B^3-PWL: GPU-Batched Branch-and-Bound for Piecewise-Linear Optimization with SOS2 Constraints

Yilin Guan [1] **Shuqing Luo** [2] **Pingzhi Li** [2] **Tianlong Chen** [2,†] **Kaidi Xu** [3,4†]

[1]Johns Hopkins University [2]University of North Carolina, Chapel Hill
[3]City University of Hong Kong [4]The Hong Kong Institute of AI for Science

Abstract

Piecewise-linear (PWL) optimization problems arise in many mixed-integer programming (MIP) optimization applications, including portfolio optimization, workforce scheduling, and resource allocation. But solving them to global optimality remains computationally expensive because branch-and-bound repeatedly solves LP relaxation subproblems. Existing solvers are largely CPU-centric, leaving the scalability of modern GPUs underutilized. Few prior GPU-accelerated branch-and-bound frameworks either target neural network which is not suitable for general PWL optimization, or accelerate only auxiliary subroutines such as strong branching heuristics within CPU-centric MIP solvers. To bridge this gap, we propose B^3-PWL, a GPU-centric batched branch-and-bound framework for piecewise-linear optimization with Special Ordered Set of type 2 (SOS2) constraints. Our method solves batches of LP relaxation subproblems concurrently on the GPU using a first-order primal-dual solver, enabled by a specialized batched block-tiled sparse matrix kernel. To complement bound computation, we further introduce a unified feasibility search module that combines an SOS2 repair primal heuristic with a batched feasibility pump to rapidly obtain feasible incumbents and improve pruning efficiency. On a benchmark of 43 PWL-MIP instances, B^3-PWL achieves a 9.25$\times$ geometric-mean speedup over NVIDIA cuOpt while reaching high-quality feasible incumbents on every tested instance. On a public valve-point unit-commitment benchmark, it further outperforms NVIDIA cuOpt and the open-source CPU solvers SCIP and HiGHS, demonstrating the potential of first-order LP methods as the central engine of GPU-accelerated branch-and-bound.

1 Introduction

Combinatorial optimization with piecewise-linear (PWL) structure underlies many real-world decision-making tasks, including portfolio optimization [Lobo et al., 2007], workforce scheduling [Ernst et al., 2004], and large-scale resource allocation [Karimi et al., 2003]. In mixed-integer programming (MIP), PWL formulations are commonly used to approximate nonlinear relationships while preserving compatibility with optimization frameworks. However, solving such problems to global optimality remains computationally expensive, because the integer constraints induce a combinatorial search space that grows exponentially with problem size. The standard branch-and-bound (B&B) method tackles this by recursively partitioning the search space and solving an LP relaxation at every node it explores, accumulating thousands to millions of LP solves on a single in-

[†]Corresponding authors. Contact: yguan29@jh.edu, tianlong@cs.unc.edu, kaidixu@cityu.edu.hk.

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

stance [Land and Doig, 2009, Nemhauser and Wolsey, 1988]. In conventional MIP solvers, these LP relaxations are handled by simplex-based methods such as dual simplex [Dantzig, 1990], whose basis re-optimization and sequential pivoting are difficult to parallelize. As a result, although modern GPUs have utilized massive parallel computing cores to revolutionize large-scale machine learning, the same accelerated computing potential remains largely unexploited for PWL optimization problems.

GPU-accelerated branch-and-bound has been studied in two adjacent settings. The first line of work targets neural network verification, where frameworks such as $\alpha\beta$-CROWN Xu et al. [2021], Wang et al. [2021] run B&B almost entirely on the GPU. Their efficiency relies on two properties specific to neural network verification. First, neural networks have a layered structure that supports GPU-friendly bound propagation methods such as LiRPA Xu et al. [2020], which compute linear lower bounds on the network output layer by propagating linear relaxations through the layers, avoiding LP solves at each node. Second, verification only asks whether the worst-case network output violates a safety property, not what its exact value is. Each LP subproblem therefore does not need to be solved exactly: a loose lower bound is enough to prune or further branch the subproblem. This leads to efficient pruning that drastically reduces the number of nodes explored in search tree. However, PWL optimization is more general in structure and it requires solving each LP subproblem precisely to get the globally minimal objective value. Obtaining tight enough bounds and preventing the search tree from exploding are therefore both substantially harder in PWL optimization.

The second line of work uses GPU-accelerated LP solving as an auxiliary component of MIP solvers. NVIDIA cuOpt NVIDIA [2025], Blin et al. [2026] apply GPU-batched first-order LP methods to specific subroutines such as full strong branching (FSB) [Achterberg et al., 2005] and optimization-based bound tightening (OBBT) Gleixner et al. [2017]. However, the LP relaxation at each node of the main search tree is still solved on the CPU using simplex-based methods such as dual simplex. The GPU here only accelerates expensive auxiliary procedures rather than the core B&B loop.

In this paper, we propose `B`^{3}`-PWL`, a GPU-batched branch-and-bound framework for piecewise-linear optimization problems with Special Ordered Set of type 2 (SOS2) constraints. Unlike prior work that relegates GPU LP solving to auxiliary subroutines, `B`^{3}`-PWL` puts batched LP relaxations of subproblems on the GPU as the central compute primitive of the B&B loop. To support this paradigm, we extend the first-order method cuPDLPx Lu et al. [2025] to concurrently solve multiple node-specific LP relaxations within a single batched dispatch, backed by a custom block-tiled sparse matrix kernel that handles the ragged constraint matrices. In addition, we introduce a unified feasibility search module that integrate an SOS2 repair primal heuristic [Keha et al., 2006] and batched feasibility pump strategy tightly and seamlessly into our solving pipeline. It effectively and rapidly repairs and finds feasible solutions to update the best incumbent and tighten the global upper-bound for efficient search tree pruning.

Our contributions are as follows:

- We present `B`^{3}`-PWL`, a GPU-batched branch-and-bound framework for piecewise-linear optimization with SOS2 constraints, in which routine node LP relaxations are batched and solved on the GPU via a modified first-order solver for batched LP problems.
- We develop a unified feasibility search module combining an SOS2 repair primal heuristic and a batched feasibility pump to rapidly obtain feasible incumbents for efficient pruning.
- On a 43-instance benchmark, `B`^{3}`-PWL` achieves a 9.25$\times$ geometric-mean speedup and obtain higher quality solutions over cuOpt. We further evaluate on a public valve-point unit-commitment benchmark, where `B`^{3}`-PWL` outperforms cuOpt, and open-source CPU solvers SCIP and HiGHS, highlighting the potential of first-order LP methods for GPU-accelerated branch-and-bound.

2 Related Work

First-order methods for linear programming. Simplex algorithm [Dantzig, 1990] and interior-point methods [Wright, 1997, Karmarkar, 1984] have long dominated linear programming solving. While these algorithms remain the formidable foundation of state-of-the-art CPU solvers, their reliance on matrix factorizations and sequential computations limits their scalability on GPUs. To unlock GPU acceleration for large-scale optimization, first-order methods (FOMs) have emerged as

a scalable alternative. PDLP [Applegate et al., 2021, 2025] reformulates LP as a saddle-point problem and solve it via primal-dual hybrid gradient (PDHG) iterations [Chambolle and Pock, 2011] with a range of practical enhancements, which transforms FOMs into a practical, numerically stable paradigm for large-scale LPs. To leverage the massive memory bandwidth of modern accelerators, cuPDLP [Lu and Yang, 2025, Lu et al., 2023] extends the CPU-based PDLP to GPUs by offloading core sparse matrix computations and introducing hardware-aware heuristics. cuPDLPx [Lu et al., 2025] further accelerates this paradigm by tailoring the algorithm, achieving multi-fold speedups over its predecessor on large-scale benchmarks. Nevertheless, these works focus primarily on accelerating individual LP solves, rather than integrating batched LP computation into B&B.

GPU-accelerated branch-and-bound. GPU acceleration of B&B has been particularly successful in neural network verification, where $\alpha\beta$-CROWN [Xu et al., 2021, Wang et al., 2021] pioneered this problem by combining branch-and-bound with GPU-accelerated linear bound propagation (e.g., LiRPA) [Xu et al., 2020] and tightening loose bounds with batched gradient descent on GPUs. In traditional mixed-integer programming (MIP), however, the use of GPUs within exact LP-based branch-and-bound remains more limited. Early efforts incorporating GPUs, such as Neural Branching [Nair et al., 2020] uses a GPU-scalable ADMM-based variant of full strong branching to generate offline imitation-learning targets for branching policies. More recently, Blin et al. [2026] extended batched primal-dual hybrid gradient (PDHG) [Chambolle and Pock, 2011] methods to execute online, exact LP batches on GPUs. However, their application is limited to accelerating specific heuristic sub-routines, including full strong branching (FSB) [Achterberg et al., 2005] and optimization-based bound tightening (OBBT) [Gleixner et al., 2017]. In contrast, we use batched GPU LP solving as the core engine for routine node processing in exact B&B for PWL optimization with SOS2 constraints.

3 Preliminaries

3.1 PWL Problem Formulation with SOS2 Constraints

We consider an optimization problem in variables $x \in \mathbb{R}^n$, where the first J components $\{x_j\}_{j=1}^J$ are linked to PWL functions. Each PWL function $f_j(x_j)$ is defined over the interval $[\ell_j, u_j]$ with $T_j + 1$ ordered breakpoints $\{d_{j,0}, d_{j,1}, \ldots, d_{j,T_j}\}$, partitioning the domain into T_j linear segments with corresponding values $f_j(d_{j,i})$. Using the λ-formulation with Special Ordered Sets of type 2 (SOS2), we formulate the problem as:

$$\min_{x,z,\lambda} \quad c^\top x + e^\top z \tag{1}$$

$$\text{s.t.} \quad Ax + Gz \le b, \quad \ell \le x \le u,$$

$$\sum_{i=0}^{T_j} \lambda_{j,i} = 1, \quad \forall\, j \in [J], \tag{2}$$

$$x_j = \sum_{i=0}^{T_j} d_{j,i}\, \lambda_{j,i}, \quad z_j = \sum_{i=0}^{T_j} f_j(d_{j,i})\, \lambda_{j,i}, \quad \forall\, j \in [J], \tag{3}$$

$$\lambda_{j,i} \ge 0, \quad \forall\, i, j,$$

$$\text{SOS2}(\lambda_{j,\cdot}), \quad \forall\, j \in [J].$$

Here, $z_j \in \mathbb{R}$ denotes the value of $f_j(x_j)$, and $\lambda_{j,\cdot} = (\lambda_{j,0}, \ldots, \lambda_{j,T_j}) \in \mathbb{R}_+^{T_j+1}$ are convex combination weights. The SOS2 constraint requires that $\lambda_{j,\cdot}$ to have at most two nonzero components at adjacent indices. That is, if $\lambda_{j,i} > 0$ and $\lambda_{j,i'} > 0$ with $i \neq i'$, then $|i - i'| = 1$. This ensures that (x_j, z_j) lies on a single linear segment or exactly on a breakpoint of the PWL graph of f_j.

