Adaptive Sparsification for Linear Programming

Étienne Objois* Adrian Vladu†

Abstract

We introduce a generic framework for solving linear programs (LPs) with many constraints ($n \gg d$) via adaptive sparsification. Our approach provides a principled generalization of the techniques of [Assadi '23] from matching problems to general LPs and robustifies [Clarkson's '95] celebrated algorithm for the exact setting. The framework reduces LP solving to a sequence of calls to a "low-violation oracle" on small, adaptively sampled subproblems, which we analyze through the lens of the multiplicative weight update method.

Our main results demonstrate the versatility of this paradigm. First, we present a quantum version of Clarkson's algorithm that finds an exact solution to an LP using $\tilde{O}(\sqrt{n}d^3)$ row-queries to the constraint matrix. This is achieved by accelerating the classical bottleneck (the search for violated constraints) with a generalization of Grover search, decoupling the quantum component from the classical solver. Second, our framework yields new state-of-the-art algorithms for mixed packing and covering problems when the packing constraints are "simple". By retaining all packing constraints while sampling only from the covering constraints, we achieve a significant width reduction, leading to faster solvers in both the classical and quantum query models. Our work provides a modular and powerful approach for accelerating LP solvers.

1 Introduction

Linear Programming (LP) is a central primitive in optimization, with applications ranging from combinatorial optimization to quantum information. Classical research has long pursued faster solvers, both in the high-precision regime (polynomial in $\log 1/\varepsilon$) via interior point and cutting plane methods [Vai89; LS15; CLS19; Bra+21], and in the low-precision regime (polynomial in $1/\varepsilon$) using first-order techniques such as multiplicative weights [PST91; GK95; AHK12].

A particularly intriguing regime arises when the number of constraints n far exceeds the dimension d. Since only d constraints suffice to pin down an optimal solution, one may hope to reduce n dramatically without losing accuracy. In the context of graph algorithms, this idea underlies sparsification techniques [BK15; SS08], which replace large inputs with much smaller sketches that preserve the structure of (near-)optimal solutions. However, its potential for linear programming remains largely unexplored.

A classical result of Clarkson [Cla95] gives a concrete way to reduce constraints adaptively. Instead of attempting a one-shot reduction of all n constraints, Clarkson iteratively samples small subsets, solves the resulting reduced LPs, and uses the solutions to adaptively adjust sampling probabilities. This iterative reweighting guarantees that after only $\widetilde{O}(d)$[1] iterations, the algorithm identifies the true solution while never solving an LP with more than $O(d^2)$ constraints. More recently, Assadi [Ass25] showed how related ideas yield powerful semi-streaming algorithms for approximate matching, with performance reminiscent of multiplicative weights but with strictly better dependence on the error parameter ε. Together, these results suggest that adaptive sparsification could form a general principle for reducing large LP instances to efficiently solvable ones.

This perspective is particularly timely in the quantum setting. In the row-query model, solving an LP to constant precision requires at least $\Omega(\sqrt{nd})$ queries to the constraint matrix [Ape+20], a

*IRIF, Université Paris Cité, objois@irif.fr

†CNRS, IRIF, Université Paris Cité, vladu@irif.fr

[1] $\widetilde{O}(.)$ hides poly-logarithmic dependencies on n, d, and ε^{-1}.

bound conjectured to be tight [AG24]. Moreover, quantum algorithms can sample r elements from a distribution over n elements in just $\widetilde{O}(\sqrt{nr})$ queries [AW23]. Clarkson's and Assadi's frameworks, which only needs $\widetilde{O}(d)$ and $\widetilde{O}(\frac{1}{\varepsilon})$ iterations of sampling a small number of constraints each, therefore naturally aligns with quantum speedups.

These observations lead to a guiding question:

Can adaptive sparsification provide a unifying framework for quantum algorithms that efficiently solve LPs exactly or approximately?

1.1 Our Contributions

A linear program is defined as

$$\max_{\boldsymbol{x}\in\mathcal{D}} \langle \boldsymbol{c}, \boldsymbol{x}\rangle, \qquad \text{subject to } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \tag{1}$$

where $\boldsymbol{A} \in \mathbb{R}^{n\times d}$, $\boldsymbol{b} \in \mathbb{R}^n$ and $\boldsymbol{c} \in \mathbb{R}^d$. $\mathcal{D}$ should be seen as a simple convex set, present to upper bound the *width* of the linear program. We denote by OPT the optimum value, that is $\max\{\langle \boldsymbol{c}, \boldsymbol{x}\rangle : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \text{ and } \boldsymbol{x} \in \mathcal{D}\}$.

We first present a quantum version of Clarkson's algorithm that uses a generalization of Grover Search to sample the constraints achieves low query complexity of $\widetilde{O}(\sqrt{n}d^3)$. We present the comparison with other algorithms in Table 1.

Theorem 1. *There is a randomized quantum algorithm that finds an exact solution of (1) using $\widetilde{O}(\sqrt{n}d^3)$ row-queries to $\boldsymbol{A}$. Furthermore, this algorithm runs in expected time*

$$\widetilde{O}\left(\sqrt{n}d^3 r + d \cdot \mathcal{T}_{\text{exact}}(d, 24d^2)\right),$$

where r is the row-sparsity of $\boldsymbol{A}$, and $\mathcal{T}_{\text{exact}}(d, n)$ is the time required to compute an exact solution to a linear program with d variables and n constraints.

Remark. *When the bit complexity of the input is not large, one can rely on high-precision solvers such as interior point methods. If L is the bit-complexity of the input, that is, the number of bits required to write $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$,[2] then we can combine Clarkson's algorithm with [LS15; vdBra+21] to achieve an expected time of $\widetilde{O}(d \cdot \text{nnz}(\boldsymbol{A}) + L \cdot d^{3.5} \min\{r, \sqrt{d}\})$. In this case, our algorithm still requires $\widetilde{O}(\sqrt{n}d^3)$ row-queries to $\boldsymbol{A}$, and runs in time $\widetilde{O}(\sqrt{n}d^3 r + L \cdot d^{3.5} \min\{r, \sqrt{d}\})$.*

Low-Precision. In the low-precision regime, we give an algorithm that computes an ε-approximate solution using $O(V_{\max}/\varepsilon \log n)$ iterations, where $V_{\max}$ corresponds to the largest violation achievable by a solution in $\mathcal{D}$. $V_{\max}$ can be much smaller than the width, for pure covering problems for instance, $V_{\max} = 1$, while the width can be as large as d.

We provide two quantum versions of this classical algorithm. The main difference comes from computing the query access to the sampling probabilities. In the first version, we require fewer iterations, but querying the sampling probability of a constraint at iteration t requires t row-queries to $\boldsymbol{A}$. In the second version, we perform more iterations, but querying the sampling probability of a constraint requires only one row-query to $\boldsymbol{A}$. We present the different results in Table 2.

Mixed Packing and Covering Problems. A mixed packing and covering problem is of the form

$$\text{find } \boldsymbol{x} \in [0,1]^d, \qquad \text{subject to } \boldsymbol{C}\boldsymbol{x} \geq \mathbf{1}, \boldsymbol{P}\boldsymbol{x} \leq \mathbf{1},$$

where $\boldsymbol{C} \in [0,1]^{n_c\times d}$, and $\boldsymbol{P} \in [0,1]^{n_p\times d}$.[3] When the matrix $\boldsymbol{C}$ is tall, i.e. $n_c \gg n_p + d + \varepsilon^{-1}$, we give a randomized algorithm improving the current state-of-the-art. We also provide a quantum version of this algorithm. We present those results in Table 3.

[2] Assume $\boldsymbol{A} \in \mathbb{R}^{n\times d}$, $\boldsymbol{b} \in \mathbb{R}^n$ and $\boldsymbol{c} \in \mathbb{R}^d$ are integral, and $n \gg d$. One can think of L being the same order of magnitude as $\log n + \log(1 + d_{\max}) + \log(1 + \max\{\|\boldsymbol{c}\|_\infty, \|\boldsymbol{d}\|_\infty\})$ where $d_{\max}$ is the largest subdeterminant of $\boldsymbol{A}$ in absolute value.

[3] Solving a mixed packing and covering LP can be reduced to multiple problems of this form by adding a poly-logarithmic number of constraints, see appendix B in [BSW19].

Paper	Query model	Total runtime	Query complexity
		exact solver	
[Cla95]	classical	$O\left(d \log n \cdot (\mathrm{nnz}(\boldsymbol{A}) + \mathcal{T}_{\mathrm{exact}}(d, 6d^2))\right)$	-
Theorem 10	quantum	$\widetilde{O}\left(d(\sqrt{n}d^2 r + \mathcal{T}_{\mathrm{exact}}(d, 24d^2))\right)$	$\widetilde{O}\left(\sqrt{n}d^3\right)$
		high-precision solver	
[vdBra+21]	classical	$\widetilde{O}\left(nd + d^{2.5}\right)$	-
[LS15]	classical	$\widetilde{O}\left(\mathrm{nnz}(\boldsymbol{A})\sqrt{d} + d^{2.5}\right)$	-
[AG24]	quantum	$\widetilde{O}\left(\sqrt{n}d^7 r^2 + d^{\omega+7}\right)$	$\widetilde{O}\left(\sqrt{n}d^5\right)$
quantum version of [LSW15]	quantum	$\widetilde{O}\left(\sqrt{n}dr + d^3\right)$	$\widetilde{O}\left(\sqrt{n}d\right)$

Table 1: Comparison between algorithms for solving a linear program of the form $\max\langle \boldsymbol{c}, \boldsymbol{x}\rangle$ subject to $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ and $\boldsymbol{x} \in \mathcal{D}$. We consider $\boldsymbol{A} \in \mathbb{R}^{n\times d}$, r is the row-sparsity of $\boldsymbol{A}$, $\rho = \max_{\boldsymbol{x}\in\mathcal{D}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_\infty$ is the width of the LP, $\varepsilon > 0$, and ω is the matrix multiplication exponent.

Paper	Query model	Type	Total runtime
[AHK12]	classical	deterministic	$O\left(\frac{\rho V_{\max}}{\varepsilon^2} \log n \cdot (\mathrm{nnz}(\boldsymbol{A}) + \mathcal{T}_{\mathrm{linear}}(d))\right)$
Theorem 7	classical	randomized	$\widetilde{O}\left(\frac{V_{\max}}{\varepsilon}(\mathrm{nnz}(\boldsymbol{A}) + \mathcal{T}_{\mathrm{exact}}(d, 24d\frac{V_{\max}}{\varepsilon}))\right)$
Theorem 12	quantum	randomized	$\widetilde{O}\left(\frac{V_{\max}}{\varepsilon}(\sqrt{nd\frac{V_{\max}}{\varepsilon}}\frac{V_{\max}}{\varepsilon}r + \mathcal{T}_{\mathrm{exact}}(d, 24d\frac{V_{\max}}{\varepsilon}))\right)$
Theorem 13	quantum	randomized	$\widetilde{O}\left(\frac{\rho}{\varepsilon}(\sqrt{nd\frac{\rho}{\varepsilon}}r + \mathcal{T}_{\mathrm{exact}}(d, 24d\frac{\rho}{\varepsilon}))\right)$

Table 2: Comparison between algorithms for finding an ε-approximate solution to a linear program of the form $\max\langle \boldsymbol{c}, \boldsymbol{x}\rangle$ subject to $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ and $\boldsymbol{x} \in \mathcal{D}$. We consider $\boldsymbol{A} \in \mathbb{R}^{n\times d}$, r is the row-sparsity of $\boldsymbol{A}$, $\rho = \max_{\boldsymbol{x}\in\mathcal{D}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_\infty$ is the width of the LP, $\varepsilon > 0$, and $V_{\max} = \max_{\boldsymbol{x}\in\mathcal{D}} \max_{i\in[n]} (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})_i$ is the largest violation possible.

Paper	Query model	Type	Total runtime
[You14; Qua19]	classical	deterministic	$\widetilde{O}\left(\frac{\text{nnz}}{\varepsilon^2}\right)$
[BSW19]	classical	deterministic	$\widetilde{O}\left(\frac{r}{\varepsilon}\text{nnz}\right)$
[CQ18]	classical	randomized	$\widetilde{O}\left(\frac{\text{nnz}}{\varepsilon}+\frac{n}{\varepsilon^2}+\frac{d}{\varepsilon^3}\right)$
Theorem 8	classical	randomized	$\widetilde{O}\left(\frac{1}{\varepsilon}\cdot(\text{nnz}(\boldsymbol{C})+\mathcal{T}_{\text{mixed}}(d,n_p,\widetilde{O}\left(\frac{d}{\varepsilon}\right),\varepsilon))\right)$
Theorem 11	quantum	randomized	$\widetilde{O}\left(\frac{1}{\varepsilon}\left(\sqrt{n_c\frac{d}{\varepsilon}}\frac{r_c}{\varepsilon}+\mathcal{T}_{\text{mixed}}(d,n_p,\widetilde{O}\left(\frac{d}{\varepsilon}\right),\varepsilon)\right)\right)$

Table 3: Comparison between algorithms finding a $1+\varepsilon$ approximate solution of mixed packing and covering problems of the form: find $\boldsymbol{x}\in[0,1]^d$ subject to $\boldsymbol{P}\boldsymbol{x}\leq\mathbf{1}$ and $\boldsymbol{C}\boldsymbol{x}\geq\mathbf{1}$. We assume $\boldsymbol{P}\in[0,1]^{n_p\times d}$ and $\boldsymbol{C}\in[0,1]^{n_c\times d}$. Here r_p (resp. r_c) denotes the row-sparsity of $\boldsymbol{P}$ (resp. $\boldsymbol{C}$). $n=n_p+n_c$, $r=\max\{r_p,r_c\}$, and $\text{nnz}=\text{nnz}(\boldsymbol{P})+\text{nnz}(\boldsymbol{C})$. $\mathcal{T}_{\text{mixed}}(d,n_p,n_c,\varepsilon)$ denotes the running time of a $1+\varepsilon$ approximate mixed packing and covering LP solver on d variables with n_p (resp. n_c) packing (resp. covering) constraints.

Remark. *Assume there is a constant number of packing constraints (but strictly larger than* 1*, otherwise it becomes a pure covering LP with other, faster solvers). In this case, we may have* $\text{nnz}(\boldsymbol{P})=O(d)$*, assume* $\boldsymbol{C}$ *is tall and dense* $\text{nnz}(\boldsymbol{C})=n\cdot d$*, and* $n\gg d^2\varepsilon^{-2}+\varepsilon^{-3}$*. In that case, if we use the algorithm from [Qua19] at each iteration, we find an* ε*-approximate solution in time* $\widetilde{O}\left(\frac{nd}{\varepsilon}+\frac{d^2}{\varepsilon^4}\right)$ *which is faster than other mixed packing and coverings problems. The only draw back is our algorithm is randomized while [You14; Qua19; BSW19] are deterministic.*

1.2 Overview of Approach

Our core contribution is a unified and versatile framework for solving linear programs (LPs) through *constraint sparsification*. We present a single, iterative algorithm that uses the multiplicative weights update (MWU) method as its central engine. By simply adjusting its internal parameters, this one algorithm can function as either a low-precision approximate solver or an exact solver. Our framework consists of 4 steps performed iteratively.

1. *Sparsify:* Construct a sparse LP by selecting a small, weighted subset of constraints.
2. *Solve:* Find a solution for the sparse LP.
3. *Verify:* Check the solution against the entire original set of n constraints to find violations.
4. *Refine:* Update the weighting scheme to increase the importance of the violated constraint and return to step 1, creating a new, refined sparsifier.

