymbol{x}_i^{(k_{i-1})}\|_{\boldsymbol{x}_i^{(k_0)}} \leq \frac{\delta}{2\lceil \log n \rceil} \cdot s \leq \delta.$$

The proof for $\boldsymbol{s}$ is similar; we omit it here.

Now, we bound the number of blocks which change.

Let ℓ_k be the largest integer ℓ with $k \equiv 0 \bmod 2^\ell$. We claim that at the k-th step, the data structure updates $(\overline{\boldsymbol{x}}_i, \overline{\boldsymbol{s}}_i) \leftarrow (\boldsymbol{x}_i, \boldsymbol{s}_i)$ for $O(2^{2\ell_k}(\beta/\delta)^2 \log^2 n)$ many blocks.

Fix some ℓ with $k \equiv 0 \bmod 2^\ell$. By our definition of $I_\ell^{(k)}$, we have for any i, we know that the induced norm does not change for all $j > k - 2^\ell$. Then, we have

$$\sum_{i \in I_\ell^{(k)}} \|\boldsymbol{x}_i^{(k)} - \boldsymbol{x}_i^{(k-2^\ell)}\|^2_{\overline{\boldsymbol{x}}_i} \leq \sum_{i \in [r]} \|\boldsymbol{x}_i^{(k)} - \boldsymbol{x}_i^{(k-2^\ell)}\|^2_{\overline{\boldsymbol{x}}_i} \leq 2^{2\ell}\beta^2.$$

Note that for any i in $I_\ell^{(k)}$, we have $\|\boldsymbol{x}_i^{(k)} - \boldsymbol{x}_i^{(k-2^\ell)}\|^2_{\overline{\boldsymbol{x}}_i} \leq \delta^2$. This gives the bound on size of $I_\ell^{(k)}$:

$$|I_\ell^{(k)}| \leq \frac{2^{2\ell}\beta^2}{\delta^2} \leq 2^{2\ell}(\beta/\delta)^2.$$

Summing over all $\ell < \ell_k$, we get the desired bound. ☐

Lemma 5.4. *Given the solution* $(\delta_{\boldsymbol{x}}, \delta_{\boldsymbol{s}}, \delta_{\boldsymbol{y}})$ *for the following linear system*

$$\begin{aligned} \frac{1}{t}\delta_{\boldsymbol{s}} + \nabla^2\phi(\overline{\boldsymbol{x}})\delta_{\boldsymbol{x}} &= \boldsymbol{v}, \\ \mathbf{A}\delta_{\boldsymbol{x}} &= 0, \\ \mathbf{A}^\top\delta_{\boldsymbol{y}} + \delta_{\boldsymbol{s}} &= 0, \end{aligned}$$

we have $\|\delta_{\boldsymbol{x}}\|_{\overline{\boldsymbol{x}}} = \frac{1}{t}\|\delta_{\boldsymbol{s}}\|^*_{\overline{\boldsymbol{x}}} = \|\boldsymbol{v}\|^*_{\overline{\boldsymbol{x}}}$.

Proof. For simplicity of notation, we use $\mathbf{H}$ to denote $\nabla^2\phi(\overline{\boldsymbol{x}})$ in this proof. Multiplying $\mathbf{A}\mathbf{H}^{-1}$ on both sides of the first equation and $\mathbf{A}\delta_{\boldsymbol{x}} = 0$, we have

$$\frac{1}{\overline{t}}\mathbf{A}\mathbf{H}^{-1}\delta_{\boldsymbol{s}} = \mathbf{A}\mathbf{H}^{-1}\boldsymbol{v}.$$

Replacing $\delta_{\boldsymbol{s}}$ with $-\mathbf{A}^\top\delta_{\boldsymbol{y}}$, we get

$$-\frac{1}{\overline{t}}\mathbf{A}\mathbf{H}^{-1}\mathbf{A}^\top\delta_{\boldsymbol{y}} = \mathbf{A}\mathbf{H}^{-1}\boldsymbol{v}.$$

Since $\mathbf{A}$ has full column rank, $(\mathbf{A}\mathbf{H}^{-1}\mathbf{A}^\top)$ is invertible, and we have

$$\delta_{\boldsymbol{y}} = -\overline{t}(\mathbf{A}\mathbf{H}^{-1}\mathbf{A}^\top)^{-1}\boldsymbol{v}.$$

Then, we have

$$\begin{aligned}\delta_{\boldsymbol{s}} &= \overline{t}\mathbf{A}^\top(\mathbf{A}\mathbf{H}^{-1}\mathbf{A}^\top)^{-1}\boldsymbol{v},\\ \delta_{\boldsymbol{x}} &= \mathbf{H}^{-1}\boldsymbol{v} - \mathbf{H}^{-1}\mathbf{A}^\top(\mathbf{A}\mathbf{H}^{-1}\mathbf{A}^\top)^{-1}\boldsymbol{v}.\end{aligned}$$

Then, we can check that

$$\|\delta_{\boldsymbol{x}}\|_{\overline{\boldsymbol{x}}} = \frac{1}{\overline{t}}\|\delta_{\boldsymbol{s}}\|_{\overline{\boldsymbol{x}}}^* = \|\boldsymbol{v}\|_{\overline{\boldsymbol{x}}}^*$$

by direct calculation. □

5.2 Inverse Maintenance

In this subsection, we show how to maintain the inverse of $\mathbf{M}(t, \boldsymbol{x}, \boldsymbol{s})$ under sparse block changes. Before we proceed, we need several ingredients. The first one is the Sherman-Morison-Woodbury matrix identity.

Lemma 5.5 (Sherman-Morison-Woodbury matrix identity [Woo50])**.** *The Sherman-Morison-Woodbury matrix identity is given by*

$$(\mathbf{M} + \mathbf{U}\mathbf{C}\mathbf{V})^{-1} = \mathbf{M}^{-1} - \mathbf{M}^{-1}\mathbf{U}\left(\mathbf{C}^{-1} + \mathbf{V}\mathbf{M}^{-1}\mathbf{U}\right)^{-1}\mathbf{V}\mathbf{M}^{-1}.$$

where $\mathbf{A} \in \mathbb{R}^{n\times n}, \mathbf{U} \in \mathbb{R}^{n\times k}, \mathbf{C} \in \mathbb{R}^{k\times k}, \mathbf{V} \in \mathbb{R}^{k\times n}$.

