Solving Linear and Integer Programs Faster Using Data

Yuhan Ye
MIT
yyh03@mit.edu

Omar Bennouna
MIT
omarben@mit.edu

Abstract

We study how much a linear program (LP) or an integer program (IP) can be compressed when solved repeatedly over a fixed feasible region with varying objective coefficients. We provide a framework that compresses LP and IP formulations with guarantees on the probability of preserving optimality, using either an uncertainty set for the cost vector or historical samples. Our framework exposes a tunable tradeoff between compression and the probability of recovering an optimal solution. We characterize the minimum dimension of affine reformulations preserving the full optimizer set over a given cost region and provide sample-based learning algorithms with fast-rate guarantees, without requiring a nondegeneracy assumption for LPs. With high confidence over the training data, the learned reformulation preserves the optimizer set for an independent test cost with probability at least $1-\rho-\widetilde{O}(d_\rho^\star/\hat{n})$, where ρ bounds the excluded probability mass, $d_\rho^\star$ is the retained region's decision-relevant dimension, and $\hat{n}$ is the number retained from n training samples. Existing data-driven approaches to LP and IP reduction do not, to our knowledge, learn a single low-dimensional affine reformulation with a prescribed probability of preserving the full optimizer set. Experimental results show substantial improvements over the compared baselines in accuracy, sample efficiency, and computational speed.

1 Introduction

Many decision systems repeatedly solve linear programs (LPs) or integer programs (IPs) over the same feasible region while the objective changes. This setting arises in routing, resource allocation, network operations, and other large-scale planning tasks. In such applications, one would like to compress the repeated optimization problem once and reuse the compressed representation on future instances. Existing approaches address this goal by learning or sampling low-dimensional projections for LPs [Sakaue and Oki, 2024, Iwata and Sakaue, 2025, Vu et al., 2018], or by predicting variable fixings and other affine restrictions for IPs [Xavier et al., 2021, Nair et al., 2020]. These methods can yield substantial speedups, but may either incur large optimality gaps or require a significant time to learn compressed formulations, and do not guarantee a prescribed probability of recovering an optimal solution.

In this paper, we study how to find the lowest-dimensional reformulation of an LP or IP that preserves optimality with a prescribed probability. Let

$$\min_{x\in\mathcal{X}} c^\top x \tag{1}$$

be a family of LPs or IPs with a fixed nonempty bounded feasible set $\mathcal{X}$ and a cost vector c known to lie in a prior set $\mathcal{C}\subseteq\mathbb{R}^d$. For LPs, $\mathcal{X}=\{x\in\mathbb{R}^d : Ax\le b\}$ is a polytope. For IPs, $\mathcal{X}=\mathcal{X}_I:=\{x\in\mathbb{Z}^d : Ax\le b\}$ is a bounded polyhedron intersected with $\mathbb{Z}^d$. The central question we propose is:

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

> When an LP or an IP is solved repeatedly with varying costs, how much can future instances be reduced in dimension while guaranteeing a probability $1-\rho$ of optimality, given prior knowledge about the cost vector?

Here, prior knowledge takes the form of either a set $\mathcal{C} \subset \mathbb{R}^d$ containing the cost vector, historical samples of c, or both. We first focus on the known-prior-set setting and later extend our results to the case where only historical samples are available. Such problems are common in the optimization literature, with examples including shortest-path routing [Delling et al., 2017], min-cost flow [Ahuja et al., 1995, Frangioni and Manca, 2006], knapsack [Mandi et al., 2020], and traveling-salesperson problems [Avraham and Raviv, 2020]. More generally, contextual linear and integer optimization models predict objective coefficients from context while keeping the feasible region fixed. In such settings, compression can speed up repeated LP or IP solves while controlling the probability of preserving optimality, and reduce the amount of information that must be learned about c.

Against this background, we show throughout the paper that the quantity governing LP and IP compression is not the ambient dimension d, but the dimension of the *decision-relevant subspace*: the subspace spanned by differences between optimal solutions reachable when the cost vector c belongs to a confidence set $\mathcal{C}$. Crucially, even when d is very large, this subspace can be low-dimensional. Building on Bennouna et al. [2025] and Bennouna et al. [2026], when $\mathcal{C}$ is convex and open, this subspace can be written as

$$\mathrm{dir}(\mathcal{X}^\star(\mathcal{C})) := \mathrm{span}\{x - x' : x, x' \in \mathcal{X}^\star(\mathcal{C})\},$$

where $\mathcal{X}^\star(\mathcal{C})$ denotes the set of all optimal solutions as c varies over $\mathcal{C}$. Equivalently, $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$ is the span of differences between reachable optimal solutions. An important insight from their results is that a set of function evaluations $c^\top q_1, \ldots, c^\top q_r$ contains the necessary information to recover the solution set of (1) if and only if $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C})) \subseteq \mathrm{span}(q_1, \ldots, q_r)$. Such a set $\{q_1, \ldots, q_r\}$ is called a *sufficient decision dataset* (SDD). The key implication is that the optimal solution depends on c only through its projection onto $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$, and $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$ itself is the smallest subspace with this property.

Recent work by Ye et al. [2026] clarifies the computational and statistical landscape of this characterization. On the negative side, constructing a basis of the decision-relevant space $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$ is NP-hard and coNP-hard. On the positive side, the same work shows that one can learn decision-relevant directions from i.i.d. cost vector samples. In particular, they provide an algorithm that outputs a set of vectors $\{q_1, \ldots, q_r\}$ such that, with high confidence over the training sample, the measurements $c^\top q_1, \ldots, c^\top q_r$ have enough information to solve (1) for an independent cost drawn from the same distribution with probability at least $1 - \widetilde{O}(d^\star/n)$ where n is the sample size and $d^\star := \dim\ \mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$.

From the standpoint of LP and IP compression, two challenges remain. First, the existing SDD literature [Bennouna et al., 2025, 2026] characterizes which measurements suffice to recover optimal decisions, but it does not provide compressed reformulations of LPs induced by the decision-relevant subspace. Second, the sample-based guarantee for the algorithm in Ye et al. [2026] assumes a strong nondegeneracy condition, whereas many repeated-LP and IP models of practical interest are degenerate.

1.1 Our contributions

1. **Exact reduced LPs and IPs with a fast-rate learning certificate.** We first study the problem of finding an exact LP or IP reformulation when the cost vector ranges over a known confidence set $\mathcal{C}$. We turn the decision-relevant subspace characterization of Bennouna et al. [2025, 2026] into an explicit solver-side reduction: any subspace containing $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$ yields an exact reduced LP or IP, and a basis of $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$ gives a minimal exact $d^\star$-variable reformulation (Propositions 2 and 15). Unlike approximate projection methods, which treat the projection as a black-box parameter to be optimized without identifying what makes it optimal, our framework reveals $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$ as the unique object that governs exact compression. When the prior set is known, this reduction requires no historical cost samples, although computing it can be expensive. With high confidence over the training data, our sample-based algorithms preserve the optimizer set for an independent test cost drawn from the same distribution with probability at least $1-\widetilde{O}(d^\star/n)$. This exactness-failure bound has a $1/n$ dependence, while approximate methods bound objective-value

generalization gaps at rate $\widetilde{O}(1/\sqrt{n})$. In contrast with the learner of Ye et al. [2026], our approach does not require any nondegeneracy assumption (Theorems 5 and 16).

2. **Sample-based compression at a prescribed probability of optimality.** We extend our framework to the setting where the uncertainty set $\mathcal{C}$ is unknown, using historical samples to construct a calibrated confidence set and learn a compressed LP or IP formulation. With high confidence over the training data, the resulting reformulation preserves the optimizer set for an independent test cost with probability at least $1 - \rho - \widetilde{O}(d^\star_\rho/\hat{n})$, where ρ bounds the probability mass outside the confidence set, $d^\star_\rho$ is its decision-relevant dimension, and $\hat{n}$ is the number retained from n training samples (Theorem 7 and its IP extension in Appendix C.2). This guarantee allows the confidence-set coverage and sample size to be calibrated to a prescribed probability of optimality, exposing a tunable tradeoff between compression and optimality.
3. **Faster learning, fewer training samples, and higher accuracy.** We compare our method with projection-learning baselines based on Sakaue and Oki [2024], Iwata and Sakaue [2025] and a random variable-selection baseline [Akchen and Mišić, 2024] for LPs. For IPs, we compare with baselines [Xavier et al., 2021, Bertsimas and Stellato, 2022, Nair et al., 2020, Li et al., 2025]. The experiments cover synthetic and benchmark instances (Section 4 and Appendix C.3). We show that our exact reduction substantially outperforms the projection learning baselines in terms of data requirement, accuracy, and speed, and recovers the original optimal value more often than the variable-fixing baseline. Our method can also be parallelized when optimizers are unique almost surely, because each training LP or IP can be solved independently. Even without parallelization, our method learns the compression substantially faster than the compared baselines.

1.2 Related work

Decision-sufficient representations for linear programs. Bennouna et al. [2025] and Bennouna et al. [2026] characterize what information about the cost vector is needed to recover optimal decisions. Their key finding is that optimality is governed by a single geometric object, the subspace $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$ spanned by differences between reachable optimal solutions. For open convex priors, a set of linear measurements $\{c^\top q_1, \ldots, c^\top q_r\}$ is sufficient to solve (1) if and only if the span of the directions $\{q_1, \ldots, q_r\}$ contains $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$. Ye et al. [2026] move to the data-driven setting, where the cost vector is random but samples are available, and show that directions in $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$ can be learned incrementally from cost samples with provable out-of-sample guarantees. Our work builds on this line of research but shifts the focus from information-theoretic sufficiency to optimization: we ask how a known $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$ yields a smaller optimization problem, how to learn this subspace under weaker geometric assumptions, and how to calibrate a high-coverage working prior when the prior set $\mathcal{C}$ is unknown.

Approximate projections for repeated LPs. Random projections reduce the size of LPs while approximately preserving their objective values [Vu et al., 2018, Poirion et al., 2023]. Sakaue and Oki [2024] learn a shared projection for repeated LPs and provide generalization bounds based on pseudo-dimension. Iwata and Sakaue [2025] instead learn a projection for each instance, allowing them to handle heterogeneous LP families. Projection methods have also been studied for quadratic programs [d'Ambrosio et al., 2020, Nguyen and Nguyen, 2026]. Whereas these LP projection methods target objective-value performance, our learning algorithms are guided by a geometric characterization of the minimum-dimensional affine restriction preserving the full optimizer set over a given cost region. We compare our method with projection baselines in Section 4.

Learning reduced IP formulations. Xavier et al. [2021] predict which integer variables to fix and which pairs of variables should have the same value, while Neural Diving [Nair et al., 2020] predicts values for a subset of the integer variables and solves the remaining MIP. Other approaches predict which constraints are tight [Li et al., 2026] or learn to select a reduced formulation from a finite set of strategies [Bertsimas and Stellato, 2021, 2022, Li et al., 2025]. Bertsimas and Stellato [2021, Theorem 3.1] provide a bound on the probability that a new instance requires an optimal strategy outside this set, but their IP formulations only consider integer assignments seen during training. Our approach instead learns a single affine reformulation that can recover optimal integer solutions not seen during training and provides a guarantee on the probability of preserving the optimizer set for a new cost vector, with a sample requirement that depends on the decision-relevant dimension.

Additional related work is given in Section A.

2 Preliminaries

We first introduce the notation. Consider the polyhedral LP

$$\min_{x\in\mathcal{X}} c^\top x, \qquad \mathcal{X} := \{x \in \mathbb{R}^d : Ax \le b\}, \tag{2}$$

where the cost vector c lies in a convex prior set $\mathcal{C} \subseteq \mathbb{R}^d$. Here, $\mathcal{X}$ is the continuous feasible polytope and $\mathcal{X}^\angle$ denotes its vertices. For IPs, we optimize over $\mathcal{X}_I = \mathcal{X} \cap \mathbb{Z}^d$, which we assume is nonempty. For $c \in \mathbb{R}^d$, set $\mathcal{X}^\star(c) := \arg\min_{x\in\mathcal{X}} c^\top x$ and $\mathcal{X}^\star(\mathcal{C}) := \bigcup_{c\in\mathcal{C}} \mathcal{X}^\star(c)$. Following Bennouna et al. [2025, 2026], $\mathcal{D} := \{q_1, \dots q_r\} \subset \mathbb{R}^d$ is a sufficient decision dataset (SDD) for $(\mathcal{X}, \mathcal{C})$ if for any $c \in \mathcal{C}$ the observation of the evaluations $c^\top q_1, \dots, c^\top q_r$ can be mapped to the solution set $\arg\min_{x\in\mathcal{X}} c^\top x$, i.e. they contain sufficient information to find the solution to (1). Bennouna et al. [2025, 2026] argue that the quantity of interest that determines the information required to solve (1) is the decision-relevant space $\dim(\mathcal{X}^\star(\mathcal{C}))$, which is equal to the span of all directions along which reachable optimal solutions can differ. In particular, they provide the following exact characterization.

Theorem 1. *Assume $\mathcal{C}$ is open and convex. A finite dataset $\mathcal{D} \subseteq \mathbb{R}^d$ is an SDD for $(\mathcal{X}, \mathcal{C})$ if and only if $\dim(\mathcal{X}^\star(\mathcal{C})) \subseteq \operatorname{span}(\mathcal{D})$. In particular, the minimum cardinality of an SDD is $d^\star := \dim \dim(\mathcal{X}^\star(\mathcal{C}))$.*

This characterization has consequences not only from an experimental design perspective, but also for the computational speedup of LPs and IPs. In particular, we show that a basis for the span of a sufficient dataset, together with a reachable optimizer as anchor, yields a reformulation of (1) whose dimension equals the rank of the dataset. The question we address can be stated rigorously as follows: given an LP or IP of the form (1) with a random cost vector c, and given either an uncertainty set for c or samples from its distribution, how can we construct a lower-dimensional formulation of the form

$$\min_{y\in\mathcal{X}'} c'^\top y, \tag{3}$$

where c' and $\mathcal{X}'$ are built from c and $\mathcal{X}$, such that an optimizer y can be mapped to a solution of (1) with high probability over a new cost?

We show that both settings, samples from the distribution of c and an uncertainty set for c, can be handled within the same framework. The key step in obtaining a lower-dimensional reformulation of (1) is to construct a sufficient dataset for the relevant costs. Two approaches are available. If an uncertainty set is available and either the LP instance is of moderate size or sufficient computational power is available, Bennouna et al. [2025] formulate a mixed-integer program (MIP) that iteratively constructs a sufficient dataset for LPs. When the instance is too large for an MIP to be tractable or is an IP, and data is available, we instead process cost samples sequentially, learning a new decision-relevant direction only when the current dataset fails to recover the full optimizer set at the incoming sample. The dataset therefore grows only on informative samples and contains at most $d^\star$ directions.

3 LP and IP Compression via Sufficient-Decision Subspaces

3.1 Exact reduction from a sufficient-decision subspace

The characterization in Theorem 1 identifies which cost measurements are sufficient for recovering optimal decisions. We now use the same geometry on the solver side. Once a subspace contains the decision-relevant directions, the original LP or IP can be solved exactly on a lower-dimensional affine slice anchored at a reachable optimizer. Proofs and further details of this section are deferred to Section B.1.

Fix a prior set $\mathcal{C}$ containing the possible cost vectors. We call an affine subspace $V \subseteq \mathbb{R}^d$ *exact-sketching* if it preserves the full optimizer set on the prior, namely, for every $c \in \mathcal{C}$,

$$\arg\min_{x\in\mathcal{X}\cap V} c^\top x = \arg\min_{x\in\mathcal{X}} c^\top x.$$

Proposition 2 (Exact LP and IP reduction from a sufficient-decision subspace)**.** *An affine subspace $V \subseteq \mathbb{R}^d$ is exact-sketching on $\mathcal{C}$ if and only if*

$$\dim(\mathcal{X}^\star(\mathcal{C})) \subseteq \dim(V) \quad \textit{and} \quad V \cap \mathcal{X}^\star(\mathcal{C}) \neq \emptyset.$$

Equivalently, V is exact-sketching if and only if $\mathcal{X}^\star(\mathcal{C}) \subseteq V$. To construct such a space, choose a reachable optimizer as anchor and include the differences between reachable optimizers.

