Adaptive Sparsification for Linear Programming

Etienne Objois
IRIF, Paris, France
Université Paris Cité, France

Adrian Vladu
CNRS, Paris, France
IRIF, Paris, France
Université Paris Cité, France

Abstract

We provide a generic toolkit for sparsifying the constraint set of linear programs (LPs). To this end, we reduce solving a linear program with n constraints and d variables ($n \gg d$), to solving a sequence of LPs defined over only a small subset of the constraints, obtained by adaptively sub-sampling the original set. We provide results for both the low and high precision regimes. To achieve the former result, we streamline and generalize the techniques from [Assadi '25] for approximately computing maximum matchings in the semi-streaming setting to the case of general LPs. For the latter, we robustify the methods of [Clarkson '95], which were originally designed for exact LP solvers. As a consequence we obtain fast approximate LP solvers which reduce the dependence on width and error from quadratic to linear, compared to vanilla multiplicative-weights based approaches.

Additionally, we leverage our findings to obtain fast LP solvers in the quantum query access model, where the running time scales with $\sqrt{n}$. This completely decouples the component responsible for quantum speed-ups, solely represented by a generalization of Grover's search, from its classical algorithmic counterpart.

2012 ACM Subject Classification Theory of computation → Approximation algorithms analysis

Keywords and phrases linear programming, sparsification, sampling, quantum algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2026.40

Related Version *Full Version*: `https://arxiv.org/abs/2510.08348` [25]

Funding This work was partially supported by the French Agence Nationale de la Recherche (ANR) under grant ANR-21-CE48-0016 (project COMCOPT).

1 Introduction

Linear Programming (LP) serves as a foundation for both continuous and combinatorial optimization. Substantial algorithmic progress over the past several decades has been achieved in two distinct optimization regimes [18]. The high-precision regime, characterized by runtimes that scale polynomially with the number of bits of precision of the returned solution $\log(1/\varepsilon)$, has been dominated by interior point methods (IPMs) and cutting-plane techniques [20, 12, 10, 22]. Conversely, the low-precision regime, characterized by runtimes scaling polynomially with $1/\varepsilon$, has been largely defined by first-order techniques, most notably the multiplicative weights update (MWU) method [14, 11, 4].

An important computational bottleneck arises in the tall LP regime where the number of constraints n vastly exceeds the ambient dimension d ($n \gg d$). Geometrically, only d hyperplanes are strictly required to identify an exact optimal vertex in a non-degenerate polytope. In the context of graph algorithms, this idea underlies sparsification techniques [6, 19], 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 one-shot sparsification procedure for such tall LPs is unlikely to exist without heavily factoring in the objective. This is exemplified by directed max flow, where removing a single arc/constraint can reduce a directed cut from a positive value to zero, entirely ruining the approximate solution. However, this limitation does not rule out an *adaptive sparsification* algorithm, which iteratively solves a sequence of sparse sub-problems to recover the original solution.

A classical result of Clarkson [9] 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 [5] 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.

To achieve this result, [5] builds heavily on the combinatorial intuition from bipartite matching, and provides an analysis extremely similar in spirit with the well known multiplicative weights update method [4]. It appears, though, that the connection is unclear, especially since the iteration complexity of the algorithm has a single dependence in $1/\varepsilon$, while standard multiplicative weights typically has a running time that scales quadratically with the target error. Hence, developing a better understanding of these techniques has the potential to provide further important improvements in the context of more general optimization problems based on adaptive sparsification.

In this paper, we introduce a unified meta-algorithmic framework for solving asymmetric linear programs via *adaptive sparsification*. We establish that solving a massive, highly over-constrained LP can be rigorously reduced to solving a short sequence of sparse, adaptively sampled sub-problems. This framework provides a principled continuous generalization of the exact constraint sampling introduced by Clarkson [9] for small-dimension LPs, and bridges it with the semi-streaming combinatorial techniques recently developed by Assadi [5] for matching. By casting these diverse techniques through the lens of a binary-loss multiplicative weights update method, we extend the paradigm of adaptive sparsification to encompass low-precision optimization over general asymmetric LPs, and then focus on two applications: (1) a solver designed for mixed packing and covering LPs, and (2) in the quantum regime where we are given row-query access to the constraint matrix.

1.1 Our Contributions

The traditional Plotkin-Shmoys-Tardos (PST) [14] framework for solving linear programs via MWU suffers from a fundamental structural limitation: the iteration complexity is strictly bounded by the square of the problem's width, defined as the two-sided infinite norm $\rho = \max_{\boldsymbol{x}\in\mathcal{D}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_\infty$. Standard MWU algorithms require $O(\rho^2 \log n/\varepsilon^2)$ iterations to converge, where each iteration involves calling a linear optimization oracle over $\mathcal{D}$, and updating the weight fed into the oracle in time $O(\text{nnz}(\boldsymbol{A}))$.

Our primary theoretical contribution is the algebraic circumvention of this quadratic dependency. We parameterize the complexity of the optimization not by the traditional two-sided width ρ, but by the one-sided worst-case violation bound, defined as the non-

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

negative scalar $V_{\max} = \max_{\boldsymbol{x}\in\mathcal{D}} \max_{i\in[n]}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})_i$. We define a "low-violation oracle" that solves small, sampled sub-problems and returns strictly binary indicator vectors $\boldsymbol{v} \in \{0,1\}^n$ corresponding to constraint violations. Because the loss vector is constrained to the boolean domain, the second-order Taylor expansion term of the MWU analysis remains bounded even with constant step size. In the classical analysis, this second order term could be as large as ρ, hence each step is required to have size $1/\rho$. This new analysis allows the framework to tolerate massive, constant step sizes, strictly reducing the overall iteration dependence from quadratic to linear, scaling purely as $O(V_{\max}/\varepsilon \log n)$.

