A Deterministic Linear Program Solver in Current Matrix Multiplication Time

Jan van den Brand[1]

[1]KTH Royal Institute of Technology, Sweden

Interior point algorithms for solving linear programs have been studied extensively for a long time [e.g. Karmarkar 1984; Lee, Sidford FOCS'14; Cohen, Lee, Song STOC'19]. For linear programs of the form $\min_{Ax=b,x\geq 0} c^\top x$ with n variables and d constraints, the generic case $d = \Omega(n)$ has recently been settled by Cohen, Lee and Song [STOC'19]. Their algorithm can solve linear programs in $\widetilde{O}(n^\omega \log(n/\delta))$ expected time[1], where δ is the relative accuracy. This is essentially optimal as all known linear system solvers require up to $O(n^\omega)$ time for solving $Ax = b$. However, for the case of *deterministic* solvers, the best upper bound is Vaidya's 30 years old $O(n^{2.5} \log(n/\delta))$ bound [FOCS'89]. In this paper we show that one can also settle the deterministic setting by derandomizing Cohen et al.'s $\widetilde{O}(n^\omega \log(n/\delta))$ time algorithm. This allows for a strict $\widetilde{O}(n^\omega \log(n/\delta))$ time bound, instead of an expected one, and a simplified analysis, reducing the length of their proof of their central path method by roughly half. Derandomizing this algorithm was also an open question asked in Song's PhD Thesis.

The main tool to achieve our result is a new data-structure that can maintain the solution to a linear system in subquadratic time. More accurately we are able to maintain $\sqrt{U}A^\top(AUA^\top)^{-1}A\sqrt{U}v$ in subquadratic time under ℓ_2 multiplicative changes to the diagonal matrix U and the vector v. This type of change is common for interior point algorithms. Previous algorithms [e.g. Vaidya STOC'89; Lee, Sidford FOCS'15; Cohen, Lee, Song STOC'19] required $\Omega(n^2)$ time for this task. In [Cohen, Lee, Song STOC'19] they managed to maintain the matrix $\sqrt{U}A^\top(AUA^\top)^{-1}A\sqrt{U}$ in subquadratic time, but multiplying it with a dense vector to solve the linear system still required $\Omega(n^2)$ time. To improve the complexity of their linear program solver, they restricted the solver to only multiply sparse vectors via a random sampling argument. In comparison, our data-structure maintains the entire product $\sqrt{U}A^\top(AUA^\top)^{-1}A\sqrt{U}v$ additionally to just the matrix. Interestingly, this can be viewed as a simple modification of Cohen et al.'s data-structure, but it significantly simplifies their analysis of their central path method and makes their whole algorithm deterministic.

[1] Here $\widetilde{O}$ hides polylog(n) factors and $O(n^\omega)$ is the time required to multiply two $n \times n$ matrices. The stated $\widetilde{O}(n^\omega \log(n/\delta))$ bound holds for the current bound on ω with $\omega \approx 2.38$ [V.Williams, STOC'12; Le Gall, ISSAC'14]. The upper bound for the solver will become larger than $\widetilde{O}(n^\omega \log(n/\delta))$, if $\omega < 2 + 1/6$.

Contents

1 Introduction

Fast algorithms for solving linear programs have a long history in computer science. Solving linear programs was first proven to be in P in 1979 by Khachiyan [Kha79]; and later Karmarkar [Kar84] found the first polynomial time algorithm that was feasible in practice. This initiated the long line of work of solving linear programs using interior point algorithms, motivated by the fact that many problems can be stated as linear programs and solved using efficient solvers. [Ren88, Vai87, Vai89b, Vai89a, Meg89, NN89, NN91, VA93, Ans96, NT97, Ans99, LS14, LS15]

For linear programs of the form $\min_{Ax=b,x\geq 0} c^\top x$ with n variables, d constraints and $nnz(A)$ non-zero entries, the current fastest algorithms are $\widetilde{O}(\sqrt{d}(nnz(A)+d^2))$ [LS14, LS15][2] and $\widetilde{O}(n^\omega)$-time [CLS18][3], where the bound $O(n^\omega)$ is the number of arithmetic operations required to multiply two $n \times n$ matrices.[4] For the generic case $d = \Omega(n)$, the latter complexity is essentially optimal as all known linear system solvers require up to $O(n^\omega)$ time for solving $Ax = b$. As the complexity is essentially optimal, but the algorithm is randomized, a typical next step (e.g. [KT19, Cha00, PR02, MRSV17]) is to attempt to derandomize this algorithm. Derandomizing algorithms has the benefit that the required analysis can lead to further understanding of the studied problem. There have been precedences where derandomizing algorithms required developing new techniques, which then allowed for improvements in other settings. For example, in order to derandomize Karger's edge connectivity algorithm [Kar00], Kawarabayashi and Thorup [KT19] had to develop new techniques, which then lead to new results in the distributed setting [DHNS19].

In the related area of linear program solvers in the *real RAM model* (i.e. when analyzing the complexity only in terms of the dimension, but not the bit-complexity of the input), a lot of effort has been put in derandomization and finding fast deterministic algorithms (see e.g. [CM96, BCM99, Cha16]). Yet, there is still a wide gap between the best randomized and deterministic complexity bounds.[5] The same observation can be made in our setting, when analyzing the complexity with respect to the bit-complexity of the input, where the best deterministic bounds are $\widetilde{O}(\sqrt{n}\cdot nnz(A)+nd^{1.38})$ [Kar84][6], $\widetilde{O}(\sqrt{n}\cdot nnz(A)+n^{1.34}d^{1.15})$ [Vai89b] and $\widetilde{O}(d\cdot nnz(A)+d^{\omega+1})$ [Vai89a].[7] For $d = \Omega(n)$, all deterministic algorithms are stuck at $\Omega(n^{2.5})$ time. Further, these bounds are at least 30 years old and all new algorithms, that have been able to improve upon these bounds, crucially use randomized techniques. This raises the question: *Is there a deterministic algorithm that can close the gap between deterministic and randomized complexity bounds, or at least break the 30 years old $\Omega(n^{2.5})$ barrier?*

We are able to answer this question affirmatively by derandomizing the algorithm of Cohen et al. [CLS18]. Our deterministic algorithm is not just able to break the 30 years old barrier, it even matches one of the fastest randomized bounds of $\widetilde{O}(n^\omega)$. This closes the complexity gap between randomized and deterministic algorithms for large d. More formally, we prove the following result:

[2] Here $\widetilde{O}(\cdot)$ hides polylog(n) and polylog$(1/\delta)$ terms.

[3] The algorithm of [CLS18] runs in $O((n^\omega + n^{2.5-\alpha/2+o(1)} + n^{2+1/6})\log(n)\log(n/\delta))$ time, where δ is the relative accuracy and α is the dual matrix exponent. The dual exponent α is the largest a such that an $n \times n$ matrix can be multiplied with an $n \times n^a$ matrix in $n^{2+o(1)}$ arithmetic operations. For current $\omega \approx 2.38$ and $\alpha \approx 0.31$ this time complexity is just $O(n^\omega \log(n)\log(n/\delta))$.

[4] The parameter ω is also called *matrix exponent*.

[5] For an overview see [Cha16]. The fastest deterministic algorithm requires $O(nd^{d(1/2+o(1))})$ time [Cha16], while with randomized techniques an $O(nd^2 + \exp(O(\sqrt{d\log d})))$ time bound is possible (a combination of [Cla95, Kal92, MSW96]).

[6] When using the $\widetilde{O}(\sqrt{n})$-iterations short step method.

[7] For curious readers we recommend [LS15]. They give a brief overview of these algorithms and offer a helpful graph that shows which algorithm is fastest for which range of $n, d, nnz(A)$.

Theorem 1.1. *Let $\min_{Ax=b,x\geq 0} c^\top x$ be a linear program without redundant constraints. Let R be a bound on $\|x\|_1$ for all $x \geq 0$ with $Ax = b$. Then for any $0 < \delta \leq 1$ we can compute $x \geq 0$ such that*

$$c^\top x \leq \min_{Ax=b,x\geq 0} c^\top x + \delta \|c\|_\infty R \qquad \text{and} \qquad \|Ax - b\|_1 \leq \delta \left(R \sum_{i,j} |A_{i,j}| + \|b\|_1 \right)$$

in time $O(n^\omega \log^2(n) \log(n/\delta))$, for the current matrix multiplication time with $\omega \approx 2.38$ [Wil12, Gal14].

Remark 1.2. The real complexity of Theorem 1.1 is

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

which can be simplified to $O(n^\omega \log^2(n) \log(n/\delta))$ for current values of $\omega \approx 2.38$ [Wil12, Gal14], $\alpha \approx 0.31$ [GU18]. For integral A, b, c the parameter $\delta = 2^{-O(L)}$ is enough to round the approximate solution of Theorem 1.1 to an exact solution. Here $L = \log(1 + \det_{\max} + \|c\|_\infty + \|b\|_\infty)$ is the bit-complexity, where $\det_{\max}$ is the largest determinant of any square submatrix of A. [Ren88, LS13]

Derandomizing the $\widetilde{O}(n^\omega)$ algorithm of [CLS18] was stated by Song as an open question in [Son19]. In addition to answering this open question, our techniques also allow us to simplify the analysis of the central path method used in [CLS18], reducing the length by roughly half.

Technical Ideas Interior point algorithms must typically repeatedly compute the projection of a certain vector v, i.e. they must compute Pv where P is a projection matrix. It suffices to use an approximation $\widetilde{P}$ of P, and in each iteration the matrix P changes only a bit, which allowed previous results to maintain the approximation $\widetilde{P}$ quickly (See for example [Kar84, NN89, Vai89b, LS15]). A natural barrier for improving linear program solvers is the fact that computing $\widetilde{P}v$ requires $\Omega(n^2)$ for dense v. This leads to the $\Omega(n^{2.5})$ barrier for linear program solvers, because a total of $\Omega(\sqrt{n})$ projections must be computed. Cohen et al. were able to break this barrier in [CLS18] by sparsifying v to some approximate $\widetilde{v}$ via random sampling and computing $\widetilde{P}\widetilde{v}$ instead of $\widetilde{P}v$. Our new approach for breaking this barrier deterministically is to maintain the product $\widetilde{P}\widetilde{v}$ directly for some approximation $\widetilde{v}$ of v. This is the key difference of our linear program solver compared to previous results, which only maintained $\widetilde{P}$.

We will now outline the difference between our deterministic $\widetilde{O}(n^\omega)$ solver for linear programs and the randomized result of Cohen et al. [CLS18]. They managed to obtain a fast solver for linear programs by computing the projection $\widetilde{P}\widetilde{v}$ in subquadratic time using two clever tools:

(1) They created a data-structure to maintain $\widetilde{P}$ in sub-quadratic time, amortized over $\sqrt{n}$ iterations.

(2) They created a novel stochastic central path method which can sparsify the vector v to some approximate $\widetilde{v}$ via random sampling. Thus the projection $\widetilde{P}\widetilde{v}$ could be computed in sub-quadratic worst-case time.

Derandomizing this algorithm seems like a difficult task as it is not clear how to obtain a deterministic sparsification of v. Recently [LSZ19] derandomized the central path method (2), so they could extend their linear program solver to the problem of Empirical Risk Minimization. However, in order to achieve $\widetilde{O}(n^\omega)$ total time, they had to reduce some dimension in the representation of $\widetilde{P}$ via random sketching, which resulted in randomizing the data-structure (1).

In this paper we show how to completely derandomize the algorithm of [CLS18] via a data-structure that can maintain the projection $\widetilde{P}\widetilde{v}$ directly for some *dense* approximate $\widetilde{v} \approx v$, instead

of just maintaining the matrix $\widetilde{P}$ as in [CLS18]. This result can be obtained in two different ways. One option is to use a dynamic linear system algorithm (e.g. [San04, vdBNS19]) via a black-box reduction, or alternatively one can interpret the resulting data-structure as a surprisingly simple extension of the data-structure used in [CLS18]. Indeed the algorithmic description of the data-structure (1) of [CLS18] grows only by a few lines (see Algorithm 1 in Section 4).

The high level idea of our new data-structure is that the vector v can be written as some function $v_i = f(w_i)$, where the argument vector w does not change much between two iterations of the central path method. By approximating w by some $\widetilde{w}$ we can re-use information of the previous iteration when computing the projection of $\widetilde{v} = f(\widetilde{w})$. One difficulty, that we must overcome, is that $\widetilde{v} := f(\widetilde{w})$ is not an approximation of $v = f(w)$ in the classical sense (i.e. we can not satisfy $\|v - \widetilde{v}\| \leq \varepsilon \|v\|$ or even $\widetilde{v}_i \approx v_i$), even if $\widetilde{w}$ is an approximation of w, because for non-monotonous f, the vectors v and $\widetilde{v}$ could point in opposite directions. This is a problem for the short step central path method, because these algorithm can be interpreted as some gradient descent and here v depends on the gradient of some potential function. So if $\widetilde{v}$ points in the opposite direction, then the algorithm will actually increase the potential function instead of decreasing it.

To solve this issue, we must also perform some small modifications to the short step central path method, so it is able to handle our approach of "approximating" v. The main modification to the short step central path method is that we measure our progress with respect to the ℓ_∞-norm, similar to [AB95, CLS18]. The proof for this will be based on [CLS18], where their random sampling (2) $\widetilde{v}$ of v can be interpreted as some approximation of v. This allows us to adapt their proof to our non-standard way of approximating $v = f(w)$ via $\widetilde{v} = f(\widetilde{w})$. The removal of all randomized components from their proof also allows us to reduce the length of their central path analysis by roughly half. This reduction of the analysis, together with the simple extension of their data-structure (1) to maintain $\widetilde{P}\widetilde{v}$, is an interesting difference to other derandomization results, where complicated tools need to be created for replacing the randomized components.

2 Outline

In this section we outline how our algorithm works and how we adapt existing ideas such as the short step central path method and the projection maintenance. Readers only interested in verifying our algorithm can skip this overview, but reading it can help provide some intuition for how the different parts of our algorithm interact and what difficulties must be solved in order for our algorithm to work.

We start the outline with a brief summary of the short step central path method, which motivates why we must maintain a certain projection. Readers already familiar with the short step central path method can skip ahead to the next subsection 2.2.

In Section 2.2 we describe the task of the projection maintenance, and how we are able to perform this task quickly by using a certain notion of approximation (details in Section 4). The next Section 2.3 of the outline explains the difficulties that we encounter by using this type of approximation, and how we are able to solve these problems (details in Section 5).

Before outlining our linear program solver, we want to quickly define some important notation: For two n dimensional vectors v, w we write vw for the entry-wise product and v/w for the entry-wise division, so $(vw)_i := v_i w_i$ and $(v/w)_i := v_i/w_i$. For a scalar s the product sv is the typical entry-wise product and analogously we define $v - s$ as the entry-wise difference, so $(v-s)_i := v_i - s$. For two vectors v, w, we write $v \leq w$ if $v_i \leq w_i$ for all $i = 1, ..., n$. We write $\|v\|_p$ for the ℓ_p-norm, so $\|v\|_p = (\sum_{i=1}^n |v_i|^p)^{1/p}$ for $0 < p < \infty$ and $\|v\|_\infty = \max_i |v_i|$.

2.1 Short Step Central Path Method

We first give a brief summary of the short step central path method. Readers familiar with these types of algorithms can skip ahead to the next subsection 2.2.

Consider the linear program $\min_{Ax=b,x\geq 0} c^\top x$ and its dual program $\max_{A^\top y\leq c} b^\top y$. Given a feasible dual solution y (a vector y s.t. $A^\top y \leq c$), we can define the slack vector $s := c - A^\top y$. Based on the complementary slackness condition (see e.g. [PS82]) we know a triple (x, y, s) is optimal, if and only if

$$\begin{aligned} x_i s_i &= 0 \text{ for all } i, \\ Ax &= b, \\ A^\top y + s &= c, \\ x_i, s_i &\geq 0 \text{ for all } i. \end{aligned}$$

If only the last three conditions are satisfied, then we call the triple (x, y, s) feasible. Given such a feasible triple, we define the vector μ such that $\mu_i := x_i s_i$ and the complementary slackness theorem motivates why we should try to minimize the entries of μ.

It is known how to transform the LP in such a way, that we can easily construct a feasible solution triple (x, y, s) with $x_i s_i \approx 1$ for all $i = 1, ..., n$ (e.g. Lemma A.3 [YTM94]). Thus for $t := 1$ we have $\mu_i \approx t$. The idea is to repeatedly decrease t and to modify the solution $x \leftarrow x + \delta_x, y \leftarrow y + \delta_y, s \leftarrow s + \delta_s$ in such a way, that the entries of μ stay close to t. The change of μ is given by $\mu_i^{\text{new}} = (x + \delta_x)_i (s + \delta_s)_i = \mu_i + x_i \delta_{s,i} + s_i \delta_{x,i} + \delta_{x,i}\delta_{s,i}$ and if δ_x, δ_s are small enough, this can be approximated via $\mu_i^{\text{new}} \approx \mu_i + x_i \delta_{s,i} + s_i \delta_{x,i}$. Thus to change μ by (approximately) δ_μ, we can solve the following linear system

$$\begin{aligned} X\delta_s + S\delta_x &= \delta_\mu, \\ A\delta_x &= 0, \\ A^\top \delta_y + \delta_s &= 0, \end{aligned} \tag{1}$$

where $X = \text{diag}(x)$ and $S = \text{diag}(s)$ are diagonal matrices with the entries of x and s on the diagonal respectively. The solution to this system is given by the following lemma:

Lemma 2.1 ([CLS18])**.** *The solution for δ_x, δ_s in (1) is given by*

$$\delta_x = \frac{X}{\sqrt{XS}}(I - P)\frac{1}{\sqrt{XS}}\delta_\mu \text{ and } \delta_s = \frac{S}{\sqrt{XS}}P\frac{1}{\sqrt{XS}}\delta_\mu.$$

where

$$P := \sqrt{\frac{X}{S}}A^\top \left(A\frac{X}{S}A^\top\right)^{-1} A\sqrt{\frac{X}{S}}.$$

A typical choice for the decrement of t is to multiply it by $1 - O(\frac{1}{\sqrt{n}})$, which means it takes about $O(\sqrt{n}/\delta)$ iterations until t reaches some desired accuracy parameter $\delta > 0$ [Ren88, Vai87].

