A GRADIENT-BASED PRIMAL HEURISTIC FOR MULTI-DIMENSIONAL KNAPSACK

Anonymous authors
Paper under double-blind review

ABSTRACT

The Multidimensional Knapsack (MKP) is a canonical combinatorial optimization problem underlying many problems such as resource-allocation and selection tasks. At large scale, obtaining strong feasible solutions under limited computational budgets remains practically important at which mature exact solvers and specialized heuristics struggle. A central goal of this paper is to obtain high-quality feasible solutions quickly, motivating primal heuristics. Towards that goal, we introduce a continuous relaxation for MKP, combined with a GPU-parallelizable primal-search approach. Our formulation penalizes the maximum normalized constraint ReLU-based violation rather than an aggregate penalty, so that each first-order update is driven by a maximally violated constraint. Hence, we term our method as MAX-ReLU. Building on that, the resulting algorithm applies straight-through estimator rounding and gradient updates to a large batch of trajectories, evaluates binary candidates throughout optimization, and retains the best feasible incumbent found so far. We theoretically analyze feasibility and near-binarity properties of the relaxation, with key arguments extending beyond nonnegative constraint matrices. Experiments on small- and large-scale multidimensional-knapsack instances (synthesized by a well-known MKP generator) show that MAX-ReLU, as compared to GPU- and CPU-based exact and greedy solvers, rapidly finds high-quality feasible solutions, with particularly strong early-time performance in regimes with large number of constraints.

1 INTRODUCTION

Multidimensional knapsack (MKP) is a fundamental problem for discrete resource allocation and selection under multiple simultaneous constraints Fréville (2004); Cacchiani et al. (2022). MKP entails choosing a subset of projects, jobs, or items to maximize total value while respecting several limited resources, such as budget, capacity, time, or energy. The same structure appears broadly whenever many binary decisions compete for multiple shared resources.

MKP arises in many domains such as manufacturing Cherbaka & Meller (2008), project portfolio selection Kocamaz (2014); Bas (2012), and computing resource allocation Abdel-Basset et al. (2019); Mengistu et al. (2019); Ajayi et al. (2023); Yu et al. (2021).

Unlike the standard single-constraint knapsack problem, MKP couples each decision variable to several constraints at once, making simple one-dimensional ranking rules significantly less effective and creating substantially richer interactions among candidate selections. This makes MKP a much harder task when compared to single-constraint knapsack Fréville (2004). For this reason, MKP has long served both as a practical optimization model and as a canonical testbed for constrained combinatorial optimization Dokka et al. (2022).

Large MKP instances often require strong feasible solutions well before optimality can be certified. This is particularly important in resource-allocation settings where a high-quality early incumbent may already support downstream decisions, while proving optimality can require substantially more computation Cherbaka & Meller (2008); Mengistu et al. (2019). The challenge becomes more pronounced as both the number of variables and the number of constraints grow. At the same time, modern GPUs make massively parallel search increasingly practical, motivating continuous and first-order approaches that can update many candidate solutions simultaneously Alkhouri et al. (2025); Feng & Yang (2025).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

A natural direction for exploiting GPUs is to replace binary variables by continuous ones and optimize the resulting relaxation with first-order methods Li & Zhang (2025). Such methods are attractive because simple updates can be evaluated efficiently over large batches of candidate solutions in parallel. Recent work has explored this strategy through relaxed QUBO formulations, with encouraging results on graph problems Alkhouri et al. (2026); Sun et al. (2026). For constrained problems such as MKP, however, the design could be more challenging as a simple continuous relaxation can enforce the linear constraints while sacrificing binary structure, producing feasible but highly fractional solutions at which rounding may yield infeasible or sub-optimal results. Conversely, formulations that more strongly encode binary structure can introduce their own problematic optimization landscapes. For knapsack-type constraints, standard quadratic-penalty relaxations have infeasible local minima even as the penalty increases, as recently shown in Huang et al. (2026).

This creates a design challenge where a useful relaxation for gradient-based primal search should not only make feasible points accessible, but should have local minima with enough discrete structure to serve as useful surrogates for binary solutions. We therefore ask:

Can one construct a relaxation for MKP whose local minima are both feasible and close to binary, while enabling an efficient parallel algorithm?

To address this question, we introduce a max-violation relaxation that penalizes the largest normalized constraint violation together with a concave quadratic term that favors binary values. These two components play complementary roles. The max-violation term controls feasibility, while the concave term drives local minima toward vertices of the feasible polytope. Above an explicit penalty threshold, every local minimizer is feasible, and with a positive quadratic coefficient, every local minimizer is a vertex of the feasible polytope and therefore has at most $(\mathrm{rank}(A))$ fractional coordinates.

These guarantees characterize the structure of the relaxation rather than the convergence of a particular optimizer. We therefore use the relaxation as the basis of a GPU primal-search heuristic built from a straight-through rounded estimator combined with gradient updates and massively parallel first-order trajectories. The method evaluates binary feasible candidates throughout the search and retains the best incumbent found. On the large dense MKP instances studied here, this simple approach is highly competitive in the short-time regime, with its advantage becoming more pronounced as the number of variables and the number of constraints grow. These results suggest that improving the structure of the relaxation itself can substantially enhance the effectiveness of GPU-based first-order search for constrained binary optimization. We summarize our contributions next.

- **Structured Continuous Relaxation:** We introduce a max-ReLU continuous relaxation for multidimensional knapsack and establish an explicit penalty condition guaranteeing feasibility, together with a bound on the number of fractional coordinates at every local minimizer.
- **Massively Parallel GPU Primal Search:** We develop a massively parallel GPU primal heuristic based on this relaxation using binary rounding, the maximally violated constraint, straight-through gradients, and first-order optimizers (e.g., Adam Kingma & Ba (2015)).
- **Comprehensive Empirical Evaluation:** We evaluate time to target, capped mean time to target, target-attainment curves, reference-relative primal integral, and solution quality after 60 seconds against Gurobi, cuOPT, Local-MIP, greedy search, GFORS, and PQQA. On the tested large dense instances, MAX-ReLU finds high-quality feasible solutions within seconds.

2 Related Work

Primal heuristics for MKP. Primal heuristics aim to find good feasible solutions within a limited time budget. For MKP, Linear Programming (LP) and surrogate relaxations help guide the search for good item selections (Fréville, 2004; Puchinger et al., 2010). General mixed-integer programming (MIP) heuristics also use constraint violations and neighborhood search to find and improve feasible solutions (Luteberget & Sartor, 2023; Lin et al., 2024; Fischetti & Lodi, 2003). Recent GPU Integer LP frameworks such as cuOPT combine several of these approaches (Çördük et al., 2025). Our method, MAX-ReLU, shares this practical goal. Since the empty selection is already feasible in our setting, we focus on finding high-profit feasible solutions quickly.

Gradient-based search on GPUs. Recent GPU methods for combinatorial optimization use gradients in different ways to search for binary solutions. For instance, pCQO-MIS uses parallel gradient

updates on a quadratic relaxation of the maximum independent set (MIS) problem (Alkhouri et al., 2025). mQO combines gradient search, mutation, and discrete local search for MIS and MaxCut (Sun et al., 2026). PQQA explores a continuous relaxation through gradient-based sampling, a changing annealing objective, and communication between parallel runs (Ichikawa & Arai, 2025). GFORS uses primal–dual updates to handle constraints and samples binary candidates from continuous states (Wei & Liang, 2025). MAX-ReLU also utilizes a continuous relaxation for MKP, but uses a specialized fixed objective that penalizes the largest normalized constraint violation instead of a direct quadratic penalty as in pCQO for MIS. MAX-ReLU's binary candidates come from rounding and also determine the update directions, while maintaining parallel trajectories that evolve independently.

Penalty objectives and local minima. A continuous penalty objective can have local minima that are infeasible or fractional. Classical exact-penalty theory studies when finite penalty weights preserve constrained local minima (Han & Mangasarian, 1979). Concave reformulations also encourage variables to take binary values (Raghavachari, 1969). GFORS uses the same form of quadratic binary penalty as MAX-ReLU (Wei & Liang, 2025). The relevant question is therefore what the combined objective guarantees about its local minima. The recent study in Huang et al. (2026) shows feasibility and binary structure for quadratic-penalty relaxations under structural conditions. We combine a concave quadratic term with a penalty on the largest normalized constraint violation. Under our sufficient conditions, every local minimizer of the continuous objective is a feasible vertex of the LP relaxation. Such vertices have at most $\mathrm{rank}(A)$ fractional coordinates, as follows from standard LP geometry.