We instantiate this meta-algorithmic framework across classical and quantum regimes, yielding three primary applications:

Reducing Dependency on Input Sparsity in General LPs. By iteratively optimizing over highly sparse sub-problems, our framework mathematically restricts the number of full passes over the constraint matrix $\boldsymbol{A}$ to strictly $O(\frac{V_{\max}}{\varepsilon} \log n)$. This linear decoupling of the error tolerance from the input sparsity is highly advantageous in data-streaming environments where memory access and matrix queries represent the primary computational bottleneck.

▶ **Theorem 1.** *Let the feasibility problem $\{\boldsymbol{x} \in \mathcal{D} : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$, where $\boldsymbol{A} \in \mathbb{R}^{n\times d}$, $\boldsymbol{b} \in \mathbb{R}^n$, $\mathcal{D}$, and let $\varepsilon > 0$. There is a randomized algorithm that finds an ε-approximate solution $\boldsymbol{x}' \in \mathcal{D}$ in the sense that $\boldsymbol{A}\boldsymbol{x}' \leq \boldsymbol{b} + \varepsilon\mathbf{1}$ in time*

$$\widetilde{O}\left(\frac{V_{\max}}{\varepsilon}\left(\mathrm{nnz}(\boldsymbol{A}) + \mathcal{T}_{\mathrm{exact}}\left(d, 24d\frac{V_{\max}}{\varepsilon}\right)\right)\right),$$

where $V_{\max}$ is the one-sided width of the linear program, and $\mathcal{T}_{\mathrm{exact}}(d,n)$ is the time required to find an exact solution to a linear program with d dimensions and n constraints.

Targeted Asymmetric Speedups for Mixed Packing and Covering. We apply our abstraction to the notoriously difficult class of mixed packing and covering linear programs. Because pure covering constraints cannot be violated by more than a constant additive term, their inherent one-sided worst-case violation ($V_{\max}$) is strictly bounded by a constant. By structurally embedding the packing constraints directly into the inner solver's hard solution space $\mathcal{D}$, we are able to selectively sample only the covering constraints.

Using a discretized ε-net over the solver's output space, we show that each of the $\widetilde{O}(\frac{1}{\varepsilon})$ iterations requires sampling only $\widetilde{O}(\frac{d}{\varepsilon})$ covering constraints. Hence, we achieve a one-sided width reduction that accelerates optimization in regimes with tall dense covering matrices and small d. We present these results in Table 1.

Applications to the Quantum Query Model. The main computational bottleneck of our adaptive sparsification framework is the time required to sample the constraints. In the quantum query model, one can sample constraints using an adaptation of Grover's search. In the classical regime, the probabilities are computed by checking violation at each iteration. Since many constraints are violated at each iteration, this approach does not give a $\sqrt{n}$-scaling row-query complexity. Instead, we benefit from our small iteration count to recompute the probability distribution at each iteration. Using this approach gives us an algorithm with row-query complexity $O\left(\sqrt{n}d^3 \mathrm{poly}\log n\right)$ that finds an *exact* solution to a linear program. It is known that an algorithm that finds a constant approximation to a linear program requires at least $\Omega(\sqrt{nd})$ row-queries [21, 3]. In the high-precision regime, cutting plane method achieves $\widetilde{O}(\sqrt{n}d)$ row-queries while interior point methods achieves $\widetilde{O}(\sqrt{n}d^7)$ row-queries [13, 3].

Table 1 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 $\mathrm{nnz} = \mathrm{nnz}(\boldsymbol{P}) + \mathrm{nnz}(\boldsymbol{C})$. $\mathcal{T}_{\mathrm{mixed}}(d, \mathrm{nnz}(\boldsymbol{P}), \mathrm{nnz}(\boldsymbol{C}), \varepsilon)$ denotes the running time of a $1+\varepsilon$ approximate mixed packing and covering LP solver on d variables with $\mathrm{nnz}(\boldsymbol{P})$ (resp. $\mathrm{nnz}(\boldsymbol{C})$) non-zero entries in the packing-constraint (resp. covering-constraint) matrix.

References	Query model	Type	Total runtime in $\widetilde{O}(\cdot)$
[24, 15]	classical	deterministic	$\frac{\mathrm{nnz}}{\varepsilon^2}$
[7]	classical	deterministic	$\frac{r}{\varepsilon}\mathrm{nnz}$
[8]	classical	randomized	$\frac{\mathrm{nnz}}{\varepsilon} + \frac{n}{\varepsilon^2} + \frac{d}{\varepsilon^3}$
Theorem 12	classical	randomized	$\frac{1}{\varepsilon}\left(\mathrm{nnz}(\boldsymbol{C}) + \mathcal{T}_{\mathrm{mixed}}(d, \mathrm{nnz}(\boldsymbol{P}), \widetilde{O}\left(\frac{d}{\varepsilon}r_c\right), \varepsilon)\right)$
Theorem 18	quantum	randomized	$\frac{1}{\varepsilon}\left(\sqrt{n_c\frac{d}{\varepsilon}}\frac{r_c}{\varepsilon} + \mathcal{T}_{\mathrm{mixed}}(d, \mathrm{nnz}(\boldsymbol{P}), \widetilde{O}\left(\frac{d}{\varepsilon}r_c\right), \varepsilon)\right)$

▶ **Theorem 2** (Quantum Version of Clarkson's Algorithm [9]). *Let $\boldsymbol{A} \in \mathbb{R}^{n\times d}$, $\boldsymbol{b} \in \mathbb{R}^n$, $\mathcal{D}$, and $\boldsymbol{c} \in \mathbb{R}^d$. In the row-query model, there is a randomized algorithm that finds an exact solution to the linear program defined by $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ in time*

$$O\left(d\left(\sqrt{n}d^3 + \mathcal{T}_{\mathrm{exact}}(d, 24d^2)\right)\mathrm{poly}\log n\right),$$

where $\mathcal{T}_{\mathrm{exact}}(d,n)$ is the time required to find an exact solution to a linear program with d dimensions and n constraints. Moreover, the algorithm has row-query complexity $\widetilde{O}(\sqrt{n}d^3)$.