We next show how an exact-sketching affine subspace yields a potentially lower-dimensional reformulation of Equation (1). Fix an anchor $x_0 \in \mathcal{X}$ and a matrix $U \in \mathbb{R}^{d\times r}$ with linearly independent columns. Let $T_{U,x_0}(z) := x_0 + Uz$, let $\mathcal{Z}_{U,x_0} := \{z \in \mathbb{R}^r : T_{U,x_0}(z) \in \mathcal{X}\}$ be the compressed feasible region[1], and let $F_{U,x_0}(c) := \arg\min_{z\in\mathcal{Z}_{U,x_0}} (U^\top c)^\top z$ be the compressed optimizer set. If $x_0 \in \mathcal{X}^\star(\mathcal{C})$ and $\mathrm{range}(U) \supseteq \mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$, then Proposition 2 applies to the affine space $x_0 + \mathrm{range}(U)$, and the following optimizer-set identity holds for every $c \in \mathcal{C}$:

$$\mathcal{X}^\star(c) = T_{U,x_0}\big(F_{U,x_0}(c)\big). \qquad (\mathsf{Exact}(U, x_0, c))$$

Moreover,

$$\min_{x\in\mathcal{X}} c^\top x = c^\top x_0 + \min_{z\in\mathcal{Z}_{U,x_0}} (U^\top c)^\top z. \qquad (4)$$

Thus solving the reduced LP in (4) and lifting by T_{U,x_0} recovers an optimal solution of the original LP. For IPs, the same identity holds with $\mathcal{X}_I$ and $\mathcal{X}_I^\star(c)$ in place of $\mathcal{X}$ and $\mathcal{X}^\star(c)$, an anchor $x_0 \in \mathcal{X}_I^\star(\mathcal{C})$, and a subspace containing $\mathrm{dir}(\mathcal{X}_I^\star(\mathcal{C}))$. The corresponding exactness characterization is given in Proposition 15 (Appendix C.1). The compressed formulation retains the integrality constraint $x_0 + Uz \in \mathbb{Z}^d$. To guarantee that $\mathsf{Exact}(U, x_0, c)$ holds for every $c \in \mathcal{C}$, the minimal choice $\mathrm{range}(U) = \mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$ gives an exact $d^\star$-variable affine reformulation. Constructing this minimal subspace, however, can be both NP-hard and coNP-hard [Ye et al., 2026], even though it can be done in the LP case using the MIP formulation in Bennouna et al. [2026]. Our approach aims to find U such that $\mathsf{Exact}(U, x_0, c)$ holds with high probability for an independent cost drawn from the same distribution as the training samples. This allows us to design a tractable algorithm to do so.

The next result shows that vertex optimizers are preserved under $\mathsf{Exact}(U, x_0, c)$. This gives integral optimal solutions when the original LP has integral vertices. For general IPs, exactness instead follows from preserving the integer optimizer set while retaining the integrality constraint, as shown in Proposition 15 (Appendix C.1).

Corollary 3 (Vertex optimizers are preserved)**.** *If* $\mathsf{Exact}(U, x_0, c)$ *holds, then*

$$T_{U,x_0}\big(F_{U,x_0}(c) \cap \mathcal{Z}_{U,x_0}^{\angle}\big) = \mathcal{X}^\star(c) \cap \mathcal{X}^{\angle}.$$

Thus, under the exactness event $\mathsf{Exact}(U, x_0, c)$, any vertex optimizer of the reduced LP lifts to a vertex optimizer of the original LP. In particular, when the original LP has integral vertices, as in shortest-path formulations with totally unimodular constraints and integral right-hand sides, solving the reduced LP to a vertex optimum recovers an integral optimal solution, and hence IPs that can be relaxed into LPs will not lose their tractability when compressed. For general IPs, the reduced formulation retains the integrality requirement $x_0 + Uz \in \mathbb{Z}^d$, so any reduced optimizer likewise lifts to an integral optimal solution of the original problem.

Example 4 illustrates the resulting exact reduction.

Example 4 (Exact compression onto a reachable optimal edge)**.** *Let* $\mathcal{X} = [-1, 1]^2$ *and* $\mathcal{C}_\rho = (-1-\rho, -1) \times (-1, 1)$ *for* $\rho > 0$*. Every* $c \in \mathcal{C}_\rho$ *has* $c_1 < 0$*, while costs with* $c_2 = 0$ *expose the entire right edge. Thus* $\mathcal{X}^\star(\mathcal{C}_\rho) = \{1\} \times [-1, 1]$ *and* $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}_\rho)) = \mathrm{span}\{(0, 1)\}$*. Taking* $U = (0, 1)^\top$ *yields the exact one-variable LP* $\min_{z\in[-1,1]} c_2 z$*, with the original optimal value recovered by adding* c_1*. The map* $x = (1, z)$ *recovers the full optimizer set, including the entire edge when* $c_2 = 0$*. Figure 1 illustrates this reduction.*

3.2 Learning exact compression from samples

We now describe the sample-based learning algorithm (Algorithm 1).[2] It maintains a matrix U of decision-relevant directions discovered so far. Together with a fixed anchor $x_0 \in \mathcal{X}^\star(\mathcal{C})$, obtained by picking any $c_0 \in \mathcal{C}$ and taking $x_0 \in \mathcal{X}^\star(c_0)$, it defines the candidate affine slice $x_0 + \mathrm{range}(U)$. Training costs $c_1, \ldots, c_n$ are processed one at a time, and U grows whenever a new sample reveals a

[1] In the polyhedral model $\mathcal{X} = \{x \in \mathbb{R}^d : Ax \le b\}$, $\mathcal{Z}_{U,x_0} = \{z \in \mathbb{R}^r : A(x_0 + Uz) \le b\}$.

[2] The algorithm and tractability statement below are for LPs. The IP counterpart is given in Algorithm 2 and Theorem 16 (Appendix C.2).

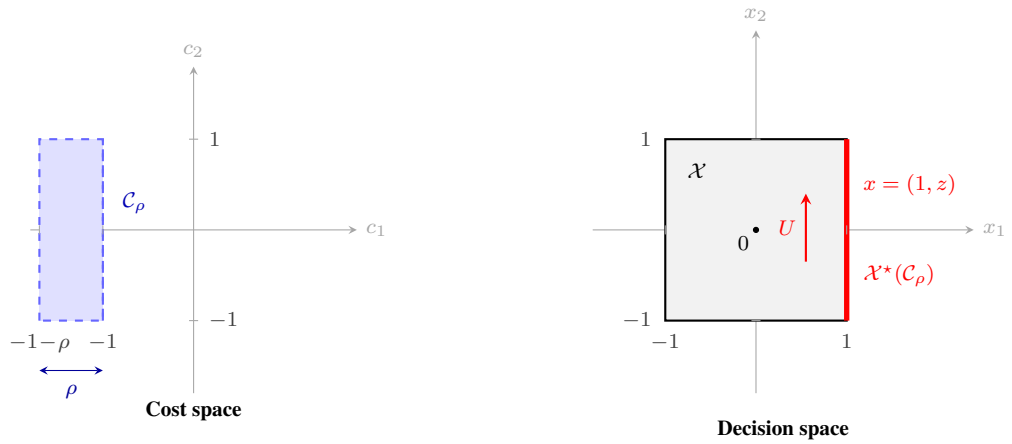


Figure 1: Exact compression through the decision-relevant subspace. Costs in $\mathcal{C}_\rho$ have $c_1 < 0$, so all reachable optimizers lie on the highlighted right edge. The direction $U = (0, 1)^\top$ parameterizes this edge as $x = (1, z)$, reducing the LP to one variable while preserving its full optimizer set.

Algorithm 1 Learning exact LP compression

Require: Feasible polytope $\mathcal{X}$, fixed reachable anchor $x_0 \in \mathcal{X}^\star(\mathcal{C})$, training samples $c_1, \ldots, c_n$, and deterministic tie-breaking for all choices.
Ensure: A compression matrix U_n and a hard-sample index set T_n.

```
Initialize U_0 ← [ ] ∈ R^{d×0} and T_n ← ∅.
for i = 1, ..., n do
    Set U_i ← U_{i−1}.
    while X*(c_i) ⊈ x_0 + range(U_i) do
        Choose x ∈ X*(c_i) ∩ X^∠ with x − x_0 ∉ range(U_i).
        U_i ← [U_i  x − x_0] and T_n ← T_n ∪ {i}.
    end while
end for
return U_n and T_n.
```

direction the current slice misses. For each c_i, the algorithm checks whether the optimal face $\mathcal{X}^\star(c_i)$ is contained in the current slice $x_0 + \mathrm{range}(U)$. If yes, it proceeds to c_{i+1}. If no, then since $\mathcal{X}^\star(c_i)$ is the convex hull of its vertices, some vertex x lies outside the slice. The algorithm appends $x - x_0$ as a new column of U and rechecks. This test-and-append loop repeats until the full optimal face is contained. Samples that trigger at least one append are called *hard instances*.

Let P_c denote the cost distribution supported on $\mathcal{C}$. In the following guarantee, the confidence $1 - \delta$ is over the i.i.d. training sample. Conditional on the learned U_n, the probability in (5) is over an independent test cost $c \sim P_c$.

Theorem 5 (Exact-compression certificate and tractability)**.** *Fix an anchor $x_0 \in \mathcal{X}^\star(\mathcal{C})$ independently of the n training costs. Run Algorithm 1 on n i.i.d. costs drawn from a distribution supported on $\mathcal{C}$. Then*

$$\mathrm{range}(U_n) \subseteq \mathrm{dir}(\mathcal{X}^\star(\mathcal{C})), \qquad |T_n| \le \mathrm{rank}(U_n) \le d^\star,$$

and $\mathsf{Exact}(U_n, x_0, c_i)$ *holds for every training sample $i = 1, \ldots, n$. Moreover, for any $\delta \in (0, 1)$, with probability at least $1 - \delta$,*

$$\mathbb{P}_{c \sim P_c}[\mathsf{Exact}(U_n, x_0, c)] \ge 1 - \frac{4}{n}\big(6|T_n| + \log(e/\delta)\big) \ge 1 - \frac{4}{n}\big(6d^\star + \log(e/\delta)\big). \tag{5}$$

Implementation and parallelism. When U has rank r, each containment test uses one LP to find the optimal value and at most $2(d - r)$ auxiliary LPs over the optimal face. A failed test supplies a new extreme optimizer (Lemma 9). With at most $n + d^\star$ tests, the algorithm runs in polynomial time in the LP-oracle model. If optimizers are unique almost surely, as for an absolutely continuous cost distribution, we can instead solve each training LP once and compute a basis of

$\operatorname{span}\{x_i - x_0 : i = 1, \ldots, n\}$ using the same independent anchor x_0. Beyond the anchor, this requires only n independent LP solves, all parallelizable. By comparison, the gradient-based methods of Sakaue and Oki [2024], Iwata and Sakaue [2025] require En projected-LP solves over E training passes, in addition to any initialization or normalization solves. Although computations within a batch can be parallelized, successive parameter updates create dependencies between batches. Our construction avoids this iterative training, requiring only the independent solves and a basis computation.

Sample-free construction and the computational tradeoff. The sample-based learner is complementary to the sample-free alternative. When $\mathcal{C}$ is known, Bennouna et al. [2025] construct a global SDD directly from the pair $(\mathcal{X}, \mathcal{C})$ using an iterative construction with a mixed-integer implementation, requiring no historical cost samples. This construction can be computationally expensive. Ye et al. [2026] show that, for open convex $\mathcal{C}$, constructing a minimum-size global SDD and even computing its minimum size are NP-hard and coNP-hard. Our sample-based learner trades this combinatorial hardness for a statistical cost. Each update requires only LP or IP solves (depending on whether the problem at hand is an LP or IP), optimal-face containment tests, and rank tests, but the algorithm needs representative cost samples to discover the relevant directions. Once the sampled directions span $\operatorname{dir}(\mathcal{X}^\star(\mathcal{C}))$, $\mathsf{Exact}(U_n, x_0, c)$ holds for every $c \in \mathcal{C}$. This follows from Proposition 2 for LPs and Proposition 15 (Appendix C.1) for IPs. Before this occurs, Theorem 5 gives the LP failure bound, and the IP counterpart is Theorem 16 (Appendix C.2).

3.3 Ruling out instances that are too hard

Algorithm 1 does not require knowledge of $\mathcal{C}$. Seeking an exact reformulation of (1) for c ranging over the *entire* support of its distribution can be too conservative. A natural way to address this is to rule out training instances that are *too hard* to accommodate in a small reformulation, because the extra directions they require are needed only on a low-probability part of the cost distribution. This also allows us to potentially trade optimality for LP or IP compression by only learning the most important decision-relevant directions. To do so, we construct a convex estimated prior $\widehat{\mathcal{C}}$ from the data, and only learn the relevant decision directions for the cost vector samples that are inside this set, which we define in the following.

Definition 6 ((ρ, δ_0)-estimated prior)**.** *Let P_c be the cost distribution. A pilot-measurable random convex set $\widehat{\mathcal{C}} \subseteq \mathbb{R}^d$ is a (ρ, δ_0)-estimated prior if*

$$\mathbb{P}_{\text{pilot}}\left[P_c(c \notin \widehat{\mathcal{C}}) \leq \rho\right] \geq 1 - \delta_0.$$

We next state the learning guarantee for this new framework. Let $\widehat{\mathcal{C}}$ be a convex (ρ, δ_0)-estimated prior[3], and define $\mathcal{E}_0 := \{P_c(c \notin \widehat{\mathcal{C}}) \leq \rho\}$. On $\mathcal{E}_0$, set $P_{\widehat{\mathcal{C}}} := P_c(\cdot \mid c \in \widehat{\mathcal{C}})$ and $\widehat{d}_\star := \dim \operatorname{dir}(\mathcal{X}^\star(\widehat{\mathcal{C}}))$. Choose an anchor $\widehat{x}_0 \in \mathcal{X}^\star(\widehat{\mathcal{C}})$ by selecting any $\widehat{c}_0 \in \widehat{\mathcal{C}}$ and solving the original LP. Draw $n \geq 1$ i.i.d. training costs independently of the pilot and retain the $\hat{n}$ that fall inside $\widehat{\mathcal{C}}$.[4] Run Algorithm 1 with anchor $\widehat{x}_0$ on these retained costs in their original order, and denote its output by $U_{\hat{n}}$.

Theorem 7 (Learning with a calibrated estimated prior)**.** *Under the preceding setup, for any $\delta_1 \in (0, 1)$, with probability at least $1 - \delta_0 - \delta_1$ over the pilot and the n training costs, the following bound holds whenever $\hat{n} \geq 1$:*

$$\mathbb{P}_{c \sim P_c}[\mathsf{Exact}(U_{\hat{n}}, \widehat{x}_0, c)] \geq 1 - \rho - \frac{4}{\hat{n}}\big(6\widehat{d}_\star + \log(e/\delta_1)\big). \tag{6}$$

Consequently, taking $\widehat{\mathcal{C}} = \widehat{\mathcal{C}}_{\rho,\delta_0}$ from Proposition 13 gives the same distribution-free unknown-prior certificate.

The first term, ρ, bounds the probability mass left outside the calibrated estimated prior, while the second term bounds the stable-compression risk for costs drawn from the conditional distribution

[3]The distribution-free construction for such a prior and its proof are deferred to Section B.3.1.

[4]When the prior's coverage is close to $1 - \rho$, $\hat{n} \approx (1 - \rho)n$. Appendix B.3.2 gives the precise relation and a concentration bound.

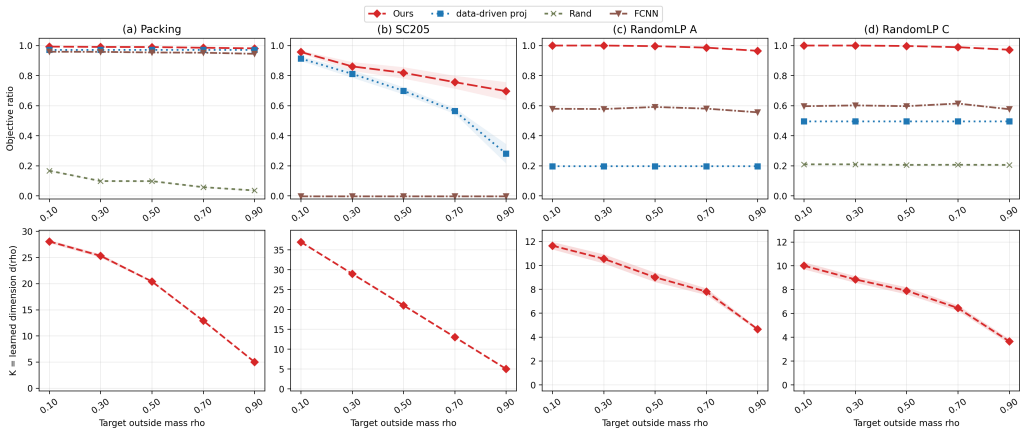


Figure 2: Comparison across target outside-mass levels. Rand corresponds to random projections.

inside the realized prior. The parameter ρ controls the degree to which low-probability instances may be discarded. A larger ρ permits a smaller prior and can reduce its decision-relevant dimension $\widehat{d}_\star$, which would result in a more compressed LP reformulation. This lets us calibrate ρ to prioritize optimality (small ρ) or compression (large ρ). In some cases, a small increase in ρ can substantially reduce the number of variables in the reduced LP. We examine this tradeoff in Section 4.

4 Numerical Experiments

We evaluate the sample-based exact-compression framework in the unknown-prior setting of Section 3.3.[5] The learner first builds a convex working prior from historical cost samples and then learns the decision-relevant directions inside that set. The main text reports the two diagnostics that directly test this mechanism: how performance changes as the target outside-mass level ρ is varied, and how our method compares with a neural projection baseline as the number of training costs changes. We also compare the time needed by our method and the baselines to learn a compression and to solve LPs using the learned compression. Additional experiments and details are reported in Section D.