Continuous states and rounded updates. The local minima of the objective do not by themselves describe the behavior of the rounded algorithm. Similar to BinaryConnect (Courbariaux et al., 2015; Dockhorn et al., 2021), MAX-ReLU retains continuous states to accumulate updates while using binary states in computation. It uses a straight-through rule for threshold rounding. The direction computed at a binary candidate is passed to the underlying continuous state (Bengio et al., 2013). This direction need not be a subgradient at the continuous state, so we analyze the rounded Adam iteration separately. Under sufficient conditions, every visit to a fixed infeasible rounded state ends after finitely many updates.

3 The Proposed Continuous Relaxation & Its Landscape

Formally, the 0-1 MKP is given as

$$\max_{z\in\{0,1\}^n} c^\top z \quad \text{subject to} \quad Az \le b, \tag{1}$$

where $z \in \{0,1\}^n$ is the vector of binary decision variables, and $c \in \mathbb{R}^n_+$ is the profit vector. The m constraints are encoded using $A \in \mathbb{R}^{m\times n}$ with each coordinate nonnegative, and $b \in \mathbb{R}^m$ with each coordinate positive. Here, $A_{k,i}$ indicates resource $k \in [m]$ that is consumed by item $i \in [n]$, where b_k indicates the capacity of resource k.

Continuous relaxations provide a convenient way to apply first-order methods to binary optimization. For a quadratic unconstrained binary optimization problem (QUBO),

$$\min_{z\in\{0,1\}^n} z^\top Qz + d^\top z,$$

where Q and d are problem-specific parameters[1], one may replace $\{0,1\}^n$ by $[0,1]^n$ and optimize over the box. This viewpoint has been used in recent gradient-based search methods for combinatorial problems as was discussed in the previous section for MIS and MaxCut Sun et al. (2026). For knapsack problems, however, the choice of constraint penalty is important as some quadratic penalties can retain infeasible local minima even as the penalty coefficient increases Huang et al. (2026).

The continuous relaxation developed below will later be used to construct the rounded straight-through search in Section 4. To distinguish the continuous relaxation from the original binary variable z, we use $y \in [0,1]^n$ to denote the relaxed decision variable. For MKP, we penalize the largest normalized constraint violation.

[1]As an example, for MIS, Q and d are the adjacency matrix and the negative all-ones vector, respectively.

Let $a_k^\top$ denote row $k \in [m]$ of A, and define $q = \frac{c}{\mathbf{1}^\top c}$, $r_k(y) = \frac{a_k^\top y - b_k}{b_k}$, $V(y) = \max\{0, r_1(y), \ldots, r_m(y)\}$. Then, $V(y) = 0$ exactly when $Ay \le b$, while $V(y) > 0$ measures the largest relative constraint violation.

For $\lambda > 0$ and $\beta \ge 0$, we propose

$$\boxed{\min_{y \in [0,1]^n} F(y), \qquad F(y) = -q^\top y + \lambda V(y) + \frac{\beta}{n} y^\top (\mathbf{1} - y)} \tag{2}$$

The three terms have distinct roles. The first favors high-profit solutions, the second penalizes infeasibility, and the last favors binary values. Indeed, $y^\top(\mathbf{1}-y)$ is concave and nonnegative on $[0,1]^n$, and it vanishes exactly on $\{0,1\}^n$. Hence, β changes the relaxed landscape without changing the objective on binary points. Equivalently, the objective $F(y) = y^\top \left(-\frac{\beta}{n} I\right) y + \left(\frac{\beta}{n}\mathbf{1} - q\right)^\top y + \lambda V(y)$ is quadratic apart from the piecewise-linear max-ReLU penalty V.

The normalization gives λ a more consistent scale across instances. Dividing the profit vector by $\mathbf{1}^\top c$ fixes the total profit scale, while dividing row k of A by b_k measures violation relative to the corresponding capacity.

The maximum penalty is also insensitive to duplicated constraints: duplicating a row does not change V, and adding a row that remains strictly satisfied in a neighborhood of y does not affect V locally.

We next ask when the penalty is large enough to exclude infeasible local minima. Let

$$P = \{y \in [0,1]^n : Ay \le b\}, \qquad \lambda_0 = \max_{y \in P} \left(q + \frac{\beta}{n}\mathbf{1}\right)^\top y. \tag{3}$$

This threshold is the value of a linear program over P. Moreover, $\max_{y \in P} q^\top y \le \lambda_0 \le \max_{y \in P} q^\top y + \beta \le 1 + \beta$.

Theorem 1. *Assume $b > 0$, $q > 0$, and $\beta > 0$. If $\lambda \ge \lambda_0$, then every local minimizer $y^\star$ of F over $[0,1]^n$ satisfies*

$$Ay^\star \le b, \qquad y^\star \text{ is a vertex of } P, \qquad \left|\{j : 0 < y_j^\star < 1\}\right| \le \operatorname{rank}(A) \le m. \tag{4}$$

These conclusions do not require $A \ge 0$.

The theorem gives two properties of the relaxed landscape:

1. A sufficiently large λ rules out infeasible local minima.
2. When $\beta > 0$, any remaining local minimizer must lie at a vertex of the feasible polytope. Such a vertex need not be binary, but it can have at most $\operatorname{rank}(A)$ fractional coordinates. Equivalently, at least $n - \operatorname{rank}(A)$ coordinates are binary.

Thus, λ and β play complementary roles, with λ controlling feasibility of local minima and $\beta > 0$ inducing the vertex structure through strict concavity on P. We defer the detailed proof to Appendix A.2.

Remark 1. Theorem 1 characterizes the continuous objective rather than the convergence of a particular optimizer. Landscape analyses in nonconvex optimization commonly focus on local minima, motivated in part by results showing that gradient-based methods can avoid strict saddles under suitable assumptions Lee et al. (2016); Jin et al. (2017); Staib et al. (2019). We treat the landscape and solver dynamics separately: this section characterizes the continuous objective defining the search direction, while Appendix A.4 analyzes the solver.

Remark 2. The condition $\lambda \ge \lambda_0$ is sufficient, but it can be conservative. Consider the example

$$A = (1,1), \qquad b = 3/2, \qquad q = (0.9, 0.1), \qquad \beta = 0.1.$$

In this case, $\lambda_0 = 1.025$. For every infeasible $y \in [0,1]^2$, we have $y_1 + y_2 > 3/2$, and

$$\frac{\partial F}{\partial y_2}(y) = \frac{2\lambda}{3} - 0.05 - 0.1 y_2 \ge \frac{2\lambda}{3} - 0.15.$$

Since infeasibility implies $y_2 > 1/2$, the direction $-e_2$ is feasible for a sufficiently small step. Hence, whenever $\lambda > 0.225$, $-e_2$ is a feasible descent direction at every infeasible point. No infeasible point can therefore be a local minimizer, even though λ may be well below λ_0.

The threshold λ_0 is used only to characterize the landscape; the solver does not compute it or require $\lambda \ge \lambda_0$. As shown in Remark 2, the sufficient condition can be conservative, and in practice we treat λ as a search parameter.

4 THE MAX-RELU ALGORITHM

The landscape results in Section 3 describe the structure of the continuous objective. In this section, we introduce a primal heuristic, MAX-ReLU, that uses this landscape structure to search efficiently for high-quality solutions.

4.1 ROUNDED UPDATE DIRECTION

Each trajectory forms a binary state by thresholding its continuous vector $z_t = R(x_t),\ R(x)_j = \mathbf{1}\{x_j \geq 1/2\}$. For an infeasible binary vector z, let $k(z) \in \arg\max_{1\leq k\leq m} r_k(z)$, where ties are resolved by a fixed deterministic rule. Since $V(z) = \max_k \mathrm{ReLU}(r_k(z))$, the selected row is one of the maximally violated normalized constraints. We use the following subgradient of F at the rounded point z:

$$g_j(z) = -q_j + \frac{\beta}{n}(1-2z_j) + \begin{cases} 0, & (Az)_k \leq b_k,\ \forall k \in [m], \\ \lambda a_{k(z)j}/b_{k(z)}, & \text{otherwise.} \end{cases} \tag{5}$$

For an infeasible vector z, $a_{k(z)}/b_{k(z)}$ is a valid subgradient of V. If z is feasible, then $V(z) = 0$ and zero is a valid choice for the penalty subgradient. Hence, $g(z) \in \partial F(z)$.

Each term in $g(z)$ has a distinct role in the rounded search. Under a descent update, the profit term $-q_j$ tends to increase x_j and favors high-profit items. For standard MKP instances with $A \geq 0$, the max-ReLU term places larger positive pressure on coordinates of items that consume the maximally violated resource, biasing the update toward reducing the current worst normalized violation. The concave term remains useful even though z is already binary: its descent direction is $\frac{\beta}{n}(2z_j - 1)$, which pushes x_j away from the rounding threshold and toward its current rounded value z_j. Thus, the surrogate direction balances profit improvement, feasibility correction, and stabilization of the current rounded representation.