1.2 Related Work

Exact Linear Programming and Iterative Reweighting. The foundation of our adaptive constraint sampling approach is rooted in exact algorithms for low-dimensional linear programming. Clarkson [9] introduced an iterative Las Vegas algorithm that adaptively subsamples constraints. By iteratively solving small sub-problems and doubling the weights of violated constraints, Clarkson's algorithm achieves an exact solution while strictly controlling the problem size. However, this classical reweighting scheme is inherently restricted to exact, discrete solutions and does not seamlessly permit continuous approximations.

Combinatorial Streaming and Adaptive Sparsification. Recently, the principles of iterative reweighting have been successfully translated into the semi-streaming model for combinatorial problems. Assadi [5] demonstrated that adaptive sparsification yields a highly efficient approximation for matching in $O(\varepsilon^{-1} \cdot \log n)$ passes, using a multiplicative weight update framework over discrete combinatorial subsets. Concurrently, Quanrud [16] successfully extended this adaptive sparsification paradigm to achieve nearly linear time approximation schemes for matroid intersection in the independence oracle model. Our work formally generalizes the discrete formulations of Assadi and Quanrud into a unified continuous optimization framework.

Mixed Packing and Covering Solvers. For the specialized class of mixed packing and covering linear programs, extensive effort has been dedicated to breaking the traditional width dependence inherent to standard MWU frameworks [14]. For pure covering and pure packing programs, the current state-of-the-art are achieved by width-independent algorithms and run deterministically in time $\widetilde{O}(\frac{\text{nnz}}{\varepsilon})$. These algorithms perform more iteration (order n/ε), but achieve smaller per-iteration cost [1, 23].

For the more general case of mixed packing and covering programs, the current state-of-the-art for width-independent algorithms runs deterministically in $\widetilde{O}(\frac{\text{nnz}}{\varepsilon^2})$ [24, 15]. Regarding width-dependent algorithm, where the width r is defined as the largest row-sparsity of any constraint, the current state-of-the-art is $\widetilde{O}(\frac{r}{\varepsilon}\text{nnz})$ [7] and relies on area-convexity techniques introduced by Sherman [17]. On the other hand, Theorem 12 is a randomized algorithm whose running time scales linearly with respect to $\text{nnz}(\boldsymbol{C})$. When the number of covering constraints is much larger than d, $1/\varepsilon$, and the number of packing constraints, our algorithm is faster. See Table 1 for a comparison.

1.3 Technical Roadmap

The remainder of the paper is structured as follows. In Section 2, we formalize the mathematical distinction between the classical two-sided width (ρ) and the one-sided maximum violation ($V_{\max}$). Section 3 introduces the general Adaptive Sparsification meta-algorithm, formally defining the low-violation oracle and proving the iteration number scaling with $1/\varepsilon$ in the binary-loss MWU framework. Section 4 presents the instantiations for classical exact solvers and mixed packing/covering problems. Finally, Section 5 introduces the dynamic quantum weight-estimation data structure and derives the quantum query LP solver.

2 Preliminaries

2.1 Notation and Linear Programming Variants

We use bold upper case for matrices, and lower case for vectors. We also use $\Delta_n := \{\boldsymbol{p} \in [0,1]^n : \sum_i \boldsymbol{p}_i = 1\}$. We consider a constraint matrix $\boldsymbol{A} \in \mathbb{R}^{n\times d}$ and a vector $\boldsymbol{b} \in \mathbb{R}^n$. A general linear program is formulated as

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

for some bounded domain $\mathcal{D} \subset \mathbb{R}^d$. Let OPT denote the optimal value of this linear program. An ε-approximate solution is a vector $\boldsymbol{x} \in \mathcal{D}$ such that $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} + \varepsilon \mathbf{1}_n$ and $\langle \boldsymbol{c}, \boldsymbol{x}\rangle \geq \text{OPT}$.

In addition to general LPs, we explicitly consider the class of *positive linear programs*. A mixed packing and covering linear program is defined by the feasibility problem of finding $\boldsymbol{x} \in [0,1]^d$ such that $\boldsymbol{P}\boldsymbol{x} \leq \mathbf{1}$ and $\boldsymbol{C}\boldsymbol{x} \geq \mathbf{1}$, where $\boldsymbol{P} \in [0,1]^{n_p\times d}$ is the packing matrix and $\boldsymbol{C} \in [0,1]^{n_c\times d}$ is the covering matrix.[2]

2.2 Two Width Parameters: ρ versus $V_{\max}$

The iteration complexity of classical first-order solvers based on the multiplicative weights update (MWU) method inherently depends on the width of the linear program. Traditionally, the Plotkin-Shmoys-Tardos (PST) framework defines this width via a two-sided infinite norm:

$$\rho := \max_{\boldsymbol{x}\in\mathcal{D}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_\infty \tag{2}$$

[2] This assumption can be made without loss of generality by adding a poly-logarithmic number of variables, see Appendix B of [7].

This metric captures the absolute maximum value of both constraint violations (where $\langle \boldsymbol{A}_{i:}, \boldsymbol{x} \rangle - \boldsymbol{b}_i > 0$) and constraint slacks (where $\langle \boldsymbol{A}_{i:}, \boldsymbol{x} \rangle - \boldsymbol{b}_i < 0$).

