GFORS: GPU-Accelerated First-Order Method with Randomized Sampling for Binary Integer Programs

Ningji Wei * Jiaming Liang †

October 30, 2025

Abstract

We present GFORS, a GPU-accelerated framework for large binary integer programs. It couples a first-order (PDHG-style) routine that guides the search in the continuous relaxation with a randomized, feasibility-aware sampling module that generates batched binary candidates. Both components are designed to run end-to-end on GPUs with minimal CPU–GPU synchronization. The framework establishes near-stationary-point guarantees for the first-order routine and probabilistic bounds on the feasibility and quality of sampled solutions, while not providing global optimality certificates. To improve sampling effectiveness, we introduce techniques such as total-unimodular reformulation, customized sampling design, and monotone relaxation. On classic benchmarks (set cover, knapsack, max cut, 3D assignment, facility location), baseline state-of-the-art exact solvers remain stronger on small–medium instances, while GFORS attains high-quality incumbents within seconds; on large instances, GFORS yields substantially shorter runtimes, with solution quality often comparable to—or better than—the baseline under the same time limit. These results suggest that GFORS can complement exact solvers by delivering scalable, GPU-native search when problem size and response time are the primary constraints.

Keywords: GPU-accelerated optimization, binary integer programs, first-order methods

1 Introduction

This paper develops a GPU-accelerated algorithmic framework targeting the following type of binary integer programs (BIPs)

$$\min_{x\in\{0,1\}^n} \ \langle x, Qx\rangle + \langle c, x\rangle \tag{1a}$$

$$Ax \geq b \tag{1b}$$

$$Bx = d, \tag{1c}$$

where $A \in \mathbb{R}^{m_1\times n}$ and $B \in \mathbb{R}^{m_2\times n}$ encode the inequality and equality constraints, and $Q \in \mathbb{R}^{n\times n}$ is assumed, without loss of generality, to be symmetric but not necessarily positive semidefinite (PSD).

*Department of Industrial, Manufacturing & Systems Engineering, Texas Tech University, Lubbock, TX 79409 (email: ningji.wei@ttu.edu).

†Goergen Institute for Data Science and Artificial Intelligence (GIDS-AI) and Department of Computer Science, University of Rochester, Rochester, NY 14620 (email: jiaming.liang@rochester.edu). This work was partially supported by GIDS-AI seed funding and AFOSR grant FA9550-25-1-0182.

Approach	Complexity & Structure	Certifiability	Scalability
Exact methods	High (exploitation of polyhedral/structural complexity)	Strong (optimality certificates)	Limited at very large scale
Metaheuristics	Low (limited structure use, problem-agnostic)	Weak (no formal guarantees)	High (flexible across problems)
GFORS	Moderate (first-order information, sampling signals)	Moderate (near-stationary-point convergence, probability bounds)	High (GPU-accelerated)

Table 1: Comparison of the three solution frameworks.

This problem class encompasses many canonical combinatorial optimization problems (e.g., set cover, knapsack) and is NP-hard in general. The area is well studied: a large body of literature spans problem-specific algorithms [3, 4, 39, 43, 50, 52], exact formulations and approximations [10, 19, 24, 49, 55], and general-purpose solvers [1, 8, 25, 28]. These developments have pushed exact solvers to tackle instance sizes previously impractical [31], enabling decision support across domains such as defense, transportation, and communications. Meanwhile, decision problems have grown both larger and more time-critical, driven by advances in technology, business scale, and data availability over the past decade. For instance, logistics networks have expanded from local or national operations to international flows, while delivery windows have reduced from multi-week horizons to days or even hours in many markets [48, 59]. As a result, when facing large-scale BIP instances under tight decision windows, practitioners often resort to ad-hoc procedures, greedy strategies, or metaheuristics, prioritizing efficiency over exactness.

These challenges stem from a practical three-way trade-off among (i) problem complexity (NP-hardness and structure), (ii) solution exactness (certifiability), and (iii) algorithmic scalability. Exact solution methods are primarily grounded in (i) and (ii) and have significantly advanced the frontier of (iii), but they often plateau on sufficiently large instances. Metaheuristics emphasize (iii) and can deliver strong incumbents, but they typically make limited use of model structure and offer no certifiability. Our approach targets (i) and (iii) through the *GPU-accelerated First-Order method with Randomized Sampling* (GFORS). We *do not* claim global optimality bounds; rather, we establish near-stationary-point bound for the implemented first-order routine and probability guarantees on feasibility and optimality for the sampling scheme. A high-level comparison of these approaches is provided in Table 1. The GFORS framework consists of three components:

- *First-Order Method for Region Exploration:* We apply a first-order method to a modified form of (1) that steers fractional solutions toward promising regions. Section 2.1 establishes convergence results for the implemented *primal–dual hybrid gradient* (PDHG) variant [12, 40, 41].
- *Randomized Sampling for Candidate Generation:* Each iteration of the first-order method yields a fractional solution $x \in [0,1]^n$, which admits a natural probabilistic interpretation (Section 2.2). This enables a sampling subroutine guided by the central path of the first-order component. In Section 2.3, we will derive probability bounds on feasibility and optimality for this subroutine.
- *GPU Acceleration for Scalability:* Both components are designed for GPU execution, enabling the framework to run with minimal CPU–GPU synchronization and to fully leverage the scalability of modern GPU architectures.

To strengthen the framework and improve sampling efficiency, we introduce additional techniques such as total-unimodular reformulation, customized sampling design, and monotone relaxation. We evaluate the resulting GFORS framework on a range of classic BIPs, including set cover, knapsack, max cut, 3D assignment, and facility location. An aggregated overview is presented in Figure 1 (see Section 4.1): on small- to medium-scale instances, Gurobi achieves stronger performance, yet GFORS still delivers high-quality solutions within runtimes of seconds; for large-scale instances, however, GFORS achieves substantially shorter runtimes, while its solution quality is often comparable to—or better than—that of Gurobi within the prescribed time limit. This is *not* a head-to-head algorithmic comparison, as GFORS offers neither optimality guarantees nor the generality of Gurobi; instead, the findings highlight GFORS's potential to complement exact methods by providing scalability on large BIP instances with short decision windows.

1.1 Related Literature

Exact solvers. Each state-of-the-art exact MIP solver (e.g., Gurobi, CPLEX, SCIP, Xpress, and CBC) deploys a portfolio of techniques—strong presolve [2, 22], cutting-plane separation [11, 25], local-search heuristics [15, 20, 21], and parallel branch-and-bound/branch-and-cut [3, 35]—to deliver certified optimality on large, heterogeneous instances. For comprehensive overviews, see [7, 9, 27]. These components are combined within a parallelized tree search with adaptive cut/heuristic scheduling, representing the prevailing standard for exact MILP/MIQP/BIP.

First-order methods. First-order methods such as gradient descent [6, 46], proximal algorithms [45, 53], and PDHG [12, 13] are widely used in large-scale convex optimization due to their low per-iteration cost and amenability to parallelization. They have been extensively applied in continuous domains such as signal processing [14], imaging [12], and, more recently, large-scale linear programming [40, 41]. Their direct application to mixed-integer programming (MIP), however, remains nascent: the loss of convexity arising from integrality limits direct applicability, and the tree-based search procedures underlying exact MIP solvers are difficult to parallelize efficiently on GPUs.

GPU-accelerated algorithms. GPUs are now central to large-scale machine learning, where massive data and models benefit from highly parallel, vectorized first-order methods with low per-iteration cost, enabling deep neural networks training via gradient-based algorithms [17, 30]. Beyond machine learning, GPU-accelerated solvers exist for linear, conic, and nonlinear optimization, including cuPDLP [40, 41], SCS [47], OSQP [56], and MadNLP.jl [54]. Notably, cuPDLP demonstrates substantial performance gains on large-scale linear programs. However, these tools target continuous optimization and do not directly handle binary integer programs.

Quadratic unconstrained binary optimization (QUBO). QUBO addresses problems of the form $\min_{x\in\{0,1\}^n} \langle x, Qx\rangle + \langle c, x\rangle$ with *no constraints*, which are equivalent to Ising formulations and thus attractive for specialized hardware and sampling-based methods [32, 42]. Leading approaches include classical metaheuristics, exact branch-and-bound variants, and quantum/quantum-inspired algorithms (e.g., quantum annealing [29], QAOA [18]) that naturally return batches of samples per evaluation. Despite a shared probabilistic perspective, the mechanisms differ: QAOA/annealing sample from fixed, parameterized states over unconstrained encodings, whereas GFORS performs

adaptive, GPU-batched sampling guided by first-order signals and uses dual updates to explicitly drive constraint feasibility.

Throughout the paper, for a matrix A and index sets J (rows) and I (columns), we denote by A_J the submatrix consisting of rows in J, by $A_{\cdot I}$ the submatrix consisting of columns in I, and by A_{JI} the submatrix restricted to both rows J and columns I. For individual indices $j \in J$ and $i \in I$, A_j, $A_{\cdot i}$, and A_{ji} refer to the corresponding row, column, and entry, respectively. For any function f differentiable at y, the affine function $\ell_f(\cdot; y)$ denotes its first-order approximation at y. We use (x_*, y_*) to denote a saddle point, and x^* to denote an optimal solution of an optimization problem. The remainder of the paper is organized as follows. Section 2 presents the main design of the GFORS framework. Section 3 describes additional implementation details. Section 4 reports both high-level and problem-specific computational results. Finally, Section 5 concludes with further discussions. The code repository is available at https://github.com/tidues/GFORS.

2 GFORS Framework

This section introduces the main design of GFORS for solving BIPs of the form (1). To enable GPU acceleration, we reformulate this problem into the following saddle-point form with an additional fractional penalty term. To ease the notation, let $y = (u, v)$ be the concatenated dual variables for (1b) and (1c), and let $K := -[A; B]$ and $r := (b; d)$ be the row-wise concatenation of the respective matrices and vectors.

$$\min_x \max_y \left\{ \Phi(x; u, v) := \hat{\mathcal{L}}(x; y) + h_1(x) - h_2(y) \right\} \tag{2a}$$

$$\text{where } \hat{\mathcal{L}}(x; y) := \langle x, Qx \rangle + \langle c, x \rangle + \langle y, Kx + r \rangle + \rho \langle x, 1 - x \rangle, \tag{2b}$$

$$h_1(x) := I_{[0,1]^n}(x), \tag{2c}$$

$$h_2(x) := I_{\mathbb{R}_+^{m_1} \times \mathbb{R}^{m_2}}(y). \tag{2d}$$

The function $\hat{\mathcal{L}}$ denotes the Lagrangian of the linear relaxation of (1) along with the fractional penalty term $\rho\langle x, 1 - x\rangle$, and $I_{\mathcal{X}}(x)$ is the indicator function that equals 0 when $x \in \mathcal{X}$ and $+\infty$ otherwise. For fixed ρ and y, the function $\hat{\mathcal{L}}(\cdot; y)$ is weakly convex and L-smooth with $L := 2(\|Q\| + \rho)$.

Building on this formulation, GFORS is structured around two GPU-friendly components. The underlying intuition is that the first-order method guides the fractional solution toward increasingly promising regions, while the randomized sampling module explores nearby binary solutions according to their likelihood of optimality.

First-Order Component: First-order methods guide solutions toward promising regions. In particular, when the objective function is weakly convex, many first-order methods guarantee convergence to a stationary point [23, 33, 34, 37, 38]. Within the GFORS framework, however, stationarity guarantees and convergence rates are not the primary focus. Instead, the central path generated by the algorithm balances exploration and exploitation: it sweeps through promising regions to ensure diverse coverage while simultaneously refining solutions within those regions using the sampling component. We choose to implement a variant of the PDHG algorithm [12, 62] due to its demonstrated strong performance for linear programs and compatibility with GPU acceleration [40]. In Section 2.1, we provide the associated near-stationarity bound of (2) in the general setting for a fixed ρ and well-tuned step sizes.