To bound the run time for computing the matrix identity above, we need the following two lemmas.

Lemma 5.6 (Rectangular matrix multiplication [CLS21])**.** *For any $r < n$, multiplying an $n \times r$ with an $r \times n$ matrix or $n \times n$ with $n \times r$ takes time*

$$n^{2+o(1)} + r^{\frac{\omega-2}{1-\alpha}}n^{2-\frac{\alpha(\omega-2)}{1-\alpha}+o(1)}.$$

Lemma 5.7 ([CGLZ20])**.** $\omega \le 3 - \frac{1}{2}\omega\alpha$.

Now, we can state our result for maintaining the inverse of $\mathbf{M}(t, \boldsymbol{x}, \boldsymbol{s})$.

Theorem 5.8. *Given $\overline{\boldsymbol{x}}^{(\mathrm{prev})}, \overline{\boldsymbol{s}}^{(\mathrm{prev})}$ in the previous IPM iteration and $\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}$, suppose $\overline{\boldsymbol{x}}$ and $\overline{\boldsymbol{s}}$ differ from $\overline{\boldsymbol{x}}^{(\mathrm{prev})}$ and $\overline{\boldsymbol{s}}^{(\mathrm{prev})}$ in at most q blocks, i.e., $|\{i \in [r] : \overline{\boldsymbol{x}}_i \ne \overline{\boldsymbol{x}}_i^{(\mathrm{prev})}$ or $\overline{\boldsymbol{s}}_i \ne \overline{\boldsymbol{s}}_i^{(\mathrm{prev})}\}| = q$. Let k be the total dimension of these blocks. Given $(\mathbf{M}(\overline{t}, \overline{\boldsymbol{x}}^{(\mathrm{prev})}, \overline{\boldsymbol{s}}^{(\mathrm{prev})}))^{-1}$ and some vector $\boldsymbol{v}$, we can*

Algorithm 3 Data Structure Approximate Solution Maintenance

```
1: data structure ApproxSolution
2: private : member
3:     $\delta > 0$: additive approximation error
4:     $k$: current IPM step
5:     $(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}) \in \mathbb{R}^n \times \mathbb{R}^n$: current valid approximate solution pair
6:     $\{\boldsymbol{x}^{(j)}, \boldsymbol{s}^{(j)} \in \mathbb{R}^n \times \mathbb{R}^n\}_{j=0}^{k}$: list of previous inputs
7:
8: procedure Initialize($\boldsymbol{x}, \boldsymbol{s}, t, \delta$)
9:     $k \leftarrow 0$.
10:    $\boldsymbol{x}^{(k)} \leftarrow \boldsymbol{x}, \boldsymbol{s}^{(k)} \leftarrow \boldsymbol{s}$.
11:    $\overline{\boldsymbol{x}} \leftarrow \boldsymbol{x}, \overline{\boldsymbol{s}} \leftarrow \boldsymbol{s}$.
12:    $t \leftarrow t, \delta \leftarrow \delta$.
13: end procedure
14:
15: procedure Approximate($\boldsymbol{x}^{(\mathrm{new})}, \boldsymbol{s}^{(\mathrm{new})}$).
16:    $k \leftarrow k+1, \boldsymbol{x}^{(k)} \leftarrow \boldsymbol{x}^{(\mathrm{new})}, \boldsymbol{s}^{(k)} \leftarrow \boldsymbol{s}^{(\mathrm{new})}$.
17:    $I \leftarrow \emptyset$.
18:    for all $0 \le \ell < \lceil \log k \rceil$ such that $k \equiv 0 \bmod 2^{\ell}$ do
19:        Compute $I_{\ell}^{(k)}$, where
               $$I_{\ell}^{(k)} \overset{\mathrm{def}}{=} \{i \in [r] : \max\{\|\boldsymbol{x}_i^{(k)} - \boldsymbol{x}_i^{(k-2^{\ell})}\|_{\overline{\boldsymbol{x}}_i}, \|\boldsymbol{s}_i^{(k)} - \boldsymbol{s}_i^{(k-2^{\ell})}\|_{\overline{\boldsymbol{x}}_i}^{*}/\overline{t}\} \ge \frac{\delta}{2\lceil \log n \rceil}$$
               and $(\overline{\boldsymbol{x}}_i, \overline{\boldsymbol{s}}_i)$ has not been updated since the $(k-2^{\ell})$-th step$\}$.
20:        $I \leftarrow I \cup I_{\ell}^{(k)}$.
21:    end for
22:    if $k \equiv 0 \bmod 2^{\lceil \log n \rceil}$ then
23:        $I \leftarrow [r]$.
24:    end if
25:    $\overline{\boldsymbol{x}}_i \leftarrow \boldsymbol{x}_i^{(k)}, \overline{\boldsymbol{s}}_i \leftarrow \boldsymbol{s}_i^{(k)}$ for all $i \in I$.
26:    return $(\overline{\boldsymbol{x}}, \overline{\boldsymbol{s}})$.
27: end procedure
```

- *Compute* $(\mathbf{M}(\bar{t}, \overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}))^{-1}$ *in time* $O(T(n,k,n))$
- *Compute* $(\mathbf{M}(\bar{t}, \overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}))^{-1}\boldsymbol{v}$ *in time* $O(T(k,k,k)+nk)$

where $T(a,b,c)$ *is the time required for multiplying an* $a \times b$ *matrix with a* $b \times c$ *matrix.*