In our adaptive sparsification framework, we refine this parameterization by defining the *one-sided worst-case violation bound*, denoted as $V_{\max}$:

$$V_{\max} := \max_{\boldsymbol{x} \in \mathcal{D}} \max_{i \in [n]} (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})_i \,. \tag{3}$$

Since we may restrict attention to instances where some constraint in $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$ is tight at an optimum, we have $V_{\max} \geq 0$. Degenerate cases where all constraints have nonzero slack on $\mathcal{D}$ are trivial. By definition, we always have $V_{\max} \leq \rho$. While ρ heavily penalizes solutions that satisfy constraints with massive slack, $V_{\max}$ is completely oblivious to constraint slackness. This mathematical asymmetry is particularly exploitable in specific constraint formulations, such as mixed packing and covering problems, where the one-sided width of pure covering constraints can be structurally isolated and strictly bounded by a constant, whereas the classical two-sided width ρ would remain heavily dependent on the dimension d. On the other hand, for packing constraints, the one-sided and the two-sided widths are the same.

2.3 Low-Violation Oracles

A central component of our meta-algorithm is the low-violation oracle. An (ε, μ)-low-violation oracle is a procedure that, given any probability distribution $\boldsymbol{p} \in \Delta_n$ over the constraints, outputs a solution $\boldsymbol{x} \in \mathcal{D}$ satisfying $\langle \boldsymbol{c}, \boldsymbol{x} \rangle \geq \text{OPT}$ and $\langle \boldsymbol{p}, \boldsymbol{v}^{\varepsilon}(\boldsymbol{x}) \rangle \leq \mu$, where $\boldsymbol{v}_i^{\varepsilon}(\boldsymbol{x})$ is a binary indicator equal to 1 if $\langle \boldsymbol{A}_{i:}, \boldsymbol{x} \rangle > \boldsymbol{b}_i + \varepsilon$ and 0 otherwise. We provide a formal definition in Definition 4.

2.4 The Quantum Query Model and Boolean Feasibility Oracles

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 [2, 3]. 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 (e.g., QRAM). Instead, we access the constraints strictly 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 *Boolean 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:

$$O_{\boldsymbol{x},\varepsilon} : |i\rangle|\boldsymbol{z}\rangle \mapsto |i\rangle|\boldsymbol{z} \oplus \boldsymbol{v}_i^{\varepsilon}(\boldsymbol{x})\rangle \quad \forall i \in [n], \boldsymbol{z} \in \{0,1\} \tag{4}$$

where $\oplus$ denotes addition modulo 2, and $\boldsymbol{v}_i^{\varepsilon}(\boldsymbol{x}) \in \{0,1\}$ is the binary indicator of violation defined in the previous subsection. The primary metric for the complexity of our quantum algorithms is the *row-query complexity*, defined as the number of calls made to oracles of this type. Because the implementation of the oracle $O_{\boldsymbol{x},\varepsilon}$ relies solely on computing the inner product $\langle \boldsymbol{A}_{i:}, \boldsymbol{x} \rangle$ and comparing it to $\boldsymbol{b}_i + \varepsilon$, each query to $O_{\boldsymbol{x},\varepsilon}$ is modeled precisely as a single row query to the pair $(\boldsymbol{A}, \boldsymbol{b})$. Operating over this Boolean oracle completely decouples the quantum acceleration from the classical solver's continuous mathematical structures, forming the basis for our condition-number-independent quantum analysis.

3 The Meta-Algorithm: Adaptive Sparsification via MWU

In this section, we formalize the general adaptive sparsification framework. Our goal is not to perform a static, one-shot sparsification, which is generally intractable for asymmetric constraints, but rather to construct a small sequence of solutions where any individual constraint is violated by only a strictly bounded fraction of the sequence.

3.1 The Low-Violation Solution Set

We formalize this goal in the following problem.

▶ **Definition 3** ((ε, μ)-Low-Violating Set). *Let $\varepsilon \geq 0$, $\mu > 0$, and suppose we are given a linear program $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c})$ with optimal value* OPT. *Find a set of candidate solutions $\{\boldsymbol{x}^{(t)}\}_{t=1}^{T}$ such that:*

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

$$|\{t \in [T] : \langle \boldsymbol{A}_{i:}, \boldsymbol{x}^{(t)} \rangle > \boldsymbol{b}_i + \varepsilon\}| < \mu \cdot T \tag{5}$$

Rather than attempting to find this set of T solutions simultaneously, we construct them iteratively using the MWU framework.

3.2 The Low-Violation Oracle

To power the MWU framework, we require a call to an oracle that guarantees total low-violation. More formally, given weights to each constraint, a low-violation oracle outputs a solution for which the total weight of violated constraints is small.

▶ **Definition 4** ((ε, μ)-Low-Violation Oracle). *An (ε, μ)-low-violation oracle is a procedure that, given any probability distribution $\boldsymbol{p} \in \Delta_n$ over the constraints, outputs a solution $\boldsymbol{x} \in \mathcal{D}$ satisfying $\langle \boldsymbol{c}, \boldsymbol{x} \rangle \geq \text{OPT}$ and $\langle \boldsymbol{p}, \boldsymbol{v}^{\varepsilon}(\boldsymbol{x}) \rangle \leq \mu$, where $\boldsymbol{v}_i^{\varepsilon}(\boldsymbol{x})$ is a binary indicator equal to 1 if $\langle \boldsymbol{A}_{i:}, \boldsymbol{x} \rangle > \boldsymbol{b}_i + \varepsilon$ and 0 otherwise.*

To build low-violation oracles, Clarkson and Assadi use an inner solver capable of optimizing over a subset of constraints. Our low-violation oracles can be decomposed into two steps. The first is sampling a subset of constraints using the probabilities $\boldsymbol{p}$, and the second is (approximately) solving the LPs defined by the subset of sampled constraints. The size of the sampled subset depends on μ, and the solver used. We provide more details in Section 4.