Algorithm 1 GFORS Framework

Parameters: binary penalty ρ; sampling interval k_{int}; sampling rounds k_r; batch size k_b

CPU:

TUReformulate() ▷ perform TU reformulation if necessary (Section 2.4.1)

Preprocess() ▷ normalize model parameters (Section 3)

GPU:

initialization: $x_0 \in [0,1]^n$; $k \leftarrow 0$; $z_{\text{best}} \leftarrow \infty$; $x_{\text{best}} \leftarrow \varnothing$; halt $\leftarrow$ false

while not halt **do**

$k \leftarrow k+1$

$\rho \leftarrow$ UpdatePenalty() ▷ increase penalty by schedule (Section 3)

$x_k \leftarrow$ FirstOrderStep(x_{k-1}, ρ) ▷ update x by a first-order method (Section 2.1)

if $k \equiv 0 \pmod{k_{\text{int}}}$ **then** ▷ sampling trigger for efficiency

for $i \in [k_r]$ **do**

$\bar{X}^k \leftarrow$ RandSampleStep(x_k, k_b) ▷ randomized/customized sampling (Section 2.2)

$(z_{\text{best}}, x_{\text{best}}) \leftarrow$ EvalBest$(\bar{X}^k, z_{\text{best}}, x_{\text{best}})$;

end for

end if

halt $\leftarrow$ CheckHalt(); ▷ stopping criteria (Section 3)

end while

$(z_{\text{best}}, x_{\text{best}}) \leftarrow$ EvalBest$(\text{round}(x_k), z_{\text{best}}, x_{\text{best}})$;

CPU:

return z_{best}, x_{best}

Randomized Sampling Component: Each iteration of the first-order method produces a fractional solution $x \in [0,1]^n$, which admits a natural probabilistic interpretation (see Section 2.2). This facilitates a sampling subroutine that, after every k_{int} iterations, efficiently generates candidate binary solutions from the current fractional solution, thereby promoting exploration through a set of sampled outcomes. While the first-order component steers the central path toward increasingly promising regions, the sampling component complements it by continually probing candidates within these regions. When the entropy of x is high (i.e., x is far from binary points), the sampling is more exploratory; when the entropy is low, the sampling concentrates on binary solutions with higher potential. Together, these two components allow GFORS to balance broad exploration with focused refinement throughout the iterative process.

Because both components are GPU-friendly, the entire framework can be executed on GPU hardware with minimal CPU–GPU synchronization. Algorithm 1 summarizes the overall procedure. It begins with reformulation and preprocessing subroutines on the CPU, followed by a GPU-accelerated iterative loop. In each iteration, the binary penalty parameter ρ is updated, a fractional solution x_k is advanced using a first-order method, and a batch of binary solutions is generated through randomized sampling and evaluated for incumbent update. The batch size k_b is to limit the GPU memory peak usage in the sampling subroutine. This loop continues until the halting criterion is satisfied. Finally, the incumbent solution and objective value are synchronized back to the CPU for output. To maintain a clear, high-level view of the framework, we omit certain input and output parameters from these subroutines.

This section therefore will focus on the core functions FirstOrderStep and RandSampleStep, along with the supporting TUReformulate subroutine that enhances sampling when equality constraints are present. Additional subroutines are mainly implementation details and will be

Algorithm 2 PDHG Update

Parameters: preprocessed input (Q, K, c, r); step size $\tau_1, \tau_2 > 0$
function FIRSTORDERSTEP(x_{k-1}, ρ)
  $y_k \leftarrow \Pi_{\mathbb{R}_+^{m_1} \times \mathbb{R}^{m_2}}(y_{k-1} + \tau_2(K\bar{x}_{k-1} + r))$
  $\delta \leftarrow c + \rho + K^\top y_k + 2Qx_{k-1} - 2\rho x_{k-1}$
  $x_k \leftarrow \Pi_{[0,1]^n}(x_{k-1} - \tau_1\delta)$
  $\bar{x}_k \leftarrow 2x_k - x_{k-1}$
  return x_k
end function

deferred to Section 3.

2.1 First-Order Update

The FIRSTORDERSTEP subroutine can be instantiated with any first-order method tailored for the saddle-point problem (2). In our implementation, we adopt the update mechanism of the PDHG algorithm [12, 62], which has demonstrated strong performance for linear programs and efficient compatibility with GPU acceleration [40].

Algorithm 2 presents the detailed pseudocode. This subsection analyzes its convergence behavior under convex settings (e.g., Q is PSD and $\rho = 0$) and the more general weakly convex cases. The following relations will be used to derive these results.

$$\text{Gradients: } \nabla_x \hat{\mathcal{L}}(x, y) = c + \rho + K^\top y + 2Qx - 2\rho x,\ \nabla_y \hat{\mathcal{L}}(x, y) = Kx + r. \tag{3a}$$

$$\text{Smoothness: } \|\nabla_x \hat{\mathcal{L}}(x_1, y) - \nabla_x \hat{\mathcal{L}}(x_2, y)\| \le 2(\|Q\| + \rho)\|x_1 - x_2\| = L\|x_1 - x_2\|. \tag{3b}$$

$$\text{PDHG updates: } y_k = \operatorname*{argmin}_{y \in \mathbb{R}^m} \left\{ -\hat{\mathcal{L}}(\bar{x}_{k-1}, y) + h_2(y) + \frac{1}{2\tau_2}\|y - y_{k-1}\|^2 \right\}, \tag{3c}$$

$$x_k = \operatorname*{argmin}_{x \in \mathbb{R}^n} \left\{ \ell_{\hat{\mathcal{L}}(\cdot, y_k)}(x; x_{k-1}) + h_1(x) + \frac{1}{2\tau_1}\|x - x_{k-1}\|^2 \right\}, \tag{3d}$$

$$\bar{x}_k = 2x_k - x_{k-1}. \tag{3e}$$

$$\text{Auxiliaries: } x_{-1} = x_0 = \bar{x}_0,\ y^0 = y_0,\ y^k = y_k + \tau_2 K(\bar{x}_k - x_k). \tag{3f}$$

In particular, (3b) ensures that $\hat{\mathcal{L}}(\cdot, y_k)$ is L-smooth, and (3c)–(3e) follow the update rules in Algorithm 2.

In the following, Proposition 1 and Theorem 1 establish convergence guarantees under the convexity assumption of $\hat{\mathcal{L}}(\cdot, y)$, while the subsequent Proposition 2 and Theorem 2 extend the results to near-stationarity for general nonconvex settings (e.g., Q is non-PSD and/or $\rho > 0$).

Proposition 1. *Assume $\hat{\mathcal{L}}(\cdot, y_k)$ is convex, $\tau_1\tau_2\|K\|^2 \le 1/4$, and $\tau_1 \le 1/(4L)$. Define*

$$\varepsilon_k := \hat{\mathcal{L}}(x_k, y_k) - \ell_{\hat{\mathcal{L}}(\cdot, y_k)}(x_k; x_{k-1}). \tag{4}$$

Then, the following statements hold:

$$(a)\ r_k^y := \frac{y^{k-1} - y^k}{\tau_2} \in \partial\left[-\hat{\mathcal{L}}(x_k, \cdot) + h_2\right](y_k),\ r_k^x := \frac{x_{k-1} - x_k}{\tau_1} \in \partial_{\varepsilon_k}[\hat{\mathcal{L}}(\cdot, y_k) + h_1(\cdot)](x_k); \tag{5}$$

(b) $\varepsilon_k \le \dfrac{1}{8\tau_1}\|x_k - x_{k-1}\|^2,\ \dfrac{1}{\tau_2}\|y^k - y_k\|^2 \le \dfrac{1}{4\tau_1}\|x_k - x_{k-1}\|^2;$ (6)

(c) $\dfrac{1}{4\tau_1}\sum_{k=1}^{N}\|x_k - x_{k-1}\|^2 + \dfrac{1}{2\tau_2}\sum_{k=1}^{N}\|y_k - y^{k-1}\|^2 \le \dfrac{1}{2\tau_1}\|x_* - x_0\|^2 + \dfrac{1}{2\tau_2}\|y_* - y_0\|^2.$ (7)

Proof. (a) The optimality condition of (3c) is

$$0 \in \frac{y_k - y_{k-1}}{\tau_2} - \nabla_y \hat{\mathcal{L}}(\bar{x}_{k-1}, y_k) + \partial h_2(y_k),$$

which together with (3a) and the definition of y^k in (3f) implies that

$$\begin{aligned} 0 &\in \frac{y^k - y^{k-1}}{\tau_2} - K(\bar{x}_k - x_k - \bar{x}_{k-1} + x_{k-1}) - (K\bar{x}_{k-1} + r) + \partial h_2(y_k) \\ &= \frac{y^k - y^{k-1}}{\tau_2} - (Kx_k + r) + \partial h_2(y_k) \overset{(3a)}{=} \frac{y^k - y^{k-1}}{\tau_2} - \nabla_y \hat{\mathcal{L}}(x_k, y_k) + \partial h_2(y_k). \end{aligned}$$

Hence, the first inclusion in (5) follows. The optimality condition of (3d) is

$$\frac{x_{k-1} - x_k}{\tau_1} \in \partial[\ell_{\hat{\mathcal{L}}(\cdot, y_k)}(\cdot; x_{k-1}) + h_1(\cdot)](x_k),$$

which together with the convexity of $\hat{\mathcal{L}}(\cdot, y_k)$ further implies that for every $x \in \operatorname{dom} h_1$,

$$\begin{aligned} &\hat{\mathcal{L}}(x, y_k) + h_1(x) \ge \ell_{\hat{\mathcal{L}}(\cdot, y_k)}(x; x_{k-1}) + h_1(x) \\ \ge &\ell_{\hat{\mathcal{L}}(\cdot, y_k)}(x_k; x_{k-1}) + h_1(x_k) + \frac{1}{\tau_1}\langle x_{k-1} - x_k, x - x_k\rangle \\ = &\hat{\mathcal{L}}(x_k, y_k) - \varepsilon_k + h_1(x_k) + \frac{1}{\tau_1}\langle x_{k-1} - x_k, x - x_k\rangle. \end{aligned} \tag{8}$$

Hence, the second inclusion in (5) follows.

(b) It follows from the definition of ε_k in (4) and the assumption that $\hat{\mathcal{L}}(\cdot, y_k)$ is L-smooth that

$$\varepsilon_k \le \frac{L}{2}\|x_k - x_{k-1}\|^2 \le \frac{1}{8\tau_1}\|x_k - x_{k-1}\|^2,$$

where the second inequality is due to the assumption that $\tau_1 \le 1/(4L)$. Hence, the first inequality in (6) holds. Plugging this inequality into (8), we have

$$\hat{\mathcal{L}}(x, y_k) + h_1(x) - \hat{\mathcal{L}}(x_k, y_k) - h_1(x_k) \ge \frac{3}{8\tau_1}\|x_k - x_{k-1}\|^2 + \frac{1}{2\tau_1}\|x - x_k\|^2 - \frac{1}{2\tau_1}\|x - x_{k-1}\|^2. \tag{9}$$

Using (3e) and (3f), we obtain

$$\frac{1}{\tau_2}\|y^k - y_k\|^2 \overset{(3f)}{=} \tau_2\|K(\bar{x}_k - x_k)\|^2 \overset{(3e)}{=} \tau_2\|K(x_k - x_{k-1})\|^2 \le \tau_2\|K\|^2\|x_k - x_{k-1}\|^2.$$

Hence, the second inequality in (6) follows from the assumption that $\tau_1\tau_2\|K\|^2 \le 1/4$.

