

Acceleration Meets Inverse Maintenance: Faster ℓ_∞ -Regression

Deeksha Adil
Institute for Theoretical Studies
ETH Zürich
deeksha.adil@eth-its.ethz.ch

Shunhua Jiang
Department of Computer Science
Columbia University
sj3005@columbia.edu

Rasmus Kyng
Department of Computer Science
ETH Zürich
kyng@inf.ethz.ch

Abstract

We propose a randomized multiplicative weight update (MWU) algorithm for ℓ_∞ regression that runs in $\tilde{O}\left(n^{2+1/22.5}\text{poly}(1/\epsilon)\right)$ time when $\omega = 2 + o(1)$, improving upon the previous best $\tilde{O}\left(n^{2+1/18}\text{poly}\log(1/\epsilon)\right)$ runtime in the low-accuracy regime. Our algorithm combines state-of-the-art inverse maintenance data structures with acceleration. In order to do so, we propose a novel acceleration scheme for MWU that exhibits *stability* and *robustness*, which are required for the efficient implementations of the inverse maintenance data structures.

We also design a faster *deterministic* MWU algorithm that runs in $\tilde{O}\left(n^{2+1/12}\text{poly}(1/\epsilon)\right)$ time when $\omega = 2 + o(1)$, improving upon the previous best $\tilde{O}\left(n^{2+1/6}\text{poly}\log(1/\epsilon)\right)$ runtime in the low-accuracy regime. We achieve this by showing a novel stability result that goes beyond previously known works based on interior point methods (IPMs).

Our work is the first to use acceleration and inverse maintenance together efficiently, finally making the two most important building blocks of modern structured convex optimization compatible.

Contents

1	Introduction	4
1.1	Our Results	4
1.2	Background: The Ingredients of Fast ℓ_∞ -Regression Methods.	5
1.3	Discussion of Techniques	6
2	Technical Overview	9
2.1	Deterministic MWU Algorithm via One-Level Inverse Maintenance	9
2.2	Randomized MWU Algorithm via Two-Level Inverse Maintenance	10
3	Fast Width-Reduced MWU Algorithms	13
3.1	Lazy Update Procedure	13
3.2	Monotone Multiplicative Weights Update Algorithm	14
3.3	Algorithm with Non-Monotone Weights, Stability and Robustness	14
A	Preliminaries	20
B	Low Rank Update Scheme under Stability Guarantees	21
B.1	Low Rank Update under ℓ_2 Stability	21
B.2	Low Rank Update under ℓ_3 Stability	22
B.2.1	Decomposition of Iterations	22
B.2.2	Low Rank Update Scheme under ℓ_3 Stability	22
C	Data Structures	23
C.1	Inverse Maintenance Data Structure	23
C.2	Implicit Inverse Maintenance	24
C.3	ℓ_3 and ℓ_2 -Norm Estimations	25
C.4	ℓ_2 Heavy Hitter	25
D	Time Complexity of the Randomized Algorithm Using Fast Data Structures	26
D.1	Implementing MWU Using Fast Data Structures	26
D.2	Correctness of Algorithm	26
D.3	Time Complexity under ℓ_2 Stability	26
E	Time Complexity of the Deterministic Algorithm Using Fast Data Structures	26
F	Guarantees of Algorithm 1: MWU with Monotone Weights	28
G	Guarantees of Algorithm 3: Robust Primal Step and Stable Width Reduction Step	31
G.1	Starting Point: Non-Monotone MWU Algorithm	31
G.2	Warm Up: Stable Width Reduction Step	33
G.2.1	Definitions and Basic Properties	33
G.2.2	Change in Φ	35
G.2.3	Change in Ψ	35
G.2.4	Putting Everything Together: Analysis of Algorithm	35
G.3	Guarantees of Algorithm 3: Robust Primal Step	36
G.3.1	Sketching Bounds	36
G.3.2	Analysis of Algorithm 3	38

H	Stability Guarantees of Algorithms 1 and 3	39
H.1	Stability Guarantees of Algorithm 1	39
H.2	Algorithm Warm Up: Low-Rank Update Scheme	40
H.3	Algorithm 3	40
I	Missing Proofs	41
J	MWU with Non-Monotone Weights and $n^{1/3}$ Iterations (Optional)	42

1 Introduction

In this paper, we study the ℓ_∞ -regression problem. Given $\epsilon > 0$, a matrix $\mathbf{C} \in \mathbb{R}^{n \times d}$ and vector $\mathbf{d} \in \mathbb{R}^n$, $d \leq n$, we want to find $\tilde{\mathbf{x}} \in \mathbb{R}^d$ such that,

$$\|\mathbf{C}\tilde{\mathbf{x}} - \mathbf{d}\|_\infty \leq (1 + \epsilon) \min_{\mathbf{x} \in \mathbb{R}^d} \|\mathbf{C}\mathbf{x} - \mathbf{d}\|_\infty. \quad (1)$$

Some of the popular approaches to obtaining fast algorithms for ℓ_∞ -regression include using multiplicative weight update (MWU) routines [BN51; AHK12; Chr+11; Chi+13; Adi+19; EV19; ABS21], gradient descent [She13; Kel+14] and other ways to optimize a softmax function [Car+20; ST18; ABS21], and using interior point methods (IPM) [Kar84; Ren88; NN94]. Interior point methods can find a high-accuracy solution, i.e., an ϵ -approximate solution in $\tilde{O}(\sqrt{n} \log(1/\epsilon))^1$ linear system solves, whereas most of the other methods are low accuracy solvers, i.e., their running time scales as $\text{poly}(1/\epsilon)$. Naively using gradient descent or MWU requires $O(\sqrt{n} \cdot \text{poly}(1/\epsilon))$ linear system solves. Multiplicative weight update based approaches can be accelerated via a technique called *width reduction* to converge in $O(n^{1/3} \cdot \text{poly}(1/\epsilon))$ linear system solves [Chr+11; Chi+13; Adi+19; EV19; ABS21; Adi+24]. Several acceleration techniques have also been developed to improve the iteration complexity of other low-accuracy regression algorithms [MS13; Bul18; Car+20; ST18; ABS21].

To get an overall fast runtime, apart from improving the iteration complexity, a useful approach is to reduce the per-iteration cost. This can be done using *inverse maintenance*, which reduces the cost via *lazy-update* schemes. Notions of inverse maintenance appear in the very first interior point methods, [Kar84; NN89], but the modern form was introduced by Vaidya [Vai89]. There have been many important developments in inverse maintenance algorithms since then, and state-of-the-art algorithms use both linear algebraic data structures and dimensionality reduction routines, such as sketching [BNS19]. The improvements in runtimes of interior point methods including the state-of-the-art algorithms depend heavily on these developments in inverse maintenance routines [LS15; CLS21; Bra20; Jia+21; LV21].

1.1 Our Results

For simplicity, in the discussion of our results and prior work on this problem, we focus on the case $\omega = 2 + o(1)$ – but our full technical theorems give results for all ω . In the low-accuracy regime of $\epsilon = 1/\text{polylog}(n)$ the state-of-the-art running time for ℓ_∞ -regression is $\tilde{O}(n^{2+1/18})$, obtained via the randomized algorithm of [Jia+21], and $\tilde{O}(n^{2+1/6})$ for deterministic algorithms via [Bra20]. Both these algorithms in fact obtain high-accuracy solutions, and they use inverse maintenance, but no acceleration. In this work, we push the running time further in the low-accuracy regime by combining the state-of-the-art inverse maintenance techniques of these results with new multiplicative weight methods which allow us to perform acceleration, yielding running times of $\tilde{O}(n^{2+1/22.5} \text{poly}(\epsilon^{-1}))$ with randomization and $\tilde{O}(n^{2+1/12} \text{poly}(\epsilon^{-1}))$ without.

Our first result is a deterministic algorithm that combines acceleration and lazy inverse updates in a novel, more sophisticated way, and achieves a running time of $\tilde{O}(n^{2+1/12} \text{poly}(\epsilon^{-1}))$. This improves on deterministic state-of-the-art $\tilde{O}(n^{2+1/6} \log(\epsilon^{-1}))$ [Bra20] in the low-accuracy regime. The key to this result is a new notion of ℓ_3 -*stability* which is tailored to the accelerated MWU.

Theorem 1.1 (Informal statement of Theorem E.1). *There is a deterministic algorithm that solves Problem (1) in $\tilde{O}(n^{2+1/12} \text{poly}(\epsilon^{-1}))$ time when $\omega = 2 + o(1)$. This algorithm converges in $\tilde{O}(n^{1/3} \text{poly}(\epsilon^{-1}))$ iterations.*

¹We use $\tilde{O}(\cdot)$ to hide $\text{poly log } n$ factors, and we use $\tilde{O}_\epsilon(\cdot)$ to additionally hide $\text{poly}(\epsilon^{-1})$ factors.

Our main result is our randomized algorithm with running time $\tilde{O}(n^{2+1/22.5} \text{poly}(\epsilon^{-1}))$.

Theorem 1.2 (Informal statement of Theorem D.2). *There is a randomized algorithm that solves Problem (1) in $\tilde{O}(n^{2+1/22.5} \text{poly}(\epsilon^{-1}))$ time when $\omega = 2 + o(1)$. This algorithm converges in $\tilde{O}(n^{1/2.5} \text{poly}(\epsilon^{-1}))$ iterations.*

To obtain this result, we introduce the first MWU which can combine all three key techniques for ℓ_∞ -regression: (a) acceleration, (b) lazy inverse updates, and (c) sketching.

Thus, we give the optimization approach method which is able to efficiently combine these three key techniques of structured convex optimization. This is likely an essential building block toward $n^{2+o(1)}$ optimization for many objectives. If, some day, acceleration is achieved for linear programming, an equivalent integration will be necessary for optimal algorithms in this context. Before describing our new approach, we first review existing techniques for fast ℓ_∞ -regression.

1.2 Background: The Ingredients of Fast ℓ_∞ -Regression Methods.

Both MWUs and IPMs that solve ℓ_∞ -regression methods rely on a sequence of calls to ℓ_2 -oracles, i.e. a subroutine that solves an ℓ_2 -minimization problem, or equivalently, solves a linear equation. In order to solve the ℓ_∞ -regression problem (1), a standard MWU approach repeatedly solves a sequence of ℓ_2 -oracle problems of the form

$$\mathbf{x}^{(i)} = \arg \min_{\mathbf{x} \in \mathbb{R}^d} \sum_e r_e^{(i)} (\mathbf{C} \mathbf{x} - \mathbf{d})_e^2 \quad (2)$$

where the weights $\{r_e^{(i)}\}$ are chosen by the MWU depending on the magnitude of previous iterates.

Inverse maintenance via stability and robustness. The ℓ_2 -oracles of MWUs and IPMs can be implemented by applying the inverse of a matrix, and inverse maintenance can be used to solve the sequence of ℓ_2 -oracle calls faster than simply performing a full matrix inversion or linear equation solve on each call. Two key phenomena drive inverse maintenance: *stability* and *robustness*. *Stability* is the property that the inputs to the ℓ_2 -oracle only change slowly. In the MWU case, this means the weights $\{r_e^{(i)}\}$ change slowly. We say an optimizer is *robust* if it can make progress using answers from ℓ_2 -oracles with somewhat inaccurate inputs. The combination of stability and robustness is especially powerful. Together, these properties ensure that we can delay making small coordinate updates to inputs until they build up to a large cumulative update, and that we only get few large cumulative updates, enabling the use of coordinate-sparse update techniques. This approach of batching together small updates is known as *lazy* inverse updating. Obtaining further speed-ups using sketching also crucially relies on robustness. Because of robustness, we can afford to use sketching to estimate $\mathbf{x}^{(i)}$, as long as our estimates allow sufficiently accurate updates to the weights $\{r_e^{(i)}\}$.

The IPM of [CLS21] first achieved a running time of $\tilde{O}(n^{2+1/6} + n^\omega)$ by introducing a method with excellent stability and robustness, which in turn allowed them to implement a powerful inverse maintenance approach using lazy updates and sketching. Later, [Bra20] showed that the same running time can be obtained deterministically using only lazy updates, and finally [Jia+21] gave an improved running time of $\tilde{O}(n^{2+1/18} + n^\omega)$ using both lazy updates and sketching. The approach of [Jia+21] can be thought of as a two-level inverse maintenance, and the use of the randomized sketching techniques is crucial for them to efficiently implement the *query* operation of this data structure. It remains open if there exists any deterministic IPM that can run faster than $\tilde{O}(n^{2+1/6} + n^\omega)$.

Acceleration via width-reduction. In oracle-based optimization, there is a long history of developing *accelerated* methods, which reduce the iteration count compared to more basic approaches. This can be traced back to accelerated solvers for quadratic objectives [Lan52; HS52] and first-order acceleration for gradient Lipschitz functions ([Nes83] and earlier works by Nemirovski). Christiano et al. [Chr+11] developed an acceleration method for multiplicative weight methods that reduces the iteration count for solving ℓ_∞ regression with ℓ_2 -oracles from $\tilde{O}(\sqrt{n} \cdot \text{poly}(1/\epsilon))$ to $\tilde{O}(n^{1/3} \cdot \text{poly}(1/\epsilon))$. An alternative approach to acceleration for ℓ_∞ -regression can be obtained via the methods of Monteiro and Svaiter [MS13], and has also been a major research topic, but is beyond the scope of our discussion. For simplicity of our remaining discussion, we ignore ϵ dependencies. A rough outline of the MWU acceleration approach of Christiano et al. [Chr+11] is as follows: The MWU solves a sequence of ℓ_2 -oracle problems returning iterates $\mathbf{x}^{(i)}$. If we scale the problem so that $\|\mathbf{C}\mathbf{x}^* - \mathbf{d}\|_\infty \leq 1$, then weights ensure that (a) in each iteration, $\|\mathbf{C}\mathbf{x}^{(i)} - \mathbf{d}\|_\infty \lesssim \sqrt{n}$ and (b) after $T = \tilde{O}(\sqrt{n})$ iterations, $\tilde{\mathbf{x}} = \frac{1}{T} \sum_i \mathbf{x}^{(i)}$ has $\|\mathbf{C}\tilde{\mathbf{x}} - \mathbf{d}\|_\infty \leq 1 + \epsilon$. [Chr+11] made an important modification: if in some iteration we have $\|\mathbf{C}\mathbf{x}^{(i)} - \mathbf{d}\|_\infty \geq \rho \approx n^{1/3}$, then instead of using $\mathbf{x}^{(i)}$, we will adjust the weights $\{r_e^{(i)}\}$ in order to reduce the value of $\|\mathbf{C}\mathbf{x}^{(i')} - \mathbf{d}\|_\infty$ for future iterates $\mathbf{x}^{(i')}$. Using this method, an approximately optimal $\tilde{\mathbf{x}} = \frac{1}{T} \sum_i \mathbf{x}^{(i)}$ can be found in $T = \tilde{O}(n^{1/3})$ iterations. The parameter ρ measures the ℓ_∞ -norm $\|\mathbf{C}\mathbf{x}^{(i)} - \mathbf{d}\|_\infty$ of each iterate, sometimes known as the *width*, and the weight-adjustment steps of Christiano et al. are hence known as *width reduction steps*. When the oracle width can be reduced in this way, we will say our method is *width-reducible*. This acceleration has never been developed for ℓ_∞ -regression in the high-accuracy regime (i.e. running times that scale as $\text{polylog}(1/\epsilon)$), and whether this is possible is one of the major open questions in convex optimization.

Weight monotonicity in MWUs: an obstacle to sketching. Many MWU methods are designed to have an important property, which we call *weight monotonicity*. Concretely, in [Chr+11] and many other MWUs, the oracle weights $\{r_e^{(i)}\}$ are only growing. This often simplifies analyses greatly, and helps establish other properties including stability, robustness, and width-reducibility. Referring back to our oracle queries introduced above in (2), let us define $\tilde{\mathbf{x}}^{(i)} = \frac{1}{T} \sum_{j \leq i} \mathbf{x}^{(j)}$. Weight monotonicity arises because we choose the weights based on an overestimate of $|(\mathbf{C}\tilde{\mathbf{x}}^{(i)} - \mathbf{d})_e|$ given by $\gamma_i = \frac{1}{T} \sum_{j \leq i} |(\mathbf{C}\mathbf{x}^{(j)} - \mathbf{d})_e|$. In particular, choosing $r_e^{(i)} = \exp(\alpha \gamma_i)$ for some scaling factor α will ensure the weights only grow. As we will discuss later, *weight monotonicity* seems inherently incompatible with sketching, and thus we will need to develop a non-monotone MWU. Prior work by Madry [Mad13; Mad16] introduced non-monotone weights in a highly specialized IPM for unit-capacity maximum flow. This IPM of Madry has MWU-like properties and allows for some acceleration. The method has other drawbacks including low stability and robustness, but nonetheless inspired some of our design choices.

Prior inverse maintenance with acceleration. We are aware of a single prior work which combined lazy inverse updates with an accelerated MWU to obtain a running time of $\tilde{O}(n^{2+1/3} + n^\omega)$ for ℓ_p -regression [Adi+19]. This approach is relatively naive, falling short of the $\tilde{O}(n^{2+1/6} + n^\omega)$ running time which can be achieved using only lazy inverse updates.

1.3 Discussion of Techniques

The crucial algorithmic techniques we rely on for speeding up ℓ_∞ -regression are (a) acceleration, (b) lazy inverse updates, and (c) sketching. We can view each of these techniques as being en-

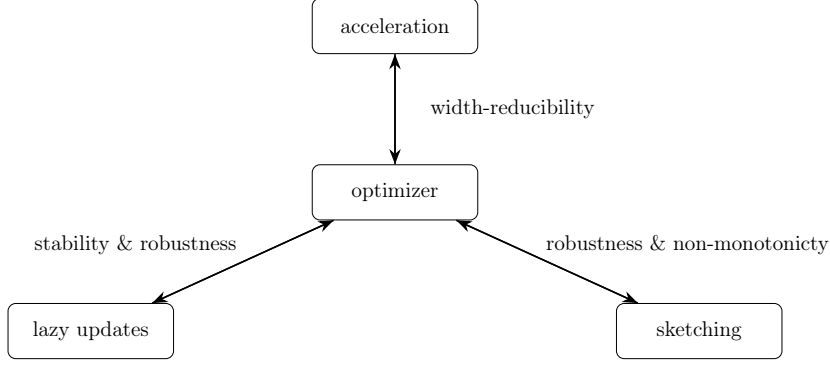


Figure 1: Algorithmic techniques and their requirements on our optimizer.

abled by different properties of the overall optimization approach. Our approach to acceleration is enabled by *width-reducibility*, while lazy updates require *stability* and *robustness*, and finally sketching requires *robustness* and *non-monotonicity*. This means we need to develop an MWU which simultaneously exhibits all these properties, i.e. it must be stable, robust, non-monotone, and width-reducible. In Figure 1, we summarize how our algorithmic techniques impose different requirements on our optimization approach. Again, for simplicity, in the remaining discussion of our results and prior work on this problem, we focus on the case $\omega = 2 + o(1)$.

We first discuss how to combine stability, robustness, and width-reducibility in a *monotone* MWU, which leads to a comparatively simple, deterministic algorithm using acceleration and lazy inverse updates, but no sketching.

Stability and robustness of a monotone, width-reducible MWU. [Adi+19] showed how to obtain stability, robustness, and width-reducibility together, with a monotone MWU. However, this work only established a weak notion of stability and hence comparatively slow running time of $\tilde{O}(n^{2+1/3})$. In contrast, one can show that by directly using stability and robustness in a monotone accelerated MWU, one can adapt the data structure approach of [Bra20] to achieve a running time of $\tilde{O}(n^{2+1/9})$, yielding a faster MWU.

Our first result Theorem E.1 is based on the observation that monotone MWU also enables a new, stronger notion of stability, which we call ℓ_3 -stability. This allows us to further reduce the number of lazy updates we make and lets us achieve a deterministic running time of $\tilde{O}(n^{2+1/12})$.

Non-monotone MWU - a key ingredient for sketching. As we described above, it is relatively easy to improve the running time of low-accuracy ℓ_∞ -regression among deterministic algorithms, by designing a monotone, robust, width-reducible MWU with a novel ℓ_3 -stability.

To further accelerate the algorithm by using a two-level inverse maintenance data structure, we need to use randomized sketching techniques to efficiently implement the query operation, which is required in every iteration of the MWU algorithm. Unfortunately, weight monotonicity is in conflict with sketching, because monotonicity arises from ignoring cancellations in $(\mathbf{C}\tilde{\mathbf{x}}^{(i)} - \mathbf{d})_e$ between different iterations.² In contrast, when using sketching, we want to crucially rely on cancellation between different iterations, as we sometimes overestimate $(\mathbf{C}\mathbf{x}^{(i)} - \mathbf{d})_e$ and sometimes underestimate it, but get it right on average. Because of this, we design an MWU with non-monotone weights. This in turn makes width-reducibility, robustness, and stability much harder to obtain.

²Recall that the final output of our MWU is the last averaged iterate $\tilde{\mathbf{x}}^{(T)}$.