3.2 Branch and Bound (B&B) for SOS2 Constraints

The combinatorial difficulty of the PWL optimization arises from the SOS2 constraints. Each PWL function f_j admits several candidate active segments, and identifying the correct one for every f_j requires searching over an exponentially large combination of segment choices. Branch-and-bound (B&B) solves this problem by recursively restricting the admissible breakpoint range for each PWL

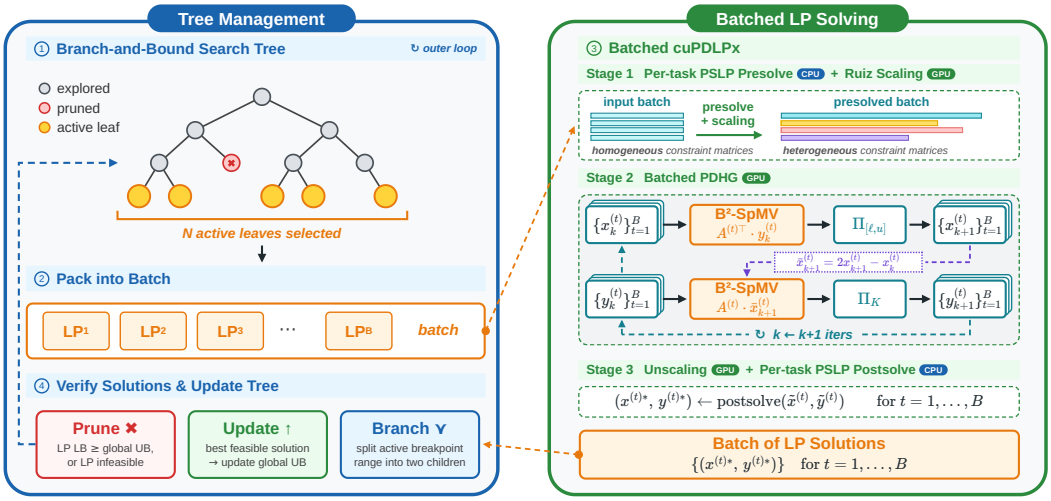


Figure 1: **Overall workflow for B^3-PWL.** Each round alternates between B&B tree management and batched LP solving. The CPU (left) selects N active subproblems from the search tree (①), packs their LP relaxations into a batch (②), and after the solutions are returned, verifies feasibility and updates the tree by pruning, updating the incumbent, or branching (④). The GPU (right) executes a batched cuPDLPx pipeline (③), with only PSLP presolve and postsolve offloaded to the CPU: per-task presolve and Ruiz scaling produce heterogeneous constraint matrices, batched PDHG iterations (built on our B^2-SpMV kernel) run until convergence, and rescaling and postsolve yield primal-dual solutions. The outer loop continues until the active queue is empty or the global gap closes.

function. At the root node, all breakpoints are admissible. Each node $\mathcal{D}$ is characterized by per-function index intervals $[L_j, U_j] \subseteq [0, T_j]$ and adds the restriction

$$\lambda_{j,i} = 0, \quad \forall j \in [J],\ i \notin \{L_j, \ldots, U_j\}. \tag{4}$$

The subproblem at $\mathcal{D}$ is therefore the original SOS2 formulation (1)(3) together with the interval restriction (4). Its LP relaxation, obtained by dropping the SOS2 constraints, provides a lower bound for $\mathcal{D}$. Three cases may arise after the LP problem is solved: (i) if the LP is infeasible or its optimum is no better than the incumbent, $\mathcal{D}$ is fathomed; (ii) if the LP solution satisfies all SOS2 constraints, it updates the incumbent if improved and $\mathcal{D}$ is fathomed; (iii) otherwise, $\mathcal{D}$ is branched by splitting one active range into two children. The algorithm terminates when the active queue is empty or the global gap falls within tolerance.

4 Method

We introduce B^3-PWL, a GPU-batched branch-and-bound framework for piecewise-linear optimization with SOS2 constraints. In our framework, batched LP solving is the central compute primitive: the CPU organizes the global search state and assembles task batches, while the GPU executes batches of LP tasks and returns primal and dual solutions. We describe the end-to-end workflow in Section 4.1, then present the batched node-LP solver in Section 4.2, the block-tiled sparse backend for heterogeneous matrices in Section 4.3, and the unified feasibility-search module in Section 4.4.

4.1 Overall Workflow

Different node subproblems within a single PWL B&B tree descend from a common root formulation and differ only in their variable bounds (the active breakpoint ranges), which is naturally suited to batching. Exploiting this, we extend the GPU LP solver cuPDLPx [Lu et al., 2025] to concurrently solve multiple node LP relaxations within a single batched dispatch.

Building on this batched solver, our framework proceeds in rounds of three stages. (1) In the *assembly* stage, the CPU selects active nodes from the active nodes queue according to the current B&B policy and packs their LP relaxations into a single GPU batch. (2) In the *solve* stage, the GPU solves all LP relaxations in the batch in parallel through batched matrix operations, and returns their primal-dual solutions. (3) In the *postprocess* stage, the CPU uses the returned LP solutions to update

the global lower and upper bounds, prune fathomed nodes, and branch unresolved nodes into new subproblems that are enqueued into the active node queue. Beyond these standard B&B operations, we further exploit the LP solutions through a unified feasibility-search module which aims at recovering high-quality feasible incumbents that tighten the upper bound (Section 4.4). The next round then assembles its batch from the updated queue, and the loop continues until the active queue is empty or the global gap falls within tolerance.

4.2 Heterogeneous Batched LP Engine on GPU

Each batch consists of B tasks $\mathcal{T} = \{1, \ldots, B\}$, each corresponding to one node LP relaxation. SOS2 constraints are dropped, and the active breakpoint range $[L_j^{(t)}, U_j^{(t)}]$ is encoded through the fixings $\lambda_{j,i}^{(t)} = 0$ for $i \notin [L_j^{(t)}, U_j^{(t)}]$. The objective $c^{(t)}$ is either the original node objective for regular B&B search tasks or a feasibility-pump objective for feasibility tasks (see Section 4.4). Each task is independently transformed by task-local presolve and Ruiz rescaling into the reduced-space boxed LP form

$$\min_{x^{(t)}} \; c^{(t)\top} x^{(t)} \quad \text{s.t.} \quad \ell_A^{(t)} \le A^{(t)} x^{(t)} \le u_A^{(t)}, \qquad \ell_x^{(t)} \le x^{(t)} \le u_x^{(t)}, \tag{5}$$

yielding reduced matrices $A^{(t)}$ that differ across tasks in dimensions and sparsity patterns. Unlike the prior batched LP engine for heuristic subroutines in cuOpt, where approximate estimation allows presolve to be skipped so that all tasks share a common constraint matrix A, our setting requires presolve and therefore motivates a dedicated batched SpMV implementation for heterogeneous matrices $A^{(t)}$ (Section 4.3).

Building on cuPDLPx, our engine batches its inner PDHG operator across all B tasks in a single GPU dispatch, while task-local stepsizes, primal weights, projections, restart states, and stopping criteria are maintained independently. The surrounding reflected Halpern updates and adaptive-restart logic follow cuPDLPx and are described in Appendix E. The inner PDHG operator is dominated by two sparse matrix–vector products.

Let $\mathcal{X}^{(t)} = [\ell_x^{(t)}, u_x^{(t)}]$ and $\mathcal{S}^{(t)} = [\ell_A^{(t)}, u_A^{(t)}]$, and let $X_k = \{x_k^{(t)}\}_{t=1}^B$ and $Y_k = \{y_k^{(t)}\}_{t=1}^B$ denote the batch-wise primal and dual iterates at PDHG iteration k. For each task t, the inner PDHG operator takes the form

$$x_{k+1}^{(t)} = \Pi_{\mathcal{X}^{(t)}}\left(x_k^{(t)} - \tau_k^{(t)}\left(c^{(t)} - A^{(t)\top} y_k^{(t)}\right)\right), \tag{6}$$

$$\bar{x}_{k+1}^{(t)} = 2x_{k+1}^{(t)} - x_k^{(t)}, \tag{7}$$

$$y_{k+1}^{(t)} = y_k^{(t)} - \sigma_k^{(t)} A^{(t)} \bar{x}_{k+1}^{(t)} - \sigma_k^{(t)} \Pi_{-\mathcal{S}^{(t)}}\left((\sigma_k^{(t)})^{-1} y_k^{(t)} - A^{(t)} \bar{x}_{k+1}^{(t)}\right). \tag{8}$$

Here, $\tau_k^{(t)} = \eta_k^{(t)} / \omega_k^{(t)}$ and $\sigma_k^{(t)} = \eta_k^{(t)} \omega_k^{(t)}$ are the task-local primal and dual stepsizes induced by the cuPDLPx stepsize $\eta_k^{(t)}$ and primal weight $\omega_k^{(t)}$. The row-bound proximal operator projects onto $-\mathcal{S}^{(t)} = [-u_A^{(t)}, -\ell_A^{(t)}]$. Full details of cuPDLPx, including the reflected Halpern scheme, adaptive restarting, and primal-weight updates, are provided in Appendix E.

4.3 Accelerating Ragged SpMV with Batched Blocking

A key challenge in our approach is that different B&B nodes may produce LP relaxations with different numbers of active variables and constraints after presolve, delivering a *ragged* collection of sparse matrices $\{A^{(t)}\}_{t=1}^B$ with heterogeneous dimensions. Standard GPU-accelerated sparse linear algebra libraries such as cuSPARSE NVIDIA Corporation [2024] provide no batched SpMV primitives. Sequentially launching an SpMV kernel per subproblem fails to fully utilize the GPU parallel resources, while flattening the batch into a single block-diagonal sparse matrix leads to poor load balance and inefficient indexing overhead. To support

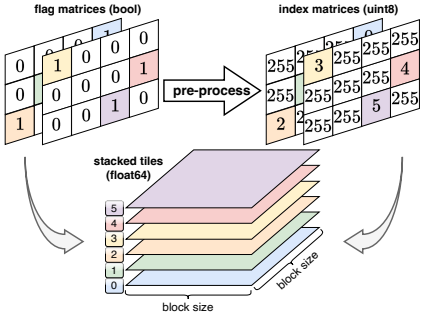


Figure 2: **Building stacked tiles for B²-SpMV.**

efficient batching on such heterogeneous workloads, we introduce a *batched blocking* scheme for efficient and parallel solving, dubbed `B²-SpMV`. It partitions each sparse matrix $A^{(t)}$ into fixed-size tiles, removes all-zero tiles, and packs the remaining nonzero tiles from all subproblems into a single contiguous GPU tensor. For each subproblem, we build a tile index map that identifies, for every tile position, either the location of the corresponding nonzero tile in the packed tensor or an empty marker, enabling both sparse computing for each LP sub-problem and coalesced memory access for efficient GPU execution. `B²-SpMV` executes both $A^{(t)}x^{(t)}$ and $A^{(t)\top}y^{(t)}$ with a single batched kernel over the entire ragged batch, where each thread block processes one tiled row or column for one subproblem, with empty tiles skipped based on the index map.