3.3 Binary Loss MWU and Linear $1/\varepsilon$ Iteration Count

The traditional Plotkin-Shmoys-Tardos (PST) framework requires $O(\rho^2 \log n/\varepsilon^2)$ iterations, bounded by the two-sided width ρ. We demonstrate that by restricting our loss vectors strictly to the Boolean domain (via the low-violation oracle), we can analytically cancel the quadratic dependency on ρ/ε, yielding a linear iteration bound parameterized solely by $V_{\max}$.

▶ **Theorem 5.** *Let $f : \Delta_n \to \{0,1\}^n$ be a routine which, given any probability distribution $\boldsymbol{p} \in \Delta_n$, outputs a binary vector $\boldsymbol{v} \in \{0,1\}^n$ such that $\langle \boldsymbol{p}, \boldsymbol{v} \rangle \leq \mu$. Then we can construct a sequence of distributions $\{\boldsymbol{p}^{(t)}\}_{t=1}^{T}$ such that the corresponding outputs $\{\boldsymbol{v}^{(t)}\}_{t=1}^{T}$ satisfy:*

$$\max_{i \in [n]} \left\{ \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{v}_i^{(t)} \right\} \leq 3\mu \tag{6}$$

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

Proof. We track the accumulation of violations using the softmax function $\operatorname{smax}(\boldsymbol{x}) = \log \sum_{i=1}^{n} \exp(\boldsymbol{x}_i)$. The gradient is exactly our probability distribution:

$$\nabla \operatorname{smax}(\boldsymbol{x}) = \frac{\exp(\boldsymbol{x})}{\sum_{i=1}^{n} \exp(\boldsymbol{x}_i)} \,. \tag{7}$$

Crucially, because our oracle returns strictly binary vectors $\boldsymbol{v}^{(t)} \in \{0,1\}^n$ rather than continuous magnitudes, we satisfy the condition $\|\boldsymbol{\delta}\|_\infty \leq 1$ with strictly non-negative entries. For such binary updates, the smax function entirely escapes the second-order Taylor expansion penalty that typically produces the ρ^2 term. Instead, we have the tight algebraic bound:

$$\operatorname{smax}(\boldsymbol{x} + \boldsymbol{\delta}) \leq \operatorname{smax}(\boldsymbol{x}) + 2\langle \nabla \operatorname{smax}(\boldsymbol{x}), \boldsymbol{\delta} \rangle \,. \tag{8}$$

By iteratively setting our distribution $\boldsymbol{p}^{(t)} = \nabla \operatorname{smax}\left(\sum_{\tau=1}^{t-1} \boldsymbol{v}^{(\tau)}\right)$ and substituting $\boldsymbol{\delta} = \boldsymbol{v}^{(t)}$, we can apply (8) sequentially. Because $\langle \boldsymbol{p}^{(t)}, \boldsymbol{v}^{(t)} \rangle \leq \mu$ by the definition of the oracle, we obtain:

$$\operatorname{smax}\left(\sum_{t=1}^{T} \boldsymbol{v}^{(t)}\right) \leq \operatorname{smax}(0) + 2\sum_{t=1}^{T} \langle \boldsymbol{p}^{(t)}, \boldsymbol{v}^{(t)} \rangle \leq \log n + 2T\mu \,. \tag{9}$$

Since $\max_i \boldsymbol{x}_i \leq \operatorname{smax}(\boldsymbol{x})$, dividing by T yields:

$$\max_{i \in [n]} \left\{ \frac{1}{T} \sum_{t=1}^{T} \boldsymbol{v}_i^{(t)} \right\} \leq \frac{\log n}{T} + 2\mu \,. \tag{10}$$

Setting $T = \frac{\log n}{\mu}$ bounds the maximum fractional violation by 3μ. ◀

A direct corollary of Theorem 5 is that we are able to find a (ε, μ)-low-violating set using $O(\frac{\log n}{\mu})$ calls to a $(\varepsilon, \mu/3)$-low-violation oracle.

▶ **Theorem 6.** *Consider a linear program of the form* $\max_{\boldsymbol{x}} \langle \boldsymbol{c}, \boldsymbol{x} \rangle$, *subject to* $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}$. *Assume we are given a* $(\varepsilon, \mu/3)$*-low-violation oracle (see Definition 4) with running time* $\mathcal{T}$, *then in* $T = O\left(\frac{\log n}{\mu}\right)$ *iterations, we are able to compute a* (ε, μ)*-low-violation set (see Definition 3)* $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}\}$. *Moreover, each iteration runs in time* $O(\operatorname{nnz}(\boldsymbol{A}) + \mathcal{T})$.

4 Applications

In this section, we focus on corollaries of Theorem 6. We first develop the techniques from [9, 5] 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. The proofs can be found in the full version [25].

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 decompose low-violation oracles as three-step procedures:

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 low-violation oracle is an exact solver, we use Clarkson's sampling lemma (Lemma 21 from [9]) to bound the required number of sampled constraint. When we use any other solver that has a sparse output set, we bound the number of sampled constraints using a technique similar to Assadi [5].

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.[3]

▶ **Lemma 7** (Corollary of [9]). *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 [5] 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 8.** *Consider a linear program of the form* (1)*. 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 requires 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 defined by 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.*

4.2 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 $\langle \boldsymbol{c}, \boldsymbol{x}\rangle \geq \mathrm{OPT}$ and $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} + \varepsilon\boldsymbol{1}$. First, we reduce this problem to finding a low-violation set (see Definition 3). For a solver $\mathcal{A}$ whose outputs are in $\mathcal{D}$, we consider the largest violation possible in $\mathcal{D}$: $V_{\max} := \max_{\boldsymbol{x}\in\mathcal{D}} \max_{i\in[n]} (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})_i$.