In step 1, we sparsify the linear program by sampling constraints from a distribution. In step 2 we solve the partial linear program obtained, in this step we show that we can either use an exact solver, or an ε-approximation one. We then need to verify which constraint has been violated, this is used to compute a new distribution (step 4) for the next iteration. This framework is also exceptionally amenable to quantum speedups. One difficulty we face is that the distribution changes a lot from one iteration to the other so maintaining it can be challenging. Fortunately, we are making few iterations which allows us to recompute it from scratch each iteration.

Constraint sparsification for low-precision algorithms. We first present our approach for approximately solving LPs via an adaptive sequence of calls to an exact LP solver on a *sparse* subset

of constraints. The algorithm generalizes Assadi's approach [Ass25] for maximum matching to general settings, and our analysis casts his approach in a principled framework.

Formally, suppose we aim to approximate the decision problem $\{\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \in \mathcal{D}\}$, in the sense that we seek to either return some $\boldsymbol{x} \in \mathcal{D}$ such that $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} + \varepsilon\mathbf{1}$, or return INFEASIBLE if the input problem does not have a solution.

The textbook approach for solving this problem is via the MWU framework put forth in [PST91], which runs in $O\left(\rho^2 \log n/\varepsilon^2\right)$ iterations, each of which involves calling a linear optimization oracle over $\mathcal{D}$, and updating the weights fed into the oracle in time $O(\text{nnz}(\boldsymbol{A}))$. Here $\rho = \max_{\boldsymbol{x}\in\mathcal{D}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_\infty$ represents the *width* of the problem.

In our case, we instead solve a sequence of $O(\rho \log n/\varepsilon)$ sub-problems, each of which involves solving exactly an LP over a sparse subset of constraints. The approach is as follows: in each iteration we sample a subset of constraints Q, and exactly solve the LP containing only the constraints in Q. Then we check the solution, and for each of the original constraints that is violated, we increase its sampling probability. To understand why this approach works, we can cast it in the MWU framework by defining a *low-violation oracle*, namely a procedure which given a distribution $\boldsymbol{p} \in \Delta_n$ of constraint weights, outputs a solution $\boldsymbol{x} \in \mathcal{D}$ such that the *violation vector* $\boldsymbol{v} = \mathbf{1}_{\boldsymbol{A}\boldsymbol{x}>\boldsymbol{b}}$ represented by the indicator vector of constraints violated by $\boldsymbol{x}$, violates only constraints with low total weight, in the sense that $\langle \boldsymbol{p}, \boldsymbol{v}\rangle \leq \frac{\varepsilon}{3\rho}$. Simply running MWU on these low-weight violation vectors for $T = O\left(\rho \log n/\varepsilon\right)$ iterations will yield a sequence $\{\boldsymbol{v}_t\}_{t=1}^T$ for which $\max\left\{\frac{1}{T}\sum_{t=1}^T \boldsymbol{v}_t\right\} \leq \varepsilon/\rho$. This in turn translates into a bound on $\max\left\{\boldsymbol{A}\left(\frac{1}{T}\sum_{t=1}^T \boldsymbol{x}_t\right) - \boldsymbol{b}\right\} \leq \varepsilon$, since each constraint violation corresponds to an iterate $\boldsymbol{x}_t$ for which the corresponding violated constraint $\boldsymbol{A}_{i:}$, satisfies $\langle \boldsymbol{A}_{i:}, \boldsymbol{x}_t\rangle - \boldsymbol{b}_i \leq \rho$. So if each constraint is violated at most $(\varepsilon/\rho)\cdot T$ times, over all the T iterations, the total violation suffered by that constraint is at most εT.

The advantageous feature of this approach is that, since the loss vectors $\boldsymbol{v}_t$ that we run MWU have only 0-1 entries, and we tolerate a large (constant) step size, the number of iterations required scales only linearly with ρ/ε rather than quadratically as in standard instantiations of [PST91]. We discuss this further in Section A.

We can obtain a low-violation oracle simply by sub-sampling the constraints, and exactly solving an LP for the sampled subset. Indeed, per the analysis in [Cla95], to achieve this we can simply sample each constraint i with probability proportional to $\boldsymbol{p}_i$. The main challenge is bounding the number of sampled constraints required, which we show is the product of the number of variables and the number of iterations.

We can use an ε-approximation solver as long as the solver's output set contains few elements. Assadi used this technique on some combinatorial problems, known to have few solutions, such as matching [Ass25]. When the solver's output set is sparse, we can still sample constraint i with probability proportional to $\boldsymbol{p}_i$. Per the analysis in [Ass25], we can upper bound the number of sampled constraints by the logarithm of the size of the solver's output set. In particular, for LPs for which a sparse ε-net covers all the possible outputs of the approximate LP solver subroutine exists, we can sample a number of constraints proportional to the logarithm of this ε-net's size.

We observe that for *nice* solution space, a sparse ε-net can be obtained. For mixed packing and covering problems, we can assume a solution has entry of the form $\boldsymbol{x}_i = \varepsilon/d(1+\varepsilon)^{k_i}$ for some integer $k_i \in [1 + \frac{1}{\varepsilon}\log\frac{d}{\varepsilon}]$. This allows us to obtain an ε-net of size $O((\varepsilon^{-1}\log d/\varepsilon)^d)$. We prove we can reduce a mixed packing and covering LP by solving $\widetilde{O}(\frac{1}{\varepsilon})$ times a sparser version of the LP containing all the packing constraints, but only $O(\frac{d}{\varepsilon}\cdot\log(\frac{1}{\varepsilon}\log\frac{d}{\varepsilon}))$ of its covering constraints.

A unified framework for constraint sparsification. Consider the decision problem $\{\boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}, \boldsymbol{x} \in \mathcal{D}\}$. We generalize the analysis for low-precision algorithms to exact solvers. We in fact prove that it is sufficient to find a sequence of solutions $\{\boldsymbol{x}_j\}_{j\in[T]}$ such that

$$|\{j \in [T] : \langle \boldsymbol{A}_{i:}, \boldsymbol{x}_j\rangle > \boldsymbol{b}_i + \varepsilon\}| < \mu \cdot T, \tag{2}$$

for some value of ε and μ. This abstraction is the foundation of our work. Per Clarkson's analysis, solving this problem with $\varepsilon = 0$, and $\mu = d$ guarantees one of the $\boldsymbol{x}_j$ is an exact solution. On

the other hand, per Assadi's analysis, it is sufficient to find solutions for ε, and $\mu = \varepsilon$ to obtain $(1+2\varepsilon)$-approximation of pure covering problems.

Solving 2 fits naturally inside the MWU framework. Consider $f_i : X \mapsto |\{\boldsymbol{x} \in X : \langle \boldsymbol{A}_{i:}, \boldsymbol{x}\rangle > \boldsymbol{b}_i + \varepsilon\}|$. The goal is to find a set X of solutions such that $\max_{i\in[n]} f_i(X) < \mu\,|X|$.

Our main algorithm finds the desired set of solutions $\{\boldsymbol{x}_j\}_{j\in[T]}$ iteratively. In each of the T iterations, we use an *MWU-based procedure as a black box* to find the next solution, $\boldsymbol{x}_j$. This procedure maintains a distribution of weights over the n constraints. Its goal is to call a base LP solver on a sparse sample of the constraints to find a candidate solution $\boldsymbol{x}_j$ that satisfies the "low-violation" property relative to the current weights. That is, the solution primarily satisfies the constraints that the MWU framework currently deems important (i.e., those with high weight). MWU reduces finding $\{\boldsymbol{x}_j\}_{j\in[T]}$ satisfying 2 to iteratively finding $\boldsymbol{x}$ such that for a given $\boldsymbol{p} \in \Delta_n$, $\langle \boldsymbol{p}, \mathbf{1}_{\boldsymbol{A}\boldsymbol{x} > \boldsymbol{b}+\varepsilon\mathbf{1}}\rangle < \mu$.

In order to solve this easier problem, we rely on sampling constraints and using a base solver on the sampled constraints. We consider what we call (ε, μ)-low-violation oracles. Given $\boldsymbol{p} \in \Delta_n$, a low-violation oracle returns $\boldsymbol{x} \in \mathbb{R}^d$ such that $\langle \boldsymbol{p}, \mathbf{1}_{\boldsymbol{A}\boldsymbol{x} > \boldsymbol{b}+\varepsilon\mathbf{1}}\rangle < \mu$. Hence, we obtain the following theorem.

Theorem 2 (Short version of 3)**.** *It is possible to find a set of vectors $\{\boldsymbol{x}_j\}_{j\in[T]}$ solution to 2 using $T = O(\frac{\log n}{\mu})$ calls to an (ε, μ)-low-violation oracle.*

Exploiting quantum speedups via Grover search. We show that our framework is extremely amenable to quantum speedups. In fact, since the running time of our algorithms is largely dominated by the time required to identify violated constraints, being able to efficiently do so could lead to important improvements.

We show in Section 5 that in the quantum query access model, where each constraint has an associated feasibility oracle given a candidate solution $\boldsymbol{x}$, one can replace the time dependence in input sparsity with $\sqrt{n}d^{O(1)}$, which yields sublinear time algorithms when $n \gg d$. While previous works [AW23; AG24] have developed algorithms that are carefully crafted to fit the restrictive requirements of quantum computational models, in our case the source of speedups stems uniquely from quickly identifying violated constraints. This is achieved via a generalization of Grover's algorithm, which is the only quantum power tool we rely on (c.f. Lemma 9).

Our main challenge is to access the sampling probabilities. Our sampling probabilities are proportional to the weights inside the MWU framework. For a set of solutions X, the weight of a constraint is an exponential of the number of solutions from X violating this same constraint. Since we update the weights *aggressively*, any constant upper bound is no longer true after a constant number of iterations. This prevents us from reducing the update time of the sampling probabilities even in the classical scheme. With quantum query access, we do not maintain the weight value from one iteration to the other, hence we need to recompute it at each iteration. That means at iteration t, we require t row-query access to the constraint matrix. In the ε-approximation setting, we show that we can actually use one query access to $\boldsymbol{A}$ at each iteration, but this comes at the cost of a larger width.

We also need to estimate the sum of the weights to a constant factor at each iteration, this is performed at each iteration by sampling few constraints.

1.3 Organization

The remainder of this paper is organized as follows. We begin in Section 2 by establishing the necessary preliminaries, including our notation, formal definitions for the classes of linear programs we consider, and the quantum query model used in our analysis. In Section 3, we introduce our unified iterative sparsification paradigm, which forms the theoretical core of our work. We formalize the problem of finding a set of solutions with a low frequency of constraint violations and introduce the key algorithmic primitive, the low-violation oracle, which underpins our entire framework. Section 4 demonstrates the versatility of this framework in the classical setting; we show how to construct these oracles and use them to generalize the seminal results of Clarkson and Assadi, to efficient algorithms for low-precision general LPs and mixed packing and covering problems. Finally, in Section 5, we show that our framework is amenable to quantum acceleration, demonstrating how replacing the classical

bottleneck of identifying violated constraints with a quantum search subroutine leads to significant speedups and improved query complexities.

2 Preliminaries

2.1 Notation

We write vectors and matrices in bold. Matrices are written in uppercase. We use $\langle ., . \rangle$ notation to denote the inner product between two vectors. For a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ and an integer $i \in [n]$, $\boldsymbol{A}_{i:}$ denotes the i^{th} row of $\boldsymbol{A}$. For a subset $S \subseteq [n]$, we let $\boldsymbol{A}_S$ be the matrix $\boldsymbol{A}$ whose rows are restricted to those in set S. We define $\text{nnz}(\boldsymbol{A})$ as the number of nonzero entries of $\boldsymbol{A}$. For a linear program of the form 3, there is a set B of d constraints such that the solution $\boldsymbol{x}$ satisfies $\boldsymbol{A}_B \boldsymbol{x} = \boldsymbol{b}_B$, which we call the base of the linear program.

2.2 Classes of Linear Programs

In the literature, LPs have been analyzed under different assumptions. Consider a linear program of the form $\{\boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$. The first important remark on solving LPs is whether we have an *implicit* or *explicit* access to $\boldsymbol{A}$. The classic MWU approach can be used with implicit access as long as a linear optimization oracle exists. Often, implicit LPs arise when the number variables d is exponential while the number of constraint is not. This can be the case for spanning tree covering problems for instance, the number of spanning trees on a graph is exponential while there is only one constraint per edge. In this work, we focus on LPs given *explicitly*, meaning $\boldsymbol{A}$ can be stored in the (Q)RAM.

General LPs. In general, a linear program can be written as:

$$\max \langle \boldsymbol{c}, \boldsymbol{x} \rangle, \qquad \text{subject to } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}. \tag{3}$$

We denote by OPT the optimal value of (3). For exact solvers, we consider LPs that are not degenerate, that means there is only one set of d constraints $B \subseteq [n]$ such that the optimum of the linear program lies at their intersection. An ε-feasible solution of 3 is a vector $\boldsymbol{x}$ such that $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} + \varepsilon \boldsymbol{1}$. An ε-approximation of 3 is an ε-feasible solution $\boldsymbol{x}$ such that $\langle \boldsymbol{c}, \boldsymbol{x} \rangle \geq \text{OPT}$.

Positive LPs. Positive LPs are of the form 3 with an added constraint on the sign of rows of $\boldsymbol{A}$. We distinguish three types of positive LPs, first *pure covering* LPs are of the form $\min\{\|\boldsymbol{x}\|_1 : \boldsymbol{C}\boldsymbol{x} \geq \boldsymbol{1}\}$ where $\boldsymbol{C}$ has positive entries. Second, there are *pure packing* LPs with the form $\max\{\|\boldsymbol{x}\|_1 : \boldsymbol{P}\boldsymbol{x} \leq \boldsymbol{1}\}$ where $\boldsymbol{P}$ has positive entries. Note that the dual of a pure covering LP is a pure packing LP. Finally, there are mixed packing/covering LPs, they are of the form $\{\boldsymbol{x} : \boldsymbol{C}\boldsymbol{x} \geq \boldsymbol{1}, \boldsymbol{P}\boldsymbol{x} \leq \boldsymbol{1}\}$.

By adding a poly-logarithmic number of variables, we can assume that $\boldsymbol{C}$ and $\boldsymbol{P}$ have entries in $[0, 1]$. Moreover, we can assume that the exact solution is $\boldsymbol{x}$ such that $\boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{1}$ coordinate-wise.

We say that $\boldsymbol{x}$ is a $(1+\varepsilon)$-approximation of a covering problem if $(1+\varepsilon)\text{OPT} \geq \|\boldsymbol{x}\|_1$, and $\boldsymbol{C}\boldsymbol{x} \geq \boldsymbol{1}$. Note that if we have $\tilde{\boldsymbol{x}}$ such that $\|\tilde{\boldsymbol{x}}\|_1 \leq \text{OPT}$ and $\boldsymbol{C}\tilde{\boldsymbol{x}} \geq (1-\varepsilon)\boldsymbol{1}$, then $\boldsymbol{x} = \tilde{\boldsymbol{x}}/(1-\varepsilon)$ is a $(1+2\varepsilon)$-approximation.

2.3 Quantum Complexity Model

To analyze the performance of our quantum algorithms, we adopt a standard quantum query model, consistent with recent literature on quantum algorithms for optimization and linear algebra [AW23; AG24]. In this model, the input data of the linear program (specifically the constraint matrix $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ and the vector $\boldsymbol{b} \in \mathbb{R}^n$) is not assumed to be stored in active memory (RAM). Instead, we access the constraints through queries to a quantum oracle.