Proof. The proof is similar to Lemma 21 in [LV21]; we include it for completeness. First, we note that sparses updates on $\overline{\boldsymbol{x}}$ and $\overline{\boldsymbol{s}}$ do not directly translate to sparse update on $\mathbf{M}$. Recall the definition of δ_μ:

$$\delta_\mu(t, \boldsymbol{x}, \boldsymbol{s})_i \overset{\text{def}}{=} -\alpha \cdot \frac{\sinh(\lambda\gamma(t, \boldsymbol{x}, \boldsymbol{s})_i)}{\|\cosh(\lambda\gamma(t, \boldsymbol{x}, \boldsymbol{s}))\|_2} \cdot \mu_i(t, \boldsymbol{x}, \boldsymbol{s}),$$

where we rescale by $\alpha/\|\cosh(\lambda\gamma(t, \boldsymbol{x}, \boldsymbol{s}))\|_2$ to make sure its norm is bounded by α. We can decouple δ_μ by let $\delta_\mu = h \cdot \widehat{\delta_\mu}$ where $h = -\alpha/\|\cosh(\lambda\gamma(t, \boldsymbol{x}, \boldsymbol{s}))\|_2$ and $\widehat{\delta_\mu}(t, \boldsymbol{x}, \boldsymbol{s})_i \overset{\text{def}}{=} \sinh(\lambda\gamma(t, \boldsymbol{x}, \boldsymbol{s})_i) \cdot \mu_i(t, \boldsymbol{x}, \boldsymbol{s})$.

Moreover, we note that by plugging $\boldsymbol{v} = h\boldsymbol{u}$ in Lemma 3.4, we get

$$\begin{bmatrix} \mathbf{M} & h\boldsymbol{u} \\ 0 & -1 \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{M}^{-1} & h\mathbf{M}^{-1}\boldsymbol{u} \\ 0 & -1 \end{bmatrix}.$$

Then, we define $\widehat{\mathbf{M}}$ by

$$\widehat{\mathbf{M}}(t, \boldsymbol{x}, \boldsymbol{s}) \overset{\text{def}}{=} \begin{bmatrix} \nabla^2\phi(\boldsymbol{x}) & \mathbf{I}/t & \mathbf{0} & \widehat{\delta_\mu}(t, \boldsymbol{x}, \boldsymbol{s}) \\ \mathbf{A} & \mathbf{0} & \mathbf{0} & 0 \\ \mathbf{0} & \mathbf{I} & \mathbf{A}^\top & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}.$$

Then, it suffices to show that we can maintain $\widehat{\mathbf{M}}$ under sparse changes of blocks. Since the total dimension of the updating blocks can be bounded by k, then we have $\widehat{M}(\bar{t}, \overline{\boldsymbol{x}}, \overline{\boldsymbol{s}}) - \widehat{M}(\bar{t}, \overline{\boldsymbol{x}}^{(\text{prev})}, \overline{\boldsymbol{s}}^{(\text{prev})})$ lives in a $(k+1)\times(k+1)$ submatrix. For simplicity of notation, we denote $\mathbf{M}_0 = \widehat{M}(\bar{t}, \overline{\boldsymbol{x}}^{(\text{prev})}, \overline{\boldsymbol{s}}^{(\text{prev})})$ and $\mathbf{M}_1 = \widehat{M}(\bar{t}, \overline{\boldsymbol{x}}, \overline{\boldsymbol{s}})$.

Hence, we can rewrite

$$\mathbf{M}_1 = \mathbf{M}_0 + \mathbf{UCV},$$

where $\mathbf{U} \in \mathbb{R}^{n \times k+1}$ is $k+1$ column of identity matrix, $\mathbf{V}$ is $k+1$ row of identity matrix, and $\mathbf{C}$ is a $(k+1)\times(k+1)$ matrix.

Using Sherman-Morrison-Woodbury identity (Lemma 5.5), we have

$$(\mathbf{M}_1)^{-1} = \mathbf{M}_0^{-1} - \mathbf{M}_0^{-1}\mathbf{U}\left(\mathbf{C}^{-1} + \mathbf{V}\mathbf{M}_0^{-1}\mathbf{U}\right)^{-1}\mathbf{V}\mathbf{M}_0^{-1}.$$

Since U and V are just submatrices of the identity matrix, we can directly read $\mathbf{M}_0^{-1}\mathbf{U}, \mathbf{V}\mathbf{M}_0^{-1}\mathbf{U}, \mathbf{V}\mathbf{M}_0^{-1}$ from $\mathbf{M}_0^{-1}$. Hence, computing $\left(\mathbf{C}^{-1} + \mathbf{V}\mathbf{M}_0^{-1}\mathbf{U}\right)^{-1}$ takes $O(T(k,k,k))$ time. The rest of the formula we can compute in $O(T(n,k,n))$ time. The total time is $O(T(n,k,n))$.

To compute $\mathbf{M}_1^{-1}\boldsymbol{v}$, we note that

$$(\mathbf{M}_1)^{-1}\boldsymbol{v} = \mathbf{M}_0^{-1}\boldsymbol{v} - \mathbf{M}_0^{-1}\mathbf{U}\left(\mathbf{C}^{-1} + \mathbf{V}\mathbf{M}_0^{-1}\mathbf{U}\right)^{-1}\mathbf{V}\mathbf{M}_0^{-1}\boldsymbol{v}.$$

We can compute the formula above in $O(T(k,k,k)+nk)$ time since $\mathbf{M}_0^{-1}\boldsymbol{v}$ is given. $\square$

5.3 Proof of Theorem 5.1

Proof. The correctness of the algorithm directly follows by Lemma 3.3.

Now, we prove the run time. Since t is decreasing by a multiplicative factor $1-\frac{1}{\sqrt{r}\log(r)}$, the total number of iterations is $\widetilde{O}(\sqrt{r}\log(t_{\text{start}}/t_{\text{end}}))$, and the data structure restarts every $O(\sqrt{r})$ iterations.

By our choice of $\beta=\bar{\varepsilon}/2$ and $\delta=\bar{\varepsilon}$ and Theorem 5.2, the data structure at most updates $O(2^{2\ell_k}\log^2 n)$ blocks, where ℓ_k is largest integer ℓ with $k \bmod 2^\ell = 0$. Thus the run time for the data structure `approxSolution` can be bounded by $\widetilde{O}(n^\omega \log(t_{\text{start}}/t_{\text{end}}))$.