4.4 Unified Feasibility Search for Efficient Upper-Bound Tightening

Efficient pruning in B&B depends on tightening either the lower bounds for each node or the global upper bound from the best feasible incumbent. Therefore, high-quality incumbents are critical for pruning by bounding and help drastically reducing the search space. However, LP solutions typically violate SOS2 constraints. We therefore introduce a feasibility-search module that attempts to convert LP primal solutions into feasible incumbents through two-stage procedure: a fast and direct SOS2 repair primal heuristic, followed by a batched feasibility pump when direct repair fails.

SOS2 repair primal heuristic. Consider an LP solution $(\hat{x}, \hat{z}, \hat{\lambda})$ at node $\mathcal{D}$. We attempt to repair it into an SOS2-feasible one by re-interpolating $\hat{\lambda}_{j,\cdot}$ for each PWL function. The idea is to use $\hat{x}_j$ to identify the active segment it lies on and reset $\hat{\lambda}_{j,\cdot}$ to the convex-combination weights of that segment's two endpoints.

Concretely, for each PWL function j with active breakpoint range $[L_j, U_j]$, we clip $\hat{x}_j$ to $[d_j^{L_j}, d_j^{U_j}]$, locate the current segment $k_j \in \{L_j, \dots, U_j - 1\}$ with $d_j^{k_j} \le \hat{x}_j \le d_j^{k_j+1}$, and reset $\tilde{\lambda}_{j,\cdot}$ as the convex-combination weights of $d_j^{k_j}$ and $d_j^{k_j+1}$:

$$\tilde{\lambda}_j^{k_j} = \frac{d_j^{k_j+1} - \hat{x}_j}{d_j^{k_j+1} - d_j^{k_j}}, \qquad \tilde{\lambda}_j^{k_j+1} = \frac{\hat{x}_j - d_j^{k_j}}{d_j^{k_j+1} - d_j^{k_j}},$$

with $\tilde{\lambda}_j^k = 0$ for all other $k \neq k_j, k_j + 1$. The repaired outputs follow as $\tilde{x}_j = \sum_k d_j^k \tilde{\lambda}_j^k$ and $\tilde{z}_j = \sum_k f_j(d_j^k)\, \tilde{\lambda}_j^k$, yielding a candidate $(\tilde{x}, \tilde{z}, \tilde{\lambda})$ that satisfies SOS2 by construction.

Since the repaired candidate satisfies SOS2 by construction, it remains to verify the original linear constraints $A\tilde{x} + G\tilde{z} \le b$ and variable bounds. If satisfied, $(\tilde{x}, \tilde{z}, \tilde{\lambda})$ is feasible for the original PWL problem and updates the incumbent upper bound.

Batched Feasibility-Pump (FP). When first-stage repair fails, the repaired vector $\bar{\lambda}^{(r)}$ still encodes which SOS2 segment each $\hat{x}_j$ most naturally suggests. The idea of objective feasibility pump (OFP) Achterberg and Berthold [2007] is to treat $\bar{\lambda}^{(r)}$ as a *soft target* and re-solve the LP under a combined objective that balances the original PWL minimization objective with proximity to $\bar{\lambda}^{(r)}$, so that successive LP solves are pulled toward an SOS2-feasible point while remaining close to optimality, until repair eventually succeeds.

We adapt OFP to our pipeline in two ways: the objective is tailored to the SOS2 formulation, and FP tasks are integrated into the same GPU batch with our B&B search LP tasks. Specifically, each pump round solves the same node LP with the original constraints intact but a modified objective:

$$c_{\mathrm{FP}}^{(r)} = \alpha_r\, c_{\mathrm{prox}}^{(r)} + (1 - \alpha_r)\, \gamma_r\, c_{\mathrm{orig}}.$$

where $c_{\mathrm{prox}}^{(r)}$ has entries $(c_{\mathrm{prox}}^{(r)})_{j,k} = -\bar{\lambda}_{j,k}^{(r)}$ on the λ block and zero elsewhere, so that minimizing it rewards λ-mass on the two breakpoints selected by repair, thus pulls the LP solution toward the SOS2 target $\bar{\lambda}^{(r)}$. c_{orig} is the original objective matrix and keeps the solution close to optimal. $\alpha_r \in [0, 1]$ trades off feasibility against quality, and the rescaling factor $\gamma_r = \|c_{\mathrm{prox}}^{(r)}\|_2 / (\|c_{\mathrm{orig}}\|_2 + \varepsilon)$ aligns their ℓ_2 magnitudes so α_r remains effective. We set $\alpha_0 = 1$ and decay $\alpha_{r+1} = 0.9\,\alpha_r$ on each failure.

Since FP tasks differ from search tasks only in their objective vectors and share the same constraint structure, our batched cuPDLPx solver, which supports per-task objectives, can dispatch them in the same GPU batch as regular B&B search tasks at no additional scheduling cost. After each FP solve, the returned primal is repaired and checked: accepted candidates update the incumbent, otherwise the repaired point becomes $\bar{\lambda}^{(r+1)}$ for the next round.

5 Experiments

5.1 Experimental Setup

Benchmarks. Public PWL-MIP instances that keep SOS2 constraints in native form are limited (detailed in Appendix I), *e.g.* MIPLIB 2017, the most widely used MIP benchmark library, contains no instance with SOS or piecewise-linear constraints, even though the MPS format supports them [Gleixner et al., 2021]. Recent PWL work also relies on its own instance generators [Hübner et al., 2026]. We therefore evaluate `B`^{3}`-PWL` on two benchmarks.

The first is a suite of 43 synthetic PWL-MIP instances from three application families: portfolio optimization, workforce scheduling, and resource allocation. The suite contains 18 portfolio, 9 workforce, and 16 resource-allocation instances, with 100–1,200 PWL functions and 15–25 breakpoints per function. We can control LP size, breakpoint count, sparsity, and SOS2 density through generator to characterize the behavior of our GPU B&B framework across different scales and structures. Appendix I reports the formulations, generator parameters, seeds, and per-instance size statistics.

The second is the public valve-point unit commitment benchmark of Pedroso et al. [2014] built on real-world cost data. This benchmark considers thermal unit commitment, where the objective is to determine unit commitment and generation schedules that satisfy multi-period demand while minimizing valve-point fuel costs.

Hardware Settings. All end-to-end solver comparisons are run on the same Blackwell node, equipped with NVIDIA RTX PRO 6000 Blackwell Server Edition GPUs (96 GB each) and a dual-socket x86-64 host with 192 physical cores (384 logical threads) and 718 GB RAM.

Solver Configurations. All solvers are run with a 1% relative MIP gap and a 300-second wall-clock limit. `B`^{3}`-PWL` uses a PDLP feasibility/optimality tolerances of 10^{-4}. The feasibility-pump module is enabled in the main configuration. cuOpt is run with v26.02.00 which combines CPU-side dual simplex for node LPs, PaPILO presolve, MIR / Gomory / knapsack cutting planes, and GPU-accelerated batched LP for FSB and OBBT.

Metrics. Our evaluation focuses on two aspects: solution quality and solve speed.

Solution quality. we use Gurobi as an external oracle reference. We run Gurobi with tight tolerance gap ($= 10^{-6}$) and denote its reference objective by $z^\star_{\text{ref}}$. We evaluate solvers on (1) *Exact Match* counts instances on which the solver returns the objective value that exactly matches the solution of Gurobi. (2) ***Eff-Opt*** counts instances on which the solver's objective value z_{solver} satisfying

$$\frac{|z_{\text{solver}} - z^\star_{\text{ref}}|}{\max\{1, |z^\star_{\text{ref}}|\}} \leq 10^{-2}.$$

Thus, Eff-Opt corresponds to finding a feasible incumbent within 1% of the Gurobi reference objective.

Solve time. All solvers terminate upon reaching a 1% MIP gap, exhausting the active search queue, or hitting the 300 s wall-clock budget, whichever comes first; total time is the wall-clock time from solver start to termination. For aggregate runtimes, we report *shifted geometric means* with shift $s = 1$ s following standard MIP benchmarking practice [Achterberg, 2007, 2009]:

$$\text{GeoTime}(t) = \exp\left(\frac{1}{n}\sum_{i=1}^{n}\log(t_i + s)\right) - s.$$

The shift reduces the influence of trivially easy instances and avoids overweighting sub-second tim-

Table 1: Solution quality and solve time of B^3-PWL vs. cuOpt on 43 PWL-MIP instances.

		Eff-Opt		Exact		Arith. Time (s)		Geo. Time (s)		Geo-Spd	B^3-PWL
Category	n	B^3-PWL	cuOpt	B^3-PWL	cuOpt	B^3-PWL	cuOpt	B^3-PWL	cuOpt	(cu/B^3-PWL)	Wins
Portfolio	18	**18**	10	**15**	0	2.5	142.2	2.1	57.0	**29.35**×	**18/18**
Workforce	9	**9**	8	**9**	1	2.6	68.9	2.2	35.7	17.69×	**9/9**
Resource Alloc.	16	**16**	12	**13**	8	82.3	107.9	39.0	67.3	1.75×	11/16
Total	43	**43**	30	**37**	9	32.2	114.1	**7.0**	55.0	**9.25**×	**38/43**

Table 2: Comparison with established CPU MIP solvers on the same 43 instances.

Solver	Eff-Opt	Shifted Geo. Time (s)
B^3-PWL	**43/43**	7.0
Gurobi 13.0.2	**43/43**	1.0
SCIP 10.0.1	37/43	27.1
HiGHS 1.15.1	40/43	25.2
cuOpt 26.02	30/43	55.0

ing differences. *Arithmetic means* report the average of wall-clock times. *Geo-Spd* is the geometric mean of per-instance speedup ratios:

$$\mathrm{GeoSpd}(\mathrm{cuOpt}/\mathrm{B}^3\text{-PWL}) = \exp\left(\frac{1}{n}\sum_{i=1}^{n}\log\frac{t_i^{\mathrm{cuOpt}}}{t_i^{\mathrm{B}^3\text{-PWL}}}\right).$$

with values > 1 indicating B^3-PWL is faster on average. We additionally report *Time to Best Incumbent (TTB)*: the wall-clock time at which a solver first reaches the incumbent it ultimately returns, censored at 300 s for runs that fail to reach Eff-Opt.

5.2 Main Results

Table 1 reports the performance comparisons between B^3-PWL and cuOpt across our PWL benchmark. Detailed per-instance objectives and wall-clock times are provided in Appendix G.

Solution quality. B^3-PWL reaches Eff-Opt on all 43 instances, compared with 30 / 43 for cuOpt, and matches the Gurobi reference objective (Exact Match) on 37 / 43 instances, compared with 9 / 43 for cuOpt. These results indicate that B^3-PWL consistently returns higher-quality feasible incumbents than cuOpt on our benchmarks.

Solve speed. In total wall-clock time, B^3-PWL achieves a shifted geometric mean of 7.0 seconds, compared with 55.0 seconds for cuOpt, and an arithmetic mean of 32.2 seconds, compared with 114.1 seconds for cuOpt. Under the reported Geo-Spd metric, this corresponds to a 9.25× aggregate speedup. We say B^3-PWL *wins* on an instance if it reaches Eff-Opt while cuOpt does not, or if both reach Eff-Opt and B^3-PWL terminates faster. Under this criterion B^3-PWL wins on 38 / 43 instances.