The fundamental operation we consider is the verification of a single constraint against a candidate solution vector. For any given classical vector $\boldsymbol{x} \in \mathbb{R}^d$, we assume access to a *feasibility oracle*, denoted $O_{\boldsymbol{x},\varepsilon}$. This oracle acts on the indices of the n constraints and identifies whether a specific constraint

is violated by $\boldsymbol{x}$. Formally, the oracle is a unitary transformation that acts on the standard basis as follows:

$$O_{\boldsymbol{x},\varepsilon} : |i\rangle|z\rangle \mapsto |i\rangle|z \oplus \boldsymbol{v}_i^{\varepsilon}(\boldsymbol{x})\rangle \quad \forall i \in [n], z \in \{0,1\}$$

where $\oplus$ denotes addition modulo 2, and $\boldsymbol{v}_i^{\varepsilon}(\boldsymbol{x})$ is a binary indicator of violation:

$$\boldsymbol{v}_i^{\varepsilon}(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \langle \boldsymbol{A}_{i:}, \boldsymbol{x} \rangle > \boldsymbol{b}_i + \varepsilon \quad \text{(constraint } i \text{ is violated)} \\ 0 & \text{if } \langle \boldsymbol{A}_{i:}, \boldsymbol{x} \rangle \leq \boldsymbol{b}_i + \varepsilon \quad \text{(constraint } i \text{ is satisfied)} \end{cases}$$

The primary metric for the complexity of our quantum algorithms is the *quantum query complexity*, defined as the number of calls made to oracles of this type.

The implementation of the oracle $O_{\boldsymbol{x},\varepsilon}$ for a given $\boldsymbol{x}$ requires the ability to compute the inner product $\langle \boldsymbol{A}_{i:}, \boldsymbol{x} \rangle$ and compare it to $\boldsymbol{b}_i + \varepsilon$. This, in turn, requires access to the data of the i-th row of $\boldsymbol{A}$ and the i-th entry of $\boldsymbol{b}$. We therefore model each query to $O_{\boldsymbol{x},\varepsilon}$ as a single *row query* to the pair $(\boldsymbol{A}, \boldsymbol{b})$, which provides the necessary information to evaluate the i-th constraint.

We will use quantum query access to $(\boldsymbol{A}, \boldsymbol{b})$ to sample the constraints faster than in the classical query model. The execution of the classical LP solver on small, sampled sub-problems, is performed on a classical computer. This hybrid model allows us to precisely isolate the source of quantum speedup, which, in our framework, stems from the ability to find a violated constraint (an index i for which $\boldsymbol{v}_i^{\varepsilon}(\boldsymbol{x}) = 1$) more efficiently than is possible classically.

3 The Iterative Sparsification Framework

In this section, we present the general sparsification framework. Formally, our goal is to solve the following problem. In Section 4 we present how to reduce LP problems to Problem 1.

Problem 1. *Let $\varepsilon \geq 0$, $\mu > 0$, and suppose we have a linear program $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ with optimal value* OPT. *In the* low-violation solution set problem, *we must find a set $\{\boldsymbol{x}_j\}_{j=1}^{T}$ such that:*

1. *$\langle \boldsymbol{c}, \boldsymbol{x}_j \rangle \geq$ OPT for all $j \in [T]$;*
2. *For every constraint $i \in [n]$, the set of solutions violating it is small:*

$$|\{j \in [T] : \langle \boldsymbol{A}_{i:}, \boldsymbol{x}_j \rangle > \boldsymbol{b}_i + \varepsilon\}| < \mu \cdot T \,.$$

We show that it is possible to solve Problem 1 by iteratively using a low-violation oracle (Definition 1). The central result of this section is the following.

Theorem 3. *Consider a linear program of the form*

$$\max_{\boldsymbol{x}} \langle \boldsymbol{c}, \boldsymbol{x} \rangle, \qquad \text{subject to } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}.$$

Assume we are given a $(\varepsilon, \mu/3)$-low-violation oracle with running time $\mathcal{T}$, then in $T = O\left(\frac{\log n}{\mu}\right)$ iterations, we are able to compute a solution to Problem 1 $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T\}$. Moreover, each iteration runs in time $O(\mathrm{nnz}(\boldsymbol{A}) + \mathcal{T})$.

Based on this, we can now focus exclusively on efficiently implementing a low-violation oracle. To do so it suffices to solve an LP on a sub-sampled set of constraints. To ensure that this procedure yields a low-violation oracle we need to sample sufficiently many constraints (the number depends on the output space), and we need to sample them with appropriate probabilities. It turns out that these probabilities are exactly those used by the MWU routine. For the quantum implementation of our algorithms, we will leverage the following structural property of these weights.

Claim 1. *In iteration t, we have $\boldsymbol{p}_t \propto 2^{\sum_{\tau \in [t]} \boldsymbol{v}^{\varepsilon}(\boldsymbol{x}_\tau)}$, where $\boldsymbol{v}_i^{\varepsilon}(\boldsymbol{x})$ as 1 if $\langle \boldsymbol{A}_{i:}, \boldsymbol{x} \rangle > \boldsymbol{b}_i + \varepsilon$, and 0 otherwise.*

3.1 The Abstract Framework

We would like to sample from a set of n constraints, a set of r constraints with $r \ll n$ such that a solution on these r constraints gives us a solution of the LP. Such a one-shot sparsification technique is unlikely to exists as mentioned in the introduction. A simpler problem is to find solutions $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$ such that any constraint is violated by a small portion of $\{\boldsymbol{x}_j\}_{j\in[T]}$. Formally, instead of sparsifying the LP directly, we want to solve the following problem.

Problem 1. *Let $\varepsilon \geq 0$, $\mu > 0$, and suppose we have a linear program $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ with optimal value* OPT*. In the* low-violation solution set problem*, we must find a set $\{\boldsymbol{x}_j\}_{j=1}^T$ such that:*

1. *$\langle \boldsymbol{c}, \boldsymbol{x}_j \rangle \geq$ OPT for all $j \in [T]$;*
2. *For every constraint $i \in [n]$, the set of solutions violating it is small:*

$$|\{j \in [T] : \langle \boldsymbol{A}_{i:}, \boldsymbol{x}_j \rangle > \boldsymbol{b}_i + \varepsilon\}| < \mu \cdot T\,.$$

First, we remark that Problem 1 is easier than finding an exact solution of the LP or even an ε-approximation. Indeed, if $\boldsymbol{x}$ is an exact solution of the linear program, then $\{\boldsymbol{x}\}$ is a solution to Problem 1. Also, if $\boldsymbol{x}$ is an ε-approximation, then $\{\boldsymbol{x}\}$ is a solution to Problem 1.

Problem 1 is easier since we accept $T \geq 2$. With $\mu < 1$, we should have $T \geq \mu^{-1}$. We will prove we can construct a set of solutions with $T = O(\frac{\log n}{\mu})$. While it could be possible, it is unlikely for one to be able to find the T solutions without them being correlated. Meaning, we select the sets of constraints without computing a solution in between. Assume one seeks an exact solution to the LP using an exact solver. In order to create the T solutions, one may use the exact solver on sampled constraints. Using Lemma 18 we can upper bound the average number of violated constraint, however this does not mean anything on the probability of a specific constraint to be violated. In fact, constraints from the basis are far more likely to be violated. Instead of trying to find the set of T solutions in a one-shot fashion, we build them iteratively using the MWU framework.

Theorem 4. *Let $f : \Delta_n \mapsto \{0,1\}^n$ be a (randomized) routine which given any probability distribution $\boldsymbol{p} \in \Delta_n$, it outputs a vector $\boldsymbol{v} \in \{0,1\}^n$ such that $\langle \boldsymbol{p}, \boldsymbol{v} \rangle \leq \mu$. Then one can provide a sequence $\{\boldsymbol{p}_t\}_{t=1}^T$ such that the corresponding outputs $\{\boldsymbol{v}_t\}_{t=1}^T$ satisfy*

$$\max\left\{\frac{1}{T}\sum_{t=1}^{T} \boldsymbol{v}_t\right\} \leq 3\mu,$$

where $T = O(\frac{\log n}{\mu})$.

We can solve Problem 1 using Theorem 4, given any probability distribution $\boldsymbol{p} \in \Delta_n$, we need to find a solution $\boldsymbol{x}$ such that $\langle \boldsymbol{p}, \boldsymbol{v}^\varepsilon(\boldsymbol{x}) \rangle \leq \mu$ where $\boldsymbol{v}_i^\varepsilon(\boldsymbol{x})$ is 1 if $\langle \boldsymbol{A}_{i:}, \boldsymbol{x} \rangle > \boldsymbol{b}_i + \varepsilon$ and 0 otherwise.

3.2 Low-Violation Oracles

Low-violation oracles are used inside our MWU framework. Given any probability distribution, we seek to find $\boldsymbol{x} \in \mathbb{R}^d$ such that

$$\begin{cases} \langle \boldsymbol{c}, \boldsymbol{x} \rangle & \geq \text{OPT}; \\ \langle \boldsymbol{p}, \boldsymbol{v}^\varepsilon(\boldsymbol{x}) \rangle & \leq \mu. \end{cases} \tag{4}$$

Where $\boldsymbol{v}_i^\varepsilon(\boldsymbol{x})$ is 1 if $\langle \boldsymbol{A}_{i:}, \boldsymbol{x} \rangle > \boldsymbol{b}_i + \varepsilon$ and 0 otherwise.

Definition 1. *A (ε, μ)-*low-violation oracle *is an oracle such that given any probability distribution $\boldsymbol{p} \in \Delta_n$ outputs a solution $\boldsymbol{x}$ such that $\langle \boldsymbol{p}, \boldsymbol{v}^\varepsilon(\boldsymbol{x}) \rangle \leq \mu$.*

We can prove Theorem 3 using iteratively a low-violation oracle.

Proof of Theorem 3. The proof will rely on the MWU method, more precisely on Theorem 4. Theorem 4 provides the sequence $\{\boldsymbol{p}_t\}_{t=1}^T$ such that, if at iteration t we are able to compute $\boldsymbol{v}_t$ with $\langle \boldsymbol{p}_t, \boldsymbol{v}_t \rangle \leq \mu$, then $\sum_{t=1}^T \boldsymbol{v}_t \leq 3T\mu\mathbf{1}$. Given any $\boldsymbol{p}_t$, we use a (ε, μ)-low-violation oracle to compute $\boldsymbol{x}_t$ such that $\langle \boldsymbol{p}_t, \boldsymbol{v}^\varepsilon(\boldsymbol{x}_t) \rangle \leq \mu$. Hence, after T iterations, we have $\max\left\{\sum_{t=1}^T \boldsymbol{v}^\varepsilon(\boldsymbol{x}_t)\right\} \leq 3T\mu$. Thus, with $T = O(\log n/\mu)$ calls to a $(\varepsilon, \mu/3)$-low-violation oracle, it is possible to solve Problem 1. □

4 Applications

In this section, we focus on corollaries of Theorem 3. We first develop the techniques from [Cla95; Ass25] to design (ε, μ)-low-violation oracles. We then show that we can extend our analysis to find low-precision solutions for general LPs, and how to perform *width* reduction for mixed packing and covering problems. We present the proofs in Section C.

4.1 Constructing (ε, μ)-Low-Violation Oracles

In this subsection, we construct two (ε, μ)-low-violation oracles. Both of them rely on an inner LP solver, which is used on a sparse subset of constraints. We present low-violation oracles that can be decomposed of three steps:

1. *Sparsify:* Construct a sparse LP by selecting a small subset of constraints using the probability vector $\boldsymbol{p}$.
2. *Solve:* Use the solver to find a solution to the sparse LP.
3. *Verify:* Verify the solution violates constraints with small total probability.

When the solver used in the subroutine is an exact solver, we use Clarkson's sampling lemma (Lemma 18 from [Cla95]) to bound the quantity of sampled constraint required. When we use any other solver that has a sparse output set, we bound the number of sampled constraint using a technique similar to Assadi [Ass25].

Our first (ε, μ)-low-violation oracle arises from using an exact solver on a small set of constraints. This result is a corollary of Clarkson's sampling lemma.[4]

Lemma 5 (Corollary of [Cla95])**.** *Consider a linear program of the form* $\max\langle \boldsymbol{c}, \boldsymbol{x}\rangle$ *subject to* $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$*, where* $\boldsymbol{A} \in \mathbb{R}^{n\times d}$*. For a parameter* μ*, given a probability distribution* $\boldsymbol{p} \in \Delta_n$*, the following procedure is a* (ε, μ)*-low-violation oracle.*

1. Sparsify: *Consider* S *a subset of constraints such that every* i *is in* S *independently with probability at least* $\min\{2d/\mu \cdot \boldsymbol{p}_i, 1\}$*.*
2. Solve: *Let* $\boldsymbol{x}$ *be the exact solution of the LP subject to constraint set* S*.*
3. Verify: *If* $\langle \boldsymbol{p}, \boldsymbol{v}^{\varepsilon}(\boldsymbol{x})\rangle > \mu$ *go back to* 1.*, else return* $\boldsymbol{x}$*.*

Moreover, this procedure takes expected time $O(\mathrm{nnz}(\boldsymbol{A}) + \mathcal{T}_{\mathrm{exact}}(d, 2d/\mu))$*, where* $\mathcal{T}_{\mathrm{exact}}(d, n)$ *is the time required to find an exact solution of a linear program with* d *variables and* n *constraints.*

Similarly, we use the same technique as used in [Ass25] to prove that as long as the solver has a small output space, the number of sampled constraints does not have to be large.

Lemma 6. *Consider a linear program of the form*

$$\max\langle \boldsymbol{c}, \boldsymbol{x}\rangle, \qquad \text{subject to } \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}.$$

For $\varepsilon \geq 0$ *and* $\mu > 0$*, assume we have an* ε*-approximate solver* $\mathcal{A}$ *such that the number of distinct outputs of* $\mathcal{A}$ *is upper bounded by* N*. The following procedure describes a* (ε, μ)*-low-violation solver and require one call to* $\mathcal{A}$ *with high-probability.*

1. Sparsify: *Consider* S *a subset of constraints such that every* i *is in* S *independently with probability at least* $\min\{\frac{\log(Nn)}{\mu} \cdot \boldsymbol{p}_i, 1\}$*.*
2. Solve: *Compute* $\boldsymbol{x}$ *using* $\mathcal{A}$ *on the LP made of constraints from* S*.*
3. Verify: *If* $\langle \boldsymbol{p}, \boldsymbol{v}^{\varepsilon}(\boldsymbol{x})\rangle > \mu$ *go back to* 1.*, else return* $\boldsymbol{x}$*.*

Moreover, this procedure takes expected time $O(\mathrm{nnz}(\boldsymbol{A}) + \mathcal{T}_{\mathrm{approx}}(d, \frac{\log(Nn)}{\mu}, \varepsilon))$*, where* $\mathcal{T}_{\mathrm{approx}}(d, n, \varepsilon)$ *is the time required to find an* ε*-approximate solution of a linear program with* d *variables and* n *constraints.*

[4] See Lemma 18 for the definition of the sampling lemma.

4.2 Solving LPs to Low-Precision

In this subsection, we present how to obtain an ε-approximation using multiple calls to an exact solver on a sparse subset of constraints.

Consider a linear program $\max_{\boldsymbol{x}\in\mathcal{D}}\langle \boldsymbol{c}, \boldsymbol{x}\rangle$ subject to $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$. Let OPT denote the optimal value for that LP. We would like to obtain an ε-approximation, that means $\boldsymbol{x}$ such that $\langle \boldsymbol{c}, \boldsymbol{x}\rangle \geq \text{OPT}$ and $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}+\varepsilon\boldsymbol{1}$. First, we reduce this problem to a problem of the form Problem 1. For a solver $\mathcal{A}$ whose outputs are in $\mathcal{D}$, we consider the largest violation possible in $\mathcal{D}$: $V_{\max} := \max_{\boldsymbol{x}\in\mathcal{D}} \max_{i\in[n]}(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})_i$.