Cost of updating $\boldsymbol{v}$: We update $\boldsymbol{u}$ every 2^{ℓ_*} steps; hence we can ignore the case where $2^{2\ell_*}=1$. By guarantee of Theorem 5.2, the total number of blocks updated can be bound by

$$q=\sum_{\ell=0}^{\ell_*-1}\frac{2^{\ell_*}}{2^\ell}\cdot(2^{2\ell}\log^2 n)=O(2^{2\ell_*}\log^2 n).$$

Then, we can bound the total dimension of these blocks by $\bar{d}=\min(n^\alpha, nr^{-1/3})\cdot O(\log^2 n)$. Using Theorem 5.8, we can update $\boldsymbol{v}$ in time $\widetilde{O}(\bar{d}^\omega+n\bar{d})$. Note that for any $d<n^\alpha$, we have

$$d^\omega < d^{3-\alpha} < d\cdot d^{2-\alpha} < d\cdot(n^\alpha)^{2-\alpha} \le dn,$$

where we used $\omega \le 3-\frac{1}{2}\omega\alpha \le 3-\alpha$ in Lemma 5.7. Hence, we can bound the cost for each update by $\widetilde{O}(n\bar{d})=\widetilde{O}(n^2 r^{-1/3})$. Then, the total cost for updating $\boldsymbol{v}$ is $\widetilde{O}(n^2 r^{1/6}\log(t_{\text{start}}/t_{\text{end}}))$.

Cost of updating $\mathbf{T}$ and $\boldsymbol{u}$: We note the data structure restarts every $O(\sqrt{r})$ many steps. Here, we calculate the total cost for updating $\mathbf{T}$ during $O(\sqrt{r})$ steps:

$$\begin{aligned}\widetilde{O}\left(\sum_{\ell=\ell_*}^{\log(\sqrt{r})}\frac{\sqrt{r}}{2^\ell}T(n,2^{2\ell}n/r,n)\right) &= \widetilde{O}(n^\omega+\sqrt{r}2^{-\ell_*}T(n,\bar{d},n)) \\ &= \widetilde{O}(n^\omega)+\widetilde{O}(\sqrt{r}2^{-\ell_*})\cdot\widetilde{O}(n^{2+o(1)}+n^{\omega-\frac{\omega-2}{1-\alpha}+o(1)}\bar{d}^{\frac{\omega-2}{1-\alpha}}),\end{aligned}$$

where the first step follows by the fact that $T(n,x,n)$ is a convex function for x, so the largest term of the sum must be either the first or last one, and the second step follows by Lemma 5.6.

Now, we bound the cases $r\le n^{1-\alpha}$ and $r>n^{1-\alpha}$ separately.

For the case where $r\le n^{1-\alpha}$, we have $2^{2\ell_*}=\min\{r^{2/3},\max(1,r\cdot n^{\alpha-1})\}=1$. Then, $\bar{d}=n/r$, and

$$\begin{aligned}\widetilde{O}(\sqrt{r}2^{-\ell_*})\cdot\widetilde{O}(n^{2+o(1)}+n^{\omega-\frac{\omega-2}{1-\alpha}+o(1)}\bar{d}^{\frac{\omega-2}{1-\alpha}}) &= \widetilde{O}(\sqrt{r}\cdot n^{2+o(1)}+n^{\omega+o(1)}r^{0.5-\frac{\omega-2}{1-\alpha}}) \\ &= \widetilde{O}(n^{2.5-\alpha/2+o(1)}),\end{aligned}$$

where the second step follows by $r\le n^{1-\alpha}$.

For the case where $r>n^{1-\alpha}$, we have $\min\{r^{2/3},\max(1,r\cdot n^{\alpha-1})\}=\min\{r^{2/3},r\cdot n^{\alpha-1}\}$. Then,

$$\begin{aligned}\widetilde{O}(\sqrt{r}2^{-\ell_*})\cdot\widetilde{O}(n^{2+o(1)}+n^{\omega-\frac{\omega-2}{1-\alpha}+o(1)}\bar{d}^{\frac{\omega-2}{1-\alpha}}) &= \widetilde{O}(\sqrt{r}2^{-\ell_*})\cdot\widetilde{O}(n^{2+o(1)}) \\ &= \widetilde{O}(n^{2.5-\alpha/2+o(1)}+n^{2+o(1)}r^{1/6}),\end{aligned}$$

where the first step follows by $\bar{d}<n^\alpha$, and the second step follows by the our choice of ℓ_*.

Hence, we can bound the cost of updating $\mathbf{T}$ and $\boldsymbol{u}$ for $O(\sqrt{r})$ steps by $\widetilde{O}(n^\omega+n^{2.5-\alpha/2+o(1)}+n^{2+o(1)}r^{1/6})$.

Cost of restarting: We note that the data structure restarts for every $O(\sqrt{r})$ many steps. During each restart, we need to compute $\mathbf{M}(t, \boldsymbol{x}, \boldsymbol{s})^{-1}$, which takes $O(n^{\omega})$ time. $\square$

6 Proof of Theorems 1.2 and 1.3

Proof of Theorem 1.2. We first invoke Theorem 4.1 to reduce the maximum dimension of the cones to be n/r, then we use Lemma B.1 with $\delta = \frac{\varepsilon}{10nR^2}$ to reduce the SOCP into the convex problem with known initial feasible point $\boldsymbol{x}, \boldsymbol{s}$ with $t = 1$. We note that this requires that $\boldsymbol{x}_c = \arg\min_{\boldsymbol{x}} \sum_i \phi_i(\boldsymbol{x}_i)$. Since the barrier function ϕ_i is given explicitly as

$$\phi_i(\boldsymbol{x}_i) \overset{\text{def}}{=} -\log(\boldsymbol{x}_{i,\triangleleft}) - \log(\boldsymbol{x}_{i,\triangleleft}^2 - \|\boldsymbol{x}_{i,\triangleright}\|_2^2) - \log(R - \boldsymbol{x}_{i,\triangleleft}),$$

the problem has the closed-form solution $\boldsymbol{x}_{c,i} = (3R/4, \mathbf{0})$ for all i. We note that each ϕ_i is a 4-self-concordant barrier, and the gradient and hessian of ϕ_i can be computed in $O(n_i^2)$ time.

