Learning to Cut: Reinforcement Learning for Benders Decomposition

Haochen Cai
The Ohio State University
Columbus, OH, USA, 43210
cai.1256@osu.edu

Xian Yu*
The Ohio State University
Columbus, OH, USA, 43210
yu.3610@osu.edu

Abstract

Benders decomposition (BD) is a widely used solution approach for solving two-stage stochastic programs arising in real-world decision-making under uncertainty. However, it often suffers from slow convergence as the master problem grows with an increasing number of cuts. In this paper, we propose Reinforcement Learning for BD (RLBD), a framework that adaptively selects cuts using a neural network-based stochastic policy. The policy is trained using a policy gradient method via the REINFORCE algorithm. We evaluate the proposed approach on a two-stage stochastic electric vehicle charging station location problem and compare it with vanilla BD, Random-K, Top-K, and LearnBD [Jia and Shen, 2021], a supervised learning approach that classifies cuts using a support vector machine. Numerical results demonstrate that RLBD achieves substantial improvements in computational efficiency and exhibits strong generalization to problems with similar structures but varying data inputs and decision variable dimensions. Our code is available at https://github.com/HaochenCai/RLBD.

1 Introduction

Two-stage stochastic programs provide a general framework for modeling decision-making under uncertainty, where decisions are separated into *here-and-now* (first-stage) decisions, made before uncertainty is realized, and *wait-and-see* (second-stage) decisions, which adapt to observed outcomes through recourse actions after the uncertainty unfolds. This modeling paradigm has been widely used in a broad range of applications, including facility location problems [Magnanti and Wong, 1981, Fischetti et al., 2016, 2017], supply chain management [Keyvanshokooh et al., 2016, Basciftci et al., 2023], scheduling and routing problems [Adulyasak et al., 2015, Bodur and Luedtke, 2017, Yu et al., 2021, 2023], energy system operations [Alguacil and Conejo, 2000, Wang et al., 2012, Daneshvar et al., 2020], among others. A key challenge in these problems is that the second-stage decisions depend on random realizations, leading to large-scale optimization models. As the number of scenarios increases, directly solving the resulting extensive-form problem becomes computationally prohibitive.

Benders decomposition (BD) has emerged as a popular method for solving two-stage stochastic programs by decomposing the original problem into a master problem and a set of subproblems [Benders, 1962]. Specifically, at each iteration, the algorithm generates cuts that approximate the recourse function (the second-stage value function) and iteratively tighten the master problem. As a result, the computational performance of BD depends critically on the quality of the cuts that are added during this iterative process. In the multi-cut formulation, the algorithm generates separate cuts for each scenario or subproblem, producing a tighter and more accurate representation of the recourse function; however, adding all of them can significantly increase the size of the master

*Corresponding author: Xian Yu (yu.3610@osu.edu).

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

problem and slow down subsequent solves. On the other hand, in the single-cut formulation, all subproblem (scenario-wise) information is aggregated into one cut per iteration, leading to a smaller master problem but potentially weaker approximations of the recourse function. This introduces a trade-off between strengthening the master problem and maintaining computational efficiency. We refer to Birge and Louveaux [2011] for a detailed introduction of these two formulations.

In this work, we model cut selection in BD as a sequential decision-making problem. At each iteration, the state encodes current convergence statistics, along with features of the master problem, subproblems, and individual cuts, while the action corresponds to selecting a subset of candidate cuts to add to the master problem. Based on this formulation, we develop a reinforcement learning (RL) approach that trains an optimal cut-selection policy using policy gradient algorithms. Our contributions can be summarized as follows:

- **Stochastic cut-selection policy.** Different than the existing literature in BD that selects cuts deterministically, we consider a stochastic policy that selects each cut with a certain probability, based on global algorithmic information and local cut features. This allows the policy to exploit the most informative cuts given the current information while still maintaining the ability to explore alternative cuts that may become important later.
- **Strong generalization to problems with similar structures.** Our numerical results demonstrate that the trained cut-selection policy exhibits strong generalization to problems with similar structures but different data inputs and decision variable dimensions.
- **Substantial improvements over existing benchmarks.** Compared to existing benchmarks such as vanilla BD, Random-K, Top-K, and LearnBD [Jia and Shen, 2021], our RLBD consistently reduces computational time across all test settings. On small- and medium-sized instances, RLBD achieves approximately fivefold speedups over vanilla BD and LearnBD. On larger instances, RLBD reaches the 1% optimality gap tolerance in around 1,000 seconds, whereas vanilla BD and LearnBD fail to reach this tolerance within one hour.