Comparison with established CPU solvers. Table 2 places these results in the context of mature CPU MIP solvers. Gurobi remains the fastest solver on this benchmark, reaching Eff-Opt on all 43 instances with a shifted geometric-mean time of 1.0 s, compared with 7.0 s for B^3-PWL. B^3-PWL nevertheless reaches Eff-Opt on all 43 instances and achieves substantially lower aggregate runtime than the open-source CPU solvers SCIP and HiGHS, which reach Eff-Opt on 37/43 and 40/43 instances, respectively. These results position B^3-PWL as a proof of concept that batched first-order LP solving can serve as the core engine of GPU-centric branch-and-bound, rather than as a replacement for mature commercial MIP solvers.

5.3 Public valve-point UC benchmark

We further evaluate B^3-PWL on the public valve-point unit commitment benchmark of Pedroso et al. [2014]. Since valve-point costs are nonconvex and nonsmooth, the original paper solves the problem through an adaptive piecewise-linear approximation, yielding a sequence of PWL-MIP instances.

Table 3: Time to a 1% relative MIP gap (s) on the valve-point UC benchmark. Rows 1-5 sweep the horizon T at 20 breakpoints; rows 5-7 sweep breakpoints at $T = 24$. Geo-Spd is the geometric mean of per-instance ratios against `B3-PWL`.

Instance	`B3-PWL`	Gurobi	SCIP	HiGHS	cuOpt
$T = 1$	1.13	4.02	0.12	0.06	3.33
$T = 3$	3.47	0.45	37.8	211.4	245.9
$T = 6$	2.12	0.61	21.1	11.0	10.9
$T = 12$	2.92	1.19	36.4	14.2	209.0
$T = 24$, bp=20	5.93	3.04	>300	39.9	>300
$T = 24$, bp=15	5.73	2.36	>300	34.6	>300
$T = 24$, bp=30	9.02	3.95	>300	254.0	>300
Geo-Spd	—	0.47×	10.34×	5.13×	25.40×

Table 4: Time to best incumbent (TTB) on the 43-instance benchmark. Geo. TTB: shifted geometric mean ($s = 1$ s); Geo-Spd: geometric mean of per-instance cuOpt/`B3-PWL` TTB ratios.

		Geo. TTB (s)			Arith. TTB (s)	
Category	n	`B3-PWL`	cuOpt	Geo-Spd	`B3-PWL`	cuOpt
Portfolio	18	2.08	55.70	28.66×	2.5	141.9
Workforce	9	2.15	35.23	17.56×	2.5	68.4
Resource Alloc.	16	11.53	42.48	3.87×	60.4	96.0
Total	43	**4.22**	45.77	**12.28**×	**24.1**	109.4

We use the public 40-unit `eld40`/`ucp40` data with the original horizons $T \in \{1, 3, 6, 12, 24\}$. Unlike the adaptive breakpoint-refinement procedure of the original paper, we fix each cost curve to a static breakpoint count (20 unless noted) to obtain deterministic solver-comparison instances.

As shown in Table 3, Gurobi remains fastest and we do not claim otherwise. Against the open-source simplex-based solvers B^3-PWL is faster on 6 of 7 instances (geomean 10.3× over SCIP, 5.1× over HiGHS). The exception is $T = 1$, too small to amortize GPU setup. The margin widens with horizon length. At $T = 24$, SCIP and cuOpt exhaust the budget at every breakpoint setting while `B3-PWL` certifies the gap in under 10 s. Final relative gaps and best objective values for all solvers are reported in Appendix C.

5.4 Time to Best Incumbent

Total wall-clock time conflates two phases of branch-and-bound: finding a high-quality feasible incumbent and closing the final dual-bound gap. We separate them by reporting Time to Best Incumbent (TTB) (defined in Section 5.1), censored at 300 s for runs that do not reach Eff-Opt. Since `B3-PWL` reaches Eff-Opt on every instance, its TTB is observed in all 43 cases. cuOpt's TTB is censored on the 13 instances where it fails to reach Eff-Opt within the budget.

Table 4 reports the comparison. Across all 43 instances, `B3-PWL` achieves a 12.28× geometric-mean TTB speedup over cuOpt overall (4.22 s vs. 45.77 s shifted geomean), and a strictly lower TTB on every instance. Per-category speedups follow the same pattern as total runtime in Table 1: 28.66× on portfolio and 17.56× on workforce, narrowing to 3.87× on resource allocation, where cuOpt finds incumbents quickly via dual simplex.

Comparing Table 4 with the total solve times in Table 1 reveals `B3-PWL`'s end-to-end advantage stems primarily from rapid primal progress: high-quality incumbents are typically found within the first few GPU batches, after which most of the remaining wall-clock budget is spent on dual-bound tightening to certify optimality. This profile suggests that further end-to-end gains will come from accelerating dual-side convergence such as tighter LP relaxations.

5.5 Unified Feasibility Search Module Ablation

We ablate the two stages of the feasibility-search module in Section 4.4, the SOS2 repair primal heuristic and the batched Feasibility Pump (FP), under three configurations: (i) no feasibility heuristic, (ii) repair only, and (iii) repair + FP. Table 5 reports solve time across the 43-instance benchmark.

Table 5: Ablation of the feasibility-search module on the 43-instance benchmark. *Feas* reports instances on which a feasible incumbent is recorded; configurations (ii) and (iii) achieve 43/43 on every category. The last column reports the runtime ratio (ii)/(iii); values > 1 mean FP accelerates over repair alone.

		(i) No heuristic		(ii) Repair only		(iii) Repair + FP		FP Speedup
Category	n	Feas	Geo. (s)	Geo. (s)	Arith. (s)	Geo. (s)	Arith. (s)	(ii)/(iii)
Portfolio	18	0/18	300.0	4.97	36.2	**2.09**	**2.5**	**2.18**×
Workforce	9	0/9	300.0	6.10	37.2	**2.17**	**2.6**	**2.61**×
Resource Alloc.	16	0/16	39.8	38.98	82.3	38.98	82.3	1.00×
Total	43	0/43	142.1	11.56	53.6	**7.05**	**32.2**	**1.69**×

SOS2 repair primal heuristic is necessary for PDLP outputs. Unlike simplex-based solvers, which return vertex solutions that naturally satisfy integrality at leaf nodes, PDHG iterates in PDLP converge from the interior of the LP polytope. Its postsolved primal iterate may satisfy the LP relaxation within global tolerance while still leaving small nonzero mass on SOS2 coefficients that should be exactly zero. Consequently, even leaf nodes whose branching restrictions identify a single admissible SOS2 segment may fail the strict incumbent check. Column (i) of Table 5 confirms this empirically: no feasible incumbent is recorded on any of the 43 instances. The SOS2 repair step resolves the mismatch by projecting each PWL component onto an admissible segment. With repair enabled (column (ii)), feasibility is recovered on all 43 instances at low overhead (Table 7) in Appendix D).

FP provides category-dependent acceleration. The Feasibility Pump injects proximity-objective LP tasks into the GPU batch to accelerate discovery of high-quality incumbents. Its benefit depends on whether early incumbent quality is the bottleneck for pruning. On portfolio and workforce instances, where the first high-quality incumbent dominates total solve time, FP yields geometric-mean speedups of $2.18\times$ and $2.61\times$ respectively over repair alone. On resource allocation, however, the branching heuristic already discovers incumbents cheaply at nearby leaves, so FP yields no measurable acceleration ($1.00\times$).

5.6 Accelerating Ragged SpMV with Batched Blocking

We benchmark the efficiency of `B`^{2}`-SpMV` with varied batch size, where the size of the constraint matrix per sample is randomly generated within $8000 \sim 25000$. We use the default SpMV interface from cuSPARSE as baseline, with looped execution for batching. It scales with batch size and saturates beyond $50\times$ at larger batches, confirming that the kernel effectively amortizes per-task launch and memory overhead across the batch.

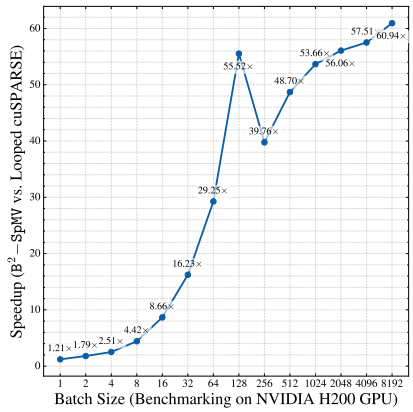


Figure 3: **Speedup Visualization for `B`^{2}`-SpMV` vs. Looped cuSPARSE.**

6 Conclusion

We introduced `B`^{3}`-PWL`, a GPU-batched branch-and-bound framework for PWL optimization with SOS2 constraints that solves batched node LP relaxations on the GPU as the main engine of the B&B loop. On a 43-instance benchmark, `B`^{3}`-PWL` achieves a $9.25\times$ geometric-mean speedup over NVIDIA cuOpt while reaching high-quality incumbents on every tested instance. We see this work as a first step toward using batched first-order method for LP as the core engine of B&B, beyond its current role as auxiliary acceleration. Extending the paradigm to broader PWL and MIP settings, including the incorporation of cutting planes for tighter dual bounds and more general branching heuristics for efficient tree exploration, remains an open direction for future work.

Acknowledgments and Disclosure of Funding

This work received no specific third-party funding. The authors declare no competing interests.

References

T. Achterberg. *Constraint integer programming*. PhD thesis, 2007.

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

T. Achterberg and T. Berthold. Improving the feasibility pump. *Discrete Optimization*, 4(1):77–86, 2007.

T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33 (1):42–54, 2005.

D. Applegate, M. Díaz, O. Hinder, H. Lu, M. Lubin, B. O'Donoghue, and W. Schudy. Practical large-scale linear programming using primal-dual hybrid gradient. *Advances in Neural Information Processing Systems*, 34:20243–20257, 2021.

D. Applegate, M. Díaz, O. Hinder, H. Lu, M. Lubin, B. O'Donoghue, and W. Schudy. Pdlp: A practical first-order method for large-scale linear programming. *arXiv preprint arXiv:2501.07018*, 2025.

N. Blin, S. Gualandi, C. Maes, A. Lodi, and B. Stellato. Batched first-order methods for parallel lp solving in mip. *arXiv preprint arXiv:2601.21990*, 2026.

A. Chambolle and T. 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.

G. B. Dantzig. *Origins of the simplex method*, page 141151. Association for Computing Machinery, New York, NY, USA, 1990. ISBN 0201508141. URL `https://doi.org/10.1145/87252.88081`.

A. T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: A review of applications, methods and models. *European journal of operational research*, 153(1):3–27, 2004.

A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. Christophel, K. Jarck, T. Koch, J. Linderoth, et al. Miplib 2017: data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3):443–490, 2021.

A. M. Gleixner, T. Berthold, B. Müller, and S. Weltge. Three enhancements for optimization-based bound tightening. *Journal of Global Optimization*, 67(4):731–757, 2017.

T. Hübner, A. Gupte, and S. Rebennack. Spatial branch-and-bound for nonconvex separable piecewise linear optimization. *INFORMS Journal on Computing*, 38(2):645–675, 2026.