For the short step central path method the distance between μ and t is typically measured by $\sum_{i=1}^n (\mu_i - t)^2 = \|\mu - t\|_2^2$ and one tries to maintain x, s in such a way that $\|\mu - t\|_2^2 \leq O(t^2)$. This can be modelled via the potential function $\Phi(x) = \|x\|_2^2$, and then one tries to maintain μ such that $\Phi(\mu/t - 1) = O(1)$, which is equivalent to $\|\mu - t\|_2^2 \leq O(t^2)$. Thus a good choice for δ_μ would be a vector with the same direction as $-\nabla\Phi(\mu/t - 1)$, as this allows us to decrease the potential, which then means the distance between μ and t is reduced.

2.2 Projection Maintenance (Details in Section 4)

In this subsection we outline one of the main results of this paper and sketch its proof. As described in the previous section, we must repeatedly compute Pv for $P := \sqrt{\frac{X}{S}}A^\top \left(A\frac{X}{S}A^\top\right)^{-1} A\sqrt{\frac{X}{S}}$ and $v := \frac{\delta_\mu}{\sqrt{XS}}$, where the matrix A describes the constraints of the linear program, X and S are diagonal matrices that depend on some current feasible solution and δ_μ is some vector.

Our main result is to maintain an approximation of Pv deterministically in $\widetilde{O}(n^{\omega-0.5}+n^{2.5-\alpha})$ amortized time, where ω is the current matrix multiplication exponent and α is the dual exponent. This new data-structure is a simple extension of the data-structure presented in [CLS18], which was able to maintain an approximation of P within the same time bound, but their data-structure required up to $O(n^2)$ time for computing Pv for dense v.

The exact statement of our result involves various details, for example how P and v change over time. So we first want to describe the task of maintaining Pv in more detail.

The task The matrix $P = \sqrt{\frac{X}{S}}A^\top \left(A\frac{X}{S}A^\top\right)^{-1} A\sqrt{\frac{X}{S}}$ shares a lot of structure between two iterations. Indeed only the diagonal matrices X and S change, while the matrix A stays fixed. Thus for simplicity we define $U := X/S$, in which case $P := \sqrt{U}A^\top \left(AUA^\top\right)^{-1} A\sqrt{U}$ and only the diagonal matrix U changes from one iteration to the next one.

For this task we would wish for a data-structure that can compute Pv for any vector v in $O(n^{\omega-0.5})$ time, which with $O(\sqrt{n})$ iterations would then result in an $O(n^\omega)$-time solver for linear programs. More accurately, we hope for an algorithm that solves the following task:

Task 2.2. *Let $A \in \mathbb{R}^{d\times n}$ be a rank d matrix with $n \geq d$. We wish for a deterministic data-structure with the following operations*

- INITIALIZE(A, u, v)*: Given matrix A and two n dimensional vectors u, v we preprocess the matrix and return*
$$Pv := \sqrt{U}A^\top(AUA^\top)^{-1}A\sqrt{U}v,$$
where $U = diag(u)$ is the diagonal matrix with u on the diagonal.
- UPDATE(u, v)*: Given two n dimensional vectors u, v, we must compute*
$$Pv := \sqrt{U}A^\top(AUA^\top)^{-1}A\sqrt{U}v.$$

It is not clear whether a data-structure exists for this task with $O(n^{\omega-0.5})$ update time, but due to the very first short step linear program solver by Karmarkar [Kar84] it is known that one can relax the requirements. Indeed it is enough to use an approximation of P.

Relaxation and result Due to [Kar84] it is known, that it is enough to use an approximation $\widetilde{P} := \sqrt{\widetilde{U}}A^\top(A\widetilde{U}A^\top)^{-1}A\sqrt{\widetilde{U}}$ for $(1-\varepsilon)U \leq \widetilde{U} \leq (1+\varepsilon)U$, instead of the exact matrix P. We show later in Section 5, that it is also enough to approximate the vector v via some $\widetilde{v}$. The type of approximation for v is a bit different: We show in Section 5 that we can write $v = \delta_\mu/\sqrt{XS}$ as a function of μ/t, so $v = f(\mu/t)$. We then "approximate" v via some $\widetilde{v} := f(\widetilde{\mu}/t)$, where $(1-\varepsilon)\mu \leq \widetilde{\mu} \leq (1+\varepsilon)\mu$. Note that thus $\widetilde{v}$ itself is not necessarily an approximation of v in the classical sense (i.e. $\|v-\widetilde{v}\|_2 \gg \varepsilon\|v\|_2$) and the two vectors might even point in opposite directions.

Motivated by these observations we want to maintain $\widetilde{P}\widetilde{v}$ instead of Pv. This idea allows for a speed-up, because in each iteration we only need to change the entries of $\widetilde{u}$ and $\widetilde{\mu}$ for which the

$(1+\varepsilon)$-approximation condition is broken. Thus if the vectors u and μ do not change much per iteration, then we only need to change few entries of $\widetilde{u}$ and $\widetilde{\mu}$. We prove in Section 5.2 that u and μ satisfy the following condition:

Lemma 2.3 (Proven in Section 5.2, Lemma 5.7)**.** *Let $(u^k)_{k\geq 1}$ be the sequence of vectors u, generated by the central path method. Then $\|(u^{k+1}-u^k)/u^k\|_2 \leq C$ for all k and some constant C. (A similar statement can be made for μ)*

Thus, while we are not able to solve Task 2.2 exactly, we do obtain a data-structure that (i) maintains the solution approximately, and (ii) is fast if $\|(u^{k+1}-u^k)/u^k\|_2$ and $\|(\mu^{k+1}-\mu^k)/\mu^k\|_2$ are small.

Theorem 2.4 (Proven in Section 4, Lemma 4.1)**.** *Let $A \in \mathbb{R}^{d\times n}$ be a full rank matrix with $n \geq d$, v be an n-dimensional vector and $0 < \varepsilon_{mp} < 1/4$ be an accuracy parameter. Let $f : \mathbb{R} \to \mathbb{R}$ be some function that can be computed in $O(1)$ time, and define $f(v)$ to be the vector with $f(v)_i := f(v_i)$. Given any positive number $a \leq \alpha$ there is a deterministic data-structure with the following operations*

- Initialize$(A, u, f, v, \varepsilon_{mp})$*: The data-structure preprocesses the given two n dimensional vectors u, v, the $d \times n$ matrix A the function f in $O(n^2 d^{\omega-2})$ time. The given parameter $\varepsilon_{mp} > 0$ specifies the accuracy of the approximation.*
- Update(u, v)*: Given two n dimensional vectors u, v. Then the data-structure returns four vectors*

$$\widetilde{u}, \quad \widetilde{v}, \quad f(\widetilde{v}), \quad \sqrt{\widetilde{U}}A^\top(A\widetilde{U}A^\top)^{-1}A\sqrt{\widetilde{U}}f(\widetilde{v}).$$

Here $\widetilde{U}$ is the diagonal matrix $\mathrm{diag}(\widetilde{u})$ *and $\widetilde{v}$ a vector such that*

$$(1-\varepsilon_{mp})\widetilde{v}_i \leq v_i \leq (1+\varepsilon_{mp})\widetilde{v}_i$$

$$(1-\varepsilon_{mp})\widetilde{u}_i \leq u_i \leq (1+\varepsilon_{mp})\widetilde{u}_i.$$

If the update sequence $u^{(1)}, ..., u^{(T)}$ (and likewise $v^{(1)}, ..., v^{(T)}$) satisfies

$$\sum_{i=1}^{n}\left(\frac{u_i^{(k+1)}-u_i^{(k)}}{u_i^{(k)}}\right)^2 \leq C^2, \qquad \left|\frac{u_i^{(k+1)}-u_i^{(k)}}{u_i^{(u)}}\right| \leq 1/4, \tag{2}$$

for all $k = 1, ..., T$ then the total time for the first T updates is

$$O\left(T\cdot\left(C/\varepsilon_{mp}(n^{\omega-1/2}+n^{2-a/2+o(1)})\log n + n^{1+a}\right)\right)$$

There are two equivalent ways to prove Theorem 2.4: One could use the data-structures of [San04, vdBNS19] which maintain $M^{-1}b$ for some non-singular matrix M and some vector b. Via a black-box reduction these data-structures would then be able to maintain $\widetilde{P}\widetilde{v}$ and applying the tools of [CLS18] for optimizing the amortized complexity would then result in Theorem 2.4.

If one tries to write down a pseudo-code description of the resulting data-structure, then the code is very similar to the data-structure from [CLS18]. This is because all these data-structures are based on the Sherman-Morrison-Woodbury identity. Hence an alternative way to prove Theorem 2.4 is to take the data-structure from [CLS18], which already maintains $\widetilde{P}$, and extend such that it also maintains $\widetilde{P}\widetilde{v}$.

In this paper we present the second option, where we modify the existing data-structure of [CLS18]. This is because we want to highlight that our derandomization result can be obtained from a simple modification of the existing randomized algorithm. Though for the curious reader we also give a sketch of the first variant in Appendix B.

Proof sketch (Details in Section 4) We now outline how to obtain Theorem 2.4 by extending the data-structure of [CLS18] to also maintain $\widetilde{P}\widetilde{v}$, instead of just $\widetilde{P}$. Their data-structure internally has three matrices M, L, R with the property

$$\widetilde{P} = M + LR^\top \tag{3}$$

where M is some $n \times n$ matrix and L, R are rectangular matrices with some $m \ll n$ columns. With each update, the matrices L, R change and the number of their columns may increase. This way the n^2 entries of the matrix $\widetilde{P}$ are not explicitly computed and a sub-quadratic update time can be achieved.

As the number of columns m of L, R grows, the data-structure will become slower and slower. Once these matrices have too many columns, the data-structure performs a "reset". This means we set

$$M \leftarrow M + LR^\top, \tag{4}$$

and the matrices L, R are set to be empty (so zero columns). Thus after the reset we have $\widetilde{P} = M + LR^\top = M$, so (3) is still satisfied. Such a reset requires $\Omega(n^2)$ time, but it does not happen too often so the cost is small on average.[8]

One can now easily maintain $\widetilde{P}f(\widetilde{v})$ as follows: Assume we already know $Mf(\widetilde{v})$, then a new solution $\widetilde{P}f(\widetilde{v})$ is given by

$$\widetilde{P}f(\widetilde{v}) = Mf(\widetilde{v}) + LR^\top f(\widetilde{v}), \tag{5}$$

because of (3). Here the term $LR^\top f(\widetilde{v})$ can be computed in $O(nm) \ll O(n^2)$ time, because L, R have $m \ll n$ columns. The assumption, that $Mf(\widetilde{v})$ is known, can be satisfied easily: During the initialization of the algorithm we compute this value, and whenever M changes (i.e. during the reset (4)) we can compute the new $Mf(\widetilde{v})$ in $O(n^2)$ time. This does not affect the complexity of the data-structure, because a reset does already require $\Omega(n^2)$ time to compute the new M.

At last, we must handle the case where entries of $\widetilde{v}$ are changed. Let's say $\widetilde{v}^{\text{new}} \leftarrow \widetilde{v} + \delta_v$, then

$$\widetilde{P}f(\widetilde{v}^{\text{new}}) = \widetilde{P}f(\widetilde{v}) + \widetilde{P}(f(\widetilde{v}^{\text{new}}) - f(\widetilde{v})) = \widetilde{P}f(\widetilde{v}) + M(f(\widetilde{v}^{\text{new}}) - f(\widetilde{v})) + LR^\top(f(\widetilde{v}^{\text{new}}) - f(\widetilde{v})),$$

where the last equality comes from (3). The complexity can be bounded as follows: The term $\widetilde{P}f(\widetilde{v})$ is computed as described in (5). The second term $M(f(\widetilde{v}^{\text{new}}) - f(\widetilde{v}))$ can be computed quickly because on average $\widetilde{v}^{\text{new}}$ and $\widetilde{v}$ differ in only few entries, because of the small change to v per iteration (as given by (2) of Theorem 2.4). The last term $LR^\top(f(\widetilde{v}^{\text{new}}) - f(\widetilde{v}))$ is again computed quickly because the matrices L, R have very few columns.

2.3 Adapting the Central Path Method for Approximate Projection Maintenance (Details in Section 5)

We now outline difficulties that occur, if one tries to use the projection maintenance algorithm (Theorem 2.4, outlined in Section 2.2) in the classical central path method (outlined in Section 2.1), and how we are able to solve these issues in Section 5.

The central path method can be interpreted as some gradient descent, where we try to minimize some potential. When we use the data-structure of Theorem 2.4, then we are essentially performing this gradient descent while using some approximate gradient. This approximation is of such low

[8] Section 5 of [CLS18] is about bounding this amortized cost.

quality, that the approximate gradient occasionally points in a completely wrong direction, effectively increasing the potential instead of decreasing it. By adapting the potential function, we are able to prove that the approximate gradient only points in the wrong direction when the potential is small. Whenever the potential is large, the approximate gradient points in the correct direction. (A formal proof of this will be in Section 5.3, Lemma 5.14.) This adaption to the short step central path method allows us to handle these faulty approximate gradients. Before we can outline why this is true, we must first explain why we obtain these faulty approximate gradients in the first place.

Faulty gradients The central path method tries to maintain some vector μ close to a scalar t, where the relative distance is measured via some potential function $\Phi(\mu/t-1)$. The central path method tries to minimize this potential function by solving some linear system that depends on the gradient $\nabla\Phi(\mu/t-1)$.

In Section 5.1 we show that when solving this linear system via Theorem 2.4, then we are essentially solving the system for the approximation $\nabla\Phi(\widetilde{\mu}/t-1)$, where $\widetilde{\mu}$ is an approximation of μ with $(1-\varepsilon_{mp})\widetilde{\mu} \leq \mu \leq (1+\varepsilon_{mp})\widetilde{\mu}$ for some accuracy parameter $\varepsilon_{mp} > 0$. This is problematic because $\nabla\Phi(\mu/t-1)$ and $\nabla\Phi(\widetilde{\mu}/t-1)$ could point in opposite directions. For example for any i with $\mu_i > t$ we might have $\widetilde{\mu}_i < t$, so since Φ tries to keep μ close to t, the approximate gradient can not reliably tell if μ_i should be increased or decreased. However, if $\mu_i > (1+\varepsilon_{mp})t$, then $\widetilde{\mu}_i > t$, so the approximate gradient will correctly try to decrease μ_i. On one hand this shows, that we can not use the classical short step central path method, where one tries to maintain μ such that $\|\mu/t-1\|_2^2 = O(1)$. This is because $\|\mu/t-1\|_2^2$ could be as large as $\Omega(n\varepsilon_{mp})$. On the other hand, we are able to prove in Section 5.3 that $\|\mu/t-1\|_\infty = O(1)$, because once some entry μ_i is further from t than some $(1\pm\varepsilon_{mp})$-factor, then the approximate gradient will correctly try to move μ_i closer to t. Hence, we adapt the short step central path method by guaranteeing μ close to t in ℓ_∞-norm, instead of ℓ_2-norm.

Adapting the central path method Luckily, maintaining μ close to t in ℓ_∞-norm was previously done in [CLS18], so we can simply adapt their proof for our algorithm. The high-level idea is to use $\Phi(x) = \sum_{i=1}^n (e^{\lambda x_i} + e^{-\lambda x_i})/2$ for some parameter $\lambda = \Theta(\log n)$ as the potential function. This potential is useful because $\|x\|_\infty \leq \lambda^{-1}\log 2\Phi(x)$ (proven in Lemma 5.10). This means bounding $\Phi(\mu/t-1)$ by some polynomial in n is enough to prove $\|\mu/t-1\|_\infty = O(1)$, which will be done in Section 5.3.

The majority of the proof that this choice for Φ works, is adapted from [CLS18]. For their *stochastic central path method*, Cohen et al. sparsify the gradient $\nabla\Phi(\mu/t-1)$ via randomly sampling its entries. This sparsification could be interpreted as some type of approximation of the gradient, which allows us to adapt their proof to our new notion of "approximating" the gradient via $\nabla\Phi(\widetilde{\mu}/t-1)$ for $\widetilde{\mu} \approx \mu$. The main difference is that in [CLS18], the exact and approximate gradient always point in the same direction (i.e. their inner product is positive), so in [CLS18] it was a bit easier to show that the potential $\Phi(\mu/t-1)$ decreases in each iteration. For comparison, when using our approximation, the inner product of exact gradient $\nabla\Phi(\mu/t-1)$ and the approximation $\nabla\Phi(\widetilde{\mu}/t-1)$ may become negative. So we must spend some extra effort in Section 5.3 to show that the approximate gradient points in the correct direction, whenever $\Phi(\mu/t-1)$ is large (this will be proven in Lemma 5.14). Intuitively, this is true because when $\Phi(\mu/t-1)$ is large, then there are many indices i such that μ_i is further from t than some $(1\pm\varepsilon_{mp})$-factor. As outlined before, this means the approximate gradient tries to change the ith coordinate of μ in the correct direction, i.e. ith entry of the exact and approximate gradient have the same sign.

We also want to point out, that our approach of using a gradient w.r.t the approximate $\widetilde{\mu} \approx \mu$ means it is enough to maintain an approximate $\widetilde{x} \approx x$, $\widetilde{s} \approx s$, so $\widetilde{x}\widetilde{s} =: \widetilde{\mu} \approx \mu$. The same observation was made independently in [LSZ19], where that property was exploited to compute the steps δ_x, δ_s approximately via random sketching. The analysis of their central path method is based on modifying the standard newton steps to be a variant of gradient descent in some hessian norm space. In comparison our proof is arguably simpler, as we perform a typical gradient descent w.r.t $\Phi(\widetilde{\mu}/t)$.