(c) The first inclusion in (5) implies that for every $y \in \operatorname{dom} h_2$,

$$-\hat{\mathcal{L}}(x_k, y) + h_2(y) \ge -\hat{\mathcal{L}}(x_k, y_k) + h_2(y_k) + \frac{1}{\tau_2}\langle y^{k-1} - y^k, y - y_k\rangle$$

$$= -\hat{\mathcal{L}}(x_k, y_k) + h_2(y_k) + \frac{1}{2\tau_2}\|y - y^k\|^2 - \frac{1}{2\tau_2}\|y - y^{k-1}\|^2 + \frac{1}{2\tau_2}\|y_k - y^{k-1}\|^2 - \frac{1}{2\tau_2}\|y_k - y^k\|^2.$$

Plugging the second inequality in (6) into the above inequality yields

$$\begin{aligned} &-\hat{\mathcal{L}}(x_k, y) + h_2(y) + \hat{\mathcal{L}}(x_k, y_k) - h_2(y_k) \\ \geq &\frac{1}{2\tau_2}\left(\|y - y^k\|^2 - \|y - y^{k-1}\|^2 + \|y_k - y^{k-1}\|^2\right) - \frac{1}{8\tau_1}\|x_k - x_{k-1}\|^2. \end{aligned} \tag{10}$$

Summing (9) and (10) gives

$$\begin{aligned} \Phi(x, y_k) - \Phi(x_k, y) \geq &\frac{1}{4\tau_1}\|x_k - x_{k-1}\|^2 + \frac{1}{2\tau_1}\|x - x_k\|^2 - \frac{1}{2\tau_1}\|x - x_{k-1}\|^2 \\ &+ \frac{1}{2\tau_2}\|y - y^k\|^2 - \frac{1}{2\tau_2}\|y - y^{k-1}\|^2 + \frac{1}{2\tau_2}\|y_k - y^{k-1}\|^2. \end{aligned}$$

Taking $(x, y) = (x_*, y_*)$ as the saddle point of $\Phi(\cdot, \cdot)$ and using the fact that $\Phi(x_*, y_k) \leq \Phi(x_k, y_*)$, we obtain

$$\frac{1}{4\tau_1}\|x_k - x_{k-1}\|^2 + \frac{1}{2\tau_2}\|y_k - y^{k-1}\|^2 \leq \frac{1}{2\tau_1}\|x_* - x_{k-1}\|^2 - \frac{1}{2\tau_1}\|x_* - x_k\|^2 + \frac{1}{2\tau_2}\|y_* - y^{k-1}\|^2 - \frac{1}{2\tau_2}\|y_* - y^k\|^2.$$

Finally, (7) follows by summing the above inequality over iterations. □

Theorem 1. *Assume $\hat{\mathcal{L}}(\cdot, y_k)$ is convex. Denote*

$$\Delta_0 := \frac{\|x_* - x_0\|^2}{2\tau_1} + \frac{\|y_* - y_0\|^2}{2\tau_2},$$

then for every $N \geq 1$, there exists $k_0 \leq N$ such that

$$r^y_{k_0} \in \partial\left[-\hat{\mathcal{L}}(x_{k_0}, \cdot) + h_2\right](y_{k_0}), \quad r^x_{k_0} \in \partial_{\varepsilon_{k_0}}[\hat{\mathcal{L}}(\cdot, y_{k_0}) + h_1(\cdot)](x_{k_0}), \tag{11}$$

$$\|r^y_{k_0}\| \leq \frac{\sqrt{3\Delta_0}}{\sqrt{\tau_2 N}}, \quad \|r^x_{k_0}\| \leq \frac{2\sqrt{\Delta_0}}{\sqrt{\tau_1 N}}, \quad \varepsilon_{k_0} \leq \frac{\Delta_0}{2N}. \tag{12}$$

Proof. Let k_0 be the index such that

$$k_0 = \operatorname{argmin}\left\{\frac{1}{4\tau_1}\|x_k - x_{k-1}\|^2 + \frac{1}{2\tau_2}\|y_k - y^{k-1}\|^2 : k = 1, \ldots, N\right\}.$$

It follows from Proposition 1 (a) that (11) holds. Using Proposition 1 (c), we have

$$\frac{1}{4\tau_1}\|x_{k_0} - x_{k_0-1}\|^2 + \frac{1}{2\tau_2}\|y_{k_0} - y^{k_0-1}\|^2 \overset{(7)}{\leq} \frac{\|x_* - x_0\|^2}{2\tau_1 N} + \frac{\|y_* - y_0\|^2}{2\tau_2 N} = \frac{\Delta_0}{N}. \tag{13}$$

Hence, it follows from the first inequality in Proposition 1 (b) that

$$\|r^x_{k_0}\| \overset{(5)}{=} \frac{\|x_{k_0} - x_{k_0-1}\|}{\tau_1} \leq \frac{2\sqrt{\Delta_0}}{\sqrt{\tau_1 N}}, \quad \varepsilon_{k_0} \overset{(6)}{\leq} \frac{1}{8\tau_1}\|x_{k_0} - x_{k_0-1}\|^2 \leq \frac{\Delta_0}{2N}.$$

We have thus proved the last two inequalities in (12). Using the Cauchy-Schwarz inequality and the second inequality in (6), we have

$$\frac{1}{\tau_2}\|y^k - y^{k-1}\|^2 \le 3\left(\frac{1}{\tau_2}\|y^k - y_k\|^2 + \frac{1}{2\tau_2}\|y_k - y^{k-1}\|^2\right) \overset{(6)}{\le} 3\left(\frac{1}{4\tau_1}\|x_k - x_{k-1}\|^2 + \frac{1}{2\tau_2}\|y_k - y^{k-1}\|^2\right),$$

which together with (13) implies that

$$\|r_{k_0}^y\| \overset{(5)}{=} \frac{\|y^{k_0} - y^{k_0-1}\|}{\tau_2} \le \frac{\sqrt{3\Delta_0}}{\sqrt{\tau_2 N}}.$$

Therefore, we have thus proved the first inequality in (12). □

Proposition 2. *Assume*

$$\tau_1\tau_2\|K\|^2 \le 1/2, \quad \tau_1 \le 1/(2L). \tag{14}$$

Then, the following statement holds:

$$\frac{1}{\tau_1}\sum_{k=1}^N \|x_k - x_{k-1}\|^2 + \frac{1}{\tau_2}\sum_{k=1}^N \|y_k - y_{k-1}\|^2 \le 4\Delta\Phi_N + \frac{8}{\tau_1}\sum_{k=1}^N \|x_{k-1}\|^2 + 16\tau_2\|r\|^2 N, \tag{15}$$

where $\Delta\Phi_N := \Phi(x_0, y_0) - \Phi(x_N, y_N)$.

Proof. It follows from (3d) that for every x,

$$\begin{aligned}
&\ell_{\hat{\mathcal{L}}(\cdot,y_k)}(x; x_{k-1}) + h_1(x) + \frac{1}{2\tau_1}\|x - x_{k-1}\|^2 - \frac{1}{2\tau_1}\|x - x_k\|^2 \\
\ge &\ell_{\hat{\mathcal{L}}(\cdot,y_k)}(x_k; x_{k-1}) + h_1(x_k) + \frac{1}{2\tau_1}\|x_k - x_{k-1}\|^2 \\
\ge &\hat{\mathcal{L}}(x_k, y_k) + h_1(x_k) - \frac{L}{2}\|x_k - x_{k-1}\|^2 + \frac{1}{2\tau_1}\|x_k - x_{k-1}\|^2.
\end{aligned}$$

Taking $x = x_{k-1}$ yields

$$\hat{\mathcal{L}}(x_{k-1}, y_k) + h_1(x_{k-1}) \ge \hat{\mathcal{L}}(x_k, y_k) + h_1(x_k) + \left(\frac{1}{\tau_1} - \frac{L}{2}\right)\|x_k - x_{k-1}\|^2.$$

Noting h_2 is an indicator function, and thus $h_2(y_{k-1}) = h_2(y_k) = 0$. This observation and the above inequality thus imply that

$$\begin{aligned}
\Phi(x_{k-1}, y_{k-1}) - \Phi(x_k, y_k) &= \hat{\mathcal{L}}(x_{k-1}, y_{k-1}) - \hat{\mathcal{L}}(x_k, y_k) + h_1(x_{k-1}) - h_1(x_k) \\
&= \hat{\mathcal{L}}(x_{k-1}, y_{k-1}) - \hat{\mathcal{L}}(x_{k-1}, y_k) + \hat{\mathcal{L}}(x_{k-1}, y_k) - \hat{\mathcal{L}}(x_k, y_k) + h_1(x_{k-1}) - h_1(x_k) \\
&\overset{(3a)}{\ge} \langle Kx_{k-1} + r, y_{k-1} - y_k\rangle + \left(\frac{1}{\tau_1} - \frac{L}{2}\right)\|x_k - x_{k-1}\|^2.
\end{aligned}$$

Rearranging the terms gives

$$\left(\frac{1}{\tau_1} - \frac{L}{2}\right)\|x_k - x_{k-1}\|^2 \le \Phi(x_{k-1}, y_{k-1}) - \Phi(x_k, y_k) + \langle Kx_{k-1} + r, y_k - y_{k-1}\rangle. \tag{16}$$

Similarly, it follows from (3c) that for every y,

$$-\hat{\mathcal{L}}(\bar{x}_{k-1}, y) + h_2(y) + \frac{1}{2\tau_2}\|y - y_{k-1}\|^2 - \frac{1}{2\tau_2}\|y - y_k\|^2 \geq -\hat{\mathcal{L}}(\bar{x}_{k-1}, y_k) + h_2(y_k) + \frac{1}{2\tau_2}\|y_k - y_{k-1}\|^2.$$

Taking $y = y_{k-1}$ yields

$$-\hat{\mathcal{L}}(\bar{x}_{k-1}, y_{k-1}) + h_2(y_{k-1}) \geq -\hat{\mathcal{L}}(\bar{x}_{k-1}, y_k) + h_2(y_k) + \frac{1}{\tau_2}\|y_k - y_{k-1}\|^2.$$

Noting $h_2(y_{k-1}) = h_2(y_k) = 0$ and using (3a), the above inequality becomes

$$\langle K\bar{x}_{k-1} + r, y_k - y_{k-1}\rangle \overset{(3a)}{=} \hat{\mathcal{L}}(\bar{x}_{k-1}, y_k) - \hat{\mathcal{L}}(\bar{x}_{k-1}, y_{k-1}) \geq \frac{1}{\tau_2}\|y_k - y_{k-1}\|^2.$$

Summing the above inequality and (16), we have

$$\begin{aligned}
&\left(\frac{1}{\tau_1} - \frac{L}{2}\right)\|x_k - x_{k-1}\|^2 + \frac{1}{\tau_2}\|y_k - y_{k-1}\|^2 - [\Phi(x_{k-1}, y_{k-1}) - \Phi(x_k, y_k)] \\
\leq& 2\langle Kx_{k-1} + r, y_k - y_{k-1}\rangle + \langle K(\bar{x}_{k-1} - x_{k-1}), y_k - y_{k-1}\rangle \\
\overset{(3e)}{=}& 2\langle Kx_{k-1} + r, y_k - y_{k-1}\rangle + \langle K(x_{k-1} - x_{k-2}), y_k - y_{k-1}\rangle \\
\leq& 2(\|K\|\|x_{k-1}\| + \|r\|)\|y_k - y_{k-1}\| + \|K\|\|x_{k-1} - x_{k-2}\|\|y_k - y_{k-1}\|,
\end{aligned}$$

where the last inequality is due to the Cauchy-Schwarz inequality. It thus follows from the Young's and (14) that

$$\begin{aligned}
&\left(\frac{1}{\tau_1} - \frac{L}{2}\right)\|x_k - x_{k-1}\|^2 + \frac{1}{\tau_2}\|y_k - y_{k-1}\|^2 - [\Phi(x_{k-1}, y_{k-1}) - \Phi(x_k, y_k)] \\
\leq& 4\tau_2\|K\|^2\|x_{k-1}\|^2 + 4\tau_2\|r\|^2 + \frac{1}{2\tau_2}\|y_k - y_{k-1}\|^2 + \tau_2\|K\|^2\|x_{k-1} - x_{k-2}\|^2 + \frac{1}{4\tau_2}\|y_k - y_{k-1}\|^2 \\
\overset{(14)}{\leq}& \frac{2}{\tau_1}\|x_{k-1}\|^2 + 4\tau_2\|r\|^2 + \frac{3}{4\tau_2}\|y_k - y_{k-1}\|^2 + \frac{1}{2\tau_1}\|x_{k-1} - x_{k-2}\|^2.
\end{aligned}$$

Rearranging the terms and summing the resulting inequality over iterations, we have

$$\left(\frac{1}{2\tau_1} - \frac{L}{2}\right)\sum_{k=1}^{N}\|x_k - x_{k-1}\|^2 + \frac{1}{4\tau_2}\sum_{k=1}^{N}\|y_k - y_{k-1}\|^2 \leq \Delta\Phi_N + \frac{2}{\tau_1}\sum_{k=1}^{N}\|x_{k-1}\|^2 + 4\tau_2\|r\|^2 N,$$

where we use the fact that $x_0 = x_{-1}$. □

Theorem 2. *Denote the primal and dual residuals as*

$$s_k^x = \frac{x_{k-1} - x_k}{\tau_1} + \nabla_x\hat{\mathcal{L}}(x_k, y_k) - \nabla_x\hat{\mathcal{L}}(x_{k-1}, y_k),\ \ s_k^y = \frac{y_{k-1} - y_k}{\tau_2} - \nabla_y\hat{\mathcal{L}}(x_k, y_k) + \nabla_y\hat{\mathcal{L}}(\bar{x}_{k-1}, y_k).$$