B. Karimi, S. F. Ghomi, and J. Wilson. The capacitated lot sizing problem: a review of models and algorithms. *Omega*, 31(5):365–378, 2003.

N. Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311, 1984.

A. B. Keha, I. R. de Farias Jr, and G. L. Nemhauser. A branch-and-cut algorithm without binary variables for nonconvex piecewise linear optimization. *Operations research*, 54(5):847–858, 2006.

B. Knueven, J. Ostrowski, and J.-P. Watson. On mixed-integer programming formulations for the unit commitment problem. *INFORMS Journal on Computing*, 32(4):857–876, 2020.

A. H. Land and A. 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.

M. S. Lobo, M. Fazel, and S. Boyd. Portfolio optimization with linear and fixed transaction costs. *Annals of Operations Research*, 152(1):341–365, 2007.

H. Lu and J. Yang. cupdlp. jl: A gpu implementation of restarted primal-dual hybrid gradient for linear programming in julia. *Operations Research*, 73(6):3440–3452, 2025.

H. Lu, J. Yang, H. Hu, Q. Huangfu, J. Liu, T. Liu, Y. Ye, C. Zhang, and D. Ge. cupdlp-c: A strengthened implementation of cupdlp for linear programming by c language. *arXiv preprint arXiv:2312.14832*, 2023.

H. Lu, Z. Peng, and J. Yang. cupdlpx: A further enhanced gpu-based first-order solver for linear programming. *arXiv preprint arXiv:2507.14051*, 2025.

V. Nair, S. Bartunov, F. Gimeno, I. Von Glehn, P. Lichocki, I. Lobov, B. O'Donoghue, N. Sonnerat, C. Tjandraatmadja, P. Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*, volume 18. Wiley New York, 1988.

NVIDIA. Nvidia cuopt: GPU accelerated decision optimization, 2025. URL `https://github.com/NVIDIA/cuopt`.

NVIDIA Corporation. *cuSPARSE Library*, 2024. URL `https://developer.nvidia.com/cusparse`. Included in CUDA Toolkit.

J. P. Pedroso, M. Kubo, and A. Viana. Unit commitment with valve-point loading effect. *arXiv preprint arXiv:1404.4944*, 2014.

S. Wang, H. Zhang, K. Xu, X. Lin, S. Jana, C.-J. Hsieh, and J. Z. Kolter. Beta-crown: Efficient bound propagation with per-neuron split constraints for neural network robustness verification. *Advances in neural information processing systems*, 34:29909–29921, 2021.

S. J. Wright. *Primal-dual interior-point methods*. SIAM, 1997.

K. Xu, Z. Shi, H. Zhang, Y. Wang, K.-W. Chang, M. Huang, B. Kailkhura, X. Lin, and C.-J. Hsieh. Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33:1129–1141, 2020.

K. Xu, H. Zhang, S. Wang, Y. Wang, S. Jana, X. Lin, and C.-J. Hsieh. Fast and complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=nVZtXBI6LNn`.

A Broader Impacts

B^3-PWL is a foundational systems contribution: a GPU-accelerated framework for solving piecewise-linear mixed-integer optimization problems to global optimality. It is not tied to any specific deployed application, but PWL-MIP formulations underlie a wide range of decision-making tasks, including portfolio optimization, workforce scheduling, energy and resource allocation, and supply-chain planning. Faster and more accessible exact solvers for this class of problems reduce the computational and energy cost of solving real-world planning problems, and lower the barrier for smaller organizations and academic users who lack access to commercial-grade CPU solvers.

We do not foresee direct negative societal impacts from this work. The framework solves classical optimization problems whose objectives and constraints are specified by the user; it does not learn from data, generate content, or make autonomous decisions. As with any general-purpose optimization tool, downstream impact depends on the application, and we view such considerations as properties of mathematical optimization as a whole rather than risks specific to our contribution.

B Limitations and Future Work

While B^3-PWL demonstrates that batched first-order LP solvers can serve as the central engine of branch-and-bound for PWL optimization, the current framework has several limitations that point to natural directions for future work.

Slow dual-bound certification. First-order methods such as PDHG converge sublinearly in high-accuracy regimes, making dual-bound tightening substantially slower than with simplex-based solvers. Empirically (Section 5.4), B^3-PWL typically locates a high-quality incumbent within the first few GPU batches, but the remaining wall-clock budget is dominated by closing the final dual gap to certify optimality, and on a small number of instances (e.g., the largest resource-allocation cases in Appendix G) the 1% MIP gap is not closed within the 300 s budget despite a near-optimal incumbent being found early. Hybridizing first-order LP solves with simplex- or interior-point-based crossover for high-accuracy phases is a promising direction for narrowing this gap.

No warm starting between subproblems. Simplex-based MIP solvers exploit the structural similarity between parent and child node LPs through dual-simplex warm starting from the parent basis, which often resolves a child node in only a handful of pivots. PDHG, in contrast, lacks a basis representation, and our batched first-order method solvers re-solve each subproblem essentially from scratch despite the high similarity between sibling LPs. Designing warm-start strategies for batched first-order B&B (e.g., transferring iterate scaling, primal weights, or partial restart history across related subproblems) could substantially reduce per-node work and is, to our knowledge, an open problem in the first-order MIP literature.

Simple branching and no cutting planes. PWL optimization is a broad problem class with highly heterogeneous structure across applications. Our current pipeline focuses primarily on GPU-batched node-LP solving and SOS2-aware feasibility search, but it does not yet incorporate the full range of solver machinery used by highly engineered commercial MIP solvers, such as sophisticated branching rules, cut generation, bound tightening, and advanced presolve strategies. Future work could explore branching heuristics that are robust across diverse PWL structures, as well as GPU-compatible cutting-plane and bound-tightening routines that can be integrated into the batched first-order pipeline. By tightening LP relaxations and guiding the search toward more informative regions of the B&B tree, these extensions may substantially improve pruning efficiency, reduce dual-bound certification time, and narrow the gap between first-order GPU-centric B&B and mature commercial solvers.

C Solution Quality on the Valve-Point UC Benchmark

Table 6 reports the final relative gap and the best objective value returned by each solver on the valve-point UC benchmark.

Table 6: Final relative gap (%) and best objective on the valve-point UC benchmark. "n/a": no feasible solution within the budget.

	B^3-PWL		Gurobi		SCIP		HiGHS		cuOpt	
Instance	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.	Gap	Obj.
$T = 1$	0.052	122,356	0.485	122,889	0.258	122,609	0.010	122,305	0.086	122,401
$T = 3$	0.999	293,101	0.990	293,375	0.929	293,394	0.460	292,890	0.876	293,266
$T = 6$	0.544	587,487	0.371	586,675	0.634	588,350	0.288	586,440	0.436	586,981
$T = 12$	0.327	1,192,044	0.673	1,196,353	0.827	1,198,228	0.289	1,191,898	0.905	1,199,171
$T = 24$, bp=20	0.369	2,401,839	0.182	2,397,989	6.781	2,556,152	0.262	2,400,451	5.721	2,538,496
$T = 24$, bp=15	0.345	2,404,897	0.201	2,402,118	6.242	2,547,243	0.233	2,403,133	1.682	2,437,913
$T = 24$, bp=30	0.414	2,398,461	0.189	2,393,431		n/a	0.115	2,391,921	1.894	2,434,827

D Time Breakdown

Table 7 reports the time-weighted breakdown across the 43 instances. Batched LP solving (PDHG loop, task-local PSLP presolve and rescaling) consumes over 85% of B&B time, with the GPU-side PDHG loop alone accounting for 74.0%, while the remaining B&B operations (SOS2 checking, primal heuristics, branching, batch construction, and postprocessing) account for about 15% combined. This pattern holds consistently across all three categories, indicating that B^3-PWL is bound by batched LP solving, dominated by GPU-side PDHG, rather than by tree management.

Table 7: Time-weighted breakdown of B^3-PWL's branch-and-bound phase. Each entry reports $\sum_i t_i^{\text{component}} / \sum_i t_i^{\text{BaB}}$ within the corresponding category.

Component	Portfolio	Workforce	Resalloc	Overall
PDHG loop & GPU setup	**82.1%**	**70.4%**	**73.9%**	**74.0%**
PSLP presolve/rescale	9.1%	15.9%	11.4%	11.3%
SOS2 check & primal heuristic	2.0%	1.0%	6.0%	5.9%
Branch selection & node expansion	4.4%	4.9%	3.7%	3.8%
Batch construction	3.8%	4.7%	3.3%	3.3%
Other postprocessing	0.1%	3.2%	2.0%	2.0%

E cuPDLPx: GPU-Accelerated First-Order LP Solver

Our framework utilizes cuPDLPx [Lu et al., 2025] as the linear programming solver of subproblem relaxations in the B&B search tree, as it is the state-of-the-art, GPU-accelerated first-order solver designed to solve LP problems as an equivalent primal-dual saddle point problem.

Primal-Dual Hybrid Gradient (PDHG) The foundational algorithm of cuPDLPx is the Primal-Dual Hybrid Gradient (PDHG). Given primal variables x, dual variables y, objective vector c, and constraint matrix A, PDHG iteratively updates the primal-dual iterate $z^k = [x^k; y^k]$ by alternating between a primal projection step and a dual update step:

$$x^{k+1} = \text{proj}_{\mathcal{X}}\Big(x^k - \tau(c - A^\top y^k)\Big), \tag{9}$$

$$y^{k+1} = y^k - \sigma A(2x^{k+1} - x^k) - \sigma \text{proj}_{\mathcal{S}}\Big(\sigma^{-1} y^k - A(2x^{k+1} - x^k)\Big), \tag{10}$$

where $\mathcal{X}$ and $\mathcal{S}$ denote the primal and constraint feasible domains respectively, and τ, σ are the primal and dual step sizes. cuPDLPx further reparameterizes τ, σ with a base step size η and a primal weight ω (i.e., $\tau = \eta/\omega$, $\sigma = \eta\omega$). Because operations in (9) and (10) are dominated by matrix-vector multiplications, PDHG are well suited for GPU acceleration.

Algorithmic Enhancements in cuPDLPx To overcome the slow tail convergence of vanilla PDHG algorithm, cuPDLPx augments this base routine with four advanced mechanisms, achieving robust and accelerated convergence for large-scale LPs.

Table 8: Validity and stability of final lower bounds under tighter PDLP feasibility/optimality tolerances. LB shifts are instance-wise relative changes from the 10^{-4} baseline.

PDLP tolerance	Valid finite bounds	Median LB shift	95th-percentile LB shift	Maximum LB shift
10^{-4}	43/43	Baseline	Baseline	Baseline
10^{-5}	43/43	0.00193%	0.167%	0.206%
10^{-6}	43/43	0.00329%	0.371%	0.989%

Reflected Halpern Scheme Instead of standard average schemes, cuPDLPx employs a Reflected Halpern update. It interpolates between an over-relaxed PDHG step and an anchor point z^0:

$$z^{k+1} = \frac{k+1}{k+2}\Big((1+\gamma)\text{PDHG}(z^k) - \gamma z^k\Big) + \frac{1}{k+2}z^0, \tag{11}$$

where $\gamma \in [0,1]$ is the reflection parameter. This mechanism enables larger effective step sizes and strictly bounds the fixed-point residual.