To allow us to work with non-monotone weights and still obtain acceleration, we introduce a more delicate width-reduction scheme, inspired by [Mad16]. We also provide a tighter analysis of the sketching technique (it was named coordinate-wise embedding by [LSZ19; Jia+21]) that upper bounds its total noise across different iterations using martingale concentration inequalities. This tighter analysis is necessary to control the overall error introduced by the sketching technique in our MWU algorithm. We believe this tighter analysis could also provide a simpler analysis for the IPM results of [CLS21; Jia+21].

This new width-reduction approach in turn also requires us to estimate an ℓ_3 -norm associated with each iterate $\mathbf{x}^{(i)}$, and to do this quickly, we need to employ new sketching tools. To implement this approach, we also need an additional heavy-hitter sketch that allows us to identify which weights to adjust during width reduction.

Stability and robustness of a non-monotone, width-reducible MWU. Stability and robustness are crucial when we want to use lazy updates and sketching for inverse maintenance. Standard techniques for acceleration by width-reduction are unstable in the context of *non-monotone* MWU. Thus, to combine stability, width-reduction, and non-monotonicity, we have to further change our width-reduction strategy.

A central challenge is that width-reducibility is inherently in tension with the other properties. To simultaneously achieve stability and width-reducibility, we introduce a new and rather different approach to width-reduction, which we call *stable width-reduction*. This approach is more conservative than existing methods, and uses smaller width-reduction steps to achieve stability.

Combining width-reducibility with robustness is also difficult. Width-reduction relies on identifying too-large entries of the oracle outputs and making adjustments to the corresponding weights. But, robustness requires us to operate with inaccurate weights. We want to allow for weights that are inaccurate up to a factor $(1 \pm 1/\text{polylog}(n))$, and this is enough to completely change which oracle outputs are too large. In fact, we do not achieve general robust, but instead show that our method is robustness to (1) the errors induced by our specific lazy update scheme and (2) the errors induced by sketching.

Future perspectives. It remains open to design any algorithm for low-accuracy ℓ_∞ regression beyond $\tilde{O}(n^{2+1/22.5})$ when $\omega = 2 + o(1)$. We remark that if it were possible to use ℓ_3 -stability with the two-level data structure and an algorithm that converges in $n^{1/3}$ iterations, then we would achieve a runtime of $\tilde{O}(n^{2+1/48})$. However, the current techniques for inverse maintenance and acceleration are not sufficient to achieve $\tilde{O}(n^{2+o(1)} + n^\omega)$, which we believe would require substantially new techniques. On the other hand, even obtaining slight improvements in the runtime would require more robust acceleration and inverse maintenance frameworks which would be of independent interest.

In this paper, we analyze our algorithms in the RealRAM model. Establishing a similar analysis in finite precision arithmetic is an interesting open problem. Inverse maintenance-based IPM with finite precision arithmetic was studied by [GPV23].

We have demonstrated that acceleration techniques for MWU can be efficiently combined with inverse maintenance methods. For linear programming, no similar acceleration techniques exist and it is a major open problem to design these or rule out the possibility in various computational models. If acceleration can be achieved for linear programming, deploying it in conjunction with inverse maintenance will likely require techniques similar to those we introduce in this work.

2 Technical Overview

2.1 Deterministic MWU Algorithm via One-Level Inverse Maintenance

MWU methods reduce ℓ_∞ -regression problems to a sequence of ℓ_2 -minimization problems, which can be solved by solving systems of linear equations – or equivalently, applying the inverse of some matrix. More concretely, an MWU for finding approximate solutions to $\min_{\mathbf{x} \in \mathbb{R}^d} \|\mathbf{C}\mathbf{x} - \mathbf{d}\|_\infty$ requires us to repeatedly solve problems of the form

$$\min_{\Delta \in \mathbb{R}^d} \sum_e \mathbf{r}_e^{(i)} (\mathbf{C}\Delta - \mathbf{d})_e^2$$

across iterations $i = 1, \dots, T$. The exact solution to these minimization problems is given by

$$\Delta^{(i)} = (\mathbf{C}^\top \mathbf{R}^{(i)} \mathbf{C})^{-1} \mathbf{C}^\top \mathbf{R}^{(i)} \mathbf{d}.$$

The multiplicative weight update method iteratively updates the weights using $\Delta^{(i)}$ and “penalizes” the coordinates e that have large $|\mathbf{C}\Delta^{(i)} - \mathbf{d}|_e$ by increasing their weights $\mathbf{r}_e^{(i+1)}$ in the next iteration. In the end the method outputs $\mathbf{x} = \sum_{i=1}^T \Delta^{(i)} / T$ as the approximate ℓ_∞ minimizer.

The cost of each iteration is dominated by the time required to solve the corresponding system of linear equations for $\Delta^{(i)}$ – or equivalently, applying the inverse of some matrix. If solving this sequence of systems of linear equations can be done faster than naively solving each system separately, then we can speed up the cost per iteration of the MWU algorithm, and hence make the algorithm faster. A similar problem of solving a sequence of systems of linear equations was studied for the IPM algorithms [CLS21; Bra20; Jia+21], and they achieved speed-ups by using lazy updates with *inverse maintenance* data structures. They could use lazy updates because the IPM algorithm satisfies a stability guarantee and a robustness guarantee. More precisely, (1) IPMs satisfy an ℓ_2 -stability guarantee that the ℓ_2 -norm of the relative changes between two iterations is bounded, i.e., $\|\frac{\mathbf{r}^{(i+1)} - \mathbf{r}^{(i)}}{\mathbf{r}^{(i)}}\|_2^2 \leq O(1)$. (2) IPMs are still correct if the system of linear equations is solved with coordinate-wise approximate weights $\bar{\mathbf{r}} \approx_\delta \mathbf{r}$ for some $\delta > 0$.

As it turns out, the *monotone* MWU algorithm is also inherently stable and robust, even with acceleration. We can therefore use coordinate-wise approximate weights $\bar{\mathbf{r}}^{(i)} \approx_\delta \mathbf{r}^{(i)}$ in each iteration, and only update $\bar{\mathbf{r}}_e^{(i)}$ when it differs from $\mathbf{r}_e^{(i)}$ by more than δ . This ensures that the approximate weights $\bar{\mathbf{r}}^{(i)}$ undergoes low-rank updates. We present a robust version of the known *accelerated* multiplicative weights update method for ℓ_∞ -regression from [Chr+11; Chi+13] below, where when solving the system of linear equations for $\Delta^{(i)}$ we use the approximate weights $\bar{\mathbf{r}}^{(i)}$.

Theorem 2.1 ([Chi+13]). *Let $0 < \epsilon < 1/2$ and $0 \leq \delta \leq \epsilon/6$. Algorithm 1 returns $\hat{\mathbf{x}}$ such that $\|\mathbf{C}\hat{\mathbf{x}} - \mathbf{d}\|_\infty \leq 1 + O(\epsilon)$ in $\tilde{O}(n^{1/3}\epsilon^{-7/3})$ iterations. Each iteration solves a linear system as specified in Line 9 of the algorithm.*

In fact, we can prove that this algorithm satisfies an even stronger stability guarantee – a quantitatively strong type of ℓ_3 -stability, namely

$$\sum_{i=1}^T \left\| \frac{\mathbf{r}^{(i+1)} - \mathbf{r}^{(i)}}{\mathbf{r}^{(i)}} \right\|_3^3 \leq O(n^{1/3}).$$

The ℓ_3 -stability guarantee allows for the following lazy-update scheme: for every ℓ , in every 2^ℓ iterations perform an update of size $O(2^{3\ell})$ to $\bar{\mathbf{r}}^{(i)}$.

Together with the one-level inverse maintenance of Brand, Nanongkai, and Saranurak [BNS19], this improves upon the previous best deterministic algorithm for low-accuracy ℓ_∞ regression that runs in $O(n^\omega + n^{2+1/6})$. We present a simplified version of the data structure below, and the formal version tailored to our application is in Section C.1.

Algorithm 1 Monotone Width Reduced MWU Algorithm

```

1: procedure MWU-SOLVER( $\epsilon, \mathbf{C}, \mathbf{d}$ )
2:    $\mathbf{w}^{(0,0)} \leftarrow \mathbf{1}_n, \quad \mathbf{x}^{(0)} \leftarrow \mathbf{0}_d$ 
3:    $\tau \leftarrow \Theta\left(\frac{n^{\frac{1}{3}}}{\epsilon^{\frac{1}{3}}} \log \frac{n}{\Psi_0}\right), \alpha \leftarrow \Theta\left(n^{-\frac{1}{2}+\eta} \epsilon^{\frac{1}{3}} \left(\log \frac{n}{\Psi_0}\right)^{-1}\right), \eta \leftarrow \frac{1}{6}$ 
4:    $T \leftarrow \alpha^{-1} \epsilon^{-2} \log n$ 
5:    $i \leftarrow 0, k \leftarrow 0$ 
6:   while  $i < T$  do
7:      $\mathbf{r}_e^{(i,k)} \leftarrow \mathbf{w}_e^{(i,k)} + \frac{\epsilon}{n} \|\mathbf{w}^{(i,k)}\|_1$ 
8:      $\bar{\mathbf{r}}^{(i,k)} \leftarrow \text{SELECTVECTOR}(\mathbf{r}^{(i,k)}, i+k, \delta) \quad \triangleright \bar{\mathbf{r}} \approx_{\delta} \mathbf{r}$ 
9:      $\Delta^{(i,k)} \leftarrow \arg \min_{\Delta \in \mathbb{R}^d} \sum_e \bar{\mathbf{r}}_e^{(i,k)} (\mathbf{C}\Delta - \mathbf{d})_e^2 \quad \triangleright \Delta = (\mathbf{C}^\top \bar{\mathbf{R}}^{(i,k)} \mathbf{C})^{-1} \mathbf{C}^\top \bar{\mathbf{R}}^{(i,k)} \mathbf{d}$ 
10:    if  $\left\| \mathbf{C}\Delta^{(i,k)} - \mathbf{d} \right\|_\infty \leq \tau$  then  $\triangleright$  primal step
11:       $\mathbf{w}^{(i+1,k)} \leftarrow \mathbf{w}^{(i,k)} (1 + \epsilon \alpha |\mathbf{C}\Delta^{(i,k)} - \mathbf{d}|)$ 
12:       $\mathbf{x}^{(i+1)} \leftarrow \mathbf{x}^{(i)} + \Delta^{(i,k)}$ 
13:       $i \leftarrow i + 1$ 
14:    else
15:      For all coordinates  $e$  with  $|\mathbf{C}\Delta^{(i,k)} - \mathbf{d}|_e \geq \tau$   $\triangleright$  width reduction step
16:         $\mathbf{w}_e^{(i,k+1)} \leftarrow (1 + \epsilon) \mathbf{w}_e^{(i,k)} + \frac{\epsilon^2}{n} \|\mathbf{w}^{(i,k)}\|_1$ 
17:         $k \leftarrow k + 1$ 
18:  return  $\hat{\mathbf{x}} = \frac{\mathbf{x}^{(T)}}{T}$ 

```

Lemma 2.2 (One-level inverse maintenance, (Informal) Theorem 4.1 of [BNS19]). *There is a data structure that supports the following two operations to maintain the inverse of an $n \times n$ matrix M :*

- **Reset:** Reset M^{-1} to $(M + \Delta)^{-1}$, where Δ has k_0 non-zero entries. This operation can be done in $O(\mathcal{T}_{\text{mat}}(n, n, k_0))$ time.³
- **Query:** Output the vector $(M + \Delta)^{-1} \cdot v$ using the maintained M^{-1} and $M^{-1}v$, where Δ has at most n^{a_0} non-zero entries. This operation can be done in $O(n^{\omega_{a_0}} + n^{1+a_0})$ time.

Runtime when $\omega = 2$. For simplicity, we only show the runtime of our algorithm when $\omega = 2$ in this section and omit polylogarithmic factors. Let us choose the parameter $a_0 = 3/4$, so that we perform a reset operation whenever we accumulate more than $n^{a_0} = n^{3/4}$ updates to $\bar{\mathbf{r}}$. From our low-rank update scheme under the ℓ_3 stability guarantee, this only happens in every $n^{1/4}$ iterations. So we perform a reset operation with cost $O(n^2)$ (since $\omega = 2$) in every $O(n^{1/4})$ iterations, and over the total $O(n^{1/3})$ iterations this gives a total reset time of $O(n^{2-1/4} \cdot n^{1/3}) = O(n^{2+1/12})$.

We perform a query operation in every iteration with cost $O(n^{2a_0} + n^{1+a_0}) = O(n^{1+3/4})$. Over all $O(n^{1/3})$ iterations this gives a total query time of $O(n^{1+3/4} \cdot n^{1/3}) = O(n^{2+1/12})$. Therefore, the total runtime is the sum of the reset time and the query time, which is $O(n^{2+1/12})$ as claimed in Theorem 1.1.

2.2 Randomized MWU Algorithm via Two-Level Inverse Maintenance

To further improve the runtime of the algorithm, we will use the following, more efficient two-level inverse maintenance data structure.

³ $\mathcal{T}_{\text{mat}}(n, r, m)$ denotes the time complexity of multiplying an $n \times r$ matrix with an $r \times m$ matrix.

Lemma 2.3 (Two-level inverse maintenance, (Informal) Theorem 4.2 of [BNS19]). *There is a data structure that supports the following three operations to explicitly maintain the inverse of an $n \times n$ matrix M . The algorithm achieves the goal via explicitly maintaining the inverse of an $n \times n$ matrix M_0 and implicitly maintaining the inverse of another $n \times n$ matrix M_1 that differs from M_0 on at most n^{a_0} entries, and the true matrix M always differ from M_1 on at most n^{a_1} entries where $a_1 \leq a_0$:*

- **Reset:** *Reset M_0^{-1} to $(M_0 + \Delta_0)^{-1}$, where Δ_0 has k_0 non-zero entries. This operation can be done in $\mathcal{T}_{\text{mat}}(n, n, k_0)$ time.*
- **Partial reset:** *Implicitly reset M_1^{-1} to $(M_1 + \Delta_1)^{-1}$, where Δ_1 has k_1 non-zero entries. This operation can be done in $\mathcal{T}_{\text{mat}}(n, n^{a_0}, k_1)$ time.*
- **Query:** *Output ℓ entries of the vector $M^{-1} \cdot v$ using the maintained M_0^{-1} , M_1^{-1} (implicitly). This operation can be done in $\mathcal{T}_{\text{mat}}(n^{a_0}, n^{a_1}, \max\{n^{a_1}, \ell\})$ time.*

The total runtime of the above data structure is the sum of its reset, partial reset, and query times. Let us now compare the query times of this two-level data structure with the one-level version. Observe that, the query time of the one-level data structure is n^{1+a_0} and that of the two-level data structure is better than n^{1+a_0} only if $\ell = o(n)$. In other words, we get an improvement via the two-level data structure only if we have an algorithm that does not require querying the entire maintained vector $M^{-1}v$.

So far, such an improvement via the two-level data structure has only been utilized, although in a complicated way, in the work of Jiang et al. [Jia+21] where they give a fast algorithm for linear programming by using the data structure within the robust interior point method framework and querying a *sketch* of the vector at every iteration. It is still an open problem if one can achieve their runtime of $\approx n^{2+1/18}$ via a deterministic algorithm and it is conjectured that improving the runtime either requires an improved data structure or, a more sophisticated “dimension reduction technique” to work with the algorithm.

Sketching and non-monotone MWU. Similar to [Jia+21], in our work we also query a sketch of the maintained vector in every iteration. More precisely, in each iteration we use a random matrix $\mathbf{S} \in \mathbb{R}^{n^{1/2+\eta} \times n}$ where η is the acceleration that we get, i.e., the total number of iterations is $O(n^{1/2-\eta})$, and we compute an approximate step $\mathbf{S}^\top \cdot \mathbf{S} \cdot (\mathbf{C}^\top \Delta^{(i,k)} - \mathbf{d})$. Using the coordinate-wise embedding guarantee of the random matrix \mathbf{S} , we can ensure that for each coordinate we have

$$\left(\mathbf{S}^\top \mathbf{S} (\mathbf{C}^\top \Delta^{(i,k)} - \mathbf{d}) \right)_e \approx (\mathbf{C}^\top \Delta^{(i,k)} - \mathbf{d})_e.$$

We now require to change Line 11 of Algorithm 1 to update the weights by

$$\mathbf{w}^{(i+1,k)} \leftarrow \mathbf{w}^{(i,k)} \left(1 + \epsilon \alpha \cdot \mathbf{S}^\top \mathbf{S} (\mathbf{C} \Delta^{(i,k)} - \mathbf{d}) \right).$$

Note that we lose monotonicity of the weights with this new primal step. We have to use this non-monotone update because the absolute values $|\mathbf{S}^\top \mathbf{S} (\mathbf{C}^\top \Delta^{(i,k)} - \mathbf{d})|$ would result in an error that is around the standard deviation of the estimator in every update of $\mathbf{w}^{(i,k)}$ ’s, and this would add up over iterates. Since the entire analysis of the MWU methods depends on tracking potentials which are functions of the weights, we would incur a large error. To circumvent this issue we require a version of the MWU method where the weights are not updated monotonically, and the random noise introduced by the sketching matrix \mathbf{S} can cancel out with each other across different coordinates e and across different iterations i .

Monotonicity is crucial in accelerating MWU methods and it is non-trivial to achieve accelerated rates without it. A few works in graph algorithms have been successful in obtaining

accelerated rates without monotonicity [Mad16; LS20] for specific algorithms. In this paper, we extend the algorithm of Madry [Mad16] to regression and obtain an algorithm with non-monotone updates that also converges in $n^{1/3}$ iterations and is robust (Refer to Appendix J for the complete algorithm and analysis).

Interior point methods directly control the solution quality of the last iterate. In contrast, MWU algorithms only measure the quality of the average of the primal iterates $\Delta^{(i,k)}$. As a result, our bound on the final solution requires a new MWU analysis that can handle cancellations between iterates of the errors arising from using sketching. We achieve this by developing a tighter analysis that upper bounds the sum of the sketching error over multiple iterations:

$$\sum_{i=0}^t \left(\left(\mathbf{S}^\top \mathbf{S} (\mathbf{C} \Delta^{(i)} - \mathbf{d}) \right)_e - (\mathbf{C} \Delta^{(i)} - \mathbf{d})_e \right) \lesssim \frac{\sqrt{nt}}{\sqrt{b}}.$$

We prove this bound using Freedman’s concentration bound for martingales. We also believe this tighter analysis can simplify the sketching analysis for the previous IPM papers [CLS21; LSZ19; Jia+21].

Stability and robustness of non-monotone MWU. The non-monotone MWU with standard width reduction steps is neither stable nor satisfies a low-rank update per iteration. We propose a new width reduction step that satisfies a low-rank update scheme which is sufficient for our data structure. Our steps, however, do not satisfy ℓ_2 stability, which is a sufficient condition for the low-rank update scheme. Instead of increasing all weights by a factor of $(1 + \epsilon)$ as in Line 16 of Algorithm 1, our new width reduction step increases a carefully selected set of weights. As a result, we can ensure that whenever we increase a large set of weights, we also increase the potential by a lot, so this event doesn’t happen very often. This helps ensure that weight updates from width-reduction steps occur on a similar “schedule” to weight updates from our primal update steps, and it allows us to efficiently handle both in the inverse maintenance data structure (Refer to Algorithm 3 and Appendix G for the complete algorithm and analysis). To efficiently find the coordinates e to perform width reduction on, we use an additional *heavy-hitter* data structure to identify these Δ_e exactly. We can only afford to find $n^{1/2+\eta}$ such coordinates in each iteration. This restriction on the number of coordinates restricts us to set η to be $1/10$, and our final iteration complexity is $n^{1/2-\eta} = n^{2/5}$ instead of $n^{1/3}$. The non-monotone algorithm also requires estimating a weighted ℓ_3 -norm of $\hat{\Delta}^{(i,k)}$ ’s for which we use an additional sketch from [WZ13].

Unlike the width reduction steps, the primal steps are stable, and they satisfy the ℓ_2 stability,

$$\left\| \frac{\mathbf{r}^{(i+1)} - \mathbf{r}^{(i)}}{\mathbf{r}^{(i)}} \right\|_2^2 \leq O(n^{2\eta}).$$

Given the ℓ_2 stability guarantee, we again use coordinate-wise approximate weights $\bar{\mathbf{r}}^{(i)} \approx_\delta \mathbf{r}^{(i)}$ in each primal step, and only update $\bar{\mathbf{r}}_e^{(i)}$ to be $\mathbf{r}_e^{(i)}$ if it differs from $\mathbf{r}_e^{(i)}$ by more than δ . This again guarantees a low-rank update scheme for the primal steps: for every ℓ , in every 2^ℓ iterations we only perform an update of size $O(2^{2\ell} \cdot n^{2\eta})$ to $\bar{\mathbf{r}}^{(i)}$.