▷ Claim 9. For $\varepsilon > 0$, and a linear program of the form (1). 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 $\mathrm{nnz}(\boldsymbol{A})$ time, it is possible to find $\boldsymbol{x}$ such that $\boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} + 2\varepsilon\boldsymbol{1}$.

[3] See Lemma 21 for the definition of the sampling lemma.

Proof. Using Theorem 6, 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 obtain a low-violation set with parameters ε and $\mu = \varepsilon/V_{\max}$.

Given an $(\varepsilon, \varepsilon/V_{\max})$ low-violation set $\{\boldsymbol{x}^{(j)}\}_{t\in[T]}$, consider for a constraint i, let $V_i \subseteq [T]$ be the indices of solutions violating constraint i: $\{t \in [T] : \langle \boldsymbol{A}_{i:}, \boldsymbol{x}^{(t)}\rangle > \boldsymbol{b}_i + \varepsilon\}$. We have

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

Hence, $\bar{\boldsymbol{x}} := \frac{1}{T}\sum_{t\in[T]} \boldsymbol{x}^{(t)}$ is 2ε-feasible. Moreover, we have $\langle \boldsymbol{c}, \bar{\boldsymbol{x}}\rangle \geq \mathrm{OPT}$ since for any $t \in [T]$, $\langle \boldsymbol{c}, \boldsymbol{x}^{(t)}\rangle \geq \mathrm{OPT}$, hence $\bar{\boldsymbol{x}}$ is a 2ε-approximation. ◁

Using an exact solver together with Lemma 7, it is possible to obtain a $(\varepsilon/2, \varepsilon/(6V_{\max}))$-low-violation oracle. It follows from Claim 9 that this oracle can then be used to prove the following theorem.

▶ **Theorem 10.** *For $\boldsymbol{A} \in \mathbb{R}^{n\times d}$, $\boldsymbol{b} \in \mathbb{R}^n$, $\boldsymbol{c} \in \mathbb{R}^d$, $\mathcal{D}$ and $\varepsilon > 0$. Consider the 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*

$$O\left(\frac{V_{\max}}{\varepsilon} \log n \left(\mathrm{nnz}(\boldsymbol{A}) + \mathcal{T}_{\mathrm{exact}}\left(d, 24d\frac{V_{\max}}{\varepsilon}\right)\right)\right),$$

where $\mathcal{T}_{\mathrm{exact}}(d, n)$ 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.

4.3.1 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 solving 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:

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

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.

4.3.2 Reducing the Cardinality of the Output Space

A primary challenge in applying Lemma 8 to mixed packing and covering problems is the size of the output space. 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 $\boldsymbol{y}$ satisfying $\boldsymbol{x} \leq \boldsymbol{y} \leq (1+\varepsilon)\boldsymbol{x}$ is a $(1+4\varepsilon)$-approximation, coordinate-wise. 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 11. For $0 < \varepsilon \leq 1$. Given $\boldsymbol{x}$, a $1+\varepsilon$ approximation of a mixed packing and covering problem with d variables, it is possible to find a $(1+4\varepsilon)$-approximation $\tilde{\boldsymbol{x}}$ for the same mixed packing and covering program such that $\tilde{\boldsymbol{x}} \in \mathcal{S}$ where $|\mathcal{S}| \leq \lceil 2 + 4\frac{\log(d/\varepsilon)}{\varepsilon} \rceil^d$.

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 12.** *Given a mixed packing and covering program with packing constraints $\boldsymbol{P} \in [0,1]^{n_p \times d}$ and covering constraints $\boldsymbol{C} \in [0,1]^{n_c \times d}$, it is possible to find with high-probability a $1 + O(\varepsilon)$-approximation $\boldsymbol{x}$ or certify infeasibility in time*

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

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

Inside the low-violation oracle, we can use state-of-the-art solvers for mixed packing and covering programs such as [8, 7, 15].

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's 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's 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's search becomes probabilistic. Instead of using Grover's 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 13 to sample the constraints using queries to the binary violation oracle.

▶ **Lemma 13** (Claim 3, [2]). *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.*

More details are given in the full version of the paper [25].

Procedure 1 `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 2

3: $\widetilde{W'} \leftarrow \frac{\widetilde{W}}{2s}\left(\sqrt{3 + 2|S|} - \sqrt{3}\right)^2$

4: **return** $\widetilde{W'}$

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 13, 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 $\boldsymbol{p} \leq \boldsymbol{q} \leq 2\boldsymbol{p}$. This ensures that on average, $2s$ constraints are sampled, hence the number of queries to $\boldsymbol{p}$ and to $\boldsymbol{q}$ in Lemma 13 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 design, at iteration t, we have $\boldsymbol{w}_i = e^{\sum_{\tau=1}^{t-1} \boldsymbol{v}_i^{\varepsilon}(\boldsymbol{x}^{(\tau)})}$. 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 $(\boldsymbol{A}, \boldsymbol{b})$.

Computing the exact normalization factor $W = \|\boldsymbol{w}\|_1$ requires aggregating all n weights, an $\Omega(n)$ classical operation that would completely negate our desired sublinear quantum query complexity. Therefore, instead of sampling from $\boldsymbol{p}$ directly, we simulate the process by sampling from an unnormalized distribution $\boldsymbol{q}$ satisfying $\boldsymbol{p} \leq \boldsymbol{q} \leq 2\boldsymbol{p}$. By maintaining a constant-factor under-approximation $\widetilde{W} \in [\frac{1}{2}W, W]$, we can implicitly define our proxy distribution as $\boldsymbol{q} = \boldsymbol{w}/\widetilde{W}$. Because $\widetilde{W}$ bounds W from below, this strictly guarantees $\boldsymbol{p} \leq \boldsymbol{q} \leq 2\boldsymbol{p}$. Consequently, querying $\boldsymbol{q}$ carries the same asymptotic cost as querying $\boldsymbol{p}$, while limiting the expected number of sampled constraints to at most $2s$.