Experimental protocol. We compare our exact-compression method, denoted OURS, with two projection-learning baselines. The first is DATADRIVENPROJ, a data-driven projection method motivated by Sakaue and Oki [2024]: it learns a shared K-dimensional projection from training LP instances and then solves the projected LP at test time. The second is FCNN, a cost-only fully connected neural network baseline following the neural projection paradigm of Iwata and Sakaue [2025]: it maps the observed cost vector to an instance-specific projection and solves the resulting reduced LP. We also include RAND, a random projection baseline that selects K variables uniformly at random in the preprocessed LP and fixes the others to zero [Akchen and Mišić, 2024].

In the ρ-sweep, we construct an estimated prior of the form $\widehat{\mathcal{C}} = \{c : s_{\widehat{\theta}}(c) \leq S_{(k)}\}$, where $s_{\widehat{\theta}}$ is the ridge Mahalanobis score from Section B.3.1 and $S_{(k)}$ is the calibration order statistic associated with the target outside-mass level ρ. The radius is therefore calibrated from empirical scores. After forming $\widehat{\mathcal{C}}$, we run Algorithm 1 on the retained training costs to obtain an affine exact-compression matrix of learned dimension $d(\rho)$. We use the synthetic repeated-LP families and selected Netlib instances used throughout the paper[6] as a benchmark. Accuracy is measured by the average test objective ratio, i.e., the objective value returned by a method divided by the full-LP objective value on the same test instance. Each plotted setting uses twelve held-out test LPs. Shaded regions indicate uncertainty around the plotted means.

[5] Code and results for the LP and IP experiments are available at `https://github.com/yeyuhanyyh/Solving-Linear-and-Integer-Programs-Faster-Using-Data`.

[6] Details of the LP instances and cost distributions are deferred to Appendix D.

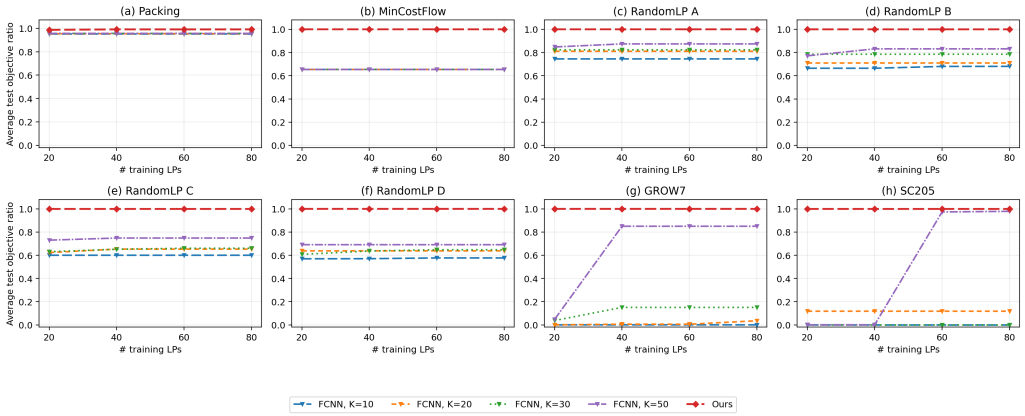


Figure 3: Comparison against the FCNN baseline.

Compression–coverage tradeoff through ρ. Figure 2 varies the target outside-mass level ρ. For each ρ, our approach constructs a different calibrated set $\widehat{\mathcal{C}}_\rho$, learns a compression of dimension $d(\rho)$, and solves the corresponding affine reduced LP. The projection baselines are then given the same dimensional budget $K = d(\rho)$, so all methods use the same number of reduced LP variables.

This experiment illustrates the main message of Section 3.3. Increasing ρ intentionally discards more low-probability cost mass, which can lower $d(\rho)$ and hence produce a smaller reduced LP. The empirical curves show that this compression can be obtained with little loss on many test instances, while the projection baselines often fail to match the exact method even at the same dimension. In other words, the estimated prior does not merely choose a smaller K. It focuses learning on optimizer differences within the estimated prior.

Comparison against neural projections. Figure 3 compares our method with the FCNN baseline as the number of training costs varies. For FCNN, each dashed curve reports the objective ratio using the displayed number of training samples and one of the reduced dimensions $K \in \{10, 20, 30, 50\}$. For this comparison, OURS uses the same training costs to fit the prior and learn the compression, with the largest training score as the threshold (Appendix D.3). These results indicate that our algorithm works substantially better than the compared baseline.

			Learning (s)			LP solve (ms)			
				DATADRIVEN				DATADRIVEN	
Instance	d	K	**OURS**	PROJ	FCNN	Full	**OURS**	PROJ	FCNN
Packing	360	21	**0.104±0.006**	3.314±0.106	37.998±0.247	3.381±0.112	**2.720±0.067**	9.192±1.278	5.755±0.347
RandomLP A	140	9	**0.246±0.001**	0.913±0.039	2.864±0.010	10.790±0.685	**1.944±0.031**	5.227±0.303	5.655±1.693
RandomLP C	220	10	**0.690±0.001**	1.955±0.250	4.929±0.077	30.216±0.082	**2.893±0.062**	11.809±1.428	9.286±2.038
SC205	203	21	**0.110±0.004**	6.632±0.095	18.635±0.243	4.273±0.527	**4.251±0.219**	4.740±0.116	2.336±0.237

Table 1: Timing on the Figure 2 instances at $\rho = 0.5$, using $n = 40$ training costs before prior filtering. Here d and K count original and reduced LP variables, respectively. All compressed methods use the same K. Entries are means $\pm$ sample standard deviations over three timing repeats.

Learning and LP solving time. Tables 1 and 2 report learning and LP solving times for the instances in Figures 2 and 3, respectively. We report learning time separately from the solve time per test LP. As discussed in Section 3.2, a major advantage of our method is the low computational cost of learning the compression. The stochastic gradient implementation of DATADRIVENPROJ makes one pass through the training set. Each update solves one projected LP and K quadratic programs (QPs) for the projection step. FCNN makes multiple passes through the training set, solving one projected LP for each sample in each pass. Our method avoids these QPs and iterative neural network training.

			Learning (s)		LP solve (ms)		
Instance	d	K	**OURS**	FCNN	Full	**OURS**	FCNN
Packing	360	$(19, 19, 19)$	$\mathbf{0.084 \pm 0.005}$	40.066 ± 1.766	7.208 ± 0.387	$\mathbf{3.204 \pm 0.136}$	11.249 ± 0.931
MinCostFlow	360	$(3, 1, 1)$	$\mathbf{0.189 \pm 0.017}$	25.973 ± 5.973	12.616 ± 0.497	$\mathbf{1.890 \pm 0.351}$	3.927 ± 1.025
RandomLP A	140	$(5, 11, 14)$	$\mathbf{0.276 \pm 0.016}$	25.985 ± 5.165	18.761 ± 2.068	$\mathbf{2.789 \pm 0.478}$	6.933 ± 0.997
RandomLP B	180	$(10, 7, 8)$	$\mathbf{0.459 \pm 0.021}$	29.022 ± 3.529	31.295 ± 2.158	$\mathbf{2.774 \pm 0.465}$	6.703 ± 1.666
RandomLP C	220	$(5, 7, 3)$	$\mathbf{0.810 \pm 0.050}$	28.304 ± 3.920	48.423 ± 4.036	$\mathbf{2.765 \pm 0.297}$	5.462 ± 0.636
RandomLP D	260	$(7, 10, 5)$	$\mathbf{1.197 \pm 0.056}$	36.051 ± 5.238	67.593 ± 1.726	$\mathbf{3.433 \pm 0.605}$	7.719 ± 2.428
GROW7	301	$(19, 19, 19)$	$\mathbf{0.378 \pm 0.020}$	60.714 ± 0.634	25.622 ± 1.438	$\mathbf{6.870 \pm 0.431}$	16.727 ± 3.412
SC205	203	$(1, 1, 1)$	$\mathbf{0.114 \pm 0.007}$	14.193 ± 0.578	7.243 ± 0.369	$\mathbf{1.638 \pm 0.094}$	2.763 ± 0.297

Table 2: Timing on the Figure 3 instances with $n = 20$ training costs and 20 test LPs per seed. Entries are means $\pm$ sample standard deviations across three independent seeds. Each K triple gives the number of LP variables for these seeds. OURS and FCNN use the same K.

Specifically, for each problem and seed in Table 2, our method and FCNN use the same 20 training LPs. Our method solves each training LP once and computes a basis of the optimizer differences. In these runs, FCNN makes 120 passes through the training set, requiring $20 \times 120 = 2{,}400$ projected-LP solves. It also evaluates six separate validation LPs before training and after each pass, requiring $6 \times 121 = 726$ projected-LP solves. In these experiments, our approach is $30.1\times$–$477.0\times$ faster to learn than FCNN. Its reduced LP is faster than both the full LP and FCNN on all eight matched-K instances.

For the highly sparse Packing instance in Table 2, matching the reduced dimension K across methods does not imply equally sparse reduced constraint matrices. The original constraint matrix has 720 nonzeros (0.48% density), our compressed constraint matrix has 2,090 nonzeros (26.19% density), and the FCNN compressed constraint matrix is essentially dense, with approximately 7,980 nonzeros (approximately 100% density). Thus, at the same reduced dimension, our representation retains substantially more sparsity than FCNN. Together with dimension reduction, this sparsity advantage helps explain our method's shorter solve time. A smaller reduced dimension alone need not yield a faster LP solve.

For IPs, we also evaluate our method on unit commitment and knapsack problems. Our method achieves higher recovery rates and objective ratios than the other compression methods. It also learns much faster than the baselines [Xavier et al., 2021, Bertsimas and Stellato, 2022, Nair et al., 2020, Li et al., 2025] (Appendix C.3, Tables 3–5).

5 Conclusion

In this paper, we showed how to compress LPs and IPs that are solved repeatedly with varying costs. We characterized the smallest affine reformulation that preserves the full optimizer set over a given cost region and provided sample-based algorithms with fast-rate guarantees, without requiring nondegeneracy. When the cost region is unknown, we estimate it from historical samples, allowing a tradeoff between compression and the probability of optimality. When optimizers are unique almost surely, each training problem only needs to be solved once, and these solves can run in parallel. Our experiments show substantial improvements over the compared baselines in accuracy, sample efficiency, and computational speed. Future work includes improving the sparsity of the reduced formulations and extending the approach to problems whose feasible regions also change, as well as more general optimization problems.

Acknowledgments and Disclosure of Funding

Yuhan Ye is supported by the MIT Schwarzman College of Computing PhD Fellowship, and Omar Bennouna is supported by the MathWorks Fellowship.

References

Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin, and M. R. Reddy. Applications of network optimization. In *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 1–83. Elsevier, 1995. doi: 10.1016/S0927-0507(05)80118-5.

Yi-Chun Akchen and Velibor V. Mišić. Column-randomized linear programs: Performance guarantees and applications. *Operations Research*, 73(3):1366–1383, 2024. doi: 10.1287/opre.2020.0494. URL `https://doi.org/10.1287/opre.2020.0494`.

Alessandro Alessio and Alberto Bemporad. A survey on explicit model predictive control. In *Nonlinear Model Predictive Control: Towards New Challenging Applications*, volume 384 of *Lecture Notes in Control and Information Sciences*, pages 345–369. Springer, Berlin, 2009.

Brandon Amos. Tutorial on amortized optimization. *Foundations and Trends in Machine Learning*, 16(5):592–732, 2023.

Edison Avraham and Tal Raviv. The data-driven time-dependent traveling salesperson problem. *Transportation Research Part B: Methodological*, 134:25–40, 2020. doi: 10.1016/j.trb.2020.01.005.

Maria-Florina Balcan. Data-driven algorithm design. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 626–645. Cambridge University Press, 2021.

Maria-Florina Balcan, Vaishnavh Nagarajan, Ellen Vitercik, and Colin White. Learning-theoretic foundations of algorithm configuration for combinatorial partitioning problems. In *Proceedings of the 30th Conference on Learning Theory*, volume 65 of *Proceedings of Machine Learning Research*, pages 213–274. PMLR, 2017.

Maria-Florina Balcan, Dan DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. How much data is sufficient to learn high-performing algorithms? Generalization guarantees for data-driven algorithm design. *Journal of the ACM*, 71(5):32:1–32:58, 2024.

Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N. Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

Omar Bennouna, Amine Bennouna, Saurabh Amin, and Asuman E. Ozdaglar. What data enables optimal decisions? an exact characterization for linear optimization. In *Advances in Neural Information Processing Systems*, volume 38, 2025. NeurIPS 2025.

Omar Bennouna, Amine Bennouna, Saurabh Amin, and Asuman Ozdaglar. Data informativeness in linear optimization under uncertainty. *arXiv preprint arXiv:2602.15365*, 2026. URL `https://arxiv.org/abs/2602.15365`.

Dimitris Bertsimas and Bartolomeo Stellato. The voice of optimization. *Machine Learning*, 110(2): 249–277, 2021.

Dimitris Bertsimas and Bartolomeo Stellato. Online mixed-integer optimization in milliseconds. *INFORMS Journal on Computing*, 34(4):2229–2248, 2022. doi: 10.1287/ijoc.2022.1181. URL `https://doi.org/10.1287/ijoc.2022.1181`.

Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, Cambridge, 2017.

Olivier Bousquet, Steve Hanneke, Shay Moran, and Nikita Zhivotovskiy. Proper learning, Helly number, and an optimal SVM bound. In *Proceedings of the 33rd Conference on Learning Theory*, volume 125 of *Proceedings of Machine Learning Research*, pages 582–609. PMLR, 2020.

Marco C. Campi and Simone Garatti. Compression, generalization and learning. *Journal of Machine Learning Research*, 24(339):1–74, 2023.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *Journal of Machine Learning Research*, 23(189):1–59, 2022.

Claudia d'Ambrosio, Leo Liberti, Pierre-Louis Poirion, and Ky Vu. Random projections for quadratic programs. *Mathematical Programming*, 183(1–2):619–647, 2020.

Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning in road networks. *Transportation Science*, 51(2):566–591, 2017. doi: 10.1287/trsc.2014.0579.

Anthony V. Fiacco. *Introduction to Sensitivity and Stability Analysis in Nonlinear Programming*, volume 165 of *Mathematics in Science and Engineering*. Academic Press, New York, 1983.

Sally Floyd and Manfred K. Warmuth. Sample compression, learnability, and the vapnik–chervonenkis dimension. *Machine Learning*, 21(3):269–304, 1995.

Antonio Frangioni and Antonio Manca. A computational study of cost reoptimization for min-cost flow problems. *INFORMS Journal on Computing*, 18(1):61–70, 2006. doi: 10.1287/ijoc.1040.0081.

Thore Graepel, Ralf Herbrich, and John Shawe-Taylor. PAC-bayesian compression bounds on the prediction error of learning algorithms for classification. *Machine Learning*, 59(1–2):55–76, 2005.

Martin Grohe, Kristian Kersting, Martin Mladenov, and Erkal Selman. Dimension reduction via colour refinement. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms – ESA 2014*, volume 8737 of *Lecture Notes in Computer Science*, pages 505–516. Springer, 2014. doi: 10.1007/978-3-662-44777-2_42. URL https://doi.org/10.1007/978-3-662-44777-2_42.

Rishi Gupta and Tim Roughgarden. A PAC approach to application-specific algorithm selection. *SIAM Journal on Computing*, 46(3):992–1017, 2017.

Rishi Gupta and Tim Roughgarden. Data-driven algorithm design. *Communications of the ACM*, 63 (6):87–94, 2020.

Steve Hanneke and Aryeh Kontorovich. A sharp lower bound for agnostic learning with sample compression schemes. In *Proceedings of the 30th International Conference on Algorithmic Learning Theory*, volume 98 of *Proceedings of Machine Learning Research*, pages 489–505. PMLR, 2019.

Steve Hanneke and Aryeh Kontorovich. Stable sample compression schemes: New applications and an optimal SVM margin bound. In Vitaly Feldman, Katrina Ligett, and Sivan Sabato, editors, *Proceedings of the 32nd International Conference on Algorithmic Learning Theory*, volume 132 of *Proceedings of Machine Learning Research*, pages 697–721. PMLR, 2021.

Tomoharu Iwata and Shinsaku Sakaue. Learning to generate projections for reducing dimensionality of heterogeneous linear programming problems. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems*. Springer, Berlin, Heidelberg, 2004. doi: 10.1007/978-3-540-24777-7. URL https://doi.org/10.1007/978-3-540-24777-7.

Olivier Ledoit and Michael Wolf. A well-conditioned estimator for large-dimensional covariance matrices. *Journal of Multivariate Analysis*, 88(2):365–411, 2004.