3 Preliminaries

For the linear program $\min_{Ax=b,x\geq 0} c^\top x$ we assume there are no redundant constraints, i.e. the matrix A is of rank d and $n \geq d$.

Arithmetic Notation For two n dimensional vectors v, w their inner products is written as $v^\top w$ or alternatively $\langle v, w\rangle$. We write vw for the entry-wise product, so $(vw)_i := v_i w_i$. The same is true for all other arithmetic operations, for example $(v/w)_i := v_i/w_i$ and $(\sqrt{v})_i := \sqrt{v_i}$. For a scalar s the product sv is the typical entry-wise product and analogously we define $v - s$ as the entry-wise difference, so $(v-s)_i := v_i - s$.

Inequalities We write $v \leq w$ if $v_i \leq w_i$ for all $i = 1, ..., n$ and we use the notation $v \approx_\varepsilon w$ to express a $(1 \pm \varepsilon)$ approximation, defined as $(1-\varepsilon)w \leq v \leq (1+\varepsilon)w$. Note that $v \approx_\varepsilon w$ is not symmetric, but it implies $w \approx_{2\varepsilon} v$ for $\varepsilon \leq 1/2$.

Relative error and multiplicative change We will often bound the relative difference of two vectors in ℓ_2-norm: $\|(v-w)/w\|_2 = (\sum_{i=1}^n ((v_i - w_i)/w_i)^2)^{0.5}$. Sometimes we will also write this as $\|v/w - 1\|_2$. If we have $v^{\text{new}} = v + \delta_v$, then the relative difference $\|v^{\text{new}}/v - 1\|_2$ can also be written as $\|v^{-1}\delta_v\|_2$. In this context the relative difference will also be called multiplicative change, because $v^{\text{new}} = v \cdot (1 + v^{-1}\delta_v)$.

The multiplicative change of a product of two vectors, whose multiplicative change is bounded in ℓ_2-norm, can also be bounded:

Lemma 3.1. *Let v, w, δ_v, δ_w be vectors, such that $v^{\text{new}} = v + \delta_v$, $w^{\text{new}} = w + \delta_w$ then*

$$\|\frac{v^{\text{new}}w^{\text{new}}}{vw} - 1\|_2 \leq \|v^{-1}\delta_v\|_2 + \|w^{-1}\delta_w\|_2 + \|v^{-1}\delta_v\|_2\|w^{-1}\delta_w\|_2$$

Proof.

$$\begin{aligned}\|\frac{v^{\text{new}}w^{\text{new}}}{vw} - 1\|_2 &= \|\frac{v^{\text{new}}w^{\text{new}} - vw}{vw}\|_2 = \|\frac{(v+\delta_v)(w+\delta_w) - vw}{vw}\|_2 = \|\frac{v\delta_w + w\delta_v + \delta_v\delta_w}{vw}\|_2 \\ &= \|\frac{\delta_w}{w} + \frac{\delta_v}{v} + \frac{\delta_v}{v}\frac{\delta_w}{w}\|_2 \leq \|\frac{\delta_w}{w}\|_2 + \|\frac{\delta_v}{v}\| + \|\frac{\delta_v}{v}\frac{\delta_w}{w}\|_2\end{aligned}$$

Here the last term can be bounded via $\|\frac{\delta_v}{v}\frac{\delta_w}{w}\|_2 \leq \|\frac{\delta_v}{v}\|_\infty \|\frac{\delta_w}{w}\|_2 \leq \|\frac{\delta_v}{v}\|_2 \|\frac{\delta_w}{w}\|_2$ □

Further, if v^{new} has small multiplicative change compared to v, then the same is true for $1/v^{\text{new}}$ and $1/v$.

Lemma 3.2.

$$\|\frac{(v+\delta_v)^{-1} - v^{-1}}{v^{-1}}\|_2 \leq \frac{\|v^{-1}\delta_v\|_2}{1 - \|v^{-1}\delta_v\|_2}$$

Proof.

$$\|\frac{v^{-1}-(v+\delta_v)^{-1}}{v^{-1}}\|_2 = \|1-\frac{v}{v+\delta_v}\|_2 = \|\frac{\delta_v}{v+\delta_v}\|_2 = \|v^{-1}\delta_v\frac{v}{v+\delta_v}\|_2$$
$$\leq \|v^{-1}\delta_v\frac{1}{1-\|v^{-1}\delta_v\|_\infty}\|_2 \leq \frac{\|v^{-1}\delta_v\|_2}{1-\|v^{-1}\delta_v\|_2}$$

□

Fast Matrix Multiplication We write $O(n^\omega)$ for the arithmetic complexity of multiplying two $n \times n$ matrices. Computing the inverse has the same complexity. The exponent ω is also called the matrix exponent. We call α the dual matrix exponent, which is the largest value such that multiplying a $n \times n$ matrix with an $n \times n^\alpha$ requires $O(n^{2+o(1)})$ time. The current best bounds are $\omega \approx 2.38$ [Wil12, Gal14] and $\alpha \approx 0.31$ [GU18].

4 Projection Maintenance

In this section we prove Lemma 4.1, which specifies the result obtained by Algorithm 1. Given a matrix A, diagonal matrix U, vector v and function $f : \mathbb{R} \to \mathbb{R}$ (with $f(v)_i := f(v_i)$), the data-structure given by Algorithm 1/Lemma 4.1 maintains the solution $\sqrt{U}A^\top(AUA^\top)^{-1}A\sqrt{U}f(v)$ in an approximate way, by $(1 \pm \varepsilon_{mp})$-approximating U and v. This is an extension of the algorithm from [CLS18], which maintained only the matrix $\sqrt{U}A^\top(AUA^\top)^{-1}A\sqrt{U}$ approximately. We restate the formal description of the result for convenience:

Lemma 4.1 (Previously stated as Theorem 2.4 in Section 2.2)**.** *Let $A \in \mathbb{R}^{d\times n}$ be a full rank matrix with $n \geq d$, v be an n-dimensional vector and $0 < \varepsilon_{mp} < 1/4$ be an accuracy parameter. Let $f : \mathbb{R} \to \mathbb{R}$ be some function that can be computed in $O(1)$ time, and define $f(v)$ to be the vector with $f(v)_i := f(v_i)$. Given any positive number $a \leq \alpha$ there is a deterministic data-structure with the following operations*

- INITIALIZE$(A, u, f, v, \varepsilon_{mp})$*: The data-structure preprocesses the given two n dimensional vectors u, v, the $d \times n$ matrix A the function f in $O(n^2 d^{\omega-2})$ time. The given parameter $\varepsilon_{mp} > 0$ specifies the accuracy of the approximation.*
- UPDATE(u, v)*: Given two n dimensional vectors u, v. Then the data-structure returns four vectors*

$$\widetilde{u}, \quad \widetilde{v}, \quad f(\widetilde{v}), \quad \sqrt{\widetilde{U}}A^\top(A\widetilde{U}A^\top)^{-1}A\sqrt{\widetilde{U}}f(\widetilde{v}).$$

Here $\widetilde{U}$ is the diagonal matrix $\mathrm{diag}(\widetilde{u})$ *and $\widetilde{v}$ a vector such that*

$$(1-\varepsilon_{mp})\widetilde{v}_i \leq v_i \leq (1+\varepsilon_{mp})\widetilde{v}_i$$
$$(1-\varepsilon_{mp})\widetilde{u}_i \leq u_i \leq (1+\varepsilon_{mp})\widetilde{u}_i.$$

If the update sequence $u^{(1)}, ..., u^{(T)}$ (and likewise $v^{(1)}, ..., v^{(T)}$) satisfies

$$\sum_{i=1}^{n}\left(\frac{u_i^{(k+1)}-u_i^{(k)}}{u_i^{(k)}}\right)^2 \leq C^2, \qquad |\frac{u_i^{(k+1)}-u_i^{(k)}}{u_i^{(u)}}| \leq 1/4, \tag{6}$$

for all $k = 1, ..., T$ then the total time for the first T updates is

$$O\left(T \cdot \left(C/\varepsilon_{mp}(n^{\omega-1/2}+n^{2-a/2+o(1)})\log n + n^{1+a}\right)\right)$$

This section is split into three parts: We first present the algorithm and give a high-level description in Section 4.1. The next subsection (Section 4.2) proves that the algorithm returns the correct result, and at last in Section 4.3 we bound the complexity of the algorithm.

4.1 Outline of Algorithm 1

Algorithm 1 describes a data-structure, so we have variables that persist between calls to its function UPDATE. What these variables represent might be a bit hard to deduce from just reading the pseudo-code, so we want to give a brief outline of Algorithm 1 here. This outline is not required for verifying the proofs, but it might help for understanding how the algorithm works.

The internal variables are n-dimensional vectors $\widetilde{u}, \widetilde{v}, w$ and an $n \times n$ matrix M. The relationship between them is

$$M = A^\top (A\widetilde{U}A^\top)^{-1}A \text{ and } w = M\sqrt{\widetilde{U}}f(\widetilde{v}), \tag{7}$$

where $\widetilde{U} = \operatorname{diag}(\widetilde{u})$.

These internal variables are useful because of the following reason: In each call to UPDATE, the data-structure receives two new vectors $u^{\text{new}}, v^{\text{new}}$ and for $U^{\text{new}} = \operatorname{diag}(u^{\text{new}})$ the task is to return an approximation of $\sqrt{U^{\text{new}}}A^\top (AU^{\text{new}}A^\top)^{-1}A\sqrt{U^{\text{new}}}f(v^{\text{new}})$ by $(1 \pm \varepsilon_{mp})$-approximating U^{new} and v^{new}. Thus if

$$u^{\text{new}} \approx_{\varepsilon_{mp}} \widetilde{u}, v^{\text{new}} \approx_{\varepsilon_{mp}} \widetilde{v}, \tag{8}$$

then $\sqrt{\widetilde{U}}w$ would be the desired approximate result. If this $(1+\varepsilon_{mp})$-approximation condition (8) is not satisfied, then we can define two new valid approximations

$$\widetilde{u}_i^{\text{new}} := \begin{cases} \widetilde{u}_i & \text{if } u_i^{\text{new}} \approx_{\varepsilon_{mp}} \widetilde{u}_i \\ u_i^{\text{new}} & \text{othewise} \end{cases} \quad \widetilde{v}_i^{\text{new}} := \begin{cases} \widetilde{v}_i & \text{if } v_i^{\text{new}} \approx_{\varepsilon_{mp}} \widetilde{v}_i \\ v_i^{\text{new}} & \text{otherwise} \end{cases}$$

for all $i = 1, ..., n$. If $\widetilde{u}^{\text{new}}$ and $\widetilde{u}$ (and respectively $\widetilde{v}^{\text{new}}$, $\widetilde{v}^{\text{new}}$) differ in at most k many entries, then it is known (via Sherman-Morrison-Woodbury identity Lemma 4.3) that one can quickly construct two $n \times k$ matrices R, L such that

$$A^\top (A\widetilde{U}^{\text{new}}A^\top)^{-1}A = M - RL^\top.$$

This in turn means that we can get the desired approximate result as follows:

$$\sqrt{\widetilde{U}^{\text{new}}}A^\top (A\widetilde{U}^{\text{new}}A^\top)^{-1}A\sqrt{\widetilde{U}^{\text{new}}}f(\widetilde{v}^{\text{new}}) = \sqrt{\widetilde{U}^{\text{new}}}(M - RL^\top)\sqrt{\widetilde{U}^{\text{new}}}f(\widetilde{v}^{\text{new}})$$

$$= \sqrt{\widetilde{U}^{\text{new}}}\left(\underbrace{M\sqrt{\widetilde{U}}f(\widetilde{v})}_{=w} + M \underbrace{\left(\sqrt{\widetilde{U}^{\text{new}}}f(\widetilde{v}^{\text{new}}) - \sqrt{\widetilde{U}}f(\widetilde{v})\right)}_{\text{at most } 2k \text{ non-zero entries}} - RL^\top\sqrt{\widetilde{U}^{\text{new}}}f(\widetilde{v}^{\text{new}}) \right)$$

Here each term can be computed in at most $O(nk)$ time, because the first term is the already known vector w, the vector of the second term is sparse, and R, L are $n \times k$ matrices.

Thus, if k is small, then we can maintain the solution quickly. In [CLS18], Cohen et al. have developed a strategy with low amortized cost, that specifies when to recompute M for some new $\widetilde{u}$, such that k stays small. In their algorithm they do not maintain the matrix-vector product, so their data-structure does not have the internal variables w and $\widetilde{v}$. We extend their strategy to also recompute w for some new $\widetilde{v}$, such that the above outlined procedure has low amortized cost.

Algorithm 1 Projection Maintenance Data-Structure (difference to [CLS18] highlighted in blue)

datastructure MAINTAINPROJECTION ▷ Lemma 4.1
members
  $\widetilde{u}, \widetilde{v}, w \in \mathbb{R}^n$
  $f : \mathbb{R} \to \mathbb{R}$
  $A \in \mathbb{R}^{d \times n}$, $M \in \mathbb{R}^{n \times n}$
  $\epsilon_{mp} \in (0, 1/4)$ ▷ Accuracy parameter
  $a \leftarrow \min\{\alpha, 2/3\}$ ▷ Minimum batch size is n^a.
end members
procedure INITIALIZE($A, u, f, v, \epsilon_{mp}$)
  $u \leftarrow u$, $v \leftarrow v$, $f \leftarrow f$, $\epsilon_{mp} \leftarrow \epsilon_{mp}$
  $\widetilde{u} \leftarrow u$, $\widetilde{v} \leftarrow v$
  $M \leftarrow A^\top (AUA^\top)^{-1} A$
  $w \leftarrow M\sqrt{U} f(v)$
end procedure
procedure UPDATE($u^{\text{new}}, v^{\text{new}}$)
  ▷ The variables in this method represent the following: $u^{\text{new}}, v^{\text{new}}$ are the new exact values. $\widetilde{u}^{\text{new}}, \widetilde{v}^{\text{new}}$ are approximations $u^{\text{new}} \approx_{\varepsilon_{mp}} \widetilde{u}^{\text{new}}, v^{\text{new}} \approx_{\varepsilon_{mp}} \widetilde{v}^{\text{new}}$.
  ▷ Vector r will be the result: $r = \sqrt{\widetilde{U}^{\text{new}}} A^\top (A\widetilde{U}^{\text{new}} A^\top)^{-1} A \sqrt{\widetilde{U}^{\text{new}}} f(\widetilde{v}^{\text{new}})$
  ▷ For the member variables $\widetilde{u}, \widetilde{v}, M$ we have $w = M\sqrt{\widetilde{U}} f(\widetilde{v})$ and $M = A^\top (A\widetilde{U}A^\top)^{-1} A$. Note that $\widetilde{u}, \widetilde{v}$ are generally *not* approximate versions of $u^{\text{new}}, v^{\text{new}}$.
  $y_i \leftarrow u_i^{\text{new}} / \widetilde{u}_i - 1$, $\forall i \in [n]$
  Let $\pi : [n] \to [n]$ be a sorting permutation such that $|y_{\pi(i)}| \geq |y_{\pi(i+1)}|$
  $k \leftarrow$ the number of indices i such that $|y_i| \geq \epsilon_{mp}$.
  if $k \geq n^a$ **then**
    while $1.5 \cdot k < n$ and $|y_{\pi(1.5k)}| \geq (1 - 1/\log n)|y_{\pi(k)}|$ **do**
      $k \leftarrow \min(\lceil 1.5 \cdot k \rceil, n)$
    end while
  end if
  $\widetilde{u}^{\text{new}}_{\pi(i)} \leftarrow \begin{cases} u^{\text{new}}_{\pi(i)} & i \in \{1, 2, \cdots, k\} \\ \widetilde{u}_{\pi(i)} & i \in \{k+1, \cdots, n\} \end{cases}$ ▷ $\widetilde{u}^{\text{new}} \approx_{\varepsilon_{mp}} u^{\text{new}}$
  $\Delta \leftarrow \text{diag}(\widetilde{u}^{\text{new}} - \widetilde{u})$ ▷ $\Delta \in \mathbb{R}^{n \times n}$ and Δ has k non-zero entries.
  Let $S \leftarrow \pi([k])$ be the first k indices in the permutation.
  Let $M_S \in \mathbb{R}^{n \times k}$ be the k columns from S of M.
  Let $M_{S,S}, \Delta_{S,S} \in \mathbb{R}^{k \times k}$ be the k rows and columns from S of M and Δ.
  if $k \geq n^a$ **then** ▷ Perform a rank $k = 2^\ell$ update to M.
    $M \leftarrow M - M_S \cdot (\Delta_{S,S}^{-1} + M_{S,S})^{-1} \cdot (M_S)^\top$ ▷ Compute $M = A^\top (A\widetilde{U}^{\text{new}} A^\top)^{-1} A$ via Sherman-Morrison-Woodbury identity
    $w \leftarrow M\sqrt{\widetilde{U}^{\text{new}}} f(v^{\text{new}})$
    $\widetilde{u} \leftarrow \widetilde{u}^{\text{new}}$, $\widetilde{v} \leftarrow v^{\text{new}}$, $\widetilde{v}^{\text{new}} \leftarrow v^{\text{new}}$, $r \leftarrow \sqrt{\widetilde{U}^{\text{new}}} w$
  else ▷ This else-branch is the main difference compared to [CLS18].
    Let T be the set of indices i without $(1 - \varepsilon_{mp})\widetilde{v}_i \leq v_i^{\text{new}} \leq (1 + \varepsilon_{mp})\widetilde{v}_i$.
    if $|T| \geq n^a$ **then** ▷ We reset $\widetilde{v} = v^{\text{new}}$
      $r \leftarrow \sqrt{\widetilde{U}^{\text{new}}} M \sqrt{\widetilde{U}^{\text{new}}} f(v^{\text{new}}) - \sqrt{\widetilde{U}^{\text{new}}} M_S \cdot (\Delta_{S,S}^{-1} + M_{S,S})^{-1} \cdot (M_S)^\top \sqrt{\widetilde{U}^{\text{new}}} f(v^{\text{new}})$
      $w \leftarrow M\sqrt{\widetilde{U}} f(v^{\text{new}})$
      $\widetilde{v} \leftarrow v^{\text{new}}$, $\widetilde{v}^{\text{new}} \leftarrow v^{\text{new}}$
    else
      $\widetilde{v}_i^{\text{new}} \leftarrow \begin{cases} v_i^{\text{new}} & i \in T \\ \widetilde{v}_i & i \notin T \end{cases}$ ▷ $\widetilde{v}^{\text{new}} \approx_{\varepsilon_{mp}} v^{\text{new}}$
      $r \leftarrow \sqrt{\widetilde{U}^{\text{new}}} \left(w + M(\sqrt{\widetilde{U}^{\text{new}}} f(\widetilde{v}^{\text{new}}) - \sqrt{\widetilde{U}} f(\widetilde{v})) - M_S \cdot (\Delta_{S,S}^{-1} + M_{S,S})^{-1} \cdot (M_S)^\top \sqrt{\widetilde{U}^{\text{new}}} f(\widetilde{v}^{\text{new}}) \right)$
    end if
  end if
  ▷ At the end of the procedure, we still have $w = M\sqrt{\widetilde{U}} f(\widetilde{v})$ and $M = A^\top (A\widetilde{U}A^\top)^{-1} A$
  ▷ Return triple with $u^{\text{new}} \approx_{\varepsilon_{mp}} \widetilde{u}^{\text{new}}$, $v^{\text{new}} \approx_{\varepsilon_{mp}} \widetilde{v}^{\text{new}}$ and $r = \sqrt{\widetilde{U}^{\text{new}}} A^\top (A\widetilde{U}^{\text{new}} A)^{-1} A \sqrt{\widetilde{U}^{\text{new}}} f(\widetilde{v}^{\text{new}})$
  return $\widetilde{u}^{\text{new}}$, $\widetilde{v}^{\text{new}}$, $f(\widetilde{v}^{\text{new}})$, r
end procedure
end datastructure

4.2 Correctness

The task of this subsection is to prove the following lemma, which says that the vectors returned by Algorithm 1 are as specified in Lemma 4.1.