Remark. *We can compare $V_{\max}$ to the* width *parameter in the Plotkin-Shmoys-Tardos (PST) framework, and other classical techniques using MWU [PST91; AHK12]. In those techniques, the width ρ is defined as $\max_{\boldsymbol{x}\in\mathcal{D}} \|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_\infty$. Hence, we always have $V_{\max} \leq \rho$. ρ not only captures the largest violation, but also the largest slack, that means, if a solution can "largely satisfy" a constraint, the width is large. We develop this in Section 4.3.*

Claim 2. *For $\varepsilon > 0$, and a linear program of the form (3). Let $V_{\max}$ be the largest violation of $\mathcal{D}$, $V_{\max} := \max_{\boldsymbol{x}\in\mathcal{D}} \max_{i\in[n]} (\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})_i$. Given an $(\varepsilon, \varepsilon/3V_{\max})$-low-violation oracle $\mathcal{A}$. Within $O(\frac{V_{\max}}{\varepsilon}\log n)$ iterations, each one requiring one call to $\mathcal{A}$ and $\text{nnz}(\boldsymbol{A})$ time, it is possible to find $\boldsymbol{x}$ such that $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} + 2\varepsilon\boldsymbol{1}$.*

Proof. Using Theorem 3, it is possible to use only $T = O(\frac{V_{\max}}{\varepsilon}\log n)$ calls to an $(\varepsilon, \varepsilon/3V_{\max})$-low-violation oracle to solve Problem 1 with parameters ε and $\mu = \varepsilon/V_{\max}$.

Assume we have a solution $\{\boldsymbol{x}_j\}_{j\in[T]}$ to Problem 1 with those parameters. For a constraint i, let $V_i \subseteq [T]$ be the indices of solutions violating constraint i: $\{j \in [T] : \langle \boldsymbol{A}_{i:}, \boldsymbol{x}_j\rangle \geq \boldsymbol{b}_i + \varepsilon\}$. We have

$$\sum_{j\in[T]} \langle \boldsymbol{A}_{i:}, \boldsymbol{x}_j\rangle - \boldsymbol{b}_i = \sum_{j\in V_i} \langle \boldsymbol{A}_{i:}, \boldsymbol{x}_j\rangle - \boldsymbol{b}_i + \sum_{j\in[T]\setminus V_i} \langle \boldsymbol{A}_{i:}, \boldsymbol{x}_j\rangle - \boldsymbol{b}_i \leq \sum_{j\in V_i} V_{\max} + \sum_{j\in[T]\setminus V_i} \varepsilon \leq 2\varepsilon\cdot T.$$

Hence, $\bar{\boldsymbol{x}} := \frac{1}{T}\sum_{j\in[T]} \boldsymbol{x}_j$ is 2ε-feasible. Moreover, we have $\langle \boldsymbol{c}, \bar{\boldsymbol{x}}\rangle \geq \text{OPT}$ since for any $j \in [T]$, $\langle \boldsymbol{c}, \boldsymbol{x}_j\rangle \geq \text{OPT}$, hence $\bar{\boldsymbol{x}}$ is a 2ε-approximation. □

Using an exact solver and Lemma 5, we create an $(\varepsilon/2, \varepsilon/6V_{\max})$-low-violation oracle that we use with Claim 2. This gives us the following theorem.

Theorem 7. *For a linear program of the form (3) with largest violation*

$$V_{\max} := \max_{\boldsymbol{x}\in\mathcal{D}} \max_{i\in[n]} (\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})_i \ ,$$

there exists a randomized algorithm that runs in expected time

$$O\left(\frac{V_{\max}}{\varepsilon}\log n\left(\text{nnz}(\boldsymbol{A}) + \mathcal{T}_{\text{exact}}(d, 6d\frac{V_{\max}}{\varepsilon})\right)\right),$$

where $\mathcal{T}_{\text{exact}}(d, n)$ is the running time of an exact solver on d variables and n constraints.

4.3 Solving Mixed Packing and Covering LPs to Low-Precision

For a linear program, the largest violation $V_{\max} := \max_{\boldsymbol{x}\in\mathcal{D}} \max_{i\in[n]} (\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b})_i$ corresponds to a single-sided width. Generally, the width is $\rho := \max_{\boldsymbol{x}\in\mathcal{D}} \|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_\infty$. There are linear programs for which $V_{\max} \ll \rho$, for instance, pure covering problems. In a pure covering LP, the width ρ can be as large as d, while our single-sided width is constant, equal to 1. On the other hand, for pure packing LPs, the standard definition of the width and our single-sided width are equal.

We show how we can leverage this different notion of width to achieve better guarantees for mixed packing and covering problems with numerous dense covering constraints.

Problem 2. *For positive integers n_c, n_p, and d. Consider $\boldsymbol{C} \in [0,1]^{n_c \times d}$, and $\boldsymbol{P} \in [0,1]^{n_p \times d}$.*[5] *We denote by r_c the row-sparsity of $\boldsymbol{C}$, and r_p the row-sparsity of $\boldsymbol{P}$.*

In the mixed packing and covering program *problem, we seek $\boldsymbol{x} \in [0,1]^d$ such that $\boldsymbol{P}\boldsymbol{x} \leq \mathbf{1}$, and $\boldsymbol{C}\boldsymbol{x} \geq \mathbf{1}$, or certify that there is none.*

A $(1+\varepsilon)$-approximation of Problem 2 is a vector $\boldsymbol{x} \in [0,1]^d$ such that $\boldsymbol{C}\boldsymbol{x} \geq \mathbf{1}$, and $\boldsymbol{P}\boldsymbol{x} \leq (1+\varepsilon)\mathbf{1}$. Given a problem of the form 2, the one-sided width is at most r_p.

Reducing the one-sided width. We can reduce the one-sided width by keeping some constraints. In the classical PST framework, a lot of iterations are performed, each one requires to solve a linear optimization problem within $\mathcal{D}$. Since a lot of iterations are required, the structure of $\mathcal{D}$ should be "simple". This comes at the cost of a potentially large width. In our case, $\mathcal{D}$ is the output set of the solver used in each call to the low-violation oracle. Since we are performing few iterations, it can be beneficial to add some constraints that are kept between iterations. In particular, we can add the constraints with large one-sided width to it. For our mixed packing and covering problem, we add the packing constraints, we consider:

$$\mathcal{D} := \{\boldsymbol{x} \in [0,1]^d : \boldsymbol{P}\boldsymbol{x} \leq \mathbf{1}\}.$$

Since the packing constraints are captured by the solution space, we can simulate solving a linear program with constant one-sided width, and solution space $\mathcal{D}$.

Consider a solver $\mathcal{A}$ such that given any subset $S \subseteq [n_c]$ of constraints, $\mathcal{A}$ finds a $1+\varepsilon$ approximation $\boldsymbol{x}$ of the mixed packing and covering program defined by $\boldsymbol{C}_S$ and $\boldsymbol{P}$. We have for any packing constraint: $\langle \boldsymbol{P}_{i:}, \boldsymbol{x}\rangle - 1 \leq \varepsilon$, and for any covering constraint: $1 - \langle \boldsymbol{C}_{i:}, \boldsymbol{x}\rangle \leq 1$. Hence, the one-sided width of such a solver is 1.

Reducing the cardinality of the output space. The next difficulty we have to handle is using Lemma 6. Lemma 6 requires the output space to contain few elements. Assadi leverages the small cardinality of the vertex cover problem in a bipartite graph. For mixed packing and covering problems, the cardinality of the solution space can be large, however we can "discretize" the output space. Assume $\boldsymbol{x}$ is a $(1+\varepsilon)$-approximation, then any vector $\boldsymbol{y}$ such that $\boldsymbol{x} \leq \boldsymbol{y} \leq (1+\varepsilon)\boldsymbol{x}$ coordinate-wise is a $(1+4\varepsilon)$-approximation. Hence, we can reduce the output set to vectors of the form $\boldsymbol{x}_i = C(1+\varepsilon)^{k_i}$ for some integer k_i, and a small value C. We give the size of this ε-net in the following claim.

Claim 3. *For $0 < \varepsilon \leq 1$. Given $\boldsymbol{x}$, a $1+\varepsilon$ approximation of a mixed packing and covering problem of the form Problem 2, it is possible to find $\tilde{\boldsymbol{x}}$ a $1+4\varepsilon$ approximation of the same mixed packing and covering program such that $\tilde{\boldsymbol{x}} \in \mathcal{S}$ where $|\mathcal{S}| \leq \lceil 2 + 4\frac{\log(r_p/\varepsilon)}{\varepsilon}\rceil^d$.*

Note that it is not necessary to have a solver that outputs in $\mathcal{S}$. This is an artifact of the analysis. With our two ingredients, we can obtain the following theorem.

Theorem 8. *Given a mixed packing and covering program 2, it is possible to find with high-probability a $1+O(\varepsilon)$-approximation $\boldsymbol{x}$ or certify infeasibility in time*

$$\widetilde{O}\left(\frac{1}{\varepsilon}\left(\operatorname{nnz}(\boldsymbol{C}) + \mathcal{T}_{\text{mixed}}(d, \operatorname{nnz}(\boldsymbol{P}), O\left(\frac{dr_c}{\varepsilon}\log(2+4\frac{\log\frac{r_p}{\varepsilon}}{\varepsilon})\right), \varepsilon)\right)\right),$$

where $\mathcal{T}_{\text{mixed}}(d, \operatorname{nnz}(\boldsymbol{P}), \operatorname{nnz}(\boldsymbol{C}), \varepsilon)$ is the time required to compute a $1+\varepsilon$-approximation of a mixed packing and covering problem with at most $\operatorname{nnz}(\boldsymbol{P})$ (resp. $\operatorname{nnz}(\boldsymbol{C})$) non-zeros inside the packing (resp. covering) constraint matrix, and d variables.

We can use state-of-the-art solvers for mixed packing and covering programs such as [CQ18; BSW19; Qua19].

[5] This assumption can be made without loss of generality by adding poly-logarithmic number of constraints, see Appendix B of [BSW19].

5 Speedups via Quantum Sampling

In the classical regime, the running time is dominated by the time needed to find violated constraints. Given a quantum query access oracle and Grover search we can improve the running time of the algorithm. Given a word $\boldsymbol{x} \in \{0,1\}^n$, the hamming weight of $\boldsymbol{x}$ is its number of non-zero entries. If we have query access to a word $\boldsymbol{x}$ with known hamming weight t, Grover search finds an i such that $\boldsymbol{x}_i = 1$ in time $O(\sqrt{n/t})$. However, if the hamming weight of $\boldsymbol{x}$ is unknown, then Grover search becomes probabilistic. Instead of using Grover search to find the violated constraints, we will use it to sample from the probability vector $\boldsymbol{p}$ at each iteration. One bottleneck of our approach is that at each iteration, we increase the cost of querying from $\boldsymbol{p}$. At iteration t, we have t solutions to check violation for, hence we need to perform t row-queries to $(\boldsymbol{A}, \boldsymbol{b})$. We strongly rely on Lemma 9 to sample the constraints using queries to the binary violation oracle.

Lemma 9 (Claim 3, [AW23])**.** *Assume we have query access to a list of probabilities $(\boldsymbol{q}_i)_{i\in[n]}$, each of which is described with $\widetilde{O}(1)$ bits of precision. Then there is a quantum algorithm that samples a subset $S \subseteq [n]$, such that S contains every i independently with probability $\boldsymbol{q}_i$, in expected time $\widetilde{O}(\sqrt{n\sum_i \boldsymbol{q}_i})$ and using a QRAM of $\widetilde{O}(\sqrt{n\sum_i \boldsymbol{q}_i})$ bits.*

We give the proofs in Section D.

5.1 Sampling the Constraints

Consider now a quantum low-violation oracle. Our low-violation oracles sample constraints using quantum queries to $(\boldsymbol{A}, \boldsymbol{b})$, and then use a classical solver on the sampled constraints.

A low-violation oracle receives a vector $\boldsymbol{p}$ on which it has quantum query access. Assume for some value $s > 1$, the low-violation oracle requires to sample every constraint i independently with probability $s\boldsymbol{p}_i$. Using Lemma 9, we can perform this sampling step with $\widetilde{O}(\sqrt{ns})$ queries to $\boldsymbol{p}$. In the classical regime, we had access to $\boldsymbol{p}$. In the quantum regime, we only have query access to the binary violation vector $\boldsymbol{v}^\varepsilon$.

Instead of sampling from $s\boldsymbol{p}$, we will sample from $s\boldsymbol{q}$ where $\boldsymbol{q}$ satisfies $\boldsymbol{p} \le \boldsymbol{q} \le 2\boldsymbol{p}$. This ensures that on average, $2s$ constraints are sampled, hence the number of queries to $\boldsymbol{p}$ and to $\boldsymbol{q}$ in Lemma 9 remains the same up to a constant factor.

We now describe how we obtain our query access to $\boldsymbol{q}$. Consider $\boldsymbol{w}$ the vector proportional to $\boldsymbol{p}$ but not normalized. By Claim 1, at iteration t, we have $\boldsymbol{w}_i = 2^{\sum_{\tau=1}^{t} \boldsymbol{v}_i^\varepsilon(\boldsymbol{x}_\tau)}$. A query to $\boldsymbol{w}$ can be made through t queries to $\boldsymbol{v}^\varepsilon$. Consider W, the normalization factor such that $\boldsymbol{p}_i = \boldsymbol{w}_i/W$. If W is known, a query to $\boldsymbol{p}$ can be made through a query to $\boldsymbol{w}$, which can be done with t queries to $\boldsymbol{v}^\varepsilon$. Since each query to $\boldsymbol{v}^\varepsilon$ is performed using a row-query to $(\boldsymbol{A}, \boldsymbol{b})$, if W is known, a query to $\boldsymbol{p}$ at iteration t would require t row-queries to $(\boldsymbol{A}, \boldsymbol{b})$.

An important issue is we do not have easy access to W. However, if we are able to maintain a constant factor estimation of W throughout the iterations, we can create query access to a vector $\boldsymbol{q}$ such that $\boldsymbol{p} \le \boldsymbol{q} \le 2\boldsymbol{p}$. Indeed, assume we have $\widetilde{W} \in [\frac{1}{2}W, W]$, then a query to $\boldsymbol{q} = \boldsymbol{w}/\widetilde{W}$ cost the same as a query to $\boldsymbol{p}$ when W is known, moreover $\boldsymbol{p} \le \boldsymbol{q} \le 2\boldsymbol{p}$.

In order to maintain the estimation of W, notice that between two consecutive iterations, the sum of weights stays within a constant factor from its previous value. Hence, we use the constant approximation factor of the previous sum of weights to estimate its new value. This can be done by sampling a constant quantity of constraints using Lemma 9 and Chernoff bound.

Claim 4. *For an integer $n > 0$, a non-negative vector $\boldsymbol{w} \in \mathbb{R}^n$, a real $W > 0$, a constant $C > 1$, and a probability $p \in [0,1]$. Assume $\widetilde{W} \le \|\boldsymbol{w}\|_1 \le C\widetilde{W}$, then Procedure 2 outputs $\widetilde{W}'$ such that with probability over $1-p$, $\widetilde{W}' \le \|\boldsymbol{w}\|_1 \le 2\widetilde{W}'$ using*

$$\widetilde{O}(\sqrt{n}\log\frac{1}{p})$$

queries to $\boldsymbol{w}$.

Now that we are able to maintain a constant factor approximation $\widetilde{W}$ of the sum of weights throughout iterations, we want to ensure that the set we are sampling does not have to many constraints. Sampling from $\boldsymbol{q} = s\boldsymbol{w}/\widetilde{W}$ ensures on average less than $2s$ constraints are sampled, and the set obtained through Lemma 9 satisfies the probability requirement of Lemmas 5 and 6. However, the set S may have a lot more constraints. We use the following claim to ensure this is not possible with high-probability.