Jing Lei, Max G'Sell, Alessandro Rinaldo, Ryan J. Tibshirani, and Larry Wasserman. Distribution-free predictive inference for regression. *Journal of the American Statistical Association*, 113(523): 1094–1111, 2018.

Jiajun Li, Yixuan Li, Ran Hou, Yu Ding, Shisi Guan, Jiahui Duan, Xiongwei Han, Tao Zhong, Vincent Chau, Weiwei Wu, Zhiyuan Liu, and Wanyuan Wang. Constraint matters: Multi-modal representation for reducing mixed-integer linear programming. In *International Conference on Learning Representations*, 2026. URL https://arxiv.org/abs/2508.18742.

Yixuan Li, Can Chen, Jiajun Li, Jiahui Duan, Xiongwei Han, Tao Zhong, Vincent Chau, Weiwei Wu, and Wanyuan Wang. Fast and interpretable mixed-integer linear program solving by learning model reduction. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(25):27090–27098, 2025. doi: 10.1609/aaai.v39i25.34916.

Nick Littlestone and Manfred K. Warmuth. Relating data compression and learnability. Technical report, University of California, Santa Cruz, 1986.

Prasanta Chandra Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences of India*, 2(1):49–55, 1936.

Jayanta Mandi, Emir Demirović, Peter J. Stuckey, and Tias Guns. Smart predict-and-optimize for hard combinatorial optimization problems. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(2):1603–1610, 2020. doi: 10.1609/aaai.v34i02.5521.

Sidhant Misra, Line Roald, and Yeesian Ng. Learning for constrained optimization: Identifying optimal active constraint sets. *INFORMS Journal on Computing*, 34(1):463–480, 2022.

Shay Moran and Amir Yehudayoff. Sample compression schemes for VC classes. *Journal of the ACM*, 63(3):21:1–21:10, 2016.

Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, Ravichandra Addanki, Tharindi Hapuarachchi, Thomas Keck, James Keeling, Pushmeet Kohli, Ira Ktena, Yujia Li, Oriol Vinyals, and Yori Zwols. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020. URL `https://arxiv.org/abs/2012.13349`.

Anh Tuan Nguyen and Viet Anh Nguyen. Provably data-driven projection method for quadratic programming. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(29):24541–24548, 2026.

Ulrich Pferschy and Joachim Schauer. The knapsack problem with conflict graphs. *Journal of Graph Algorithms and Applications*, 13(2):233–249, 2009. doi: 10.7155/jgaa.00186. URL `https://doi.org/10.7155/jgaa.00186`.

David Pisinger. Where are the hard knapsack problems? *Computers & Operations Research*, 32(9): 2271–2284, 2005. doi: 10.1016/j.cor.2004.03.002.

Pierre-Louis Poirion, Bruno F. Lourenço, and Akiko Takeda. Random projections of linear and semidefinite problems with linear inequalities. *Linear Algebra and its Applications*, 664:24–60, 2023.

Shinsaku Sakaue and Taihei Oki. Generalization bound and learning methods for data-driven projections in linear programming. In *Advances in Neural Information Processing Systems*, 2024.

Rajiv Sambharya, Georgina Hall, Brandon Amos, and Bartolomeo Stellato. Learning to warm-start fixed-point optimization algorithms. *Journal of Machine Learning Research*, 25(166):1–46, 2024.

Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Vladimir Vovk. Conditional validity of inductive conformal predictors. In *Proceedings of the Asian Conference on Machine Learning*, volume 25 of *Proceedings of Machine Learning Research*, pages 475–490. PMLR, 2012. URL `https://proceedings.mlr.press/v25/vovk12.html`.

Vladimir Vovk, Alex Gammerman, and Glenn Shafer. *Algorithmic Learning in a Random World*. Springer, 2005.

Ky Vu, Pierre-Louis Poirion, and Leo Liberti. Random projections for linear programming. *Mathematics of Operations Research*, 43(4):1051–1071, 2018.

Samuel S. Wilks. Determination of sample sizes for setting tolerance limits. *The Annals of Mathematical Statistics*, 12(1):91–96, 1941.

Álinson S. Xavier, Feng Qiu, and Shabbir Ahmed. Learning to solve large-scale security-constrained unit commitment problems. *INFORMS Journal on Computing*, 33(2):739–756, 2021. doi: 10.1287/ijoc.2020.0976. URL https://doi.org/10.1287/ijoc.2020.0976.

Alinson S. Xavier, Aleksandr M. Kazachkov, Ogün Yurdakul, Jun He, and Feng Qiu. UnitCommitment.jl: A Julia/JuMP optimization package for security-constrained unit commitment (version 0.4), 2024. URL https://doi.org/10.5281/zenodo.4269874.

Yuhan Ye, Saurabh Amin, and Asuman Özdağlar. Learning decision-sufficient representations for linear optimization. In Steve Hanneke and Tor Lattimore, editors, *Proceedings of Thirty Ninth Conference on Learning Theory*, volume 336 of *Proceedings of Machine Learning Research*, pages 6924–6975. PMLR, 2026. URL https://proceedings.mlr.press/v336/ye26a.html.

A Additional Related Work

Parametric and multiparametric optimization. Repeated optimization with varying parameters has long been studied through parametric programming and sensitivity analysis [Fiacco, 1983]. In control and explicit model predictive control, a classical strategy is to precompute the full piecewise-affine solution map offline through multiparametric programming [Bemporad et al., 2002, Alessio and Bemporad, 2009, Borrelli et al., 2017]. These methods target an explicit representation of the optimizer over a parameter region. Our goal is different. We do not attempt to compute the entire solution map, but rather to identify the decision-relevant subspace spanned by differences between reachable optimizers and to use it for exact low-dimensional reformulation.

Structural LP reductions. Grohe et al. [2014] use colour refinement to reduce the numbers of variables and constraints while preserving the optimal value. We study a common affine restriction that preserves the full optimizer set over a region of cost vectors.

Learning to accelerate optimization. A broader recent literature uses data to accelerate future solves without explicitly learning a decision-sufficient subspace. Representative directions include learning warm starts for iterative solvers [Sambharya et al., 2024], predicting active sets or optimization strategies [Bertsimas and Stellato, 2021, Misra et al., 2022], and amortized optimization methods that map instances directly to approximate solutions [Amos, 2023, Chen et al., 2022]. These approaches can substantially reduce computation, but they typically do not identify a task-dependent exact low-dimensional LP together with a sufficiency guarantee.

Data-driven algorithm design and generalization guarantees. Gupta and Roughgarden [2017], Gupta and Roughgarden [2020], Balcan et al. [2017], Balcan [2021], and Balcan et al. [2024] formalize the broader viewpoint that one can learn algorithmic components from representative instances and still obtain out-of-sample guarantees. The LP-projection works above fit this template, with sample complexity bounds obtained via uniform-convergence arguments based on pseudo-dimension. Our framework departs from this approach in two ways. First, the object being learned is different. Instead of a generic projection matrix, we learn a specific geometric object, the decision-relevant subspace, and our algorithm grows this subspace one direction at a time, only when a training sample exposes a direction the current subspace misses. Second, the proof technique is different. This incremental structure makes our learner a *stable compression scheme* in the sense of Hanneke and Kontorovich [2021], which yields a fast $\widetilde{O}(d^\star/n)$ bound on exactness failure. The uniform-convergence bounds for projection methods instead concern objective-value generalization gaps and scale as $\widetilde{O}(1/\sqrt{n})$ [Sakaue and Oki, 2024, Iwata and Sakaue, 2025]. In our unknown-prior guarantee, we separate the samples spent on calibrating the working prior from those spent on discovering decision-relevant directions.

B Deferred proofs

B.1 Proofs for Subsection 3.1

Proof of Proposition 2. Suppose first that V is exact-sketching on $\mathcal{C}$. Then every reachable optimizer belongs to V: if $v \in \mathcal{X}^\star(\mathcal{C})$, then for some $c \in \mathcal{C}$ we have $v \in \arg\min_{x \in \mathcal{X}} c^\top x$, and exactness gives

$v \in \arg\min_{x \in \mathcal{X} \cap V} c^\top x \subseteq V$. Hence $V \cap \mathcal{X}^\star(\mathcal{C}) \neq \emptyset$. Moreover, for any $v, v' \in \mathcal{X}^\star(\mathcal{C})$, both points lie in V, so $v - v' \in \mathrm{dir}(V)$. Taking spans over all such pairs yields $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C})) \subseteq \mathrm{dir}(V)$.

Conversely, assume $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C})) \subseteq \mathrm{dir}(V)$ and choose $x_0 \in V \cap \mathcal{X}^\star(\mathcal{C})$. For every $v \in \mathcal{X}^\star(\mathcal{C})$, we have $v - x_0 \in \mathrm{dir}(\mathcal{X}^\star(\mathcal{C})) \subseteq \mathrm{dir}(V)$, so $v \in V$. Thus V contains all reachable optimizers. Fix $c \in \mathcal{C}$. The optimal face $\arg\min_{x \in \mathcal{X}} c^\top x$ is the convex hull of its extreme points, and these extreme points belong to $\mathcal{X}^\star(\mathcal{C})$. Therefore the whole optimal face is contained in V. Hence, the restricted and original LPs have the same optimizer set, and V is exact-sketching on $\mathcal{C}$. □

Justification of (4) *and* (Exact(U, x_0, c)). Let $V := x_0 + \mathrm{range}(U)$. Under the assumptions stated before (Exact(U, x_0, c)), Proposition 2 gives

$$\arg\min_{x \in \mathcal{X} \cap V} c^\top x = \mathcal{X}^\star(c), \qquad c \in \mathcal{C}.$$

The map T_{U,x_0} is an affine bijection from $\mathcal{Z}_{U,x_0}$ onto $\mathcal{X} \cap V$, and $c^\top T_{U,x_0}(z) = c^\top x_0 + (U^\top c)^\top z$. Therefore

$$T_{U,x_0}\big(F_{U,x_0}(c)\big) = \arg\min_{x \in \mathcal{X} \cap V} c^\top x = \mathcal{X}^\star(c),$$

which is (Exact(U, x_0, c)). The same parametrization gives

$$\min_{x \in \mathcal{X}} c^\top x = \min_{x \in \mathcal{X} \cap V} c^\top x = c^\top x_0 + \min_{z \in \mathcal{Z}_{U,x_0}} (U^\top c)^\top z,$$

which is (4). □

Write $V_{U,x_0} = x_0 + \mathrm{range}(U)$ for the affine slice.

Proof of Corollary 3. Let $F_Z := F_{U,x_0}(c)$ and $F_X := \mathcal{X}^\star(c)$. Exactness gives $T_{U,x_0}(F_Z) = F_X$. Since T_{U,x_0} is an affine bijection from $\mathcal{Z}_{U,x_0}$ onto $\mathcal{X} \cap V_{U,x_0}$, it is also an affine bijection from the face F_Z onto the face F_X. Hence it maps extreme points of F_Z exactly to extreme points of F_X:

$$T_{U,x_0}(F_Z^{\angle}) = F_X^{\angle}.$$

Because F_Z is a face of $\mathcal{Z}_{U,x_0}$, $F_Z^{\angle} = F_Z \cap \mathcal{Z}_{U,x_0}^{\angle}$. Because F_X is a face of $\mathcal{X}$, $F_X^{\angle} = F_X \cap \mathcal{X}^{\angle}$. Substituting these identities yields

$$T_{U,x_0}\big(F_{U,x_0}(c) \cap \mathcal{Z}_{U,x_0}^{\angle}\big) = \mathcal{X}^\star(c) \cap \mathcal{X}^{\angle}.$$

□

Verification of Example 4. For every $c \in \mathcal{C}_\rho$, the inequality $c_1 < 0$ implies that $c_1 x_1$ is uniquely minimized over $[-1, 1]$ at $x_1 = 1$. Separability therefore gives

$$\mathcal{X}^\star(c) = \{1\} \times \operatorname*{argmin}_{z \in [-1,1]} c_2 z.$$

The second-coordinate optimizer set is $\{-1\}$ if $c_2 > 0$, $\{1\}$ if $c_2 < 0$, and $[-1, 1]$ if $c_2 = 0$. Since $\mathcal{C}_\rho$ includes costs with $c_2 = 0$, the reachable optimizer set is the full right edge, and its decision-relevant subspace is $\mathrm{span}\{(0, 1)\}$. The substitution $x = (1, z)$ maps the reduced optimizer set exactly onto $\mathcal{X}^\star(c)$ and gives the optimal value $c_1 - |c_2|$. The reduction is minimal among affine restrictions preserving the full optimizer set throughout $\mathcal{C}_\rho$, because any such restriction must contain this nontrivial segment. □

B.2 Proofs for Subsection 3.2

We first provide a geometric characterization of Exact(U, x_0, c).

Lemma 8 (Characterization of Exact(U, x_0, c))**.** *Fix $x_0 \in \mathcal{X}$ and a full-column-rank matrix U, and set*

$$V_{U,x_0} := x_0 + \mathrm{range}(U).$$

For any $c \in \mathbb{R}^d$,

$$\text{Exact}(U, x_0, c) \quad \Longleftrightarrow \quad \mathcal{X}^\star(c) \subseteq V_{U,x_0}.$$

Consequently, if U^+ is obtained from U by appending columns and Exact(U, x_0, c) *holds, then* Exact(U^+, x_0, c) *holds.*

Proof. The map T_{U,x_0} is an affine bijection from $\mathcal{Z}_{U,x_0}$ onto $\mathcal{X} \cap V_{U,x_0}$. Hence

$$T_{U,x_0}\big(F_{U,x_0}(c)\big) = \operatorname*{arg\,min}_{x \in \mathcal{X} \cap V_{U,x_0}} c^\top x.$$

If $\mathsf{Exact}(U, x_0, c)$ holds, then the right-hand side is $\mathcal{X}^\star(c)$, so $\mathcal{X}^\star(c) \subseteq V_{U,x_0}$.

Conversely, suppose $\mathcal{X}^\star(c) \subseteq V_{U,x_0}$. Since $\mathcal{X} \cap V_{U,x_0} \subseteq \mathcal{X}$, the restricted LP cannot have value smaller than the original LP. On the other hand, the original optimal face $\mathcal{X}^\star(c)$ is feasible for the restricted LP by assumption. Thus the two optimal values are equal, and the restricted optimizer set is

$$\operatorname*{arg\,min}_{x \in \mathcal{X} \cap V_{U,x_0}} c^\top x = \mathcal{X}^\star(c) \cap V_{U,x_0} = \mathcal{X}^\star(c).$$

This is exactly $\mathsf{Exact}(U, x_0, c)$.

Finally, appending columns enlarges V_{U,x_0}. Therefore containment of the optimal face is preserved, and the monotonicity statement follows from the equivalence just proved. □

To proceed, we show that the containment test and update in Algorithm 1 can be implemented by solving LPs.

Lemma 9 (Containment test via LPs)**.** *Fix $c \in \mathbb{R}^d$, an anchor $x_0 \in \mathcal{X}$, and a matrix U. Let $S := \mathrm{range}(U)$, let*

$$v(c) := \min_{y \in \mathcal{X}} c^\top y, \qquad G(c) := \mathcal{X}^\star(c) = \{x \in \mathcal{X} : c^\top x = v(c)\},$$

and let $a_1, \dots, a_{d-r}$ be any basis of $S^\perp$, where $r := \mathrm{rank}(U)$. Then

$$G(c) \subseteq x_0 + S$$

if and only if, for every $j = 1, \dots, d - r$,

$$\max_{x \in G(c)} a_j^\top (x - x_0) = 0 \quad \textit{and} \quad \min_{x \in G(c)} a_j^\top (x - x_0) = 0.$$

Moreover, if the containment fails, then one of these auxiliary LPs returns an extreme optimizer $x \in G(c) \cap \mathcal{X}^\angle$ satisfying $x - x_0 \notin S$.

Proof. For any $x \in \mathbb{R}^d$, we have $x \in x_0 + S$ if and only if $x - x_0 \in S$, which is equivalent to

$$a_j^\top (x - x_0) = 0 \qquad \text{for all } j = 1, \dots, d - r.$$

Therefore $G(c) \subseteq x_0 + S$ holds if and only if each of the linear functionals $a_j^\top (x - x_0)$ is identically zero on $G(c)$. This is equivalent to requiring both its maximum and its minimum over $G(c)$ to be zero.

If the containment fails, then for some j there exists $\bar{x} \in G(c)$ with $a_j^\top (\bar{x} - x_0) \neq 0$. Hence either

$$\max_{x \in G(c)} a_j^\top (x - x_0) > 0 \quad \text{or} \quad \min_{x \in G(c)} a_j^\top (x - x_0) < 0.$$

Choose an extreme optimal solution x of the corresponding auxiliary LP over $G(c)$. Since $G(c)$ is a face of the polytope $\mathcal{X}$, every extreme point of $G(c)$ is an extreme point of $\mathcal{X}$. The chosen point therefore belongs to $G(c) \cap \mathcal{X}^\angle$ and satisfies $a_j^\top (x - x_0) \neq 0$ for some $a_j \in S^\perp$, so $x - x_0 \notin S$. □

The next three lemmas formalize the key structural properties of our algorithm. Together, they show that Algorithm 1 induces a *stable, realizable* sample compression scheme [Hanneke and Kontorovich, 2021, Definitions 7–8] with compression size at most $d^\star$. This is the key mechanism we use to obtain a distribution-free fast-rate certificate (5).