Lemma 4.2 (Returned vectors of Lemma 4.1)**.** *After every update to Algorithm 1 with input* $(u^{\text{new}}, v^{\text{new}})$ *the returned vectors* $\widetilde{u}^{\text{new}}, \widetilde{v}^{\text{new}}, f(\widetilde{v}^{\text{new}}), r$ *satisfy* $u^{\text{new}} \approx_{\varepsilon_{mp}} \widetilde{u}^{\text{new}}$, $v^{\text{new}} \approx_{\varepsilon_{mp}} \widetilde{v}^{\text{new}}$ *and*

$$r = \sqrt{\widetilde{U}^{\text{new}}} A^\top (A\widetilde{U}^{\text{new}} A)^{-1} A \sqrt{\widetilde{U}^{\text{new}}} f(\widetilde{v}^{\text{new}}).$$

Before we can prove this lemma, we must first prove that the internal variables of the data-structure save the correct values, i.e. we want to prove that equation (7) is correct. For this we must first state the following lemma from [CLS18], based on Sherman-Morison-Woodbury identity.

Lemma 4.3 ([CLS18], based on Sherman-Morison-Woodbury identity)**.** *If* $M = A^\top (A\widetilde{U}A^\top)^{-1}A$ *at the start of the update of Algorithm 1 and* $M_S, M_{S,S}, \Delta_{S,S}$ *are chosen as described in Algorithm 1, then we have*

$$M - M_S \cdot (\Delta_{S,S}^{-1} + M_{S,S})^{-1} \cdot (M_S)^\top = A^\top (A\widetilde{U}^{\text{new}} A^\top)^{-1} A$$

We can now prove that the internal variables store the correct values.

Lemma 4.4. *At the start of every update to Algorithm 1 we have*

$$w = M\sqrt{\widetilde{U}} f(\widetilde{v}) \qquad \text{and} \qquad M = A^\top (A\widetilde{U}A)^{-1}A. \tag{9}$$

Proof. If it is the first update after the initialization, then the claim is true by definition of the procedure Initialize. Next, we prove that at the end of every call to Update we satisfy (9), if (9) was satisfied at the start of Update. This then implies Lemma 4.4. If $k \geq n^a$, then line 33 makes sure that $M = A^\top (A\widetilde{U}^{\text{new}} A^\top)^{-1} A$ (see Lemma 4.3). The next lines set $w \leftarrow M\sqrt{\widetilde{U}^{\text{new}}} f(v^{\text{new}})$, $\widetilde{v} \leftarrow v^{\text{new}}$ and $\widetilde{u} \leftarrow \widetilde{u}^{\text{new}}$. Thus (9) is satisfied for the case $k \geq n^a$. If $|T| \geq n^a$, then we compute $w \leftarrow M\sqrt{\widetilde{U}} f(v^{\text{new}})$ and set $\widetilde{v} \leftarrow v^{\text{new}}$. The matrices M and $\widetilde{U}$ are not modified, so (9) is satisfied. If $|T| < n^a$, then we do not change $M, \widetilde{u}, \widetilde{v}$ or r, so (9) is satisfied. □

We now prove the correctness of Algorithm 1 by proving Lemma 4.2.

Proof of Lemma 4.2. Note that we always have $u^{\text{new}} \approx_{\varepsilon_{mp}} \widetilde{u}^{\text{new}}$ by line 27.

Case $k \geq n^a$: In line 33 we have set $M = A^\top (A\widetilde{U}^{\text{new}} A^\top)^{-1} A$ (see Lemmas 4.3 and 4.4). Hence by setting $r \leftarrow \sqrt{\widetilde{U}^{\text{new}}} w = \sqrt{\widetilde{U}^{\text{new}}} M \sqrt{\widetilde{U}^{\text{new}}} f(v^{\text{new}})$, and $\widetilde{v}^{\text{new}} \leftarrow v^{\text{new}}$, all claims of Lemma 4.2 are satisfied.

Case $|T| \geq n^a$: In this case we set r to the following value:

$$\begin{aligned} r &\leftarrow \sqrt{\widetilde{U}^{\text{new}}} M \sqrt{\widetilde{U}^{\text{new}}} f(v^{\text{new}}) - \sqrt{\widetilde{U}^{\text{new}}} M_S \cdot (\Delta_{S,S}^{-1} + M_{S,S})^{-1} \cdot (M_S)^\top \sqrt{\widetilde{U}^{\text{new}}} f(v^{\text{new}}) \\ &= \sqrt{\widetilde{U}^{\text{new}}} (A^\top (A\widetilde{U}^{\text{new}} A^\top)^{-1} A) \sqrt{\widetilde{U}^{\text{new}}} f(v^{\text{new}}) \end{aligned}$$

Here the equality comes from Lemmas 4.3 and 4.4. Further, we set $\widetilde{v}^{\text{new}} \leftarrow v^{\text{new}}$, so Lemma 4.2 is correct for the case $|T| \geq n^a$.

Case $|T| < n^a$: Here $v^{\text{new}} \approx_{\varepsilon_{mp}} \widetilde{v}^{\text{new}}$ by line 43, so we are left with verifying r. First note that $w = M\sqrt{\widetilde{U}} f(\widetilde{v})$ by Lemma 4.4, so $w + M(\sqrt{\widetilde{U}^{\text{new}}} f(\widetilde{v}^{\text{new}}) - \sqrt{\widetilde{U}} f(\widetilde{v})) = M\sqrt{\widetilde{U}^{\text{new}}} f(\widetilde{v}^{\text{new}})$. Thus r is set to the following term:

$$\begin{aligned}
r &\leftarrow \sqrt{\widetilde{U}^{\text{new}}} \left(w + M(\sqrt{\widetilde{U}^{\text{new}}} f(\widetilde{v}^{\text{new}}) - \sqrt{\widetilde{U}} f(\widetilde{v})) - M_S \cdot (\Delta_{S,S}^{-1} + M_{S,S})^{-1} \cdot (M_S)^\top \sqrt{\widetilde{U}^{\text{new}}} f(\widetilde{v}^{\text{new}}) \right) \\
&= \sqrt{\widetilde{U}^{\text{new}}} \left(M\sqrt{\widetilde{U}^{\text{new}}} f(\widetilde{v}^{\text{new}}) - M_S \cdot (\Delta_{S,S}^{-1} + M_{S,S})^{-1} \cdot (M_S)^\top \sqrt{\widetilde{U}^{\text{new}}} f(\widetilde{v}^{\text{new}}) \right) \\
&= \sqrt{\widetilde{U}^{\text{new}}} \left(M - M_S \cdot (\Delta_{S,S}^{-1} + M_{S,S})^{-1} \cdot (M_S)^\top \right) \sqrt{\widetilde{U}^{\text{new}}} f(\widetilde{v}^{\text{new}}) \\
&= \sqrt{\widetilde{U}^{\text{new}}} A^\top (A\widetilde{U}^{\text{new}} A^\top)^{-1} \sqrt{\widetilde{U}^{\text{new}}} f(\widetilde{v}^{\text{new}})
\end{aligned}$$

Where the last equality comes from Lemmas 4.3 and 4.4.

□

4.3 Complexity

In this section we will bound the complexity of Algorithm 1, proving the stated complexity bound in Lemma 4.1:

Lemma 4.5 (Complexity bound of Lemma 4.1)**.** *If the updates to Algorithm 1 satisfy the condition (6) as stated in Lemma 4.1, then after T updates the total update time of Algorithm 1 is*

$$O\left(T \cdot \left(C/\varepsilon_{mp}(n^{\omega-1/2} + n^{2-a/2+o(1)}) \log n + n^{1+a}\right)\right).$$

The preprocessing requires $O(n^2 d^{\omega-2})$ time.

As our data-structure is a modification of the data-structure presented in [CLS18], we must only analyze the complexity of the modified part. To bound the complexity of the unmodified sections of our algorithm, we will here refer to [CLS18]. The complexity analysis in [CLS18] requires an entire section (about 7 pages) via analysis of some complicated potential function. In the Appendix (Lemma A.2) we present an alternative simpler proof.

Lemma 4.6 ([CLS18], alternatively Lemma A.2)**.** *The preprocessing requires $O(n^2 d^{\omega-2})$ time. After T updates the total time of all updates of Algorithm 1, when ignoring the branch for $k < n^a$ (so we assume that branch of line 36 has cost 0), is*

$$O(T \cdot C/\varepsilon_{mp}(n^{\omega-1/2} + n^{2-a/2+o(1)}) \log n).$$

Proof. When ignoring the branch of line 36, then our algorithm performs the same operations as [CLS18][Algorithm 3] and we both maintain M in the exact same way. The only difference is that we also compute the vector r in line 34, but this requires only $O(n^2)$ time and is subsumed by the complexity of line 33. Thus our time complexity (when ignoring the branch of line 36) can be bounded by the update complexity of [CLS18][Algorithm 3], which is the complexity stated in Lemma 4.6. In the same fashion we can bound the complexity of the preprocessing. The preprocessing of [CLS18][Algorithm 3] takes $O(n^2 d^{\omega-2})$ time, where their algorithm computes only the matrix M. The only difference in our algorithm is that we also compute the vector w in line 13. The required $O(n^2)$ time to compute w is subsumed by computing M. □

Proof of Lemma 4.5. In order to prove Lemma 4.5 we only need to bound the complexity of the branch for the case $k < n^a$. The time required by all other steps of Algorithm 1 is already bounded by Lemma 4.6.

In every update we must compute $(\Delta_{S,S}^{-1}+M_{S,S})^{-1}$, which takes $O(n^{a\cdot\omega})$ time via the assumption $k < n^a$. Additionally, if $|T| < n^a$, then one update requires additional $O(n^{1+a})$ operations to compute r and w, because $(f(\widetilde{v}^{\text{new}}) - f(\widetilde{v}))$ and $(\sqrt{\widetilde{U}} - \sqrt{\widetilde{U}^{\text{new}}})$ both have at most n^a non-zero entries and M_S is a $n \times n^a$ matrix.

If $T \geq n^a$, then computing r and w can take up to $O(n^2)$ operations. This can happen at most every $O(n^{a/2}\varepsilon_{mp}/C)$ updates by Lemma A.1, because $\sum_{i=1}^{n}((v_i^{\text{new}} - v_i)/v_i)^2 \leq C^2$, Hence the amortized time per update is $O(n^{2-a/2}C/\varepsilon_{mp})$.

Note that by assuming $a \leq \alpha \leq 1$ the term $O(n^{a\cdot\omega})$ is subsumed by $O(n^{1+a})$, because $\omega \leq 3-\alpha$, so $a \cdot \omega \leq a(3-\alpha) \leq a(3-a) \leq 1 + a$. □

This concludes the proof of Lemma 4.1.

5 Central Path Method

In this section we prove the main result Theorem 1.1, by showing how to use the projection maintenance algorithm of Section 4 to obtain a fast deterministic algorithm for solving linear programs.

The algorithm for Theorem 1.1 is based on the short step central path method, outlined in Section 2.1: We construct some feasible solution triple (x, y, s) with $xs =: \mu \approx 1$ and then repeatedly decrease t while maintaining x, s such that μ stays close to t. Once t is small enough, we have a good approximate solution. This is a high-level summary of Algorithm 2, which first constructs a solution, and then runs a WHILE-loop until t is small enough. The actual hard part, maintaining the solution pair x, s with $\mu \approx t$, is done in Algorithm 3. For this task, Algorithm 3 solves a linear system (similar to (1) in Section 2.1) via the data-structure of Lemma 4.1. The majority of this section is dedicated to proving that Algorithm 3 does not require too much time and does indeed maintain the solution pairs (x, s) with $\mu \approx t$. For this we must verify the following three properties:

- Algorithm 3 does solve an approximate variant of the linear system (1).
- We do not change the linear system too much between two calls to Algorithm 3. Otherwise the data-structure of Lemma 4.1 would become too slow.
- The approximate result obtained in Algorithm 3 is good enough to maintain x, s such that μ is close to t.

The proof for this is based on the *stochastic central path method* by Cohen et al. [CLS18]. In [CLS18], they randomly sampled a certain vector, while in our algorithm this vector will be approximated deterministically via the data-structure of Algorithm 1. This derandomization has the nice side-effect, that we can skip many steps of Cohen et al.'s proof. For example they had to bound the variance of random vectors, which is no longer necessary for our algorithm.

The outline of this section is as follows. We first explain in more detail how Algorithm 3 works in Section 5.1, where we also verify the first requirement, that Algorithm 3 does indeed solve the system (1) approximately. In the next Section 5.2, we check that the input parameters for the data-structure of Lemma 4.1, used by Algorithm 3, do not change too much per iterations. The last Section 5.3 verifies, that we indeed always have $\mu \approx t$. We also consolidate all results in the last subsection by proving the main result Theorem 1.1.

Algorithm 2 Iterative loop of the central path method

procedure MAIN(A, b, c, δ) ▷ Theorem 1.1
  $\varepsilon \leftarrow 1/(1500 \ln n)$ ▷ Step size. Controls how much we decrease t in each iteration.
  $\varepsilon_{mp} \leftarrow 1/(1500 \ln n)$ ▷ Accuracy parameter for Algorithms 1 and 3
  $\lambda \leftarrow 40 \ln n$ ▷ Parameter for the potential function in Algorithm 3.
  $t \leftarrow 1$ ▷ Measures the progress so far.
  Modify the linear program according to *Lemma A.3* for $\gamma = \min\{\delta, 1/\lambda\}$ and obtain an initial x and s.
  INITIALIZEAPPROXIMATESTEP($A, x, s, t, \lambda, \varepsilon_{mp}$) ▷ Initialize Algorithm 3
  while $t > \delta^2/(2n)$ **do** ▷ We stop once the precision is good
    ▷ Decrease t to t^{new} and find new $x^{\text{new}}, s^{\text{new}}$ such that $x^{\text{new}} s^{\text{new}} =: \mu^{\text{new}} \approx_{0.1} t^{\text{new}}$
    $(x^{\text{new}}, s^{\text{new}}, t^{\text{new}}) \leftarrow$ APPROXIMATESTEP(x, s, t, ε)
    $(x, s) \leftarrow (x^{\text{new}}, s^{\text{new}})$, $t \leftarrow t^{\text{new}}$
  end while
  Use *Lemma A.3* to transform x to an approximate solution of the original linear program.
end procedure

5.1 Using the Projection Maintenance Data-Structure in the Central Path Method

In this section we outline how Algorithm 3 works and we prove that it does indeed solve the linear system (1) (outlined in Section 2.1) in some approximate way. The high-level idea of Algorithm 3 is as follows: In order to maintain μ close to t, we want to measure the distance via some potential function $\Phi(\mu/t - 1)$. As we want to minimize the distance, it makes sense to change μ by some δ_μ, which points in the same direction as $-\nabla\Phi(\mu/t - 1)$. We can find out how to change x and s, in order to change μ by approximately δ_μ, by solving the linear system (1) via the data-structure of Lemma 4.1.

In reality, we choose δ_μ to be slightly different:

$$\delta_\mu := (\frac{t^{\text{new}}}{t} - 1)\mu - \frac{\varepsilon}{2} \cdot t^{\text{new}} \cdot \frac{\nabla\Phi(\mu/t - 1)}{\|\nabla\Phi(\mu/t - 1)\|_2},$$

where $t^{\text{new}} := (1 - \frac{\varepsilon}{3\sqrt{n}})$ is the new smaller value that we want to set t to, and ε is a parameter for how large our step size should be for decreasing t.

This choice of δ_μ is motivated by the fact, that the first term $(\frac{t^{\text{new}}}{t} - 1)\mu$ leads to some helpful cancellations in later proofs. The second term is the one pointing in the direction of $-\nabla\Phi(\mu/t - 1)$, which is motivated by decreasing $\Phi(\mu/t - 1)$.