Assume $\Phi(x, y)$ is bounded by Φ_* from below. Then, for every $N \geq 1$, there exists $k_0 \leq N$ such that

$$s_{k_0}^x \in \nabla_x\hat{\mathcal{L}}(x_{k_0}, y_{k_0}) + \partial h_1(x_{k_0}),\ \ s_k^y \in -\nabla_y\hat{\mathcal{L}}(x_{k_0}, y_{k_0}) + \partial h_2(y_{k_0}), \tag{17}$$

$$\|s_{k_0}^x\| + \|s_{k_0}^y\| \leq 2\left[\left(2\|K\| + \frac{3}{2\tau_1}\right)^2\tau_1 + \frac{1}{\tau_2}\right]^{1/2}\left[\frac{\Phi(x_0, y_0) - \Phi_*}{N} + \frac{2}{\tau_1 N}\sum_{k=1}^{N}\|x_{k-1}\|^2 + 4\tau_2\|r\|^2\right]^{1/2}. \tag{18}$$

Proof. It follows from (3d) that

$$0 \in \nabla_x \hat{\mathcal{L}}(x_{k-1}, y_k) + \partial h_1(x_k) + \frac{x_k - x_{k-1}}{\tau_1},$$

which together with the definition of s_k^x implies that

$$s_k^x = \frac{x_{k-1} - x_k}{\tau_1} + \nabla_x \hat{\mathcal{L}}(x_k, y_k) - \nabla_x \hat{\mathcal{L}}(x_{k-1}, y_k) \in \nabla_x \hat{\mathcal{L}}(x_k, y_k) + \partial h_1(x_k).$$

Similarly, it follows from (3c) that

$$0 \in -\nabla_y \hat{\mathcal{L}}(\bar{x}_{k-1}, y_k) + \partial h_2(y_k) + \frac{y_k - y_{k-1}}{\tau_1},$$

which together with the definition of s_k^y implies that

$$s_k^y = \frac{y_{k-1} - y_k}{\tau_2} - \nabla_y \hat{\mathcal{L}}(x_k, y_k) + \nabla_y \hat{\mathcal{L}}(\bar{x}_{k-1}, y_k) \in -\nabla_y \hat{\mathcal{L}}(x_k, y_k) + \partial h_2(y_k).$$

Thus, we conclude that (17) holds. Moreover, the definition of s_k^x, (3b), and (14) yield that

$$\|s_k^x\| \leq \left(\frac{1}{\tau_1} + L\right) \|x_{k-1} - x_k\| \overset{(14)}{\leq} \frac{3}{2\tau_1} \|x_{k-1} - x_k\|. \tag{19}$$

The definition of s_k^y, (3a), and (3e) imply that

$$\begin{aligned} \|s_k^y\| &\leq \frac{1}{\tau_2} \|y_k - y_{k-1}\| + \|K\| \|x_k - \bar{x}_{k-1}\| \\ &\overset{(3e)}{\leq} \frac{1}{\tau_2} \|y_k - y_{k-1}\| + \|K\| (\|x_k - x_{k-1}\| + \|x_{k-1} - x_{k-2}\|). \end{aligned}$$

It follows from the fact that $x_0 = x_{-1}$ that

$$\begin{aligned} \sum_{k=1}^N \|s_k^y\| &\leq \frac{1}{\tau_2} \sum_{k=1}^N \|y_k - y_{k-1}\| + \|K\| \sum_{k=1}^N (\|x_k - x_{k-1}\| + \|x_{k-1} - x_{k-2}\|) \\ &\leq \frac{1}{\tau_2} \sum_{k=1}^N \|y_k - y_{k-1}\| + 2\|K\| \sum_{k=1}^N \|x_k - x_{k-1}\| \end{aligned}$$

Combining this inequality with (19), we obtain

$$\sum_{k=1}^N (\|s_k^x\| + \|s_k^y\|) \leq \frac{1}{\tau_2} \sum_{k=1}^N \|y_k - y_{k-1}\| + \left(2\|K\| + \frac{3}{2\tau_1}\right) \sum_{k=1}^N \|x_k - x_{k-1}\|. \tag{20}$$

Using the Cauchy-Schwarz inequality, we have

$$\left(\frac{1}{\tau_2} \sum_{k=1}^N \|y_k - y_{k-1}\| + \left(2\|K\| + \frac{3}{2\tau_1}\right) \sum_{k=1}^N \|x_k - x_{k-1}\|\right)^2$$

$$\leq \left[\left(2\|K\| + \frac{3}{2\tau_1}\right)^2 \tau_1 N + \frac{N}{\tau_2}\right] \left[\frac{1}{\tau_1}\sum_{k=1}^{N}\|x_k - x_{k-1}\|^2 + \frac{1}{\tau_2}\sum_{k=1}^{N}\|y_k - y_{k-1}\|^2\right].$$

It thus follows from Proposition 2 that

$$\begin{aligned}
&\frac{1}{\tau_2}\sum_{k=1}^{N}\|y_k - y_{k-1}\| + \left(2\|K\| + \frac{3}{2\tau_1}\right)\sum_{k=1}^{N}\|x_k - x_{k-1}\| \\
&\overset{(15)}{\leq} 2\sqrt{N}\left[\left(2\|K\| + \frac{3}{2\tau_1}\right)^2 \tau_1 + \frac{1}{\tau_2}\right]^{1/2}\left[\Phi(x_0, y_0) - \Phi_* + \frac{2}{\tau_1}\sum_{k=1}^{N}\|x_{k-1}\|^2 + 4\tau_2\|r\|^2 N\right]^{1/2}.
\end{aligned}$$

Finally, (18) immediately follows by plugging the above inequality into (20). □

2.2 Randomized Sampling

Consider the binary space $\{0,1\}^n$. Any fractional solution $p \in [0,1]^n$ naturally induces a fully independent joint distribution over $\{0,1\}^n$, defined for each $\hat{x} \in \{0,1\}^n$ as

$$\pi(p, \hat{x}) := \Pr(\hat{x} \mid p) \;=\; \prod_{i\in[n]} p_i^{\hat{x}_i}(1-p_i)^{1-\hat{x}_i}.$$

The associated expectation vector $x_p = \sum_{\hat{x}\in\{0,1\}^n} \pi(p,\hat{x})\hat{x}$ represents the probabilistic mixture of binary solutions governed by p. With these definitions, we obtain the following intuitive proposition.

Proposition 3. *$x_p = p$ for every $p \in [0,1]^n$.*

Proof. The following identity holds for every $p \in [0,1]^n$

$$\sum_{\hat{x}\in\{0,1\}^n} \pi(p,\hat{x}) = 1,$$

since $\pi(p,\cdot)$ is a probability distribution over $\{0,1\}^n$. Without loss of generality, we focus on expanding the last entry of $x_p := \sum_{\hat{x}\in\{0,1\}^n} \pi(p,\hat{x})\hat{x}$ as follows, where $\bar{x} \in \{0,1\}^{n-1}$ and $\bar{p} \in [0,1]^{n-1}$ denote the projections of $\hat{x}$ and p onto the first $n-1$ entries:

$$\begin{aligned}
(x_p)_n &= \sum_{\{(\bar{x},1)\mid \bar{x}\in\{0,1\}^{n-1}\}} \pi(p,(\bar{x},1))\cdot 1 + \sum_{\{(\bar{x},0)\mid \bar{x}\in\{0,1\}^{n-1}\}} \pi(p,(\bar{x},0))\cdot 0 \\
&= \sum_{\bar{x}\in\{0,1\}^{n-1}} \left(\prod_{i\in[n-1]} p_i^{\bar{x}_i}(1-p_i)^{1-\bar{x}_i}\right) p_n \\
&= p_n \sum_{\bar{x}\in\{0,1\}^{n-1}} \prod_{i\in[n-1]} p_i^{\bar{x}_i}(1-p_i)^{1-\bar{x}_i} \\
&= p_n \sum_{\bar{x}\in\{0,1\}^{n-1}} \pi(\bar{p},\bar{x}) \\
&= p_n.
\end{aligned}$$

The last equality is due to the first identity applied to the case of $n-1$. □

Algorithm 3 Generic Bernoulli Sampling

function RANDSAMPLESTEP(p, k)
  for $l \in [k], i \in [n]$ **do**
    $\bar{X}_{l,i} \leftarrow \text{Bernoulli}(p_i)$
  end for
  return $\bar{X} \in \{0,1\}^{k \times n}$
end function

We note that this equivalence is mathematically straightforward. Nevertheless, it provides the following useful equivalence

$$\min_{x \in [0,1]^n} f(x) \Longleftrightarrow \min_{p \in [0,1]^n} f(x_p),$$

thereby offering two equivalent interpretations of the first-order optimization process: it can be viewed either as iteratively refining a fractional relaxation of the binary solution, or as successively updating a product distribution over $\{0,1\}^n$ with respect to the objective f. Furthermore, if f guarantees that an optimal solution must be binary, then the optimal distribution is indeed fully independent and coincide with the optimal solution x^*. Particularly, this interpretation enables the generic Bernoulli sampling subroutine presented in Algorithm 3.

2.3 Sampling Algorithm Analysis

This subsection derives probability bounds that establish the optimality and feasibility guarantees of Algorithm 3, offering insight into the behavior of GFORS.

Theorem 3. *Let $p \in [0,1]^n$ and $k \in \mathbb{N}$ in Algorithm 3. Let f and $x^* \in \{0,1\}^n$ be the objective function and an optimizer of (1), and fix an optimality tolerance $\delta > 0$. Set $\mu := \mathbb{E}_{x \sim p}[\|x - x^*\|_1]$ as the expected 1-norm distance from x^*, and let L_f be the Lipschitz constant of f under 1-norm. Then, given $\mu \leq \delta / L_f$, the following bound holds,*

$$\psi(p,k) := \Pr\big(\exists\, l \in [k] \mid f(\bar{X}_l) \leq f(x^*) + \delta\big) \;\geq\; 1 - \exp\Big(k\left(\tfrac{\delta}{L_f}\big(1 - \log \tfrac{\delta}{\mu L_f}\big) - \mu\right)\Big).$$

Proof. For any random vector $x \in \{0,1\}^n$ follows p, $\|x - x^*\|_1 \leq \delta / L_f$ ensures $f(x) \leq f(x^*) + \delta$. Thus, it suffices to show

$$\Pr(\|x - x^*\|_1 > \delta / L_f) \leq \exp\left(\tfrac{\delta}{L_f}\big(1 - \log \tfrac{\delta}{\mu L_f}\big) - \mu\right).$$

Since x is entrywise Bernoulli, the distance $D := \|x - x^*\|_1 = \sum_{i \in [n]} \mathbb{1}(x_i \neq x_i^*)$ follows the Poisson binomial distribution, which adopts the following Chernoff right-tail bound when $\mu \leq t$

$$\Pr(D > t) \leq \exp\left(t(1 - \log \frac{t}{\mu}) - \mu\right).$$

This proves the bound of $\Pr(\|x - x^*\|_1 > \delta / L_f)$, and the claim follows from the sample size k. □

Since $\psi(p,k)$ increases as μ decreases, an intuitive implication is that when p is closer to a binary solution x^*, the likelihood of obtaining a sample with a comparable objective value increases. This observation further suggests two points: (i) the binary penalty term in (2) heuristically increases

the probability of generating candidate solutions with good objective values, and (ii) the relaxation strength of (1) remains important in GFORS, since the optimal solution converges to the global fractional optimum when Q is psd and $\rho = 0$ according to Theorem 1.

The next theorem provides the probability bound for feasibility guarantee. Without loss of generality, we assume that only inequality constraints $Ax \geq b$ are present in (1).

Theorem 4. *Given $p \in [0,1]^n$ and $k \in \mathbb{N}$ in Algorithm 3, the feasibility guarantee is bounded by*

$$\phi(p,k) := \Pr\left(\exists l \in [k] \mid A\bar{X}_l \geq b\right) \geq 1 - \exp\left(-2k\left[\frac{\gamma_+^2(p)}{\eta^2} - \frac{\log m}{2}\right]_+\right),$$

where $A \in \mathbb{R}^{m\times n}$, $\gamma(p) = \min_{j\in[m]}\{\langle A_j, p\rangle - b_j\}$ is the minimum slack, $\gamma_+(p) := \max\{\gamma(p), 0\}$, and $\eta = \max_{j\in[m]} \|A_j\|_2$ denotes the maximum row-wise Euclidean norm of A.