Claim 5. *For an integer $n > 0$, a non-negative vector $\boldsymbol{w} \in \mathbb{R}^n$, a real $\widetilde{W} > 0$, an integer $s \geq 6$, and a probability $p \in [0, 1]$. Assume $\widetilde{W} \leq \|\boldsymbol{w}\|_1 \leq 2\widetilde{W}$. Using $\widetilde{O}(\sqrt{ns}\log\frac{1}{p})$ queries to $\boldsymbol{w}$, Procedure 1 outputs a set S such that with probability over $1 - p$:*

1. *S contains every i independently with probability greater than $\min\{s\frac{\boldsymbol{w}_i}{\|\boldsymbol{w}\|_1}, 1\}$;*
2. *S is small, $|S| \leq 4s$.*

First, notice that since we need an algorithm that works with high probability, and we use Claims 4 and 5 once per iteration, we can assume p is sufficiently small so that the algorithm does not fail because of those procedures with high probability. Moreover, Claim 4 uses far fewer queries to $\boldsymbol{w}$ (hence to $(\boldsymbol{A}, \boldsymbol{b})$) compared Claim 5, hence maintaining a constant factor estimation of normalization factor through the iterations is essentially "free".

5.2 Quantum Speedups for Exact Solvers

We need another procedure for exact solvers. We need to verify the feasibility of the solution we add to the current set of solutions. If it is feasible, we return it, else the algorithm should continue. We use Grover search on $\boldsymbol{v}$ where $\boldsymbol{v}_i(\boldsymbol{x})$ is 1 if $\boldsymbol{x}$ violates constraint i, 0 otherwise. We can use the following lemma from [Wol23].

Claim 6 (Exercise 6.c. p60 from [Wol23])**.** *Given query access to $\boldsymbol{x} \in \{0, 1\}^n$, it is possible with $O(\sqrt{n\log\frac{1}{p}})$ queries to $\boldsymbol{x}$ to find an index $i \in \{0, \ldots, n-1\}$ such that $\boldsymbol{x}_i = 1$ or output $\boldsymbol{x} = \boldsymbol{0}$ with probability over $1 - p$.*

Since we are performing a linear number of iterations, and we are interested in time complexity up to poly-logarithmic factors, we assume p is sufficiently small so that Claim 6 does not fail throughout iterations with high probability.

Theorem 10. *For a linear program of the form 3 with n constraints and d variables, Algorithm 1 outputs the optimum with high probability using on average $\widetilde{O}(\sqrt{n}d^3)$ row-queries to $(\boldsymbol{A}, \boldsymbol{b})$, and time*

$$\widetilde{O}\left(d(\sqrt{n}d^2r + \mathcal{T}_{\text{exact}}(d, 24d^2))\right),$$

where r is the row-sparsity of $\boldsymbol{A}$, and $\mathcal{T}_{\text{exact}}(d, n)$ is the time required to find an exact solution on a set of n constraints with d variables.

5.3 Quantum Speedups for Mixed Packing and Covering LPs

For mixed packing and covering problems, we can perform the same trick to sample constraints using row-queries to $\boldsymbol{C}$. We provide a quantum version of Theorem 8.

Theorem 11. *Given a mixed packing and covering problem 2, it is possible to find a $1 + O(\varepsilon)$-approximation $\boldsymbol{x}$ or certify infeasibility with high probability using $\widetilde{O}(\sqrt{n_c\frac{d}{\varepsilon}}\frac{1}{\varepsilon^2})$ row-queries to $\boldsymbol{C}$, moreover the algorithm runs in time*

$$\widetilde{O}\left(\frac{1}{\varepsilon}\left(\sqrt{n_c\frac{d}{\varepsilon}}\frac{r_c}{\varepsilon} + \mathcal{T}_{mixed}(d, \text{nnz}(\boldsymbol{P}), O\left(\frac{dr_c}{\varepsilon}\log(\frac{\log\frac{r_p}{\varepsilon}}{\varepsilon})\right), \varepsilon)\right)\right),$$

where $\mathcal{T}_{mixed}(d, \text{nnz}(\boldsymbol{P}), \text{nnz}(\boldsymbol{C}), \varepsilon)$ is the time required to compute a $1 + \varepsilon$ approximation of a linear program with at most $\text{nnz}(\boldsymbol{P})$ (resp. $\text{nnz}(\boldsymbol{C})$) non-zeros inside the packing (resp. covering) matrix, and d variables.

Algorithm 1 Quantum Clarkson

Input: Row-query access to $\boldsymbol{A}$ and $\boldsymbol{b}$, and $\mathcal{A}$ an exact solver
Output: $\boldsymbol{x}$ a feasible solution

$X_0 \leftarrow \emptyset$
$p \leftarrow \frac{1}{32n\log n}\frac{1}{100n^2}$
$\widetilde{W}_0 \leftarrow n$
$s \leftarrow 6d^2$
for $t = 0$ to $T = 24d\log n$ **do**
    Define the query access to $\boldsymbol{w}(X)$ as $2^{\sum_{\boldsymbol{x}\in X}\boldsymbol{v}^0(\boldsymbol{x})}$
    $S_t \leftarrow \texttt{quantum\_sampling}(\boldsymbol{w}(X_t), s, \widetilde{W}_t, p)$ ▷ Using Procedure 1.
    $\boldsymbol{x} \leftarrow \mathcal{A}(S_t)$
    $X_{t+1} \leftarrow X_t \cup \{\boldsymbol{x}\}$
    $\widetilde{W}_{t+1} \leftarrow \texttt{estimation\_sum}(\boldsymbol{w}(X_{t+1}), \widetilde{W}_t, p)$ ▷ Using Procedure 2.
end for
for $\boldsymbol{x} \in X_T$ **do**
    if $\boldsymbol{x}$ is feasible **then** ▷ By invoking Claim 6.
        return $\boldsymbol{x}$
    end if
end for
return $\perp$

Algorithm 2 Quantum Mixed Packing and Covering LP Solver

Input: $\boldsymbol{P} \in [0,1]^{n_p\times d}$, and quantum query access to $\boldsymbol{C} \in [0,1]^{n_c\times d}$, $\varepsilon > 0$ and $\mathcal{A}$ a mixed packing and covering solver
Output: $\boldsymbol{x}$ a $1 + O(\varepsilon)$-approximation

$X_0 \leftarrow \emptyset$
$p \leftarrow \frac{1}{32n\log n}\frac{1}{100n^2}$
$\widetilde{W}_0 \leftarrow n_c$
$s \leftarrow 6\frac{d}{\varepsilon}\log\frac{\log(r_p/\varepsilon)}{\varepsilon}$
for $t = 0$ to $T = \frac{24}{\varepsilon}\log n_c$ **do**
    Define the query access to $\boldsymbol{w}(X)$ as $2^{\sum_{\boldsymbol{x}\in X}\boldsymbol{v}^\varepsilon(\boldsymbol{x})}$
    $S_t \leftarrow \texttt{quantum\_sampling}(\boldsymbol{w}(X_t), s, \widetilde{W}_t, p)$ ▷ Using Procedure 1.
    $\boldsymbol{C}_t \leftarrow$ the matrix $\boldsymbol{C}$ but only with rows in S_t
    $\boldsymbol{x} \leftarrow \mathcal{A}(\boldsymbol{P}, \boldsymbol{C}_t, \varepsilon)$
    $X_{t+1} \leftarrow X_t \cup \{\boldsymbol{x}\}$
    $\widetilde{W}_{t+1} \leftarrow \texttt{estimation\_sum}(\boldsymbol{w}(X_{t+1}), \widetilde{W}_t, p)$ ▷ Using Procedure 2.
end for
return $\sum_{\boldsymbol{x}\in X_T}\boldsymbol{x}/T$.

5.4 Quantum Speedups for Low-Precision Solvers

We provide two quantum versions of Theorem 7. For low-precision solvers, we can perform the same trick to sample constraints using row-queries to $\boldsymbol{A}$.

Theorem 12. *For a linear program of the form 3 with largest violation $V_{\max}$, Algorithm 3 finds an ε-approximation $\boldsymbol{x}$ with high-probability using $\widetilde{O}(\sqrt{nd\frac{V_{\max}}{\varepsilon}}\frac{V_{\max}^2}{\varepsilon^2})$ row-queries to $\boldsymbol{A}$, and running time*

$$\widetilde{O}\left(\frac{V_{\max}}{\varepsilon}\left(\sqrt{nd\frac{V_{\max}}{\varepsilon}}\frac{V_{\max}}{\varepsilon}r+\mathcal{T}_{\text{exact}}(d,24d\frac{V_{\max}}{\varepsilon})\right)\right),$$

where r is the row-sparsity of $\boldsymbol{A}$, and $\mathcal{T}_{\text{exact}}(d,n)$ is the running time of an exact solver on d variables and n constraints.

Assume we have a set of solutions $\{\boldsymbol{x}_j\}_{j=1}^t$, previously, our weight was an exponential on $\sum_{j\in[t]}\boldsymbol{v}^\varepsilon(\boldsymbol{x}_j)$. For low-precision solvers, it may be tempting to use a "continuous" version that is simpler to compute such as $\frac{1}{V_{\max}}\sum_{j\in[t]}(\boldsymbol{A}\boldsymbol{x}_j-\boldsymbol{b})$. A query to the weight vector would thus be implemented with one row-query to $(\boldsymbol{A},\boldsymbol{b})$. Unfortunately, we need to be careful about how much each weight can change from one iteration to the other. Both the MWU framework, and our ability to estimate the sum of weights requires at most a constant multiplicative change in the weights, hence we need to use the two-sided width $\rho:=\max_{\boldsymbol{x}\in\mathcal{D}}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_\infty$ instead of the largest violation $V_{\max}$ (i.e. one-sided width) used in Theorems 7 and 12.

Theorem 13. *For a linear program of the form 3 with width ρ, Algorithm 4 finds an ε-approximation $\boldsymbol{x}$ with high-probability using $\widetilde{O}(\sqrt{nd\frac{\rho}{\varepsilon}}\frac{\rho}{\varepsilon})$ row-queries to $\boldsymbol{A}$, and running time*

$$\widetilde{O}\left(\frac{\rho}{\varepsilon}\left(\sqrt{nd\frac{\rho}{\varepsilon}}r+\mathcal{T}_{\text{exact}}(d,24d\frac{\rho}{\varepsilon})\right)\right),$$

where r is the row-sparsity of $\boldsymbol{A}$, and $\mathcal{T}_{\text{exact}}(d,n)$ is the running time of an exact solver on d variables and n constraints.

Algorithm 3 Quantum Low-Precision Solver with One-Sided Width

Input: Query access to $\boldsymbol{A}$ and $\boldsymbol{b}$, and $\mathcal{A}$ an exact solver.
Output: $\boldsymbol{x}$ an ε-approximation.

$X_0\leftarrow\emptyset$
$p\leftarrow\frac{1}{32n\log n}\frac{1}{100n^2}$
$\widetilde{W_0}\leftarrow n$
$s\leftarrow 6d\frac{V_{\max}}{\varepsilon}$
for $t=0$ to $T=24\frac{V_{\max}}{\varepsilon}\log n$ **do**
  Define the query access to $\boldsymbol{w}(X)$ as $2^{\sum_{\boldsymbol{x}\in X}\boldsymbol{v}^\varepsilon(\boldsymbol{x})}$
  $S_t\leftarrow$ `quantum_sampling`$(\boldsymbol{w}(X_t),s,\widetilde{W_t},p)$ ▷ Using Procedure 1.
  $\boldsymbol{x}\leftarrow\mathcal{A}(S_t)$
  $X_{t+1}\leftarrow X_t\cup\{\boldsymbol{x}\}$
  $\widetilde{W_{t+1}}\leftarrow$ `estimation_sum`$(\boldsymbol{w}(X_{t+1}),\widetilde{W_t},p)$ ▷ Using Procedure 2.
end for
return $\sum_{\boldsymbol{x}\in X_T}\boldsymbol{x}/|X_T|$

Algorithm 4 Quantum Low-Precision Solver with Two-Sided Width

Input: Query access to $\boldsymbol{A}$ and $\boldsymbol{b}$, and $\mathcal{A}$ an exact solver.
Output: $\boldsymbol{x}$ an ε-approximation.

$X_0 \leftarrow \emptyset$
$p \leftarrow \frac{1}{32n\log n}\frac{1}{100n^2}$
$\widetilde{W}_0 \leftarrow n$
$s \leftarrow 6d\frac{\rho}{\varepsilon}$
for $t = 0$ to $T = 16\frac{\rho}{\varepsilon}\log n$ **do**
  Define the query access to $\boldsymbol{w}(X)$ as $2^{\frac{1}{\rho}(\boldsymbol{A}\sum_{\boldsymbol{x}\in X}\boldsymbol{x}-t\boldsymbol{b})}$
  $S_t \leftarrow \texttt{quantum\_sampling}(\boldsymbol{w}(X_t), s, \widetilde{W}_t, p)$ ▷ Using Procedure 1.
  $\boldsymbol{x} \leftarrow \mathcal{A}(S_t)$
  $X_{t+1} \leftarrow X_t \cup \{\boldsymbol{x}\}$
  $\widetilde{W}_{t+1} \leftarrow \texttt{estimation\_sum}(\boldsymbol{w}(X_{t+1}), \widetilde{W}_t/2, p)$ ▷ Using Procedure 2.
end for
return $\sum_{\boldsymbol{x}\in X_T}\boldsymbol{x}/\left|X_T\right|$

References

[AG24] Simon Apers and Sander Gribling. *Quantum Speedups for Linear Programming via Interior Point Methods*. Apr. 11, 2024. DOI: 10.48550/arXiv.2311.03215. arXiv: 2311.03215 [quant-ph]. URL: http://arxiv.org/abs/2311.03215. Pre-published.

[AHK12] Sanjeev Arora, Elad Hazan, and Satyen Kale. "The Multiplicative Weights Update Method: A Meta-Algorithm and Applications". In: *Theory of Computing* 8.6 (May 1, 2012), pp. 121–164. DOI: 10.4086/toc.2012.v008a006. URL: https://theoryofcomputing.org/articles/v008a006/.

[Ape+20] Joran van Apeldoorn, András Gilyén, Sander Gribling, and Ronald de Wolf. "Quantum SDP-Solvers: Better Upper and Lower Bounds". In: *Quantum* 4 (Feb. 14, 2020), p. 230. ISSN: 2521-327X. DOI: 10.22331/q-2020-02-14-230. arXiv: 1705.01843 [quant-ph]. URL: http://arxiv.org/abs/1705.01843.

[Ass25] Sepehr Assadi. "A Simple $(1-\varepsilon)$-Approximation Semi-Streaming Algorithm for Maximum (Weighted) Matching". In: *TheoretiCS* Volume 4 (Aug. 1, 2025), p. 13451. ISSN: 2751-4838. DOI: 10.46298/theoretics.25.16. arXiv: 2307.02968 [cs]. URL: http://arxiv.org/abs/2307.02968.

[AW23] Simon Apers and Ronald de Wolf. *Quantum Speedup for Graph Sparsification, Cut Approximation and Laplacian Solving*. May 8, 2023. DOI: 10.48550/arXiv.1911.07306. arXiv: 1911.07306 [quant-ph]. URL: http://arxiv.org/abs/1911.07306. Pre-published.