Maintaining approximate solutions One can split δ_μ into the two terms $\delta_t = (\frac{t^{\text{new}}}{t} - 1)\mu$ and $\delta_\Phi = \frac{\varepsilon}{2} \cdot t^{\text{new}} \cdot \frac{\nabla\Phi(\mu/t-1)}{\|\nabla\Phi(\mu/t-1)\|_2}$.

Algorithm 3 approximates both vectors in a different way. Specifically, given $x, s, \mu, \delta_t, \delta_\Phi, \delta_\mu$, Algorithm 3 internally maintains approximations $\widetilde{x}, \widetilde{s}, \widetilde{\mu}, \widetilde{\delta}_t, \widetilde{\delta}_\Phi, \widetilde{\delta}_\mu$ with the following properties (here $\varepsilon_{mp} > 0$ is the accuracy parameter for Lemma 4.1)

$$\begin{aligned} x &\approx_{\varepsilon_{mp}} \widetilde{x}, & s &\approx_{2\varepsilon_{mp}} \widetilde{s} \\ xs = \mu &\approx_{\varepsilon_{mp}} \widetilde{\mu} = \widetilde{x}\widetilde{s}, & \widetilde{\delta}_\Phi &= \frac{\varepsilon}{2} \cdot t^{\text{new}} \cdot \frac{\nabla\Phi(\widetilde{\mu}/t - 1)}{\|\nabla\Phi(\widetilde{\mu}/t - 1)\|_2}, \\ \delta_t &\approx_{\varepsilon_{mp}} \widetilde{\delta}_t, & \widetilde{\delta}_\mu &= \widetilde{\delta}_t + \widetilde{\delta}_\Phi \end{aligned} \tag{10}$$

Algorithm 3 ApproximateStep For the given solution pair x, s move $xs \approx t$ closer to t^{new}

global variables
  $\text{mp}_{\sqrt{\mu}}$, $\text{mp}_{\nabla\Phi}$
end global variables

procedure InitializeApproximateStep$(A, x, s, t, \lambda, \varepsilon_{mp})$
  $u \leftarrow \frac{x}{s}, \quad \mu \leftarrow xs$
  ▷ Maintains approximation of $\sqrt{U}A^\top(AUA^\top)^{-1}A\sqrt{U}\sqrt{\mu}$ via Algorithm 1.
  $\text{mp}_{\sqrt{\mu}}$.Initialize$(A, u, x \mapsto \sqrt{x}, \mu, \varepsilon_{mp},)$
  ▷ Maintains approximation of $\sqrt{U}A^\top(AUA^\top)^{-1}A\sqrt{U}\frac{\nabla\Phi_\lambda(\mu/t-1)}{\sqrt{\mu/t}}$ via Algorithm 1.
  $\text{mp}_{\nabla\Phi}$.Initialize$(A, u, x \mapsto \lambda\sinh(\lambda(x-1))/\sqrt{x}, \mu/t, \varepsilon_{mp})$
end procedure

procedure ApproximateStep(x, s, t, ε)
  ▷ One step of the modified short step central path method
  $t^{\text{new}} \leftarrow (1 - \frac{\varepsilon}{3\sqrt{n}})t, \quad \mu \leftarrow xs$
  $(\widetilde{u}, \cdot, v, p_v) \leftarrow \text{mp}_{\sqrt{\mu}}$.Update$(u, \mu), \quad (\widetilde{u}, m, w, p_w) \leftarrow \text{mp}_{\nabla\Phi}$.Update$(u, \mu/t)$
  ▷ Note that both instances always receive the same u, so they also return the same $\widetilde{u}$.
  $\widetilde{\mu} \leftarrow mt$
  $\widetilde{x} \leftarrow x\sqrt{\frac{\widetilde{\mu}}{\mu}\frac{\widetilde{u}}{u}}, \quad \widetilde{s} \leftarrow s\sqrt{\frac{\widetilde{\mu}}{\mu}\frac{u}{\widetilde{u}}}$ ▷ Thus $\widetilde{x}\widetilde{s} = \widetilde{\mu}$ and $\widetilde{x}/\widetilde{s} = \widetilde{u}$
  $\widetilde{\delta}_t \leftarrow (\frac{t^{\text{new}}}{t} - 1)v\sqrt{\widetilde{\mu}}, \quad \widetilde{\delta}_\Phi \leftarrow -\frac{\varepsilon}{2} \cdot t^{\text{new}} \cdot \frac{\sqrt{\widetilde{\mu}/t}\, w}{\|\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2}, \quad \widetilde{\delta}_\mu \leftarrow \widetilde{\delta}_t + \widetilde{\delta}_\Phi$
  $p \leftarrow (\frac{t^{\text{new}}}{t} - 1)p_v - \frac{\varepsilon}{2} \cdot t^{\text{new}} \cdot \frac{p_w}{\sqrt{t}\|\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2}$
  $\widetilde{\delta}_s \leftarrow \frac{\widetilde{s}}{\sqrt{\widetilde{\mu}}}p, \quad \widetilde{\delta}_x \leftarrow \frac{1}{\widetilde{s}}\widetilde{\delta}_\mu - \frac{\widetilde{x}}{\sqrt{\widetilde{\mu}}}p$
  return $(x + \widetilde{\delta}_x, s + \widetilde{\delta}_s, t^{\text{new}})$
end procedure

and for these approximate values, we solve the following system (which is the same as (1), but using the approximate values):

$$\begin{aligned} \widetilde{X}\widetilde{\delta}_s + \widetilde{S}\widetilde{\delta}_x &= \widetilde{\delta}_\mu, \\ A\widetilde{\delta}_x &= 0, \\ A^\top\widetilde{\delta}_y + \widetilde{\delta}_s &= 0. \end{aligned} \tag{11}$$

We prove in two steps that Algorithm 3 does indeed solve (11) for approximate values as in (10): First we prove in Lemma 5.2 that the approximations are as stated in (10), then we show in Lemma 5.3 that we indeed solve the linear system (11).

Note that $\widetilde{\delta}_\Phi$ is not an approximation of δ_Φ in the classical sense (likewise $\widetilde{\delta}_\mu$ and δ_μ) and the vectors could point in completely different directions. They are only "approximate" in the sense that their definition is the same, but for $\widetilde{\delta}_\Phi$ we replace μ by the approximate $\widetilde{\mu}$.

As outlined in the overview Section 2.3, this results in our algorithm not always decreasing the difference between μ and t. We prove in Section 5.3 that this is not a problem, if we use the following potential function Φ, accuracy parameter ε_{mp} (for Lemma 4.1) and step size ε.

Definition 5.1.

$$\Phi_\lambda(x) := \sum_{i=1}^{n} \cosh(\lambda x_i),$$

where $\cosh(x) := (e^x + e^{-x})/2$, $\lambda = 40 \ln n$.
For the step size ε *and the accuracy parameter* ε_{mp} *for Lemma 4.1, assume* $0 < \varepsilon_{mp} \leq \varepsilon \leq 1/(1500 \ln n)$.

Lemma 5.2. *The computed vectors* $\widetilde{x}, \widetilde{s}, \widetilde{\mu}, \widetilde{\delta}_t, \widetilde{\delta}_\Phi, \widetilde{\delta}_\mu$ *in Algorithm 3 satisfy the following properties: Let* $\widetilde{\mu}/t$ *be the approximation of* μ/t *maintained internally by* mp_Φ*, then* $\mu \approx_{\varepsilon_{mp}} \widetilde{\mu}$ *and* $\widetilde{\delta}_\Phi = -\frac{\varepsilon}{2} \cdot t^{\text{new}} \cdot \frac{\nabla\Phi_\lambda(\widetilde{\mu}/t-1)}{\|\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|}$. *Further* $\delta_t \approx_{\varepsilon_{mp}} \widetilde{\delta}_t$, $x \approx_{\varepsilon_{mp}} \widetilde{x}$, $s \approx_{2\varepsilon_{mp}} \widetilde{s}$.

Proof. The returned vector m in line 16 is an approximation in the sense that $\mu/t \approx_{\varepsilon_{mp}} m$, which means $\mu \approx_{\varepsilon_{mp}} mt =: \widetilde{\mu}$. We have $\frac{x}{s} =: u \approx_{\varepsilon_{mp}} \widetilde{u}$, hence $\frac{u}{\widetilde{u}} \approx_{\varepsilon_{mp}} 1$ and $1 \approx_{\varepsilon_{mp}} \frac{\widetilde{u}}{u}$. Thus $x \approx_{\varepsilon_{mp}} x\sqrt{\frac{\widetilde{\mu}}{\mu}\frac{\widetilde{u}}{u}} = \widetilde{x}$ and $s \approx_{2\varepsilon_{mp}} s\sqrt{\frac{\widetilde{\mu}}{\mu}\frac{u}{\widetilde{u}}} = \widetilde{s}$.

As potential function we have chosen $\Phi_\lambda(x) = \sum_{i=1}^n \cosh(x_i)$, so $(\nabla\Phi_\lambda(x-1)/\sqrt{x})_i = \lambda \sinh(\lambda(x_i - 1))/\sqrt{x_i}$. This means $\lambda \sinh(\lambda(x-1))/\sqrt{x}$ for $x = \mu/t$ is $\nabla\Phi_\lambda(\mu/t-1)/\sqrt{\mu/t}$ and $w = \lambda \sinh(\lambda(m-1))/\sqrt{m} = \nabla\Phi_\lambda(\widetilde{\mu}/t - 1)/\sqrt{\widetilde{\mu}/t}$. Hence we have that $\widetilde{\delta}_\Phi = -\frac{\varepsilon}{2} \cdot t^{\text{new}} \cdot \frac{\sqrt{\widetilde{\mu}/t}w}{\|\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2} = -\frac{\varepsilon}{2} \cdot t^{\text{new}} \cdot \frac{\nabla\Phi_\lambda(\widetilde{\mu}/t-1)}{\|\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2}$. We also have $\delta_t = (\frac{t^{\text{new}}}{t} - 1)\mu$, $\mu \approx_{\varepsilon_{mp}} v^2$ and $\mu \approx_{\varepsilon_{mp}} \widetilde{\mu}$, so $\mu \approx_{\varepsilon_{mp}} v\sqrt{\widetilde{\mu}}$ which implies $\delta_t \approx_{\varepsilon_{mp}} (\frac{t^{\text{new}}}{t} - 1)v\sqrt{\widetilde{\mu}} =: \widetilde{\delta}_t$.

□

Lemma 5.3. *The computed vectors in Algorithm 3 satisfy the following linear system:*

$$\widetilde{X}\widetilde{\delta}_s + \widetilde{S}\widetilde{\delta}_x = \widetilde{\delta}_\mu,$$
$$A\widetilde{\delta}_x = 0$$

Proof. We define the following projection matrix:

$$P := \sqrt{\widetilde{U}}A^\top(A\widetilde{U}A^\top)^{-1}A\sqrt{\widetilde{U}} = \sqrt{\widetilde{X}/\widetilde{S}}A^\top(A(\widetilde{X}/\widetilde{S})A^\top)^{-1}A\sqrt{\widetilde{X}/\widetilde{S}}$$

Then we have

$$p_v(\frac{t^{\text{new}}}{t} - 1) = Pv(\frac{t^{\text{new}}}{t} - 1) = P\frac{1}{\sqrt{\widetilde{x}\widetilde{s}}}\sqrt{\widetilde{\mu}}\, v(\frac{t^{\text{new}}}{t} - 1) = P\frac{1}{\sqrt{\widetilde{x}\widetilde{s}}}\widetilde{\delta}_t$$

and

$$-\frac{\varepsilon}{2} \cdot \frac{t^{\text{new}} p_w}{\sqrt{t}\|\nabla\Phi_\lambda(\widetilde{\mu}/t - 1)\|_2} = -\frac{\varepsilon}{2} \cdot \frac{t^{\text{new}} Pw}{\sqrt{t}\|\nabla\Phi_\lambda(\widetilde{\mu}/t - 1)\|_2} = -\frac{\varepsilon}{2} \cdot \frac{t^{\text{new}} P\frac{1}{\sqrt{\widetilde{x}\widetilde{s}}}\sqrt{\widetilde{\mu}}\, w}{\sqrt{t}\|\nabla\Phi_\lambda(\widetilde{\mu}/t - 1)\|_2} = P\frac{1}{\sqrt{\widetilde{x}\widetilde{s}}}\widetilde{\delta}_\Phi.$$

Hence the change to x and s is given by

$$\widetilde{\delta}_s = \frac{\widetilde{s}}{\sqrt{\widetilde{x}\widetilde{s}}}p = \frac{\widetilde{s}}{\sqrt{\widetilde{x}\widetilde{s}}}P\frac{1}{\sqrt{\widetilde{x}\widetilde{s}}}(\widetilde{\delta}_t + \widetilde{\delta}_\Phi) = \frac{\widetilde{s}}{\sqrt{\widetilde{x}\widetilde{s}}}P\frac{1}{\sqrt{\widetilde{x}\widetilde{s}}}\widetilde{\delta}_\mu$$

$$\widetilde{\delta}_x = \frac{1}{\widetilde{s}}\widetilde{\delta}_\mu - \frac{\widetilde{x}}{\sqrt{\widetilde{x}\widetilde{s}}}P\frac{1}{\sqrt{\widetilde{x}\widetilde{s}}}p = \frac{\widetilde{x}}{\sqrt{\widetilde{x}\widetilde{s}}}(\mathbb{I} - P)\frac{1}{\sqrt{\widetilde{x}\widetilde{s}}}\widetilde{\delta}_\mu$$

Lemma 5.3 is thus given by Lemma 2.1. □

5.2 Bounding the change per iteration

Algorithm 3 uses the data-structure of Lemma 4.1. The complexity of this data-structure depends on how much the input parameters (in our case $u := x/s$, μ and μ/t) change per iteration. In this section we prove:

Lemma 5.4. *Assume $\mu \approx_{0.1} t$. Let $\mu^{\text{new}} := (x + \widetilde{\delta}_x)(s + \widetilde{\delta}_s)$, the value of μ in the upcoming iteration, and let $u := \frac{x}{s}$, $u^{\text{new}} := \frac{x+\widetilde{\delta}_x}{s+\widetilde{\delta}_s}$, then*

$$\|\mu^{-1}(\mu^{\text{new}} - \mu)\| \leq 2.5\varepsilon, \quad \|(\mu/t)^{-1}(\mu^{\text{new}}/t^{\text{new}} - \mu/t)\| \leq 3\varepsilon, \quad \|(u^{\text{new}} - u)/u\|_2 \leq 3\varepsilon.$$

In order to prove this lemma, we must assume that μ is currently a good approximation of t. We assume the following proposition, which is proven in the next subsection.

Proposition 5.5. *For the input to Algorithm 3 we have $\mu \approx_{0.1} t$*

How much we change x, s depends on how long the vector δ_μ is, so we start by bounding that length.

Lemma 5.6. $\|\delta_t\|_2 \leq 1.1\frac{\varepsilon}{3}t$, $\|\delta_\Phi\|_2 \leq \frac{\varepsilon}{2}t$, $\|\delta_\mu\|_2 \leq \varepsilon t$
$\|\widetilde{\delta}_t\|_2 \leq 1.2\frac{\varepsilon}{3}t$, $\|\widetilde{\delta}_\Phi\|_2 \leq \frac{\varepsilon}{2}t$, $\|\widetilde{\delta}_\mu\|_2 \leq \varepsilon t$

Proof.

$$\|\delta_t\|_2 = \|(t^{\text{new}}/t - 1)\mu\|_2 \leq 1.1\sqrt{n}(1 - t^{\text{new}}/t)t = 1.1\sqrt{n}\frac{\varepsilon}{3\sqrt{n}}t = 1.1\frac{\varepsilon}{3}t$$

Here the first inequality comes from $\mu \approx_{0.1} t$. This then also implies $\|\widetilde{\delta}_t\|_2 \leq 1.2\frac{\varepsilon}{3}t$, because $\delta_t \approx_{\varepsilon_{mp}} \widetilde{\delta}_t$ from Lemma 5.2. Next we handle the length of δ_Φ:

$$\|\delta_\Phi\|_2 = \left\|\frac{\varepsilon}{2}t^{\text{new}}\frac{\nabla\Phi_\lambda(\mu/t - 1)}{\|\nabla\Phi_\lambda(\mu/t - 1)\|_2}\right\|_2 = \frac{\varepsilon t^{\text{new}}}{2} = \frac{\varepsilon(1 - \frac{\varepsilon}{3\sqrt{n}})t}{2} \leq \frac{\varepsilon}{2}t$$

The same proof also yields the bound for $\widetilde{\delta}_\Phi$ as we just replace μ/t by $\widetilde{\mu}/t$, but because of the normalization this does not change the length. By combining the past results via triangle inequality we obtain

$$\|\delta_\mu\| \leq \|\delta_t\|_2 + \|\delta_\Phi\|_2 \leq 1.1\frac{\varepsilon}{3}t + \frac{\varepsilon}{2}t \leq \varepsilon t$$

and likewise $\|\widetilde{\delta}_\mu\| \leq \varepsilon t$. □

Next we show that the multiplicative change to x and s is small.