Proof. Let $\gamma^j(p) := \langle A_j, p\rangle - b_j$ and $\gamma_+^j(p) := \max\{\gamma^j(p), 0\}$. For each random binary vector x that follows p, the following holds due to the union bound.

$$\Pr(Ax \not\geq b) = \Pr\left(\exists j \in [m] \mid \langle A_j, x\rangle < b_j\right) \leq \sum_{j\in[m]} \Pr\left(\langle A_j, x\rangle < b_j\right).$$

For each $j \in [m]$, let $s > 0$ and $\eta_j = \|A_j\|_2$, we compute

$$\begin{aligned}
\Pr(\langle A_j, x\rangle < b_j) &= \Pr\left(\mathbb{E}[\langle A_j, x\rangle] - \langle A_j, x\rangle > \mathbb{E}[\langle A_j, x\rangle] - b_j\right) \\
&= \Pr\left(\langle A_j, p - x\rangle > \gamma^j(p)\right) \\
&\leq \exp(-s\gamma^j(p))\mathbb{E}[\exp(s\langle A_j, p - x\rangle)] \\
&= \exp(-s\gamma^j(p)) \prod_{i\in[n]} \mathbb{E}[\exp(-sA_{ji}(x_i - p_i))] \\
&\leq \exp(-s\gamma^j(p)) \prod_{i\in[n]} \exp\left(s^2A_{ji}^2/8\right) \\
&= \exp\left(s^2\eta_j^2/8 - s\gamma^j(p)\right),
\end{aligned}$$

where the first inequality is due to the Chernoff bound $\Pr(X \geq a) \leq \exp(-sa)\mathbb{E}[\exp(sX)]$ for every $s > 0$, and the second is an application of Hoeffding's lemma. To obtain the tightest bound, we minimize the last expression within domain $s \geq 0$ to obtain $s^* = 4\gamma_+^j(p)/\eta_j^2$. This provides the bound $\Pr(\langle A_j, x\rangle < b_j) \leq \exp(-2[\gamma_+^j(p)]^2/\eta_j^2)$ for every j. Then, we have

$$\Pr(Ax \not\geq b) \leq m\exp(-2\gamma_+^2(p)/\eta^2) = \exp\left(-2\gamma_+^2(p)/\eta^2 + \log m\right) = \exp\left(-2(\gamma_+^2(p)/\eta^2 - \log m/2)\right).$$

Note that when $\gamma_+^2(p)/\eta^2 < \log m/2$, this bound becomes trivial. Thus, we can tighten it as $\exp(-2[\gamma_+^2(p)/\eta^2 - \log m/2]_+)$, which implies the claim with k independent samples. □

One implication of this theorem is that a larger minimum slack in the fractional solution p increases the likelihood of producing feasible binary samples. On the other hand, when the feasible region $\{x \in [0,1]^n \mid Ax \geq b\}$ occupies only a small portion of the hypercube, the guarantee of feasibility becomes weaker. In particular, under equality constraints (1c), the probability of obtaining feasible samples can be extremely small. To address this sampling pitfall, we will develop three methods in the next subsection.

2.4 Sampling with Equality Constraints

We introduce three mechanisms to enhance the feasibility guarantees of sampling under equality constraints (1c). Each mechanism has distinct strengths and is suited to different settings.

2.4.1 Totally Unimodular (TU) Reformulation

A matrix is totally unimodular if every square submatrix has determinant in $\{-1, 0, 1\}$. Many binary integer programs adopt TU substructures [51, 57, 60]. In (1), suppose a TU submatrix B_J is identified with J as the row index set, the following reformulation is exact and can remove all equality constraints associated with J.

Theorem 5. *In (1), given row and column index sets J and I such that B_J and d_J are integral, B_J is TU, and B_{JI} is invertible, define $s := B_{JI}^{-1} d_J$, $S := -B_{JI}^{-1} B_{J\bar{I}}$, and let $\bar{J}$ and $\bar{I}$ denotes the complement indices in the rows and columns of B, respectively. The following reformulation of (1) is exact,*

$$\begin{aligned} \min_{x_{\bar{I}} \in \{0,1\}^{n-|I|}} \quad & \langle x_{\bar{I}}, Q' x_{\bar{I}} \rangle + \langle c', x_{\bar{I}} \rangle + c'_0 \\ & A' x_{\bar{I}} \geq b' \\ & B' x_{\bar{I}} = d' \\ & S x_{\bar{I}} \geq -s \\ & S x_{\bar{I}} \leq 1 - s, \end{aligned}$$

where the model parameters $(Q', A', B', b', c', d', c'_0)$ are defined as

$$\begin{aligned} & Q' := S^\intercal Q_{II} S + S^\intercal Q_{I\bar{I}} + Q_{I\bar{I}}^\intercal S + Q_{\bar{I}\bar{I}};\ A' := A_{\cdot I} S + A_{\cdot \bar{I}};\ B' := B_{\bar{J}I} S + B_{\bar{J}\bar{I}};\ b' := b - A_I s; \\ & c' = 2 S^\intercal Q_{II} s + 2 Q_{I\bar{I}}^\intercal s + S^\intercal c_I + c_{\bar{I}};\ d' := d_{\bar{J}} - B_{\bar{J}I} s;\ c'_0 := \langle s, Q_{II} s \rangle + \langle c_I, s \rangle. \end{aligned}$$

Proof. By splitting matrices and vectors by the index sets I and replacing $x_I = s + S x_{\bar{I}}$ into (1), we obtain the above reformulation with computed new model parameters along with the last two constraints to ensure $x_I \in [0, 1]^n$. Compared with the original problem, the only missing constraints is the binary requirement on x_I, which is guaranteed since B_J is a TU matrix. □

Remark 1. In this reformulation, the matrices Q', A', and B' often lose sparsity because of the inverse in S, leading to a substantial memory burden. In the GPU implementation of the TUReformulate subroutine in Algorithm 1, we instead store the LU decomposition of B_{JI} and use it to apply any multiplication of the form $B_{JI}^{-1} x$ efficiently during the gradient update steps.

This reformulation is particularly effective when the entire matrix B is totally unimodular (see Section 4.2.5). Otherwise, the remaining equality constraints still make feasibility difficult to achieve through sampling. To address more general cases, we introduce two additional methods.

2.4.2 Customized Sampling

A customized sampling routine refers to any realization of RandSampleStep in Algorithm 1 that, given the current fractional solution p and a sample size k, returns the corresponding candidate

Algorithm 4 Customized Sampling for 3D Assignment

Parameters: entry size multiple γ; improvement steps L
function RandSampleStep(p, k)
  $p_{\gamma n} \leftarrow$ first $\gamma \cdot n$ largest entries in p
  $x_{\text{partial}} \leftarrow$ partial non-conflict assignment according to the sorted $p_{\gamma n}$
  for $l \in [k]$ **do**
    $\bar{X}_l \leftarrow$ random completion of x_{partial}
    $\bar{X}_l \leftarrow$ local improvement of $\bar{X}_l$ with L steps
  end for
  return $\bar{X} \in \{0,1\}^{k \times n}$
end function

solutions. An additional requirement is that this routine must be GPU-compatible. We illustrate through the example of *3D assignment problem*, formulated as follows.

$$\text{(3D Assignment)} \quad \min_{x \in \{0,1\}^{n^3}} \langle c, x \rangle \tag{21a}$$

$$\text{s.t.} \sum_{(j,k) \in [n]^2} x_{ijk} = 1, \quad \forall i \in [n], \tag{21b}$$

$$\sum_{(i,k) \in [n]^2} x_{ijk} = 1, \quad \forall j \in [n], \tag{21c}$$

$$\sum_{(i,j) \in [n]^2} x_{ijk} = 1, \quad \forall k \in [n]. \tag{21d}$$

We can still apply the TU reformulation to eliminate the first two sets of constraints, but n equality constraints remain. Algorithm 4 introduces a customized sampling subroutine that guarantees feasibility. The main idea is to first construct a feasible 3D assignment from p, and then apply parallelized local improvements using feasibility-preserving *pairwise interchange* operations [5]. We construct a partial solution from the largest γn entries of p, which both reduces sorting cost and retains the most informative entries for candidate generation. This design enables a significantly more GPU-efficient implementation. The performance of this customized sampling subroutine will be presented in Section 4.2.4.

A similar idea applies to other customized sampling designs: first generate a feasible candidate from the fractional solution p, then perform parallel local searches to explore additional candidates.

2.4.3 Monotone Relaxation

It is well-known that when the objective function f is monotone, the binary solution space can be relaxed to its upper closure, i.e., all supersets of feasible solutions (for example, all s–t connected subgraphs in the shortest path problem). Such relaxations often allow equality constraints to be replaced by inequalities.

For instance, if c is nonnegative or nonpositive in the 3D assignment formulation, all equalities can be relaxed to inequalities without changing the optimal solution (the same solution remains optimal under the relaxation). Our sampling algorithm can then operate on this relaxed formulation

to generate candidate supersets of true assignment solutions, followed by a customized repair step to recover a valid assignment.

This approach extends naturally to the case where f is bimonotone, i.e., it becomes monotone after flipping certain entries of x.

3 Implementation Details

This section provides the implementation details of the remaining subroutines in Algorithm 1. Apart from the straightforward EvalBest, which performs feasibility testing and optimality comparison via matrix multiplications, the others include: Preprocess, which normalizes the input parameters to handle heterogeneous problem instances; UpdatePenalty, which gradually increases the binary gap penalty multiplier ρ; and CheckHalt, which enforces the stopping criteria.

Preprocess. This subroutine normalizes the model parameters (Q, K, c, r), which may include those produced by the TUReformulate subroutine. Such normalization is a standard procedure in many mathematical programming solvers to ensure consistent algorithmic performance across distinct instances [26, 31, 61]. Specifically, the following steps are performed in this function:

- Normalize each row in K by its 2-norm and update r accordingly.
- Normalize Q and c by $\|Q\|_2 + \|c\|_2$ (i.e., spectral norm of Q plus 2-norm of c).
- Normalize K and r by the spectral norm of K.

The purpose of the last step is to reduce the PDHG convergence condition from $\tau_1\tau_2\|K\|_2^2 \leq 1$ to the simplified form $\tau_1\tau_2 \leq 1$. In our implementation, we further introduce the tuning parameter $\sigma \in (0, 1)$ to control the exploration aggressiveness via $\tau_1\tau_2 \leq \sigma$.

UpdatePenalty. The strength of the binary penalty parameter ρ dictates how strongly the current iterate x_k is attracted toward binary solutions. Intuitively, we initialize ρ with a small value so that the central path first follows the relaxation of (1), allowing x_k to move toward a global optimum when Q is positive semidefinite. Subsequently, ρ is gradually increased to encourage convergence to binary solutions in later stages, thereby boosting the probability of obtaining a high-quality feasible solution as suggested by Theorem 3. With tunable parameters $(\rho_{\min}, \rho_{\max}, T, p, \delta)$, we employ the following heuristic update:

$$\tilde{\rho}_n = \rho_{\min}\left(1 + \tfrac{n}{T}\right)^p,$$
$$\rho_n = \mathrm{clip}(\tilde{\rho}_n, \rho_{n-1} + \delta, \rho_{\max}),$$

where $\rho_{\min}$ and $\rho_{\max}$ set the lower and upper bounds of the penalty, T and p control the growth rate, and δ enforces a minimum increment pace. Here, n is a counter that increases only after designated iterations. Overall, this update strategy balances early-stage exploration guided by the continuous relaxation with late-stage exploitation that strengthens the pull toward integrality, thereby balancing global search and binary feasibility.

CheckHalt. Since solution generation primarily relies on the sampling subroutine, the stopping criterion is designed to detect whether further improvement is unlikely, thereby enabling potential early termination. With the aid of a value-stalling detector, this subroutine returns `true` once all three indicators—the primal feasibility gap, the dual feasibility gap, and the binary gap—are either satisfied within tolerance or have remained stalled for a sufficient number of iterations.

4 Numerical Experiments

In this section, we evaluate the GFORS framework on a collection of classic binary integer programs, comparing its performance against the baseline solver Gurobi. We first present an overview of aggregated computational performance, followed by problem-specific analyses. Since integer programs often exhibit heterogeneous performance patterns, these detailed results provide further insight into GFORS and the its individual components introduced in Sections 2 and 3.