To efficiently maintain this approximation $\widetilde{W}$ across iterations, we rely on a fundamental structural property of the MWU framework: the total weight W fluctuates by at most a constant factor between any two consecutive steps. This allows us to bootstrap the estimation process. By using the approximation from iteration $t-1$ as a baseline, we can accurately estimate the new normalization factor for iteration t using only a constant number of quantum samples combined with standard concentration bounds.

▷ Claim 14. 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 $\widetilde{W} \leq \|\boldsymbol{w}\|_1 \leq C\widetilde{W}$, then Procedure 1 outputs $\widetilde{W'}$ such that with probability at least $1 - p$, $\widetilde{W'} \leq \|\boldsymbol{w}\|_1 \leq 2\widetilde{W'}$ using $\widetilde{O}\left(\sqrt{n} \log \frac{1}{p}\right)$ queries to $\boldsymbol{w}$.

Procedure 2 `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 $\min\{\frac{s}{\widetilde{W}}\boldsymbol{w}_i, 1\}$.

1: **for** $r = 1$ to $1 + 5\log\frac{1}{p}$ **do**
2: $\quad S_r \leftarrow$ Output of algorithm from Lemma 13 with probabilities $\min\{\frac{s}{\widetilde{W}}\boldsymbol{w}, 1\}$
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** $\bar{S}$

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 too 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 13 satisfies the probability requirement of Lemmas 7 and 8.

However, the set S may have a lot more constraints. We use the following claim to ensure it does not happen with high probability.

▷ Claim 15. 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 2 outputs a set S such that with probability at least $1 - p$:

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 14 and 15 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 14 uses far fewer queries to $\boldsymbol{w}$ (hence to $(\boldsymbol{A}, \boldsymbol{b})$) compared to Claim 15, 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, otherwise the algorithm should continue. We use Grover's 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 result.

▷ Claim 16 (Folklore). 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 16 does not fail throughout iterations with high probability.

▶ **Theorem 17.** *For a linear program of the form* (1) *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*

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

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 12.

▶ **Theorem 18.** *Given a mixed packing and covering problem, it is possible to find a $1+O(\varepsilon)$-approximation $\boldsymbol{x}$ or certify infeasibility with high probability using $\widetilde{O}(\sqrt{n_c \frac{d}{\varepsilon}} \cdot \frac{1}{\varepsilon^2})$ row-queries to $\boldsymbol{C}$. Moreover, the algorithm runs in time*

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

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

5.4 Quantum Speedups for Low-Precision Solvers

We provide two quantum versions of Theorem 10. For low-precision solvers, we can perform the same trick to sample constraints using row-queries to $\boldsymbol{A}$.

▶ **Theorem 19.** *For a linear program of the form* (1) *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*

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

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 10 and 19.

▶ **Theorem 20.** *For a linear program of the form* (1) *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*

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

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.*

References

1 Zeyuan Allen-Zhu and Lorenzo Orecchia. Nearly linear-time packing and covering LP solvers. *Math. Program.*, 175(1-2):307–353, May 2019. doi:10.1007/s10107-018-1244-x.

2 Simon Apers and Ronald de Wolf. Quantum Speedup for Graph Sparsification, Cut Approximation, and Laplacian Solving. *SIAM Journal on Computing*, 2023.

3 Simon Apers and Sander Gribling. Quantum Speedups for Linear Programming via Interior Point Methods. *SIAM Journal on Computing*, 55(1):93–134, February 2026. doi:10.1137/25M1736098.

4 Sanjeev Arora, Elad Hazan, and Satyen Kale. The Multiplicative Weights Update Method: A Meta-Algorithm and Applications. *Theory of Computing*, 8(6):121–164, May 2012. doi:10.4086/toc.2012.v008a006.

5 Sepehr Assadi. A Simple $(1-\varepsilon)$-Approximation Semi-Streaming Algorithm for Maximum (Weighted) Matching. *TheoretiCS*, Volume 4, August 2025. doi:10.46298/theoretics.25.16.

6 András A. Benczúr and David R. Karger. Randomized Approximation Schemes for Cuts and Flows in Capacitated Graphs. *SIAM Journal on Computing*, 44(2):290–319, January 2015. doi:10.1137/070705970.

7 Digvijay Boob, Saurabh Sawlani, and Di Wang. Faster width-dependent algorithm for mixed packing and covering LPs. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

8 Chandra Chekuri and Kent Quanrud. Randomized MWU for Positive LPs. In *Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 358–377. Society for Industrial and Applied Mathematics, January 2018. doi:10.1137/1.9781611975031.25.

9 Kenneth L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small. *J. ACM*, 42(2):488–499, March 1995. doi:10.1145/201019.201036.

10 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, pages 938–942, New York, NY, USA, June 2019. Association for Computing Machinery. doi:10.1145/3313276.3316303.

11 Michael D. Grigoriadis and Leonid G. Khachiyan. A sublinear-time randomized approximation algorithm for matrix games. *Operations Research Letters*, 18(2):53–58, September 1995. doi:10.1016/0167-6377(95)00032-0.

12 Yin Tat Lee and Aaron Sidford. Efficient Inverse Maintenance and Faster Algorithms for Linear Programming. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 230–249, October 2015. doi:10.1109/FOCS.2015.23.

13 Yin Tat Lee, Aaron Sidford, and Sam Chiu-Wai Wong. A Faster Cutting Plane Method and its Implications for Combinatorial and Convex Optimization. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 1049–1065, October 2015. doi:10.1109/FOCS.2015.68.

14 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*, pages 495–504, October 1991. doi:10.1109/SFCS.1991.185411.

15 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, pages 69–80. Society for Industrial and Applied Mathematics, December 2019. doi:10.1137/1.9781611976014.11.