It is non-trivial to show that the accelerated non-monotone MWU is robust under such coordinate-wise approximations to the weights. This is because we do not update the weights in every primal step, and we lazily update them in future iterations. We use an amortization argument to show that we can still gain enough changes in the required potentials even when we defer some updates to the future. However, this means our accelerated non-monotone MWU is only robust under the specific approximate weights $\bar{\mathbf{r}}_e^{(i)}$ that are updated to be $\mathbf{r}_e^{(i)}$ whenever

it differs too much from $\mathbf{r}_e^{(i)}$. We cannot guarantee robustness if in every iteration we choose an arbitrary coordinate-wise approximation unless we consider the unaccelerated algorithm, which was guaranteed in the IPM algorithms.

Runtime when $\omega = 2$. Finally, we sketch the time complexity of our non-monotone MWU algorithm using sketching when $\omega = 2$. For simplicity, we omit polylogarithmic factors. Using the two-level inverse maintenance data structure of Lemma 2.3, we perform a reset operation whenever we accumulate more than n^{a_0} updates to $\bar{\mathbf{r}}$, and by our low-rank update scheme under the ℓ_2 stability guarantee, this only happens in every $n^{a_0/2-\eta}$ iterations. Similarly, we perform a partial reset operation whenever we accumulate more than n^{a_1} updates to $\bar{\mathbf{r}}$, and this only happens in every $n^{a_1/2-\eta}$ iterations. Finally, note that our query time is bounded by $n^{a_0+a_1}$ since we always ensure that we query for at most $\ell = O(n^{1/2+\eta})$ coordinates in each iteration. So our total runtime over $T = n^{1/2-\eta}$ iterations is

$$\underbrace{T \cdot \frac{n^2}{n^{a_0/2-\eta}}}_{\text{reset}} + \underbrace{T \cdot \frac{n^{1+a_0}}{n^{a_1/2-\eta}}}_{\text{partial reset}} + \underbrace{T \cdot n^{a_0+a_1}}_{\text{reset}} = n^{2.5-a_0/2} + n^{1.5+a_0-a_1/2} + n^{0.5-\eta+a_0+a_1}.$$

Choosing the parameters $a_0 = 1 - \frac{1-2\eta}{9}$ and $a_1 = 1 - \frac{1-2\eta}{3}$, we have that the total runtime is bounded by $O(n^{2+1/18-\eta/9})$. Since we achieve an acceleration of $\eta = 1/10$ and $n^{1/2-\eta} = n^{2/5}$ iterations, this gives the claimed $O(n^{2+1/18-\eta/9}) = O(n^{2+1/22.5})$ time complexity of Theorem 1.2.

3 Fast Width-Reduced MWU Algorithms

In this section, we present the formal guarantees of our multiplicative weight update routines: a deterministic MWU algorithm with monotone weights (Algorithm 1) that is used in Theorem 1.1, and a randomized MWU algorithm with non-monotone weights and stable and robust steps (Algorithm 3) that is used in Theorem 1.2.

3.1 Lazy Update Procedure

We first present the SELECTVECTOR algorithm (Algorithm 2) from [LV21] that computes a coordinate-wise approximate vector $\bar{\mathbf{r}}$ of \mathbf{r} such that $\bar{\mathbf{r}}$ undergoes small updates.

We remark that the only difference between our algorithm and that of [LV21] is in Line 11 where we only include a coordinate e in S if \mathbf{w}_e is not being updated by a width reduction step between primal iterations $i - 2^\ell$ and i . This is due to a minor technicality of dealing with the two kinds of steps, primal and width reduction, in Algorithm 11 and 3. In all our algorithms, if we toggle a coordinate e in a width reduction step, then we always update the “lazy” approximate vector $\bar{\mathbf{r}}_e$ to be the same as \mathbf{r}_e , so the guarantees of the SELECTVECTOR algorithm still hold under this change in Line 11.

Algorithm 2 Compute a coordinate-wise approximate vector that undergoes small updates [LV21]

```

1: procedure SELECTVECTOR( $\mathbf{r}^{(i)}, i, \delta$ )
2:    $\triangleright$  This procedure stores all previous  $\mathbf{r}^{(0)}, \dots, \mathbf{r}^{(i-1)}$ , and the  $\bar{\mathbf{r}}$  in the previous iteration
3:   if  $i = 0$  then
4:     return  $\bar{\mathbf{r}} \leftarrow \mathbf{r}^{(0)}$ 
5:    $S \leftarrow \emptyset$ 
6:   for  $\ell = 0, 1, \dots, \log n$  do
7:     if  $i \equiv 0 \pmod{2^\ell}$  then
8:       if  $\ell = \log n$  then
9:          $S \leftarrow [n]$ 
10:      else
11:         $S \leftarrow S \cup \{e : |\ln(\frac{\mathbf{r}_e^{(i)}}{\mathbf{r}_e^{(i-2^\ell)}})| \geq \frac{\delta}{2 \log n} \text{ and } \text{LASTWIDTH}(i, e) \leq i - 2^\ell\}$ 
12:         $\triangleright \text{LASTWIDTH}(i, e) \leq i$  is the last primal step during which a width
        reduction step updates  $w_e$ 
13:       $\bar{\mathbf{r}}_e \leftarrow \mathbf{r}_e^{(i)}$  for all  $e \in S$ 
14:      return  $\bar{\mathbf{r}}$ 

```

3.2 Monotone Multiplicative Weights Update Algorithm

We have already presented the convergence guarantees of Algorithm 1 in Theorem 2.1. We now add the stability guarantees that we use to prove the guarantees of our fast deterministic algorithm. The analysis of the algorithm is in Appendix F and the stability guarantees are in Appendix H.1.

Lemma 3.1 (Stability bound of ℓ_3 norm over all primal iterations). *Let k_i denote the number of width reduction steps taken by the algorithm when the i^{th} primal step is being executed. Then over all T primal steps of Algorithm 1, we have*

$$\sum_{i=0}^{T-1} \sum_{e \in S_i} \left(\frac{\mathbf{r}_e^{(i+1, k_i)} - \mathbf{r}_e^{(i, k_i)}}{\mathbf{r}_e^{(i, k_i)}} \right)^3 \leq \tilde{O}(\alpha^2 n) = \tilde{O}(n^{1/3} \epsilon^{2/3}).$$

Here S_i is the set of coordinates e at primal iteration i such that $\mathbf{r}_e^{(i+1, k_i)} \geq \mathbf{r}_e^{(i, k_i)} (1 + 3\epsilon\alpha)^4$.

Lemma 3.2 (Stability bound of ℓ_3 norm over all width reduction iterations). *Let i_k denote the number of primal steps taken before the execution of the k^{th} width reduction step. Then, over all K width reduction steps of Algorithm 1, we have*

$$\sum_{k=0}^{K-1} \left(\frac{\mathbf{r}_e^{(i_k, k+1)} - \mathbf{r}_e^{(i_k, k)}}{\mathbf{r}_e^{(i_k, k)}} \right)^3 \leq \tilde{O}(n^{1/3}).$$

3.3 Algorithm with Non-Monotone Weights, Stability and Robustness

We now give our main algorithm which can be used with our two-level inverse maintenance data structure. Algorithm 3 updates the weights in a non-monotone way, and additionally has stable primal and width reduction steps. It is also compatible with sketching as required by the data structure. We can prove the following guarantees.

⁴We note that it is sufficient to consider these sets S_i 's since any change that is smaller than the ones captured here can happen only $\tilde{O}(1)$ times.

Theorem 3.3. For $\eta \leq 1/10$, with probability $1-1/n^3$, Algorithm 3 with input $(\begin{bmatrix} \mathbf{C} \\ -\mathbf{C} \end{bmatrix}, \begin{bmatrix} \mathbf{d} \\ -\mathbf{d} \end{bmatrix}, \epsilon)$ finds $\hat{\mathbf{x}} \in \mathbb{R}^n$ such that $\|\mathbf{C}\hat{\mathbf{x}} - \mathbf{d}\|_\infty \leq 1 + O(\epsilon)$ in at most $\tilde{O}(n^{1/2-\eta}\epsilon^{-4})$ iterations. Furthermore, the algorithm satisfies the following extra guarantees:

1. In the width reduction step of the algorithm, the algorithm only requires to find at most $\tilde{O}(n^{1/2+\eta})$ large coordinates per iteration.
2. The algorithm satisfies the following low-rank update scheme: There are at most $\frac{T+K}{2^\ell}$ number of iterations where $\bar{\mathbf{r}}$ receives an update of rank $\tilde{O}_\epsilon(n^{1/5}2^\ell)$.

In order to get Algorithm 3 we begin by extending the graph based algorithms of [Mad16] to ℓ_∞ -regression. A direct extension (Algorithm 10, analysis included in Appendix J) does not have stable width reduction steps. We therefore design a new set of width reduction steps (Algorithm 11, Appendix G.2), which necessitates a new analysis for bounding the number of such steps. We then additionally add sketching to the primal steps to get the final algorithm which is analyzed in Appendix G.3.

Organization of Appendix

The appendix contains detailed proofs of our algorithms. In Appendix A we give some preliminaries and basic results we use for our proofs. In Appendix B we prove the guarantees of our low-rank update scheme under ℓ_2 and ℓ_3 stability, given by the subroutine SELECTVECTOR in Algorithm 3. Further in Appendix C we provide the guarantees for all the data structures required to implement our final algorithms. In Appendix D we show how to implement our randomized algorithm (Algorithm 3) using the data structures from Appendix C, and prove Theorem 1.2. In Appendix E we show how to implement our deterministic MWU algorithm (Algorithm 1) using the data structures from Appendix C, and prove Theorem 1.1. In Appendix F and G we prove the guarantees of Algorithm 1 and Algorithm 3 respectively. In Appendix H we prove the stability guarantees of Algorithms 1 and 3. In Appendix I we provide the missing proofs for standard results from Appendix A and C. Finally, in Appendix J (optional) we provide the analysis of a non-monotone MWU that is robust but not stable and it has $O_\epsilon(n^{1/3})$ iterations. This analysis is included for completeness, and is not required to prove our results.

Algorithm 3 Accelerated MWU algorithm with non-monotone weights and stable and robust steps

```

1: procedure MWU-NONMONOTONEROBUST( $\tilde{\mathbf{C}}, \tilde{\mathbf{d}}, \epsilon$ )
2:    $\mathbf{w}^{(0,0)} \leftarrow \mathbf{1}_{2n}, \quad \bar{\mathbf{r}}^{(0,0)} \leftarrow \mathbf{r}^{(0,0)} \leftarrow (1 + \epsilon)\mathbf{1}_{2n}, \quad \mathbf{x}^{(0)} \leftarrow \mathbf{0}_d$ 
3:    $\alpha \leftarrow \tilde{\Theta}(n^{-1/2+\eta\epsilon})$ 
4:    $\tau \leftarrow \tilde{\Theta}(n^{1/2+\eta\epsilon^{-4}}), \quad \rho \leftarrow \tilde{\Theta}(n^{1/2-3\eta\epsilon^{-2}})$ 
5:    $T \leftarrow \alpha^{-1}\epsilon^{-2} \ln n$ 
6:    $i, k = 0$ 
7:    $b \leftarrow \tilde{\Theta}(n^{1/2+\eta\epsilon^{-2}})$ 
8:   Let  $\mathbf{S}^{(0)}, \mathbf{S}^{(1)}, \dots, \mathbf{S}^{(T-1)} \in \mathbb{R}^{b \times 2n}$  be random matrices as described in Lemma G.8.
9:   while  $i < T$  do
10:      $\Delta^{(i,k)} \leftarrow (\tilde{\mathbf{C}}^\top \bar{\mathbf{R}}^{(i,k)} \tilde{\mathbf{C}})^{-1} \tilde{\mathbf{C}}^\top \bar{\mathbf{R}}^{(i,k)} \tilde{\mathbf{d}} \quad \triangleright \Delta^{(i,k)} = \arg \min_{\Delta} \sum_e \bar{\mathbf{r}}_e^{(i,k)} (\tilde{\mathbf{C}}\Delta - \tilde{\mathbf{d}})_e^2$ 
11:      $\mathbf{u}^{(i,k)} \leftarrow \tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}}$ 
12:      $\hat{\mathbf{u}}^{(i,k)} \leftarrow (\bar{\mathbf{R}}^{(i,k)})^{-1/2} \cdot (\mathbf{S}^{(i)})^\top \mathbf{S}^{(i)} \cdot (\bar{\mathbf{R}}^{(i,k)})^{1/2} \mathbf{u}^{(i,k)}$ 
13:      $\Psi(\bar{\mathbf{r}}^{(i,k)}) \leftarrow \sum_e \bar{\mathbf{r}}_e^{(i,k)} (\mathbf{u}_e^{(i,k)})^2$ 
14:     if  $\sum_e \bar{\mathbf{r}}_e^{(i,k)} |\mathbf{u}_e^{(i,k)}|^3 \leq C_3 \rho \Psi(\bar{\mathbf{r}}^{(i,k)})$  then  $\triangleright$  primal step
15:        $\mathbf{w}^{(i+1,k)} \leftarrow \mathbf{w}^{(i,k)} \left( 1 + \epsilon \bar{\alpha}^{(i,k)} \hat{\mathbf{u}}^{(i,k)} \right), \quad \bar{\alpha}_e^{(i,k)} = \begin{cases} \alpha \cdot (1 + \epsilon \alpha \hat{\mathbf{u}}_e^{(i,k)}) & \text{if } \hat{\mathbf{u}}_e^{(i,k)} \geq 0 \\ \alpha / (1 - \epsilon \alpha \hat{\mathbf{u}}_e^{(i,k)}) & \text{else} \end{cases}$ 
16:        $\mathbf{r}^{(i+1,k)} \leftarrow \mathbf{w}^{(i+1,k)} + \frac{\epsilon}{2n} \sum_e \mathbf{w}_e^{(i+1,k)}$ 
17:        $\bar{\mathbf{r}}^{(i+1,k)} \leftarrow \text{SELECTVECTOR}(\mathbf{r}^{(i+1,k)}, i+1, \delta)$   $\triangleright$  Algorithm 2
18:        $\mathbf{x}^{(i+1)} \leftarrow \mathbf{x}^{(i)} + \Delta^{(i,k)}$ 
19:        $i \leftarrow i + 1$ 
20:     else if  $\sum_e \bar{\mathbf{r}}_e^{(i,k)} |\mathbf{u}_e^{(i,k)}|^3 \geq C_3^{-1} \rho \Psi(\bar{\mathbf{r}}^{(i,k)})$  then  $\triangleright$  width reduction step
21:       Let  $S$  be the set of coordinates  $e$  such that  $|\mathbf{u}_e^{(i,k)}| \geq \rho/(2C_3)$ 
22:        $H \subseteq S$  be maximal subset such that  $\sum_{e \in H} \bar{\mathbf{r}}_e^{(i,k)} \leq \tau^{-1} \Psi(\bar{\mathbf{r}}^{(i,k)})$ 
23:       if  $H \neq S$  then
24:         Pick any  $\bar{e} \in S \setminus H$ .
25:         For all  $e \in H \cup \{\bar{e}\}$ ,  $\mathbf{w}_e^{(i,k+1)} \leftarrow (1 + \epsilon) \mathbf{w}_e^{(i,k)} + \frac{\epsilon^2}{2n} \Phi(\mathbf{w}^{(i,k)})$ 
26:          $\mathbf{r}^{(i,k+1)} \leftarrow \mathbf{w}^{(i,k+1)} + \frac{\epsilon}{2n} \Phi(\mathbf{w}^{(i,k+1)})$ 
27:         For all  $e \in H \cup \{\bar{e}\}$ ,  $\bar{\mathbf{r}}_e^{(i,k+1)} \leftarrow \mathbf{r}_e^{(i,k+1)}$ 
28:       else
29:         for  $\zeta = \rho, 2\rho, 4\rho, \dots, 2^{c_\rho} \rho$  do
30:            $\triangleright c_\rho$  is defined to be the smallest integer  $c$  that satisfies  $2^c \rho \geq \sqrt{n/\epsilon}$ 
31:           Define the set  $H_\zeta = \{e \in H \mid |\tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}}|_e \in [\zeta, 2\zeta]\}$ .
32:           If  $\sum_{e \in H_\zeta} \bar{\mathbf{r}}_e^{(i,k)} |\tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}}|_e^3 \geq \frac{\rho \Psi(\bar{\mathbf{r}}^{(i,k)})}{\log(\frac{n}{\epsilon\rho})}$ , set  $\zeta^* \leftarrow \zeta$ , and break.
33:           For all  $e \in H_{\zeta^*}$ ,  $\mathbf{w}_e^{(i,k+1)} \leftarrow (1 + \epsilon) \mathbf{w}_e^{(i,k)} + \frac{\epsilon^2}{2n} \Phi(\mathbf{w}^{(i,k)})$ 
34:            $\mathbf{r}^{(i,k+1)} \leftarrow \mathbf{w}^{(i,k+1)} + \frac{\epsilon}{2n} \Phi(\mathbf{w}^{(i,k+1)})$ 
35:           For all  $e \in H_{\zeta^*}$ ,  $\bar{\mathbf{r}}_e^{(i,k+1)} \leftarrow \mathbf{r}_e^{(i,k+1)}$ 
36:          $k \leftarrow k + 1$ 
37:   return  $\mathbf{x}^{(T)}/T$ 

```

References

- [ABS21] D. Adil, B. Bullins, and S. Sachdeva. “Unifying width-reduced methods for quasi-self-concordant optimization”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 19122–19133 (cit. on p. 4).
- [Adi+19] D. Adil, R. Kyng, R. Peng, and S. Sachdeva. “Iterative refinement for ℓ_p -norm regression”. In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2019, pp. 1405–1424 (cit. on pp. 4, 6, 7, 41).
- [Adi+24] D. Adil, R. Kyng, R. Peng, and S. Sachdeva. “Fast algorithms for ℓ_p -regression”. In: *Journal of the ACM* 71.5 (2024), pp. 1–45 (cit. on pp. 4, 28).
- [AHK12] S. Arora, E. Hazan, and S. Kale. “The Multiplicative Weights Update Method: A Meta-Algorithm and Applications”. In: *Theory of Computing* 8.6 (2012), pp. 121–164 (cit. on p. 4).
- [BCS97] P. Bürgisser, M. Clausen, and M. A. Shokrollahi. *Algebraic complexity theory*. Vol. 315. Springer Science & Business Media, 1997 (cit. on p. 21).
- [BN51] G. W. Brown and J. V. Neumann. “6. SOLUTIONS OF GAMES BY DIFFERENTIAL EQUATIONS”. In: *Contributions to the Theory of Games (AM-24), Volume I*. Ed. by H. W. Kuhn and A. W. Tucker. Princeton University Press, 1951, pp. 73–80. ISBN: 978-1-4008-8172-7 (cit. on p. 4).
- [BNS19] J. v. d. Brand, D. Nanongkai, and T. Saranurak. “Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds”. In: *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2019, pp. 456–480 (cit. on pp. 4, 9, 10, 11, 23).
- [Bra20] J. Brand. “A Deterministic Linear Program Solver in Current Matrix Multiplication Time”. In: *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2020, pp. 259–278 (cit. on pp. 4, 5, 7, 9).
- [Bra21] J. van den Brand. “Unifying Matrix Data Structures: Simplifying and Speeding up Iterative Algorithms”. In: *Symposium on Simplicity in Algorithms (SOSA)*. SIAM. 2021, pp. 1–13 (cit. on p. 24).
- [Bul18] B. Bullins. “Fast Minimization of Structured Convex Quartics”. In: *arXiv preprint arXiv:1812.10349* (2018). arXiv: 1812.10349 (cit. on p. 4).
- [Car+20] Y. Carmon, A. Jambulapati, Q. Jiang, Y. Jin, Y. T. Lee, A. Sidford, and K. Tian. “Acceleration with a ball optimization oracle”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 19052–19063 (cit. on p. 4).
- [Chi+13] H. H. Chin, A. Madry, G. L. Miller, and R. Peng. “Runtime guarantees for regression problems”. In: *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*. 2013, pp. 269–282 (cit. on pp. 4, 9).
- [Chr+11] P. Christiano, J. A. Kelner, A. Madry, D. A. Spielman, and S.-H. Teng. “Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs”. In: *Proceedings of the forty-third annual ACM symposium on Theory of computing*. 2011, pp. 273–282 (cit. on pp. 4, 6, 9).
- [CLS21] M. B. Cohen, Y. T. Lee, and Z. Song. “Solving linear programs in the current matrix multiplication time”. In: *Journal of the ACM (JACM)* 68.1 (2021), pp. 1–39 (cit. on pp. 4, 5, 8, 9, 12).