Importantly, $g(z_t)$ is a subgradient of F at the rounded state z_t, not generally at the continuous state x_t. MAX-ReLU nevertheless applies this direction to x_t, corresponding to a straight-through treatment of the rounding operation Bengio et al. (2013); Courbariaux et al. (2015). Thus, the continuous state determines which rounding region is visited, while the rounded state determines the direction used to update the continuous trajectory. This should be viewed as a straight-through surrogate search direction rather than ordinary subgradient descent on $F(x_t)$.

The max-violation update also admits a simple batched implementation. For a batch of rounded vectors $Z_t \in \{0,1\}^{B\times n}$, all constraint loads are evaluated through the matrix product $Z_t A^\top$. This requires $O(Bmn)$ dense arithmetic per iteration. All row residuals are needed to identify the maximum violation, while only the selected row contributes to the penalty part of $g(z)$. The resulting operations consist mainly of dense matrix multiplication, row reduction, and elementwise updates, which are well suited to GPU execution.

The rounded iteration also admits a direct local characterization. For a fixed binary state z, the update direction $g(z)$ remains constant as long as the continuous trajectory stays in the corresponding rounding region. Appendix A.4 shows that, under the sufficient condition $\lambda > \lambda_0$, every visit to an infeasible rounded state ends after finitely many updates. Thus, an infeasible binary state cannot stall the iteration indefinitely. This is a local non-stalling property and does not imply that the next rounded state is feasible or that the full iteration converges. This sufficient condition is used only for the analysis and is not imposed on the experimental choice of λ.

4.2 PARALLEL ITERATION AND OUTPUT

MAX-ReLU runs B trajectories in parallel. Each trajectory has its own continuous state and Adam moments, while all trajectories share the problem data and are updated in a single batch. At every iteration, the continuous states are rounded, one Adam step is taken using equation 5, and the updated binary candidates are checked against the original constraints.

Each trajectory is initialized below the rounding threshold so that its initial rounded state is $z_0 = 0$. In the experiments, the continuous coordinates are sampled independently as $x_0 \sim \mathrm{Unif}([0.30, 0.33]^n)$. The continuous states therefore differ across trajectories even though all initial rounded states equal zero. The full procedure is given in Algorithm 1.

Algorithm 1: MAX-ReLU

Input: A, b, c, batch size B, penalty parameter λ, binary parameter β, learning rate α, and time limit T.

1. Initialization. Sample B continuous starts below the rounding threshold (experiments use $[0.30, 0.33]^n$). Initialize all Adam moments to zero and set the incumbent $z^\star = 0$.

2. Parallel search. Until time T, repeat the following for all B trajectories in parallel:
(a) Set $z_t = R(x_t)$, compute all normalized row residuals, select $k(z_t)$ for each infeasible candidate using the maximally violated constraint rule, and get $g(z_t)$ from equation 5;
(b) Take one Adam step using $g(z_t)$ and project the result onto the box: $x_{t+1} \in [0, 1]^n$;
(c) Round the updated states $z_{t+1} = R(x_{t+1})$, and check the resulting binary candidates for feasibility;
(d) For each trajectory's candidate z_{t+1}, if $Az_{t+1} \leq b$ and $c^\top z_{t+1} > c^\top z^\star$, set $z^\star \leftarrow z_{t+1}$.

3. Output. Return $z^\star$.

The Adam recurrence used in step 2(b) is given in Appendix A.3. The feasibility check used to update the incumbent is performed on the original MKP constraints. Thus, the returned solution is binary and feasible by construction.

Overall, the continuous relaxation supplies a structured search direction, the straight-through iteration explores the corresponding rounded states, and the incumbent archive converts these visits into a primal heuristic whose output is selected only by feasibility and the original profit.

5 Experiments

5.1 Instances and solvers

Classical public MKP benchmarks are relatively small by modern solver standards and are often solved to high quality within a fraction of a second by current commercial solvers. Since our goal is to study early primal progress in regimes where obtaining a strong incumbent remains nontrivial, we use a standard Chu–Beasley-style MKP generation procedure Chu & Beasley (1998) to construct substantially larger instances and systematically vary both the number of variables and the constraint density. The purpose of these larger instances is therefore not to replace classical benchmarks, but to create a regime in which the quality and speed of primal heuristics can be meaningfully distinguished.

For each n and instance seed, the base instance has $0.06n$ rows with entries uniform on $\{1, \ldots, 999\}$. We set $b_k = \left\lfloor 0.25 \sum_j a_{kj} \right\rfloor$. Each profit is the item's mean coefficient across these base rows plus an independent uniform value on $[0, 500)$. To increase m, we append independently generated rows while keeping the profit vector and all existing rows fixed. The ratios are $m/n \in \{0.06, 0.12, 0.24, 0.36, 0.48, 0.60, 0.72\}$. m/n measures the number of constraints relative to the number of variables. Added rows are independent of the profits, which depend on the base rows. This nested construction also changes the coefficient–profit correlation as m/n increases.

The main grid has $n \in \{500, 1000, 2000\}$ and eight instance seeds per $(n, m/n)$ pair, giving 168 instances. We also test $n = 10{,}000$ at $m/n \in \{0.12, 0.36, 0.72\}$ with four seeds per pair. The batch size is 4096 when $n \leq 2000$ and 1024 at $n = 10{,}000$. Unless stated otherwise, $\lambda = 0.3$, $\beta = 0.05$, $\alpha = 0.005$, and solver seed 12. Adam Kingma & Ba (2015) uses $(\mu_1, \mu_2) = (0.9, 0.999)$ and $\epsilon = 10^{-8}$. Ties between maximally violated rows use the first row.

We compare Gurobi 13.0.3 with `MIPFocus`=0, `MIPFocus`=1, and `NoRelHeurTime`=60 Gurobi Optimization, LLC (2026). Each Gurobi run uses eight CPU threads and seed 12. We use default cuOPT MILP as the primary cuOPT configuration throughout the full grid, and also compare LocalMIP CP2024. The greedy baseline orders items by $\frac{c_j}{\sum_i a_{ij}/b_i}$ and inserts an item when the result remains feasible. A Gurobi baseline starts from this greedy solution and restricts the search to Hamming radius $\max\{20, \lceil 0.05n \rceil\}$. CPU runs use AMD EPYC 9365 processors. GPU runs use NVIDIA RTX PRO 6000 Blackwell Server Edition cards.

We evaluate the first 60 seconds under each implementation's natural solver boundary. Appendix B.1 gives the complete timing and feasibility conventions. To test whether the clock boundary materially affects the large-instance comparison, we additionally perform a timing sensitivity study on the $n = 10{,}000$ instances. The sensitivity results are reported in Section 5.5 and Appendix B.2.

GFORS Wei & Liang (2025) and PQQA Ichikawa & Arai (2025) are evaluated on a 63-instance subset with three seeds for each of the 21 $(n, m/n)$ pairs and reported in Appendix B.4. Their records are not used for exact first-hit timing. We therefore report checked final binary quality for these two methods without assigning an earlier target time. MAX-ReLU has higher checked final objective than each method on all paired instances.

5.2 METRICS

Let $T = 60$ seconds. For instance i, let $v_{\mathrm{ref},i} > 0$ be the best feasible objective from a 20 minute Gurobi (`MIPFocus`=1) run, and let $v_i(t)$ be the best feasible value recorded by time t. Before the first recorded feasible solution, set $v_i(t) = 0$, the value of the feasible zero vector.

For a relative target gap $\eta \in (0, 1)$, define $t_{\eta,i} = \inf\{t \in [0, T] : v_i(t) \geq (1-\eta)v_{\mathrm{ref},i}\}$, $\inf \varnothing = \infty$. For example, the 0.5% target uses $\eta = 0.005$. We report the number reached by T and the median of the finite first-hit times Aiex et al. (2007).

The capped mean time (PAR1) and the target-attainment curve $A_\eta(t)$ are

$$\mathrm{PAR1}_\eta = \frac{1}{N}\sum_{i=1}^{N} \min\{t_{\eta,i}, T\}, \quad A_\eta(t) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\{t_{\eta,i} \leq t\},\ 0 \leq t \leq T. \tag{6}$$

These summaries are related by

$$\mathrm{PAR1}_\eta = \int_0^T [1 - A_\eta(t)]\, dt. \tag{7}$$

The curve shows when targets are reached. PAR1 summarizes the same distribution in seconds. The plotted percentage is $100A_\eta(t)$.