16 Kent Quanrud. Adaptive Sparsification for Matroid Intersection. *LIPIcs, Volume 297, ICALP 2024*, 297:118:1–118:20, 2024. doi:10.4230/LIPIcs.ICALP.2024.118.

17 Jonah Sherman. Area-convexity, ℓ_∞ regularization, and undirected multicommodity flow. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 452–460, New York, NY, USA, June 2017. Association for Computing Machinery. doi:10.1145/3055399.3055501.

18 Steve Smale. Mathematical problems for the next century. *The Mathematical Intelligencer*, 20(2):7–15, March 1998. doi:10.1007/BF03025291.

19 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, pages 563–568, New York, NY, USA, May 2008. Association for Computing Machinery. doi:10.1145/1374376.1374456.

20 P.M. Vaidya. Speeding-up linear programming using fast matrix multiplication. In *30th Annual Symposium on Foundations of Computer Science*, pages 332–337, October 1989. doi:10.1109/SFCS.1989.63499.

21 Joran van Apeldoorn, András Gilyén, Sander Gribling, and Ronald de Wolf. Quantum SDP-Solvers: Better upper and lower bounds. *Quantum*, 4:230, February 2020. doi:10.22331/q-2020-02-14-230.

22 Jan van den Brand, Yin Tat Lee, Aaron Sidford, and Zhao Song. Solving tall dense linear programs in nearly linear time. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2020, pages 775–788, New York, NY, USA, June 2020. Association for Computing Machinery. doi:10.1145/3357713.3384309.

23 Di Wang, Satish Rao, and Michael W. Mahoney. Unified Acceleration Method for Packing and Covering Problems via Diameter Reduction. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:13, Dagstuhl, Germany, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2016.50.

24 Neal E. Young. Nearly Linear-Work Algorithms for Mixed Packing/Covering and Facility-Location Linear Programs, November 2014. arXiv:1407.3015.

25 Étienne Objois and Adrian Vladu. Adaptive sparsification for linear programming, 2025. doi:10.48550/arXiv.2510.08348.

A Recovering Clarkson's and Assadi's Results

A.1 Recovering Clarkson's Result

We first show how one can recover the result from Clarkson [9] 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 Definition 3. 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 find a (ε, μ)-low-violation set (see Definition 3) with $\varepsilon = 0$ and $\mu = 1/(d+1)$. Indeed, assume for any $i \in [n]$, one has $\left|\{t \in [T] : \boldsymbol{x}^{(t)} \text{ violates constraint } i\}\right| < \frac{T}{d}$. If we take in particular the d constraints from B, we obtain

$$\sum_{i \in B} \left|\{t \in [T] : \boldsymbol{x}^{(t)} \text{ violates constraint } i\}\right| < d \cdot \frac{T}{d} = T. \tag{11}$$

Assume none of the $\{\boldsymbol{x}^{(t)}\}_{t=1}^{T}$ is a solution of the LP. That means, each $\boldsymbol{x}^{(t)}$ violates at least one constraint from B. Hence, we have $\sum_{i \in B} \left|\{t \in [T] : \boldsymbol{x}^{(t)} \text{ violates constraint } i\}\right| \geq T$. This contradicts (11) which means at least one vector from $\{\boldsymbol{x}^{(t)}\}_{t=1}^{T}$ satisfies all the constraint from B. Hence, we have an exact solution of the LP.

In order to obtain this $(0, 1/(d+1))$-low-violation set, Clarkson uses the following sampling lemma.

▶ **Lemma 21** (Sampling lemma [9]). *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*

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

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

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

We are now ready to state Clarkson's main result as a corollary of our framework.

▶ **Corollary 22** (Theorem in [9]). *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*

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

A.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 23** (Lemma 3.3 and 4.3 from [5]). *Consider a graph* G *with* n *vertices and* m *edges.*[4]

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* $\exp\left(3n \cdot \log(nW)\right)$ *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.

[4] For the proof, see the proof of equations (4) and (7) of [5].

▶ **Corollary 24** (Theorem in [5]). *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*

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

   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.
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*

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

   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.

B Quantum Algorithms

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

$X_0 \leftarrow \emptyset$
$p \leftarrow \frac{1}{32n\log n}\cdot\frac{1}{100n^2}$
$\widetilde{W}_0 \leftarrow n$
$s \leftarrow 6d^2$
for $t=0$ to $T=24d\log n$ **do**
    Define the query access to $\boldsymbol{w}(X)$ as $2^{\sum_{\boldsymbol{x}\in X}\boldsymbol{v}^0(\boldsymbol{x})}$
    $S_t \leftarrow$ `quantum_sampling`$(\boldsymbol{w}(X_t), s, \widetilde{W}_t, p)$ ▷ Using Procedure 2.
    $\boldsymbol{x} \leftarrow \mathcal{A}(S_t)$
    $X_{t+1} \leftarrow X_t \cup \{\boldsymbol{x}\}$
    $\widetilde{W}_{t+1} \leftarrow$ `estimation_sum`$(\boldsymbol{w}(X_{t+1}), \widetilde{W}_t, p)$ ▷ Using Procedure 1.
end for
for $\boldsymbol{x} \in X_T$ **do**
    if $\boldsymbol{x}$ is feasible **then** ▷ By invoking Claim 16.
        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

$X_0 \leftarrow \emptyset$
$p \leftarrow \frac{1}{32n \log n} \cdot \frac{1}{100n^2}$
$\widetilde{W}_0 \leftarrow n_c$
$s \leftarrow 6\frac{d}{\varepsilon} \log \frac{\log(r_p/\varepsilon)}{\varepsilon}$
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 2.
  $\boldsymbol{C}_t \leftarrow$ 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 1.
end for
return $\sum_{\boldsymbol{x} \in X_T} \boldsymbol{x} / |X_T|$.

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.

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

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