Lemma 10 (Compression size and termination)**.** *Fix an anchor $x_0 \in \mathcal{X}^\star(\mathcal{C})$ and run Algorithm 1. Then the algorithm makes at most $d^\star$ total appends. Moreover, for its output (U_n, T_n),*

$$\mathrm{range}(U_n) \subseteq \mathrm{dir}(\mathcal{X}^\star(\mathcal{C})), \qquad |T_n| \leq \mathrm{rank}(U_n) \leq d^\star.$$

Proof. Every appended column has the form $x - x_0$, where $x \in \mathcal{X}^\star(c_i) \cap \mathcal{X}^\angle$ for some training sample $c_i \in \mathcal{C}$. Since $x \in \mathcal{X}^\star(\mathcal{C})$ and $x_0 \in \mathcal{X}^\star(\mathcal{C})$, we have

$$x - x_0 \in \mathrm{dir}(\mathcal{X}^\star(\mathcal{C})).$$

Thus every appended direction lies in $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$.

The update rule appends such a vector only when $x - x_0 \notin \mathrm{range}(U_i)$. Hence each append increases the rank of the current matrix by exactly one. Starting from the empty matrix, after k appends the current rank is k, and its range is a k-dimensional subspace of $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$. Since $\dim(\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))) = d^\star$, there can be at most $d^\star$ appends.

This also proves termination: if the while-loop were still active after the current range had dimension $d^\star$, then the algorithm would find $x \in \mathcal{X}^\star(c_i) \cap \mathcal{X}^\angle$ with $x - x_0 \notin \mathrm{range}(U_i)$. But all such differences belong to $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$, and the current range is already a $d^\star$-dimensional subspace of $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$, hence equals $\mathrm{dir}(\mathcal{X}^\star(\mathcal{C}))$, a contradiction.

Finally, every hard sample triggers at least one append, while each append increases the rank by one. Therefore

$$|T_n| \le \mathrm{rank}(U_n) \le d^\star,$$

and the range inclusion follows from the first paragraph. □

Lemma 11 (Realizability)**.** *Under the assumptions of Lemma 10, the output U_n of Algorithm 1 satisfies*

$$\mathsf{Exact}(U_n, x_0, c_i) \qquad \textit{for every } i = 1, \dots, n.$$

Equivalently, the empirical failure loss $1\{\neg\mathsf{Exact}(U_n, x_0, c_i)\}$ is zero on the training sample.

Proof. Fix a training index i. When the while-loop for c_i terminates, it has certified

$$\mathcal{X}^\star(c_i) \subseteq x_0 + \mathrm{range}(U_i).$$

By Lemma 8, this is equivalent to $\mathsf{Exact}(U_i, x_0, c_i)$.

For later samples $t > i$, Algorithm 1 only appends columns to the matrix. Therefore

$$\mathrm{range}(U_i) \subseteq \mathrm{range}(U_t) \qquad \text{for all } t \ge i.$$

The monotonicity statement in Lemma 8 then implies $\mathsf{Exact}(U_n, x_0, c_i)$. □

Lemma 12 (Stability)**.** *Let $S = (c_1, \dots, c_n)$ be the training sequence and let (U_n, T_n) be the output of Algorithm 1. Write*

$$T_n = \{i_1 < \cdots < i_k\}, \qquad S_T := (c_{i_1}, \dots, c_{i_k})$$

where S_T lists the hard samples in their original order. Rerunning Algorithm 1 on S_T with the same anchor x_0 returns U_n. More generally, deleting any subset of non-hard samples from S and rerunning the algorithm in the original sample order also returns U_n.

Proof. For both algorithms, all choices follow fixed rules depending only on the current matrix, the fixed anchor, and the current cost. Let $U^{(j)}$ be the matrix after the jth hard sample c_{i_j} in the original run, with $U^{(0)}$ empty. We show by induction that rerunning on $(c_{i_1}, \dots, c_{i_j})$ produces $U^{(j)}$.

The claim holds for $j = 0$. Suppose it holds for $j - 1$. Between the previous hard sample and c_{i_j}, all samples in the original run are non-hard and leave $U^{(j-1)}$ unchanged. Thus both runs process c_{i_j} with the same matrix and anchor. They append the same directions and end with $U^{(j)}$.

Taking $j = k$ gives the same final matrix U_n. The same induction applies if any non-hard samples are retained: each enters with the original matrix and appends nothing. Hence deleting any subset of non-hard samples also leaves U_n unchanged. □

We now provide the full proof of Theorem 5.

Proof of Theorem 5. The structural claims

$$\mathrm{range}(U_n) \subseteq \mathrm{dir}(\mathcal{X}^\star(\mathcal{C})), \qquad |T_n| \le \mathrm{rank}(U_n) \le d^\star$$

are exactly Lemma 10. The zero-training-failure claim is Lemma 11. It remains to prove the out-of-sample certificate (5).

Sample-compression background. Sample compression is a classical way to prove distribution-free generalization: rather than controlling the complexity of an entire hypothesis class, one shows that the learned predictor can be reconstructed from a small subset of the training sample. This idea goes back to the PAC-learning and compression literature [Valiant, 1984, Littlestone and Warmuth, 1986, Floyd and Warmuth, 1995]; see also Graepel et al. [2005], Moran and Yehudayoff [2016] for later developments. The particular form needed here is the *stable, realizable* version of sample compression. Stable compression schemes and their fast-rate consequences are developed in Bousquet et al. [2020] and Hanneke and Kontorovich [2021]; related compression-based certificates in optimization and scenario analysis are discussed by Campi and Garatti [2023].

We recall the specific form used below. A sample compression scheme for binary prediction consists of a compression map κ, which selects an ordered subsequence of the training sample, and a reconstruction map ρ, which maps the compressed subsequence to a prediction rule. The scheme is *stable* if deleting any uncompressed training examples does not change the reconstructed rule. In the realizable stable case, where the reconstructed rule has zero empirical error, Hanneke and Kontorovich [2021, Definitions 7–8 and Corollary 11] imply that, with probability at least $1-\delta$ over an i.i.d. sample S of size n,

$$R(\rho(\kappa(S))) \le \frac{4}{n}\left(6|\kappa(S)| + \log\frac{e}{\delta}\right),$$

where $R(\cdot)$ is the true 0–1 risk and $|\kappa(S)|$ is the realized compression size. This fast $1/n$ rate is specific to the realizable stable regime; in agnostic compression settings, such fast rates are not available in general [Hanneke and Kontorovich, 2019].

The induced binary learning problem. For a matrix U, define the binary failure rule

$$h_U(c) := 1\{\neg\mathsf{Exact}(U, x_0, c)\}.$$

Equivalently, we may view the data as a supervised binary problem with examples (c, Y), where $c \sim P_c$ and $Y = 0$ almost surely. Under this convention, the risk of h_U is exactly the exact-compression failure probability:

$$R(h_U) = \mathbb{P}_{c\sim P_c}[h_U(c) = 1] = \mathbb{P}_{c\sim P_c}[\neg\mathsf{Exact}(U, x_0, c)]\,.$$

Let $S = (c_1, \dots, c_n)$ and let (U_n, T_n) be the output of Algorithm 1. Define the compression map κ by

$$\kappa(S) := (c_i)_{i\in T_n},$$

where the subsequence is stored in its original order. Since all labels are identically zero, we suppress them in the notation; equivalently, κ stores the labeled examples $(c_i, 0)_{i\in T_n}$. Define the reconstruction map ρ by rerunning Algorithm 1 on any supplied subsequence S' using the same fixed anchor x_0, and returning the classifier $h_{U(S')}$ induced by the resulting matrix $U(S')$.

By Lemma 12, rerunning the algorithm on the compressed hard subsequence $\kappa(S)$ returns the same matrix U_n. Therefore

$$\rho(\kappa(S)) = h_{U_n}.$$

The same lemma also gives the stability property required by Hanneke and Kontorovich [2021]: deleting any subset of the uncompressed samples does not change the reconstructed classifier. By Lemma 11,

$$h_{U_n}(c_i) = 0 \qquad \text{for every } i = 1, \dots, n,$$

so the reconstructed classifier has zero empirical error.

Applying Hanneke and Kontorovich [2021, Corollary 11] to this stable realizable compression scheme yields, with probability at least $1-\delta$,

$$\begin{aligned}
\mathbb{P}_{c\sim P_c}[\neg\mathsf{Exact}(U_n, x_0, c)] &= R(h_{U_n}) \\
&= R(\rho(\kappa(S))) \\
&\le \frac{4}{n}\left(6|\kappa(S)| + \log\frac{e}{\delta}\right) \\
&= \frac{4}{n}\left(6|T_n| + \log\frac{e}{\delta}\right).
\end{aligned}$$

Taking complements gives

$$\mathbb{P}_{c\sim P_c}[\mathsf{Exact}(U_n, x_0, c)] \geq 1 - \frac{4}{n}\left(6|T_n| + \log\frac{e}{\delta}\right).$$

The final inequality in the theorem follows from $|T_n| \leq d^\star$ in Lemma 10.

Anchor generated from data. The theorem assumes that x_0 is fixed independently of the n training costs. If instead x_0 is selected from $\mathcal{X}^\star(c_1)$ by a fixed deterministic rule, retain c_1 in the compressed subsequence so that reconstruction can first recover the same anchor. The bound is then unchanged except that $|T_n|$ is replaced by $|T_n| + 1$. Equivalently, one may draw an independent anchor sample first and apply the theorem to the remaining representation-learning samples.

LP implementation and tractability. It remains to justify the claimed LP-based implementation. By Lemma 9, each containment test

$$\mathcal{X}^\star(c_i) \subseteq x_0 + \mathrm{range}(U_i)$$

can be implemented by first solving the LP defining $v(c_i)$ and then solving, for a basis $a_1, \ldots, a_{d-r}$ of $\mathrm{range}(U_i)^\perp$, the auxiliary LPs

$$\max_{x\in\mathcal{X}:\ c_i^\top x = v(c_i)} a_j^\top(x - x_0), \qquad \min_{x\in\mathcal{X}:\ c_i^\top x = v(c_i)} a_j^\top(x - x_0).$$

If all these optima are zero, containment is certified. If not, an extreme optimal solution of a violated auxiliary LP is a valid vertex $x \in \mathcal{X}^\star(c_i) \cap \mathcal{X}^\angle$ with $x - x_0 \notin \mathrm{range}(U_i)$, so $x - x_0$ is a valid update direction in Algorithm 1.

There is one final containment check for each of the n samples and one additional check for each append. Since Lemma 10 gives at most $d^\star$ total appends, the number of containment checks is at most $n + d^\star$. Each check solves only a polynomial number of LPs over $\mathcal{X}$ or over the optimal face of $\mathcal{X}$ obtained by adding the equality $c_i^\top x = v(c_i)$, and the required orthogonal bases and rank tests are computable by standard linear algebra. Thus, when $\mathcal{X}$ is given in the H-representation of (2), the cumulative learner is polynomial time in the usual LP oracle model. □

B.3 Proofs for Subsection 3.3

B.3.1 Construction of calibrated estimated priors

We construct a distribution-free calibrated prior using the one-sided tolerance-region argument of Wilks [1941]. The construction allows arbitrary data-fitted scores and is closely related to split conformal calibration [Vovk et al., 2005, Vovk, 2012, Lei et al., 2018]. Split an independent pilot sample into a fitting sample $\mathcal{D}_{\mathrm{fit}}$ and a calibration sample $\mathcal{D}_{\mathrm{cal}} = \{c_1^{\mathrm{cal}}, \ldots, c_m^{\mathrm{cal}}\}$. Fit a measurable score $s_{\widehat{\theta}} : \mathbb{R}^d \to \mathbb{R}$ using only $\mathcal{D}_{\mathrm{fit}}$, and assume every sublevel set $\{c : s_{\widehat{\theta}}(c) \leq t\}$ is convex. Let $S_j = s_{\widehat{\theta}}(c_j^{\mathrm{cal}})$, write $S_{(1)} \leq \cdots \leq S_{(m)}$ for the order statistics, and set $S_{(m+1)} := +\infty$.

Fix $\rho, \delta_0 \in (0, 1)$. Define

$$k_{\rho,\delta_0} := \min\left\{k \in \{1, \ldots, m+1\} : \mathbb{P}[\mathrm{Bin}(m, 1-\rho) \geq k] \leq \delta_0\right\},$$
$$\widehat{\mathcal{C}}_{\rho,\delta_0} := \{c : s_{\widehat{\theta}}(c) \leq S_{(k_{\rho,\delta_0})}\}.$$

The number of calibration scores strictly below a population $(1-\rho)$-quantile is stochastically dominated by $\mathrm{Bin}(m, 1-\rho)$. Lemma 14 bounds the resulting tail cutoff. If the calibration sample is too small for the requested confidence, $k_{\rho,\delta_0} = m + 1$ and the calibrated set is all of $\mathbb{R}^d$.

Proposition 13 (Calibrated convex estimated prior)**.** *Under the preceding calibration setup, $\widehat{\mathcal{C}}_{\rho,\delta_0}$ is convex and is a (ρ, δ_0)-estimated prior.*

A simple score with convex sublevel sets is the ridge Mahalanobis score $s_{\widehat{\theta}}(c) := (c - \widehat{\mu})^\top(\widehat{\Sigma} + \lambda I)^{-1}(c - \widehat{\mu})$, where $\lambda > 0$ and $\widehat{\mu}, \widehat{\Sigma}$ are fitted on $\mathcal{D}_{\mathrm{fit}}$. This is the squared Mahalanobis distance [Mahalanobis, 1936] computed with a ridge covariance estimate; the ridge term makes the inverse well-defined and stabilizes high-dimensional covariance estimation [Ledoit and Wolf, 2004]. When the cutoff is finite, the calibrated estimated prior is an ellipsoid.

Proof of Proposition 13. Condition on the fitting sample $\mathcal{D}_{\mathrm{fit}}$, so the score $s_{\widehat{\theta}}$ is fixed. Let $S = s_{\widehat{\theta}}(c)$ for an independent fresh draw $c \sim P_c$, and write $F(t) := \mathbb{P}[S \le t]$. Let

$$q_{1-\rho} := \inf\{t \in \mathbb{R} : F(t) \ge 1-\rho\}$$

be a $(1-\rho)$-quantile of the score distribution. By right-continuity of F, $F(q_{1-\rho}) \ge 1-\rho$, and $F(q_{1-\rho}^-) := \mathbb{P}[S < q_{1-\rho}] \le 1-\rho$.

If $F(S_{(k)}) < 1-\rho$, then $S_{(k)} < q_{1-\rho}$. Hence at least k of the calibration scores are strictly smaller than $q_{1-\rho}$. The number of such scores has distribution $\mathrm{Bin}(m, F(q_{1-\rho}^-))$ and is stochastically dominated by $\mathrm{Bin}(m, 1-\rho)$. Therefore, for $k = k_{\rho,\delta_0}$,

$$\begin{aligned}\mathbb{P}_{\mathrm{cal}}\big[F(S_{(k)}) < 1-\rho \mid \mathcal{D}_{\mathrm{fit}}\big] &\le \mathbb{P}\big[\mathrm{Bin}(m, F(q_{1-\rho}^-)) \ge k\big] \\ &\le \mathbb{P}\big[\mathrm{Bin}(m, 1-\rho) \ge k\big] \le \delta_0.\end{aligned}$$

Equivalently, with conditional probability at least $1-\delta_0$, $P_c(c \in \widehat{\mathcal{C}}_{\rho,\delta_0}) = F(S_{(k)}) \ge 1-\rho$. The same conclusion holds after removing the conditioning on $\mathcal{D}_{\mathrm{fit}}$. Convexity follows because $\widehat{\mathcal{C}}_{\rho,\delta_0}$ is a sublevel set of $s_{\widehat{\theta}}$; if $k = m+1$, then $S_{(m+1)} = +\infty$ and the set is all of $\mathbb{R}^d$. □

Lemma 14 (Size of the cutoff)**.** *Let $X \sim \mathrm{Bin}(m, 1-\rho)$ and $L := \log(1/\delta_0)$. Then*

$$k_{\rho,\delta_0} \le \min\left\{m+1,\ \left\lfloor m(1-\rho) + \sqrt{2m\rho(1-\rho)L} + \frac{2L}{3} \right\rfloor + 1\right\}.$$

Proof. The Bernstein–Chernoff bound for a binomial random variable gives, for every $t \ge 0$,

$$\mathbb{P}[X - m(1-\rho) \ge t] \le \exp\left(-\frac{t^2}{2(m\rho(1-\rho) + t/3)}\right).$$

With $t = \sqrt{2m\rho(1-\rho)L} + 2L/3$, $\mathbb{P}[X \ge m(1-\rho) + t] \le e^{-L} = \delta_0$. Therefore any integer strictly larger than $m(1-\rho) + t$ is feasible in the definition of k_{ρ,δ_0}; the displayed bound follows after truncating at $m+1$. □

B.3.2 Total and retained sample sizes

Here n is the total number of training costs before filtering, separate from the independent pilot used to construct $\widehat{\mathcal{C}}$. Conditional on the pilot, write $p := P_c(c \in \widehat{\mathcal{C}})$. The retention indicators are independent Bernoulli variables with success probability p, so

$$\hat{n} \mid \widehat{\mathcal{C}} \sim \mathrm{Bin}(n, p).$$

For any $\eta \in (0,1)$, Hoeffding's inequality gives

$$\mathbb{P}\left[\left|\frac{\hat{n}}{n} - p\right| > \sqrt{\frac{\log(2/\eta)}{2n}} \,\middle|\, \widehat{\mathcal{C}}\right] \le \eta.$$

Thus $\hat{n} \approx np$, and $\hat{n} \approx (1-\rho)n$ when p is close to $1-\rho$. On $\mathcal{E}_0$, we have $p \ge 1-\rho$, which gives

$$\hat{n} \ge (1-\rho)n - \sqrt{\frac{n\log(2/\eta)}{2}}$$

with joint probability at least $1-\delta_0-\eta$. For fixed $\rho < 1$, the $1/\hat{n}$ term in the learning bound therefore retains the $1/n$ rate as n grows.