Since we have $\|\boldsymbol{s}+\nabla\phi(\boldsymbol{x})\|_{\boldsymbol{x}}^* \leq \delta$, we have $\gamma_i \leq \frac{1}{n}$. Hence we have $\Phi(t, \boldsymbol{x}, \boldsymbol{s}) = \sum_{i=1}^{r} \cosh(\lambda/(10n)) \leq \cosh(\lambda/128)$. Then, we can use Theorem 5.1, we get $\boldsymbol{x}$

$$\boldsymbol{c}^\top \boldsymbol{x} \leq \min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b}, \boldsymbol{x}_i \in K_i \text{ for all } i\in[r]} \boldsymbol{c}^\top \boldsymbol{x} + 4\nu t_{\text{end}}.$$

By our choice of $t_{\text{end}} = \frac{1}{4\delta^2 n}$, we have

$$\boldsymbol{c}^\top \boldsymbol{x} \leq \min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b}, \boldsymbol{x}_i \in K_i \text{ for all } i\in[r]} \boldsymbol{c}^\top \boldsymbol{x} + \delta^2.$$

We finish the proof by invoking Lemma B.1 with $\delta = \frac{\varepsilon}{10nR^2}$ and Lemma 4.2 with $\delta = \frac{\varepsilon}{2R}$. $\square$

Proof of Theorem 1.3. Since $\nabla^2\phi_i(\boldsymbol{x}_i)$ can be computed in $O(n_i^{\omega})$ time, then we can find $\boldsymbol{x}_c$ using Newton's Method in time

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

Then, it directly follows by Lemma B.1 and Theorem 5.1 with choosing $\delta = \frac{\varepsilon}{10nR^2}$. $\square$

7 Building and Testing a SOCP Solver

We implemented our theoretical results in practice by building a SOCP solver in MATLAB. Through empirical tests, we found that in practice, our algorithm has $O(n^{1.82})$ time complexity, which is faster than the worst case theoretical bound of $O^*(n^{\omega})$.

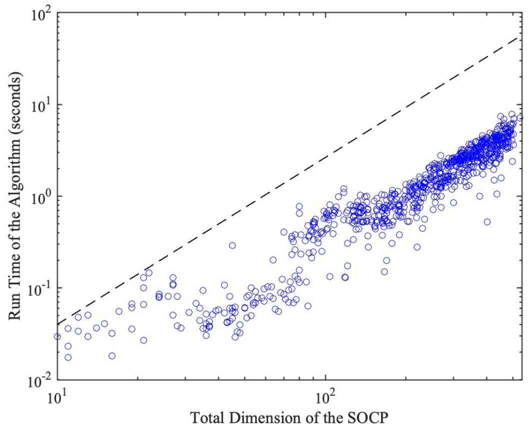
7.1 Generating a Dataset of SOCP Problems and Solutions

To evaluate our implementation of the algorithm over a comprehensive range of problem scenarios, we created a dataset of SOCP problems and solutions. We first randomly generated the constraint sizes. For each value of r from 6 to 75 and k from 6 to 20, we set $n_i = \texttt{rand} \cdot k$ and $m = \texttt{rand} \cdot n$ for each $i \in [r]$, where k is a parameter used to generate the cone sizes and `rand` is a random scalar in the interval $(0, 1)$. Then, we randomly generated the values in $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, and $\boldsymbol{c} \in \mathbb{R}^n$ accordingly. We then filtered out invalid SOCPs by running them through MATLAB's built-in SOCP solver, `coneprog`, repeating this process until 10 valid SOCP problems are generated for each combination of r and k. We created a total of 10,500 SOCP problems, and we have made this dataset of problems available to future researchers.

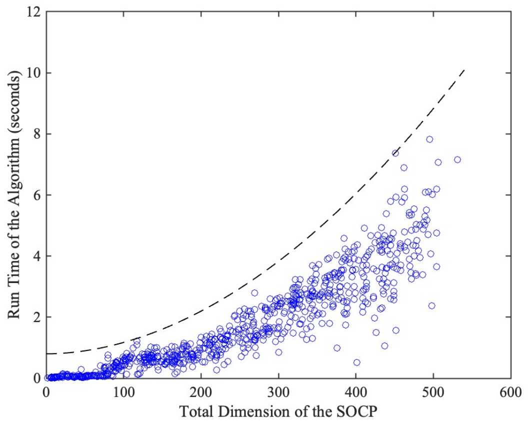
7.2 Algorithm Testing Results

To evaluate the performance of the algorithm, we tested it on the dataset we created. This dataset contained SOCP problems with n up to about 500, and the results showed that our algorithm successfully and efficiently converged to a solution for a wide range of n.

Figure 7.1 shows graphs of the run time of the algorithm with respect to n, where the run time is computed as the time it takes to reach within 0.001 of the optimal solution. For consistency, the parameter k, which is used to scale the cone sizes, is set to 10.



(a) Log-Log Plot. An upper bound line for the run time is drawn.



(b) Linear Plot. An upper bound curve for the run time is drawn.

Figure 7.1: Run time vs. Total Dimension of the SOCP

The log-log plot in Figure 7.1a is approximately linear, indicating that the time complexity of the algorithm is polynomial in n. The upper bound line has slope 1.82, which means the in-practice run time of the algorithm is bounded by $O(n^{1.82})$.

In 7.1b, the points lie below a $O(n^{1.82})$ curve. As n increases, the run time becomes increasingly scattered, with some SOCP instances converging much faster than $O(n^{1.82})$. For these cases, it is likely that the initial point in the IPM derived using B.1, which serves as an initial guess of the optimal solution, was close to the minimum point.

8 Applications

Building on the theoretical foundations of this research, we have extended its results to several applications and demonstrated significant improvements in solving problems in these areas.