Adaptive Restart The anchor z^0 is periodically dynamically reset. cuPDLPx monitors the fixed-point error metric $r(z) = \|z - \text{PDHG}(z)\|_P$. When specific decay conditions are met (e.g., sufficient decay or lack of local progress), it restarts by resetting z^0 to the current iterate, keeping the search tightly focused on the optimal neighborhood.

Constant Stepsize To maximize parallel efficiency on GPUs, cuPDLPx eschews the sequential adaptive stepsize heuristics typical of CPU solvers. Instead, it adopts a constant stepsize $\eta \approx 1/\|A\|_2$, efficiently approximated via power iteration.

PID-Controlled Primal Weight To balance the progress between primal and dual spaces, the primal weight ω is dynamically regulated. At each restart occurrence, cuPDLPx updates ω using a Proportional-Integral-Derivative (PID) controller based on the logarithmic gap between the primal and dual distances to optimality.

F Numerical Robustness and Stopping-Criterion Sensitivity

We further examine the numerical robustness of `B3-PWL` with respect to the PDLP feasibility/optimality tolerance and the global B&B stopping criterion. Specifically, we evaluate the feasibility and stability of the returned incumbents, the validity and stability of the final lower bounds, and sensitivity to tighter target MIP gaps.

Incumbent feasibility and stability. We rerun all 43 instances with the PDLP feasibility and optimality tolerances tightened from 10^{-4} to 10^{-5} and 10^{-6}. Under all three tolerance settings, every instance returns an incumbent within 1% of the Gurobi reference objective. Relative to the 10^{-4} baseline, the maximum change in the solver-reported incumbent objective is only 0.0213% under the tighter 10^{-5} and 10^{-6} tolerances.

We additionally validate each returned incumbent by fixing its integer/SOS2 pattern and re-solving the remaining continuous LP with HiGHS at a 10^{-9} feasibility tolerance. Across the three PDLP tolerance settings, this yields $43 \times 3 = 129$ incumbent patterns. All 129 patterns yield feasible solutions, with a maximum constraint violation of 2.91×10^{-11}. The maximum objective adjustment introduced by this validation is 0.0180% at 10^{-4}, 0.000457% at 10^{-5}, and 0.000933% at 10^{-6}.

Lower-bound stability and validity. For each tighter PDLP tolerance, we compare the final lower bounds against the 10^{-4} baseline. We also verify

$$LB \le z^\star_{\text{Gurobi}} \le UB_{\text{validated}}$$

for every available finite lower bound, where $UB_{\text{validated}}$ denotes the objective of the independently validated incumbent. Table 8 reports instance-wise relative changes in the final lower bounds with respect to the 10^{-4} baseline.

The final lower bounds remain valid for all 43 instances under each tolerance setting and exhibit only small changes relative to the 10^{-4} baseline.

Table 9: Sensitivity to the global B&B target MIP gap. Incumbent and final-LB shifts are measured relative to the 1% configuration.

Target MIP gap	Eff-Opt	Maximum incumbent shift	Maximum final-LB shift	Valid finite bounds	Time-limit runs
1%	43/43	Baseline	Baseline	43/43	3
0.5%	43/43	0.000217%	0.0989%	43/43	7
0.1%	43/43	0.000217%	0.102%	43/43	12

Sensitivity to the B&B stopping criterion. We further tighten the target relative MIP gap from 1% to 0.5% and 0.1%, while keeping the PDLP tolerance and 300-second wall-clock limit fixed. Table 9 reports incumbent and final-LB shifts relative to the 1% baseline.

Tightening the MIP-gap criterion does not materially change the returned incumbents or final lower bounds. Its primary effect is to increase the cost of dual-bound certification: the number of runs reaching the 300-second limit increases from 3 at a 1% target gap to 12 at a 0.1% target gap. Thus, the main solution-quality conclusions remain stable under stricter stopping criteria, while closing a tighter global optimality gap requires substantially more computation.

G Per-Instance Detailed Results

Tables 10–12 give per-instance wall-clock time and returned objective for PWL-BaB and cuOpt on every canonical instance, alongside the Gurobi certified optimum (MIPGAP$=10^{-6}$, separate 32-thread Xeon CPU run; Gurobi runtime is omitted because it is not directly comparable to the GPU-accelerated PWL/cuOpt times). **TO** marks the 300 s budget exhaustion; **—** marks no feasible incumbent returned. Each instance is tagged `pwl{N}_s{seed}`, where `N` is the number of PWL functions in the instance and `seed` is the random-generator seed. Within resource allocation, two pairs of size groups share the same `N` (`pwl300` and `pwl500`); we disambiguate them with the suffixes `a`/`b`, whose generator parameters are listed in Appendix I.

Table 10: Per-instance results: Portfolio (18 instances).

	PWL-BaB		cuOpt		Gurobi
Instance	Time (s)	Objective	Time (s)	Objective	Objective
pwl100_s42	0.6	-1636.0085	TO	—	-1636.0084
pwl100_s123	0.6	-1591.3689	TO	—	-1591.3686
pwl100_s999	0.5	-1702.2300	TO	—	-1702.2300
pwl200_s42	1.2	-3329.2898	TO	-3103.7171	-3329.2898
pwl200_s123	1.7	-3493.2999	TO	—	-3493.2999
pwl200_s999	1.2	-2820.2763	TO	-2752.9085	-2820.2763
pwl300_s42	1.4	-5132.8143	7.8	-5092.7688	-5132.8144
pwl300_s123	1.3	-5312.8877	8.8	-5270.0022	-5313.4625
pwl300_s999	1.3	-4326.8367	10.1	-4314.2420	-4326.8434
pwl400_s42	2.6	-6729.8250	12.9	-6717.2791	-6729.8211
pwl400_s123	3.0	-6930.5113	12.0	-6908.1034	-6930.5117
pwl400_s999	2.1	-6186.0892	13.8	-6150.4538	-6189.1684
pwl500_s42	5.8	-8793.2572	17.8	-8752.8616	-8793.2572
pwl500_s123	2.9	-8844.0370	18.7	-8822.4380	-8844.0369
pwl500_s999	3.2	-8396.4138	TO	—	-8398.0055
pwl600_s42	3.6	-9917.6555	TO	—	-9918.4741
pwl600_s123	3.4	-10257.0782	27.1	-10242.9353	-10257.0767
pwl600_s999	8.6	-10361.3325	30.3	-10332.3481	-10361.3306

H Implementation Details

Multi-level branching for batch filling. GPU throughput is maximized when all B slots are occupied, but the B&B queue may contain fewer than B nodes, particularly in early iterations. Inspired by the split-depth mechanism in $\alpha\beta$-CROWN [Xu et al., 2021], we preemptively branch nodes mul-

Table 11: Per-instance results: Workforce (9 instances).

	PWL-BaB		cuOpt		Gurobi
Instance	Time (s)	Objective	Time (s)	Objective	Objective
pwl300_s42	0.8	281.7799	12.5	283.5340	281.7799
pwl300_s123	0.8	279.7645	11.6	279.7645	279.7645
pwl300_s999	0.8	277.9249	12.2	279.3438	277.9249
pwl600_s42	2.1	563.5598	27.9	567.8821	563.5598
pwl600_s123	1.8	559.5290	14.8	563.2236	559.5290
pwl600_s999	2.1	555.8498	35.8	559.9900	555.8498
pwl1200_s42	4.8	1121.4303	107	1127.8207	1121.4303
pwl1200_s123	5.5	1115.9629	TO	—	1115.9629
pwl1200_s999	4.2	1121.4227	98.3	1125.9802	1121.4227

Table 12: Per-instance results: Resource Allocation (16 instances).

	PWL-BaB		cuOpt		Gurobi
Instance	Time (s)	Objective	Time (s)	Objective	Objective
pwl300a_s42	64.1	316.2431	28.5	316.2431	316.2431
pwl300a_s123	64.5	316.2431	38.4	316.2431	316.2431
pwl300a_s999	64.0	316.2431	25.4	316.2431	316.2431
pwl300b_s123	43.4	316.2431	23.2	316.2431	316.2431
pwl400_s42	9.0	421.6067	31.0	421.6067	421.6067
pwl400_s123	8.9	421.6067	TO	—	421.6067
pwl400_s999	8.9	421.6067	31.4	425.3778	421.6067
pwl500a_s42	40.0	527.0718	69.0	527.0718	527.0718
pwl500a_s123	40.0	527.0718	37.4	527.0718	527.0718
pwl500a_s999	39.8	527.0718	81.0	527.7716	527.0718
pwl500b_s42	11.3	527.0083	48.2	527.0083	527.0083
pwl500b_s123	11.4	527.0083	60.7	531.4298	527.0083
pwl500b_s999	11.4	527.0083	52.1	531.7667	527.0083
pwl1200_s42	TO	1278.4221	TO	—	1278.1789
pwl1200_s123	TO	1280.2254	TO	—	1280.0844
pwl1200_s999	TO	1287.2700	TO	—	1287.0155

tiple levels before solving. Specifically, when the queue size $|\mathcal{Q}| < 0.9B$, we compute a split depth

$$d_{\text{split}} = \left\lceil \log_2 \frac{0.9B}{\max(1, |\mathcal{Q}|)} \right\rceil, \tag{12}$$

and expand each seed node into up to $2^{d_{\text{split}}}$ leaf nodes via BFS-style recursive branching, using the branching heuristic to select variables at each level. This ensures the GPU batch is consistently saturated.

Adaptive batch sizing. We dynamically adjust B based on GPU memory utilization. After each solving round, we query the available GPU memory and compute a capacity factor $\gamma = M_{\text{target}}/M_{\text{used}}$, where $M_{\text{target}} = 0.85 \cdot M_{\text{total}}$. The batch size is updated as $B \leftarrow \min(2B,\ \lfloor \gamma B \rfloor)$, subject to a minimum increase threshold, ensuring stable growth without out-of-memory failures.

I Benchmark Details

To our knowledge, there is no broadly adopted, MIPLIB-like benchmark suite specifically for large-scale nonconvex PWL optimization that preserves native SOS2 structure. In particular, the MIPLIB 2017 paper reports that, although its input format permits SOS and piecewise-linear constraints, none of the submitted instances used these structures. Existing PWL studies therefore often rely on generated application-oriented families. Recent external studies have released reproducible generators and instances for nonconvex PWL network-flow and knapsack problems, but these remain generated benchmarks rather than a broadly adopted standard [Hübner et al., 2026]. Public application-specific libraries also exist, notably PGLib-UC, which combines convex piecewise-linear production costs

with ordinary binary commitment, startup, and shutdown decisions, and therefore targets a different problem scope [Knueven et al., 2020].

Given this landscape, we construct a controlled systems benchmark to characterize the behavior of our GPU B&B framework, rather than to claim broad industrial representativeness. We selected three classical PWL application classes, portfolio optimization, workforce scheduling, and resource allocation, whose generators allow us to systematically vary LP size, breakpoint count, sparsity, and SOS2 density and thereby characterize the behavior of our GPU B&B framework across different scales and structures. We remove root-trivial instances, since they primarily benchmark a single LP relaxation rather than the batched B&B mechanism studied here. Synthetic generation also allows us to control instance difficulty within an informative range, avoiding cases that are either too easy or too difficult to meaningfully distinguish our framework from CPU simplex-based solvers.