Proof of Theorem 7. By the (ρ, δ_0)-estimated-prior property, $\mathbb{P}_{\mathrm{pilot}}(\mathcal{E}_0) \ge 1-\delta_0$. Condition on a pilot realization in $\mathcal{E}_0$. Then $P_c(c \in \widehat{\mathcal{C}}) \ge 1-\rho > 0$, so $P_{\widehat{\mathcal{C}}} = P_c(\cdot \mid c \in \widehat{\mathcal{C}})$ is well-defined and supported on the convex prior $\widehat{\mathcal{C}}$. The chosen anchor $\widehat{x}_0$ belongs to $\mathcal{X}^\star(\widehat{\mathcal{C}})$ by construction.

Conditional also on $\hat{n} = k \ge 1$, the retained costs, in their original order, are i.i.d. from $P_{\widehat{\mathcal{C}}}$. Applying Theorem 5 to these k costs gives, with conditional probability at least $1-\delta_1$,

$$\mathbb{P}_{c\sim P_{\widehat{\mathcal{C}}}}[\neg\mathsf{Exact}(U_{\hat{n}}, \widehat{x}_0, c)] \le \frac{4}{\hat{n}}\big(6\widehat{d}_\star + \log(e/\delta_1)\big).$$

On this event,

$$\mathbb{P}_{c\sim P_c}[\neg\mathsf{Exact}(U_{\hat{n}}, \widehat{x}_0, c)] \le P_c(c \notin \widehat{\mathcal{C}}) + P_c(c \in \widehat{\mathcal{C}})\,\mathbb{P}_{c\sim P_{\widehat{\mathcal{C}}}}[\neg\mathsf{Exact}(U_{\hat{n}}, \widehat{x}_0, c)]$$
$$\le \rho + \frac{4}{\hat{n}}\big(6\widehat{d}_\star + \log(e/\delta_1)\big).$$

Taking complements gives (6). Averaging the conditional failure probability over $k \ge 1$, and adding the probability of $\mathcal{E}_0^c$, gives a joint failure probability of at most $\delta_0 + \delta_1$. The theorem makes no assertion when $\hat{n} = 0$. The final statement follows by applying Proposition 13, which supplies a (ρ, δ_0)-estimated prior. □

C Compression for Integer Programming

In this section, we present our compression framework for integer programming. We first characterize exact compression for IP (Sec. C.1), then give a learning algorithm (Sec. C.2) and evaluate it on unit commitment and knapsack problems (Sec. C.3).

Let

$$\mathcal{X}_I = \{x \in \mathbb{Z}^d : Ax \le b\}$$

be nonempty and finite, and let $\mathcal{X}_I^\star(c) = \arg\min_{x\in\mathcal{X}_I} c^\top x$. Define

$$\mathcal{X}_I^\star(\mathcal{C}) = \bigcup_{c\in\mathcal{C}} \mathcal{X}_I^\star(c), \qquad W_I^\star = \operatorname{span}\{x - y : x, y \in \mathcal{X}_I^\star(\mathcal{C})\}, \qquad d_I^\star = \dim W_I^\star.$$

Choose a reachable anchor $x_0 \in \mathcal{X}_I^\star(\mathcal{C})$. For a matrix $U \in \mathbb{R}^{d\times r}$ of full column rank, the restricted problem is

$$\min_{z\in\mathbb{R}^r} (U^\top c)^\top z \quad \text{subject to} \quad A(x_0 + Uz) \le b, \qquad x_0 + Uz \in \mathbb{Z}^d. \tag{7}$$

The original objective includes the additional constant $c^\top x_0$. Equivalently, if the columns of N span $\operatorname{range}(U)^\perp$, one can keep the original integer variables and add only

$$N^\top(x - x_0) = 0. \tag{8}$$

This form can be passed directly to an integer programming solver.

C.1 The Geometry of Exact Compression for IP

We first present a geometric characterization of IP compression. The following proposition specializes Proposition 2 to integer feasible sets.

Proposition 15. *For any affine space V, the identity*

$$\arg\min_{x\in\mathcal{X}_I\cap V} c^\top x = \mathcal{X}_I^\star(c) \quad \textit{for every } c \in \mathcal{C}$$

holds if and only if $\mathcal{X}_I^\star(\mathcal{C}) \subseteq V$. Consequently, the minimum dimension of an affine restriction preserving every optimizer for every $c \in \mathcal{C}$ is $d_I^\star$, attained by $V = x_0 + W_I^\star$.

Proof. We follow the proof of Proposition 2 in Section B.1, with $\mathcal{X}_I^\star(c)$ in place of $\mathcal{X}^\star(c)$. Suppose first that the optimizer sets are equal for every $c \in \mathcal{C}$. Then $\mathcal{X}_I^\star(c) \subseteq V$ for every $c \in \mathcal{C}$, so $\mathcal{X}_I^\star(\mathcal{C}) \subseteq V$. Conversely, suppose $\mathcal{X}_I^\star(\mathcal{C}) \subseteq V$. For each $c \in \mathcal{C}$, the restricted feasible set contains an original optimizer, so

$$\min_{x\in\mathcal{X}_I\cap V} c^\top x = \min_{x\in\mathcal{X}_I} c^\top x =: v(c).$$

It follows that

$$\arg\min_{x\in\mathcal{X}_I\cap V} c^\top x = \{x \in \mathcal{X}_I \cap V : c^\top x = v(c)\} = \mathcal{X}_I^\star(c).$$

It remains to identify the minimum dimension. Any affine space V containing $\mathcal{X}_I^\star(\mathcal{C})$ also contains the difference $x - y$ in its direction space for every $x, y \in \mathcal{X}_I^\star(\mathcal{C})$. Taking spans gives

$$W_I^\star \subseteq \operatorname{dir}(V), \qquad \dim V \ge d_I^\star.$$

Finally, $x_0 \in \mathcal{X}_I^\star(\mathcal{C})$ gives

$$\mathcal{X}_I^\star(\mathcal{C}) \subseteq x_0 + W_I^\star, \qquad \dim(x_0 + W_I^\star) = d_I^\star,$$

so this affine space attains the minimum dimension. □

Integer coordinates for the compressed formulation. In (7), z has real coordinates, while x_0+Uz must be integral. Even when U has integer entries, replacing $x_0 + Uz \in \mathbb{Z}^d$ by $z \in \mathbb{Z}^r$ can remove feasible integer solutions. For example, $U = [2]$ with $x_0 = 0$ has range $\mathbb{R}$, but restricting z to integers excludes $x = 1$.

For a subspace learned by Algorithm 2, we can express the same restriction with r integer variables. The columns of U are integer differences, so the lattice $\mathrm{range}(U) \cap \mathbb{Z}^d$ has rank r. Choose an integer basis B of this lattice. Then

$$(x_0 + \mathrm{range}(U)) \cap \mathbb{Z}^d = x_0 + B\mathbb{Z}^r,$$

and the equivalent formulation has $z \in \mathbb{Z}^r$ and constraints $A(x_0 + Bz) \le b$. The original difference columns need not form this lattice basis.

C.2 Learning Exact Compression

We next extend the sample-based algorithm of Section 3.2 to IPs. The learner adds differences from a fixed anchor to integer optimizers until the learned affine space contains all optimizers of each training instance. As in Theorem 5, we bound the probability of failing to preserve the full optimizer set for an independent test cost, with high confidence over the training sample.

Fix $x_0 \in \mathcal{X}_I^\star(\mathcal{C})$ independently of the training costs. Algorithm 2 starts with an empty matrix and processes $c_1, \dots, c_n$ one at a time. It appends $x - x_0$ whenever an optimizer $x \in \mathcal{X}_I^\star(c_i)$ is not in the current affine space. Samples that trigger at least one append are called hard instances, and their indices form T_n.

Algorithm 2 Learning exact IP compression

Require: Finite integer feasible set $\mathcal{X}_I$, fixed reachable anchor $x_0 \in \mathcal{X}_I^\star(\mathcal{C})$, training samples $c_1, \dots, c_n$, and deterministic tie-breaking for all choices.
Ensure: A compression matrix U_n and a hard-sample index set T_n.

```
Initialize U_0 ← [ ] ∈ R^{d×0} and T_n ← ∅.
for i = 1, ..., n do
    Set U_i ← U_{i−1}.
    while X*_I(c_i) ⊈ x_0 + range(U_i) do
        Choose x ∈ X*_I(c_i) with x − x_0 ∉ range(U_i).
        U_i ← [U_i  x − x_0] and T_n ← T_n ∪ {i}.
    end while
end for
return U_n and T_n.
```

Compared with Algorithm 1, Algorithm 2 uses integer optimizers $x \in \mathcal{X}_I^\star(c_i)$ in place of LP vertex optimizers and tests containment by solving IPs. For the complexity bound below, let U be the current compression matrix, $r = \mathrm{rank}(U)$, and $v_i = \min_{x \in \mathcal{X}_I} c_i^\top x$. The next theorem gives the IP counterpart of Theorem 5. Its proof implements the containment test using the orthogonal directions in Lemma 9.

Theorem 16. *For $n \ge 1$ i.i.d. costs from a distribution P_c supported on $\mathcal{C}$, this learner satisfies*

$$|T_n| \le \mathrm{rank}(U_n) \le d_I^\star, \qquad \mathcal{X}_I^\star(c_i) \subseteq x_0 + \mathrm{range}(U_n) \quad (i = 1, \dots, n).$$

For any $\delta \in (0,1)$, with probability at least $1-\delta$ over the training sample,

$$\mathbb{P}_{c \sim P_c}[\mathcal{X}_I^\star(c) \not\subseteq x_0 + \mathrm{range}(U_n)] \le \frac{4}{n}\left(6|T_n| + \log\frac{e}{\delta}\right). \tag{9}$$

There are at most $d_I^\star$ appends and $n + d_I^\star$ containment tests. Each test requires at most one IP for v_i and $2(d-r)$ auxiliary IPs.

Proof. We follow the proof of Theorem 5, replacing LP vertex optimizers by integer optimizers.

(i) Compression size and training exactness. As in Lemma 10, each update in Algorithm 2 satisfies

$$x - x_0 \in W_I^\star \setminus \mathrm{range}(U), \qquad \mathrm{rank}[U\ x - x_0] = \mathrm{rank}(U) + 1.$$

Every appended column therefore lies in $W_I^\star$, and each append increases the rank by one. There are at most $d_I^\star$ appends. Since each hard instance causes at least one append, $|T_n| \le \operatorname{rank}(U_n) \le d_I^\star$. The same monotonicity as in Lemma 11 gives

$$\mathcal{X}_I^\star(c_i) \subseteq x_0 + \operatorname{range}(U_i) \subseteq x_0 + \operatorname{range}(U_n).$$

The first inclusion holds when the while-loop for c_i ends, and the second holds because subsequent updates only append columns. Since every appended vector comes from a training optimizer, these inclusions also give

$$x_0 + \operatorname{range}(U_n) = \operatorname{aff}\left(\{x_0\} \cup \bigcup_{i=1}^{n} \mathcal{X}_I^\star(c_i)\right).$$

(ii) Stable reconstruction. For $S = (c_1, \dots, c_n)$, write $U(S)$ for the output matrix and $S_T = (c_i)_{i \in T_n}$ in its original order. If $j \notin T_n$, processing c_j leaves the matrix unchanged. Deleting c_j therefore leaves the input matrix at the next sample unchanged, and induction gives the same matrix at every later sample. This is exactly the induction in the proof of Lemma 12, and it gives

$$U(S_T) = U(S), \qquad U(S \setminus J) = U(S) \quad \text{for every } J \subseteq \{1, \dots, n\} \setminus T_n,$$

where deletion preserves the order of the remaining costs.

(iii) Failure probability. As in the proof of Theorem 5, assign the label zero to every cost and define

$$h_U(c) = 1\{\mathcal{X}_I^\star(c) \not\subseteq x_0 + \operatorname{range}(U)\}$$

The compression map retains S_T, and the reconstruction map reruns Algorithm 2 on that subsequence with the same anchor. The preceding identities show that deleting unretained costs leaves the reconstructed rule h_{U_n} unchanged. The training containment proved above gives $h_{U_n}(c_i) = 0$ for all i. Conditional on the independent anchor, Hanneke and Kontorovich [2021, Corollary 11] therefore yields (9) with compression size $|T_n|$.

(iv) Containment tests. We now implement the containment test, following Lemma 9. Let $v_i = \min_{x \in \mathcal{X}_I} c_i^\top x$, let $r = \operatorname{rank}(U)$, and choose a basis $a_1, \dots, a_{d-r}$ of $\operatorname{range}(U)^\perp$. Then

$$\begin{aligned} &\mathcal{X}_I^\star(c_i) \subseteq x_0 + \operatorname{range}(U) \\ &\quad\iff \min_{x \in \mathcal{X}_I^\star(c_i)} a_j^\top (x - x_0) = \max_{x \in \mathcal{X}_I^\star(c_i)} a_j^\top (x - x_0) = 0 \end{aligned}$$

for every $j = 1, \dots, d - r$. Indeed, these linear functionals all vanish at x exactly when $x - x_0 \in \operatorname{range}(U)$. Each extremum is attained because $\mathcal{X}_I^\star(c_i)$ is finite and nonempty. A nonzero extremum gives $x \in \mathcal{X}_I^\star(c_i)$ with $x - x_0 \notin \operatorname{range}(U)$. The test uses at most one IP to compute v_i and $2(d - r)$ IPs with the added constraint $c_i^\top x = v_i$. The value v_i can be reused while processing the same cost. There is one successful test per sample, and each failed test adds one column. The number of tests is therefore at most $n + d_I^\star$. □

We next discuss four points about implementing and extending the learning algorithm.

1. **Choosing the anchor.** As in the proof of Theorem 5, we can also choose the anchor from the first training instance. If x_0 is selected from $\mathcal{X}_I^\star(c_1)$ by a fixed deterministic rule, retain c_1 so that reconstruction can first recover the same anchor. The compression size is then $|T_n \cup \{1\}| \le |T_n| + 1$, so (9) remains valid with $|T_n| + 1$ in place of $|T_n|$.

2. **Parallel implementation.** When optimizers are unique almost surely, we can use the parallel implementation from Section 3.2. Let x_i be the optimizer for c_i. The affine-hull identity in the proof of Theorem 16 becomes

   $$x_0 + \operatorname{range}(U_n) = x_0 + \operatorname{span}\{x_i - x_0 : i = 1, \dots, n\}.$$

   We can therefore solve the n training IPs in parallel and then compute a basis of these differences. Absolute continuity of P_c on $\mathbb{R}^d$ suffices for uniqueness because ties between two distinct integer points require $c^\top(x - y) = 0$, and $\mathcal{X}_I$ is finite.

   If an objective has several optimizers, choose one by a fixed rule depending only on the cost, and write $x(c)$ for this choice. The proof of Theorem 16, applied to the failure rule $1\{x(c) \notin$

$x_0 + \mathrm{range}(U)\}$, gives the bound in (9) with T_n defined by updates from the selected optimizers. The restricted problem loses the optimal value exactly when

$$\mathcal{X}_I^\star(c) \cap (x_0 + \mathrm{range}(U_n)) = \varnothing.$$

This event implies $x(c) \notin x_0 + \mathrm{range}(U_n)$, so the same probability bound applies. Preserving the entire optimizer set instead uses the containment tests in Algorithm 2.