8.1 Portfolio Optimization

Since Markowitz's invention of the minimum variance portfolio model, SOCPs have been widely used to solve a variety of financial portfolio optimization problems, such as risk parity portfolios, maximum diversification portfolios, liquidity-constrained portfolios, etc. The original Markowitz minimum variance portfolio finds a portfolio that meets a given desired level of return r_{min} and has the lowest variance possible. Its standard formulation is as follows [Ahm21]:

$$\begin{aligned}
&\min_{\boldsymbol{x}\in\mathbb{R}^n} \boldsymbol{x}^\top \Sigma \boldsymbol{x} \\
&s.t. \;\; \boldsymbol{x}^\top \boldsymbol{\mu} \geq r_{min}, \quad \boldsymbol{x} \geq 0, \quad \text{and} \quad \sum_{i=1}^{n} \boldsymbol{x}_i = 1
\end{aligned}$$

where $\mu \in \mathbb{R}^n$ is the estimated mean return and Σ is the estimated covariance of return. This is a QP problem, and it can solved by a SOCP solver. As the investment market requires more and more sophisticated portfolios that consider various types of risk constraints, for example, the value at risk constraints, the diversification constraints, etc., the resulting SOCP usually contains a large number of second-order cones. This represents an ideal opportunity for our algorithm to outperform the classical IPM by $O(\sqrt{r})$ times.

8.2 Support Vector Machines

Support Vector Machines (SVMs) are a versatile class of machine learning algorithms with wide applications. Its solid theoretical foundation makes it a model of choice for solving medium-sized machine learning problems even after deep learning models became popular. Given training samples $\boldsymbol{x}_i \in \mathbb{R}^n, i = 1, \ldots, l$, and class indicators $y_i \in \{-1, 1\}$, a C-support vector classifier (also called soft-margin SVM) is represented by its dual quadratic programming problem:

$$\begin{aligned}
&\min_{\boldsymbol{\omega}, b, \boldsymbol{\xi}} \|\boldsymbol{\omega}\|^2 + C \sum_{i=1}^{l} \boldsymbol{\xi}_i \\
&s.t. \;\; y_i(\boldsymbol{\omega}^\top \phi(\boldsymbol{x}_i) + b) \geq 1 - \boldsymbol{\xi}_i, \quad \text{and} \quad \boldsymbol{\xi}_i \geq 0, \qquad\qquad i = 1, \ldots, l
\end{aligned}$$

where C is a penalty parameter, and $\phi(\boldsymbol{x}_i)$ is a nonlinear mapping of x_i in a high-dimensional space. Its dual is a convex quadratic programming problem:

$$\begin{aligned}
&\min_{\boldsymbol{\alpha}_i} \frac{1}{2} \boldsymbol{\alpha}^\top Q \boldsymbol{\alpha} - \sum_{i=1}^{l} \boldsymbol{\alpha}_i \\
&s.t. \;\; \sum_{i=1}^{l} y_i \boldsymbol{\alpha}_i = 0, \quad \text{and} \quad 0 \leq \boldsymbol{\alpha}_i \leq C, \qquad\qquad i = 1, \ldots, l
\end{aligned}$$

where each $\boldsymbol{\alpha}_i$ is a Lagrange multiplier, $\boldsymbol{\alpha} = [\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_l]^\top$, $\boldsymbol{Q}$ is a positive semi-definite matrix, $\boldsymbol{Q}_{ij} \equiv y_i y_j K(x_i, x_j)$, and $K(x_i, x_j) \equiv \langle \phi(x_i) \cdot \phi(x_j) \rangle$ is the inner product kernel. The input data x_i corresponding to non-zero $\boldsymbol{\alpha}_i$ are called support vectors.

The above is an SVM in its simplest form. Real-world problems often require more complex and robust models to deal with noises in data and to avoid overfitting. By reformulating a SVM as a SOCP and applying our algorithm, the efficiency can be improved.

8.3 AC Optimal Power Flow

AC optimal power flow (ACOPF) is a fundamental tool for decision-making in electric power system analysis with applications in power system planning, operational planning, control, electricity exchange markets, etc. In [YCMS18], it is shown that ACOPF can be formulated as a SOCP with a single $(g + 1)$-dimension cone, and n 4-dimensional cones, where g is the number of power generators and n is the number of transmission lines, easily reaching 100 or 1000. With our dimension reduction techniques, the single large cone is easily decomposed into a collection of low-dimensional cones. Thus, our algorithm outperforms the classical IPM by a factor of $O(\sqrt{n})$.

9 Conclusion

Our work presents a significant advancement in second-order cone programming. Achieving $O^*(n^\omega)$ time complexity, our algorithm is more efficient than the previous fastest solution by a factor of $O^*(\sqrt{r})$, and it matches the time complexity of solving the linear sub-problem $Ax = b$. Our novel approach to decompose large constraints into smaller ones not only is crucial to reduce run time for SOCP but also is an academically significant contribution that can be applied to other kinds of conic programming. Additionally, we developed and tested a SOCP solver software, laying the groundwork for researchers to build upon theoretical advancements. The new solution shows significant performance improvement across a variety of domains, such as machine learning, operations research, energy, transportation, and finance.

Acknowledgements

We thank Prof. Jonathan A. Kelner for his guidance. We thank Dr. Tanya Khovanova, Prof. Pavel Etingof, and Dr. Slava Gerovitch of the Department of Mathematics at MIT for their advice and helpful discussions. We thank Dr. Felix Gotti and Dr. Tanya Khovanova for their feedback on this paper. We are grateful to PRIMES-USA for creating such a rare and amazing math research opportunity.

References

[AG03] Farid Alizadeh and Donald Goldfarb. Second-order cone programming. *Mathematical programming*, 95(1):3–51, 2003.

[Ahm21] A.A. Ahmadi. Convex and conic optimization, lecture 9. 2021.

[BE14] Sébastien Bubeck and Ronen Eldan. The entropic barrier: a simple and optimal universal self-concordant barrier. *arXiv preprint arXiv:1412.1587*, 2014.

[BLL$^+$21] Jan Van Den Brand, Yin Tat Lee, Yang P Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and l1-regression in nearly linear time for dense instances. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 859–869, 2021.

[BLSS20] Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 775–788, 2020.

[Bra21] Jan van den Brand. Unifying matrix data structures: Simplifying and speeding up iterative algorithms. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 1–13. SIAM, 2021.