To compare solution quality throughout the run, we use a primal integral relative to the feasible reference Berthold (2013):

$$g_i(t) = \max\left\{0, \frac{v_{\mathrm{ref},i} - v_i(t)}{v_{\mathrm{ref},i}}\right\}, \qquad PI_{60,i} = \int_0^T g_i(t)\, dt. \tag{8}$$

The reference need not be optimal. A value above it contributes zero gap. We report the mean $\overline{PI}_{60} = N^{-1}\sum_i PI_{60,i}$ and $100\overline{PI}_{60}/T$. The latter is the mean reference gap over instances and time. Both reward early high-quality solutions and later improvement. The integral is computed from the recorded step function, with each incumbent value held until the next improvement or the cutoff.

For quality at 60 seconds, we report $100\sum_i v_i(T)/\sum_i v_{\mathrm{ref},i}$, the ratio of mean objectives over the same instance seeds. It can exceed 100% because the reference is a feasible solution. The 20 minute Gurobi runs remain roughly one half percent below their recorded bounds on this grid. Appendix B.3 gives the full 1% and 0.5% reach and conditional-median tables.

The parameter and batch sweeps use the smaller 60 second Gurobi upper bound from the default and `MIPFocus`=1 runs. Section 5.6 states those targets separately.

5.3 PRIMAL SOLUTION QUALITY AT 60 SECONDS

A strong feasible solution found efficiently can either be used directly under a limited computational budget or passed to a subsequent exact solver as a warm start. Accordingly, the 60-second experiments are intended to measure early primal progress. Table 1 gives the main 60 second comparison result. MAX-ReLU has the highest mean objective in 20 of the 21 regimes. The only exception is $n = 500$, $m/n = 0.06$. Table 2 summarizes the reference-relative primal integral over all 168 main-grid instances. MAX-ReLU has the smallest mean integral and the smallest mean reference gap over time among the compared methods. This metric rewards both early feasible quality and later improvement.

5.4 INCREASING THE NUMBER OF CONSTRAINTS

Table 3 reports $n = 2000$ at the 0.5% target. At $m/n = 0.06$, the two standard Gurobi settings have smaller PAR1 than MAX-ReLU. From $m/n = 0.12$ onward, MAX-ReLU has the smallest PAR1 among the four methods shown. At $m/n = 0.72$, MAX-ReLU reaches all eight targets with conditional median 1.042 seconds and PAR1 1.070 seconds. cuOPT reaches seven of eight, so its PAR1 rises to 14.723 seconds when its conditional median among reached runs is 7.731 seconds.

Table 1: Solution quality after 60 seconds on the main grid. Values are mean objective divided by the mean 20 minute Gurobi (`MIPFocus`=1) target in each $(n, m/n)$ regime (which is the reference). Each row has eight paired seeds. Bold marks the highest value and underline marks the second highest.

n	m/n	MAX-ReLU	cuOPT	Gurobi MIPFocus=0	Gurobi MIPFocus=1	Gurobi NoRel 60 s	Local-MIP	Greedy
500	0.06	99.938	99.864	99.944	**99.956**	99.923	94.174	95.945
500	0.12	**99.947**	99.753	99.898	99.941	99.894	94.402	94.382
500	0.24	**99.977**	99.726	99.927	99.951	99.866	95.406	94.195
500	0.36	**100.008**	99.721	99.918	99.924	99.871	95.441	93.887
500	0.48	**100.012**	99.670	99.846	99.912	99.831	95.715	94.695
500	0.60	**99.942**	99.627	99.830	99.860	99.753	95.862	94.908
500	0.72	**100.008**	99.668	99.865	99.900	99.849	96.046	94.840
1000	0.06	**99.991**	99.641	99.953	99.957	99.920	96.023	95.518
1000	0.12	**100.028**	99.620	99.940	99.955	99.905	96.535	95.930
1000	0.24	**100.041**	99.683	99.931	99.942	99.878	96.947	96.385
1000	0.36	**100.035**	99.701	99.890	99.931	99.840	97.351	96.024
1000	0.48	**100.040**	99.741	99.901	99.932	99.849	97.375	96.545
1000	0.60	**100.037**	99.660	99.926	99.931	99.837	97.245	96.589
1000	0.72	**100.048**	99.667	99.920	99.934	99.788	97.263	96.696
2000	0.06	**100.012**	99.711	99.973	99.968	99.893	96.930	96.973
2000	0.12	**100.028**	99.713	99.963	99.968	99.917	97.220	96.489
2000	0.24	**100.020**	99.658	99.965	99.973	99.914	97.492	96.821
2000	0.36	**100.022**	99.648	99.952	99.978	99.923	97.685	96.801
2000	0.48	**100.027**	99.684	99.956	99.972	99.913	97.965	96.967
2000	0.60	**100.017**	99.658	99.935	99.954	99.909	97.935	97.141
2000	0.72	**100.003**	99.613	99.946	99.949	99.904	98.011	97.154

Table 2: Reference-relative primal integral over all 168 main-grid instances. The last column is $100\overline{PI}_{60}/60$, the mean reference gap over instances and time. Lower is better.

Method	$\overline{PI}_{60}$ (s)	Mean ref. gap
MAX-ReLU $B = 4096$	**0.135405**	**0.225675%**
Gurobi MIPFocus=1	0.174043	0.290072%
Gurobi default	0.187171	0.311952%
Gurobi local branching	0.232500	0.387500%
Gurobi NoRel	0.252138	0.420231%
cuOPT default	0.484007	0.806678%
Greedy	2.436428	4.060713%
Local-MIP CP2024	4.147131	6.911884%

Table 3: $n = 2000$ MKP at the 0.5% target. Each target cell reports reach count, conditional median first-hit time, and PAR1, in that order. Times are seconds. Bold and underline mark the smallest and second-smallest PAR1 in each row.

m/n	MAX-ReLU obj./target	MAX-ReLU	Gurobi MIPFocus=0	Gurobi MIPFocus=1	cuOPT
0.06	100.012%	8/8, 0.376, 0.389	8/8, 0.180, 0.229	**8/8, 0.184, 0.208**	8/8, 1.484, 1.624
0.12	100.028%	**8/8, 0.389, 0.407**	8/8, 1.230, 1.055	8/8, 0.462, 0.490	8/8, 2.171, 2.212
0.24	100.020%	**8/8, 0.473, 0.477**	8/8, 0.809, 0.860	8/8, 0.823, 0.840	6/8, 3.686, 22.328
0.36	100.022%	**8/8, 0.664, 0.664**	8/8, 1.526, 1.498	8/8, 1.471, 1.471	7/8, 4.507, 11.491
0.48	100.027%	**8/8, 0.712, 0.710**	8/8, 2.098, 2.294	8/8, 1.949, 2.250	8/8, 5.428, 5.611
0.60	100.017%	**8/8, 0.883, 0.877**	8/8, 2.603, 2.720	8/8, 2.537, 2.624	8/8, 7.632, 9.648
0.72	100.003%	**8/8, 1.042, 1.070**	8/8, 3.140, 3.076	8/8, 2.914, 2.851	7/8, 7.731, 14.723

Figure 1 shows the same 0.5% target over the complete 168-instance main grid for the core solver set. Gurobi obtains a larger reached fraction at the first few tenths of a second. MAX-ReLU then rises more quickly and reaches all instances within the displayed window.

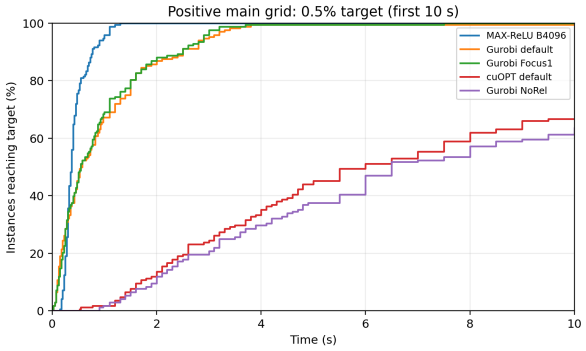


Figure 1: Target-attainment curve for the 0.5% target on the complete main grid, restricted to the first 10 seconds. The vertical axis is the percentage of instances that have reached the target by time t. The figure shows the core solver set under the clocks described in Appendix B.1.

Table 4: $n = 10{,}000$ MKP at the 0.5% target under the primary timing conventions. Each cell reports reach count, conditional median, and PAR1. A dash means there is no conditional median because no instance reaches the target. Times are seconds. Bold and underline mark the smallest and second-smallest PAR1 in each row.

m/n	MAX-ReLU	cuOPT	Gurobi MIPFocus=0	Gurobi MIPFocus=1	Gurobi NoRel 60 s
0.12	**4/4, 1.385, 1.387**	4/4, 19.701, 19.699	4/4, 18.131, 17.953	4/4, 18.968, 18.876	4/4, 23.553, 28.004
0.36	**4/4, 2.682, 2.695**	4/4, 39.824, 40.230	0/4, –, 60.000	0/4, –, 60.000	3/4, 47.175, 52.391
0.72	**4/4, 5.126, 5.079**	0/4, –, 60.000	0/4, –, 60.000	0/4, –, 60.000	0/4, –, 60.000

5.5 Increasing the problem size

Table 4 reports the primary comparison at $n = 10{,}000$. At this scale, MAX-ReLU performs better than the other methods shown. For timing sensitivity, we rerun cuOPT with `presolve`=0 under the same Solve-entry clock and form a conservative MAX-ReLU clock by adding each run's measured excluded setup time to all of its incumbent-event times. Disabling presolve improves cuOPT's early-time behavior, but setup-charged MAX-ReLU runs still reach the 0.5% target earlier in all three regimes. Because disabling presolve is not uniformly preferable when judged by final solution quality, we retain default cuOPT as the full-grid baseline. Appendix B.2 reports the corresponding results.