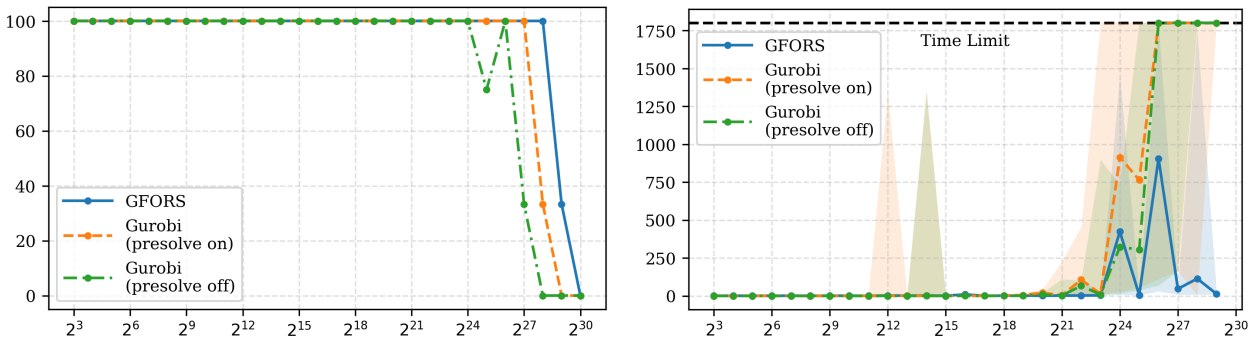
Experiment Setup. All experiments were conducted on a Slurm-managed HPC cluster (Slurm 24.05) running Red Hat Enterprise Linux 9.4 with kernel 5.14.0. Each job was allocated 16 CPU cores from an Intel Xeon Gold 6348Y processor, one NVIDIA L40S GPU (48 GB, CUDA 12.5, driver 555.42.02), and 96 GB RAM, matching the standard memory ratio used in the literature [40]. We developed GFORS in Python 3.12.11 with PyTorch 2.5.1 for GPU acceleration, and used Gurobi 12.0.3 as the baseline BIP solver. Each instance was run with a wall-clock time limit of 1,800 seconds.

Problem Instances. We evaluate our method on six problem classes: set cover, knapsack, max cut, 3D assignment, facility location, and the traveling salesman problem (TSP). The first five are classical binary integer programs with compact formulations. For TSP, we employ a binary-integer encoding of the Miller–Tucker–Zemlin formulation [44, 58], whose linear program relaxation is relatively weak [36]. We include this case primarily as a stress test, since it does not align as naturally with our framework as the other problem classes. For each class, we scale the instance size by powers of two until the number of nonzeros (NNZ) in the model parameters reaches 2×10^9. At each size, we generate five random instances, yielding 500 test instances in total. This exponential scaling systematically evaluates the scalability of the GFORS framework.

4.1 Aggregated Results Overview

Since the instances span multiple problem classes, we use the total NNZ in the model parameters (Q, K) to define instance size, and aggregate computational performance into 28 groups based on $\lfloor \log_2(\text{nnz}) \rfloor$. Because the problem classes differ in structure, the number of instances per group is not perfectly uniform. With the exception of the smallest group (2^3) containing only five instances, each NNZ group includes between 10 and 20 instances drawn from multiple classes. To ensure fair comparison, we exclude the TSP class from the aggregated results, as it differs fundamentally from the other five problem classes (lacking a natural compact BIP formulation), and GFORS solves only the smallest TSP instances, which would otherwise skew the aggregation result. A detailed analysis of the TSP class is provided in Section 4.2.6.

Though our GFORS implementation allows for many tunable parameters, in this experiment we adjust only four settings: the step-size parameter $\sigma \in (0, 1)$, which controls exploration aggressiveness;



(a) Solved percentage (%) vs NNZ

(b) Time-to-target (sec.) vs NNZ

Figure 1: Performance vs. NNZ. Solved percentages and time-to-target values are aggregated by $\lfloor \log_2(\text{nnz}) \rfloor$. Because Gurobi may spend long runtimes refining incumbents while already having strong early solutions, we adopt time-to-target as a fair measure: for GFORS, this is the time to its best solution before halting; for Gurobi (with/without presolve), it is the first time a solution improves upon the GFORS optimum, or 1,800 seconds if none is found.

the sampling batch size k_b, which can be used to limit peak GPU memory usage; the TU reformulation, applied when row and column index sets J and I are given (see Section 2.4.1); and customized sampling subroutine if provided.

We benchmark the best-performing GFORS configurations against the default settings of Gurobi 12.0.3, with presolve enabled and disabled. Because GFORS is not an exact solver but is designed to identify high-quality solutions for large instances, we evaluate performance using two complementary metrics: (i) *solved percentage*, the fraction of instances in each NNZ group for which a feasible solution is obtained within the time limit; and (ii) *time-to-target*, which provides a fair basis of comparison since Gurobi may continue refining incumbents long after producing strong early solutions. For GFORS, time-to-target is the time to its best solution before halting; for Gurobi, it is the time to first improve upon the GFORS optimum, or the time limit if no such improvement is achieved. Metric (i) captures each algorithm's ability to find feasible solutions, while (ii) measures the time required for Gurobi to surpass GFORS, thereby indicating whether GFORS can effectively fulfill its purpose of producing high-quality feasible solutions, especially on large-scale instances.

The aggregated results are presented in Figure 1, where the x-axis shows NNZ on a logarithmic scale, and the two subfigures report solved percentage and time-to-target, respectively. Figure 1a shows that GFORS maintains competitive solved percentages across NNZ groups, with performance degrading only at the largest sizes where all methods struggle. In contrast, Gurobi without presolve fails to solve any instances at scale 2^{28}, and Gurobi with presolve begins to show a similar decline at scale 2^{29}. On these large instances, presolve often enables Gurobi to obtain an initial feasible solution, but typically no further improvement is achieved over the runtime.

Figure 1b reports only those problem instances for which at least one algorithm obtained a feasible solution. The plot shows median time-to-target values as line curves, with shaded bands indicating the first and third quartiles. Both Gurobi variants consistently outperform GFORS on small- to medium-sized instances, although GFORS maintains lower overall computational time. In the range 2^{11} to 2^{15}, Gurobi exhibits noticeable time-to-target variations on certain instances, reflected in the large third-quartile values. For large-scale instances, however, GFORS consistently finds high-quality solutions quickly, whereas Gurobi requires substantially more time to surpass and

frequently reaches the time limit without improvement. These results highlight GFORS's robustness and efficiency on large-scale instances relative to Gurobi's default configurations.

4.2 Problem-Specific Analysis

This section analyzes results by problem class. We take Gurobi (presolve enabled) as the baseline, as it matches or exceeds the presolve-off variant on most instances. Instance inputs are split into (i) size parameters that determine model scale and (ii) randomization parameters (e.g., edge weights, item costs). For each size configuration, tables report the average best objective and the associated average runtime over five random instances. To isolate core solver performance, we exclude model-construction time (i.e., PREPROCESS in Algorithm 1) for all methods, since parsing/instantiation costs are interface-dependent, orthogonal to algorithmic behavior, and typically small relative to runtime. TU reformulation time is reported separately wherever applicable.

4.2.1 Set Cover: Effect of the Step-Size Parameter σ

The set cover problem seeks the minimum-cost collection of subsets whose union covers the entire ground set, formulated as

$$\min_{x\in\{0,1\}^n} \left\{ \langle c, x\rangle \;\middle|\; \sum_{i\in S_j} x_i \geq 1,\ \forall j \in [m] \right\}.$$

The test instances range from $(m, n) = (5, 10)$ up to (81,920, 163,840), doubling both m and n at each step. The cost vector c and sets S_j are randomly generated. We evaluate three GFORS configurations with $\sigma \in \{0.3, 0.5, 0.99\}$, where σ controls the step-size aggressiveness through $\tau_1\tau_2 \leq \sigma$. Detailed results are reported in Table 2.

Compared with Gurobi (GRB), the $\sigma = 0.99$ variant increasingly outperforms as instance size grows: beyond NNZ $\approx 10^7$ it attains substantially better objectives in much shorter time, while

Config	NNZ	Avg. Objective Value				Avg. Runtime (sec.)			
(m, n)		GRB	$\sigma = 0.99$	$\sigma = 0.5$	$\sigma = 0.3$	GRB	$\sigma = 0.99$	$\sigma = 0.5$	$\sigma = 0.3$
$(5, 10)$	1.74e1	7.8	7.8	7.8	7.8	0.00	1.61	1.89	5.37
$(10, 20)$	3.30e1	15.6	15.6	15.6	15.6	0.00	1.68	2.15	2.33
$(20, 40)$	6.82e1	35.0	35.2	35.2	35.2	0.00	2.62	4.96	5.34
$(40, 80)$	1.40e2	57.0	57.2	57.2	57.2	0.00	3.16	3.13	2.40
$(80, 160)$	4.55e2	73.2	73.8	73.8	74.2	0.01	3.29	5.65	9.09
$(160, 320)$	1.82e3	78.8	81.8	83.2	83.8	0.04	6.56	8.06	10.79
$(320, 640)$	7.04e3	77.4	86.4	89.2	91.6	1.14	7.42	9.17	11.16
$(640, 1280)$	2.86e4	72.4	97.6	101.6	108.8	160.47	9.02	11.76	13.34
$(1280, 2560)$	1.15e5	73.6	170.4	199.2	218.2	1800.05	12.93	16.50	19.85
$(2560, 5120)$	4.63e5	82.6	347.2	289.4	394.8	1800.06	24.09	8.64	2.41
$(5120, 10240)$	1.84e6	101.0	473.2	818.2	373.2	1800.18	2.41	2.42	5.40
$(10240, 20480)$	7.36e6	120.8	403.8	369.0	244.0	1800.64	2.50	64.54	469.06
$(20480, 40960)$	2.94e7	996.2	304.2	275.0	274.2	1800.88	246.46	1209.28	1800.00
$(40960, 81920)$	1.17e8	1120.6	645.2	1582.8	2871.2	1803.75	1800.00	1800.00	1800.00
$(81920, 163840)$	4.69e8	1271.2	4831.8	10381.2	18457.6	1813.50	1800.01	1800.01	1800.01
$(163840, 327680)$	1.88e9	–	–	–	–	–	–	–	–

Table 2: Set Cover: objective values and runtimes. The parameter σ controls the first-order step-size schedule in GFORS; "–" denotes out-of-memory.

GRB typically times out. The sole exception at $(m, n) = (81{,}920, 163{,}840)$ occurs because GRB's presolve produced a strong incumbent, even though the subsequent branch-and-bound made no progress, yielding an outlier objective despite the timeout. Across GFORS settings, on medium- to large-sized instances smaller σ values (e.g., $0.3, 0.5$) generally achieve better objectives, whereas on the largest instances larger σ (e.g., 0.99) is preferable due to its greater search efficiency (large step size) under the time limit.

4.2.2 Knapsack: Memory Control via Batch Size

The knapsack problem searches for the subset of items with maximum total value subject to a weight capacity constraint, formulated as

$$\max_{x \in \{0,1\}^n} \{\langle v, x\rangle \,|\, \langle w, x\rangle \leq W\}\,.$$

In our experiments, the number of items n scales from 10 up to 1.34×10^9, doubling at each step. Item values v and weights w are randomly generated, with W fixed at half of the total weight. Detailed results are reported in Table 3.