- [EV19] A. Ene and A. Vladu. “Improved Convergence for $\ell_{inf ty}$ and ℓ_1 Regression via Iteratively Reweighted Least Squares”. In: *Proceedings of Machine Learning Research* 97 (2019) (cit. on p. 4).
- [Fre75] D. A. Freedman. “On tail probabilities for martingales”. In: *the Annals of Probability* (1975), pp. 100–118 (cit. on p. 38).
- [GPV23] M. Ghadiri, R. Peng, and S. S. Vempala. “The bit complexity of efficient continuous optimization”. In: *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2023, pp. 2059–2070 (cit. on p. 8).
- [HS52] M. R. Hestenes and E. Stiefel. “On the Convergence of the Conjugate Gradient Method for Singular Linear Operator Equations”. In: *J. Research Nat. Bur. Standards* 49 (1952), pp. 409–436 (cit. on p. 6).
- [Jia+20] H. Jiang, T. Kathuria, Y. T. Lee, S. Padmanabhan, and Z. Song. “A faster interior point method for semidefinite programming”. In: *FOCS*. 2020 (cit. on p. 21).
- [Jia+21] S. Jiang, Z. Song, O. Weinstein, and H. Zhang. “A faster algorithm for solving general LPs”. In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. 2021, pp. 823–832 (cit. on pp. 4, 5, 8, 9, 11, 12, 24).
- [JL84] W. B. Johnson and J. Lindenstrauss. “Extensions of Lipschitz mappings into a Hilbert space”. In: *Contemporary mathematics* 26.189-206 (1984), p. 1 (cit. on p. 25).
- [Kan+11] D. M. Kane, J. Nelson, E. Porat, and D. P. Woodruff. “Fast moment estimation in data streams in optimal space”. In: *Proceedings of the forty-third annual ACM symposium on Theory of computing*. 2011, pp. 745–754 (cit. on p. 25).
- [Kar84] N. Karmarkar. “A New Polynomial-Time Algorithm for Linear Programming”. In: *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*. 1984, pp. 302–311 (cit. on p. 4).
- [Kel+14] J. A. Kelner, Y. T. Lee, L. Orecchia, and A. Sidford. “An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations”. In: *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2014, pp. 217–226 (cit. on p. 4).
- [Lan52] C. Lanczos. “Solution of Systems of Linear Equations by Minimized Iterations”. In: *J. Res. Nat. Bur. Standards* 49.1 (1952), pp. 33–53 (cit. on p. 6).
- [LS15] Y. T. Lee and A. Sidford. “Efficient inverse maintenance and faster algorithms for linear programming”. In: *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. IEEE. 2015, pp. 230–249 (cit. on p. 4).
- [LS20] Y. P. Liu and A. Sidford. “Faster energy maximization for faster maximum flow”. In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*. 2020, pp. 803–814 (cit. on p. 12).
- [LSZ19] Y. T. Lee, Z. Song, and Q. Zhang. “Solving empirical risk minimization in the current matrix multiplication time”. In: *Conference on Learning Theory*. PMLR. 2019, pp. 2140–2157 (cit. on pp. 8, 12, 36).
- [LV21] Y. T. Lee and S. S. Vempala. “Tutorial on the robust interior point method”. In: *arXiv preprint arXiv:2108.04734* (2021) (cit. on pp. 4, 13, 14, 21).
- [Mad13] A. Madry. “Navigating Central Path with Electrical Flows: From Flows to Matchings, and Back”. In: *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE, 2013, pp. 253–262 (cit. on p. 6).

- [Mad16] A. Madry. “Computing maximum flow with augmenting electrical flows”. In: *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2016, pp. 593–602 (cit. on pp. 6, 8, 12, 15, 31).
- [MS13] R. D. C. Monteiro and B. F. Svaiter. “An Accelerated Hybrid Proximal Extragradient Method for Convex Optimization and Its Implications to Second-Order Methods”. In: *SIAM Journal on Optimization* 23.2 (2013), pp. 1092–1125. ISSN: 1052-6234 (cit. on pp. 4, 6).
- [Nes83] Y. Nesterov. “A Method for Solving the Convex Programming Problem with Convergence Rate $o(1/K^2)$ ”. In: *Dokl Akad Nauk SSSR* 269 (1983), p. 543 (cit. on p. 6).
- [NN89] Y. E. Nesterov and A. Nemirovskii. “Self-Concordant Functions and Polynomial-Time Methods in Convex Programming”. In: *Report, Central Economic and Mathematical Institute, USSR Acad. Sci* (1989) (cit. on p. 4).
- [NN94] Y. Nesterov and A. Nemirovskii. *Interior-point polynomial algorithms in convex programming*. SIAM, 1994 (cit. on p. 4).
- [Pag13] R. Pagh. “Compressed matrix multiplication”. In: *ACM Transactions on Computation Theory (TOCT)* 5.3 (2013), pp. 1–17 (cit. on p. 25).
- [Ren88] J. Renegar. “A Polynomial-Time Algorithm, Based on Newton’s Method, for Linear Programming”. In: *Mathematical programming* 40.1 (1988), pp. 59–93 (cit. on p. 4).
- [She13] J. Sherman. “Nearly Maximum Flows in Nearly Linear Time”. In: *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE, 2013, pp. 263–269 (cit. on p. 4).
- [ST18] A. Sidford and K. Tian. “Coordinate methods for accelerating ℓ_∞ regression and faster approximate maximum flow”. In: *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2018, pp. 922–933 (cit. on p. 4).
- [Vai89] P. M. Vaidya. “Speeding-up Linear Programming Using Fast Matrix Multiplication”. In: *30th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1989, pp. 332–337 (cit. on p. 4).
- [WZ13] D. Woodruff and Q. Zhang. “Subspace embeddings and ℓ_p -regression using exponential random variables”. In: *Conference on Learning Theory*. PMLR. 2013, pp. 546–567 (cit. on pp. 12, 25).

A Preliminaries

Basic notations. For any vectors \mathbf{x} and \mathbf{y} with non negative entries, and $\delta > 0$ we use $\mathbf{x} \approx_\delta \mathbf{y}$ to imply that for all coordinates i , we have $e^{-\delta} \mathbf{y}_i \leq \mathbf{x}_i \leq e^\delta \mathbf{y}_i$. We use $\tilde{O}(\cdot)$ and $\tilde{\Theta}(\cdot)$ to hide poly log n factors, and we use $\tilde{O}_\epsilon(\cdot)$ and $\tilde{\Theta}_\epsilon(\cdot)$ to additionally hide poly(ϵ^{-1}) factors.

Given any two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, we use $\mathbf{x} \cdot \mathbf{y} \in \mathbb{R}^n$ to denote the coordinate-wise multiplication of the two vectors, i.e., its i -th entry is $\mathbf{x}_i \cdot \mathbf{y}_i$. Similarly, we also use other scalar operations on vectors to denote coordinate-wise operations.

For any vector $\mathbf{r} \in \mathbb{R}^n$, we use the capital letter $\mathbf{R} \in \mathbb{R}^{n \times n}$ to denote a diagonal matrix whose diagonal entries are \mathbf{r} .

Potential functions. In this paper we consider a fixed problem $\min_x \|\mathbf{C}\mathbf{x} - \mathbf{d}\|_\infty$ and assume that this has optimum objective value 1. We define the following two potential functions for weights \mathbf{w} and \mathbf{r} such that $\mathbf{r} = \mathbf{w} + \frac{\epsilon}{n} \|\mathbf{w}\|_1$:

$$\Phi(\mathbf{w}) \stackrel{\text{def}}{=} \|\mathbf{w}\|_1 \quad (3)$$

$$\Psi(\mathbf{r}) \stackrel{\text{def}}{=} \min_{\Delta \in \mathbb{R}^d} \sum_e \mathbf{r}_e (\mathbf{C}\Delta - \mathbf{d})_e^2. \quad (4)$$

The two potentials satisfy the following three lemmas. Their proofs are standard, and we defer them to Section I.

The first lemma shows how the two potentials are related.

Lemma A.1. *Let $\mathbf{w} \geq 0, \mathbf{r}, \bar{\mathbf{r}}$, such that $\forall e, \mathbf{r}_e = \mathbf{w}_e + \frac{\epsilon}{n} \|\mathbf{w}\|_1$, and $\bar{\mathbf{r}} \approx_\delta \mathbf{r}$. Then, $\Psi(\bar{\mathbf{r}}) \approx_\delta \Psi(\mathbf{r})$, and $\Psi(\bar{\mathbf{r}}) \leq e^{\epsilon+\delta} \cdot \Phi(\mathbf{w})$.*

In our algorithms we have the following lower bound on the initial Ψ potential.

Lemma A.2. *If $\mathbf{w}^{(0,0)} = 1$ and $\mathbf{r}^{(0,0)} = \mathbf{w}^{(0,0)} + \frac{\epsilon}{n} \cdot \Phi(\mathbf{w}^{(0,0)})$, then we have $\Psi(\mathbf{r}^{(0,0)}) \geq \Psi_0 \stackrel{\text{def}}{=} \min\{1, \mathbf{d}^\top (\mathbf{I} - \mathbf{C}^\top (\mathbf{C}^\top \mathbf{C})^{-1} \mathbf{C}) \mathbf{d}\}$.*

The last lemma provides a lower bound on the Ψ potential after a small perturbation to the weights \mathbf{r} .

Lemma A.3. *Let $\Psi(\mathbf{r}) = \min_\Delta \sum_e \mathbf{r}_e (\mathbf{C}\Delta - \mathbf{d})_e^2$. For any $\mathbf{r}', \mathbf{r} \geq 0$ that satisfies $|\mathbf{r}'_e - \mathbf{r}_e| \leq \mathbf{r}'_e$ for all e , we have*

$$\Psi(\mathbf{r}') \geq \Psi(\mathbf{r}) + \sum_e \left(\frac{\mathbf{r}'_e - \mathbf{r}_e}{\mathbf{r}'_e} \right) \mathbf{r}_e (\mathbf{C}\tilde{\Delta} - \mathbf{d})_e^2,$$

where $\tilde{\Delta} := \arg \min_\Delta \sum_e \mathbf{r}_e (\mathbf{C}\Delta - \mathbf{d})_e^2$.

Primal iterate and width iterate of MWU algorithms. We will use i to denote primal iterates and k to denote the width reduction iterations in our multiplicative weight update (mwu) algorithms. We use $\hat{\mathbf{u}}$ to denote the vector \mathbf{u} after applying sketching, and $\bar{\mathbf{u}}$ to denote an approximation to the vector \mathbf{u} . For the (mwu) algorithms, we would use i_k to denote the number of primal steps executed when the k^{th} width step is being taken, i.e., the k^{th} width step is from (i_k, k) to $(i_k, k+1)$, and we use k_i to denote the number of width reduction steps executed when the i^{th} primal step is taken, i.e., the i^{th} primal step is from (i, k_i) to $(i+1, k_i)$.

For any primal step i and any coordinate e , we define $\text{LASTWIDTH}(i, e)$ to be the largest $i' \leq i$ such that the algorithm executed a width reduction step from (i', k) to $(i', k+1)$ during which the weight of e is updated, i.e., $\mathbf{w}_e^{(i', k+1)} \neq \mathbf{w}_e^{(i', k)}$.

Fast matrix multiplication. We use $\mathcal{T}_{\text{mat}}(n, r, m)$ to denote the time complexity required to compute the product of an $n \times r$ matrix with an $r \times m$ matrix.

In our proofs we will frequently use the following fact. See e.g. [BCS97] for the basic properties of fast matrix multiplication exponents.

Fact A.4. $\mathcal{T}_{\text{mat}}(n, r, m) = O(\mathcal{T}_{\text{mat}}(n, m, r)) = O(\mathcal{T}_{\text{mat}}(m, n, r))$.

Definition A.5 (Fast matrix multiplication exponent). *For any β , define a function $\omega_\beta(x)$ to be the minimum value such that $\mathcal{T}_{\text{mat}}(n, n^x, n^\beta) = n^{\omega_\beta(x)+o(1)}$.*

With an abuse of notation we also define the function $\omega(x) = \omega_1(x)$, and define the value $\omega = \omega(1)$.

We also define $\alpha_ \in \mathbb{R}_+$ to be the dual exponent of matrix multiplication, i.e., $\omega(\alpha_*) = 2$.⁵*

We will also use the following fact about convexity. We present a proof (deferred to Section I) that generalizes the proof of Lemma 3.6 of [Jia+20].

Fact A.6 (Convexity). *For any β , $\omega_\beta(x)$ is convex in x .*

Fact A.7 (Upper bound of $\mathcal{T}_{\text{mat}}(n, n, r)$). *For any $r \leq n$,*

$$\mathcal{T}_{\text{mat}}(n, n, r) \leq n^{2+o(1)} + r^{\frac{\omega-2}{1-\alpha}} n^{2-\frac{\alpha(\omega-2)}{1-\alpha}+o(1)}.$$

B Low Rank Update Scheme under Stability Guarantees

B.1 Low Rank Update under ℓ_2 Stability

Algorithm 2 is the same as [LV21], and it satisfies the following lemma.

Lemma B.1 (Low-rank update scheme under ℓ_2 stability, Lemma 19 of [LV21]). *If we have the guarantee*

$$\sum_e \ln \left(\frac{\mathbf{r}_e^{(i+1)}}{\mathbf{r}_e^{(i)}} \right)^2 \leq \zeta,$$

then the Algorithm 2 outputs a vector $\bar{\mathbf{r}} \approx_\delta \mathbf{r}^{(i)}$ in each iteration, and the approximate vector $\bar{\mathbf{r}}$ undergoes a update of size $O((\frac{\log n}{\delta})^2 \cdot \zeta \cdot 2^\ell)$ in every 2^ℓ iterations for every $\ell \in [0 : \log T]$.

Next we show that the robust ℓ_2 stability guarantee also generates a δ -approximate sequence with low-rank updates.

Lemma B.2 (Low-rank update scheme under robust ℓ_2 stability). *If the sequence $\mathbf{r}_e^{(0)}, \dots, \mathbf{r}_e^{(T)}$ satisfies the following guarantee: There exists another sequence $\tilde{\mathbf{r}}_e^{(0)}, \dots, \tilde{\mathbf{r}}_e^{(T)}$ such that*

1.

$$\sum_e \ln \left(\frac{\tilde{\mathbf{r}}_e^{(i+1)}}{\mathbf{r}_e^{(i)}} \right)^2 \leq \zeta, \quad \forall i \in [0 : T],$$

2. $\forall t \leq t' \in [T], \forall e$, with probability $1 - 1/n^4$,

$$\left| \sum_{i=t'-t}^{t'} \ln \left(\frac{\tilde{\mathbf{r}}_e^{(i)}}{\mathbf{r}_e^{(i)}} \right) \right| \leq \frac{\delta}{10 \log n}.$$

⁵It's common in the literature to use α to denote the dual exponent of matrix multiplication. We use α_* here because we will use α to denote the “step size” of accelerated MWU.

Then the Algorithm 2 outputs a vector $\bar{\mathbf{r}} \approx_\delta \mathbf{r}^{(i)}$ in each iteration, and the approximate vector $\bar{\mathbf{r}}$ undergoes an update of size $O((\frac{\log n}{\delta})^2 \cdot \zeta \cdot 2^{2\ell})$ in every 2^ℓ iterations for every $\ell \in [0 : \log T]$.

Proof. **TOPROVE 0** □

B.2 Low Rank Update under ℓ_3 Stability

In this section we prove the low-rank update guarantee under ℓ_3 stability, which holds for MWU with monotone weights, and we only use it in our deterministic algorithm.

B.2.1 Decomposition of Iterations

Lemma B.3 (Decomposition of iterations). *If the weights satisfy that*

$$\sum_{i=1}^T \sum_e \left| \frac{\mathbf{r}_e^{(i+1)}}{\mathbf{r}_e^{(i)}} - 1 \right|^3 \leq \zeta,$$

then we can decompose the T iterations into $\log T + 1$ disjoint sets:

$$B_j := \left\{ i \in [T] \mid \frac{\zeta}{2^{j+1}} < \sum_e \left| \frac{\mathbf{r}_e^{(i+1)}}{\mathbf{r}_e^{(i)}} - 1 \right|^3 \leq \frac{\zeta}{2^j} \right\}, \quad \forall j \in [0 : \log T - 1],$$

$$B_{\log T} := \left\{ i \in [T] \mid \sum_e \left| \frac{\mathbf{r}_e^{(i+1)}}{\mathbf{r}_e^{(i)}} - 1 \right|^3 \leq \frac{\zeta}{T} \right\},$$

and these sets satisfy that $\cup_{j=0}^{\log T} B_j = [T]$, and $|B_j| \leq 2^{j+1}$ for all $j \in [\log T]$.

Proof. **TOPROVE 1** □

B.2.2 Low Rank Update Scheme under ℓ_3 Stability

Algorithm 4 Low rank update in the t -th iteration

- 1: **procedure** SELECTVECTORL3($\mathbf{r}^{(t)}$)
 - 2: **for** all $j \in [0 : \log T]$ **do**
 - 3: **for** all $\ell \in [0 : \log T]$ **do**
 - 4: **if** $i \in B_j$ and i is the k -th element in B_j where $k \equiv 0 \pmod{2^\ell}$ **then**
 - 5: $I \leftarrow \left\{ e \mid \sum_{k'=k-2^\ell}^k \left| \frac{\mathbf{r}_e^{(B_j[k']+1)}}{\mathbf{r}_e^{(B_j[k'])}} - 1 \right| \geq \frac{\delta}{10 \log^2 n} \right\}$
 - 6: Update the weights for all $e \in I$ to be $\bar{\mathbf{r}}_e^{(t)} \leftarrow \mathbf{r}_e^{(t)}$
-

Lemma B.4 (Low-rank update scheme under ℓ_3 stability). *Assume that the weights are monotonically increasing and satisfy*

$$\sum_{i=1}^T \sum_e \left| \frac{\mathbf{r}_e^{(i+1)}}{\mathbf{r}_e^{(i)}} - 1 \right|^3 \leq \zeta.$$

Define the sets $B_0, \dots, B_{\log T} \subseteq [T]$ as Lemma B.3, and for any j let $B_j[1], \dots, B_j[|B_j|]$ denote the elements in B_j in increasing order.

For any $\delta \leq 0.1$, Algorithm 4 maintains a vector $\bar{\mathbf{r}} \approx_\delta \mathbf{r}$ where $\bar{\mathbf{r}}$ undergoes the following updates: for any $j \in [0 : \log T]$, for any $\ell \in [0 : \log |B_j|]$, $\bar{\mathbf{r}}$ receives an update of size $O(\zeta \cdot 2^{3\ell-j} \cdot \frac{\log^6 n}{\delta^3})$ in iterations $B_j[2^\ell], B_j[2 \cdot 2^\ell], B_j[3 \cdot 2^\ell], \dots, B_j[\lfloor \frac{|B_j|}{2^\ell} \rfloor \cdot 2^\ell]$.

Proof. **TOPROVE 2** □

Corollary B.5. For any $j \in [0 : \log T]$ and any t , the total number of coordinates that are updated in iterations $B_j[k], \dots, B_j[k+t]$ is $O(\zeta \cdot 2^{3 \log t - j} \cdot \frac{\log^6 n}{\delta^3})$.

Proof. **TOPROVE 3** □

C Data Structures

In this section, we would present all the data structures we use for the various tasks in Algorithm 3.

C.1 Inverse Maintenance Data Structure

In this section we present the formal versions of Lemmas 2.2 and 2.3. These are the inverse maintenance data structures of [BNS19], and we have included a version of their results which is tailored to our notations and analysis. For completeness, we include the proofs of the following two lemmas in Section I.

Lemma C.1 (One-level inverse maintenance, Theorem 4.1 of [BNS19]). *There exists a data structure that initially has a matrix $\mathbf{M}^{(0)} \in \mathbb{R}^{n \times n}$, and in each iteration it receives an update $\Delta^{(t)} \in \mathbb{R}^{n \times n}$ to update the matrix to $\mathbf{M}^{(t)} = \mathbf{M}^{(t-1)} + \Delta^{(t)}$. The data structure maintains an iteration counter t_0 and it maintains the inverse $\mathbf{N} = (\mathbf{M}^{(t_0)})^{-1}$ internally. Let $k = \text{nnz}(\Delta^{(t_0+1)}) + \dots + \text{nnz}(\Delta^{(t)})$ denote the total size of the updates until the current iteration. The runtime for each operation of the data structure is as follows:*

- **Initialize**($\mathbf{M}^{(0)}$): Initially set $t_0 = 0$ and $\mathbf{N} = (\mathbf{M}^{(0)})^{-1}$. This operation takes $O(n^\omega)$ time.
- **Update**($\Delta^{(t)}$): The data structure receives the t -th update. This operation takes $O(\text{nnz}(\Delta^{(t)}))$ time.
- **Reset**(): Reset $t_0 = t$ and $\mathbf{N} = (\mathbf{M}^{(t_0)})^{-1}$. This operation takes $O(\mathcal{T}_{\text{mat}}(n, n, k))$ time.
- **Query**($J_r, J_c \subseteq [n]$): Output the submatrix $((\mathbf{M}^{(t)})^{-1})_{J_r, J_c}$ that has $|J_r| = \ell_r$ rows and $|J_c| = \ell_c$ columns. This operation takes $O(k^\omega + \mathcal{T}_{\text{mat}}(\ell_r, k, \ell_c))$ time.