5.6 Parameters and batch size

Parameter and batch-size studies are performed on $n = 2000$. The scanned sufficient thresholds satisfy $\lambda_0 \in [0.3151, 0.3244]$, so $\lambda = 0.33$ exceeds all of them, but is slower at reaching both the 1% and 0.5% targets. We therefore retain $\lambda = 0.3$ as the choice; the sufficient threshold is a landscape guarantee rather than a parameter-selection rule. Appendices B.5 and B.6 give the full results.

6 Conclusion

We introduced a max-ReLU continuous relaxation and a GPU primal heuristic for multidimensional knapsack. The main landscape result gives an explicit penalty value above which every continuous local minimizer is feasible. With $\beta > 0$, each local minimizer is also a vertex of the feasible polytope and has at most $\mathrm{rank}(A)$ fractional coordinates. The solver uses this relaxation as an update rule, rounds every trajectory to a binary vector, and stores the best feasible solution found during the run. On the tested large dense MKP instances, this gives fast access to high-quality feasible solutions, especially as the number of constraints grows. The objective at 60 seconds and the primal integral summarize solution quality. Reach counts, conditional medians, PAR1, and attainment curves summarize time to the target values.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish writing. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Mohamed Abdel-Basset, Doaa El-Shahat, Hossam Faris, and Seyedali Mirjalili. A binary multi-verse optimizer for 0-1 multidimensional knapsack problems with application in interactive multimedia systems. *Computers & Industrial Engineering*, 132:187–206, 2019.

R. M. Aiex, M. G. C. Resende, and C. C. Ribeiro. TTT plots: A perl program to create time-to-target plots. *Optimization Letters*, 1:355–366, 2007.

Jesutofunmi Ajayi, Antonio Di Maio, Torsten Braun, and Dimitrios Xenakis. An online multi-dimensional knapsack approach for slice admission control. In *2023 IEEE 20th Consumer Communications Networking Conference (CCNC)*, pp. 152–157, 2023.

I. Alkhouri, C. Le Denmat, Y. Li, C. Yu, J. Liu, R. Wang, and A. Velasquez. Differentiable quadratic optimization for the maximum independent set problem. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 1102–1127, 2025.

Ismail Alkhouri, Mian Wu, CUNXI YU, Jia Liu, Rongrong Wang, and Alvaro Velasquez. A scalable lift-and-project differentiable approach for the maximum cut problem. In *The 29th International Conference on Artificial Intelligence and Statistics*, 2026. URL `https://openreview.net/forum?id=ob0TVfqMIg`.

Esra Bas. Surrogate relaxation of a fuzzy multidimensional 0–1 knapsack model by surrogate constraint normalization rules and a methodology for multi-attribute project portfolio selection. *Engineering Applications of Artificial Intelligence*, 25(5):958–970, 2012.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

T. Berthold. Measuring the impact of primal heuristics. *Operations Research Letters*, 41(6):611–614, 2013.

Valentina Cacchiani, Manuel Iori, Alberto Locatelli, and Silvano Martello. Knapsack problems—an overview of recent advances. part ii: Multiple, multidimensional, and quadratic knapsack problems. *Computers & Operations Research*, 143:105693, 2022.

Natalie S Cherbaka and Russell D Meller. Single-plant sourcing decisions with a multidimensional knapsack model. *Journal of Manufacturing Systems*, 27(1):7–18, 2008.

P. C. Chu and J. E. Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63–86, 1998.

A. Çördük, P. Sielski, A. Boucher, and K. Aatish. GPU-accelerated primal heuristics for mixed integer programming, 2025.

M. Courbariaux, Y. Bengio, and J.-P. David. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, 2015.

T. Dockhorn, Y. Yu, E. Sari, M. Zolnouri, and V. Partovi Nia. Demystifying and generalizing BinaryConnect. In *Advances in Neural Information Processing Systems*, 2021.

Trivikram Dokka, Adam N Letchford, and M Hasan Mansoor. Revisiting surrogate relaxation for the multidimensional knapsack problem. *Operations Research Letters*, 50(6):674–678, 2022.

Shengyu Feng and Yiming Yang. Regularized Langevin dynamics for combinatorial optimization. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 16977–16992. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/feng25r.html.

Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003.

Arnaud Fréville. The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, 155(1):1–21, 2004.

Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*, version 13.0 edition, 2026.

S.-P. Han and O. L. Mangasarian. Exact penalty functions in nonlinear programming. *Mathematical Programming*, 17:251–269, 1979.

C.-H. Huang, Y. Sun, C. Huang, I. Alkhouri, and R. Wang. Local minima in quadratic-penalty relaxations of binary linear programs. In *International Conference on Machine Learning*, 2026.

Y. Ichikawa and Y. Arai. Optimization by parallel quasi-quantum annealing with gradient-based sampling. In *International Conference on Learning Representations*, 2025.

Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M. Kakade, and Michael I. Jordan. How to escape saddle points efficiently. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1724–1732. PMLR, 2017.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Murat Kocamaz. Multidimensional knapsack model and fuzzy analytic hierarchy process for project portfolio selection. *Ege Academic Review*, 14(3):493–498, 2014.

Jason D. Lee, Max Simchowitz, Michael I. Jordan, and Benjamin Recht. Gradient descent only converges to minimizers. In *Proceedings of the 29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pp. 1246–1257. PMLR, 2016.

Muheng Li and Ruqi Zhang. Reheated gradient-based discrete sampling for combinatorial optimization. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=uPCvfyr2KP.

P. Lin, M. Zou, and S. Cai. An efficient local search solver for mixed integer programming. In *Principles and Practice of Constraint Programming*, volume 307 of *LIPIcs*, pp. 19, 2024.

B. Luteberget and G. Sartor. Feasibility jump: An LP-free lagrangian MIP heuristic. *Mathematical Programming Computation*, 15:365–388, 2023.

Tessema Mengistu, Dunren Che, and Shiyong Lu. Multi-objective resource mapping and allocation for volunteer cloud computing. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 344–348. IEEE, 2019.

Jakob Puchinger, Günther R Raidl, and Ulrich Pferschy. The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing*, 22(2):250–265, 2010.

Madabhushi Raghavachari. On connections between zero-one integer programming and concave programming under linear constraints. *Operations Research*, 17:680–684, 1969.

Matthew Staib, Sashank Reddi, Satyen Kale, Sanjiv Kumar, and Suvrit Sra. Escaping saddle points with adaptive gradient methods. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 5956–5965. PMLR, 2019.

Yongliang Sun, Ismail Alkhouri, Cheng-Han Huang, Alvaro Velasquez, Susmit Jha, and Rongrong Wang. Mutation-guided differentiable quadratic combinatorial optimization. *arXiv preprint arXiv:2605.06921*, 2026.

N. Wei and J. Liang. GFORS: GPU-accelerated first-order method with randomized sampling for binary integer programs, 2025.

Menglu Yu, Chuan Wu, Bo Ji, and Jia Liu. A sum-of-ratios multi-dimensional-knapsack decomposition for dnn resource scheduling. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pp. 1–10, 2021.

A Proofs and Iteration Details

A.1 Subgradient of the maximum

The function $V(y) = \max_k \mathrm{ReLU}(r_k(y))$ is the maximum of affine functions and zero. At an infeasible z, any row attaining the maximum gives a valid subgradient. At a feasible z, the constant zero function is active, so zero is a valid subgradient of V. Adding the profit and quadratic terms gives equation 5.

A.2 Proof of Theorem 1

Let $x \in [0,1]^n$ be infeasible. Set

$$\delta = V(x) > 0, \qquad \rho = 1 + \delta, \qquad y = x/\rho.$$

For every row k,

$$\frac{a_k^\top x}{b_k} \le \rho \quad \Longrightarrow \quad \frac{a_k^\top y}{b_k} \le 1,$$

so $y \in P$. This argument only uses $b > 0$ and does not require $A \ge 0$.