Across nearly all instances, both algorithms yield comparable optimal objective values (with GRB often better in later digits), but GFORS demonstrates better efficiency on large instances. Notably, GRB crashes due to out-of-memory when NNZ $\geq 3.36 \times 10^8$, whereas GFORS still produces

Config	NNZ	Avg. Objective Value		Avg. Runtime (sec.)		GPU Mem (MB)
n		GRB	GFORS	GRB	GFORS	
10	1.00e1	3.80e1	3.48e1	0.00	2.40	31.94
20	2.00e1	9.02e1	8.96e1	0.00	2.30	31.92
40	4.00e1	1.67e2	1.66e2	0.00	2.53	32.11
80	8.00e1	3.37e2	3.37e2	0.00	2.34	31.92
160	1.60e2	6.89e2	6.88e2	0.00	2.13	32.93
320	3.20e2	1.37e3	1.37e3	0.00	1.72	31.75
640	6.40e2	2.79e3	2.79e3	0.00	1.52	31.17
1,280	1.28e3	5.48e3	5.48e3	0.00	1.51	31.21
2,560	2.56e3	1.10e4	1.10e4	0.01	1.52	31.28
5,120	5.12e3	2.20e4	2.20e4	0.01	2.31	33.81
10,240	1.02e4	4.41e4	4.41e4	0.01	1.72	32.31
20,480	2.05e4	8.81e4	8.80e4	0.02	1.52	34.30
40,960	4.10e4	1.77e5	1.77e5	0.04	1.91	37.87
81,920	8.19e4	3.54e5	3.54e5	0.13	1.92	44.21
163,840	1.64e5	7.08e5	7.08e5	0.21	1.74	52.70
327,680	3.28e5	1.42e6	1.42e6	0.37	2.54	72.86
655,360	6.55e5	2.83e6	2.83e6	0.76	2.14	122.42
1,310,720	1.31e6	5.66e6	5.66e6	1.72	1.93	228.52
2,621,440	2.62e6	1.13e7	1.13e7	3.72	2.16	403.72
5,242,880	5.24e6	2.26e7	2.26e7	7.40	2.68	794.49
10,485,760	1.05e7	4.53e7	4.53e7	15.19	4.11	1575.73
20,971,520	2.10e7	9.06e7	9.06e7	30.87	10.99	3138.33
41,943,040	4.19e7	1.81e8	1.81e8	63.85	24.68	6264.92
83,886,080	8.39e7	3.62e8	3.62e8	129.41	30.52	12641.14
167,772,160	1.68e8	7.25e8	7.25e8	258.54	81.61	25144.33
335,544,320	3.36e8	–	1.45e9	–	121.61	50001.20
671,088,640	6.71e8	–	–	–	–	–
1,342,177,280	1.34e9	–	–	–	–	–

Table 3: Knapsack: objective values and runtimes. *GPU Mem* reports peak GPU memory usage during computation; "–" indicates out-of-memory.

solutions within 122 seconds at this size. A distinguishing feature of the knapsack problem is its extremely high-dimensional decision space, which incurs significant memory costs for solution sampling. This issue can be alleviated by setting the batch size $k_b = 1$ in Algorithm 1, enabling GFORS to handle very large instances. The GPU memory usage is reported in the final column.

4.2.3 Max Cut: Unconstrained QP vs Linearized BIP

Given a connected network $G = (V, E)$ with signed edge weights, the max cut problem seeks a bipartition of V whose induced cut maximizes the total weight of edges crossing the partition. This problem can be expressed directly as a quadratic program (QP) or reformulated as an integer program (IP) using the McCormick envelope:

$$\text{(QP):} \quad \max_{x\in\{0,1\}^V} \sum_{(i,j)\in E} w_{ij}\,(x_i + x_j - 2x_i x_j) \qquad \text{(IP):} \quad \max_{\substack{x\in\{0,1\}^V \\ y\in\{0,1\}^E}} \sum_{(i,j)\in E} w_{ij}\,(x_i + x_j - 2y_{ij})$$

$$\begin{aligned} \text{s.t. } & y_{ij} \le x_i, \quad \forall (i,j) \in E, \\ & y_{ij} \le x_j, \quad \forall (i,j) \in E, \\ & y_{ij} \ge x_i + x_j - 1, \quad \forall (i,j) \in E. \end{aligned}$$

In our experiments, the number of vertices $n := |V|$ ranges from 10 to 20,480. Graph topologies are generated randomly with density fixed at 0.5, and edge weights w are sampled uniformly from $[-8, 10]$. Results are reported in Table 4.

Both GRB and GFORS solve the QP formulation, while GFORS (IP) applies to the IP reformulation. Across all instances, GFORS achieves solutions very close to those of GRB but with significantly shorter runtimes on medium and large instances. By contrast, GFORS (IP) performs much worse, leaving several instances unsolved. These results highlight that relaxation tightness and reformulation choice remain critical factors within the GFORS framework, strongly influencing overall performance.

Config	NNZ	Avg. Objective Value			Avg. Runtime (sec.)		
n		GRB	GFORS	GFORS (IP)	GRB	GFORS	GFORS (IP)
10	4.48e1	4.12e1	4.08e1	9.40e0	0.05	1.31	14.02
20	1.78e2	1.70e2	1.66e2	4.58e1	0.10	1.45	519.20
40	7.41e2	5.11e2	4.73e2	1.65e2	0.75	1.45	210.61
80	3.01e3	1.86e3	1.47e3	2.88e2	950.72	1.72	347.01
160	1.20e4	6.07e3	3.86e3	$-\infty$	1800.05	2.01	1800.00
320	4.82e4	2.10e4	1.41e4	2.92e2	1800.02	2.02	1800.00
640	1.94e5	7.46e4	5.38e4	1.34e3	1800.13	2.02	8.47
1,280	7.76e5	2.72e5	2.10e5	$-\infty$	1800.30	2.02	1800.00
2,560	3.10e6	1.01e6	8.32e5	5.28e3	1800.60	2.02	27.29
5,120	1.24e7	3.82e6	3.30e6	8.56e3	1803.15	2.11	112.92
10,240	4.97e7	1.46e7	1.31e7	$-\infty$	1804.05	3.15	1800.00
20,480	1.99e8	5.66e7	5.25e7	–	1817.99	7.89	–

Table 4: Max Cut: objective values and runtimes. *GRB* and *GFORS* solve the QP formulation, while *GFORS (IP)* solves the IP formulation; "–" denotes out-of-memory.

Config	NNZ	Avg. Objective Value			Avg. Runtime (sec.)		
n		GRB	Cust.	Def.	GRB	Cust.	Def.
2	1.60e1	4.6	4.6	4.6	0.00	4.38	2.33
4	1.60e2	6.2	6.4	8.0	0.00	6.02	2.83
8	1.41e3	8.0	9.2	∞	0.01	11.64	1800.00
16	1.18e4	16.0	18.8	∞	0.23	9.49	1800.00
32	9.63e4	32.0	35.8	∞	1.12	22.61	1800.00
64	7.78e5	64.0	73.4	∞	11.51	17.93	1800.00
128	6.26e6	128.0	133.4	∞	109.78	53.61	1800.00
256	5.02e7	256.0	259.0	∞	784.41	252.21	1800.00
512	4.02e8	–	513.6	∞	–	80.95	1800.00

Table 5: 3D Assignment: objective values and runtimes. *Cust.* and *Def.* denote GFORS with customized and default sampling, respectively; "–" indicates out-of-memory.

4.2.4 3D Assignment: Customized Sampling

The 3D assignment problem generalizes the classic 2D assignment to a three-dimensional setting and is formulated as (21). Even with the TU reformulation that eliminates the first two sets of constraints, n equality constraints remain, limiting sampling efficiency. To address this, we implement the proposed GPU-friendly customized sampling procedure (Algorithm 4), with results summarized in Table 5.

Columns *Cust.* and *Def.* report results for customized and default sampling, respectively. The customized scheme yields a substantial performance gain in both objective quality and runtime. Without it, most instances terminate with no feasible solution except for the smallest cases. In contrast, with customized sampling, all instances are solved with objective values comparable to GRB, while requiring significantly less time on large instances. Furthermore, the largest instances cannot be solved by GRB due to out-of-memory errors, whereas GFORS with customized sampling efficiently produces high-quality solutions. These results demonstrate that ad-hoc sampling subroutines can significantly enhance the performance of GFORS.

4.2.5 Facility Location: TU Reformulation

This version of the facility location problem seeks the optimal set of facility openings and customer assignments that minimize total cost, formulated as follows.

$$\begin{aligned} \min_{x\in\{0,1\}^{n_f},\, y\in\{0,1\}^{n_f\times n_c}} \quad & \langle f, x\rangle + \langle c, y\rangle \\ \text{s.t.} \quad & \sum_{i\in F} y_{ij} = 1, \quad \forall j \in C, \\ & y_{ij} \le x_i, \quad \forall i \in F,\ j \in C. \end{aligned}$$

The size parameters (n_f, n_c) range from $(2, 8)$ up to $(8192, 32768)$, with opening costs f and service costs c generated randomly. Results are reported in Table 5.

Columns *TU* and *No TU* correspond to GFORS implementations with and without the TU reformulation step (see Section 2.4.1). With a short execution time (see *Avg. TU Time*), the TU reformulation substantially improves GFORS performance: all but the largest configuration are solved within one minute. In terms of solution quality, GRB dominates on small- and medium-sized instances, but GFORS (TU) begins to surpass GRB beyond configuration $(512, 2048)$. These results

Config (n_f, n_c)	NNZ	Avg. Objective Value			Avg. Runtime (sec.)			Avg. TU Time
		GRB	TU	No TU	GRB	TU	No TU	
$(2, 8)$	$4.80e1$	61.4	61.4	61.4	0.00	1.63	2.34	0.00
$(4, 16)$	$1.92e2$	90.8	94.4	∞	0.00	3.46	1800.00	0.00
$(8, 32)$	$7.68e2$	133.6	141.0	∞	0.01	12.02	1800.00	0.00
$(16, 64)$	$3.07e3$	188.4	223.8	∞	0.09	13.21	1800.00	0.00
$(32, 128)$	$1.23e4$	296.0	827.8	∞	1.36	2.73	1800.00	0.00
$(64, 256)$	$4.92e4$	458.6	1561.2	∞	28.27	1.96	1800.00	0.00
$(128, 512)$	$1.97e5$	766.4	3096.8	∞	1800.14	1.76	1800.00	0.01
$(256, 1024)$	$7.86e5$	1346.2	6135.8	∞	1800.24	1.86	1800.00	0.04
$(512, 2048)$	$3.15e6$	2486.4	12260.8	∞	1800.52	2.35	1800.00	0.17
$(1024, 4096)$	$1.26e7$	40321.4	24586.2	∞	1801.58	4.11	1800.00	0.90
$(2048, 8192)$	$5.03e7$	80296.8	49157.4	∞	1806.61	13.85	1800.00	6.00
$(4096, 16384)$	$2.01e8$	160808.4	98476.7	∞	1827.85	51.38	1800.00	31.56
$(8192, 32768)$	$8.05e8$	–	–	–	–	–	–	–

Table 6: Facility Location: objective values and runtimes. *TU* and *No TU* denote the implementations of GFORS without or with TU reformulation, respectively; *Avg. TU Time* denotes the mean wall-clock time spent on the TU reformulation; "–" indicates out-of-memory. TU reformulation significantly improved the GFORS performance.

highlight that TU reformulation can greatly enhance GFORS's performance when it eliminates all inequality constraints.