Lemma C.2 (Two-level inverse maintenance, Theorem 4.2 of [BNS19]). *There exists a data structure that initially has a matrix $\mathbf{M}^{(0)} \in \mathbb{R}^{n \times n}$, and in each iteration it receives an update $\Delta^{(t)} \in \mathbb{R}^{n \times n}$ to update the matrix to $\mathbf{M}^{(t)} = \mathbf{M}^{(t-1)} + \Delta^{(t)}$. The data structure maintains two iteration counters $t_0 \leq t_1$, and it also maintains $k_0 := \text{nnz}(\Delta^{(t_0+1)}) + \dots + \text{nnz}(\Delta^{(t)})$ and $k_1 := \text{nnz}(\Delta^{(t_1+1)}) + \dots + \text{nnz}(\Delta^{(t)})$. Let $J \subseteq [n]$ denote the indexes of the non-zero columns of $\Delta^{(t_0+1)} + \dots + \Delta^{(t_1)}$. For any $t' \leq t$, define the transformation matrix*

$$\mathbf{T}^{(t', t)} := \mathbf{I} + (\mathbf{M}^{(t')})^{-1} \cdot (\mathbf{M}^{(t)} - \mathbf{M}^{(t')}) \in \mathbb{R}^{n \times n}.$$

The data structure maintains $\mathbf{N} = (\mathbf{M}^{(t_0)})^{-1} \in \mathbb{R}^{n \times n}$, and $\mathbf{B} = (\mathbf{T}_{J,J}^{(t_0,t_1)})^{-1}$ that has size at most $k_0 \times k_0$, and $\mathbf{E} = (\mathbf{T}_{J,J}^{(t_0,t_1)})^{-1} \cdot \mathbf{N}_{J,:}$. The runtime for each operation of the data structure is as follows:

- **Initialize**($\mathbf{M}^{(0)}$): Initially set $t_0 = t_1 = 0$, $\mathbf{B} = 0$, $\mathbf{E} = 0$, and $\mathbf{N} = (\mathbf{M}^{(0)})^{-1}$. This operation takes $O(n^\omega)$ time.
- **Update**($\Delta^{(t)}$): The data structure receives the t -th update. This operation takes $O(\text{nnz}(\Delta^{(t)}))$ time.
- **Reset**(): Reset $t_0 = t$ and $\mathbf{N} = (\mathbf{M}^{(t_0)})^{-1}$. This operation takes $O(\mathcal{T}_{\text{mat}}(n, n, k_0))$ time.
- **PartialReset**(): Reset $t_1 = t$, reset $J \subseteq [n]$ to be the indexes of the non-zero columns of $\Delta^{(t_0+1)} + \dots + \Delta^{(t)}$, and reset $\mathbf{B} = (\mathbf{T}_{J,J}^{(t_0,t)})^{-1}$ and $\mathbf{E} = (\mathbf{T}_{J,J}^{(t_0,t)})^{-1} \cdot \mathbf{N}_{J,:}$. This operation takes $O(\mathcal{T}_{\text{mat}}(n, k_0, k_1))$ time.
- **Query**($J_r, J_c \subseteq [n]$): Output the submatrix $((\mathbf{M}^{(t)})^{-1})_{J_r, J_c}$ that has $|J_r| = \ell_r$ rows and $|J_c| = \ell_c$ columns. This operation takes $O(\mathcal{T}_{\text{mat}}(k_0, k_1, \max\{k_1, \ell_c\}) + \mathcal{T}_{\text{mat}}(k_0, \ell_r, \ell_c))$ time.

We can maintain any matrix formula using the inverse maintenance data structure, as shown in [Bra21].

Theorem C.3 (Matrix formula as inverse, Theorem 3.1 of [Bra21]). *Given any formula f with input matrices $\mathbf{A}_1 \in \mathbb{R}^{n_1 \times m_1}, \dots, \mathbf{A}_d \in \mathbb{R}^{n_d \times m_d}$, where the formula f consists of only matrix addition, subtraction, multiplication, and inversion, define $n := \sum_{i=1}^d n_i + m_i$.*

Then there exists a symbolic block matrix \mathbf{N} of size at most $n \times n$, and sets $I, J \subset [n]$, such that for all matrices $\mathbf{A}_1, \dots, \mathbf{A}_d$ for which $f(\mathbf{A}_1, \dots, \mathbf{A}_d)$ is executable, $(\mathbf{N}(\mathbf{A}_1, \dots, \mathbf{A}_d)^{-1})_{I,J} = f(\mathbf{A}_1, \dots, \mathbf{A}_d)$.

Constructing \mathbf{N} from f can be done in $O(n^2)$ time.

C.2 Implicit Inverse Maintenance

In our algorithm, we also require a data structure that allows us to update $\mathbf{x}^{(i+1)} \leftarrow \mathbf{x}^{(i)} + \Delta^{(i,k)}$ in each primal step *implicitly* since we don't have the time budget to query the entire vector $\Delta^{(i,k)}$, and we only query the final vector $\mathbf{x}^{(T)}$ in the end. To solve this problem we present an implicit inverse maintenance data structure, and its proof can be found in Section I. Similar techniques were developed in Section I of [Jia+21] to maintain feasibility.

Lemma C.4 (Implicit two-level inverse maintenance). *There exists a data structure that initially has a matrix $\mathbf{M}^{(0)} \in \mathbb{R}^{n \times n}$ and a vector $\mathbf{v} \in \mathbb{R}^n$, and in each iteration it receives an update $\Delta^{(t)} \in \mathbb{R}^{n \times n}$ to update the matrix to $\mathbf{M}^{(t)} = \mathbf{M}^{(t-1)} + \Delta^{(t)}$. The goal of our algorithm is to support queries that output the sum of inverse vector products $\sum_{i=0}^t (\mathbf{M}^{(i)})^{-1} \cdot \mathbf{v}$ occasionally.*

The data structure maintains two iteration counters $t_0 \leq t_1$. Let $k_0 := \text{nnz}(\Delta^{(t_0+1)}) + \dots + \text{nnz}(\Delta^{(t)})$ and $k_1 := \text{nnz}(\Delta^{(t_1+1)}) + \dots + \text{nnz}(\Delta^{(t)})$. Similar as Lemma C.2, the data structure maintains $J \subseteq [n]$ that consists of the indexes of the non-zero columns of $\Delta^{(t_0+1)} + \dots + \Delta^{(t_1)}$, $\mathbf{N} = (\mathbf{M}^{(t_0)})^{-1} \in \mathbb{R}^{n \times n}$, $\mathbf{B} = (\mathbf{T}_{J,J}^{(t_0,t_1)})^{-1}$ that has size at most $k_0 \times k_0$, and $\mathbf{E} = (\mathbf{T}_{J,J}^{(t_0,t_1)})^{-1} \cdot \mathbf{N}_{J,:}$. The data structure also maintains three vector $\mathbf{u}_0, \mathbf{u}_1$, and \mathbf{u}_2 that satisfy the invariant:

$$\sum_{i=0}^t (\mathbf{M}^{(i)})^{-1} \mathbf{v} = \mathbf{u}_0 + \mathbf{N} \cdot \mathbf{u}_1 + \begin{bmatrix} \mathbf{N}_{J,:} \cdot \mathbf{u}_2 \\ 0 \end{bmatrix}.$$

The runtime for each operation of the data structure is as follows:

- **Initialize**($\mathbf{M}^{(0)}, \mathbf{v}$): Initially set $t_0 = t_1 = 0$, $\mathbf{B} = 0$, $\mathbf{E} = 0$, $\mathbf{N} = (\mathbf{M}^{(0)})^{-1}$, $\mathbf{u}_0 = (\mathbf{M}^{(0)})^{-1}\mathbf{v}$, and $\mathbf{u}_1 = \mathbf{u}_2 = 0$. This operation takes $O(n^\omega)$ time.
- **Update**($\Delta^{(t)}$): The data structure receives the t -th update and update $\mathbf{u}_0, \mathbf{u}_1$, and \mathbf{u}_2 . This operation takes $O(\mathcal{T}_{\text{mat}}(k_0, k_1, k_1) + n)$ time.
- **Reset**(): Reset $t_0 = t$ and $\mathbf{N} = (\mathbf{M}^{(t_0)})^{-1}$. This operation takes $O(\mathcal{T}_{\text{mat}}(n, n, k_0))$ time.
- **PartialReset**(): Reset $t_1 = t$, reset $J \subseteq [n]$ to be the indexes of the non-zero columns of $\Delta^{(t_0+1)} + \dots + \Delta^{(t)}$, and reset $\mathbf{B} = (\mathbf{T}_{J,J}^{(t_0,t)})^{-1}$ and $\mathbf{E} = (\mathbf{T}_{J,J}^{(t_0,t)})^{-1} \cdot \mathbf{N}_J$. This operation takes $O(\mathcal{T}_{\text{mat}}(n, k_0, k_1))$ time.
- **QuerySum**(): Output $\sum_{i=0}^t (\mathbf{M}^{(i)})^{-1}\mathbf{v}$. This operation takes $O(n^2)$ time.

C.3 ℓ_3 and ℓ_2 -Norm Estimations

We will also use the following ℓ_3 norm estimation lemma from [WZ13] to estimate the quantity on Line 14 of Algorithm 3.

Lemma C.5 (ℓ_3 norm estimation, Theorem 1 of [WZ13]). *There exists a distribution Π of matrices of size $O(n^{1/3} \log^3 n) \times n$ such that for any vector $\mathbf{x} \in \mathbb{R}^n$, with probability 0.99 we have that a random matrix $\mathbf{U} \sim \Pi$ satisfies*

$$C_3^{-1/3} \|\mathbf{x}\|_3 \leq \|\mathbf{U}\mathbf{x}\|_\infty \leq C_3^{1/3} \|\mathbf{x}\|_3,$$

where $C_3 > 1$ is a constant.

We remark that we can easily boost the success probability of the above theorem to $1 - 1/n^4$ by using $O(\log n)$ copies and take the median of the estimates.

We will also use the standard JL lemma to estimate the Ψ potential which can be written as a ℓ_2 norm.

Lemma C.6 (Johnson-Lindenstrauss Lemma [JL84]). *There exists a function $JL(n, m, \epsilon, \delta)$ that returns a random matrix $\mathbf{J} \in \mathbb{R}^{k \times n}$ where $k = O(\epsilon^{-2} \log(m/\delta))$, and \mathbf{J} satisfies that for any fixed m -element subset $V \subset \mathbb{R}^n$,*

$$\Pr [\forall \mathbf{v} \in V, (1 - \epsilon)\|\mathbf{v}\|_2 \leq \|\mathbf{J}\mathbf{v}\|_2 \leq (1 + \epsilon)\|\mathbf{v}\|_2] \geq 1 - \delta.$$

Furthermore, the function JL runs in $O(kn)$ time.

C.4 ℓ_2 Heavy Hitter

We use a heavy-hitter data structure to get a list of all the large coordinates on which we wish to perform width reduction in Algorithm 3.

Lemma C.7 (ℓ_2 heavy hitter, [Kan+11; Pag13]). *Given any n , ϵ , and δ , there exists a random matrix $\Phi \in \mathbb{R}^{O(\epsilon^{-2} \log(\delta^{-1}) \log n) \times n}$, and a decoding function DECODE , such that given a vector $\mathbf{y} = \Phi \cdot \mathbf{x}$ for some $\mathbf{x} \in \mathbb{R}^n$, $\text{DECODE}(\mathbf{y})$ outputs a list $L \subseteq [n]$ of size $|L| = O(\epsilon^{-2})$, where with probability $1 - \delta$ the list L includes all $i \in [n]$ that satisfies*

$$|\mathbf{x}_i| \geq \epsilon \cdot \|\mathbf{x}\|_2.$$

Furthermore, $\text{DECODE}(\mathbf{y})$ runs in $O(\epsilon^{-2} \log(\delta^{-1}) \log n)$ time.

D Time Complexity of the Randomized Algorithm Using Fast Data Structures

D.1 Implementing MWU Using Fast Data Structures

In this section, we give an algorithm, Algorithm 5, that implements Algorithm 3 using the data structures stated from Section C.

D.2 Correctness of Algorithm

Lemma D.1 (Correctness of Algorithm 5). *The output of Algorithm 5 is the same as that of Algorithm 3.*

Proof. TOPROVE 4 □

D.3 Time Complexity under ℓ_2 Stability

In this section we bound the time complexity of Algorithm 5.

Theorem D.2 (Time complexity of Algorithm 5). *For any parameters that satisfy $a_0 \leq \alpha_*$ and $a_1 \leq a_0 \cdot \alpha_*$, the time complexity of Algorithm 5 is*

$$\tilde{O}\left(n^\omega + n^{2.5-a_0/2} + n^{1.5+a_0-a_1/2} + n^{1/2-\eta+a_0+(\omega-1)a_1}\right) \cdot \text{poly}(\epsilon^{-1}).$$

In particular, when $\omega = 2 + o(1)$, this time complexity is bounded by

$$\tilde{O}\left(n^{2+1/22.5}\right) \text{poly}(\epsilon^{-1}).$$

Proof. TOPROVE 5 □

Time complexity when $\omega = 2$. We remark that when $\omega = 2$, we can choose the optimal trade-off $a_0 = 1 - \frac{1-2\eta}{9}$ and $a_1 = 1 - \frac{1-2\eta}{3}$, and the time complexity becomes

$$\tilde{O}\left(n^{2+(1-2\eta)/18}\right) \cdot \text{poly}(\epsilon^{-1}).$$

Since we choose $\eta = 1/10$, this become

$$\tilde{O}\left(n^{2+1/22.5}\right) \cdot \text{poly}(\epsilon^{-1}).$$

E Time Complexity of the Deterministic Algorithm Using Fast Data Structures

In this section we show that we can implement Algorithm 1 by using the one-level inverse maintenance data structure (Lemma C.1). This algorithm is deterministic since we don't use any randomized techniques.

Theorem E.1 (Time complexity of Algorithm 1 using one-level inverse maintenance). *For any parameters $\ell_0, \dots, \ell_{\log T} \in [0, \log T]$, the time complexity of Algorithm 1 when using one-level inverse maintenance is*

$$\tilde{O}\left(n^\omega + n^{5/3} \cdot \sum_{j=0}^{\log T} 2^{3\ell_j-j} + \sum_{j=0}^{\log T} \mathcal{T}_{\text{mat}}(n, n, n^{1/3} \cdot 2^{3\ell_j-j}) \cdot 2^{j-\ell_j}\right) \text{poly}(\epsilon^{-1}).$$

Algorithm 5 Implementing MWU Algorithm Using Fast Data Structures

```

1: procedure MWU-NONMONOTONEROBUST( $\mathbf{C}, \mathbf{d}, \epsilon, a_0, a_1$ )     $\triangleright$  Assume all variables are
   global
2:    $\alpha \leftarrow \tilde{\Theta}(n^{-1/2+\eta}\epsilon)$ 
3:    $\tau \leftarrow \tilde{\Theta}(n^{1/2+\eta}\epsilon^{-4}), \quad \rho \leftarrow \tilde{\Theta}(n^{1/2-3\eta}\epsilon^{-2})$ 
4:    $T \leftarrow \alpha^{-1}\epsilon^{-2} \ln n$ 
5:    $i, k = 0$ 
6:    $b \leftarrow \tilde{\Theta}(n^{1/2+\eta}\epsilon^{-3})$ 
7:    $\mathbf{w}^{(0,0)} \leftarrow \mathbf{1}_n, \quad \mathbf{x}^{(0)} \leftarrow \mathbf{0}_d$ 
8:   Let  $\mathbf{S}^{(0)}, \mathbf{S}^{(1)}, \dots, \mathbf{S}^{(T-1)} \in \mathbb{R}^{b \times n}$  be random matrices as described in Lemma G.8.
9:   Initialize data structures  $\text{DS}_{\text{INV}}, \text{DS}_{\text{NORM}}, \text{DS}_{\text{IMPLICITINV}}, \text{DS}_{\text{HEAVYHITTERS}}$   $\triangleright$  Algorithm 6,
   7, 8, 9
10:  while  $i < T$  do
11:     $\mathbf{r}^{(i,k)} \leftarrow \mathbf{w}^{(i,k)} + \frac{\epsilon}{m} \sum_e \mathbf{w}_e^{(i,k)}$ 
12:     $\bar{\mathbf{r}}^{(i,k)} \leftarrow \text{SELECTVECTOR}(\mathbf{r}^{(i,k)})$ 
13:     $\hat{\mathbf{u}}^{(i,k)} \leftarrow (\mathbf{R}^{(i,k)})^{-1/2} (\mathbf{S}^{(i)})^\top \cdot \text{DS}_{\text{INV}}.\text{UPDATEQUERY}(\bar{\mathbf{r}}^{(i,k)}, i)$ 
14:     $\Psi, \xi \leftarrow \text{DS}_{\text{NORM}}(\bar{\mathbf{r}}^{(i,k)}, i+k) \quad \triangleright \Psi \approx_\epsilon \sum_e \mathbf{r}_e^{(i,k)} (\mathbf{u}_e^{(i,k)})^2, \xi \approx_{C_3} \sum_e \mathbf{r}_e^{(i,k)} |\mathbf{u}_e^{(i,k)}|^3$ 
15:    if  $\xi \leq \rho\Psi$  then
16:       $\vec{\alpha}_e^{(i,k)} = \begin{cases} \alpha \cdot (1 + \epsilon\alpha\hat{\mathbf{u}}_e^{(i,k)}) & \text{if } \hat{\mathbf{u}}_e^{(i,k)} \geq 0 \\ \alpha/(1 - \epsilon\alpha\hat{\mathbf{u}}_e^{(i,k)}) & \text{else} \end{cases}$ 
17:       $\mathbf{w}^{(i+1,k)} \leftarrow \mathbf{w}^{(i,k)} \left(1 + \epsilon \vec{\alpha}^{(i,k)} \hat{\mathbf{u}}^{(i,k)}\right)$ 
18:       $\text{DS}_{\text{IMPLICITINV}}.\text{UPDATE}(\bar{\mathbf{r}}^{(i,k)}) \quad \triangleright$  Implicitly update  $\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \Delta^{(i,k)}$ 
19:       $i \leftarrow i + 1$ 
20:    else if  $\xi > \rho\Psi$  then
21:       $L, \mathbf{u}_L^{(i,k)} \leftarrow \text{DS}_{\text{HEAVYHITTERS}}.\text{UPDATEQUERY}(\bar{\mathbf{r}}^{(i,k)})$ 
22:      Let  $S$  be the set of coordinates  $e$  such that  $|\mathbf{u}_e^{(i,k)}| \geq \rho/(2C_3)$ 
23:       $H \subseteq S$  be maximal subset such that  $\sum_{e \in H} \bar{\mathbf{r}}_e^{(i,k)} \leq \tau^{-1}\Psi(\bar{\mathbf{r}}^{(i,k)})$ 
24:      if  $H \neq S$  then
25:        Pick any  $\bar{e} \in S \setminus H$ .
26:        For all  $e \in H \cup \{\bar{e}\}$ ,  $\mathbf{w}_e^{(i,k+1)} \leftarrow (1 + \epsilon)\mathbf{w}_e^{(i,k)} + \frac{\epsilon^2}{2n}\Phi(\mathbf{w}^{(i,k)})$ 
27:         $\mathbf{r}^{(i,k+1)} \leftarrow \mathbf{w}^{(i,k+1)} + \frac{\epsilon}{2n}\Phi(\mathbf{w}^{(i,k+1)})$ 
28:        For all  $e \in H \cup \{\bar{e}\}$ ,  $\bar{\mathbf{r}}_e^{(i,k+1)} \leftarrow \mathbf{r}_e^{(i,k+1)}$ 
29:      else
30:        for  $\zeta = \rho, 2\rho, 4\rho, \dots, 2^{c_\rho}\rho$  do
31:           $\triangleright c_\rho$  is defined to be the smallest integer  $c$  that satisfies  $2^c\rho \geq \sqrt{n/\epsilon}$ 
32:          Define the set  $H_\zeta = \{e \in H \mid |\tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}}|_e \in [\zeta, 2\zeta]\}$ .
33:          If  $\sum_{e \in H_\zeta} \bar{\mathbf{r}}_e^{(i,k)} |\tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}}|_e^3 \geq \frac{\rho\Psi(\bar{\mathbf{r}}^{(i,k)})}{\log(\frac{n}{\epsilon\rho})}$ , set  $\zeta^* \leftarrow \zeta$ , and break.
34:          For all  $e \in H_{\zeta^*}$ ,  $\mathbf{w}_e^{(i,k+1)} \leftarrow (1 + \epsilon)\mathbf{w}_e^{(i,k)} + \frac{\epsilon^2}{2n}\Phi(\mathbf{w}^{(i,k)})$ 
35:           $\mathbf{r}^{(i,k+1)} \leftarrow \mathbf{w}^{(i,k+1)} + \frac{\epsilon}{2n}\Phi(\mathbf{w}^{(i,k+1)})$ 
36:          For all  $e \in H_{\zeta^*}$ ,  $\bar{\mathbf{r}}_e^{(i,k+1)} \leftarrow \mathbf{r}_e^{(i,k+1)}$ 
37:         $k \leftarrow k + 1$ 
38:       $\mathbf{x}^{(T)} \leftarrow \text{DS}_{\text{IMPLICITINV}}.\text{QUERY}()$ 
39:    return  $\mathbf{x}^{(T)}/T$ 

```

Algorithm 6 Inverse maintenance data structure DS_{INV} to compute $\hat{\mathbf{u}}$

- 1: **procedure** INITIALIZE()
- 2: $\mathbf{S} \leftarrow [(\mathbf{S}^{(0)})^\top, (\mathbf{S}^{(1)})^\top, \dots, (\mathbf{S}^{(T-1)})^\top]^\top \in \mathbb{R}^{bT \times n}$
- 3: $\bar{\mathbf{r}} \leftarrow \mathbf{r}^{(0)}$
- 4: Let \mathbf{N} be the matrix given by Lemma C.3 that encodes the matrix formula

$$f(\mathbf{R}, \mathbf{S}, \mathbf{C}, \mathbf{d}) = \mathbf{S} \cdot \mathbf{R}^{1/2} \left(\mathbf{C}(\mathbf{C}^\top \mathbf{R} \mathbf{C})^{-1} \mathbf{C}^\top \mathbf{R} - \mathbf{I} \right) \mathbf{d},$$

i.e., there exist index sets I, J such that $(\mathbf{N}^{-1})_{I,J} = f(\mathbf{R}, \mathbf{S}, \mathbf{C}, \mathbf{d})$. Also let $I_0, I_1, \dots, I_{T-1} \subset I$ each of size b , denote the indexes of the rows corresponding to $\mathbf{S}^{(0)}, \mathbf{S}^{(1)}, \dots, \mathbf{S}^{(T-1)}$.