Our benchmark suite spans three problem families that exercise complementary structural regimes for branch-and-bound with batched LP relaxations. The portfolio benchmark (Section I.1) features dense SOS2 coverage—one SOS2 substructure per asset—coupled through a small number of global linear constraints. The workforce scheduling benchmark (Section I.2) features a multi-period grid of SOS2 substructures coupled through skill-coverage and budget constraints. The resource allocation benchmark (Section I.3) concentrates combinatorial structure on a sparse subset of variables under a block-angular linear coupling. All three families share the same univariate PWL cost-curve generator (described in Section I.1) so that differences in solver behavior across families reflect coupling structure rather than curve shape.

I.1 Portfolio Benchmark with Piecewise-Linear Transaction Costs

Problem Formulation. We consider a portfolio optimization benchmark with separable PWL transaction-cost penalties. This problem models capital allocation across N assets under a global budget, sparse factor-risk limits, and sector exposure caps. The objective balances expected return against a per-asset transaction penalty.

For each asset $j \in \{1, \ldots, N\}$, let $w_j \geq 0$ denote the dollar holding and let z_j denote the transaction-cost penalty associated with w_j. The objective is

$$\min_{w,z} \; -\sum_{j=1}^{N} \mu_j w_j + \gamma \sum_{j=1}^{N} z_j, \tag{13}$$

where μ_j is the expected return of asset j, and $\gamma > 0$ controls the trade-off between return and transaction cost.

The portfolio is subject to three classes of linear coupling constraints. First, the total capital invested cannot exceed the budget C:

$$\sum_{j=1}^{N} w_j \leq C. \tag{14}$$

Second, for each risk factor $i \in \{1, \ldots, F\}$, the factor exposure is bounded by a risk budget B_i^{risk}:

$$\sum_{j=1}^{N} \sigma_{i,j} w_j \leq B_i^{\text{risk}}, \tag{15}$$

where $\sigma_{i,j}$ is the loading of asset j on factor i. Third, sector allocations are capped. Let $s(j)$ denote the sector of asset j; then for each sector $s \in \{1, \ldots, S_{\text{sec}}\}$,

$$\sum_{j:\, s(j)=s} w_j \leq B_s^{\text{sec}}. \tag{16}$$

Transaction cost is modeled independently for each asset by a univariate PWL function. For asset j, let $\{(d_{j,k}, \phi_{j,k})\}_{k=1}^{J}$ be the breakpoints of its cost curve, where $d_{j,k}$ is a breakpoint on the holding axis and $\phi_{j,k}$ is the corresponding penalty value. We use the standard SOS2 formulation with convex-

combination variables $\lambda_{j,k} \in [0,1]$:

$$\sum_{k=1}^{J} \lambda_{j,k} = 1, \tag{17}$$

$$w_j = \sum_{k=1}^{J} d_{j,k}\lambda_{j,k}, \tag{18}$$

$$z_j = \sum_{k=1}^{J} \phi_{j,k}\lambda_{j,k}, \tag{19}$$

together with an SOS2 constraint on $\lambda_{j,\cdot}$, so that at most two adjacent breakpoints are active. This enforces that (w_j, z_j) lies on the graph of the PWL penalty function.

The complete formulation is

$$\begin{aligned}
\min_{w,z,\lambda} \quad & -\sum_{j=1}^{N} \mu_j w_j + \gamma \sum_{j=1}^{N} z_j \\
\text{s.t.} \quad & \sum_{j=1}^{N} w_j \le C, \\
& \sum_{j=1}^{N} \sigma_{i,j} w_j \le B_i^{\text{risk}}, \qquad i = 1,\dots,F, \\
& \sum_{j:\, s(j)=s} w_j \le B_s^{\text{sec}}, \qquad s = 1,\dots,S_{\text{sec}}, \\
& \sum_{k=1}^{J} \lambda_{j,k} = 1, \qquad j = 1,\dots,N, \\
& w_j = \sum_{k=1}^{J} d_{j,k}\lambda_{j,k}, \qquad j = 1,\dots,N, \\
& z_j = \sum_{k=1}^{J} \phi_{j,k}\lambda_{j,k}, \qquad j = 1,\dots,N, \\
& \lambda_{j,\cdot} \text{ is SOS2}, \qquad j = 1,\dots,N, \\
& w_j \in [\underline{w}_j, \overline{w}_j], \qquad j = 1,\dots,N.
\end{aligned} \tag{20}$$

Data Generation. The holding interval for each asset is restricted to a tight box around a reference point m_j:

$$[\underline{w}_j, \overline{w}_j] = [m_j - 0.35\,\text{span}_j,\ m_j + 0.35\,\text{span}_j]. \tag{21}$$

This narrow box is chosen deliberately so that no single SOS2 segment is admissible at the LP relaxation, ensuring that branch-and-bound must explore multiple segment combinations rather than terminating at the root LP. The breakpoints $(d_{j,k}, \phi_{j,k})$ are sampled from a univariate U-shaped curve generator that combines a low-volume penalty region, a flat economical mid-range, and a steeply rising tail; the same generator is reused across all three benchmark families to isolate the effect of coupling structure.

Expected returns are sampled independently as $\mu_j \sim \mathcal{U}(0.02, 0.15)$, and the global transaction-cost weight is fixed to $\gamma = 0.005$. Asset-wise scales are drawn as $w_j^{\max} \sim \mathcal{U}(50, 500)$. Assets are assigned to sectors in round-robin order, $s(j) = j \bmod S_{\text{sec}}$. For each factor i, the loading matrix $\sigma_{i,j}$ has approximately 30% of entries nonzero (each entry independently masked with probability 0.7), with nonzero values drawn from $\mathcal{U}(0.01, 0.10)$. Let $w_j^{\max,\text{band}} = \overline{w}_j$ denote the upper end of the tight box. The total budget is set to

$$C = \alpha_C \sum_j w_j^{\max,\text{band}}, \qquad \alpha_C \sim \mathcal{U}(0.6, 0.8).$$

Risk budgets are set by

$$B_i^{\text{risk}} = \max\bigl(\alpha_i\, \sigma_i^\top w^{\max,\text{band}},\, 1\bigr), \qquad \alpha_i \sim \mathcal{U}(0.8, 1.2),$$

and sector budgets by

$$B_s^{\text{sec}} = \alpha_s \sum_{j:\, s(j)=s} w_j^{\max,\text{band}}, \qquad \alpha_s \sim \mathcal{U}(0.75, 0.95).$$

Problem Size. For N assets and J breakpoints per asset, the model contains

$$N \text{ holding variables } + N \text{ cost variables } + NJ \text{ SOS2 weights } = N(J+2)$$

variables in total. The number of global coupling constraints is $1 + F + S_{\text{sec}}$, while the PWL representation contributes $3N$ local constraints. This benchmark is therefore characterized by many homogeneous SOS2 substructures coupled through a relatively small set of sparse linear constraints.

I.2 Workforce Scheduling with Overtime Costs

Problem Formulation. We consider a multi-period workforce scheduling problem with skill coverage and budget constraints. Let $w = 1, \ldots, W$ index workers, $p = 1, \ldots, P$ index periods, and $t = 1, \ldots, T$ index skills. The decision variable $h_{w,p}$ denotes the number of hours assigned to worker w in period p, and $z_{w,p}$ denotes the corresponding labor cost. Each worker has a skill coefficient $\kappa_{w,t} \in [0, 1]$ for skill t, and each period p requires at least $m_{t,p}$ effective hours of each skill t. In addition, each period is subject to a payroll budget B_p.

To model overtime and other nonlinear labor costs, each worker–period pair (w, p) is associated with a piecewise-linear cost function $z_{w,p} = f_{w,p}(h_{w,p})$. We represent this function using J breakpoints $\{(d_{w,p,k}, \phi_{w,p,k})\}_{k=1}^{J}$ and SOS2 variables $\lambda_{w,p,k}$, yielding

$$\sum_{k=1}^{J} \lambda_{w,p,k} = 1, \qquad h_{w,p} = \sum_{k=1}^{J} d_{w,p,k}\lambda_{w,p,k}, \qquad z_{w,p} = \sum_{k=1}^{J} \phi_{w,p,k}\lambda_{w,p,k},$$

together with an SOS2 constraint on $\lambda_{w,p,\cdot}$.

The full optimization problem is

$$\begin{aligned}
\min_{h,z,\lambda} \quad & \sum_{w=1}^{W}\sum_{p=1}^{P} z_{w,p} \\
\text{s.t.} \quad & \sum_{w=1}^{W} \kappa_{w,t} h_{w,p} \geq m_{t,p}, && \forall t, p, \\
& \sum_{w=1}^{W} r_w h_{w,p} \leq B_p, && \forall p, \\
& \sum_{k=1}^{J} \lambda_{w,p,k} = 1, && \forall w, p, \\
& h_{w,p} = \sum_{k=1}^{J} d_{w,p,k}\lambda_{w,p,k}, && \forall w, p, \\
& z_{w,p} = \sum_{k=1}^{J} \phi_{w,p,k}\lambda_{w,p,k}, && \forall w, p, \\
& h_{w,p} \in [\underline{h}_w, \overline{h}_w],\ \lambda_{w,p,k} \in [0,1], && \forall w, p, k, \\
& \text{SOS2}(\lambda_{w,p,1}, \ldots, \lambda_{w,p,J}), && \forall w, p.
\end{aligned}$$

Data Generation. Worker-specific maximum hours are sampled as $h_w^{\max} \sim \mathcal{U}(8, 16)$, wage rates as $r_w \sim \mathcal{U}(15, 50)$, and staffing requirements as $m_{t,p} \sim \mathcal{U}(2, 8)$. The skill matrix $\kappa_{w,t}$ has approximately 40% of entries nonzero, with nonzero entries sampled from $\mathcal{U}(0.3, 1.0)$; if a worker draws

no nonzero skill, one randomly chosen entry is resampled from $\mathcal{U}(0.3, 1.0)$ to ensure every worker is qualified for at least one skill. Period budgets are set to 0.7–0.9 times a reference full-capacity payroll level. Each worker–period pair receives an independently sampled PWL cost curve from the same generator used in Section I.1, with curves associated with the same worker sharing a common hour range determined by that worker's feasible shift length.

Problem Size. For W workers, P periods, and J breakpoints per cost curve, the model contains

$$WP \text{ hour variables } + WP \text{ cost variables } + WPJ \text{ SOS2 weights } = WP(J+2)$$

variables in total, plus TP skill-coverage constraints, P payroll-budget constraints, and $3WP$ local linear equations from the PWL encoding. In our benchmark suite, the number of local PWL functions (WP) typically ranges from 200 to 2000.