4.2.6 TSP: Challenges from Weak Relaxation & Sparse Feasible Set

Given a set of cities $\mathcal{N} = \{1, 2, \ldots, n\}$ with pairwise distances c, TSP seeks the shortest Hamiltonian cycle that starts at the depot 1 and visits every city exactly once. Since the GFORS framework does not support formulations with an exponential number of constraints, the classical Dantzig–Fulkerson–Johnson (DFJ) formulation [16] is not applicable. Instead, we adopt the Miller–Tucker–Zemlin (MTZ) formulation [44], which is known to yield weaker relaxations than DFJ. We use $\mathcal{N}_k := \{k, k+1, \ldots, n\}$ to denote the subset of cities starting at index k:

$$
\begin{aligned}
\min_{x \in \{0,1\}^{n^2}} \quad & \langle c, x \rangle \\
\text{s.t.} \quad & \sum_{j \neq i} x_{ij} = 1, && \forall i \in \mathcal{N}, \\
& \sum_{i \neq j} x_{ij} = 1, && \forall j \in \mathcal{N}, \\
& u_i - u_j + n x_{ij} \leq n - 1, && \forall i, j \in \mathcal{N}_2,\ i \neq j, \\
& 2 \leq u_i \leq n, && \forall i \in \mathcal{N}_2.
\end{aligned}
$$

Since this is not a pure BIP due to the integer variables u, we apply the unary (sequential) encoding technique, motivated by its favorable performance on Ising machines [58]. Specifically, we introduce binary variables y_{ik} for $i \in \mathcal{N}_2$ and $k \in \mathcal{N}_3$, where $y_{ik} = 1$ if and only if $u_i \geq k$. Because $u_i \geq 2$, we always have $y_{i1} = y_{i2} = 1$, which implies $u_i = \sum_{k \in \mathcal{N}_3} y_{ik} + 2$, yielding the following

Config	NNZ	Avg. Objective Value			Avg. Runtime (sec.)			Avg. TU Time
n		GRB	TU	No TU	GRB	TU	No TU	
3	1.60e1	14.3	14.3	14.3	0.00	2.03	1.90	0.00
6	2.65e2	22.5	∞	∞	0.02	1442.82	1800.00	0.00
12	2.76e3	35.7	∞	∞	0.21	1800.00	1800.00	0.00
24	2.48e4	50.9	∞	∞	2.35	1800.11	1800.00	0.00
48	2.10e5	82.7	∞	∞	55.79	1800.00	1800.00	0.01
96	1.72e6	200.9	∞	∞	1600.10	1800.00	1800.00	0.04
192	1.40e7	∞	∞	∞	1802.59	1800.00	1800.00	0.54
384	1.13e8	∞	∞	∞	1825.95	1800.01	1800.00	4.40
768	9.03e8	–	–	–	–	–	–	–

Table 7: TSP: objective values and runtimes. *TU* and *No TU* denote the implementations of GFORS without or with TU reformulation, respectively; "–" indicates out-of-memory. TU reformulation has a low impact potentially due to the loose relaxation and sparse feasible set.

unary-encoding reformulation,

$$
\begin{aligned}
\min_{\substack{x_{ij}\in\{0,1\}\\ y_{ik}\in\{0,1\}}} \quad & \langle c, x\rangle \\
\text{s.t.} \quad & \sum_{j\neq i} x_{ij} = 1, && \forall i \in \mathcal{N}, \\
& \sum_{i\neq j} x_{ij} = 1, && \forall j \in \mathcal{N}, \\
& \sum_{k\in\mathcal{N}_3} y_{ik} - \sum_{k\in\mathcal{N}_3} y_{jk} + n x_{ij} \leq n-1, && \forall i,j \in \mathcal{N}_2,\ i\neq j, \\
& y_{i,k-1} \geq y_{ik}, && \forall i \in \mathcal{N}_2,\ k \in \mathcal{N}_4.
\end{aligned}
$$

We consider sizes n ranging from 3 to 768, with distances c generated randomly. Results are reported in Table 7. In this case, all equality constraints can be eliminated via TU reformulation. The columns *TU* and *No TU* correspond to implementations with and without this step. However, both variants perform poorly: only the smallest instances are solved, while Gurobi can handle sizes up to $n = 96$ for the same formulation. This suggests that the loose relaxation of the MTZ formulation and the extremely sparse feasible set relative to the domain $[0, 1]^n$ are potentially the main bottlenecks for GFORS in this setting.

5 Conclusion

Across diverse problem classes, our study shows that a GPU-native combination of a PDHG-style first-order routine with feasibility-aware, batched sampling delivers strong time-to-incumbent on large BIPs. The framework runs end-to-end on GPUs with minimal synchronization, and leverages TU reformulations, customized sampling, and monotone relaxation to enhance sampling efficiency. It provides near-stationarity guarantees for the first-order component together with probabilistic bounds on sampled solutions. In practice, exact solvers such as Gurobi remain stronger on small–medium instances, while GFORS offers complementary scalability on large instances under tight time limits.

We highlight several directions for future work. First, integrating lower-bounding mechanisms is a priority to strengthen optimality guarantees. Second, the current design assumes explicitly

enumerated constraints, making it ill-suited to formulations that rely on dynamic separation or an exponential number of cuts. Third, extending beyond binary variables to general integer and mixed-integer programs requires further development. Finally, from a systems perspective, multi-GPU and distributed variants are a natural next step to expand problem scale and reduce time-to-solution.

References

[1] Tobias Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

[2] Tobias Achterberg, Robert E Bixby, Zonghao Gu, Edward Rothberg, and Dieter Weninger. Presolve reductions in mixed integer programming. *INFORMS Journal on Computing*, 32(2): 473–506, 2020.

[3] David L Applegate, Robert E Bixby, Vašek Chvátal, and William J Cook. The traveling salesman problem: a computational study. In *The Traveling Salesman Problem*. Princeton university press, 2011.

[4] Egon Balas and Manfred W Padberg. On the set-covering problem. *Operations Research*, 20 (6):1152–1161, 1972.

[5] Egon Balas and Matthew J Saltzman. An algorithm for the three-index assignment problem. *Operations Research*, 39(1):150–161, 1991.

[6] Dimitri P Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48 (3):334–334, 1997.

[7] E Robert Bixby, Mary Fenelon, Zonghao Gu, Ed Rothberg, and Roland Wunderling. Mip: Theory and practice—closing the gap. In *IFIP Conference on System Modeling and Optimization*, pages 19–49. Springer, 1999.

[8] Robert E Bixby. A brief history of linear and mixed-integer programming computation. *Documenta Mathematica*, 2012:107–121, 2012.

[9] Suresh Bolusani, Mathieu Besançon, Ksenia Bestuzheva, Antonia Chmiela, João Dionísio, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Mohammed Ghannam, Ambros Gleixner, et al. The scip optimization suite 9.0. *arXiv preprint arXiv:2402.17702*, 2024.

[10] Alberto Caprara, Hans Kellerer, Ulrich Pferschy, and David Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123(2):333–345, 2000.

[11] Sebastian Ceria, Cécile Cordier, Hugues Marchand, and Laurence A Wolsey. Cutting planes for integer programs with general integer variables. *Mathematical programming*, 81(2):201–214, 1998.

[12] Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of mathematical imaging and vision*, 40(1):120–145, 2011.

[13] Antonin Chambolle and Thomas Pock. An introduction to continuous optimization for imaging. *Acta Numerica*, 25:161–319, 2016.

[14] Patrick L Combettes and Jean-Christophe Pesquet. Proximal splitting methods in signal processing. In *Fixed-point algorithms for inverse problems in science and engineering*, pages 185–212. Springer, 2011.

[15] Emilie Danna, Edward Rothberg, and Claude Le Pape. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90, 2005.

[16] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

[17] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

[18] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[19] Uriel Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45 (4):634–652, 1998.

[20] Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical programming*, 98(1):23–47, 2003.

[21] Matteo Fischetti, Fred Glover, and Andrea Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.

[22] Gerald Gamrath, Thorsten Koch, Alexander Martin, Matthias Miltenberger, and Dieter Weninger. Progress in presolving for mixed integer programming. *Mathematical Programming Computation*, 7(4):367–398, 2015.

[23] Saeed Ghadimi and Guanghui Lan. Accelerated gradient methods for nonconvex nonlinear and stochastic programming. *Mathematical Programming*, 156(1):59–99, 2016.

[24] Michel X Goemans and David P Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995.

[25] Ralph E Gomory. Outline of an algorithm for integer solutions to linear programs and an algorithm for the mixed integer problem. In *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pages 77–103. Springer, 2009.

[26] Nick Gould and Philippe L Toint. Preprocessing for quadratic programming. *Mathematical Programming*, 100(1):95–132, 2004.

[27] *Gurobi Optimizer Documentation*. Gurobi Optimization, LLC, 2025. URL https://docs.gurobi.com/projects/optimizer/en/current/. Version 12.0. Accessed: 2025-09-30.

[28] Michael Jünger, Thomas M Liebling, Denis Naddef, George L Nemhauser, William R Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A Wolsey. *50 Years of integer programming 1958-2008: From the early years to the state-of-the-art*. Springer Science & Business Media, 2009.

[29] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum annealing in the transverse ising model. *Physical Review E*, 58(5):5355, 1998.

[30] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[31] Thorsten Koch, Timo Berthold, Jaap Pedersen, and Charlie Vanaret. Progress in mathematical programming solvers from 2001 to 2020. *EURO Journal on Computational Optimization*, 10: 100031, 2022.

[32] Gary Kochenberger, Jin-Kao Hao, Fred Glover, Mark Lewis, Zhipeng Lü, Haibo Wang, and Yang Wang. The unconstrained binary quadratic programming problem: a survey. *Journal of combinatorial optimization*, 28(1):58–81, 2014.

[33] Weiwei Kong and Renato DC Monteiro. An accelerated inexact proximal point method for solving nonconvex-concave min-max problems. *SIAM Journal on Optimization*, 31(4):2558–2585, 2021.

[34] Weiwei Kong, Jefferson G Melo, and Renato DC Monteiro. Complexity of a quadratic penalty accelerated inexact proximal point method for solving linearly constrained nonconvex composite programs. *SIAM Journal on Optimization*, 29(4):2566–2593, 2019.

[35] Ailsa H Land and Alison G Doig. An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pages 105–132. Springer, 2009.

[36] Gilbert Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.

[37] Jiaming Liang and Renato DC Monteiro. An average curvature accelerated composite gradient method for nonconvex smooth composite optimization problems. *SIAM Journal on Optimization*, 31(1):217–243, 2021.

[38] Jiaming Liang, Renato DC Monteiro, and Chee-Khian Sim. A FISTA-type accelerated gradient algorithm for solving smooth nonconvex composite optimization problems. *Computational Optimization and Applications*, 79(3):649–679, 2021.

[39] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[40] Haihao Lu and Jinwen Yang. cupdlp. jl: A gpu implementation of restarted primal-dual hybrid gradient for linear programming in julia. *arXiv preprint arXiv:2311.12180*, 2023.

[41] Haihao Lu, Zedong Peng, and Jinwen Yang. cupdlpx: A further enhanced gpu-based first-order solver for linear programming. *arXiv preprint arXiv:2507.14051*, 2025.

[42] Andrew Lucas. Ising formulations of many np problems. *Frontiers in physics*, 2:5, 2014.

[43] Silvano Martello and Paolo Toth. Algorithms for knapsack problems. *North-Holland Mathematics Studies*, 132:213–257, 1987.

[44] Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[45] Jean-Jacques Moreau. Proximité et dualité dans un espace hilbertien. *Bulletin de la Société mathématique de France*, 93:273–299, 1965.

[46] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[47] Brendan O'Donoghue. Operator splitting for a homogeneous embedding of the linear complementarity problem. *SIAM Journal on Optimization*, 31(3):1999–2023, 2021.

[48] Lauri Ojala and Dilay Celebi. The world bank's logistics performance index (lpi) and drivers of logistics performance. *Proceeding of MAC-EMM, OECD*, pages 3–30, 2015.

[49] Panos M Pardalos and Jue Xue. The maximum clique problem. *Journal of global Optimization*, 4(3):301–328, 1994.

[50] David Pisinger. Linear time algorithms for knapsack problems with bounded weights. *Journal of Algorithms*, 33(1):1–14, 1999.

[51] Kenneth R Rebman. Total unimodularity and the transportation problem: a generalization. *Linear Algebra and its Applications*, 8(1):11–24, 1974.

[52] Franz Rendl, Giovanni Rinaldi, and Angelika Wiegele. Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. *Mathematical Programming*, 121(2): 307–335, 2010.

[53] R Tyrrell Rockafellar. Monotone operators and the proximal point algorithm. *SIAM journal on control and optimization*, 14(5):877–898, 1976.

[54] Sungho Shin, Mihai Anitescu, and François Pacaud. Accelerating optimal power flow with gpus: Simd abstraction of nonlinear programs and condensed-space interior-point methods. *Electric Power Systems Research*, 236:110651, 2024.

[55] David B Shmoys, Éva Tardos, and Karen Aardal. Approximation algorithms for facility location problems. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 265–274, 1997.

[56] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. Osqp: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[57] Arie Tamir. Totally balanced and totally unimodular matrices defined by center location problems. *Discrete applied mathematics*, 16(3):245–263, 1987.

[58] Kensuke Tamura, Tatsuhiko Shirai, Hosho Katsura, Shu Tanaka, and Nozomu Togawa. Performance comparison of typical binary-integer encodings in an ising machine. *IEEE Access*, 9: 81032–81039, 2021.

[59] Stacy A Voccia, Ann Melissa Campbell, and Barrett W Thomas. The same-day delivery problem for online purchases. *Transportation Science*, 53(1):167–184, 2019.

[60] Peng Wei, Yi Cao, and Dengfeng Sun. Total unimodularity and decomposition method for large-scale air traffic cell transmission model. *Transportation research part B: Methodological*, 53:1–16, 2013.

[61] Laurence A Wolsey. *Integer programming*. John Wiley & Sons, 2020.

[62] Mingqiang Zhu and Tony Chan. An efficient primal-dual hybrid gradient algorithm for total variation image restoration. *Ucla Cam Report*, 34(2), 2008.