- 5: $\text{DS.INITIALIZE}(\mathbf{N})$ where DS is the two-level inverse maintenance data structure of Lemma C.2.
 - 6: **procedure** UPDATEQUERY($\bar{\mathbf{r}}^{\text{new}}, i$)
 - 7: $\text{DS.UPDATE}(\Delta)$, where $\Delta = \bar{\mathbf{R}}^{\text{new}} - \bar{\mathbf{R}}$
 - 8: $\bar{\mathbf{r}} \leftarrow \bar{\mathbf{r}}^{\text{new}}$
 - 9: **if** $\text{DS}.k_0 \geq n^{a_0}$ **then**
 - 10: $\text{DS.RESET}()$
 - 11: **else if** $\text{DS}.k_1 \geq n^{a_1}$ **then**
 - 12: $\text{DS.PARTIALRESET}()$
 - 13: **return** $\text{DS.QUERY}(I_i, J)$ $\triangleright |I_i| = b$ and $|J| = 1$
-

In particular, when $\omega = 2 + o(1)$, this time complexity is bounded by

$$\tilde{O}\left(n^{2+1/12}\right) \text{poly}(\epsilon^{-1}).$$

Proof. TOPROVE 6

□

F Guarantees of Algorithm 1: MWU with Monotone Weights

The algorithm in this section has been analysed previously as mentioned in the main text. We include a proof similar to that of [Adi+24] for ℓ_∞ -regression here for completeness. The analysis of Algorithm 1 is based on tracking two potential functions that were defined in Eq. (3) and (4):

$$\begin{aligned} \Phi(\mathbf{w}^{(i,k)}) &\stackrel{\text{def}}{=} \left\| \mathbf{w}^{(i,k)} \right\|_1 \\ \Psi(\bar{\mathbf{r}}^{(i,k)}) &\stackrel{\text{def}}{=} \min_{\Delta \in \mathbb{R}^d} \sum_e \bar{\mathbf{r}}_e^{(i,k)} (\mathbf{C} \Delta - \mathbf{d})_e^2. \end{aligned}$$

Also recall that by Lemma A.1 we always have $\Psi(\bar{\mathbf{r}}^{(i,k)}) \leq e^{\epsilon+\delta} \cdot \Phi(\mathbf{w}^{(i,k)})$. We will show how these potential functions change with a primal step (Line 10) and a width reduction step (Line 14) in Algorithm 1. Finally, to prove our runtime bound, we will first show that if the total number of width reduction steps K is not too large, then Φ is bounded. We then prove that the number of width reduction steps cannot be too large by using the relation between Φ and Ψ and their respective changes throughout the algorithm.

Algorithm 7 Data structures DS_{NORM} to approximately compute ℓ_2 and ℓ_3 norms

- 1: **procedure** INITIALIZE()
- 2: Let $\mathbf{J}^{(0)}, \dots, \mathbf{J}^{(T+K)} \in \mathbb{R}^{O(\epsilon^{-2} \log(n)) \times n}$ be random JL matrices as described in Lemma C.6.
- 3: $\mathbf{J} \leftarrow [(\mathbf{J}^{(0)})^\top, (\mathbf{J}^{(1)})^\top, \dots, (\mathbf{J}^{(T+K)})^\top]^\top \in \mathbb{R}^{O(\epsilon^{-2} \log(n)(T+K)) \times n}$
- 4: Let $\mathbf{U}^{(0)}, \dots, \mathbf{U}^{(T+K)} \in \mathbb{R}^{O(n^{1/3} \log^3(n)) \times n}$ be random matrices as described in Lemma C.5.
- 5: $\mathbf{U} \leftarrow [(\mathbf{U}^{(0)})^\top, (\mathbf{U}^{(1)})^\top, \dots, (\mathbf{U}^{(T+K)})^\top]^\top \in \mathbb{R}^{O(n^{1/3} \log^3(n)(T+K)) \times n}$
- 6: $\bar{\mathbf{r}} \leftarrow \mathbf{r}^{(0)}$
- 7: Let \mathbf{N}_{ℓ_2} and \mathbf{N}_{ℓ_3} be the matrices given by Lemma C.3 that encodes the matrix formulas

$$f_{\ell_2}(\mathbf{R}, \mathbf{J}, \mathbf{C}, \mathbf{d}) = \mathbf{J} \mathbf{R}^{1/2} (\mathbf{C} (\mathbf{C}^\top \mathbf{R} \mathbf{C})^{-1} \mathbf{C}^\top \mathbf{R} - \mathbf{I}) \mathbf{d},$$

$$f_{\ell_3}(\mathbf{R}, \mathbf{U}, \mathbf{C}, \mathbf{d}) = \mathbf{U} \mathbf{R}^{1/3} (\mathbf{C} (\mathbf{C}^\top \mathbf{R} \mathbf{C})^{-1} \mathbf{C}^\top \mathbf{R} - \mathbf{I}) \mathbf{d},$$

i.e., there exist index sets $I_{\ell_2}, J_{\ell_2}, I_{\ell_3}, J_{\ell_3}$ such that $(\mathbf{N}_{\ell_2}^{-1})_{I_{\ell_2}, J_{\ell_2}} = f_{\ell_2}(\mathbf{R}, \mathbf{J}, \mathbf{C}, \mathbf{d})$ and $(\mathbf{N}_{\ell_3}^{-1})_{I_{\ell_3}, J_{\ell_3}} = f_{\ell_3}(\mathbf{R}, \mathbf{U}, \mathbf{C}, \mathbf{d})$. Also let $I_{\ell_2,0}, \dots, I_{\ell_2,T+K} \subset I_{\ell_2}$ denote the index sets of the rows corresponding to $\mathbf{J}^{(0)}, \dots, \mathbf{J}^{(T+K)}$, and let $I_{\ell_3,0}, \dots, I_{\ell_3,T+K} \subset I_{\ell_3}$ denote the rows corresponding to $\mathbf{U}^{(0)}, \dots, \mathbf{U}^{(T+K)}$.

- 8: $\text{DS}_{\ell_2}.\text{INITIALIZE}(\mathbf{N}_{\ell_2})$ and $\text{DS}_{\ell_3}.\text{INITIALIZE}(\mathbf{N}_{\ell_3})$ by Lemma C.2.
 - 9: **procedure** UPDATEQUERY($\bar{\mathbf{r}}^{\text{new}}, i$)
 - 10: $\text{DS}_{\ell_2}.\text{UPDATE}(\Delta)$ and $\text{DS}_{\ell_3}.\text{UPDATE}(\Delta)$, where $\Delta = \bar{\mathbf{R}}^{\text{new}} - \bar{\mathbf{R}}$
 - 11: $\bar{\mathbf{r}} \leftarrow \bar{\mathbf{r}}^{\text{new}}$
 - 12: **if** $\text{DS}_{\ell_2}.k_0 \geq n^{a_0}$ **then**
 - 13: $\text{DS}_{\ell_2}.\text{RESET}()$ and $\text{DS}_{\ell_3}.\text{RESET}()$
 - 14: **else if** $\text{DS}_{\ell_2}.k_1 \geq n^{a_1}$ **then**
 - 15: $\text{DS}_{\ell_2}.\text{PARTIALRESET}()$ and $\text{DS}_{\ell_3}.\text{PARTIALRESET}()$
 - 16: **return** $(\|\text{DS}_{\ell_2}.\text{QUERY}(I_{\ell_2,i}, J_{\ell_2})\|_2^2, \|\text{DS}_{\ell_3}.\text{QUERY}(I_{\ell_3,i}, J_{\ell_3})\|_\infty^3)$
-

Algorithm 8 Implicit inverse maintenance data structure $\text{DS}_{\text{IMPLICITINV}}$ to compute Δ and to update \mathbf{x}

- 1: **procedure** INITIALIZE()
 - 2: $\bar{\mathbf{r}} \leftarrow \mathbf{r}^{(0)}$
 - 3: Let \mathbf{N} be the matrix given by Lemma C.3 that encodes the matrix formula $f(\mathbf{R}, \mathbf{C}) = (\mathbf{C}^\top \mathbf{R} \mathbf{C})^{-1} \mathbf{C}^\top \mathbf{R}$, i.e., there exist index sets I, J such that $(\mathbf{N}^{-1})_{I,J} = f(\mathbf{R}, \mathbf{C})$.
 - 4: $\text{DS}.\text{INITIALIZE}(\mathbf{N}, \mathbf{d}')$ where DS is the implicit inverse maintenance data structure of Lemma C.4, and \mathbf{d}' has the same size as \mathbf{N} , and it equals to \mathbf{d} in J , and its other coordinates are all zero.
 - 5: **procedure** UPDATE($\bar{\mathbf{r}}^{\text{new}}$)
 - 6: $\text{DS}.\text{UPDATE}(\Delta)$, where $\Delta = \bar{\mathbf{R}}^{\text{new}} - \bar{\mathbf{R}}$
 - 7: $\bar{\mathbf{r}} \leftarrow \bar{\mathbf{r}}^{\text{new}}$
 - 8: **if** $\text{DS}.k_0 \geq n^{a_0}$ **then**
 - 9: $\text{DS}.\text{RESET}()$
 - 10: **else if** $\text{DS}.k_1 \geq n^{a_1}$ **then**
 - 11: $\text{DS}.\text{PARTIALRESET}()$
 - 12: **procedure** QUERYSUM()
 - 13: **return** $\text{DS}.\text{QUERYSUM}()$
-

Algorithm 9 Heavy hitter data structure $\text{DS}_{\text{HEAVYHITTERS}}$ to compute the heavy entries of \mathbf{u}

- 1: **procedure** INITIALIZE()
- 2: $\epsilon_{\text{heavy}} \leftarrow \frac{\rho\sqrt{\epsilon}}{2C_3\sqrt{n}}$
- 3: $\bar{\mathbf{r}} \leftarrow \mathbf{r}^{(0)}$
- 4: Let $\Phi \in \mathbb{R}^{O(\epsilon_{\text{heavy}}^{-2} \log^2 n) \times n}$ be the random matrix as described in Lemma C.7.
- 5: Let \mathbf{N} and \mathbf{N}_Φ be the matrix given by Lemma C.3 that encodes the matrix formulas

$$f(\mathbf{R}, \mathbf{C}, \mathbf{d}) = \mathbf{R}^{1/2}(\mathbf{C}(\mathbf{C}^\top \mathbf{R} \mathbf{C})^{-1} \mathbf{C}^\top \mathbf{R} - \mathbf{I})\mathbf{d},$$

$$f_\Phi(\Phi, \mathbf{R}, \mathbf{C}, \mathbf{d}) = \Phi \cdot \mathbf{R}^{1/2}(\mathbf{C}(\mathbf{C}^\top \mathbf{R} \mathbf{C})^{-1} \mathbf{C}^\top \mathbf{R} - \mathbf{I})\mathbf{d},$$

i.e., there exist index sets I, J, I_Φ, J_Φ such that $(\mathbf{N}^{-1})_{I,J} = f(\mathbf{R}, \mathbf{C}, \mathbf{d})$ and $(\mathbf{N}_\Phi^{-1})_{I_\Phi, J_\Phi} = f_\Phi(\Phi, \mathbf{R}, \mathbf{C}, \mathbf{d})$.

- 6: $\text{DS}.\text{INITIALIZE}(\mathbf{N})$, $\text{DS}_\Phi.\text{INITIALIZE}(\mathbf{N}_\Phi)$, where DS and DS_Φ are both the inverse maintenance data structure of Lemma C.2.
 - 7: **procedure** UPDATEQUERY($\bar{\mathbf{r}}^{\text{new}}$)
 - 8: $\text{DS}.\text{UPDATE}(\Delta)$ and $\text{DS}_\Phi.\text{UPDATE}(\Delta)$, where $\Delta = \bar{\mathbf{R}}^{\text{new}} - \bar{\mathbf{R}}$
 - 9: $\bar{\mathbf{r}} \leftarrow \bar{\mathbf{r}}^{\text{new}}$
 - 10: **if** $\text{DS}.k_0 \geq n^{a_0}$ **then**
 - 11: $\text{DS}.\text{RESET}()$ and $\text{DS}_\Phi.\text{RESET}()$
 - 12: **else if** $\text{DS}.k_1 \geq n^{a_1}$ **then**
 - 13: $\text{DS}.\text{PARTIALRESET}()$ and $\text{DS}_\Phi.\text{PARTIALRESET}()$
 - 14: $\mathbf{y} \leftarrow \text{DS}_\Phi.\text{QUERY}(I_\Phi, J_\Phi)$
 - 15: $L \leftarrow \text{DECODE}(\mathbf{y})$, where $\text{DECODE}()$ is the decoding algorithm of Lemma C.7. We can view $L \subset [n]$ as a subset of I .
 - 16: **return** $(L, \text{DS}.\text{QUERY}(L, J))$
-

Convergence Analysis

Change in Φ

Lemma F.1. *After i primal steps, and k width-reduction steps, the potential Φ is bounded as follows:*

$$\Phi(\mathbf{w}^{(i,k)}) \leq \Phi(\mathbf{w}^{(0,0)}) \left(1 + e^{\epsilon+\delta} \cdot \epsilon\alpha\right)^i \left(1 + e^{\epsilon+\delta} \cdot \frac{\epsilon}{\tau}\right)^k.$$

Proof. **TOPROVE 7** □

Change in Ψ

We now prove how the Ψ potential changes throughout the algorithm.

Lemma F.2. *After i primal steps and k width reduction steps, if $\delta \leq \epsilon/6$,*

$$\Psi(\bar{\mathbf{r}}^{(i,k)}) \geq \Psi(\mathbf{r}^{(0,0)}) \left(1 + \frac{\epsilon^2 \tau^2}{4n}\right)^k.$$

Proof. **TOPROVE 8** □

Proof of Theorem 2.1

Proof. **TOPROVE 9** □

G Guarantees of Algorithm 3: Robust Primal Step and Stable Width Reduction Step

G.1 Starting Point: Non-Monotone MWU Algorithm

We first present the starting point from where we developed our final algorithm. We extend the algorithm in [Mad16] to general ℓ_∞ -regression. This gives an algorithm which converges in $\tilde{O}(n^{1/3} \text{poly}(\epsilon^{-1}))$ iterations and has weights that are not monotonically increasing. The algorithm is presented as Algorithm 10 and we have moved the analysis to the end in Appendix J since we do not really use this algorithm directly. The analysis is just kept for completeness. In particular we prove:

Theorem G.1. *Let $\eta \leq 1/6$. There is an algorithm that does not update the weights monotonically (Algorithm 10, and with input $\begin{bmatrix} \mathbf{C} \\ -\mathbf{C} \end{bmatrix}, \begin{bmatrix} \mathbf{d} \\ -\mathbf{d} \end{bmatrix}, \epsilon$) returns $\hat{\mathbf{x}}$ such that $\|\mathbf{C}\hat{\mathbf{x}} - \mathbf{d}\|_\infty \leq 1 + O(\epsilon)$ in at most $\tilde{O}\left((n^{1/2-\eta} + n^{2\eta})\epsilon^{-7/3}\right)$ iterations. Each iteration solves a system of linear equations.*

Let us consider the width reduction steps. Since $\mathbf{r}_e \geq \frac{\epsilon}{2n} \Phi(\mathbf{w}) \geq \frac{\epsilon}{2n} \Psi(\mathbf{w})$, from the condition of the width steps we get that,

$$\frac{\epsilon}{2n} \Psi(\mathbf{r}) |H| \leq \tau^{-1} \Psi(\mathbf{r}) \Rightarrow |H| \leq O(n^{4\eta}).$$

The above calculations imply that for $\eta = 1/6$, the algorithm in the worst case can update $\approx n^{2/3}$ coordinates in every iteration. There are a total of $\approx n^{1/3}$ iterations. At an intuitive level, if we change the value of \mathbf{r} so many coordinates per iteration, and that too by a factor of ϵ , then the entire idea of doing a lazy update is moot. Therefore, we require a new kind of width reduction steps that can schedule these steps in a way amenable to a lazy update schedule, which is what we do in the next section.

Algorithm 10 Accelerated MWU algorithm with non-monotone weights

```

1: procedure MWU-NONMONOTONE( $\tilde{\mathbf{C}}, \tilde{\mathbf{d}}, \epsilon$ )
2:    $\mathbf{w}^{(0,0)} \leftarrow \mathbf{1}_n, \quad \mathbf{x}^{(0,0)} \leftarrow \mathbf{0}_d$ 
3:    $\alpha \leftarrow \tilde{\Theta}\left(n^{-1/2+\eta}\epsilon^{1/3}\right), \alpha_+ \leftarrow \alpha, \alpha_- \leftarrow \alpha/(1+2\epsilon)$ 
4:    $\tau \leftarrow \tilde{\Theta}\left(n^{1-4\eta}\epsilon^{-1/3}\right), \quad \rho \leftarrow \tilde{\Theta}\left(n^{1/2-3\eta}\right)$ 
5:    $T \leftarrow \alpha^{-1} \ln \frac{n}{\Psi_0}/\epsilon^2$ 
6:    $i, k = 0$ 
7:   while  $i < T$  do
8:      $\mathbf{r}^{(i,k)} \leftarrow \mathbf{w}^{(i,k)} + \frac{\epsilon}{2n} \sum_e \mathbf{w}_e^{(i,k)}$ 
9:      $\Delta^{(i,k)} \leftarrow \arg \min_{\Delta} \sum_e \mathbf{r}_e^{(i,k)} (\tilde{\mathbf{C}}\Delta - \tilde{\mathbf{d}})_e^2 \quad \triangleright \Delta = (\tilde{\mathbf{C}}^\top \mathbf{R}^{(i,k)} \tilde{\mathbf{C}})^{-1} \tilde{\mathbf{C}}^\top \mathbf{R}^{(i,k)} \tilde{\mathbf{d}}$ 
10:     $\Psi(\mathbf{r}^{(i,k)}) \leftarrow \sum_e \mathbf{r}_e^{(i,k)} (\tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}})_e^2$ 
11:    if  $\sum_e \mathbf{r}_e^{(i,k)} |\tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}}|_e^3 \leq 2\rho\Psi(\mathbf{r}^{(i,k)})$  then  $\triangleright$  primal step
12:       $\vec{\alpha}_e^{(i,k)} = \begin{cases} \alpha_+ & \text{if } (\tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}})_e \geq 0 \\ \alpha_- & \text{else} \end{cases}$ 
13:       $\mathbf{w}^{(i+1,k)} \leftarrow \mathbf{w}^{(i,k)} \left(1 + \epsilon \vec{\alpha}^{(i,k)} (\tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}})\right)$ 
14:       $\mathbf{x}^{(i+1)} \leftarrow \mathbf{x}^{(i)} + \Delta^{(i,k)}$ 
15:       $i \leftarrow i + 1$ 
16:    else  $\triangleright$  width reduction step
17:      Let  $S$  be the set of coordinates  $e$  such that  $|\tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}}|_e \geq \rho$ 
18:       $H \subseteq S$  be maximal subset such that  $\sum_{e \in H} \mathbf{r}_e^{(i,k)} \leq \tau^{-1}\Psi(\mathbf{r}^{(i,k)})$ 
19:      For all  $e \in H$ ,  $\mathbf{w}_e^{(i,k+1)} \leftarrow (1 + \epsilon)\mathbf{w}_e^{(i,k)} + \frac{\epsilon^2}{n}\Phi(\mathbf{w}^{(i,k)})$ 
20:      If  $H \neq S$ , for one  $\bar{e} \in S \setminus H$ , let  $\gamma = \min\{1, \frac{\tau^{-1}}{\mathbf{r}_{\bar{e}}^{(i,k)}}\Psi(\mathbf{r}^{(i,k)})\}$ 
21:       $\mathbf{w}_{\bar{e}}^{(i,k+1)} \leftarrow (1 + \epsilon\gamma)\mathbf{w}_{\bar{e}}^{(i,k)} + \frac{\epsilon^2\gamma}{n}\Phi(\mathbf{w}^{(i,k)})$ 
22:       $k \leftarrow k + 1$ 
23:  return  $\mathbf{x}^{(T)}/T$ 

```

G.2 Warm Up: Stable Width Reduction Step

As a warm-up, we first analyze the required stable algorithm, but do not introduce any sketching. In the next section, we will consider the changes to the analysis from this section which occur due to the introduction of sketching. We encourage the reader to go through this section in order to understand how our final algorithm works.