Lemma 5.7. $\|\widetilde{s}^{-1}\widetilde{\delta}_s\|_2 \leq 1.2\varepsilon$, $\|s^{-1}\widetilde{\delta}_s\|_2 \leq 1.2\varepsilon$,
$\|\widetilde{x}^{-1}\widetilde{\delta}_x\|_2 \leq 1.2\varepsilon$, $\|x^{-1}\widetilde{\delta}_x\|_2 \leq 1.2\varepsilon$

Proof. Since $\widetilde{P}$ is an orthogonal projection matrix we have $\|\widetilde{P}\frac{\widetilde{\delta}_\mu}{\sqrt{\widetilde{X}\widetilde{S}}}\|_2 \leq \|\frac{\widetilde{\delta}_\mu}{\sqrt{\widetilde{X}\widetilde{S}}}\|_2$ and as $\mu \approx_{\varepsilon_{mp}} \widetilde{\mu} = \widetilde{x}\widetilde{s}$ and $\mu \approx_{0.1} t$, this can be further bounded by $\sqrt{(1+\varepsilon_{mp})/(0.9t)}\|\widetilde{\delta}_\mu\|$. This allows us to bound $\|\widetilde{s}^{-1}\widetilde{\delta}_s\|_2$ as follows:

$$\begin{aligned}\|\widetilde{s}^{-1}\widetilde{\delta}_s\|_2 = \|\frac{1}{\sqrt{\widetilde{X}\widetilde{S}}}\widetilde{P}\frac{\widetilde{\delta}_\mu}{\sqrt{\widetilde{X}\widetilde{S}}}\|_2 &\leq \sqrt{(1+\varepsilon_{mp})/(0.9t)}\|\widetilde{P}\frac{\widetilde{\delta}_\mu}{\sqrt{\widetilde{X}\widetilde{S}}}\|_2 \\ &\leq (1+\varepsilon_{mp})/(0.9t)\|\widetilde{\delta}_\mu\|_2 \leq (1+\varepsilon_{mp})/0.9\varepsilon \leq 1.2\varepsilon,\end{aligned}$$

where we used $\|\widetilde{\delta}_\mu\| \le \varepsilon t$ from Lemma 5.6. The proof for $\|\widetilde{x}^{-1}\widetilde{\delta}_x\|_2 \le 1.2\varepsilon$ is identical as $\mathbb{I} - \widetilde{P}$ is also a projection matrix.

As $x \approx_{\varepsilon_{mp}} \widetilde{x}$, $s \approx_{2\varepsilon_{mp}} \widetilde{s}$ we have $\|s^{-1}\widetilde{\delta}_s\|_2 \le (1-\varepsilon_{mp})^{-1}\|\widetilde{s}^{-1}\widetilde{\delta}_s\|_2 \le 1.2\varepsilon$, $\|x^{-1}\widetilde{\delta}_x\|_2 \le (1-\varepsilon_{mp})^{-1}\|\widetilde{x}^{-1}\widetilde{\delta}_x\|_2 \le 1.2\varepsilon$ via the same proof. □

With this we can now prove Lemma 5.4. We split the proof into two separate corollaries: one for μ and one for u.

Corollary 5.8. $\|\mu^{-1}(\mu^{\text{new}} - \mu)\| \le 2.5\varepsilon$, $\|(\mu/t)^{-1}(\mu^{\text{new}}/t^{\text{new}} - \mu/t)\| \le 3\varepsilon$

Proof. The first claim follows from $\mu = xs$, $\mu^{\text{new}} = (x+\widetilde{\delta}_x)(s+\widetilde{\delta}_s)$ and $\|x^{-1}\widetilde{\delta}_x\|, \|s^{-1}\widetilde{\delta}_s\| \le 1.2\varepsilon$, and applying Lemma 3.1:

$$\|\mu^{-1}(\mu^{\text{new}} - \mu)\| \le \|x^{-1}\widetilde{\delta}_x\|_2 + \|s^{-1}\widetilde{\delta}_s\|_2 + \|x^{-1}\widetilde{\delta}_x\|_2\|s^{-1}\widetilde{\delta}_s\|_2 \le 1.2\varepsilon + 1.2\varepsilon + (1.2\varepsilon)^2 \le 2.5\varepsilon$$

The second claim is implied by Lemma 3.1 and Lemma 3.2: Lemma 3.2 allows us to describe how much $(t^{\text{new}})^{-1} \cdot \mathbf{1}_n$ changed compared to $t^{-1} \cdot \mathbf{1}_n$:

$$\|t\cdot\mathbf{1}_n((\frac{1}{t^{\text{new}}} - \frac{1}{t})\cdot\mathbf{1}_n)\|_2 \le \frac{\|t^{-1}\cdot\mathbf{1}_n((t^{\text{new}}-t)\cdot\mathbf{1}_n)\|_2}{1-\|t^{-1}\cdot\mathbf{1}_n((t^{\text{new}}-t)\cdot\mathbf{1}_n)\|_2} \le \frac{\sqrt{n}|(t^{\text{new}}-t)/t|}{1-\sqrt{n}|(t^{\text{new}}-t)/t|} = \frac{\sqrt{n}\frac{\varepsilon}{3\sqrt{n}}}{1-\frac{\varepsilon}{3\sqrt{n}}} \le 0.35\varepsilon$$

Then Lemma 3.2 tells us $\|(\mu/t)^{-1}(\mu^{\text{new}}/t^{\text{new}} - \mu/t)\| \le 0.35\varepsilon + 2.5\varepsilon + (0.35\cdot 2.5)\varepsilon^2 \le 3\varepsilon$. □

Likewise, the multiplicative change of $u := \frac{x}{s}$ can be bounded as follows:

Corollary 5.9. *Let $u := \frac{x}{s}$, then $\|(u^{\text{new}} - u)/u\|_2 \le 3\varepsilon$*

Proof. We have $\|x^{-1}\delta_x\|_2, \|s^{-1}\delta_s\| \le 1.2\varepsilon$, see Lemma 5.7. Thus $\|s((s+\delta_s)^{-1} - s^{-1})\| \le 1.2\varepsilon/(1-1.2\varepsilon) \le 1.4\varepsilon$ by Lemma 3.2. This leads to $\|u^{-1}(u^{\text{new}} - u)\| \le 1.4\varepsilon + 1.2\varepsilon + (1.2\varepsilon)^2 < 3\varepsilon$, because of $u = x/s$ and Lemma 3.1. □

5.3 Maintaining $\mu \approx t$

In this section we prove Proposition 5.5, so $\mu \approx_{0.1} t$. An alternative way to write this statement is $\|\mu/t - 1\|_\infty \le 0.1$. We prove that this norm is small, by showing that the potential $\Phi_\lambda(\mu/t - 1)$ stays below a certain threshold. The choice of $\Phi_\lambda(x) = \sum_{i=1}^n \cosh(x_i)$ is motivated by the following lemma:

Lemma 5.10. $\|\mu/t - 1\|_\infty \le \frac{\ln 2\Phi_\lambda(\mu/t-1)}{\lambda}$

Proof. $\Phi_\lambda(x) = \frac{1}{2}\sum_{i=1}^n e^{\lambda x_i} + e^{-\lambda x_i} \ge \frac{1}{2}e^{\lambda\|x\|_\infty}$, so $\|x\|_\infty \le \frac{\ln 2\Phi_\lambda(x)}{\lambda}$. □

This means we must prove $\Phi_\lambda(\mu/t - 1) \le 0.5\cdot e^{0.1\lambda} = 0.5n^4$. We prove this in an inductive way. More accurately, in this section we prove the following lemma. (Note that $2n \le 0.5n^4$ for $n > 1$.)

Lemma 5.11. *If $\Phi_\lambda(\mu/t - 1) \le 2n$, then $\Phi_\lambda(\mu^{\text{new}}/t^{\text{new}} - 1) \le 2n$.*

In order to show that Lemma 5.11 is true, we must first bound the impact of all the approximations. We start by bounding the error that we incur based on the approximation $\mu^{\text{new}} \approx \mu + \widetilde{\delta}_\mu$, when in reality we have $\mu^{\text{new}} = (x+\widetilde{\delta}_x)(s+\widetilde{\delta}_s) = \mu + \widetilde{\delta}_\mu + \widetilde{\delta}_x\widetilde{\delta}_s$.

Lemma 5.12. *For $\mu^{\text{new}} = (x+\widetilde{\delta}_x)(s+\widetilde{\delta}_s)$ we have $\|\mu^{\text{new}} - \mu - \widetilde{\delta}_\mu\|_2 \le 6t\varepsilon^2$.*

Proof. We can expand the term for μ^{new} as follows:

$$\mu^{\text{new}} = (x+\widetilde{\delta}_x)(s+\widetilde{\delta}_s) = xs + x\widetilde{\delta}_s + s\widetilde{\delta}_x + \widetilde{\delta}_x\widetilde{\delta}_s = \mu + \underbrace{\widetilde{x}\widetilde{\delta}_s + \widetilde{s}\widetilde{\delta}_x}_{\widetilde{\delta}_\mu} + (x-\widetilde{x})\widetilde{\delta}_s + (s-\widetilde{s})\widetilde{\delta}_x + \widetilde{\delta}_x\widetilde{\delta}_s.$$

Hence the error (relative to μ) can be bounded as follows:

$$\begin{aligned}
\|\mu^{-1}(\mu^{\text{new}} - \mu - \widetilde{\delta}_\mu)\|_2 &= \|\mu^{-1}((x-\widetilde{x})\widetilde{\delta}_s + (s-\widetilde{s})\widetilde{\delta}_x + \widetilde{\delta}_x\widetilde{\delta}_s)\|_2 \\
&\le \|\mu^{-1}(x-\widetilde{x})s\cdot s^{-1}\widetilde{\delta}_s\|_2 + \|\mu^{-1}(s-\widetilde{s})x\cdot x^{-1}\widetilde{\delta}_x\|_2 + \|\mu^{-1}\widetilde{\delta}_x\widetilde{\delta}_s\|_2 \\
&\le \|\mu^{-1}(x-\widetilde{x})s\|_\infty\|s^{-1}\widetilde{\delta}_s\|_2 + \|\mu^{-1}(s-\widetilde{s})x\|_\infty\|x^{-1}\widetilde{\delta}_x\|_2 + \|x^{-1}\widetilde{\delta}_x s^{-1}\widetilde{\delta}_s\|_2 \\
&\le \frac{\varepsilon_{mp}}{1-\varepsilon_{mp}}\|s^{-1}\widetilde{\delta}_s\|_2 + \frac{2\varepsilon_{mp}}{1-2\varepsilon_{mp}}\|x^{-1}\widetilde{\delta}_x\|_2 + \|x^{-1}\widetilde{\delta}_x\|_2\|s^{-1}\widetilde{\delta}_s\|_2 \\
&\le 3.7\varepsilon_{mp}\varepsilon + (1.2\varepsilon)^2
\end{aligned}$$

For the fourth line we used $\mu = xs$, $x \approx_{\varepsilon_{mp}} \widetilde{x}$, $s \approx_{2\varepsilon_{mp}} \widetilde{s}$, which implies (for example) $\mu^{-1}(x-\widetilde{x})s = x^{-1}(x-\widetilde{x}) \le x^{-1}\varepsilon_{mp}\widetilde{x} \le \frac{\varepsilon_{mp}}{1-\varepsilon_{mp}}$. The last line uses Lemma 5.7.

By exploiting $\mu \approx_{0.1} t$ and $\varepsilon_{mp} \le \varepsilon$, we get $\|\mu^{\text{new}} - \mu - \widetilde{\delta}_\mu\|_2 \le 6t\varepsilon^2$.

□

Another source of error is that $\widetilde{\delta}_\Phi$ and δ_Φ (which depend on $\nabla\Phi_\lambda(\widetilde{\mu}/t-1)$ and $\nabla\Phi_\lambda(\mu/t-1)$) might point in two completely different directions. This issue was outlined in the overview Section 2.3, where we claimed that for $\Phi_\lambda(\mu/t-1)$ large enough, the approximate gradient $\nabla\Phi_\lambda(\widetilde{\mu}/t-1)$ does point in the same direction as $\nabla\Phi_\lambda(\mu/t-1)$. In order to prove this claim, we require some properties of $\Phi_\lambda(\cdot)$.

Lemma 5.13 ([CLS18])**.** *Let $\Phi_\lambda(x) = \sum_{i=1}^n \cosh(\lambda x_i)$, then*

1. *For any $\|v\|_\infty \le 1/\lambda$ we have*
$$\Phi_\lambda(r+v) \le \phi_\lambda(r) + \langle\nabla\Phi_\lambda(r), v\rangle + 2\|v\|_{\nabla^2\phi_\lambda(r)}.$$
2. $\|\nabla\phi_\lambda(r)\|_2 \ge \frac{\lambda}{\sqrt{n}}(\Phi_\lambda(r)-n)$
3. $(\sum_{i=1}^n \lambda^2\Phi_\lambda(r_i)_i^2)^{0.5} \le \lambda\sqrt{n} + \|\nabla\Phi_\lambda(r)\|_2$

With these tools we can now analyze the impact of approximating $\nabla\Phi_\lambda(\mu/t-1)$ via $\nabla\Phi_\lambda(\widetilde{\mu}/t-1)$. The following lemma says that, if the potential $\|\nabla\Phi_\lambda(\mu/t-1)\|_2$ is larger than $(2.5/0.9)\lambda^2\varepsilon_{mp}\sqrt{n}$, then the approximate gradient does point in the correct direction (i.e. the inner product with the real gradient is positive).

Lemma 5.14 (Good direction for large $\Phi_\lambda(\mu/t-1)$)**.**

$$\langle\nabla\Phi_\lambda(\mu/t-1), -\frac{\nabla\Phi_\lambda(\widetilde{\mu}/t-1)}{\|\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2}\rangle \le -0.9\|\nabla\Phi_\lambda(\mu/t-1)\|_2 + 2.5\lambda^2\varepsilon_{mp}\sqrt{n}$$

Proof.

$$\begin{aligned}
&\langle\nabla\Phi_\lambda(\mu/t-1), -\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\rangle \\
=& -\langle\nabla\Phi_\lambda(\widetilde{\mu}/t-1), \nabla\Phi_\lambda(\widetilde{\mu}/t-1)\rangle + \langle\nabla\Phi_\lambda(\mu/t-1) - \nabla\Phi_\lambda(\widetilde{\mu}/t-1), \nabla\Phi_\lambda(\widetilde{\mu}/t-1)\rangle \\
\le& -\|\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2^2 + \|\nabla\Phi_\lambda(\mu/t-1) - \nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2\cdot\|\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2
\end{aligned}$$

By normalizing the second vector we then obtain:

$$\langle \nabla\Phi_\lambda(\mu/t-1), -\frac{\nabla\Phi_\lambda(\widetilde{\mu}/t-1)}{\|\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2}\rangle \le -\|\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2 + \|\nabla\Phi_\lambda(\mu/t-1)-\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2$$

So in order to prove Lemma 5.14, we must bound the norm $\|\nabla\Phi_\lambda(\mu/t-1)-\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2$. Note that $\nabla\Phi_\lambda(x)_i = \lambda\sinh(\lambda x_i)$ and $\sinh(x) = (e^x - e^{-x})/2$. So for now let us bound $|\sinh(x+y)-\sinh(x)|$:

$$\begin{aligned}|\sinh(x+y)-\sinh(x)| &= |e^x\cdot e^y - e^{-x}\cdot e^{-y} - (e^x - e^{-x})|/2 = |e^x\cdot(e^y-1) + e^{-x}\cdot(1-e^{-y})|/2\\ &\le (e^x\cdot|e^y-1| + e^{-x}\cdot|1-e^{-y}|)/2 \le (e^x+e^{-x})/2\cdot\max(|e^y-1|,|1-e^{-y}|)\\ &\le (e^x+e^{-x})/2(e^{|y|}-1) = \cosh(x)(e^{|y|}-1)\end{aligned}$$

Thus we can bound the difference as follows

$$\begin{aligned}\|\nabla\Phi_\lambda(\mu/t-1)-\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2 &= \lambda\|\sinh(\lambda(\mu/t-1)) - \sinh(\lambda(\widetilde{\mu}/t-1))\|_2\\ &= \lambda\|\sinh(\lambda(\widetilde{\mu}/t-1+(\mu-\widetilde{\mu})/t)) - \sinh(\lambda(\widetilde{\mu}/t-1))\|_2\\ &\le \lambda\|\cosh(\lambda(\widetilde{\mu}/t-1))(e^{\lambda|\widetilde{\mu}-\mu|/t}-1)\|_2\\ &\le \lambda\|\cosh(\lambda(\widetilde{\mu}/t-1))\|_2(e^{\lambda\|\widetilde{\mu}-\mu\|_\infty/t}-1)\\ &\le (\lambda\sqrt{n}+\|\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2)(e^{\lambda\|\widetilde{\mu}-\mu\|_\infty/t}-1)\end{aligned}$$

For the last inequality we used the third statement of Lemma 5.13. Note that $\|(\widetilde{\mu}-\mu)/t\|_\infty \le \|\varepsilon_{mp}\widetilde{\mu}/t\|_\infty \le \|\frac{\varepsilon_{mp}}{1-\varepsilon_{mp}}\mu/t\|_\infty \le \frac{1.1\varepsilon_{mp}}{1-\varepsilon_{mp}}$. As $\varepsilon_{mp}\le 1/\lambda$ we can use $e^{|x|}\le 1+2|x|$ for $|x|<1.25$ to bound the extra factor $(e^{\lambda\|\widetilde{\mu}-\mu\|_\infty/t}-1) < 2.5\lambda\varepsilon_{mp}$.