3. **Learning with an estimated prior.** The estimated-prior construction in Theorem 7 also applies to IPs. Let $\widehat{\mathcal{C}}$ be an independently calibrated estimated prior whose excluded mass is at most ρ with confidence $1 - \delta_0$. Draw $n \geq 1$ i.i.d. costs independently of the pilot and retain the $\hat{n}$ costs inside $\widehat{\mathcal{C}}$. Choose an anchor $\widehat{x}_0 \in \mathcal{X}_I^\star(\widehat{\mathcal{C}})$ independently of these costs. Conditional on the pilot and $\hat{n} = k \geq 1$, apply Theorem 16 to the retained sample from $P_c(\cdot \mid c \in \widehat{\mathcal{C}})$. The argument in Appendix B.3.2 gives, with probability at least $1 - \delta_0 - \delta_1$, the following full-distribution failure bound whenever $\hat{n} \geq 1$:

$$\rho + \frac{4}{\hat{n}} \left(6\widehat{d}_I^\star + \log \frac{e}{\delta_1} \right),$$

where $\widehat{d}_I^\star$ is the dimension of the span of differences between optimizers reachable under $\widehat{\mathcal{C}}$.

4. **Extension to mixed integer programs.** We can also restrict a selected block of integer variables in a mixed integer program. Consider a nonempty bounded mixed integer feasible set in variables (x, y), where $x \in \mathbb{Z}^p$ is the selected block and y contains the remaining integer and continuous variables. For each cost, project the joint optimizer set onto the x coordinates and apply Algorithm 2 to these finite sets. The learned restriction is $x \in x_0 + \mathrm{range}(U)$, with all original variable domains and constraints retained. It preserves every joint optimizer exactly when it contains every optimal x vector. The proofs of Proposition 15 and Theorem 16 therefore apply with $\mathcal{X}_I^\star(c)$ replaced by the projected joint optimizer set. The containment tests become mixed integer programs over the joint optimal set. Sampling one optimizer per cost suffices for full-set preservation when the optimal x vector is unique almost surely.

C.3 Numerical Experiments

We evaluate our algorithm on unit commitment (UC) and knapsack problems with several baselines.

C.3.1 Problem formulations.

Unit commitment. Unit commitment decides which generators run and how much power they produce to meet demand at minimum cost [Xavier et al., 2021]. For generator g in period t, binary variables u_{gt}, v_{gt}, and w_{gt} indicate whether it is on, turns on, or turns off. The continuous variable $p_{gt} \geq 0$ is its power output. We minimize

$$\sum_{g,t} \left(c_g^u u_{gt} + c_g^p p_{gt} + c_g^v v_{gt} \right) + \text{penalties} \quad \text{over the fixed feasible set } \mathcal{F}_{\mathrm{UC}}.$$

The fixed feasible set $\mathcal{F}_{\mathrm{UC}}$ imposes power balance, generator output limits, ramp limits, minimum on/off durations, and reserve requirements. We retain the benchmark's network constraints and penalty variables.

In our experiments, we use the public `matpower/case14/2017-02-01` instance from UnitCommitment.jl v0.4.3 [Xavier et al., 2024]. It has five generators, 14 buses, 20 lines, and 36 hourly periods. We keep its physical data fixed and sample generator costs using the ranges in the MIPLearn v0.4 UC generator.[7] Independently for each generator and cost sample, we draw

$$c_g^u \sim \mathrm{Unif}[0, 1000], \qquad c_g^p \sim \mathrm{Unif}[0, 50], \qquad c_g^v \sim \mathrm{Unif}[0, 10000].$$

These coefficients are rounded to two decimal places and remain constant over the horizon. The benchmark's penalty costs remain fixed. All methods use the same model after exact elimination of redundant auxiliary variables, with 2,844 variables, 2,544 constraints, and 540 binary variables. We compress its 180 on/off variables.

[7] `https://anl-ceeesa.github.io/MIPLearn/0.4/guide/problems/#unit-commitment`.

Knapsack. Knapsack selects items of maximum total value subject to capacity constraints [Kellerer et al., 2004],

$$\max_{x\in\{0,1\}^d} q^\top x \quad \text{subject to} \quad \Omega x \le B.$$

Here q contains item values, and $\Omega x \le B$ describes capacities and, when present, conflicts between items. This has the minimization form above with $c = -q$. We consider two synthetic settings.

1. **Grouped knapsack.** The first knapsack model has 32 groups, each containing two basic items and one upgrade, giving 96 binary variables. Each group has its own capacity, allowing both basic items together or the upgrade alone, but preventing the upgrade from being combined with either basic item. A fixed conflict graph specifies pairs of upgrades that cannot be selected together [Pferschy and Schauer, 2009]. These constraints remain fixed across all training and test samples. Each basic item has value 0.25. For each cost sample, the 32 upgrade values are drawn independently from the uniform distribution on $[0.1, 2.5]$.
2. **Correlated knapsack.** We also use correlated knapsack instances based on Pisinger [2005]. Each instance has 100 items with weights a_j drawn independently and uniformly from the integers $1, \dots, 1000$. Its single capacity is $\lfloor \frac{1}{2} \sum_j a_j \rfloor$. After the weights and capacity are fixed, the item values (cost vectors) are independently sampled for each query as

$$q_j = a_j + 100 + \varepsilon_j, \qquad \varepsilon_j \sim \mathrm{Unif}\{-2, -1, 0, 1, 2\}.$$

C.3.2 Experimental setup.

Training and testing scale. Each sample is a draw of the objective coefficient vector c. UC experiments use 512 training samples, 64 separate validation samples, and 256 independent test samples. Grouped knapsack uses three instances, each with 36 training samples and 120 test samples. Correlated knapsack uses three instances, with nested training sets of sizes $n \in \{32, 64, 128, 256\}$ and the same 100 test samples per instance. All methods share the same training samples.

Baselines. OURS uses our proposed algorithm in Section C.2. We also compare with SVM [Xavier et al., 2021], Neural Diving [Nair et al., 2020], MLOPT [Bertsimas and Stellato, 2022], and Preference Learning [Li et al., 2025]. For each instance, all compression methods are calibrated to the same reduced dimension K for the same set of binary variables. That is, K is the dimension of the affine restriction on these binary variables.

Following the prediction approach of Xavier et al. [2021], SVM trains linear classifiers using the cost coefficients as inputs and the binary values in the optimal training solutions as labels. For UC, it also learns whether a generator's on/off status stays unchanged between consecutive periods. We use the most reliable predictions to fix variables or add equalities while matching K.

Neural Diving trains a graph network to predict binary variable values from the optimal training solutions, following Nair et al. [2020]. At test time, it fixes the variables with the highest prediction confidence.

Following Bertsimas and Stellato [2022], MLOPT trains a neural classifier to predict strategies extracted from the optimal training solutions. Each strategy records binary variable values and the constraints that hold with equality. Preference Learning uses the pairwise ranking objective of Li et al. [2025] to compare strategies. Its scores account for objective value and constraint violations. For both methods, we fix part of the suggested assignment to match K.

Metrics. For test i, let $v_i^\star$ be the original optimum and v_i^m the value returned by method m. These are total costs for UC and total profits for knapsack. Recovery is the percentage of tests where the returned solution is feasible for the original model and within $10^{-6} + 10^{-9}|v_i^\star|$ of its optimal value. The per-test budget is 30 seconds for UC and grouped knapsack, and 20 seconds for correlated knapsack.

For each UC or correlated knapsack instance, all methods use the same anchor: an optimal solution at the first training cost. Its value at test i is v_i^a. The relative objective ratio is

$$\frac{\sum_i (v_i^m - v_i^a)}{\sum_i (v_i^\star - v_i^a)}.$$

This is the fraction of possible improvement over the anchor recovered across all tests. Higher is better, and one is optimal. For grouped knapsack, we average $v_i^m / v_i^\star$ over feasible returns.

Learning time covers the computation used to learn the compression after the training costs and corresponding optimal solutions are provided. For neural methods, this includes choosing model settings and training the network. For our method, it covers Algorithm 2 excluding the training IP solves. We exclude these solves because all methods use their solutions.

Implementation details. OURS, SVM, and MLOPT use one CPU thread. Neural Diving and Preference Learning use an RTX 4090 for UC and an RTX PRO 4500 for both knapsack settings.

We select model settings on UC's separate validation set. For knapsack, we hold out approximately 20% of the training costs. After selection, each model is refitted on all n training costs. SVM uses five-fold cross-validation, with input features standardized within each training fold.

MLOPT uses three ReLU hidden layers of width 32 or 64. We train it with Adam and cross-entropy loss for 50 or 150 epochs, choosing the learning rate from $\{10^{-3}, 10^{-2}\}$. Neural Diving uses a six-layer graph network of width 32 or 64. We train it with Adam and binary cross-entropy at learning rate 10^{-3}, with early stopping within 150 epochs. For correlated knapsack, validation can select up to 2,000 further training updates.

Preference Learning uses two attention blocks of width 128. We train it with AdamW for 1,200 epochs and select the learning rate from $\{10^{-4}, 10^{-3}\}$. Its loss fits both the ordering of strategies and the differences between their quality scores.

C.3.3 Results

Tables 3 to 5 show that OURS achieves the highest recovery and objective ratios among the compression methods tested. Notably, our learning time is much shorter than that of the neural baselines.

Table 3: Results for unit commitment.

n	Method	K	Recovery (%)	Relative ratio	Learning time (s)
—	FULL	180	100.00	1.000000	—
512	**OURS**	**19**	**98.44**	**0.999952**	**0.0088**
	SVM	19	97.66	0.999818	4.18
	Neural Diving	19	73.05	0.923578	86.12
	MLOPT	19	93.75	0.997785	47.01
	Preference Learning	19	43.36	0.763776	172.38

Table 4: Results for grouped knapsack.

n	Method	K	Recovery (%)	Objective ratio	Learning time (s)
—	FULL	96	100.00	1.000000	—
36	**OURS**	**32**	**100.00**	**1.000000**	**0.0072**
	SVM	32	1.39	0.898644	0.8526
	Neural Diving	32	10.83	0.948165	13.17
	MLOPT	32	9.72	0.953409	0.6562
	Preference Learning	32	15.00	0.952625	122.00

Table 5: Results for correlated knapsack. Each K triple lists the dimensions for the three instances.

n	Method	K	Recovery (%)	Relative ratio	Learning time (s)
—	FULL	100	100.00	1.000000	—
32	**OURS**	$(29, 1, 31)$	**94.00**	**0.990033**	**0.0032**
	SVM	$(29, 1, 31)$	86.33	0.971049	0.2103
	MLOPT	$(29, 1, 31)$	89.33	0.982914	1.10
	Neural Diving	$(29, 1, 31)$	91.67	0.983863	103.81
	Preference Learning	$(29, 1, 31)$	86.00	0.972947	86.58
64	**OURS**	$(33, 1, 33)$	**96.67**	**0.995254**	**0.0072**
	SVM	$(33, 1, 33)$	92.33	0.983389	0.2653
	MLOPT	$(33, 1, 33)$	96.33	0.994779	3.07
	Neural Diving	$(33, 1, 33)$	91.67	0.897010	113.66
	Preference Learning	$(33, 1, 33)$	86.33	0.976270	86.18
128	**OURS**	$(35, 1, 37)$	**98.67**	**0.998102**	**0.0129**
	SVM	$(35, 1, 37)$	95.00	0.989084	0.3336
	MLOPT	$(35, 1, 37)$	98.33	0.997152	3.35
	Neural Diving	$(35, 1, 37)$	93.33	0.964404	124.59
	Preference Learning	$(35, 1, 37)$	89.33	0.973422	132.67
256	**OURS**	$(40, 1, 41)$	**99.67**	**0.999525**	**0.0244**
	SVM	$(40, 1, 41)$	97.00	0.992406	0.5422
	MLOPT	$(40, 1, 41)$	99.33	0.999051	6.58
	Neural Diving	$(40, 1, 41)$	94.67	0.966303	170.96
	Preference Learning	$(40, 1, 41)$	93.00	0.984813	240.31

D Additional Experimental Details and Results

This section collects the experimental material not shown in the main text.

LP instances. We write all LPs in the form

$$\min_{x \in \mathbb{R}^d} \{c^\top x : \ Ax \leq b\},$$

and keep the feasible region fixed while varying the cost vector. Equalities are written as pairs of inequalities, and variable bounds are included in $Ax \leq b$. Table 6 gives the resulting numbers of constraints and variables.

Synthetic datasets. We consider packing and network-flow LPs, as in Sakaue and Oki [2024], and also include shortest-path LPs.

Packing. We use 360 nonnegative variables and 60 resource constraints. Each variable appears in only one of these constraints.

MaxFlow. We use a directed acyclic network with 64 nodes and 300 arcs. We connect pairs of parallel paths in series and add randomly selected forward arcs between internal nodes.

MinCostFlow. We use the same graph construction with 86 nodes and 360 arcs, including a direct source-to-sink arc. All arcs have unit capacity.

ShortestPath. We use a 16×16 directed grid with rightward and downward arcs.

We also include four random bounded LPs, denoted by RandomLP A–D, with Gaussian constraint matrices.

Netlib datasets. We use GROW7, SC205, SCAGR25, and STAIR from Netlib. These instances are preprocessed similarly to Sakaue and Oki [2024], and their reported dimensions refer to the resulting inequality-form LPs.

Table 6: Here m is the number of inequality constraints and d is the number of variables. Lower and upper bounds on variables are included in m. Each equality is replaced by two inequalities.

Instance	Packing	MaxFlow	MinCostFlow	ShortestPath
m	420	724	892	1472
d	360	300	360	480
Instance	RandomLP A	RandomLP B	RandomLP C	RandomLP D
m	380	500	620	740
d	140	180	220	260
Instance	GROW7	SC205	SCAGR25	STAIR
m	861	499	1271	1126
d	301	203	500	473

Cost perturbations. We use shared factors to vary the relative rewards of competing activities or the relative costs of alternative routes. For additive Gaussian cost perturbations, we use the factor model

$$c = c_0 + U_c\theta, \qquad \theta_j = \sigma_j z_j, \qquad z_j \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0,1),$$

where $c_0 \in \mathbb{R}^d$ is the nominal cost vector and $U_c \in \mathbb{R}^{d\times r_c}$ contains the cost directions. Before clipping, the factors have standard deviations σ_j that are equal or decrease by a fixed factor across directions. The released code specifies the sampling parameters. The exact learner in the known-prior experiment is given the ball $\mathcal{C}_i = \{c : \|c - c_{0,i}\|_2 \le R_i^{\mathcal{C}}\}$. We radially clip each Gaussian perturbation to ensure $\|c - c_0\|_2 \le R_i$.

In the unknown-prior experiments the learner is not given $\mathcal{C}_i$. The costs are generated in the same way but are not clipped:

$$c = c_0 + U_c\theta, \qquad \theta_j \sim \mathcal{N}\big(0, (\eta_i\sigma_j)^2\big),$$

where η_i is an experiment-specific scale multiplier.

Table 7: Default LP sampling parameters.

Instance	r_c	σ_j	$R_i^{\mathcal{C}}$	R_i	η_i
Packing	28	$0.92R_i(0.94)^{j-1}$	1.694	1.524	0.35
MaxFlow	12	$0.95R_i$	1.278×10^{-3}	1.150×10^{-3}	—
MinCostFlow	28	$0.72R_i(0.94)^{j-1}$	3.968	3.492	0.35
ShortestPath	16	$0.70R_i(0.82)^{j-1}$	0.04725	0.04488	—
RandomLP A	24	$0.70R_i(0.82)^{j-1}$	2.866×10^{-3}	2.637×10^{-3}	20
RandomLP B	26	$0.70R_i(0.82)^{j-1}$	8.069×10^{-3}	7.262×10^{-3}	20
RandomLP C	28	$0.70R_i(0.82)^{j-1}$	2.145×10^{-3}	1.888×10^{-3}	20
RandomLP D	30	$0.70R_i(0.82)^{j-1}$	2.398×10^{-3}	2.063×10^{-3}	20
GROW7	20	$0.70R_i(0.82)^{j-1}$	0.01263	0.01162	0.35
SC205	20	$0.70R_i(0.82)^{j-1}$	3.891×10^{-4}	3.580×10^{-4}	0.35
SCAGR25	24	$0.70R_i(0.82)^{j-1}$	0.04802	0.04418	—
STAIR	10	$0.70R_i(0.82)^{j-1}$	8.144×10^{-5}	1.629×10^{-6}	—

Dashes in the η_i column mean not applicable (we did not run unknown-prior experiments for these instances).