In this section we define $\tilde{\mathbf{C}} = \begin{bmatrix} \mathbf{C} \\ -\mathbf{C} \end{bmatrix}$ and $\tilde{\mathbf{d}} = \begin{bmatrix} \mathbf{d} \\ -\mathbf{d} \end{bmatrix}$. Our goal is to prove the following result. We will focus on the convergence and runtime in this section and defer the stability guarantees of the algorithm to Appendix H.

Theorem G.2. *For $\eta \leq 1/10$, Algorithm 11 with input $(\begin{bmatrix} \mathbf{C} \\ -\mathbf{C} \end{bmatrix}, \begin{bmatrix} \mathbf{d} \\ -\mathbf{d} \end{bmatrix}, \epsilon)$ finds $\hat{\mathbf{x}} \in \mathbb{R}^d$ such that $\|\mathbf{C}\hat{\mathbf{x}} - \mathbf{d}\|_\infty \leq 1 + O(\epsilon)$ in at most $T + K \leq \tilde{O}(n^{1/2-\eta}\epsilon^{-4})$ iterations. Furthermore, the algorithm satisfies the following additional guarantees:*

1. *In the width reduction step of the algorithm, the algorithm only requires to find at most $\tilde{O}(n^{1/2+\eta})$ large coordinates per iteration.*
2. *The algorithm satisfies the following low-rank update scheme: There are at most $\frac{T+K}{2^\ell}$ number of iterations where $\bar{\mathbf{r}}$ receives an update of rank $\tilde{O}_\epsilon(n^{1/5}2^{2\ell})$.*

We first define some notation and basic properties.

G.2.1 Definitions and Basic Properties

Potentials. Recall that we had defined the two potentials of interest as,

$$\Phi(\mathbf{w}) = \|\mathbf{w}\|_1, \quad \Psi(\mathbf{r}) = \min_{\Delta \in \mathbb{R}^d} \sum_{e=1}^{2n} \mathbf{r}_e (\mathbf{C}\Delta - \mathbf{d})_e^2.$$

Also recall that by Lemma A.1, for $\mathbf{r} = \mathbf{w} + \frac{\epsilon}{2n}\|\mathbf{w}\|_1$, and $\bar{\mathbf{r}} \approx_\delta \mathbf{r}$, we always have $\Psi(\bar{\mathbf{r}}) \approx_\delta \Psi(\mathbf{r})$, and $\Psi(\bar{\mathbf{r}}) \leq e^{\epsilon+\delta}\Phi(\mathbf{w})$.

Lazy updates for primal steps. Recall that in Algorithm 2, for any primal step i and any $e \in [2n]$, we defined $\text{LASTWIDTH}(i, e)$ to be the largest $i' \leq i$ such that the algorithm executed a width reduction step from (i', k) to $(i', k+1)$ during which the weight of e is updated, i.e., $\mathbf{w}_e^{(i', k+1)} \neq \mathbf{w}_e^{(i', k)}$.

For any primal step i and any $e \in [2n]$, we also define $\text{LAST}(i, e)$ to be the largest $i'' \leq i$ such that $\bar{\mathbf{r}}_e$ was updated by SELECTVECTOR in primal step $i'' - 1$, i.e., $\bar{\mathbf{r}}_e^{(i'', k_{i''-1})} \neq \bar{\mathbf{r}}_e^{(i''-1, k_{i''-1})}$.

For any primal step i , we define $S_i \subseteq [2n]$ to be the set of coordinates that are being updated by SELECTVECTOR in the i -th primal step.

Finally, for any primal step i and any $e \in S_i$, we also define $\ell_{i,e}$ to be the smallest integer ℓ such that $i \equiv 0 \pmod{2^\ell}$ and $\left| \ln \left(\frac{\mathbf{r}_e^{(i)}}{\mathbf{r}_e^{(i-2^\ell)}} \right) \right| \geq \frac{\delta}{2 \log n}$.

Remark: We note that $\mathbf{r} = \mathbf{w} + \frac{\epsilon}{2n}\Phi(\mathbf{w})$. Now in our lazy update scheme, \mathbf{r} can also change due to changes in $\Phi(\mathbf{w})$. We claim that we do not need to consider the changes in \mathbf{r} due to the change in Φ in the lazy update scheme. This is because, the change in Φ contributes enough

Algorithm 11 Accelerated MWU algorithm with non-monotone weights and stable steps

```

1: procedure MWU-NONMONOTONESTABLE( $\tilde{\mathbf{C}}, \tilde{\mathbf{d}}, \epsilon$ )
2:    $\mathbf{w}^{(0,0)} \leftarrow \mathbf{1}_{2n}$ ,  $\bar{\mathbf{r}}^{(0,0)} \leftarrow \mathbf{r}^{(0,0)} \leftarrow (1 + \epsilon)\mathbf{1}_{2n}$ ,  $\mathbf{x}^{(0)} \leftarrow \mathbf{0}_d$ 
3:    $\alpha \leftarrow n^{-1/2+\eta} \cdot \epsilon \cdot \log(n)^{-4/3} \log(\frac{n}{\Psi_0})^{-1/3}/10$ ,  $\alpha_+ \leftarrow \alpha$ ,  $\alpha_- \leftarrow \alpha/(1 + 2\epsilon)$ 
4:    $\tau \leftarrow n^{1/2-\eta} \cdot \epsilon^{-4} \cdot \log(n)^8 \log(\frac{n}{\Psi_0})^2$ ,  $\rho \leftarrow n^{1/2-3\eta} \cdot \epsilon^{-2} \cdot \log(n)^4 \log(\frac{n}{\Psi_0})$ ,  $\delta \leftarrow \frac{\epsilon}{100}$ 
5:    $T \leftarrow \alpha^{-1} \epsilon^{-2} \log n$ 
6:    $i, k = 0$ 
7:   while  $i < T$  do
8:      $\Delta^{(i,k)} \leftarrow \arg \min_{\Delta} \sum_e \bar{\mathbf{r}}_e^{(i,k)} (\tilde{\mathbf{C}}\Delta - \tilde{\mathbf{d}})_e^2$   $\triangleright \bar{\mathbf{r}} \approx_{\delta} \mathbf{r}$ 
9:      $\Psi(\bar{\mathbf{r}}^{(i,k)}) \leftarrow \sum_e \bar{\mathbf{r}}_e^{(i,k)} (\tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}})_e^2$ 
10:    if  $\sum_e \bar{\mathbf{r}}_e^{(i,k)} |\tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}}|_e^3 \leq 2\rho\Psi(\bar{\mathbf{r}}^{(i,k)})$  then  $\triangleright$  primal step
11:       $\vec{\alpha}_e^{(i,k)} = \begin{cases} \alpha_+ & \text{if } (\tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}})_e \geq 0 \\ \alpha_- & \text{else} \end{cases}$ 
12:       $\mathbf{w}^{(i+1,k)} \leftarrow \mathbf{w}^{(i,k)} \left(1 + \epsilon \vec{\alpha}^{(i,k)} (\tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}})\right)$ 
13:       $\mathbf{r}^{(i+1,k)} \leftarrow \mathbf{w}^{(i+1,k)} + \frac{\epsilon}{2n} \sum_e \mathbf{w}_e^{(i+1,k)}$ 
14:       $\bar{\mathbf{r}}^{(i+1,k)} \leftarrow \text{SELECTVECTOR}(\mathbf{r}^{(i+1,k)}, i + 1, \delta)$   $\triangleright$  Algorithm 2
15:       $\mathbf{x}^{(i+1)} \leftarrow \mathbf{x}^{(i)} + \Delta^{(i,k)}$ 
16:       $i \leftarrow i + 1$ 
17:    else  $\triangleright$  width reduction step
18:      Let  $S$  be the set of coordinates  $e$  such that  $|\tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}}|_e \geq \rho$ 
19:       $H \subseteq S$  be maximal subset such that  $\sum_{e \in H} \bar{\mathbf{r}}_e^{(i,k)} \leq \tau^{-1}\Psi(\bar{\mathbf{r}}^{(i,k)})$ 
20:      if  $H \neq S$  then
21:        Pick any  $\bar{e} \in S \setminus H$ .
22:        For all  $e \in H \cup \{\bar{e}\}$ ,  $\mathbf{w}_e^{(i,k+1)} \leftarrow (1 + \epsilon)\mathbf{w}_e^{(i,k)} + \frac{\epsilon^2}{2n}\Phi(\mathbf{w}^{(i,k)})$ 
23:         $\mathbf{r}^{(i,k+1)} \leftarrow \mathbf{w}^{(i,k+1)} + \frac{\epsilon}{2n}\Phi(\mathbf{w}^{(i,k+1)})$ 
24:        For all  $e \in H \cup \{\bar{e}\}$ ,  $\bar{\mathbf{r}}_e^{(i,k+1)} \leftarrow \mathbf{r}_e^{(i,k+1)}$ 
25:      else
26:        for  $\zeta = \rho, 2\rho, 4\rho, \dots, 2^{c_\rho}\rho$  do
27:           $\triangleright c_\rho$  is defined to be the smallest integer  $c$  that satisfies  $2^c\rho \geq \sqrt{n/\epsilon}$ 
28:          Define the set  $H_\zeta = \{e \in H \mid |\tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}}|_e \in [\zeta, 2\zeta)\}$ .
29:          If  $\sum_{e \in H_\zeta} \bar{\mathbf{r}}_e^{(i,k)} |\tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}}|_e^3 \geq \frac{\rho\Psi(\bar{\mathbf{r}}^{(i,k)})}{\log(\frac{n}{\epsilon\rho})}$ , set  $\zeta^* \leftarrow \zeta$ , and break.
30:          For all  $e \in H_{\zeta^*}$ ,  $\mathbf{w}_e^{(i,k+1)} \leftarrow (1 + \epsilon)\mathbf{w}_e^{(i,k)} + \frac{\epsilon^2}{2n}\Phi(\mathbf{w}^{(i,k)})$ 
31:           $\mathbf{r}^{(i,k+1)} \leftarrow \mathbf{w}^{(i,k+1)} + \frac{\epsilon}{2n}\Phi(\mathbf{w}^{(i,k+1)})$ 
32:          For all  $e \in H_{\zeta^*}$ ,  $\bar{\mathbf{r}}_e^{(i,k+1)} \leftarrow \mathbf{r}_e^{(i,k+1)}$ 
33:         $k \leftarrow k + 1$ 
34:    return  $\mathbf{x}^{(T)}/T$ 

```

only when the change is $\approx n$ which can happen only $\tilde{O}_\epsilon(1)$ times (Refer to Lemma G.3), or once every $\tilde{O}_\epsilon(\alpha^{-1})$ iterations, and in such cases, the algorithm can reset the values of \mathbf{r} for all coordinates.

Sizes of width reduction steps. We say a width reduction step has size s if it updates the weight of s coordinates, and we denote the size of the k -th width reduction step as $\text{SIZE}(k)$. For example, if in the k -th width reduction step on Line 20 of Algorithm 11 we have that $H \neq S$, then we have $\text{SIZE}(k) = |H| + 1$ by Line 22 of Algorithm 11.

G.2.2 Change in Φ

Lemma G.3 (Change in Φ for Algorithm 11). *After i primal steps, and k width-reduction steps, if $\alpha\rho^{1/3} \leq \frac{\epsilon^{1/3}}{10n^{1/3}}$, the potential Φ is bounded as follows:*

$$\Phi(\mathbf{w}^{(i,k)}) \leq \Phi(\mathbf{w}^{(0,0)}) \cdot \left(1 + \epsilon\alpha_+ e^{\epsilon+\delta}\right)^i \cdot \left(1 + \epsilon e^{\epsilon+2\delta} \cdot (\tau^{-1} + \rho^{-2})\right)^k.$$

Furthermore, after every primal step, the potential can decrease by at most,

$$\Phi(\mathbf{w}^{(i+1,k)}) \geq \Phi(\mathbf{w}^{(i,k)}) \left(1 - \epsilon\alpha_+ e^{\epsilon+\delta}\right).$$

Proof. **TOPROVE 10** □

G.2.3 Change in Ψ

Lemma G.4 (Bound on $|H|$ in width reduction steps). *For any width reduction step, the size of H satisfies $|H| \leq \frac{n}{\tau\epsilon} \cdot e^{\epsilon+2\delta}$.*

Proof. **TOPROVE 11** □

Let L be the largest power of 2 such that $L \leq \frac{1}{100(\log^4 n)\epsilon\alpha\rho}$. Note that we have $L = \Theta(\frac{1}{(\log^4 n)\epsilon\alpha\rho})$. We have the following lemma.

Lemma G.5 (Change in Ψ for Algorithm 11). *For any integer $c \geq 0$, after L primal steps from $(c-1)L$ to cL , if $\rho^2\tau^{-1} \geq 0.1$, the potential Ψ is bounded as follows:*

$$\Psi(\bar{\mathbf{r}}^{(cL,k_{cL})}) \geq \Psi(\bar{\mathbf{r}}^{((c-1)L,k_{(c-1)L})}) \cdot \left(1 - \tilde{O}(\epsilon\alpha\rho L)\right) \cdot \prod_{k=k_{(c-1)L}}^{k_{cL}} \left(1 + O\left(\frac{\epsilon^{4/3}\rho^{2/3} \cdot \text{SIZE}(k)^{1/3}}{n^{1/3} \cdot \log^{2/3}(\frac{n}{\epsilon\rho})}\right)\right).$$

Proof. **TOPROVE 12** □

G.2.4 Putting Everything Together: Analysis of Algorithm

Next we analyze the iteration complexity and the error of Algorithm 11. We first bound the total number of width reduction steps. In the following lemma we denote the hidden factors in Lemma G.5 as $C_2 \leq O(\log^3 n)$ and $C_3 \geq O(1)$ such that

$$\Psi(\bar{\mathbf{r}}^{(cL,k_{cL})}) \geq \Psi(\bar{\mathbf{r}}^{((c-1)L,k_{(c-1)L})}) \cdot \left(1 - C_2\epsilon\alpha\rho L\right) \cdot \prod_{k=k_{(c-1)L}}^{k_{cL}} \left(1 + C_3 \frac{\epsilon^{4/3}\rho^{2/3} \cdot \text{SIZE}(k)^{1/3}}{n^{1/3} \cdot \log^{2/3}(\frac{n}{\epsilon\rho})}\right).$$

Lemma G.6 (Number of width reduction steps). *The total number of width reduction steps of Algorithm 11 is at most $O\left(\frac{n^{1/3}\rho^{1/3}\log^5 n}{\epsilon^{10/3}} \cdot \log\left(\frac{n}{\Psi_0}\right)\right)$ for large enough n .*

Proof. **TOPROVE 13** □

Next we bound the error of Algorithm 11.

Lemma G.7 (Error of Algorithm 11). *Algorithm 11 outputs a vector $\hat{\mathbf{x}} \in \mathbb{R}^d$ such that $\|\mathbf{C}\hat{\mathbf{x}} - \mathbf{d}\|_\infty \leq 1 + O(\epsilon)$.*

Proof. **TOPROVE 14** □

G.3 Guarantees of Algorithm 3: Robust Primal Step

In this section, we finally present the analysis of Algorithm 3 which additionally approximates the primal steps via a *sketch*. We will again use the two potentials as defined in Eq. (3) and (4).

In the next section, we will first present the properties of the sketching matrices that we need to use. This constitutes the major part of the section. The remaining would directly build on the analysis of the previous section.

G.3.1 Sketching Bounds

Lemma G.8 (Coordinate-wise embedding, Lemma E.5 of [LSZ19]). *Let $\mathbf{S} \in \mathbb{R}^{b \times n}$ be sampled from distribution Π such that each entry is $+\frac{1}{\sqrt{b}}$ with probability $1/2$ and $-\frac{1}{\sqrt{b}}$ with probability $1/2$. For any fixed vectors $\mathbf{g}, \mathbf{h} \in \mathbb{R}^n$, the following properties hold:*

1. $\mathbb{E}_{\mathbf{S} \sim \Pi} [\mathbf{g}^\top \mathbf{S}^\top \mathbf{S} \mathbf{h}] = \mathbf{g}^\top \mathbf{h},$
2. $\mathbb{E}_{\mathbf{S} \sim \Pi} [(\mathbf{g}^\top \mathbf{S}^\top \mathbf{S} \mathbf{h})^2] \leq (\mathbf{g}^\top \mathbf{h})^2 + \frac{C_1}{b} \|\mathbf{g}\|_2^2 \|\mathbf{h}\|_2^2,$
3. $\Pr_{\mathbf{S} \sim \Pi} \left[|\mathbf{g}^\top \mathbf{S}^\top \mathbf{S} \mathbf{h} - \mathbf{g}^\top \mathbf{h}| \leq \frac{C_2}{\sqrt{b}} \|\mathbf{g}\|_2 \|\mathbf{h}\|_2 \right] \geq 1 - 1/n^4.$

where $C_1 = O(1)$, $C_2 = O(\log n)$.

Lemma G.9 (Bounds for the vector $\hat{\mathbf{u}}$). *For all $i \in [0 : T]$, the vector $\hat{\mathbf{u}}^{(i,k)} = (\bar{\mathbf{R}}^{(i,k)})^{-1/2} \cdot (\mathbf{S}^{(i)})^\top \mathbf{S}^{(i)} \cdot (\bar{\mathbf{R}}^{(i,k)})^{1/2} \mathbf{u}^{(i,k)}$ satisfies the following properties:*

1. **Expectation.** $\mathbb{E}_{\mathbf{S}^{(i)}} [\hat{\mathbf{u}}^{(i,k)} \mid \mathbf{S}^{(0)}, \dots, \mathbf{S}^{(i-1)}] = \mathbf{u}^{(i,k)}.$
2. **Variance.** *For any vector $\mathbf{g} \in \mathbb{R}^n$ that is independent of $\mathbf{S}^{(i)}$,*

$$\text{Var}_{\mathbf{S}^{(i)}} \left[\sum_e \mathbf{g}_e \hat{\mathbf{u}}_e^{(i,k)} \mid \mathbf{S}^{(0)}, \dots, \mathbf{S}^{(i-1)} \right] \leq \frac{C_1}{b} \cdot \mathbf{g}^\top (\bar{\mathbf{R}}^{(i,k)})^{-1} \mathbf{g} \cdot (\mathbf{u}^{(i,k)})^\top \bar{\mathbf{R}}^{(i,k)} \mathbf{u}^{(i,k)}.$$

In particular, if $\mathbf{w}^{(i,k)} \geq 0$, then this implies that:

- $\text{Var}_{\mathbf{S}^{(i)}} \left[\hat{\mathbf{u}}_e^{(i,k)} \mid \mathbf{S}^{(0)}, \dots, \mathbf{S}^{(i-1)} \right] \leq \frac{C_1 n}{eb}.$
- $\mathbb{E}_{\mathbf{S}^{(i)}} [\hat{\Psi}(\bar{\mathbf{r}}^{(i,k)}, \mathbf{S}^{(i)}) \mid \mathbf{S}^{(0)}, \dots, \mathbf{S}^{(i-1)}] \leq (1 + \frac{C_1 n}{b}) \cdot \Psi(\bar{\mathbf{r}}^{(i,k)}).$
- $\text{Var}_{\mathbf{S}^{(i)}} \left[\sum_e \mathbf{w}_e^{(i,k)} \hat{\mathbf{u}}_e^{(i,k)} \mid \mathbf{S}^{(0)}, \dots, \mathbf{S}^{(i-1)} \right] \leq \frac{C_1}{b} \cdot \Phi(\mathbf{w}^{(i,k)}) \cdot \Psi(\bar{\mathbf{r}}^{(i,k)}).$

3. **Coordinate-wise absolute value.** For any vector $\mathbf{g} \in \mathbb{R}^n$ that is independent of $\mathbf{S}^{(i)}$, and when conditioned on any $\mathbf{S}^{(0)}, \dots, \mathbf{S}^{(i-1)}$, we have

$$\Pr_{\mathbf{S}^{(i)}} \left[\left| \sum_e \mathbf{g}_e (\hat{\mathbf{u}}_e^{(i,k)} - \mathbf{u}_e^{(i,k)}) \right| \leq \frac{C_2}{\sqrt{b}} \cdot (\mathbf{g}^\top (\bar{\mathbf{R}}^{(i,k)})^{-1} \mathbf{g})^{1/2} \cdot \Psi(\bar{\mathbf{r}}^{(i,k)})^{1/2} \right] \geq 1 - 1/n^4.$$

In particular, this implies that when conditioned on any $\mathbf{S}^{(0)}, \dots, \mathbf{S}^{(i-1)}$:

- For any $e \in [n]$, if $\mathbf{w}_e^{(i,k)} \geq 0$, then

$$\Pr_{\mathbf{S}^{(i)}} \left[\left| \hat{\mathbf{u}}_e^{(i,k)} - \mathbf{u}_e^{(i,k)} \right| \leq \frac{C_2}{\sqrt{b}} \cdot \frac{\sqrt{n}}{\sqrt{\epsilon}} \cdot \Phi(\mathbf{w}^{(i,k)})^{-1/2} \cdot \Psi(\bar{\mathbf{r}}^{(i,k)})^{1/2} \right] \geq 1 - 1/n^4.$$

- If $\mathbf{w}^{(i,k)} \geq 0$, then

$$\Pr_{\mathbf{S}^{(i)}} \left[\left| \sum_e \mathbf{w}_e^{(i,k)} \cdot (\hat{\mathbf{u}}_e^{(i,k)} - \mathbf{u}_e^{(i,k)}) \right| \leq \frac{C_2}{\sqrt{b}} \cdot \Phi(\mathbf{w}^{(i,k)})^{1/2} \cdot \Psi(\bar{\mathbf{r}}^{(i,k)})^{1/2} \right] \geq 1 - 1/n^4,$$

$$\Pr_{\mathbf{S}^{(i)}} \left[\left| \sum_e \bar{\mathbf{r}}_e^{(i,k)} \cdot (\hat{\mathbf{u}}_e^{(i,k)} - \mathbf{u}_e^{(i,k)}) \right| \leq \frac{C_2(1+\epsilon)}{\sqrt{b}} \cdot \Phi(\mathbf{w}^{(i,k)})^{1/2} \cdot \Psi(\bar{\mathbf{r}}^{(i,k)})^{1/2} \right] \geq 1 - 1/n^4.$$

4. **Symmetry.** When conditioned on any $\mathbf{S}^{(0)}, \dots, \mathbf{S}^{(i-1)}$, for any $i \in [T]$ and any $e \in [n]$, the distribution of $\hat{\mathbf{u}}_e^{(i,k)} - \mathbf{u}_e^{(i,k)}$ is symmetric around zero, i.e., for any $z \in \mathbb{R}$,

$$\Pr_{\mathbf{S}^{(i)}} [\hat{\mathbf{u}}_e^{(i,k)} - \mathbf{u}_e^{(i,k)} = z] = \Pr_{\mathbf{S}^{(i)}} [\hat{\mathbf{u}}_e^{(i,k)} - \mathbf{u}_e^{(i,k)} = -z].$$

Proof. **TOPROVE 15** □

The variance and coordinate-wise bounds of the previous lemma only hold when the weights are non negative. Next we prove that we can maintain this non negativity as long as $|\hat{\mathbf{u}}_e^{(i,k)} - \mathbf{u}_e^{(i,k)}|$ is bounded.