Finally, this allows us to obtain

$$\begin{aligned}\langle \nabla\Phi_\lambda(\mu/t-1), -\frac{\nabla\Phi_\lambda(\widetilde{\mu}/t-1)}{\|\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2}\rangle &\le -\|\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2 + \|\nabla\Phi_\lambda(\mu/t-1)-\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2\\ &< -\|\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2 + 2.5\lambda\varepsilon_{mp}(\lambda\sqrt{n}+\|\nabla\Phi_\lambda(\widetilde{\mu}/t-1)\|_2)\\ &\le -0.9\|\nabla\Phi_\lambda(\mu/t-1)\|_2 + 2.5\lambda^2\varepsilon_{mp}\sqrt{n}\end{aligned}$$

For the last inequality we used $2.5\lambda\varepsilon_{mp} < 0.1$. □

We now have all tools available to bound $\Phi_\lambda(\frac{\mu^{\text{new}}}{t^{\text{new}}}-1)$:

Lemma 5.15.

$$\Phi_\lambda(\frac{\mu^{\text{new}}}{t^{\text{new}}}-1) \le \Phi_\lambda(\mu/t-1) - \frac{\varepsilon}{3}\frac{\lambda}{\sqrt{n}}(\Phi_\lambda(\mu/t-1)-0.5n)$$

Proof. First let us write $\frac{\mu^{\text{new}}}{t^{\text{new}}}-1$ as $\frac{\mu}{t}-1+v$ for some vector v. Then

$$\begin{aligned}v &= \frac{\mu^{\text{new}}}{t^{\text{new}}} - \frac{\mu}{t} = \frac{\mu^{\text{new}}-\mu-\delta_t-\widetilde{\delta}_\Phi}{t^{\text{new}}} + \frac{\mu+\delta_t+\widetilde{\delta}_\Phi}{t^{\text{new}}} - \frac{\mu}{t}\\ &= \frac{\mu^{\text{new}}-\mu-\delta_t-\widetilde{\delta}_\Phi}{t^{\text{new}}} + \frac{\mu+(t^{\text{new}}/t-1)\mu - \frac{\varepsilon}{2}t^{\text{new}}\frac{\nabla\Phi_\lambda(\widetilde{\mu}/t-1)}{\|\Phi_\lambda(\widetilde{\mu}/t-1)\|_2}}{t^{\text{new}}} - \frac{\mu}{t}\\ &= \frac{\mu^{\text{new}}-\mu-\delta_t-\widetilde{\delta}_\Phi}{t^{\text{new}}} + \frac{\mu}{t^{\text{new}}} + \frac{\mu}{t} - \frac{\mu}{t^{\text{new}}} - \frac{\varepsilon}{2}\frac{\nabla\Phi_\lambda(\widetilde{\mu}/t-1)}{\|\Phi_\lambda(\widetilde{\mu}/t-1)\|_2} - \frac{\mu}{t}\\ &= \frac{\mu^{\text{new}}-\mu-\delta_t-\widetilde{\delta}_\Phi}{t^{\text{new}}} - \frac{\varepsilon}{2}\frac{\nabla\Phi_\lambda(\widetilde{\mu}/t-1)}{\|\Phi_\lambda(\widetilde{\mu}/t-1)\|_2}\end{aligned}$$

In order to use Lemma 5.13, we must show that $\|v\|_2 < 1/\lambda$. For that we bound the length of $\|\frac{\mu^{\text{new}}-\mu-\delta_t-\widetilde{\delta}_\Phi}{t^{\text{new}}}\|$ as follows:

$$\begin{aligned}\|\frac{\mu^{\text{new}}-\mu-\delta_t-\widetilde{\delta}_\Phi}{t^{\text{new}}}\|_2 &= \|\frac{\mu^{\text{new}}-\mu-\widetilde{\delta}_\mu+(\widetilde{\delta}_t-\delta_t)}{t^{\text{new}}}\|_2 \\ &\leq \frac{1}{t^{\text{new}}}(\|\mu^{\text{new}}-\mu-\widetilde{\delta}_\mu\|_2+\|\widetilde{\delta}_t-\delta_t\|) \\ &\leq \frac{1}{t^{\text{new}}}(6t\varepsilon^2+\varepsilon_{mp}\|\widetilde{\delta}_t\|_2) \leq \frac{t}{t^{\text{new}}}(6+\frac{1.2}{3})\varepsilon^2 < 6.5\varepsilon^2\end{aligned}$$

In the first line we used $\widetilde{\delta}_\mu = \widetilde{\delta}_t + \widetilde{\delta}_\Phi$ and in the last line we used Lemmas 5.6 and 5.12. Thus $\|v\|_2 \leq 6.5\varepsilon^2 + \varepsilon/2 < \varepsilon \leq 1/\lambda$ and we can apply Lemma 5.13:

$$\begin{aligned}\Phi_\lambda(\mu/t+v-1) &\leq \Phi_\lambda(\mu/t-1)+\langle\nabla\Phi_\lambda(\mu/t-1),v\rangle+2\|v\|^2_{\nabla^2\Phi_\lambda(\mu/t-1)} \\ &= \Phi_\lambda(\mu/t-1)-\frac{\varepsilon}{2}\langle\Phi_\lambda(\mu/t-1),\frac{\nabla\Phi_\lambda(\widetilde{\mu}/t-1)}{\|\Phi_\lambda(\widetilde{\mu}/t-1)\|_2}\rangle \\ &\quad+\langle\nabla\Phi_\lambda(\mu/t-1),\frac{\mu^{\text{new}}-\mu-\delta_t-\widetilde{\delta}_\Phi}{t^{\text{new}}}\rangle+2\|v\|^2_{\nabla^2\Phi_\lambda(\mu/t-1)} \\ &\leq \Phi_\lambda(\mu/t-1)-\frac{0.9\varepsilon}{2}\|\nabla\Phi_\lambda(\mu/t-1)\|_2+1.25\varepsilon^2\lambda^2\sqrt{n} \\ &\quad+\|\nabla\Phi_\lambda(\mu/t-1)\|_2\cdot\|\frac{\mu^{\text{new}}-\mu-\delta_t-\widetilde{\delta}_\Phi}{t^{\text{new}}}\|_2+2\|v\|^2_{\nabla^2\Phi_\lambda(\mu/t-1)} \\ &\leq \Phi_\lambda(\mu/t-1)-\frac{0.9\varepsilon}{2}\|\nabla\Phi_\lambda(\mu/t-1)\|_2+1.25\varepsilon^2\lambda^2\sqrt{n} \\ &\quad+6.5\varepsilon^2\|\nabla\Phi_\lambda(\mu/t-1)\|_2+2\|v\|^2_{\nabla^2\Phi_\lambda(\mu/t-1)}\end{aligned}$$

In the third line we used Lemma 5.14 and Cauchy-Schwarz and the last line comes from the bound we proved above. Next we bound the second order term:

$$\begin{aligned}\|v\|^2_{\nabla^2\Phi_\lambda(\mu/t-1)} &= \lambda\sum_{i=1}^n\lambda\Phi_\lambda(\mu/t-1)_i v_i^2 \\ &\leq \lambda(\sum_{i=1}^n(\lambda\Phi_\lambda(\mu/t-1)_i)^2)^{0.5}(\sum_{i=1}^n v_i^4)^{0.5} \\ &\leq \lambda(\lambda\sqrt{n}+\|\nabla\Phi_\lambda(\mu/t-1)\|_2)\|v\|_4^2 \\ &\leq \lambda(\lambda\sqrt{n}+\|\nabla\Phi_\lambda(\mu/t-1)\|_2)(\varepsilon)^2\end{aligned}$$

The first inequality comes form Cauchy-Schwarz, the second inequality from Lemma 5.13 and the

last inequality uses $\|v\|_4 \le \|v\|_2 < \varepsilon$. Plugging all these bound together we obtain:

$$\begin{aligned}
&\Phi_\lambda(\mu/t + v - 1)\\
&\le \Phi_\lambda(\mu/t-1) - \frac{0.9\varepsilon}{2}\|\nabla\Phi_\lambda(\mu/t-1)\|_2 + 1.25\varepsilon^2\lambda^2\sqrt{n}\\
&\qquad + 6.5\varepsilon^2\|\nabla\Phi_\lambda(\mu/t-1)\|_2 + 2\lambda(\lambda\sqrt{n} + \|\nabla\Phi_\lambda(\mu/t-1)\|_2)\varepsilon^2\\
&< \Phi_\lambda(\mu/t-1) + \|\nabla\Phi_\lambda(\mu/t-1)\|_2(6.5\varepsilon^2 + 2\lambda\varepsilon^2 - \frac{0.9\varepsilon}{2}) + 3.25\varepsilon^2\lambda^2\sqrt{n}\\
&< \Phi_\lambda(\mu/t-1) - \frac{\varepsilon}{3}\|\nabla\Phi_\lambda(\mu/t-1)\|_2 + 3.25\varepsilon^2\lambda^2\sqrt{n}\\
&\le \Phi_\lambda(\mu/t-1) - \frac{\varepsilon}{3}\frac{\lambda}{\sqrt{n}}(\Phi_\lambda(\mu/t-1) - n) + 3.25\varepsilon^2\lambda^2\sqrt{n}\\
&\le \Phi_\lambda(\mu/t-1) - \frac{\varepsilon}{3}\frac{\lambda}{\sqrt{n}}(\Phi_\lambda(\mu/t-1) - 10n\varepsilon\lambda)\\
&\le \Phi_\lambda(\mu/t-1) - \frac{\varepsilon}{3}\frac{\lambda}{\sqrt{n}}(\Phi_\lambda(\mu/t-1) - 0.5n)
\end{aligned}$$

Here the first inequality uses $\|v\|^2_{\nabla^2\Phi_\lambda(\mu/t-1)} \le \lambda\varepsilon^2(\lambda\sqrt{n} + \|\nabla\Phi_\lambda(\mu/t-1)\|_2)$. The third uses $\varepsilon \le 1/(1500\ln n)$ and $\lambda = 40\ln n$, so $(6.5\varepsilon^2 + 2\lambda\varepsilon^2 - \frac{0.9\varepsilon}{2}) < (6.5/1500 + 2\cdot 40/1500 - 0.9/2)\varepsilon < -\varepsilon/3$. The fourth inequality uses part 2 of Lemma 5.13.

□

Proof of Lemma 5.11. On one hand, Lemma 5.15 implies that $\Phi_\lambda(\mu^{\text{new}}/t^{\text{new}} - 1) < \Phi_\lambda(\mu/t - 1)$, if $\Phi_\lambda(\mu/t-1) > 0.5n$. On the other hand, if $\Phi_\lambda(\mu/t-1) \le 0.5n$, then $\Phi_\lambda(\mu^{\text{new}}/t^{\text{new}} - 1)\Phi_\lambda(\mu/t-1) + \frac{\varepsilon}{3}\frac{\lambda}{\sqrt{n}}0.5n \le \Phi_\lambda(\mu/t-1) + 0.005\sqrt{n} < 2n$. Thus if $\Phi_\lambda(\mu/t-1) \le 2n$, then $\Phi_\lambda(\mu^{\text{new}}/t^{\text{new}} - 1) \le 2n$.

□

We now have all intermediate results required to prove our main result of Theorem 1.1.

Proof of Theorem 1.1. We start by proving the correctness:

Correctness of the algorithm At the start of algorithm we transform the linear program as specified in Lemma A.3 to obtain a feasible solution (x, y, s). For that transformation we choose $\gamma = \min\{\delta, 1/\lambda\}$, so $\mu - 1 = \gamma c/L$ and $\|\mu/t - 1\|_\infty \le 1/\lambda$ for $t = 1$ at the start of the algorithm. This then implies $\Phi_\lambda(\mu/t-1) \le n\cosh(\lambda/\lambda) \le n(1+e)/2 \le 2n$ which for $n > 1$ is less than $0.5n^4$, and thus $\|\mu/t - 1\|_\infty \le 0.1$ throughout the entire algorithm by Lemmas 5.10 and 5.11. (This then also proves Proposition 5.5.)

The algorithm runs until $t < \delta^2/(2n)$, then we have $\|\mu\|_1 \le n\|\mu\|_\infty \le 1.1nt \le \delta^2 \le \gamma^2$, so we obtain a solution via Lemma A.3.

Complexity of the algorithm In each iteration, t decreases by a factor of $(1 - \frac{\varepsilon}{3\sqrt{n}})$, so it takes $O(\sqrt{n}\varepsilon^{-1}\log(\delta/n))$ iterations to reach $t < \delta^2/(2n)$. We now bound the cost per iteration. The vectors $u := x/s$, $\mu := xs$, and μ/t of Algorithm 3 have small multiplicative change, bounded by 3ε, 2.5ε, and 3ε respectively (Corollaries 5.8 and 5.9). Thus the amortized cost per iteration is $O(\varepsilon/\varepsilon_{mp}(n^{\omega-1/2} + n^{2-a/2+o(1)})\log n + n^{1+a})$ via Lemma 4.1. For $\varepsilon_{mp} = \varepsilon = 1/(1500\ln n)$ and $a = \min\{\alpha, 2/3\}$ this is $O(n^{\omega-1/2}\log n)$ for current $\omega \approx 2.37$, $\alpha \approx 0.31$ [Wil12, Gal14, GU18].

The total cost is $O((n^\omega + n^{2.5-\alpha/2+o(1)} + n^{2+1/6+o(1)})\log^2(n)\log(n/\delta))$ and for current ω, α this is just $O(n^\omega\log^2(n)\log(n/\delta))$. □

6 Open Problems

The $\widetilde{O}(n^\omega)$ upper bound presented in this paper (but also the one from [CLS18]) seems optimal in the sense, that all known linear system solvers require up to $O(n^\omega)$ time for solving $Ax = b$. However, this claimed optimality has two caveats: (i) The algorithm is only optimal when assuming $d = \Omega(n)$. What improvements are possible for $d \ll n$? (ii) The $\widetilde{O}(n^\omega)$ upper bound only holds for the current bounds of ω and the dual exponent α. No matter how much ω improves, the presented linear program solver can never beat $\widetilde{O}(n^{2+1/6})$ time. So if in the future some upper bound $\omega < 2 + 1/6$ is discovered, then these linear program solvers are no longer optimal. One open question is thus, if the algorithm can be improved to run in truly $\widetilde{O}(n^\omega)$ for every bound on ω, or alternatively to prove that $\omega > 2 + 1/6$.[9]

Another interesting question is, if the techniques of this paper can also be applied to other interior point algorithms. For example, can they be used to speed-up solvers for semidefinite programming?

Acknowledgement

I thank Danupon Nanongkai and Thatchaphol Saranurak for discussions. I also thanks So-Hyeon Park (Sophie) for her questions and feedback regarding the algorithm. This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme under grant agreement No 715672. The algorithmic descriptions in this paper use latex-code of [CLS18], available under CC-BY-4.0[10]

A Appendix

Lemma A.1. *Let $(x^k)_{k\geq 1}$ be a sequence of vectors, such that for every k we have $\|(x^{k+1} - x^k)/X^k\|_2 \leq C < \frac{1}{2}$, where $X^k = diag(x^k)$. Then there exist at most $O((Ck/\varepsilon)^2)$ many i s.t. $x_i^k > (1+\varepsilon)x^1$ or $x_i^k < (1-\varepsilon)x^1$.*

Proof. For $c \leq 0.5$ we have $|\log \frac{x_i^k}{x_i^1}| \leq 2|\frac{x_i^k}{x_i^1} - 1|$ which allows us to bound the following norm:

$$\|\log \frac{x^k}{x^1}\|_2 = \|\log \prod_{i=1}^{k-1} \frac{x^{i+1}}{x^i}\|_2 \leq \sum_{i=1}^{k-1} \|\log \frac{x^{i+1}}{x^i}\|_2 \leq 2\sum_{i=1}^{k-1} \|\frac{x^{i+1}}{x^i} - 1\|_2 \leq 2kC$$

Let T be the number of indices i with $x_i^k \geq (1+\varepsilon)x_i^1$ or $x_i^k \leq (1-\varepsilon)x_i^1$. We want to find an upper bound of T.

Without loss of generality we can also assume that x^k and x^1 differ in at most $T+1$ entries. The reason is as follows: Let's say we are allowed to choose the sequence of $x^1, ..., x^k$ and we want to maximize T. Assume there is more than one index i with $x_i^k \geq (1+\varepsilon)x_i^1$ or $x_i^k \leq (1-\varepsilon)x_i^1$. Let $i \neq j$ be two such indices, then we could have tried to increase T by not changing the jth entry and changing ith entry a bit more.

This leads to $T \cdot \log(1+\varepsilon) \leq \|\log \frac{x^k}{x^1}\|_1 \leq \sqrt{T+1}\|\log \frac{x^k}{x^1}\|_2 \leq 2\sqrt{T}kC$ which can be reordered to $T = O((kC/\varepsilon)^2)$.

□

[9] Recent developments indicate that at least the current techniques for fast matrix multiplication do not allow for $\omega < 2 + 1/6 < 2.168$ [Alm19, AW18a, AW18b].

[10] https://creativecommons.org/licenses/by/4.0/

Lemma A.2. *The preprocessing requires $O(n^2 d^{\omega-2})$ time. After T updates the total time of all updates of Algorithm 1, when ignoring the branch for $k < n^a$ (so we assume that branch of line 36 has cost 0), is*

$$O(T \cdot C/\varepsilon_{mp}(n^{\omega-1/2} + n^{2-a/2+o(1)}) \log n).$$

Proof. The preprocessing cost is dominated by computing $M = A^\top (AUA^\top)^{-1} A$, which takes $O(n^2 d^{\omega-2})$ time. For the update complexity, we first modify the algorithm a bit. We replace the loop of line 23 by: $k \leftarrow 2^\ell$ for the smallest integer ℓ with $y_{\pi(2^\ell)} < (1 - 0.5\ell/\log n)\varepsilon_{mp}$. ($\ell = \log n$ if no such ℓ exists.)

As we ignore the branch for $k < n^a$ in the complexity analysis, we are left with analyzing the cost of performing a rank $k = 2^\ell$ update via the Sherman-Morrison-Woodbury identity. The cost for this is $O(n^{\omega(1,1,\ell/\log n)})$, where $\omega(a,b,c)$ refers to the number of arithmetic operations required to compute the matrix product of an $n^a \times n^b$ with an $n^b \times n^c$ matrix. So the total cost for T calls to UPDATE is bounded by

$$\sum_{\ell=0}^{\log n} (\text{number of rank } 2^\ell \text{ updates}) \cdot O(n^{\omega(1,\ell/\log n,1)}).$$

We now prove that the number of rank 2^ℓ updates is at most $O(T(C/\varepsilon_{mp})2^{-\ell/2}\log n)$ by showing that there must be at least $\Omega((\varepsilon_{mp}/C)2^{\ell/2}\log^{-1} n)$ calls to UPDATE between any two rank 2^ℓ updates.