[BK15] András A. Benczúr and David R. Karger. "Randomized Approximation Schemes for Cuts and Flows in Capacitated Graphs". In: *SIAM Journal on Computing* 44.2 (Jan. 2015), pp. 290–319. ISSN: 0097-5397. DOI: 10.1137/070705970. URL: https://epubs.siam.org/doi/10.1137/070705970.

[Bra+21] Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. *Solving Tall Dense Linear Programs in Nearly Linear Time*. Aug. 22, 2021. DOI: 10.48550/arXiv.2002.02304. arXiv: 2002.02304 [cs]. URL: http://arxiv.org/abs/2002.02304. Pre-published.

[BSW19] Digvijay Boob, Saurabh Sawlani, and Di Wang. *Faster Width-Dependent Algorithm for Mixed Packing and Covering LPs*. Sept. 26, 2019. DOI: 10.48550/arXiv.1909.12387. arXiv: 1909.12387 [cs, math]. URL: http://arxiv.org/abs/1909.12387. Pre-published.

[Cla95] Kenneth L. Clarkson. "Las Vegas Algorithms for Linear and Integer Programming When the Dimension Is Small". In: *J. ACM* 42.2 (Mar. 1, 1995), pp. 488–499. ISSN: 0004-5411. DOI: 10.1145/201019.201036. URL: https://dl.acm.org/doi/10.1145/201019.201036.

[CLS19] Michael B. Cohen, Yin Tat Lee, and Zhao Song. "Solving Linear Programs in the Current Matrix Multiplication Time". In: *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2019. New York, NY, USA: Association for Computing Machinery, June 23, 2019, pp. 938–942. ISBN: 978-1-4503-6705-9. DOI: 10.1145/3313276.3316303. URL: https://dl.acm.org/doi/10.1145/3313276.3316303.

[CQ18] Chandra Chekuri and Kent Quanrud. "Randomized MWU for Positive LPs". In: *Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Proceedings. Society for Industrial and Applied Mathematics, Jan. 2018, pp. 358–377. DOI: 10.1137/1.9781611975031.25. URL: https://epubs.siam.org/doi/10.1137/1.9781611975031.25.

[GK95] Michael D. Grigoriadis and Leonid G. Khachiyan. "A Sublinear-Time Randomized Approximation Algorithm for Matrix Games". In: *Operations Research Letters* 18.2 (Sept. 1, 1995), pp. 53–58. ISSN: 0167-6377. DOI: 10.1016/0167-6377(95)00032-0. URL: https://www.sciencedirect.com/science/article/pii/0167637795000320.

[LS15] Yin Tat Lee and Aaron Sidford. *Efficient Inverse Maintenance and Faster Algorithms for Linear Programming*. Oct. 14, 2015. DOI: 10.48550/arXiv.1503.01752. arXiv: 1503.01752 [cs]. URL: http://arxiv.org/abs/1503.01752. Pre-published.

[LSW15] Yin Tat Lee, Aaron Sidford, and Sam Chiu-Wai Wong. "A Faster Cutting Plane Method and Its Implications for Combinatorial and Convex Optimization". In: *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. 2015 IEEE 56th Annual Symposium on Foundations of Computer Science. Oct. 2015, pp. 1049–1065. DOI: 10.1109/FOCS.2015.68. URL: https://ieeexplore.ieee.org/document/7354442.

[PST91] S.A. Plotkin, D.B. Shmoys, and E. Tardos. "Fast Approximation Algorithms for Fractional Packing and Covering Problems". In: *[1991] Proceedings 32nd Annual Symposium of Foundations of Computer Science*. [1991] 32nd Annual Symposium of Foundations of Computer Science. Oct. 1991, pp. 495–504. DOI: 10.1109/SFCS.1991.185411. URL: https://ieeexplore.ieee.org/document/185411.

[Qua19] Kent Quanrud. "Nearly Linear Time Approximations for Mixed Packing and Covering Problems without Data Structures or Randomization". In: *2020 Symposium on Simplicity in Algorithms (SOSA)*. Proceedings. Society for Industrial and Applied Mathematics, Dec. 17, 2019, pp. 69–80. DOI: 10.1137/1.9781611976014.11. URL: https://epubs.siam.org/doi/10.1137/1.9781611976014.11.

[SS08] Daniel A. Spielman and Nikhil Srivastava. "Graph Sparsification by Effective Resistances". In: *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*. STOC '08. New York, NY, USA: Association for Computing Machinery, May 17, 2008, pp. 563–568. ISBN: 978-1-60558-047-0. DOI: 10.1145/1374376.1374456. URL: https://dl.acm.org/doi/10.1145/1374376.1374456.

[Vai89] P.M. Vaidya. "Speeding-up Linear Programming Using Fast Matrix Multiplication". In: *30th Annual Symposium on Foundations of Computer Science*. 30th Annual Symposium on Foundations of Computer Science. Oct. 1989, pp. 332–337. DOI: 10.1109/SFCS.1989.63499. URL: https://ieeexplore.ieee.org/document/63499.

[vdBra+21] Jan van den Brand, Yin Tat Lee, Yang P. Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. "Minimum Cost Flows, MDPs, and ℓ_1-Regression in Nearly Linear Time for Dense Instances". In: *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. STOC 2021. New York, NY, USA: Association for Computing Machinery, June 15, 2021, pp. 859–869. ISBN: 978-1-4503-8053-9. DOI: 10.1145/3406325.3451108. URL: https://dl.acm.org/doi/10.1145/3406325.3451108.

[Wol23] Ronald de Wolf. *Quantum Computing: Lecture Notes*. Jan. 16, 2023. DOI: 10.48550/arXiv.1907.09415. arXiv: 1907.09415 [quant-ph]. URL: http://arxiv.org/abs/1907.09415. Pre-published.

[You14] Neal E. Young. *Nearly Linear-Work Algorithms for Mixed Packing/Covering and Facility-Location Linear Programs*. Nov. 5, 2014. arXiv: 1407.3015 [cs]. URL: http://arxiv.org/abs/1407.3015. Pre-published.

A Basics of multiplicative weights

In this section we briefly expand on the argument we developed in Section 1.2 for why $T = O(\rho \log n/\varepsilon)$ iterations suffice to generate a sequence of low-violation vectors $\{\boldsymbol{v}_t\}_{t=1}^T$ where $\max\left\{\frac{1}{T}\sum_{t=1}^T \boldsymbol{v}_T\right\} \leq \frac{\varepsilon}{\rho}$.

To do so we prove the following lemma.

Lemma 14. *Let $f : \Delta_n \to \{0,1\}^n$ be a (randomized) routine which given any probability distribution $\boldsymbol{p} \in \Delta_n$, it outputs a vector $\boldsymbol{v} \in \{0,1\}^n$ such that $\langle \boldsymbol{p}, \boldsymbol{v}\rangle \leq \varepsilon$. Then one can provide a sequence $\{\boldsymbol{p}_t\}_{t=1}^T$ such that the corresponding outputs $\{\boldsymbol{v}_t\}_{t=1}^T$ satisfy*

$$\max\left\{\frac{1}{T}\sum_{t=1}^T \boldsymbol{v}_t\right\} \leq 3\varepsilon\,,$$

where $T = O(\log n/\varepsilon)$.

Proof. This follows from the standard MWU framework, together with a few useful observations. For the reader's convenience, we provide a complete proof below. Let $\mathrm{smax} : \mathbb{R}^n \to \mathbb{R}$, defined as

$$\mathrm{smax}(\boldsymbol{x}) = \log \sum_{i=1}^n \exp(\boldsymbol{x}_i)\,.$$

One can easily verify that $\max(\boldsymbol{x}) \leq \mathrm{smax}(\boldsymbol{x}) \leq \mathrm{smax}(\boldsymbol{x}) + \log n$. Furthermore, we can verify that

$$\nabla \mathrm{smax}(\boldsymbol{x}) = \frac{\exp(\boldsymbol{x})}{\sum_{i=1}^n \exp(\boldsymbol{x}_i)}\,,$$

and for all $\boldsymbol{x}, \boldsymbol{\delta} \geq 0$, $\|\boldsymbol{\delta}\|_\infty \leq 1$,

$$\mathrm{smax}(\boldsymbol{x} + \boldsymbol{\delta}) \leq \mathrm{smax}(\boldsymbol{x}) + 2 \cdot \langle \nabla \mathrm{smax}(\boldsymbol{x}), \boldsymbol{\delta}\rangle\,. \tag{5}$$

Using these facts we can define

$$\boldsymbol{p}_t = \nabla \mathrm{smax}\left(\sum_{i=1}^{t-1} \boldsymbol{v}_i\right)\,.$$

Using the bound from Equation (5) together with the fact that for all t, $\langle \boldsymbol{p}_t, \boldsymbol{v}_t\rangle \leq \varepsilon$ by definition, we have that

$$\mathrm{smax}\left(\sum_{t=1}^T \boldsymbol{v}_t\right) \leq \mathrm{smax}(0) + \sum_{t=1}^T 2\langle \boldsymbol{p}_t, \boldsymbol{v}_t\rangle \leq \log n + T \cdot 2\varepsilon\,.$$

Therefore,

$$\max\left\{\frac{1}{T}\sum_{t=1}^T \boldsymbol{v}_t\right\} \leq \frac{1}{T}\mathrm{smax}\left(\sum_{t=1}^T \boldsymbol{v}_t\right) \leq \frac{1}{T}\left(\log n + T \cdot 2\varepsilon\right)\,.$$

Thus setting $T = \log n/\varepsilon$ makes the above expression bounded by 3ε, which completes the proof. □

B Recovering Clarkson's and Assadi's results

B.1 Recovering Clarkson's Result

We first show how one can recover the result from Clarkson [Cla95] to solve an LP by solving $O(d \log n)$ times LPs with $O(d^2)$ constraints.

In order to recover Clarkson's result, we have to prove which type of guarantees we would like to have, i.e. what are the desired parameters for ε and μ in Problem 1. Assume we would like to find an exact solution of an asymmetric linear program with n constraints and d variables. We know there is a set of d constraints such that solving the LP on those d constraints is sufficient. We call this set of constraints B.

To solve a linear program exactly, it is sufficient to solve Problem 1 with $\mu = 1/d$ and $\varepsilon = 0$. Indeed, assume for any $i \in [n]$, one has $|\{j \in [T] : \boldsymbol{x}_j \text{ violates constraint } i\}| < \frac{T}{d}$. If we take in particular the d constraints from B, we obtain

$$\sum_{i \in B} |\{j \in [T] : \boldsymbol{x}_j \text{ violates constraint } i\}| < d \cdot \frac{T}{d} = T. \tag{6}$$

Assume none of the $\{\boldsymbol{x}_j\}_{j=1}^T$ is a solution of the LP. That means, each $\boldsymbol{x}_j$ violates at least one constraint from B. Hence, we have $\sum_{i \in B} |\{j \in [T] : \boldsymbol{x}_j \text{ violates constraint } i\}| \geq T$. This contradicts Equation (6) which means at least one vector from $\{\boldsymbol{x}_j\}_{j=1}^T$ satisfies all the constraint from B. Hence, we have an exact solution of the LP. We are now ready to state Clarkson's main result as a corollary of our framework.

Corollary 15 (Theorem in [Cla95])**.** *Consider a linear program of the form* $\max\langle \boldsymbol{c}, \boldsymbol{x} \rangle$ *subject to* $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$*, where* $\boldsymbol{A} \in \mathbb{R}^{n \times d}$*. Assume we are given an exact solver* $\mathcal{A}$ *that runs in time* $\mathcal{T}_{\text{exact}}(d, n)$ *for any linear program on* d *variables and* n *constraints, then it is possible to solve exactly the LP in expected time*

$$O(d \log n \left(\text{nnz}(\boldsymbol{A}) + \mathcal{T}_{\text{exact}}(d, 6d^2)\right)).$$

B.2 Recovering Assadi's Result

The other important ingredient is to show that for both vertex cover and minimum odd-set cover, the number of solutions is upper bounded by a small exponential.

Lemma 16 (Lemma 3.3 and 4.3 from [Ass25])**.** *Consider a graph* G *with* n *vertices and* m *edges.*[6]

1. *If* G *is bipartite, then there are at most* 2^n *distinct candidates for vertex cover.*
2. *If* G *has integral weights upper bounded by* W*, then there are at most* $\exp(3n \cdot \log(nW))$ *candidates for odd-set cover.*

The proof of the first point in the above lemma is due to the fact that in bipartite graphs, minimum vertex cover is integral. For general graphs, there is a lot of structure on the solution of minimum odd-set cover, and the upper bound is obtained through a laminarity property on the set of optimal solutions.

Corollary 17 (Theorem in [Ass25])**.** *Consider a graph* G *with* n *vertices and* m *edges.*

1. *With exponentially high probability, it is possible to find a* $(1+\varepsilon)$*-approximation of minimum vertex cover on bipartite graphs (thus a* $(1-\varepsilon)$*-approximation of maximum bipartite matching) in time*

$$O\left(\frac{\log m}{\varepsilon}(m + \mathcal{T}(n, O(\frac{n}{\varepsilon})))\right),$$

where $\mathcal{T}(n, m)$ *is the time required to compute a vertex cover and a matching on a bipartite graph with* m *edges and* n *vertices.*

[6] For the proof, see the proof of equations (4) and (7) of [Ass25].

2. *If G has integral weights in $[0, W]$, with exponentially high probability it is possible to find a $(1+\varepsilon)$-approximation of minimum odd-set cover on graphs (thus a $(1-\varepsilon)$-approximation of maximum general matching) in time*

$$O\left(\frac{\log W}{\varepsilon}(m + \mathcal{T}(n, O(\frac{n\log(nW)}{\varepsilon})))\right),$$

where $\mathcal{T}(n, m)$ is the time required to compute a minimum odd-set cover and a maximum weight matching on a graph with m edges and n vertices.

C Proofs of Section 4

C.1 Proof of Lemma 5

We first prove the following lemma which states that the number of constraint violated by an exact solution is inversely proportional to the number of constraint sampled.

Lemma 18 (Sampling lemma [Cla95])**.** *Consider a linear program of the form $\max\langle \boldsymbol{c}, \boldsymbol{x}\rangle$ subject to $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$, where $\boldsymbol{A} \in \mathbb{R}^{n\times d}$. Given a probability distribution $\boldsymbol{p} \in \Delta_n$, consider a set $S \subseteq [n]$ such that*

$$\mathbb{P}[i \in S] \geq \min\{r\boldsymbol{p}_i, 1\}.$$

Let $\boldsymbol{x}$ be a solution of the partial LP defined by the constraint from S. We have

$$\mathbb{E}\left[\sum_{i\in[n]:\langle \boldsymbol{A}_{i:}, \boldsymbol{x}\rangle > \boldsymbol{b}_i} \boldsymbol{p}_i\right] \leq \frac{d}{r}.$$

Proof. For any subset of constraint $S \subseteq [n]$, we define $\boldsymbol{x}^*_S$ as the exact solution of the LP defined on those constraints. For simplicity, we define the violation vector $\boldsymbol{v}(\boldsymbol{x})$ as:

$$\boldsymbol{v}_i(\boldsymbol{x}) := \begin{cases} 1 & \text{if } \langle \boldsymbol{A}_{i:}, \boldsymbol{x}\rangle > \boldsymbol{b}_i \qquad \text{(constraint } i \text{ is violated);} \\ 0 & \text{if } \langle \boldsymbol{A}_{i:}, \boldsymbol{x}\rangle \leq \boldsymbol{b}_i \qquad \text{(constraint } i \text{ is satisfied).} \end{cases}$$

We have

$$\begin{aligned}
\mathbb{E}[\langle \boldsymbol{p}, \boldsymbol{v}(\boldsymbol{x})\rangle] &= \sum_{S\subseteq[n]} \mathbb{P}[S]\, \mathbb{E}[\langle \boldsymbol{p}, \boldsymbol{v}(\boldsymbol{x})\rangle | S] \\
&= \sum_{S\subseteq[n]} \mathbb{P}[S]\, \mathbb{E}[\langle \boldsymbol{p}, \boldsymbol{v}(\boldsymbol{x}^*_S)\rangle] \\
&= \sum_{S\subseteq[n]} \mathbb{P}[S] \sum_{i\in[n]\setminus S} \boldsymbol{p}_i \boldsymbol{v}_i(\boldsymbol{x}^*_S) \\
&= \sum_{S\subseteq[n]} \sum_{i\in[n]\setminus S} \mathbb{P}[S]\, \boldsymbol{p}_i \boldsymbol{v}_i(\boldsymbol{x}^*_S) \\
&\leq \frac{1}{r} \sum_{S\subseteq[n]} \sum_{i\in[n]\setminus S} \mathbb{P}[S\cup\{i\}]\, \boldsymbol{v}_i(\boldsymbol{x}^*_S) \\
&= \frac{1}{r} \sum_{Q\subseteq[n]} \mathbb{P}[Q] \sum_{i\in Q} \boldsymbol{v}_i(\boldsymbol{x}^*_{Q\setminus\{i\}}).
\end{aligned}$$

Since $\boldsymbol{x}^*_Q$ is fully determined by d constraints from Q, $\boldsymbol{v}_i(\boldsymbol{x}^*_{Q\setminus\{i\}})$ equals 1 on those constraints only. Hence,

$$\mathbb{E}[\langle \boldsymbol{p}, \boldsymbol{v}(\boldsymbol{x})\rangle] \leq \frac{1}{r} \sum_{Q\subseteq[n]} \mathbb{P}[Q]\, d = \frac{d}{r}.$$

□

We are now ready to prove Lemma 5.