Lemma G.10 (Positivity of the weights). *Let k_i denote the number of width reduction steps taken by the algorithm when the i^{th} primal step is being executed. For all $i \in [0 : T]$, in the i -th primal iteration of Algorithm 3, if*

$$\mathbf{w}^{(i,k_i)} \geq 0, \quad \text{and} \quad |\hat{\mathbf{u}}^{(i,k_i)} - \mathbf{u}^{(i,k_i)}| \leq \frac{100C_2\sqrt{n}}{\sqrt{b\epsilon}},$$

then we have

$$\|\mathbf{u}^{(i,k_i)}\|_\infty \leq 2C_3^{1/3} \frac{n^{1/2-\eta}}{\epsilon^{1/3}}, \quad |\vec{\alpha}^{(i,k_i)} \hat{\mathbf{u}}^{(i,k_i)}| \leq 1/10, \quad \text{and} \quad \mathbf{w}^{(i+1,k_i)} \geq 0.$$

Proof. **TOPROVE 16** □

Next we use the above basic properties of the approximate vectors $\hat{\mathbf{u}}^{(i,k)}$'s to prove a concentration property of the sum of the $\hat{\mathbf{u}}^{(i,k)}$'s over all iterations. Our proof crucially uses the following concentration inequality of martingales:

Lemma G.11 (Freedman's inequality, [Fre75]). *Consider a martingale Y_0, Y_1, \dots, Y_n with difference sequence X_1, X_2, \dots, X_n , i.e., $Y_0 = 0$, and for all $i \in [n]$, $Y_i = Y_{i-1} + X_i$ and $\mathbb{E}_{i-1}[Y_i] = Y_{i-1}$. Suppose $|X_i| \leq R$ almost surely for all $i \in [n]$. Define the predictable quadratic variation process of the martingale as $W_i = \sum_{j=1}^i \mathbb{E}_{j-1}[X_j^2]$, for all $i \in [n]$. Then for all $u \geq 0$, $\sigma^2 > 0$,*

$$\Pr\left[\exists i \in [n] : |Y_i| \geq u \text{ and } W_i \leq \sigma^2\right] \leq 2 \exp\left(-\frac{u^2/2}{\sigma^2 + Ru/3}\right).$$

For any $i \in [0 : T]$, let k_i denote the value of the width reduction step counter k when i is being incremented, and in the next lemma for simplicity of notations we will use the superscript (i) for the variables with superscript (i, k_i) .

Lemma G.12 (Bounds of sum of $\hat{\mathbf{u}}$ over all rounds). *Let k_i denote the number of width reduction steps taken by the algorithm when the i^{th} primal step is being executed. Let $a_0, \dots, a_T \in \mathbb{R}$ be an arbitrary sequence such that each a_i only depends on $\mathbf{S}^{(0)}, \dots, \mathbf{S}^{(i-1)}$ and each $|a_i| \leq C_a$. Then, for all $i \in [0 : T - 1]$, the vector $\hat{\mathbf{u}}^{(i, k_i)} = (\bar{\mathbf{R}}^{(i, k_i)})^{-1/2} \cdot (\mathbf{S}^{(i)})^\top \mathbf{S}^{(i)} \cdot (\bar{\mathbf{R}}^{(i, k_i)})^{1/2} \mathbf{u}^{(i, k_i)}$ satisfies the following properties:*

$$\begin{aligned} \mathbb{E}_{\mathbf{S}^{(0)}, \dots, \mathbf{S}^{(T-1)}} \left[\sum_{i=0}^{T-1} a_i \cdot (\hat{\mathbf{u}}_e^{(i, k_i)} - \mathbf{u}_e^{(i, k_i)}) \right] &= 0, \\ \Pr_{\mathbf{S}^{(0)}, \dots, \mathbf{S}^{(T-1)}} \left[\left| \sum_{i=0}^{T-1} a_i \cdot (\hat{\mathbf{u}}_e^{(i, k_i)} - \mathbf{u}_e^{(i, k_i)}) \right| \leq \frac{10C_a(C_1 + C_2) \log n \cdot \sqrt{nT}}{\sqrt{b\epsilon}} \right] &\geq 1 - 1/n^3. \end{aligned}$$

Proof. **TOPROVE 17** □

G.3.2 Analysis of Algorithm 3

Next we analyze the guarantee and iteration complexity of Algorithm 3, we again first prove the change of the potentials Φ (Def. Eq. (3)) and Ψ (Def. Eq. (4)) as in the previous sections.

From the argument in the proof of Lemma G.12, we have that with probability at least $1 - 1/n^3$, we have $|\hat{\mathbf{u}}_e^{(i, k_i)} - \mathbf{u}_e^{(i, k_i)}| \leq \frac{C_2 \sqrt{n}}{\sqrt{b\epsilon}}$ for all i , and then by Lemma G.10 we have $\mathbf{w}^{(i, k_i)} \geq 0$ for all i . In this section, we assume that we are conditioned on this event, and the failure of this event corresponds to the failure of our algorithm, which happens with probability at most $1/n^3$, as stated in Theorem 3.3.

Change in Φ

Lemma G.13. *With probability at least $1 - 1/n^3$, after i primal steps, and k width-reduction steps, if $\alpha \rho^{1/3} \leq \frac{\epsilon^{1/3}}{10n^{1/3}}$, the potential Φ is bounded as follows:*

$$\begin{aligned} \Phi(\mathbf{w}^{(i, k)}) &\leq \left(\Phi(\mathbf{w}^{(0, 0)}) \right) \left(1 + e^{\epsilon + \delta} \epsilon \alpha \cdot \left(1 + \frac{C_2}{\sqrt{b}} \right) + 2e^{\epsilon + 2\delta} \epsilon^2 \alpha^2 \left(1 + \frac{C_2^2 \cdot n}{b\epsilon} \right) \right)^i \\ &\quad \left(1 + \epsilon e^{\epsilon + 2\delta} \cdot (\tau^{-1} + \rho^{-2}) \right)^k. \end{aligned}$$

Furthermore, after every primal step, the potential can decrease by at most,

$$\Phi(\mathbf{w}^{(i+1, k)}) \geq \Phi(\mathbf{w}^{(i, k)}) \left(1 - e^{\epsilon + \delta} \epsilon \alpha \cdot \left(1 + \frac{C_2}{\sqrt{b}} \right) - 2e^{\epsilon + 2\delta} \epsilon^2 \alpha^2 \left(1 + \frac{C_2^2 \cdot n}{b\epsilon} \right) \right).$$

Proof. [TOPROVE 18](#) □

Change in Ψ

We recall the definitions of $\text{LASTWIDTH}(i, e)$, $\text{LAST}(i, e)$, $S_i \subseteq [2n]$, $\ell_{i,e}$, $\text{SIZE}(k)$, and $L \leq \frac{1}{100(\log^4 n)\epsilon\alpha\rho}$ from Section G.2. We will use these in the proof of the following lemma.

Lemma G.14 (Change in Ψ for Algorithm 3). *For any integer $c \geq 0$, after L primal steps from $(c-1)L$ to cL , if $\rho^2\tau^{-1} \geq 0.1$, the potential Ψ is bounded as follows:*

$$\Psi(\bar{\mathbf{r}}^{(cL, k_{cL})}) \geq \Psi(\bar{\mathbf{r}}^{((c-1)L, k_{(c-1)L})}) \cdot \left(1 - \tilde{O}(\epsilon\alpha\rho L)\right) \cdot \prod_{k=k_{(c-1)L}}^{k_{cL}} \left(1 + O\left(\frac{\epsilon^{4/3}\rho^{2/3} \cdot \text{SIZE}(k)^{1/3}}{n^{1/3} \cdot \log^{2/3}(\frac{n}{\epsilon\rho})}\right)\right).$$

Proof. [TOPROVE 19](#) □

Proof of Theorem 3.3

Proof. [TOPROVE 20](#) □

H Stability Guarantees of Algorithms 1 and 3

In this section we prove the stability guarantees of Algorithms 1 and 3.

For primal steps, we will prove that the ℓ_2 or ℓ_3 norm of the relative changes in resistances are bounded, and this means we can maintain a coordinate-wise approximation of the resistances under a low-rank update scheme.

For width reduction steps, we directly prove that the number of coordinates updated in each iteration follow the same low-rank update scheme.

In Section H.1, we prove the stability guarantees of the iterates of the multiplicative weights update algorithm with monotone weights given in Algorithm 1. Both primal and width reduction steps of this algorithm satisfy the stronger ℓ_3 -stability guarantee, i.e., the ℓ_3 norm of the relative changes in resistances is bounded.

In Section H.2, we prove that the primal steps of Algorithm 11 satisfy the weaker ℓ_2 -stability guarantee, and we also prove that its width reduction steps follow a low-rank update scheme.

In Section H.3, we prove a robust ℓ_2 -stability guarantee for the primal steps of Algorithm 3, and the width reduction steps are the same as that of Algorithm 11.

H.1 Stability Guarantees of Algorithm 1

Lemma 3.1 (Stability bound of ℓ_3 norm over all primal iterations). *Let k_i denote the number of width reduction steps taken by the algorithm when the i^{th} primal step is being executed. Then over all T primal steps of Algorithm 1, we have*

$$\sum_{i=0}^{T-1} \sum_{e \in S_i} \left(\frac{\mathbf{r}_e^{(i+1, k_i)} - \mathbf{r}_e^{(i, k_i)}}{\mathbf{r}_e^{(i, k_i)}} \right)^3 \leq \tilde{O}(\alpha^2 n) = \tilde{O}(n^{1/3} \epsilon^{2/3}).$$

Here S_i is the set of coordinates e at primal iteration i such that $\mathbf{r}_e^{(i+1, k_i)} \geq \mathbf{r}_e^{(i, k_i)}(1 + 3\epsilon\alpha)^6$.

⁶We note that it is sufficient to consider these sets S_i 's since any change that is smaller than the ones captured here can happen only $\tilde{O}(1)$ times.

Proof. TOPROVE 21 □

Lemma 3.2 (Stability bound of ℓ_3 norm over all width reduction iterations). *Let i_k denote the number of primal steps taken before the execution of the k^{th} width reduction step. Then, over all K width reduction steps of Algorithm 1, we have*

$$\sum_{k=0}^{K-1} \left(\frac{\mathbf{r}_e^{(i_k, k+1)} - \mathbf{r}_e^{(i_k, k)}}{\mathbf{r}_e^{(i_k, k)}} \right)^3 \leq \tilde{O}(n^{1/3}).$$

Proof. TOPROVE 22 □

H.2 Algorithm Warm Up: Low-Rank Update Scheme

Lemma H.1 (Low-rank update scheme of width reduction steps). *Let $\eta = 1/10$. For every $\ell = 0, 1, 2, \dots, \log(\frac{10n^{2/5}}{\tau^{1/2}\epsilon^{1/2}})$, in Algorithm 11 there are at most $\frac{T}{2^\ell}$ number of width reductions steps in which $\bar{\mathbf{r}}$ receives an update of rank $O(n^{1/5}2^{2\ell} \cdot (\log n)^{28/3} \log(\frac{n}{\Psi_0})^{4/3}\epsilon^{-1})$.*

Proof. TOPROVE 23 □

Lemma H.2 (ℓ_2 stability of primal steps). *Algorithm 11 satisfies that for all primal steps i ,*

$$\sum_e \left(\log(\mathbf{r}_e^{(i+1, k_i)}) - \log(\mathbf{r}_e^{(i, k_i)}) \right)^2 \leq \tilde{O}(n^{2\eta}\epsilon^3).$$

Proof. TOPROVE 24 □

From the above lemma and Lemma B.1, we directly have the following corollary.

Corollary H.3 (Low-rank update scheme of primal steps). *For every $\ell = 0, 1, \dots, \log T$, in Algorithm 11 there are at most $\frac{T}{2^\ell}$ number of primal steps in which $\bar{\mathbf{r}}$ receives an update of rank $O(n^{2\eta}2^{2\ell} \cdot \epsilon \cdot \log(n)^{-2/3} \log(\frac{n}{\Psi_0})^{-2/3})$.*

H.3 Algorithm 3

Low-rank update scheme First note that the width reduction steps of Algorithm 3 are the same as Algorithm 11, so they follow the same low-rank update scheme as Lemma H.1.

Next we prove the robust ℓ_2 stability guarantees of the primal steps of Algorithm 3, and this combined with Lemma B.2 will give us the desired low-rank update scheme.

Lemma H.4 (Robust ℓ_2 stability of primal steps). *For every primal step (i, k) of Algorithm 3, define a “fake” weight:*

$$\tilde{\mathbf{r}}^{(i+1, k)} = \mathbf{r}^{(i+1, k)} - \mathbf{w}^{(i, k)}\epsilon\alpha \cdot (\hat{\mathbf{u}}^{(i, k)} - \mathbf{u}^{(i, k)}) \cdot \frac{(1 + \vec{\alpha}^{(i, k)}\hat{\mathbf{u}}^{(i, k)})}{(1 + \alpha\hat{\mathbf{u}}^{(i, k)})}. \quad (5)$$

Every primal step of Algorithm 3 satisfies the following robust ℓ_2 stability property if $b \geq \frac{n\alpha \log^4 n}{\epsilon^3}$:

1.

$$\sum_e \ln \left(\frac{\tilde{\mathbf{r}}_e^{(i+1, k)}}{\mathbf{r}_e^{(i, k)}} \right)^2 \leq O(n^{2\eta}).$$

2. $\forall t \in [T], \forall e$, with probability $1 - 1/n^4$,

$$\left| \sum_{i=t'-t}^{t'} \ln \left(\frac{\tilde{\mathbf{r}}_e^{(i,k)}}{\mathbf{r}_e^{(i,k)}} \right) \right| \leq O(\epsilon).$$

Proof. [TOPROVE 25](#) □

Combining the above lemma and Lemma B.2, we directly have the following corollary.

Corollary H.5 (Low-rank update scheme of primal steps). *For every $\ell = 0, 1, \dots, \log T$, in Algorithm 3 there are at most $\frac{T}{2^\ell}$ number of primal steps in which $\bar{\mathbf{r}}$ receives an update of rank $O((\frac{\log n}{\delta})^2 \cdot n^{2\eta} \cdot 2^{2\ell})$.*

I Missing Proofs

Proof of Lemma A.1

Proof. [TOPROVE 26](#) □

Proof of Lemma A.2

Proof. [TOPROVE 27](#) □

Proof of Lemma A.3

This is similar to the proof of Lemma 5.11 of [Adi+19].

Proof. [TOPROVE 28](#) □

Proof of Fact A.6

Proof. [TOPROVE 29](#) □

Missing Proofs of inverse mainenance data structures

Proof of Lemma C.1

Proof. [TOPROVE 30](#) □

Proof of Lemma C.2

Proof. [TOPROVE 31](#) □

Proof of Lemma C.4

Proof. [TOPROVE 32](#) □

J MWU with Non-Monotone Weights and $n^{1/3}$ Iterations (Optional)

In this section, we will prove Theorem G.1. Our analysis follows a similar structure to that of Algorithm 1. We would track the same potentials as defined in Equations (3) and (4).

In this section, we again use the notation k_i to denote the total number of width reduction steps taken before the i^{th} primal step is being executed, and we use the notation i_k to denote the number of primal steps taken by the algorithm when the k^{th} width step is being executed. We show that for input $\tilde{\mathbf{C}} = \begin{bmatrix} \mathbf{C} \\ -\mathbf{C} \end{bmatrix}$ and $\tilde{\mathbf{d}} = \begin{bmatrix} \mathbf{d} \\ -\mathbf{d} \end{bmatrix}$, the algorithm returns $\tilde{\mathbf{x}}$ such that $\|\mathbf{C}\tilde{\mathbf{x}} - \mathbf{d}\|_\infty \leq 1 + O(\epsilon)$.

Convergence Analysis

We begin by showing that $\mathbf{w}^{(i,k)} > 0$ for all i and k . Observe that this was always true for Algorithm 1.

Lemma J.1. *If $\alpha\rho^{1/3} \leq \frac{\epsilon^{1/3}}{10n^{1/3}}$ and $\delta \leq 1/2$, then for every iteration i and k and every coordinate e , $\mathbf{w}_e^{(i,k)} > 0$. Furthermore, every primal update satisfies, $|\vec{\alpha}^{(i,k)}(\tilde{\mathbf{C}}\Delta^{(i,k)} - \tilde{\mathbf{d}})| \leq 1/10$.*

Proof. **TOPROVE 33** □

We now begin our analysis. The next two lemmas show how our potentials change with every iteration of the algorithm.

Change in Φ

Lemma J.2. *After i primal steps, and k width-reduction steps, the potential Φ is bounded as follows:*

$$\Phi(\mathbf{w}^{(i,k)}) \leq \left(\Phi(\mathbf{w}^{(0,0)})\right) \left(1 + \epsilon\alpha_+ e^{\epsilon/2}\right)^i \left(1 + \frac{2\epsilon e^\epsilon}{\tau}\right)^k.$$

Furthermore, after every primal step, the potential can decrease by at most,

$$\Phi(\mathbf{w}^{(i+1,k)}) \geq \Phi(\mathbf{w}^{(i,k)}) \left(1 - \epsilon\alpha_+ e^{\epsilon/2}\right)^i.$$

Proof. **TOPROVE 34** □

Change in Ψ

We now prove how our potential Ψ changes with a primal and width reduction step.

Lemma J.3. *If the parameters satisfy $\rho^2 \geq \tau\epsilon^{1/3}n^{-2\eta}$, $\rho \geq n^{1/2-3\eta}$, then after i primal and k width reduction steps, the potential $\Psi(\mathbf{r}^{(i,k)})$ satisfies,*

$$\Psi(\mathbf{r}^{(i,k)}) \geq \Psi(\mathbf{r}^{(0,0)}) (1 - 10\epsilon\alpha\rho)^i \left(1 + \frac{\epsilon^{4/3}n^{-2\eta}}{10}\right)^k$$

Proof. **TOPROVE 35** □

We will now combine the changes in the two potentials similar to the proof of Theorem 2.1.

Proof of Theorem G.1

Proof. **TOPROVE 36**

□

Lower Bound on Ψ

Lemma J.4. *For all i, k and $\mathbf{r}^{(i,k)}$ as defined in Algorithm 10, $\Psi(\mathbf{r}^{(i,k)}) \geq \frac{1}{1+2\epsilon} \cdot \Psi(\mathbf{r}^{(0,0)})$.*

Proof. **TOPROVE 37**

□