For $0 < t < \delta/\rho$,

$$V((1-t)x) = \delta - t\rho.$$

Therefore

$$\begin{aligned} F((1-t)x) - F(x) &= t\left[q^\top x - \lambda\rho - \frac{\beta}{n}\mathbf{1}^\top x + \frac{2\beta}{n}\|x\|_2^2\right] - \frac{\beta}{n}t^2\|x\|_2^2 \\ &\le t\rho\left[\left(q + \frac{\beta}{n}\mathbf{1}\right)^\top y - \lambda\right] - \frac{\beta}{n}t^2\|x\|_2^2 \\ &\le t\rho(\lambda_0 - \lambda) - \frac{\beta}{n}t^2\|x\|_2^2. \end{aligned} \tag{9}$$

The first inequality uses $\|x\|_2^2 \le \mathbf{1}^\top x$. If $\beta > 0$ and $\lambda \ge \lambda_0$, the last line is negative. Hence no infeasible point is a local minimizer. If $\beta = 0$, the same conclusion holds for $\lambda > \lambda_0$.

Now assume $\beta > 0$ and let y be a local minimizer. The first part gives $y \in P$. On P,

$$F(y) = -q^\top y + \frac{\beta}{n}\sum_j y_j(1 - y_j),$$

which is strictly concave. If y were not a vertex of P, there would be a nonzero direction h such that $y \pm th \in P$ for all sufficiently small $t > 0$. Then

$$\frac{F(y+th) + F(y-th)}{2} = F(y) - \frac{\beta}{n}t^2\|h\|_2^2 < F(y).$$

At least one nearby point has a smaller value, a contradiction. Thus y is a vertex of P.

Let

$$\mathcal{F} = \{j : 0 < y_j < 1\}$$

and let $\mathcal{J}$ index the active capacity constraints at y. If $A_{\mathcal{J},\mathcal{F}}$ did not have full column rank, a nonzero null vector would give a feasible line segment through y. This contradicts that y is a vertex. Hence

$$|\mathcal{F}| = \mathrm{rank}(A_{\mathcal{J},\mathcal{F}}) \le \mathrm{rank}(A) \le m.$$

A.3 Adam recurrence

Let u_t and v_t be the first and second moments. Operations below are coordinatewise except for R and projection. With $u_0 = v_0 = 0$,

$$z_t = R(x_t), \qquad g_t = g(z_t), \tag{10}$$

$$u_{t+1} = \mu_1 u_t + (1-\mu_1)g_t, \qquad v_{t+1} = \mu_2 v_t + (1-\mu_2)g_t^2, \tag{11}$$

$$\widehat{u}_{t+1} = \frac{u_{t+1}}{1-\mu_1^{t+1}}, \qquad \widehat{v}_{t+1} = \frac{v_{t+1}}{1-\mu_2^{t+1}}, \tag{12}$$

$$x_{t+1} = \Pi_{[0,1]^n}\left(x_t - \alpha\frac{\widehat{u}_{t+1}}{\sqrt{\widehat{v}_{t+1}} + \epsilon}\right). \tag{13}$$

Each trajectory has its own moments. A change in z_t does not reset them.

A.4 ROUNDED-STATE CHANGES

For a fixed binary vector z, define

$$d_\beta(z) = \max_j (2z_j - 1)g_j(z), \qquad C_z = \{x \in [0,1]^n : R(x) = z\}. \tag{14}$$

This scalar records the signed coordinate gradients. The first Adam moment can initially have a different sign from the current gradient, so the proof also bounds the time until it follows the current gradient.

Lemma 1 (Rounded-state changes)**.** *Assume Adam uses a constant learning rate $\alpha > 0$, moment factors $0 \le \mu_1, \mu_2 < 1$, and $\epsilon > 0$. Ties in the definition of $k(z)$ use a fixed deterministic rule. The moments are initialized at zero at the start of each search. For a fixed binary vector z,*

$$\begin{aligned} d_\beta(z) > 0 &\implies \textit{every visit to } C_z \textit{ ends in finitely many updates}, \\ d_\beta(z) \le 0 &\implies \textit{a search started in } C_z \textit{ with zero moments stays in } C_z, \\ d_\beta(z) < 0 &\implies \textit{that zero-moment search reaches } z \textit{ in finitely many updates}. \end{aligned}$$

For fixed z, λ, and selected row,

$$d_{\beta_2}(z) = d_{\beta_1}(z) - \frac{\beta_2 - \beta_1}{n}. \tag{15}$$

Equation 5 gives

$$|g_j(z_t)| \le H, \qquad H = \max_j \left(q_j + \frac{\beta}{n} + \lambda \max_k \frac{|a_{kj}|}{b_k} \right). \tag{16}$$

Zero initial moments imply $|u_{t,j}| \le H$ and $0 \le \widehat{v}_{t,j} \le H^2$.

Suppose $d_\beta(z) > 0$. Choose j with

$$s = 2z_j - 1, \qquad s g_j(z) = \delta = d_\beta(z) > 0.$$

As long as a visit starting at time t stays in C_z, the gradient is constant. After r updates,

$$u_{t+r,j} = g_j(z) + \mu_1^r \big(u_{t,j} - g_j(z)\big), \qquad s u_{t+r,j} \ge \delta - 2H\mu_1^r.$$

For $0 < \mu_1 < 1$, set

$$K(\delta) = \max\left\{1, \left\lceil \frac{\log(4H/\delta)}{-\log \mu_1} \right\rceil \right\}.$$

For $\mu_1 = 0$, set $K(\delta) = 1$. When $r \ge K(\delta)$,

$$s\widehat{u}_{t+r,j} \ge \delta/2, \qquad \frac{\alpha s \widehat{u}_{t+r,j}}{\sqrt{\widehat{v}_{t+r,j}} + \epsilon} \ge \frac{\alpha\delta}{2(H+\epsilon)}.$$

This step points toward the rounding boundary, whose distance is at most $1/2$. Projection onto the box cannot stop the crossing. Hence the visit ends within

$$N(\delta) = K(\delta) + \left\lfloor \frac{H+\epsilon}{\alpha\delta} \right\rfloor + 1 \tag{17}$$

updates.

Now suppose $d_\beta(z) \le 0$ and the search starts in C_z with zero moments. For every j,

$$(2z_j - 1)g_j(z) \le 0.$$

Each first moment has the sign of its constant coordinate gradient. Every step therefore moves toward z_j or is zero. The rounded vector remains z. If $d_\beta(z) < 0$, every coordinate gradient is nonzero. With constant gradients and zero initial moments,

$$\widehat{u}_{t,j} = g_j(z), \qquad \widehat{v}_{t,j} = g_j(z)^2.$$

The step magnitude $\alpha|g_j(z)|/(|g_j(z)| + \epsilon)$ is positive, so projection reaches each z_j in finitely many updates. Finally,

$$(2z_j - 1)(1 - 2z_j) = -1$$

gives equation 15.

The radial estimate used in Theorem 1 also gives a consequence for the rounded iteration. For an infeasible binary z, let $\rho = 1 + V(z)$ and $y = z/\rho \in P$. The selected row satisfies $a_{k(z)}^\top z / b_{k(z)} = \rho$. Thus

$$\begin{aligned}\sum_{j:z_j=1} g_j(z) &= \lambda\rho - \left(q + \frac{\beta}{n}\mathbf{1}\right)^\top z \\ &= \rho\left[\lambda - \left(q + \frac{\beta}{n}\mathbf{1}\right)^\top y\right] \\ &\geq \rho(\lambda - \lambda_0).\end{aligned}$$

Therefore $\lambda > \lambda_0$ implies $d_\beta(z) > 0$. Combined with Lemma 1, an infeasible rounded state cannot remain unchanged forever under this sufficient penalty condition. This statement does not give a time bound for reaching feasibility.

A.5 First moment after a rounded change

Suppose a coordinate enters a new rounding region with first moment $M > 0$ and then receives a constant gradient $-h$, where $h > 0$. After r updates,

$$u_r = \mu_1^r M - (1 - \mu_1^r)h.$$

For $0 < \mu_1 < 1$,

$$u_r < 0 \quad \Longleftrightarrow \quad \mu_1^r < \frac{h}{M+h} \quad \Longleftrightarrow \quad r > \frac{\log(h/(M+h))}{\log \mu_1}.$$

For $M = h$ and $\mu_1 = 0.9$, the smallest integer is $r = 7$. The second moment and bias correction change the step size but not its sign.

A.6 Initialization

Each continuous start is sampled independently from $[0.30, 0.33]^n$. Thus every initial rounded vector is zero and feasible. The continuous states differ across the batch before the first update. This initialization is used in the main experiments unless stated otherwise.

B Experiment Details and Full Results

B.1 Timing and feasibility checks

For MAX-ReLU, the primary search clock starts after explicit host-to-device data transfer, generation of the continuous starts, and allocation of the Adam state. It includes gradient updates, rounding, GPU feasibility checks, candidate ranking, transfer of an improving candidate to the CPU, and exact feasibility validation before storage. Matrix multiplication uses `float32` with TF32 disabled. Every stored solution passes an `int64` feasibility check on the CPU.