Proof of Lemma 5. We use Lemma 18 with $r = 2d/\mu$, hence we have that on average $\langle \boldsymbol{p}, \mathbf{1}_{\boldsymbol{Ax}>\boldsymbol{b}} \rangle \leq \frac{\mu}{2}$. Using Markov's inequality, with probability $1/2$, we have $\langle \boldsymbol{p}, \mathbf{1}_{\boldsymbol{Ax}>\boldsymbol{b}} \rangle \leq \mu$. Hence, on average 2 iterations are necessary, which means the expected time is $O(\mathrm{nnz}(\boldsymbol{A}) + \mathcal{T}_{\mathrm{exact}}(d, 2\frac{d}{\mu}))$. □

C.2 Proof of Lemma 6

Proof. For any $\boldsymbol{x} \in \bigcup_{S \subseteq [n]} \mathcal{A}\langle S \rangle$, consider $Q(\boldsymbol{x}) := \{i \in [n] : \langle \boldsymbol{A}_{i:}, \boldsymbol{x} \rangle > \boldsymbol{b}_i + \varepsilon\}$. Assume $\sum_{i \in Q(\boldsymbol{x})} \boldsymbol{p}_i > \mu$. We are sampling each constraint with probability at least $r\boldsymbol{p}_i$ where $r = \frac{\log(Nn)}{\mu}$. We have

$$\begin{aligned}
\mathbb{P}[\boldsymbol{x} \text{ is } \varepsilon\text{-feasible on the sampled constraints}] &= \mathbb{P}[\text{No constraints from } Q(\boldsymbol{x}) \text{ were sampled}] \\
&\leq \prod_{i \in Q(\boldsymbol{x})} (1 - r\boldsymbol{p}_i) \\
&\leq \exp\left(- \sum_{i \in Q(\boldsymbol{x})} r\boldsymbol{p}_i \right) \\
&\leq \exp\left(-r\mu\right) \leq \frac{1}{Nn}.
\end{aligned}$$

A union bound over the N possible outputs of the solver $\mathcal{A}$ ensures that with high-probability the sum of probabilities of violated constraints is smaller than μ. □

C.3 Proof of Claim 3

Proof. Assume we have $\boldsymbol{x}$ such that $\boldsymbol{Cx} \geq \mathbf{1}$, and $\boldsymbol{Px} \leq (1+\varepsilon)\mathbf{1}$. We will discretize each entry of $\boldsymbol{x}$. We compute our new solution $\tilde{\boldsymbol{x}}$ as follows:

$$\tilde{\boldsymbol{x}}_i := \begin{cases} 0 & \text{if } \boldsymbol{x}_i = 0; \\ \frac{\varepsilon}{r_p} & \text{if } 0 < \boldsymbol{x}_i \leq \frac{\varepsilon}{r_p}; \\ \min\{\frac{\varepsilon}{r_p}(1+\varepsilon)^{k+1}, 1\} & \text{if } \frac{\varepsilon}{r_p}(1+\varepsilon)^k < \boldsymbol{x}_i \leq \min\{\frac{\varepsilon}{r_p}(1+\varepsilon)^{k+1}, 1\}. \end{cases}$$

$\tilde{\boldsymbol{x}}$ verifies $\mathbf{0} \leq \boldsymbol{x} \leq \tilde{\boldsymbol{x}} \leq \mathbf{1}$ where the inequalities are taken coordinate-wise.

$\tilde{\boldsymbol{x}}$ is a $1+4\varepsilon$ approximation. Since $\boldsymbol{x} \leq \tilde{\boldsymbol{x}}$ coordinate-wise, we have $\boldsymbol{C}\tilde{\boldsymbol{x}} \geq \boldsymbol{Cx} \geq \mathbf{1}$.

We need to verify that $\boldsymbol{P}\tilde{\boldsymbol{x}} \leq (1+4\varepsilon)\mathbf{1}$. We have for any i, $\tilde{\boldsymbol{x}}_i \leq (1+\varepsilon)\boldsymbol{x}_i + \frac{\varepsilon}{r_p}$. Hence,

$$\boldsymbol{P}\tilde{\boldsymbol{x}} \leq (1+\varepsilon)\boldsymbol{Px} + \frac{\varepsilon}{r_p}\boldsymbol{P}\mathbf{1} \leq (1+\varepsilon)^2\mathbf{1} + \varepsilon\mathbf{1} \leq (1+4\varepsilon)\mathbf{1}.$$

Hence, $\tilde{\boldsymbol{x}}$ is a $1+4\varepsilon$ approximation.

Size of $\mathcal{S}$. For each coordinate i, we have $\boldsymbol{x}_i \in \{0,1\} \cup \{\frac{\varepsilon}{r_p}(1+\varepsilon)^k : k \in [K]\}$. Where K is the largest integer such that $\frac{\varepsilon}{r_p}(1+\varepsilon)^K < 1$. Hence, $K \leq \lceil 4\frac{\log(r_p/\varepsilon)}{\varepsilon} \rceil$. Since there are d coordinates, we have $|\mathcal{S}| \leq \lceil 2 + 4\frac{\log(r_p/\varepsilon)}{\varepsilon} \rceil^d$.

□

C.4 Proof of Theorem 8

Proof of Theorem 8. We prove this theorem in three steps. First, we prove the requirement on the (ε, μ)-low-violation oracle. Second, we prove that $O(\frac{\log n_c}{\varepsilon})$ iterations are sufficient. Finally, we construct our (ε, μ)-low-violation oracle, and show the expected running time of each iteration.

The required (ε, μ)-low-violation oracle. Assume we have $\{\boldsymbol{x}_j\}_{j\in[T]}$ a solution of Problem 1. Then, since at each iteration we are using a mixed packing and covering solver that contains all the packing constraints, we have for all $j \in [T]$, $\boldsymbol{P}\boldsymbol{x}_j \leq (1+\varepsilon)\mathbf{1}$. Hence, for the packing constraints, $\bar{\boldsymbol{x}} := \sum_{j\in[T]} \boldsymbol{x}_j/T$ already satisfies $\boldsymbol{P}\bar{\boldsymbol{x}} \leq (1+\varepsilon)\mathbf{1}$. For the covering constraints, for $j \in [T]$, we have $\boldsymbol{C}\boldsymbol{x}_j \geq \mathbf{0}$.

For a covering constraint i, let $V_i \subseteq [T]$ be the set of solutions violating constraint i: $\{\boldsymbol{x}_j : j \in [T] \wedge \langle \boldsymbol{C}_{i:}, \boldsymbol{x}_j\rangle < 1\}$. We have

$$\sum_{j\in[T]} \langle \boldsymbol{C}_{i:}, \boldsymbol{x}_j\rangle - 1 = \sum_{j\in V_i} \langle \boldsymbol{C}_{i:}, \boldsymbol{x}_j\rangle - 1 + \sum_{j\in[T]\setminus V_i} \langle \boldsymbol{C}_{i:}, \boldsymbol{x}_j\rangle - 1 \leq \sum_{j\in V_i} 0 + \sum_{j\in[T]\setminus V_i} 1 \geq (1-\mu)T.$$

For $\mu = \varepsilon$, $\bar{\boldsymbol{x}}/(1-\varepsilon)$ is a $(1+4\varepsilon)$-approximation.

Number of iterations. Since we are using a $(\varepsilon,\varepsilon)$-low-violation oracle, and we are sampling from a set of at most n_c constraints, $O(\frac{\log n_c}{\varepsilon})$ iterations are required per Theorem 3.

Constructing the $(\varepsilon,\varepsilon)$-low-violation oracle. We want to use Lemma 6, we need to create a mixed packing and covering solver that $1+\varepsilon$-approximations from a small set of vectors. Given any mixed packing and covering solver $\mathcal{A}$, we post-process a call on $\mathcal{A}$ to discretize the output using Claim 3. Formally, given any set of constraints S, we construct the $1+4\varepsilon$-approximate algorithm:

1. Compute $\boldsymbol{x}$ the output of $\mathcal{A}(S)$.
2. Compute $\tilde{\boldsymbol{x}}$ using Claim 3.

Our new solver has outputs in a set with at most $\lceil 2 + 4\frac{\log(r_p/\varepsilon)}{\varepsilon}\rceil^d$ elements. Using Lemma 6, it is sufficient to sample on average $O\left(\frac{d}{\varepsilon}\log(1+\frac{\log(\frac{r_p}{\varepsilon})}{\varepsilon})\right)$ constraints.

Hence, at each iteration, we need to run a mixed packing and covering solver with packing constraint $\boldsymbol{P}$, and $O\left(\frac{d}{\varepsilon}\log(1+\frac{\log(\frac{r_p}{\varepsilon})}{\varepsilon})\right)$ covering constraints, each one having row-sparsity at most r_c. Moreover, at each iteration, we need to update the sampling probabilities which cost $O(\mathrm{nnz}(\boldsymbol{C}))$. □

D Proofs of Section 5

D.1 Proofs of Claims 4 and 5

Notice that using Lemma 9 with $\boldsymbol{q} = s\frac{\boldsymbol{w}}{\widetilde{W}}$ outputs a set with on average $s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}}$ elements. Hence, we can estimate $\|\boldsymbol{w}\|_1$ by sampling multiple times using Lemma 9 and taking the set with the median number of elements. We describe in the following claim a procedure to enhance the guarantees of Lemma 9 using multiple call to it.

Claim 7. *For a non-negative vector $\boldsymbol{w} \in \mathbb{R}^n$, a real $\widetilde{W} \leq \|\boldsymbol{w}\|_1$, an integer $s \geq 6$, and a probability p. Assume one has quantum query access to $\boldsymbol{w}$. Then, using $\widetilde{O}(\sqrt{ns\|\boldsymbol{w}\|_1/\widetilde{W}}\log\frac{1}{p})$ queries to $\boldsymbol{w}$, Procedure 1 computes a set S such that:*

1. *Every $i \in [n]$ is in S with probability over $\min\{s\frac{\boldsymbol{w}_i}{\widetilde{W}}, 1\}$;*
2. *The size of S is close to its average size:*

$$\mathbb{P}\left[\left||S| - s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}}\right| \geq \sqrt{6s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}}}\right] \leq p.$$

Procedure 1 `quantum_sampling`$(\boldsymbol{w}, s, \widetilde{W}, p)$

Input: Quantum query access to a non-negative vector $\boldsymbol{w}$, $s \geq 6$, $\widetilde{W} \leq \|\boldsymbol{w}\|_1 = O(\widetilde{W})$, p a probability
Output: $\bar{S} \subseteq [n]$ such that $\bar{S}$ contains each i independently with probability $\frac{s}{\widetilde{W}}\boldsymbol{w}_i$.

1: **for** $r = 1$ to $1 + 5\log\frac{1}{p}$ **do**
2: $\quad S_r \leftarrow$ Lemma 9 with probabilities $\frac{s}{\widetilde{W}}\boldsymbol{w}$
3: $\quad s_r \leftarrow |S_r|$
4: **end for**
5: $\bar{S} \leftarrow S_{\bar{r}}$ such that $s_{\bar{r}}$ is the median of $(s_r)_r$
6: **return** $\bar{S}$

Procedure 2 `estimation_sum`$(\boldsymbol{w}, \widetilde{W}, p)$

Input: Quantum query access to a non-negative vector $\boldsymbol{w}$, $\widetilde{W} \leq \|\boldsymbol{w}\|_1 = O(\widetilde{W})$, p a probability
Output: $\widetilde{W}'$ such that $\widetilde{W}' \leq \|\boldsymbol{w}\|_1 \leq 2\widetilde{W}'$ with probability at least $1 - p$

1: $s \leftarrow 71$
2: $S \leftarrow$ `quantum_sampling`$(\boldsymbol{w}, s, \widetilde{W}, p)$ ▷ Using Procedure 1
3: $\widetilde{W}' \leftarrow \frac{\widetilde{W}}{2s}\left(\sqrt{3 + 2|S|} - \sqrt{3}\right)^2$
4: **return** $\widetilde{W}'$

Proof. Consider $X_1, \ldots, X_n$ independent random variables taking values in $\{0, 1\}$. Let X denote their sum and let $\mu = \mathbb{E}[X]$ denote the sum's expected value. Then for any $0 \leq \delta \leq 1$, multiplicative Chernoff bound states that:

$$\mathbb{P}[|X - \mu| \geq \delta\mu] \leq 2e^{-\frac{\delta^2\mu}{3}}. \tag{7}$$

Assume S is the output of a call to Lemma 9 using $\boldsymbol{q} = s\boldsymbol{w}/\widetilde{W}$. Let X_i be the event of constraint i being in S. $X_i \in \{0, 1\}$, and $|S| = \sum_i X_i = X$. Moreover, $\mathbb{E}[X] = s\|\boldsymbol{w}\|_1/\widetilde{W}$. We have using Equation (7) with $\delta = \sqrt{\frac{6}{\mathbb{E}[X]}} \leq 1$ since $s \geq 6$:

$$\mathbb{P}\left[\left||S| - s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}}\right| \geq \sqrt{6s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}}}\right] \leq 2e^{-2} \leq 1/e. \tag{8}$$

Consider $\bar{S}$ the output of Procedure 1. In Procedure 1, we have $O(\log\frac{1}{p})$ sets satisfying Equation (8). Since $\bar{S}$ has the size of the median of those R sets, if $\bar{S}$ does not satisfy Equation (8), then at least $R/2$ sets from $(S_r)_{r\in[R]}$ do not satisfy it. This happens with probability less than $1/e^{R/2}$. Hence, since $R = \lceil 5\log\frac{1}{p}\rceil$, this happens with probability less than p. Hence:

$$\mathbb{P}\left[\left||\bar{S}| - s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}}\right| \geq \sqrt{6s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}}}\right] \leq p.$$

□

Proof of Claim 4.