After a rank 2^ℓ update, we have by choice of ℓ that $|u_i^{\text{new}}/\widetilde{u}_i - 1| < (1 - 0.5\ell/\log n)\varepsilon_{mp}$ for all i. Let $u^{(0)}$ be the input vector u^{new} to UPDATE, when we performed the rank 2^ℓ update and let $u^{(1)}, u^{(2)}, ...$, be the input sequence to all further calls to UPDATE from that point on. Likewise let $\widetilde{u}^{(0)}, \widetilde{u}^{(1)}, ...$ be the internal vectors of the data-structure after these calls to UPDATE. Then we have

$$|u_i^{(0)}/\widetilde{u}_i^{(0)} - 1| < (1 - 0.5\ell/\log n)\varepsilon_{mp}$$

for all i, but when we perform another rank 2^ℓ update some t calls to UPDATE later, we have at least $2^{\ell-1}$ indices i with

$$|u_i^{(t)}/\widetilde{u}_i^{(t-1)} - 1| \geq (1 - 0.5(\ell-1)/\log n)\varepsilon_{mp}.$$

That means either $u_i^{(t)}$ differs to $u_i^{(0)}$ by some $(1 \pm \Omega(\varepsilon_{mp}/\log n))$-factor, or $\widetilde{u}_i^{(t-1)}$ differs to $u_i^{(0)}$ by some $(1 \pm \Omega(\varepsilon_{mp}/\log n))$-factor (which means there exists some $t' < t$ where $u_i^{(t')}$ differs to $u_i^{(0)}$ by some $(1 \pm \Omega(\varepsilon_{mp}/\log n))$-factor, which caused $\widetilde{u}_i$ to receive an update).

So in summary, we know there must be at least $2^{\ell-1}$ indices i for which the input vectors u changed by some $(1 \pm \Omega(\varepsilon_{mp}/\log n))$-factor compared to $u^{(0)}$. By Lemma A.1 this can happen only after at least $\Omega(\varepsilon_{mp} C^{-1} 2^{\ell/2}\log^{-1} n)$ calls to UPDATE, as the multiplicative change between any $u^{(k)}$ and $u^{(k+1)}$ is bounded by C.

Note that by definition we only perform rank $2^\ell \geq n^a$ updates. The total time can thus be bounded by

$$\begin{aligned}
&\sum_{\ell=0}^{\log n} (\text{number of rank } 2^\ell \text{ updates}) \cdot O(n^{\omega(1,\ell/\log n,1)}) \\
\leq &\sum_{\ell=\lceil a \log n \rceil}^{\log n} O(T(C/\varepsilon_{mp})2^{-\ell/2} n^{\omega(1,\ell/\log n,1)} \log n) \\
= &\, O(T(C/\varepsilon_{mp})(n^{\omega-0.5} + n^{\omega(1,a,1)-a/2}) \log n)
\end{aligned}$$

The last equality uses that $\omega(1,1,x)$ is a convex function, so the largest term of the sum must be the first or the last one. If we assume $a \leq \alpha$, then $n^{\omega(1,a,1)-a/2} = n^{2+o(1)-a/2}$, which leads to the complexity as stated in Lemma A.2. □

Lemma A.3 ([YTM94, CLS18])**.** *Consider a linear program* $\min_{Ax=b,x\geq 0} c^\top x$ *with* n *variables and* d *constraints. Assume that*

1. *Diameter of the polytope: For any* $x \geq 0$ *with* $Ax = b$*, we have that* $\|x\|_1 \leq R$*.*
2. *Lipschitz constant of the LP:* $\|c\|_\infty \leq L$*.*

For any $0 < \gamma \leq 1$*, the modified linear program* $\min_{\overline{A}\overline{x}=\overline{b},\overline{x}\geq 0} \overline{c}^\top \overline{x}$ *with*

$$\overline{A} = \begin{bmatrix} A & 0 & \frac{1}{R}b - A1_n \\ 1_n^\top & 1 & 0 \\ -1_n^\top & -1 & 0 \end{bmatrix}, \overline{b} = \begin{bmatrix} \frac{1}{R}b \\ n+1 \\ -(n+1) \end{bmatrix}, \text{ and } \overline{c} = \begin{bmatrix} \gamma/L \cdot c \\ 0 \\ 1 \end{bmatrix}$$

satisfies the following:

1. $\overline{x} = \begin{bmatrix} 1_n \\ 1 \\ 1 \end{bmatrix}, \overline{y} = \begin{bmatrix} 0_d \\ 0 \\ 1 \end{bmatrix}$ *and* $\overline{s} = \begin{bmatrix} 1_n + \frac{\gamma}{L} \cdot c \\ 1 \\ 1 \end{bmatrix}$ *are feasible primal dual vectors.*
2. *For any feasible primal dual vectors* $(\overline{x}, \overline{y}, \overline{s})$ *with* $\sum_{i=1}^n \overline{x}_i \overline{s}_i \leq \gamma^2$*, consider the vector* $\widehat{x} = R \cdot \overline{x}_{1:n}$ *(*$\overline{x}_{1:n}$ *is the first* n *coordinates of* $\overline{x}$*) is an approximate solution to the original linear program in the following sense*

$$\begin{aligned} c^\top \widehat{x} &\leq \min_{Ax=b,x\geq 0} c^\top x + LR \cdot \gamma, \\ \|A\widehat{x} - b\|_1 &\leq 2\gamma \cdot \left(R \sum_{i,j} |A_{i,j}| + \|b\|_1 \right), \\ \widehat{x} &\geq 0. \end{aligned}$$

B Projection Maintenance via Dynamic Linear System Solvers

The data-structure from [San04, vdBNS19] can maintain the solution to the following linear system: Let M be a non-singular $n \times n$ matrix and let b be an n-dimensional vector. Then the data-structures can maintain $M^{-1}b$ while supporting changing any entry of M or b in $O(n^{1.529})$ time.

This differs from the problem we must solve for the linear system (1), where we must maintain Pv for $P = \sqrt{X/S}A^\top (A\frac{X}{S}A)^{-1}A\sqrt{X/S}$ and the updates change entries of X and S. However, even though the structure seems very different, one can maintain Pv via the following reduction:

Lemma B.1. *Let* A *be a* $d \times n$ *matrix of rank* d *and let* U *be an* $n \times n$ *diagonal matrix with non-zero*

diagonal entries. Then

$$\begin{pmatrix} U^{-1} & A^\top & \sqrt{U}^{-1} & 0 \\ A & 0 & 0 & 0 \\ 0 & 0 & -I & 0 \\ (\sqrt{U}^{-1})^\top & 0 & 0 & -I \end{pmatrix}^{-1} \begin{pmatrix} 0_n \\ 0_n \\ v \\ 1_n \end{pmatrix} = \begin{pmatrix} * \\ * \\ * \\ \sqrt{U}A^\top(AUA^\top)^{-1}\sqrt{U}v \end{pmatrix},$$

where $*$ represents some entries that do not care about.

We can thus maintain Pv by using a data-structure that maintains $M^{-1}b$ by changing the diagonal entries of the U^{-1} and $\sqrt{U}^{-1}$ blocks.

Proof of Lemma B.1. The inverse of a two-blocks $\times$ two-blocks matrix is given by

$$\begin{pmatrix} Q & R \\ S & T \end{pmatrix}^{-1} = \begin{pmatrix} Q^{-1} + Q^{-1}R(T - SQ^{-1}R)^{-1}SQ^{-1} & -Q^{-1}R(T - SQ^{-1}R)^{-1} \\ -(T - SQ^{-1}R)^{-1}SQ^{-1} & (T - SQ^{-1}R)^{-1} \end{pmatrix}$$

If $Q = U^{-1}$, $T = 0$, $R = A^\top$, $T = A$, then the matrix has full-rank (i.e. it is invertible) and the top-left block of the inverse is $U + UA^\top(AUA)^{-1}A^\top U$. Further, consider the following block-matrix and its inverse:

$$\begin{pmatrix} M & N & 0 \\ 0 & -I & 0 \\ N^\top & 0 & -I \end{pmatrix}^{-1} = \begin{pmatrix} M^{-1} & M^{-1}N & 0 \\ 0 & -I & 0 \\ N^\top M^{-1} & N^\top M^{-1}N & -I \end{pmatrix}$$

When M is the previous block-matrix and N is the $(n+d) \times n$ block-matrix $(\sqrt{U}^{-1}, 0_{n\times d})^\top$, then the matrix is exactly the one given in Lemma B.1 and the bottom-center block of the inverse is

$$N^\top M^{-1}N = \sqrt{U}^{-1}(U + UA^\top(AUA)^{-1}A^\top U)\sqrt{U}^{-1} = I + \sqrt{U}A^\top(AUA)^{-1}A^\top\sqrt{U}.$$

Let C be this $(3n+d)\times(3n+d)$ block-matrix specified in Lemma B.1 and let $b = (0_n, 0_d, v, 1_n)$ be an $(3n + d)$-dimensional vector, then the bottom n coordinates of $C^{-1}b$ are exactly $\sqrt{U}A^\top(AUA)^{-1}A^\top\sqrt{U}v$. ∎

One can use the data-structure of [San04] to maintain $\sqrt{\widetilde{U}}A^\top(A\widetilde{U}A)^{-1}A\sqrt{\widetilde{U}}f(\widetilde{v})$ similar to Lemma 4.1, where $\widetilde{U} = \operatorname{diag}(\widetilde{u})$ and $\widetilde{v}$ are approximate variants of the input parameters u and v. Whenever some entry of $\widetilde{U}$ or $\widetilde{v}$ must be changed, because the approximation no longer holds, the algorithm of [San04] spends $O(n^{1.529})$ time per changed entry of $\widetilde{U}$ and $\widetilde{v}$. This is not yet fast enough for our purposes, because when using this data-structure inside our linear program solver, up to $\Omega(n)$ entries might be changed throughout the entire runtime of the solver. Thus one would require $\Omega(n^{2.529})$ time for the solver.

By applying the complexity analyzsis of [CLS18] to this data-structure, one can achieve the same amortized complexity as in Lemma 4.1. We now briefly outline how this is done.

Per iteration of the linear system solver, more than one entry of $\widetilde{u}$ and $\widetilde{v}$ may have to be changed. This can be interpreted as a so called *batch-update*, and the complexity for batch-updates

was already analyzed in [vdBNS19], but again the focus was on worst-case complexity. Both data-structure from [San04] and [vdBNS19] had the property, that the data-structure would become slower the more updates they received. This issue was fixed by re-initializing the data-structure in fixed intervals. The core new idea of [CLS18] is a new strategy for this re-initialization: They wait until n^a many entries of $\widetilde{u}$ must be changed (see line 32 of Algorithm 1), and then they change preemptively a few more entries (see line 23).

Applying the same reset strategy to [San04, vdBNS19] then results in the same complexity as Lemma 4.1. Indeed the resulting data-structure is essentially identical to Lemma 4.1/Algorithm 1, because all these algorithms are just exploiting the Sherman-Morrison-Woodbury identity.

References

[AB95] Kurt M. Anstreicher and Robert A. Bosch. A new infinity-norm path following algorithm for linear programming. *SIAM Journal on Optimization*, 5(2):236–246, 1995. 3

[Alm19] Josh Alman. Limits on the universal method for matrix multiplication. In *CCC*, volume 137 of *LIPIcs*, pages 12:1–12:24. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019. 25

[Ans96] Kurt M. Anstreicher. Volumetric path following algorithms for linear programming. *Math. Program.*, 76:245–263, 1996. 1

[Ans99] Kurt M. Anstreicher. Linear programming in o([n3/ln n]l) operations. *SIAM Journal on Optimization*, 9(4):803–812, 1999. 1

[AW18a] Josh Alman and Virginia Vassilevska Williams. Further limitations of the known approaches for matrix multiplication. In *ITCS*, volume 94 of *LIPIcs*, pages 25:1–25:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. 25

[AW18b] Josh Alman and Virginia Vassilevska Williams. Limits on all known (and some unknown) approaches to matrix multiplication. In *FOCS*, pages 580–591. IEEE Computer Society, 2018. 25

[BCM99] Hervé Brönnimann, Bernard Chazelle, and Jivr'i Matouvsek. Product range spaces, sensitive sampling, and derandomization. *SIAM J. Comput.*, 28(5):1552–1575, 1999. Announced at FOCS'93. 1

[Cha00] Bernard Chazelle. A minimum spanning tree algorithm with inverse-ackermann type complexity. *J. ACM*, 47(6):1028–1047, 2000. 1

[Cha16] Timothy M. Chan. Improved deterministic algorithms for linear programming in low dimensions. In *SODA*, pages 1213–1219. SIAM, 2016. 1

[Cla95] Kenneth L. Clarkson. Las vegas algorithms for linear and integer programming when the dimension is small. *J. ACM*, 42(2):488–499, 1995. Announced at FOCS'88. 1

[CLS18] Michael B. Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. *CoRR*, abs/1810.07896, 2018. Announced at STOC'19. 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 21, 25, 27, 28, 29

[CM96] Bernard Chazelle and Jivr'i Matouvsek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *J. Algorithms*, 21(3):579–597, 1996. 1

[DHNS19] Mohit Daga, Monika Henzinger, Danupon Nanongkai, and Thatchaphol Saranurak. Distributed edge connectivity in sublinear time. *CoRR*, abs/1904.04341, 2019. To be announced at STOC'19. 1

[Gal14] François Le Gall. Powers of tensors and fast matrix multiplication. In *ISSAC*, pages 296–303. ACM, 2014. 2, 10, 24

[GU18] Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *SODA*, pages 1029–1046. SIAM, 2018. 2, 10, 24

[Kal92] Gil Kalai. A subexponential randomized simplex algorithm (extended abstract). In *STOC*, pages 475–482. ACM, 1992. 1

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396, 1984. Announced at STOC'84. 1, 2, 5

[Kar00] David R. Karger. Minimum cuts in near-linear time. *J. ACM*, 47(1):46–76, 2000. Announced at STOC'96. 1

[Kha79] Leonid G Khachiyan. A polynomial algorithm in linear programming. In *Doklady Academii Nauk SSSR*, volume 244, pages 1093–1096, 1979. 1

[KT19] Ken-ichi Kawarabayashi and Mikkel Thorup. Deterministic edge connectivity in near-linear time. *J. ACM*, 66(1):4:1–4:50, 2019. 1

[LS13] Yin Tat Lee and Aaron Sidford. Path finding i: Solving linear programs with õ(sqrt(rank)) linear system solves. *CoRR*, abs/1312.6677, 2013. 2

[LS14] Yin Tat Lee and Aaron Sidford. Path finding methods for linear programming: Solving linear programs in õ(vrank) iterations and faster algorithms for maximum flow. In *FOCS*, pages 424–433. IEEE Computer Society, 2014. 1

[LS15] Yin Tat Lee and Aaron Sidford. Efficient inverse maintenance and faster algorithms for linear programming. In *FOCS*, pages 230–249. IEEE Computer Society, 2015. 1, 2

[LSZ19] Yin Tat Lee, Zhao Song, and Qiuyi Zhang. Solving empirical risk minimization in the current matrix multiplication time. *CoRR*, abs/1905.04447, 2019. Announced at FOCS'19. 2, 9

[Meg89] Nimrod Megiddo. Pathways to the optimal set in linear programming. In *Progress in mathematical programming*, pages 131–158. Springer, 1989. 1

[MRSV17] Jack Murtagh, Omer Reingold, Aaron Sidford, and Salil P. Vadhan. Derandomization beyond connectivity: Undirected laplacian systems in nearly logarithmic space. In *FOCS*, pages 801–812. IEEE Computer Society, 2017. 1

[MSW96] Jivr'i Matouvsek, Micha Sharir, and Emo Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4/5):498–516, 1996. Announced at SOCG'92. 1

[NN89] Yu Nesterov and Arkadi Nemirovskiy. Self-concordant functions and polynomial-time methods in convex programming. *Report, Central Economic and Mathematic Institute, USSR Acad. Sci*, 1989. 1, 2

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

[NT97] Yurii E. Nesterov and Michael J. Todd. Self-scaled barriers and interior-point methods for convex programming. *Math. Oper. Res.*, 22(1):1–42, 1997. 1

[PR02] Seth Pettie and Vijaya Ramachandran. An optimal minimum spanning tree algorithm. *J. ACM*, 49(1):16–34, 2002. Announded at ICALP'00. 1

[PS82] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982. 4

[Ren88] James Renegar. A polynomial-time algorithm, based on newton's method, for linear programming. *Math. Program.*, 40(1-3):59–93, 1988. 1, 2, 4

[San04] Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse (extended abstract). In *FOCS*, pages 509–517. IEEE Computer Society, 2004. 3, 6, 27, 28, 29

[Son19] Zhao Song. *Matrix Theory : Optimization, Concentration and Algorithms*. PhD thesis, The University of Texas at Austin, 2019. 2

[VA93] Pravin M Vaidya and David S Atkinson. A technique for bounding the number of iterations in path following algorithms. In *Complexity in Numerical Optimization*, pages 462–489. World Scientific, 1993. 1

[Vai87] Pravin M. Vaidya. An algorithm for linear programming which requires o(((m+n)nˆ2 + (m+n)ˆ1.5 n)l) arithmetic operations. In *STOC*, pages 29–38. ACM, 1987. 1, 4

[Vai89a] Pravin M. Vaidya. A new algorithm for minimizing convex functions over convex sets (extended abstract). In *FOCS*, pages 338–343. IEEE Computer Society, 1989. 1

[Vai89b] Pravin M. Vaidya. Speeding-up linear programming using fast matrix multiplication (extended abstract). In *FOCS*, pages 332–337. IEEE Computer Society, 1989. 1, 2

[vdBNS19] Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. *CoRR*, abs/1905.05067, 2019. 3, 6, 27, 29

[Wil12] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *STOC*, pages 887–898. ACM, 2012. 2, 10, 24

[YTM94] Yinyu Ye, Michael J. Todd, and Shinji Mizuno. An o($\sqrt{}$nl)-iteration homogeneous and self-dual linear programming algorithm. *Math. Oper. Res.*, 19(1):53–67, 1994. 4, 27