The setup-charged sensitivity reported in Appendix B.2 uses the same recorded incumbent stream but adds the measured excluded setup duration from that particular run to each event time. The correction is therefore performed per instance and per run; we do not subtract or add a global median offset. Target attainment is then recomputed with a strict 60 second total-time horizon, so an event shifted beyond 60 seconds is treated as unreached.

The 20 minute Gurobi reference uses one call to the solver. Environment and model construction are outside the search clock. Callbacks record objective improvements during that call. Gurobi uses seed 12 and eight threads. The Gurobi (`MIPFocus`=1) trajectory is the first 60 seconds of this same call. Reported 60 second values and target times use events recorded no later than 60 seconds.

For cuOPT, Python-side construction of the data model and solver settings is excluded. The incumbent callback wall clock starts immediately before `Solve`. Any device transfer, model processing, presolve, heuristic work, and search performed internally by that call are therefore included. We do not infer an internal transfer duration or subtract one from the recorded clock.

The original default-cuOPT recorder also contained a fallback that could append a returned final solution, if it improved on every callback incumbent, using cuOPT's returned engine duration. We audited all 180 default-cuOPT trajectories used in the main grid and the $n = 10{,}000$ extension. No trajectory contained such an engine-timestamped final event. Thus the reported default-cuOPT results are determined entirely by the `Solve`-entry wall-clock incumbent trajectory.

The primary cuOPT experiments use the solver's default presolve behavior. Appendix B.2 reruns the $n = 10{,}000$ instances with `presolve`=0 using the same `Solve`-entry wall-clock convention. This is a solver-setting sensitivity rather than a shifted or postprocessed version of the default trajectory. The presolve-disabled sensitivity uses the same wall-clock origin for callback incumbents; if the returned final solution contributes an additional improvement, it is timestamped on that same wall clock after return and exact validation.

B.2 cuOPT PRESOLVE AND TIMING SENSITIVITY

We perform the timing sensitivity on the 12 positive $n = 10{,}000$ instances used in Section 5.5: $m/n \in \{0.12, 0.36, 0.72\}$ with four instance seeds per regime. For each instance, three records are compared: (i) the original default-cuOPT trajectory under the primary `Solve`-entry clock; (ii) a new cuOPT run with `presolve`=0 under the same clock; and (iii) the original MAX-ReLU trajectory with its measured excluded setup time charged to every event. All 12 presolve-disabled cuOPT runs used here were completed without GPU-contamination flags.

Table 5 reports reach count, conditional median first-hit time, and PAR1 at the 1% and 0.5% targets. The presolve-disabled runs are genuine solver reruns rather than a constant shift of the default trajectory.

Table 5: Timing sensitivity on the positive $n = 10{,}000$ instances. Each solver cell reports reach count, conditional median first-hit time, and PAR1, in that order. Times are seconds. MAX-ReLU+setup charges each run its measured excluded setup time. cuOPT-`p0` denotes a new cuOPT run with `presolve`=0.

m/n	Target	MAX-ReLU+setup	cuOPT default	cuOPT-`p0`
0.12	1%	4/4, 1.422, 1.432	4/4, 19.558, 19.311	4/4, 6.279, 6.291
0.12	0.5%	4/4, 1.443, 1.445	4/4, 19.701, 19.699	4/4, 6.279, 6.291
0.36	1%	4/4, 2.736, 2.703	4/4, 38.559, 38.765	4/4, 17.360, 17.339
0.36	0.5%	4/4, 2.766, 2.778	4/4, 39.824, 40.230	4/4, 19.103, 19.159
0.72	1%	4/4, 4.674, 4.658	0/4, –, 60.000	4/4, 34.478, 34.584
0.72	0.5%	4/4, 5.249, 5.205	0/4, –, 60.000	4/4, 37.444, 36.995

Disabling presolve changes cuOPT substantially on these instances. At the 0.5% target, its conditional median decreases from 19.701 to 6.279 seconds at $m/n = 0.12$ and from 39.824 to 19.103 seconds at $m/n = 0.36$. At $m/n = 0.72$, default cuOPT reaches none of the four targets, whereas cuOPT-`p0` reaches all four with conditional median 37.444 seconds. The setup-charged MAX-ReLU medians at the same target are 1.443, 2.766, and 5.249 seconds.

Table 6 reports the corresponding 60 second objectives. Disabling presolve recovers the dense $m/n = 0.72$ cuOPT run from 27.613% to 99.871% of the reference, but it gives slightly lower final objective than default cuOPT at $m/n = 0.12$ and 0.36. Thus `presolve`=0 is an important large-instance sensitivity, but it is not uniformly dominant over the default setting across these regimes.

B.3 FULL MAIN GRID

The grid contains three sizes, seven values of m/n, and eight instance seeds per setting. Table 1 in the main text reports all 60 second quality values. The tables below retain the corresponding reach counts and conditional-median first-hit times for the 1% and 0.5% targets. The main text adds PAR1 for the primary $n = 2000$ slice.

Table 6: Solution quality at 60 seconds in the $n = 10{,}000$ timing sensitivity. Values are mean objective divided by the mean 20 minute Gurobi (`MIPFocus=1`) reference in each regime.

m/n	MAX-ReLU+setup	cuOPT default	cuOPT-`p0`
0.12	99.967%	99.877%	99.855%
0.36	99.942%	99.852%	99.838%
0.72	99.911%	27.613%	99.871%

Tables 7–9 give the 1% and 0.5% target times. Gurobi is faster when the number of constraints is small. MAX-ReLU becomes faster as m/n grows, with the point of change depending on n and the target.

Table 7: $n = 500$ time to the 1% and 0.5% targets. Each regime has eight seeds. Bold marks the best result and underline marks the second best. Reach count is ranked before time.

	MAX-ReLU		Gurobi MIPFocus=0		Gurobi MIPFocus=1		cuOPT		Gurobi NoRel 60 s	
m/n	1%	0.5%	1%	0.5%	1%	0.5%	1%	0.5%	1%	0.5%
0.06	8/8@0.165	8/8@0.180	8/8@0.025	8/8@0.060	**8/8@0.024**	**8/8@0.048**	8/8@0.623	8/8@2.676	8/8@1.361	8/8@3.299
0.12	8/8@0.146	8/8@0.223	**8/8@0.031**	**8/8@0.138**	8/8@0.048	8/8@0.176	8/8@1.018	8/8@7.204	8/8@0.723	8/8@1.360
0.24	8/8@0.155	**8/8@0.237**	**8/8@0.057**	8/8@0.268	8/8@0.084	8/8@0.273	8/8@1.748	8/8@2.957	8/8@0.984	8/8@2.369
0.36	8/8@0.160	**8/8@0.230**	**8/8@0.153**	8/8@0.355	**8/8@0.153**	8/8@0.272	8/8@2.208	8/8@9.234	8/8@0.875	8/8@3.997
0.48	**8/8@0.166**	**8/8@0.244**	8/8@0.225	8/8@0.485	8/8@0.200	8/8@0.493	8/8@1.519	6/8@8.496	8/8@1.127	8/8@3.369
0.60	**8/8@0.171**	**8/8@0.347**	8/8@0.290	8/8@0.418	8/8@0.259	8/8@0.741	8/8@2.650	6/8@24.757	8/8@1.405	8/8@3.849
0.72	**8/8@0.173**	**8/8@0.308**	8/8@0.338	8/8@0.516	8/8@0.331	8/8@0.634	8/8@3.065	6/8@10.020	8/8@1.107	8/8@5.014

Table 8: $n = 1000$ time to the 1% and 0.5% targets. Each regime has eight seeds. Bold marks the best result and underline marks the second best. Reach count is ranked before time.