[BTN01] Aharon Ben-Tal and Arkadi Nemirovski. On polyhedral approximations of the second-order cone. *Mathematics of Operations Research*, 26(2):193–205, 2001.

[BV03] Hande Y Benson and Robert J Vanderbei. Solving problems with semidefinite and related constraints using interior-point methods for nonlinear programming. *Mathematical Programming*, 95:279–302, 2003.

[CGLZ20] Matthias Christandl, François Le Gall, Vladimir Lysikov, and Jeroen Zuiddam. Barriers for rectangular matrix multiplication. *arXiv preprint arXiv:2003.03019*, 2020.

[Che21] Sinho Chewi. The entropic barrier is n-self-concordant. *arXiv preprint arXiv:2112.10947*, 2021.

[CLS21] Michael B. Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *J. ACM*, 68(1):3:1–3:39, 2021.

[DGG$^+$22] Sally Dong, Yu Gao, Gramoz Goranci, Yin Tat Lee, Richard Peng, Sushant Sachdeva, and Guanghao Ye. Nested dissection meets ipms: Planar min-cost flow in nearly-linear time. *CoRR*, abs/2205.01562, 2022.

[DLY20] Sally Dong, Yin Tat Lee, and Guanghao Ye. A nearly-linear time algorithm for linear programs with small treewidth: A multiscale representation of robust central path. *CoRR*, abs/2011.05365, 2020.

[Gol02] Donald Goldfarb. The simplex method for conic programming. *CORC Report*, 5, 2002.

[GSZ23] Yuzhou Gu, Zhao Song, and Lichen Zhang. A nearly-linear time algorithm for structured support vector machines. *arXiv preprint arXiv:2307.07735*, 2023.

[HJS$^+$22] Baihe Huang, Shunhua Jiang, Zhao Song, Runzhou Tao, and Ruizhe Zhang. Solving sdp faster: A robust ipm framework and efficient implementation. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 233–244. IEEE, 2022.

[JKL$^+$20] Haotian Jiang, Tarun Kathuria, Yin Tat Lee, Swati Padmanabhan, and Zhao Song. A faster interior point method for semidefinite programming. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 910–918. IEEE, 2020.

[JLSW20] Haotian Jiang, Yin Tat Lee, Zhao Song, and Sam Chiu-wai Wong. An improved cutting plane method for convex optimization, convex-concave games, and its applications. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 944–953. ACM, 2020.

[Kar84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, dec 1984.

[LS14] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in o (vrank) iterations and faster algorithms for maximum flow. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 424–433. IEEE, 2014.

[LS15] Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 230–249. IEEE, 2015.

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

[LSW15] Yin Tat Lee, Aaron Sidford, and Sam Chiu-wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 1049–1065. IEEE, 2015.

[LSZ19] Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. In Alina Beygelzimer and Daniel Hsu, editors, *Conference on Learning Theory, COLT 2019, 25-28 June 2019, Phoenix, AZ, USA*, volume 99 of *Proceedings of Machine Learning Research*, pages 2140–2157. PMLR, 2019.

[LV21] Yin Tat Lee and Santosh S Vempala. Tutorial on the robust interior point method. *arXiv preprint arXiv:2108.04734*, 2021.

[LVBL98] Miguel Sousa Lobo, Lieven Vandenberghe, Stephen Boyd, and Hervé Lebret. Applications of second-order cone programming. *Linear algebra and its applications*, 284(1-3):193–228, 1998.

[LY21] Yin Tat Lee and Man-Chung Yue. Universal barrier is n-self-concordant. *Mathematics of Operations Research*, 46(3):1129–1148, 2021.

[Nem04] Arkadi Nemirovski. Interior point polynomial time methods in convex programming. *Lecture notes*, 42(16):3215–3224, 2004.

[Nes89] Ju E Nesterov. Self-concordant functions and polynomial-time methods in convex programming. *Report, Central Economic and Mathematic Institute, USSR Acad. Sci*, 1989.

[Nes03] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2003.

[NN94] Yurii Nesterov and Arkadii Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994.

[QSZZ23] Lianke Qin, Zhao Song, Lichen Zhang, and Danyang Zhuo. An online and unified algorithm for projection matrix vector multiplication with application to empirical risk minimization. In *International Conference on Artificial Intelligence and Statistics*, pages 101–156. PMLR, 2023.

[Siv02] Kartik Krishnan Sivaramakrishnan. *Linear programming approaches to semidefinite programming problems*. Rensselaer Polytechnic Institute, 2002.

[Vai87] Pravin M Vaidya. An algorithm for linear programming which requires o (((m+ n) n 2+(m+ n) 1.5 n) l) arithmetic operations. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 29–38, 1987.

[Woo50] Max A Woodbury. *Inverting modified matrices*. Department of Statistics, Princeton University, 1950.

[YCMS18] P. Naga Yasasvi, Snehil Chandra, A. Mohapatra, and S. C. Srivastava. An exact socp formulation for ac optimal power flow. In *2018 20th National Power Systems Conference (NPSC)*, pages 1–6, 2018.

[Zha19] Vitaly Zhadan. A variant of the simplex method for second-order cone programming. In Michael Khachay, Yury Kochetov, and Panos Pardalos, editors, *Mathematical Optimization Theory and Operations Research*, pages 115–129, Cham, 2019. Springer International Publishing.

[Zha20] Vitaly Zhadan. The dual simplex-type method for linear second-order cone programming problem. In Nicholas Olenev, Yuri Evtushenko, Michael Khachay, and Vlasta Malkova, editors, *Optimization and Applications*, pages 301–316, Cham, 2020. Springer International Publishing.

A Proof of Theorem 4.3

First, we state the main result of [LSZ19].