I.3 Block-Angular Resource Allocation with Piecewise-Linear Costs

Problem Formulation. We consider a multi-line production-planning benchmark with block-angular resource structure and separable PWL operating-cost penalties on a subset of products. This problem models a manufacturer operating G product groups (e.g., plants or product lines), in which each group draws on its own dedicated equipment (local resources) but also competes for a shared pool of global resources (e.g., raw-material budget, warehouse capacity, logistics throughput). A subset of products carries a non-convex production-cost curve—capturing setup amortization at low volume, economies of scale at medium volume, and overtime or congestion at high volume—which we model as a univariate PWL penalty.

For each group $g \in \{1, \dots, G\}$, let $\mathcal{J}_g$ denote the set of products in that group, with $|\mathcal{J}_g| = K$ and $N = GK$. For each product $j \in \{1, \dots, N\}$ the decision variable $x_j \geq 0$ is the production level. A subset $\mathcal{P} \subseteq \{1, \dots, N\}$ of size N_{pwl} carries a PWL cost variable z_j, and the objective minimizes the total PWL operating cost,

$$\min_{x,z} \sum_{j \in \mathcal{P}} z_j. \tag{22}$$

The constraint matrix exhibits a block-angular structure: G diagonal local blocks coupled through one shared global block. For each group g, L local capacity constraints couple only products in $\mathcal{J}_g$:

$$\sum_{j \in \mathcal{J}_g} a^{\text{loc}}_{g,r,j}\, x_j \leq c^{\text{loc}}_{g,r}, \qquad r = 1, \dots, L, \tag{23}$$

while M global capacity constraints couple all N products:

$$\sum_{j=1}^{N} a^{\text{glob}}_{m,j}\, x_j \leq c^{\text{glob}}_{m}, \qquad m = 1, \dots, M. \tag{24}$$

All linear coupling constraints are inequalities. For each $j \in \mathcal{P}$, the PWL operating cost is parameterized by J breakpoints $\{(d_{j,k}, \phi_{j,k})\}_{k=1}^{J}$, encoded with SOS2 convex-combination variables $\lambda_{j,k} \in [0, 1]$:

$$\sum_{k=1}^{J} \lambda_{j,k} = 1, \qquad x_j = \sum_{k=1}^{J} d_{j,k}\lambda_{j,k}, \qquad z_j = \sum_{k=1}^{J} \phi_{j,k}\lambda_{j,k}, \tag{25}$$

together with an SOS2 constraint on $\lambda_{j,\cdot}$.

The complete formulation is

$$
\begin{aligned}
\min_{x,z,\lambda} \quad & \sum_{j\in\mathcal{P}} z_j \\
\text{s.t.} \quad & \sum_{j\in\mathcal{J}_g} a^{\text{loc}}_{g,r,j}\, x_j \le c^{\text{loc}}_{g,r}, && g=1,\dots,G,\ r=1,\dots,L,\\
& \sum_{j=1}^{N} a^{\text{glob}}_{m,j}\, x_j \le c^{\text{glob}}_{m}, && m=1,\dots,M,\\
& \sum_{k=1}^{J} \lambda_{j,k} = 1, && j\in\mathcal{P},\\
& x_j = \sum_{k=1}^{J} d_{j,k}\lambda_{j,k}, && j\in\mathcal{P},\\
& z_j = \sum_{k=1}^{J} \phi_{j,k}\lambda_{j,k}, && j\in\mathcal{P},\\
& \text{SOS2}(\lambda_{j,1},\dots,\lambda_{j,J}), && j\in\mathcal{P},\\
& x_j \in [\underline{x}_j, \overline{x}_j], && j=1,\dots,N,
\end{aligned}
$$

where $[\underline{x}_j, \overline{x}_j]$ is a tight box for $j \in \mathcal{P}$ (specified below) and the wider default range $[0, 100]$ for $j \notin \mathcal{P}$.

Data Generation. For each group g, the local block $A^{\text{loc}}_g \in \mathbb{R}^{L\times K}$ is generated with target sparsity $\rho_{\text{loc}} = 0.02$; nonzero positions are drawn uniformly at random and nonzero values are sampled independently from $\mathcal{U}(-2, 2)$. The global block $A^{\text{glob}} \in \mathbb{R}^{M\times N}$ is generated with sparsity $\rho_{\text{glob}} = 0.008$ and the same value distribution; allowing both signs in A^{glob} models global rows that aggregate signed contributions across products (e.g., net demand vs. supply of a shared resource). Capacities are sampled as $c^{\text{loc}}_{g,r}, c^{\text{glob}}_{m} \sim \mathcal{U}(1, 50)$.

The PWL-active subset $\mathcal{P}$ is drawn uniformly at random without replacement from $\{1,\dots,N\}$; breakpoint values $(d_{j,k}, \phi_{j,k})$ are obtained from the same univariate U-shaped curve generator described in Section I.1, applied over the nominal production range $[x_{\min}, x_{\max}] = [0.01, 20]$.

To force a non-trivial branch-and-bound search, every $j \in \mathcal{P}$ is restricted to a tight box of width $0.7(x_{\max} - x_{\min})$ around the midpoint of the PWL range,

$$[\underline{x}_j, \overline{x}_j] = [\bar{m} - 0.35\,\overline{\text{span}},\ \bar{m} + 0.35\,\overline{\text{span}}],$$

with $\bar{m} = (x_{\min} + x_{\max})/2$ and $\overline{\text{span}} = x_{\max} - x_{\min}$; products outside $\mathcal{P}$ take the wider default range $x_j \in [0, 100]$. After the boxes are imposed, the right-hand sides of both blocks are post-processed by adding a uniform slack $s \sim \mathcal{U}(5, 30)$ on top of the worst-case left-hand side $\min_{x\in[\underline{x},\overline{x}]} A_{i,:}\, x$ for each row i, so that the linear feasibility region $\{x : Ax \le c, \underline{x} \le x \le \overline{x}\}$ is non-empty by construction.

Problem Size. For $N = GK$ products, an active subset of size N_{pwl}, and J breakpoints per PWL function, the model has

$$N \text{ production variables } + N_{\text{pwl}} \text{ cost variables } + N_{\text{pwl}}J \text{ SOS2 weights } = N + N_{\text{pwl}}(J+1)$$

variables in total. The number of linear coupling constraints is $GL + M$ (block-diagonal local plus dense global), while the PWL representation contributes $3N_{\text{pwl}}$ local linear equations. In contrast to the portfolio and workforce benchmarks—where every primary variable carries an SOS2 substructure—this benchmark concentrates combinatorial structure on only a fraction of the variable set while keeping a sparse, hierarchical linear coupling.

J Problem Characteristics and Solver Behavior

We analyze how structural differences across the three problem families affect the behavior of our GPU B&B framework.

PWL coverage and sparsity. Resource-allocation instances embed a relatively small fraction of PWL variables in a much larger sparse block-angular LP. Because branching on one PWL variable modifies only a local part of this structure, its improvement to the global lower bound can be small relative to the cost of solving the full node LP. Consequently, more branching may be required before effective pruning occurs. Meanwhile, this sparse structure favors mature simplex-based solvers through basis reuse, sparse factorization, cuts, and other B&B enhancements.

LP size. The node LPs in the resource-allocation family are also substantially larger, reducing the maximum batch size that fits within the same GPU memory and increasing the cost of each batched iteration.

Search-tree structure. These properties tend to produce narrower, deeper search trees because each local PWL branch provides limited global tightening. In contrast, portfolio and workforce expose many independent PWL decisions and tend to generate shallower, wider frontiers that better match GPU batching. Branching quality is consequently more important for resource allocation: an unproductive early decision can send the search deep into a poor direction and require many weak or infeasible node LPs to be solved before pruning. Unlike simplex methods, first-order methods do not return basic vertex solutions and converge slowly at high dual accuracy, weakening lower-bound-based pruning, especially in deeper search trees.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction state that we propose a GPU-batched B&B framework for PWL optimization with SOS2 constraints, in which batched node LP relaxations are solved on the GPU as the main engine. The claimed empirical advantages ($9.25\times$ geometric-mean speedup over cuOpt and high-quality incumbents on every tested instance) match the experimental results in Section 5.2

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: We discuss limitations of $\texttt{B}^3\texttt{-PWL}$ in Appendix B, covering (i) slow dual-bound certification due to the sublinear convergence of first-order LP solvers in high-accuracy regimes, (ii) the absence of warm starting between sibling subproblems, and (iii) the lack of advanced branching rules and cutting planes compared to mature commercial MIP solvers.

   Guidelines:

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

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

   Answer: [N/A]

   Justification: This paper does not include theoretical claims requiring formal proofs.

   Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: Section 5.1 specifies hardware, solver versions (cuOpt v26.02.00, Gurobi as oracle), gap tolerances, time budgets, and baseline configurations. Section 4 describes the algorithm in full, with cuPDLPx details in Appendix E and implementation details (multi-level branching, adaptive batch sizing) in Appendix H. The benchmark generators, parameter grids, and random seeds are documented in Appendix I. Per-instance results are tabulated in Appendix G.

   Guidelines:

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

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

   Answer: [No]

   Justification: The code is not yet publicly available at the time of publication; we plan to release it after cleanup. To support reproduction, Section 5.1 and Appendices H–I describe the implementation, solver configurations, hyperparameters, and benchmark generation procedures.

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Section 5.1 provides hardware settings (Blackwell GPU, CPU thread counts), solver configurations (gap tolerance, PDLP tolerance, LP-bound acceptance threshold, time limit), and cuOpt baseline configuration (version, presolve, cuts, strong branching).

   Guidelines:

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

7. **Experiment statistical significance**

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

Answer: [No]

Justification: The runtime data in our experiment is the mean of three repeated runs on identical hardware. We do not report explicit error bars because the runtime and objective value of both solvers are highly stable across repetitions.

Guidelines:

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Section 5.1 provides hardware settings (Blackwell GPU, CPU thread counts).

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: It does not involve human subjects, personal data, or content generation, and conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.

- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: `B`^{3}`-PWL` accelerates a class of mixed-integer optimization problems used in operations-research applications such as portfolio allocation, workforce scheduling, and resource allocation. Faster and more accessible MIP solving has broad positive societal impact across logistics, energy planning, and scientific computing. The work is foundational and not tied to a specific deployment; we are not aware of direct paths to harmful applications, as our framework solves classical optimization problems whose objectives and constraints are specified by the user, not learned from data.

    Guidelines:

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

11. **Safeguards**

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

    Answer: [N/A]

    Justification: This paper does not release pre-trained models, generative systems, or scraped datasets that pose risk of misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: Justification: We use cuPDLPx [Lu et al., 2025], cuSPARSE [NVIDIA Corporation, 2024], and compare against NVIDIA cuOpt v26.02.00. Gurobi is used as an external reference oracle. All software is used under standard academic/research terms.

    Guidelines:

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

13. **New assets**

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

    Answer: [Yes]

    Justification: We introduce the B^3-PWL solver and a PWL-MIP benchmark suite. Both will be released with documentation and example scripts upon publication.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing or human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

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

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

    Answer: [N/A]

    Justification: This work does not involve human subjects.

    Guidelines:

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

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: LLMs were not used as a component of the core methodology. Any usage was limited to writing assistance and did not affect the algorithmic design, experimental setup, or scientific conclusions.

    Guidelines:

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