	MAX-ReLU		Gurobi MIPFocus=0		Gurobi MIPFocus=1		cuOPT		Gurobi NoRel 60 s	
m/n	1%	0.5%	1%	0.5%	1%	0.5%	1%	0.5%	1%	0.5%
0.06	8/8@0.246	8/8@0.285	**8/8@0.044**	**8/8@0.058**	8/8@0.045	8/8@0.060	8/8@0.661	8/8@1.421	8/8@2.214	8/8@6.590
0.12	8/8@0.232	8/8@0.283	**8/8@0.088**	**8/8@0.119**	8/8@0.089	8/8@0.121	8/8@0.958	8/8@4.707	8/8@0.756	8/8@1.501
0.24	8/8@0.217	**8/8@0.314**	**8/8@0.175**	8/8@0.425	**8/8@0.175**	8/8@0.410	8/8@1.469	8/8@5.927	8/8@1.264	8/8@3.349
0.36	**8/8@0.226**	**8/8@0.307**	8/8@0.278	8/8@0.657	8/8@0.268	8/8@0.664	8/8@2.035	8/8@11.923	8/8@1.725	8/8@5.414
0.48	**8/8@0.226**	**8/8@0.382**	8/8@0.374	8/8@0.911	8/8@0.361	8/8@0.839	8/8@2.314	8/8@19.088	8/8@2.682	8/8@8.450
0.60	**8/8@0.235**	**8/8@0.389**	8/8@0.478	8/8@1.126	8/8@0.474	8/8@1.127	8/8@2.146	8/8@24.061	8/8@3.646	8/8@11.242
0.72	**8/8@0.236**	**8/8@0.406**	8/8@0.574	8/8@1.419	8/8@0.592	8/8@1.555	8/8@2.698	8/8@15.192	8/8@3.559	8/8@12.872

Table 9: $n = 2000$ time to the 1% and 0.5% targets. Each regime has eight seeds. Bold marks the best result and underline marks the second best. Reach count is ranked before time.

	MAX-ReLU		Gurobi MIPFocus=0		Gurobi MIPFocus=1		cuOPT		Gurobi NoRel 60 s	
m/n	1%	0.5%	1%	0.5%	1%	0.5%	1%	0.5%	1%	0.5%
0.06	8/8@0.339	8/8@0.376	**8/8@0.174**	**8/8@0.180**	8/8@0.179	8/8@0.184	8/8@1.255	8/8@1.484	8/8@2.037	8/8@12.545
0.12	8/8@0.338	**8/8@0.389**	8/8@0.363	8/8@1.230	**8/8@0.321**	8/8@0.462	8/8@1.746	8/8@2.171	8/8@2.835	8/8@5.662
0.24	**8/8@0.376**	**8/8@0.473**	8/8@0.749	8/8@0.809	8/8@0.759	8/8@0.823	8/8@3.319	6/8@3.686	8/8@2.580	8/8@13.497
0.36	**8/8@0.435**	**8/8@0.664**	8/8@1.242	8/8@1.526	8/8@1.216	8/8@1.471	8/8@3.905	7/8@4.507	8/8@2.695	8/8@11.282
0.48	**8/8@0.496**	**8/8@0.712**	8/8@1.849	8/8@2.098	8/8@1.783	8/8@1.949	8/8@4.785	8/8@5.428	8/8@3.219	8/8@14.369
0.60	**8/8@0.547**	**8/8@0.883**	8/8@2.184	8/8@2.603	8/8@2.153	8/8@2.537	8/8@6.061	8/8@7.632	8/8@3.295	8/8@15.428
0.72	**8/8@0.658**	**8/8@1.042**	8/8@2.743	8/8@3.140	8/8@2.527	8/8@2.914	8/8@7.077	7/8@7.731	8/8@3.613	8/8@17.778

MAX-ReLU reaches both targets on all 168 instances. Its 60 second win/tie/loss counts are $157/5/6$ against Gurobi (`MIPFocus=0`), $150/4/14$ against Gurobi (`MIPFocus=1`), $167/0/1$ against cuOPT, $161/0/7$ against NoRel, and $168/0/0$ against Local-MIP. These counts compare objective ordering. Table 1 gives the size of the differences.

B.4 GFORS, PQQA, AND MORE CUOPT SETTINGS

GFORS uses its `GPUModel` defaults with `device=cuda` and a 60 second time limit. PQQA uses the QQA4CO implementation with its default profile, a 60 second budget, `device=cuda`, seed 12, and `exact_backend=none`.

Table 10 reports the 63-instance subset used for GFORS and PQQA. MAX-ReLU has a higher 60 second objective than each method on all 63 paired instances. GFORS reaches none of the 5% targets.

PQQA reaches 41 of 63 at 5%, 14 at 2%, and two at 1%. These counts describe final checked quality rather than exact first-hit time.

Table 10: GFORS and PQQA on the 63-instance subset. Obj./target is the median per-instance ratio to the 20 minute Gurobi (`MIPFocus`=1) target. Target columns count final solutions within the stated gap. W/T/L compares each method with MAX-ReLU on the same instances.

Method	N	Median obj./target	5% reach	2% reach	1% reach	W/T/L vs MAX-ReLU
GFORS	63	92.099%	0/63	0/63	0/63	0/0/63
PQQA	63	96.041%	41/63	14/63	2/63	0/0/63

For cuOPT, we also compare default MILP with its heuristics-only mode on nine $n = 2000$ instances at $m/n \in \{0.24, 0.48, 0.72\}$. At the 1% target, default cuOPT reaches $9/9$ in a median of 5.896 seconds, while heuristics-only reaches $7/9$ in 5.544 seconds. At the 2% target, both reach $9/9$, with medians 4.853 and 4.968 seconds. We retain default cuOPT as the fixed primary configuration for the complete comparison grid. The separate $n = 10{,}000$ timing study in Section 5.5 examines the more favorable early-time behavior obtained by disabling presolve; because that setting was introduced as a targeted sensitivity after the primary comparison and is not uniformly better in final objective, it is not substituted retrospectively for the full-grid baseline. Complete timing results for this sensitivity are given in Appendix B.2.

B.5 Parameter sensitivity

The sweeps vary one parameter at a time on $n = 2000$, with $m/n \in \{0.12, 0.36, 0.72\}$ and three seeds each. The defaults are $\lambda = 0.3$, $\beta = 0.05$, and $\alpha = 0.005$. These seeds overlap the main grid. The default was set before the sweeps, which study sensitivity to each parameter.

Let U be the smaller 60 second Gurobi upper bound from the default and `MIPFocus`=1 runs. For a solution of value p, the percentage bound gap is $100(U - p)/U$, and the target at relative gap η is $p \geq (1 - \eta)U$. These bounds are different from the 20 minute feasible primal reference used in the main comparison.

Table 11: Parameter sweeps. Median is the conditional first-hit median among reached runs. PAR1 is the mean capped time with non-hits assigned 60 seconds. The target reference here is the 60 second Gurobi upper bound U, not the main 20 minute feasible reference.

Axis	Value	Median bound gap	1% reach	1% median	1% PAR1	0.5% reach	0.5% median	0.5% PAR1
β	0	0.762%	21/21	1.854	2.232	4/21	5.447	50.879
β	0.05	0.468%	21/21	0.597	0.633	15/21	2.818	24.545
β	0.20	0.594%	21/21	20.911	22.189	5/21	24.211	52.502
λ	0.03	35.241%	0/21	–	60.000	0/21	–	60.000
λ	0.10	35.241%	0/21	–	60.000	0/21	–	60.000
λ	0.30	0.468%	21/21	0.597	0.633	15/21	2.818	24.545
λ	0.33 (Theorem 1 bound)	0.465%	21/21	0.697	0.730	15/21	3.679	25.348
λ	1.00	0.484%	21/21	1.813	1.849	12/21	6.090	34.013
λ	3.00	0.518%	21/21	4.531	4.836	9/21	11.915	40.943
lr	0.0025	0.461%	21/21	0.733	0.755	14/21	5.754	26.962
lr	0.0050	0.468%	21/21	0.597	0.633	15/21	2.818	24.545
lr	0.0100	0.450%	21/21	0.611	0.592	14/21	3.148	26.587

The two time summaries need not select the same setting. For the 1% target, $\alpha = 0.005$ has the smaller conditional median, while $\alpha = 0.010$ has the smaller PAR1. The 0.5% target favors $\alpha = 0.005$ among these three learning rates.

B.6 Batch size

The update rate falls from about 724 per second at $B = 512$ to about eight at $B = 262144$. The final objective changes little for moderate batch sizes, then decreases as the number of completed updates falls. The conditional median is smallest at $B = 512$, while PAR1 is smallest at $B = 1024$. More parallel starts therefore do not always improve a run under a fixed time limit.

Table 12: $n = 2000$ subset, 21 instances, 60 second budget. Median is the conditional first-hit median for the 1% target. PAR1 is the corresponding mean capped time. Memory is peak GPU allocation from the run record.

Batch	Median bound gap	1% reach	1% median	1% PAR1	Peak GiB
512	0.470%	21/21	0.336	0.392	0.064
1024	0.469%	21/21	0.344	0.373	0.105
2048	0.460%	21/21	0.406	0.435	0.188
4096	0.468%	21/21	0.597	0.633	0.355
8192	0.462%	21/21	1.264	1.287	0.677
16384	0.477%	21/21	2.818	2.950	1.338
32768	0.492%	21/21	5.835	5.914	2.639
65536	0.516%	21/21	11.531	11.584	5.257
131072	0.582%	21/21	21.370	22.303	10.494
262144	0.750%	19/21	40.165	43.929	20.966