D.1 Dimension growth under an estimated prior

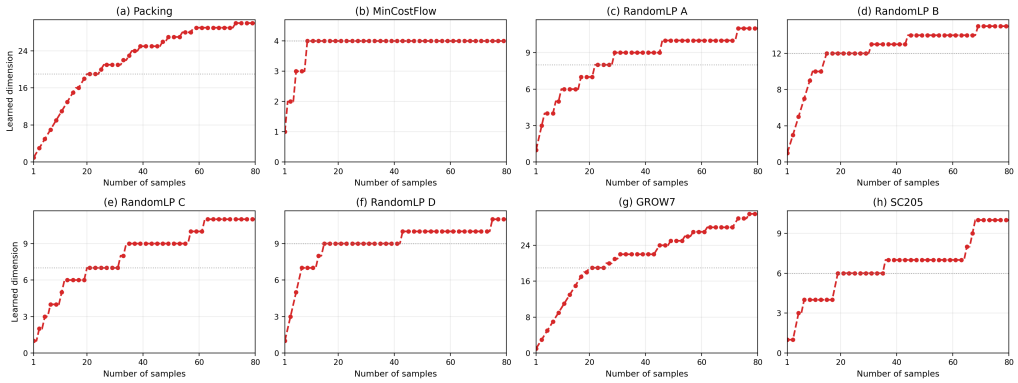


Figure 4: Learned dimension of OURS. LP sizes are given in Table 6.

Figure 4 reports the dimension learned from the costs retained by the estimated prior. The MinCostFlow curve plateaus early. Packing and GROW7 attain larger learned dimensions than the other instances. These dimensions reflect the optimizer differences observed in the retained sample.

D.2 Known-prior benchmark

Protocol. The known-prior benchmark uses Packing, MaxFlow, MinCostFlow, ShortestPath, RandomLP A–D, and the Netlib instances GROW7, SC205, SCAGR25, and STAIR. In this experiment the exact learner is given the intended prior set and is denoted OURSEXACT. We compare it with RAND, DATADRIVENPROJ, FCNN, and FULL, as discussed in Section 4.

In the reduced-dimension sweep, the projection baselines are evaluated at $K \in \{5, 10, 20, 30, 40, 50\}$. OURSEXACT learns its dimension from the training costs. Its objective ratio is therefore shown as a reference across all displayed K values. In the sample-efficiency study, $K = 20$ is fixed only for the approximate baselines, while OURSEXACT again uses the dimension discovered by Algorithm 1. The main metric is the average test objective ratio.

Standard errors are estimated by dividing the sample standard deviation by the square root of the number of observations.

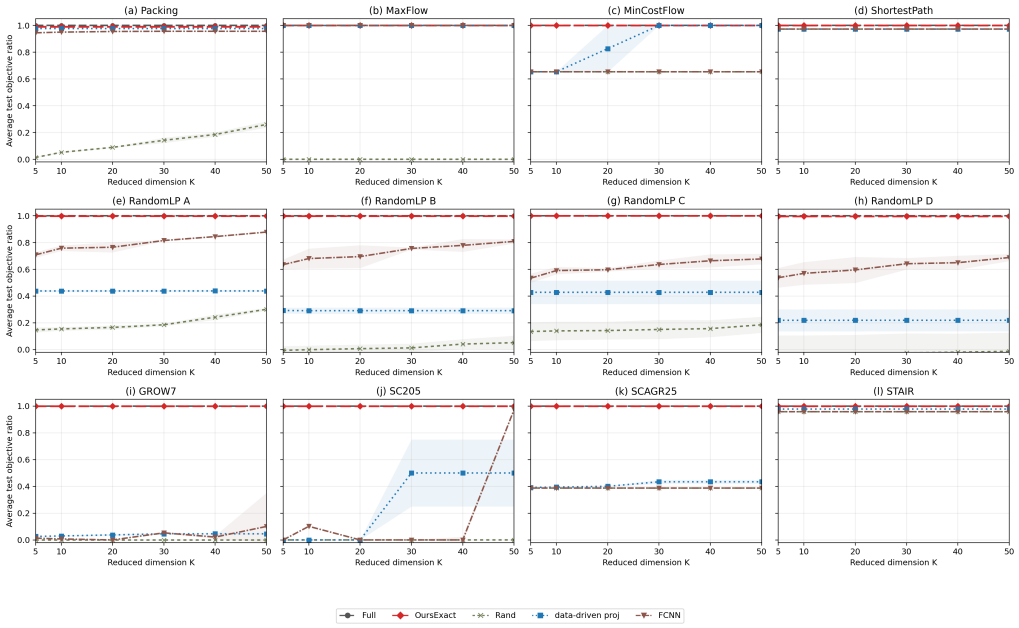


Figure 5: Objective ratio as the projection-baseline dimension K varies under known priors. Our method is plotted as a reference across all displayed K values.

Figure 5 shows higher objective ratios for OURSEXACT on the RandomLP families and several Netlib instances. OURSEXACT attains objective ratios close to one across these panels, whereas DATADRIVENPROJ remains far below one on RandomLP B/D, GROW7, SCAGR25, and much of SC205. FCNN is often the strongest approximate baseline on RandomLP A–D and can improve at larger K, but it still falls short of exact compression except in a few large-K cases.

On Packing, MaxFlow, ShortestPath, and STAIR, the learned projection baselines attain objective ratios close to one. On MinCostFlow, DATADRIVENPROJ approaches the full-LP objective as K increases. The OURSEXACT reference uses the same learned compression throughout the sweep.

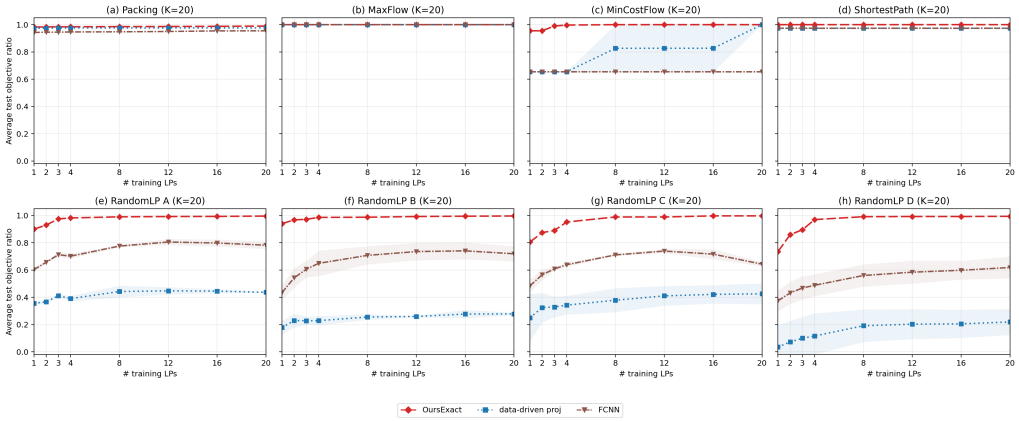


Figure 6: Objective ratio as the sample size varies under known priors.

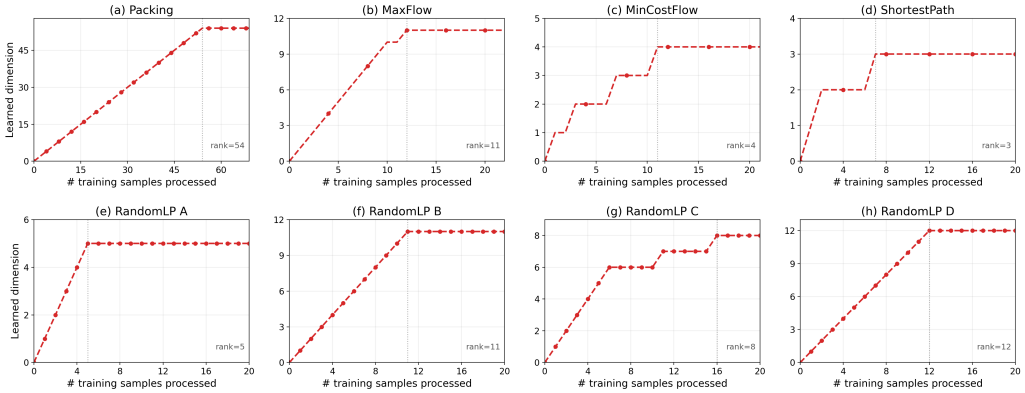


Figure 7: Learned dimension growth of our algorithm on the original synthetic LP instances.

Figure 6 highlights the sample-efficiency difference in the known-prior setting. On RandomLP A–D, OURSEXACT quickly reaches an objective ratio close to one as more optimal-face directions are observed, while DATADRIVENPROJ stays around a substantially lower plateau and FCNN improves more gradually. On Packing, MaxFlow, and ShortestPath, both OURSEXACT and the projection baselines attain objective ratios close to one.

Figure 7 reports the learned dimensions on the same synthetic LP instances. This figure is the known-prior analogue of Figure 4.

D.3 Additional unknown-prior diagnostics

Protocol. These additional experiments use the ridge Mahalanobis score with a threshold equal to the largest training score. The resulting estimated prior contains all observed training costs. The same costs are used to estimate the prior and learn the compression, so the displayed sample budgets count the historical costs available to each method. The projection baselines are trained directly on the observed cost samples and do not construct an estimated prior.

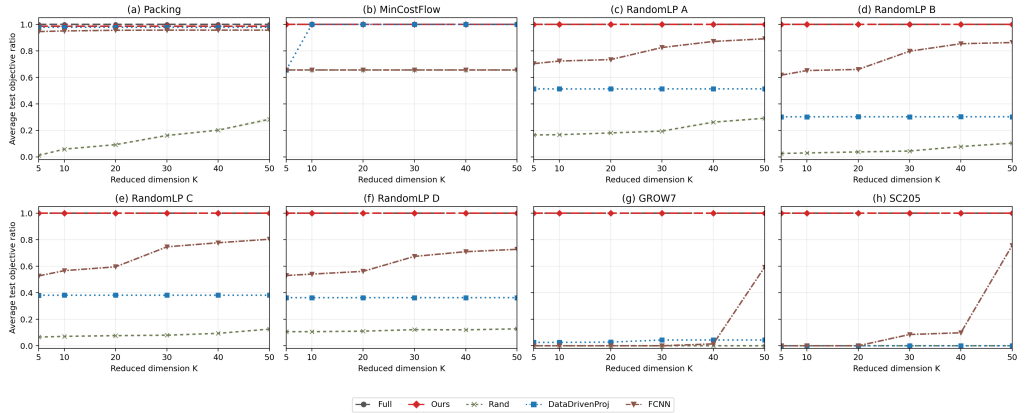


Figure 8: Reduced-dimension sweep with an unknown prior set.

Figure 8 compares objective ratios as the baseline projection dimension varies. OURS attains objective ratios close to one. The baselines can be strong on Packing and MinCostFlow, but they remain substantially below exact compression on RandomLP B–D and GROW7.

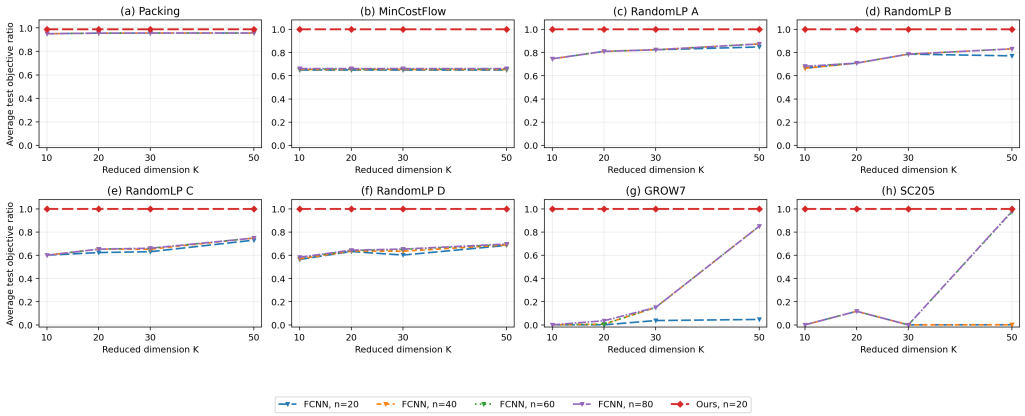


Figure 9: Reduced-dimension sweep for FCNN, with the $n = 20$ OURS result as a reference.

Figure 9 complements Figure 3 by varying the reduced dimension K for FCNN at each displayed training-sample size. Figures 10 and 11 report the corresponding diagnostics for the DATADRIVEN-PROJ baseline.

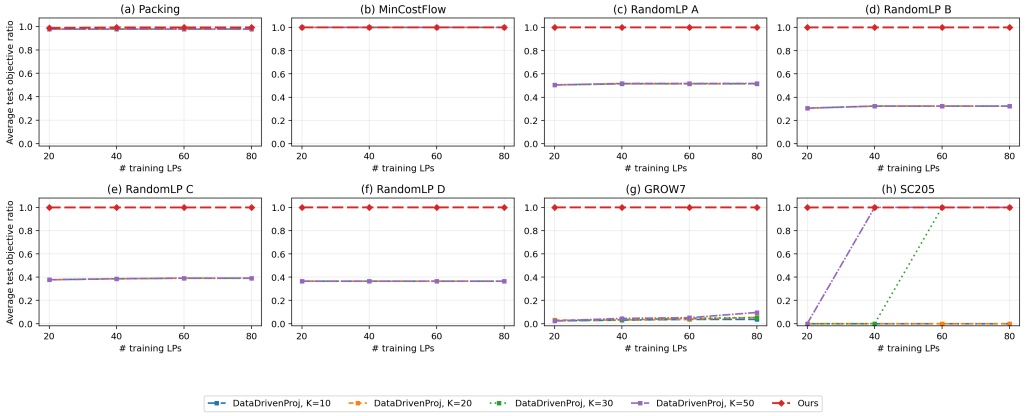


Figure 10: Sample-efficiency study for the DATADRIVENPROJ baseline, with OURS shown as the reference.

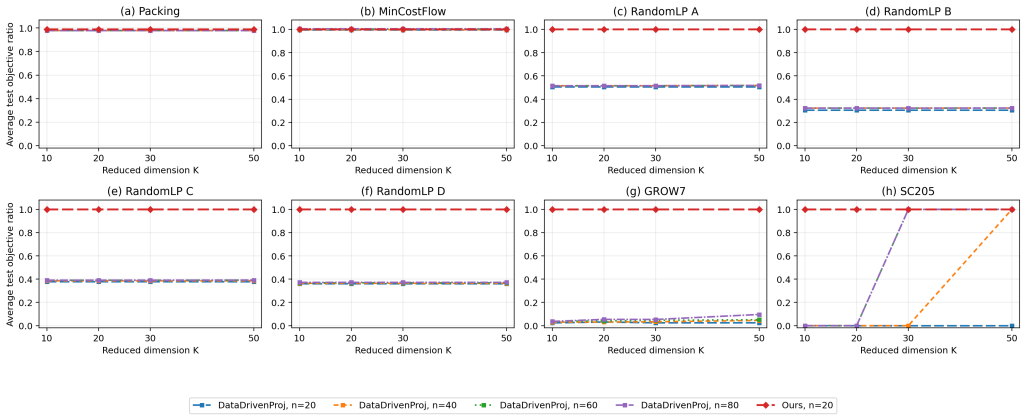


Figure 11: Reduced-dimension sweep for the DATADRIVENPROJ baseline, with the $n = 20$ OURS result shown as the reference.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims made in the abstract and introduction accurately reflect the paper's contribution and scope.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Yes, it is done in Section 5.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: Yes, all necessary proofs were provided in the appendix.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Experimental protocols are described in Section 4 and Appendices C.3 and D. Additional implementation details, configurations, and instructions for reproducing the results are provided in the GitHub repository linked in Section 4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Code, saved results, and reproduction instructions are available at `https://github.com/yeyuhanyyh/Solving-Linear-and-Integer-Programs-Faster-Using-Data`.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: Training and test protocols are described in Section 4 and Appendices C.3 and D. Additional implementation details and hyperparameter settings are provided in the GitHub repository linked in Section 4.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Error bars are provided for figures and tables that demonstrate our main results.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The LP timing experiments use an Intel Core i5-13500H CPU with 32 GB RAM, running Windows 11. The IP experiments also use CPU workers and RunPod GPUs: an RTX 4090 for UC and an RTX PRO 4500 for knapsack. Appendix C.3 specifies which methods use each device. Learning times are reported in the experimental tables. Preliminary experiments used additional compute.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The paper does conform with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

Justification: This is foundational research on compression for repeated LPs and IPs at a controlled probability of preserving optimality. Its societal impacts depend on the applications in which the underlying optimization problems are used. We see no direct path to harmful applications beyond those already enabled by standard LP and IP solvers.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper studies LP and IP compression using synthetic optimization instances and public LP benchmarks. It introduces no pretrained generative models or scraped datasets requiring these safeguards.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The Netlib LP test instances (GROW7, SC205, SCAGR25, STAIR) are obtained from the public Netlib LP repository (`https://www.netlib.org/lp/`), which distributes them for unrestricted research use. The projection baselines reimplement the

protocols of Sakaue and Oki [2024] and Iwata and Sakaue [2025], both cited in the paper. LPs are modeled and solved with CVXPY using its default open-source backend. Standard scientific Python libraries (NumPy, SciPy, PyTorch) are used under their respective open-source licenses. All assets are used consistent with their licenses and terms.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The released code and saved experimental records are documented in the repository, with instructions for installation, data preparation, running the methods, and rebuilding the result tables.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs are not used as a component of the core methodology.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.