Proof. We will use Procedure 2, with probability at least $1 - p$, we have

$$s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}} - \sqrt{6s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}}} \leq |S| \leq s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}} + \sqrt{6s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}}}. \tag{9}$$

In particular, $|S| \leq s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}} + \sqrt{6s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}}}$. Hence, with $x = \sqrt{\|\boldsymbol{w}\|_1}$, we have

$$0 \leq \frac{s}{\widetilde{W}}x^2 + \sqrt{6\frac{s}{\widetilde{W}}}x - |S|.$$

Hence, we have $x \geq \sqrt{\frac{\widetilde{W}}{2s}}(\sqrt{3+2|S|}-\sqrt{3})$. Let $\widetilde{W}' := \frac{\widetilde{W}}{2s}(\sqrt{3+2|S|}-\sqrt{3})^2$, we have $\widetilde{W}' \leq x^2 = \|\boldsymbol{w}\|_1$.

On the other hand, $s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}} - \sqrt{\frac{6s\|\boldsymbol{w}\|_1}{\widetilde{W}}} \leq |S|$. Let $x = \sqrt{\|\boldsymbol{w}\|_1}$, the inequality implies $x \leq \sqrt{\frac{\widetilde{W}}{2s}}(\sqrt{3+2|S|}+\sqrt{3})$. Hence,

$$\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}'} \leq \left(\frac{\sqrt{3+2|S|}+\sqrt{3}}{\sqrt{3+2|S|}-\sqrt{3}}\right)^2 = \left(\frac{6+2|S|+2\sqrt{9+6|S|}}{2|S|}\right)^2 = \left(1+\sqrt{\frac{6}{|S|}+\frac{9}{|S|^2}}+\frac{3}{|S|}\right)^2.$$

Using the above inequality, one can verify if $|S| \geq 50$, then $\|\boldsymbol{w}\|_1 \leq 2\widetilde{W}'$.

It remains to show $|S|$ has at least 50 elements with probability larger than $1-p$. We assume $|S|$ satisfied Equation (9). Hence,

$$|S| \geq s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}} - \sqrt{6s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}}} = s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}}\left(1-\sqrt{6\frac{\widetilde{W}}{s\|\boldsymbol{w}\|_1}}\right) \geq s\left(1-\sqrt{\frac{6}{s}}\right) = s-\sqrt{6s}.$$

Where the last inequality comes from $\widetilde{W} \leq \|\boldsymbol{w}\|_1$. Hence, with $s \geq 71$, we have $|S| \geq 50$. □

Proof of Claim 5.

Proof. We use Procedure 1 with $s \geq 6$, hence with probability at least $1-p$:

$$|S| \leq s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}} + \sqrt{6s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}}} = s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}}\left(1+\sqrt{\frac{6}{s}\frac{\widetilde{W}}{\|\boldsymbol{w}\|_1}}\right) \leq s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}}\left(1+\sqrt{\frac{6}{6}\frac{\|\boldsymbol{w}\|_1}{\|\boldsymbol{w}\|_1}}\right) = 2s\frac{\|\boldsymbol{w}\|_1}{\widetilde{W}} \leq 4s.$$

□

D.2 Proof of Theorem 10

Proof. As per Corollary 15, it is sufficient to be able to sample in S constraint i with probability larger than $\min\{6d^2\boldsymbol{p}_i, 1\}$. At iteration t, we have a set X such that $|X| = t$. Let $\boldsymbol{w}_i(X) := 2^{\sum_{\boldsymbol{x}\in X}\boldsymbol{v}_i^0(\boldsymbol{x})}$ where $\boldsymbol{v}^0(\boldsymbol{x})$ is the binary violation vector, i.e. $\boldsymbol{v}_i^0(\boldsymbol{x}) = 0$ if $\boldsymbol{x}$ satisfies constraint i ($\langle \boldsymbol{A}_{i:}, \boldsymbol{x}\rangle \leq \boldsymbol{b}_i$), and 1 if $\boldsymbol{x}$ violates constraint i.

Maintenance of the estimation of $\|\boldsymbol{w}(X_t)\|_1$: We first prove at iteration t, we have $\widetilde{W}_t \leq \|\boldsymbol{w}(X_t)\|_1 \leq 2\widetilde{W}_t$.

For $t=0$, we have $\widetilde{W}_0 = n = \|\boldsymbol{w}(X_0)\|_1 = \|\boldsymbol{w}(\emptyset)\|_1$.

Assume for $t \geq 0$ we have $\widetilde{W}_t \leq \|\boldsymbol{w}(X_t)\|_1 \leq 2\widetilde{W}_t$ where X contains t elements. Consider $X_{t+1} = X_t \cup \{\boldsymbol{x}\}$, For each $i \in [n]$, $\boldsymbol{w}_i(X_t) \leq \boldsymbol{w}_i(X_{t+1}) \leq 2\boldsymbol{w}_i(X_t)$. Hence, we have $\|\boldsymbol{w}(X_t)\|_1 \leq \|\boldsymbol{w}(X_{t+1})\|_1 \leq 2\|\boldsymbol{w}(X_t)\|_1$ which means $\widetilde{W}_t \leq \|\boldsymbol{w}(X_{t+1})\|_1 \leq 4\widetilde{W}_t$. When we use Procedure 2 to estimate $\|\boldsymbol{w}(X_{t+1})\|_1$, the conditions of Claim 4 are satisfied, hence $\widetilde{W}_{t+1}$ is such that $\widetilde{W}_{t+1} \leq \|\boldsymbol{w}(X_{t+1})\|_1 \leq 2\widetilde{W}_{t+1}$.

Hence, we always have $\widetilde{W}_t \leq \|\boldsymbol{w}(X_t)\|_1 \leq 2\widetilde{W}_t$.

Low-violation solution: Since $\widetilde{W}_t \leq \|\boldsymbol{w}(X_t)\|_1$, we have:

$$\sum_i \boldsymbol{w}(X_{t+1}) = \sum_i \boldsymbol{w}_i(X_t)(1+\boldsymbol{v}_i^0(\boldsymbol{x})) = \|\boldsymbol{w}(X_t)\|_1\left(1+\sum_i \frac{\boldsymbol{w}_i(X_t)}{\|\boldsymbol{w}(X_t)\|_1}\boldsymbol{v}_i^0(\boldsymbol{x})\right).$$

Moreover, since S_t is sampled using Claim 5, we know that each constraint i is sampled independently and is present in S_t with probability at least $6d^2\boldsymbol{w}_i(X_t)/\|\boldsymbol{w}(X_t)\|_1$. Hence, with Lemma 18, we have $\langle \frac{\boldsymbol{w}(X_t)}{\|\boldsymbol{w}(X_t)\|_1}, \boldsymbol{v}^0(\boldsymbol{x})\rangle \leq \frac{1}{6d}$.

Hence, $\mathbb{E}[\|\boldsymbol{w}(X_{t+1})\|_1] \leq (1+\frac{1}{6d})\|\boldsymbol{w}(X_t)\|_1$. Or:

$$\mathbb{E}[\|\boldsymbol{w}(X_T)\|_1] \leq (1+\frac{1}{6d})^T n \leq 2^{\frac{1}{2}\frac{T}{d}} n.$$

Returning the solution: If there is $\boldsymbol{x} \in X_T$ such that $\boldsymbol{x}$ is a solution to the linear program, then it should be detected with high probability with the call to Claim 6. Hence, we can assume that none of the $\boldsymbol{x} \in X_T$ are solutions to the LP.

Since none of the $\boldsymbol{x}$ in X_T is a solution to the linear program, at least one constraint from the basis B (i.e. the set of d constraints that described the optimum) is violated at each iteration. Hence, $\sum_{i \in B} \boldsymbol{w}_i(X_T) \geq 2^{\frac{T}{d}}$.

For some constant $c > 0$, using Markov's inequality, we have with probability at least $1 - n^{-c}$, $\|\boldsymbol{w}(X_T)\|_1 \leq n^{c+1} 2^{\frac{3}{2}\frac{T}{d}}$. Hence, with high probability we can assume $\|\boldsymbol{w}(X_T)\|_1 \leq n^3 2^{\frac{1}{2}\frac{T}{d}}$. Thus, as long as none of the $\boldsymbol{x}$ in X_T are solutions to the LP, the following should be true with high probability:

$$2^{\frac{T}{d}} \leq \|\boldsymbol{w}(X_T)\|_1 \leq n^3 2^{\frac{1}{2}\frac{T}{d}}.$$

Hence, $T \leq 6d \log_2 n \leq 24 d \log n$. This is a contradiction since we are performing more than $24 d \log n$ iterations.

Number of row-queries to $(\boldsymbol{A}, \boldsymbol{b})$: Within iteration t, since X_t has less than T elements, hence a query to $\boldsymbol{w}(X_t)$ can be done using at most T queries to $\boldsymbol{v}^0$, which correspond to T row-queries to $(\boldsymbol{A}, \boldsymbol{b})$.

We will count the number of queries to $\boldsymbol{w}(X_t)$. During iteration t, Claims 4 and 6 perform at most $\widetilde{O}(\sqrt{n})$ queries to $\boldsymbol{w}(X_t)$. On the other hand, Claim 5 performs $\widetilde{O}(\sqrt{nd^2})$ queries to $\boldsymbol{w}(X_t)$.

Since each query to $\boldsymbol{w}(X_t)$ requires at most T row-queries to $(\boldsymbol{A}, \boldsymbol{b})$, we indeed have that in total $\widetilde{O}(\sqrt{n} d^3)$ row-queries to $(\boldsymbol{A}, \boldsymbol{b})$ are performed.

Running time: $\widetilde{O}(d)$ iterations are performed, each of which requires at most $\widetilde{O}(\sqrt{nd^2} d)$ row-queries to $(\boldsymbol{A}, \boldsymbol{b})$ to sample the set of constraints. This requires time $\widetilde{O}(\sqrt{n} d^2 r)$ where t is the row-sparsity of $\boldsymbol{A}$. Moreover, one call to an exact solver is used. Since $|S_t| \leq 4s = 24d^2$, we have that the call is made on an exact solver on at most $24d^2$ constraints and d variables. □

D.3 Proof of Theorem 11

Proof. The proof is similar to Theorem 8, the difference is on how we sample the constraints at each iteration. We use the exact same techniques as in Theorem 10, and obtain a query access to $\boldsymbol{w}(X)/\|\boldsymbol{w}(X)\|_1$ at each iteration by maintaining a constant factor estimation of $\|\boldsymbol{w}(X)\|_1$ and using queries to rows of $\boldsymbol{A}$, and $\boldsymbol{b}$. □

D.4 Proof of Theorem 13

Proof. In this theorem, the proof is different since the value of $\boldsymbol{w}$ is not the same. For the sake of the proof, we do not use MWU in a black-box fashion, since this problem does not fit inside Problem 1.

At iteration t, we have $\boldsymbol{w}_i(X_t) = 2^{\sum_{\boldsymbol{x} \in X_t} \frac{\langle \boldsymbol{A}_{i,:}, \boldsymbol{x} \rangle - \boldsymbol{b}_i}{\rho}}$. Notice that one query to $\boldsymbol{w}(X)$ can be made using one row-query to $(\boldsymbol{A}, \boldsymbol{b})$, indeed, we can maintain $\bar{\boldsymbol{x}} := \sum_{\boldsymbol{x} \in X_t} \boldsymbol{x}$ and use $\boldsymbol{w}(X_t) = 2^{\frac{1}{\rho}(\boldsymbol{A}\bar{\boldsymbol{x}} - t\boldsymbol{b})}$.

We consider $\boldsymbol{p} = \boldsymbol{w}(X)/\|\boldsymbol{w}(X)\|_1$. If we seek to sample s elements from $[n]$ with the probability distribution $\boldsymbol{p}$, we require $\widetilde{O}(\sqrt{ns})$ queries to $\boldsymbol{p}$, each one requires one row-query to $(\boldsymbol{A}, \boldsymbol{b})$.

At each iteration, we use an average $(\varepsilon, \varepsilon/3\rho)$-low-violation oracle with $\boldsymbol{p}$ to obtain $\tilde{\boldsymbol{x}}$. We have

$\langle \boldsymbol{p}, \boldsymbol{v}^{\varepsilon}(\tilde{\boldsymbol{x}})\rangle \leq \mu$, hence on average:

$$\begin{aligned}
\|\boldsymbol{w}(X \cup \{\tilde{\boldsymbol{x}}\})\|_1 &= \sum_{i\in[n]} 2^{\sum_{\boldsymbol{x}\in X} \frac{\langle \boldsymbol{A}_{i,:},\boldsymbol{x}\rangle - \boldsymbol{b}_i}{\rho}} 2^{\frac{\langle \boldsymbol{A}_{i,:},\tilde{\boldsymbol{x}}\rangle - \boldsymbol{b}_i}{\rho}} \\
&\leq \sum_{i\in[n]} 2^{\sum_{\boldsymbol{x}\in X} \frac{\langle \boldsymbol{A}_{i,:},\boldsymbol{x}\rangle - \boldsymbol{b}_i}{\rho}} \left(1 + \boldsymbol{v}_i^{\varepsilon}(\tilde{\boldsymbol{x}})\right) \\
&\leq \left(1 + \frac{\varepsilon}{3\rho}\right) \sum_{i\in[n]} 2^{\sum_{\boldsymbol{x}\in X} \frac{\langle \boldsymbol{A}_{i,:},\boldsymbol{x}\rangle - \boldsymbol{b}_i}{\rho}} \\
&= \left(1 + \frac{\varepsilon}{3\rho}\right) \|\boldsymbol{w}(X)\|_1 \leq 2^{\frac{1}{2}\frac{\varepsilon}{\rho}} \|\boldsymbol{w}(X)\|_1 .
\end{aligned}$$

Therefore, we have $\mathbb{E}\left[\|\boldsymbol{w}(X_T)\|_1\right] \leq 2^{\frac{T}{2}\frac{\varepsilon}{\rho}} n$. Hence, using Markov's inequality, with high-probability we have $\|\boldsymbol{w}(X_T)\|_1 \leq 2^{\frac{T}{2}\frac{\varepsilon}{\rho}} n^2$.

For any constraint i, we can use $\boldsymbol{w}_i(X_T)$ to compute its violation, indeed, $\left(\boldsymbol{A}\frac{\bar{\boldsymbol{x}}}{T} - \boldsymbol{b}\right)_i = \frac{\rho}{T}\log_2(\boldsymbol{w}_i(X_T))$. Hence, it is sufficient to show that for any i, $\log_2 \boldsymbol{w}_i(X_T) \leq T\frac{\varepsilon}{\rho}$.

We have:

$$\log_2 \boldsymbol{w}_i(X_T) \leq \log_2 \left(\sum_j \boldsymbol{w}_j(X_T)\right) = \log_2 \|\boldsymbol{w}(X_T)\|_1 \leq \frac{T}{2}\frac{\varepsilon}{\rho} + 2\log_2 n \leq T\frac{\varepsilon}{\rho},$$

where the last inequality comes from $2\log_2 n \leq \frac{T}{2}\frac{\varepsilon}{\rho}$ since $T \geq 16\frac{\rho}{\varepsilon}\log n$.

It remains to prove we can estimate $\widetilde{W}_{t+1}$ using $\texttt{estimation\_sum}(\boldsymbol{w}(X_{t+1}), \widetilde{W}_t/2, p)$. Indeed, in our case the weights can decrease, however, since we are using ρ instead of $V_{\max}$, the decrease can not be too large. In fact, we have $\frac{1}{2}\boldsymbol{w}_i(X_t) \leq \boldsymbol{w}_i(X_{t+1}) \leq 2\boldsymbol{w}_i(X_t)$. Therefore, we have $\widetilde{W}_{t+1}$ such that $\widetilde{W}_{t+1} \leq \sum_i \boldsymbol{w}_i(X_{t+1}) \leq 2\widetilde{W}_{t+1}$. □