Theorem A.1 ([LSZ19, Theorem C.3]). *Consider a convex problem* $\min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b},\boldsymbol{x}\in\prod_{i=1}^r K_i} \boldsymbol{c}^\top \boldsymbol{x}$ *where* $\mathbf{A}\in\mathbb{R}^{m\times n}$ *and* K_i*'s are compact convex set lives in constant dimension. Define the following parameters:*

1. *Radius* R*: Assume* $\|\mathbf{A}\|_F \le R$*, and* $\|\boldsymbol{b}\|_2 \le R$ *and any feasible solution* $\boldsymbol{x}$ *satisfies* $\|\boldsymbol{x}\|_2 \le R$*.*
2. *Lipschitz constant of the program:* $\|\boldsymbol{c}\|_2 \le L$*.*

There is an algorithm outputs a vector $\boldsymbol{x}\in\prod_{i=1}^m K_i$ *such that*

$$\boldsymbol{c}^\top\boldsymbol{x} \le \min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b},\boldsymbol{x}\in\prod_{i=1}^r K_i} \boldsymbol{c}^\top\boldsymbol{x} + \varepsilon L,$$
$$\|\mathbf{A}\boldsymbol{x}-\boldsymbol{b}\|_1 \le \epsilon$$

in expected time

$$\widetilde{O}\left(\left(n^\omega + n^{2.5-\alpha/2+o(1)} + n^{2+1/6+o(1)}\right)\cdot\log\left(\frac{R}{\epsilon}\right)\right).$$

Now, we are ready to prove the theorem.

Proof of Theorem 4.3. We first call Theorem 4.1 with $d=100$ to break all the large cone constraints into constant size. We denote the modified SOCP as $\min_{\overline{\mathbf{A}}\overline{\boldsymbol{x}}=\overline{\boldsymbol{b}},\overline{\boldsymbol{x}}\in\overline{\mathcal{L}}}\langle\overline{\boldsymbol{c}},\overline{\boldsymbol{x}}\rangle$. We note that the Theorem A.1 requires each K_i to be compact, it suffices to add an inequality constraint $x_{i,\triangleleft}\le 2R$ to make the set compact.

Then, we run the algorithm in Theorem A.1 with $\varepsilon' = \varepsilon/(10R)$ on $\min_{\overline{\mathbf{A}}\overline{\boldsymbol{x}}=\overline{\boldsymbol{b}},\overline{\boldsymbol{x}}\in\overline{\mathcal{L}}}\overline{\boldsymbol{c}}^\top\overline{\boldsymbol{x}}$. The runtime directly follows by the runtime of Theorem A.1.

We complete the proof by using Lemma 4.2 to show $\boldsymbol{x}$ recovered have ε relative accuracy. $\square$

B Initial Point Reduction

The RIPM requires an initial feasible point $(\boldsymbol{x}^{(0)},\boldsymbol{s}^{(0)})$ close to the central path as input. We use the reduction from [LSZ19].

Lemma B.1. *Consider the convex program*

$$\min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b},\boldsymbol{x}_i\in K_i \text{ for all } i\in[r]} \boldsymbol{c}^\top\boldsymbol{x},$$

where $\mathbf{A}\in\mathbb{R}^{m\times n}$*,* $\boldsymbol{b}\in\mathbb{R}^m$*, and* $\boldsymbol{c}\in\mathbb{R}^n$*. Given* ν_i*-self-concordant barrier* ϕ_i *for each* K_i*, and* $\boldsymbol{x}_c = \arg\min_{\boldsymbol{x}}\sum_i\phi_i(\boldsymbol{x}_i)$*, define the following parameters:*

1. *Outer radius* R*: Assume* $\|\mathbf{A}\|_F\le R$*, and* $\|\boldsymbol{b}\|_2\le R$*, and any feasible solution* $\boldsymbol{x}$ *satisfies* $\|\boldsymbol{x}\|_2\le R$
2. *Lipschitz constant* L*:* $\|\boldsymbol{c}\|_2\le L$*.*

For any $\delta>0$*, the modified convex program*

$$\min_{\overline{\mathbf{A}}\overline{\boldsymbol{x}}=\overline{\boldsymbol{b}},\boldsymbol{x}\in\prod_{i=1}^r K_i\times\mathbb{R}_+}\overline{\boldsymbol{c}}^\top\overline{\boldsymbol{x}},$$

where

$$\overline{\mathbf{A}} = [\mathbf{A}\mid\boldsymbol{b}-\mathbf{A}\boldsymbol{x}_c],\quad \overline{\boldsymbol{b}}=\boldsymbol{b},\quad \overline{\boldsymbol{c}} = [\tfrac{\delta}{LR}\boldsymbol{c}\mid 1]^\top,$$

satisfies the following:

1. $\overline{\boldsymbol{x}} = (\boldsymbol{x}_c, 1), \overline{\boldsymbol{s}} = (\frac{\delta}{LR}\boldsymbol{c}, 1)$ *is a feasible primal dual vector with* $\|\overline{\boldsymbol{x}} + \nabla\bar{\phi}(\overline{\boldsymbol{x}})\|^*_{\overline{\boldsymbol{x}}} < \delta$ *to the modified convex program, where* $\bar{\phi}(\overline{\boldsymbol{x}}) = \sum_{i=1}^r \phi_i(\overline{\boldsymbol{x}}_i) - \log(\overline{\boldsymbol{x}}_{r+1})$.

2. *For any feasible* $\overline{\boldsymbol{x}}$ *to the modified convex program and* $\overline{\boldsymbol{c}}^\top \overline{\boldsymbol{x}} \leq \min_{\overline{\mathbf{A}}\overline{\boldsymbol{x}}=\overline{\boldsymbol{b}}, \boldsymbol{x}\in\prod_{i=1}^r K_i \times \mathbb{R}_+} \overline{\boldsymbol{c}}^\top \overline{\boldsymbol{x}} + \delta^2$*, the vector* $\boldsymbol{x} = \overline{\boldsymbol{x}}_{1:r}$ *is an approximate solution to the original convex program in the following sense:*

$$\begin{aligned} \boldsymbol{c}^\top \boldsymbol{x} &\leq \min_{\mathbf{A}\boldsymbol{x}=\boldsymbol{b}, \boldsymbol{x}_i \in K_i \textit{ for all } i\in[r]} + LR \cdot \delta, \\ \|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|_1 &\leq 3\delta(R\|\mathbf{A}\|_1 + \|\boldsymbol{b}\|_1), \\ \boldsymbol{x}_i &\in K_i \textit{ for all } i \in [r]. \end{aligned}$$