2 Related Work

A variety of classical techniques have been developed to accelerate the convergence of BD. Among them, cut management is a key technique that carefully controls which cuts to add and retain in the master problem. Instead of adding all candidate cuts from solving the subproblems, many approaches add only a selected set of informative cuts at each iteration, such as the most violated cuts or the Pareto-optimal cuts, to improve convergence without excessively enlarging the master problem [Magnanti and Wong, 1981, Fischetti et al., 2010]. Another widely used heuristic is to remove redundant or inactive cuts that no longer contribute to tightening the formulation, thereby controlling the size of the master problem. On the other hand, scenario grouping techniques have been studied in the literature to improve computational efficiency by aggregating multiple cuts from similar scenarios [Adulyasak et al., 2015, Maheo et al., 2019, Ramírez-Pico et al., 2023]. An alternative stream of research focuses on reducing the solution time per iteration by solving the master problem or subproblem inexactly, leading to the so-called inexact BD [Geoffrion and Graves, 1974, Easwaran and Üster, 2009, Poojari and Beasley, 2009].

Recent works have explored using machine learning (ML) and RL to improve the computational efficiency of solving two-stage stochastic programs. Jia and Shen [2021] proposed a learning-enhanced framework (LearnBD) that trains a support vector machine to classify Benders cuts and selectively adds those likely to be valuable to the master problem, thereby reducing the size of the master problem. Choi et al. [2025] replaced the exact subproblem solver with a pretrained transformer model, accelerating the convergence of BD by generating high-quality approximate solutions for the subproblems. Bengio et al. [2020] proposed learning a representative scenario and then solving the resulting deterministic problem to obtain high-quality solutions to two-stage stochastic integer programs. Dumouchelle et al. [2022] proposed Neur2SP, which uses neural networks to approximate the expected value function and embeds the learned models into mixed-integer optimization formulations to obtain first-stage decisions. In the context of multistage stochastic programming, Dai et al. [2022] proposed Neural Stochastic Dual Dynamic Programming, which uses a neural network to learn a mapping from problem instances to piecewise-linear value function approximations. Different than these ML-based prediction models, Li et al. [2025] applied RL to learn policies to adaptively select optimality gap tolerance for the master problem based on the features of

the problem and the solution progress in inexact BD. Mana et al. [2023] replaced the master problem with an RL-based surrogate to generate candidate solutions, without relying on repeatedly solving expensive master problems. For contextual two-stage stochastic integer programs, Nair et al. [2018] used RL to jointly learn policies for constructing initial solutions and improving them through local search.

In a different line of work, researchers have also investigated ML and RL approaches for general-purpose optimization algorithms. For example, ML has been leveraged to solve combinatorial optimization problems [Bengio et al., 2021, Khalil et al., 2017, Li et al., 2018, Khalil et al., 2016]. Recently, Bello et al. [2017] proposed Neural Combinatorial Optimization, a framework that leverages policy gradient methods to train neural networks for generating high-quality solutions to traveling salesman problems. Tang et al. [2020] introduced a deep RL formulation to adaptively select cuts within a branch-and-cut framework for solving integer programs. Chi et al. [2022] proposed a deep RL approach for column generation, where the column selection process is modeled as a sequential decision-making problem and optimized using Q-learning.

3 Preliminaries

In this section, we review the two-stage stochastic programming framework in Section 3.1, and the classical BD in Section 3.2, respectively.

3.1 Two-Stage Stochastic Programming

We consider a class of two-stage stochastic programming problems, where in the first stage, a decision vector $x \in X \subseteq \mathbb{R}^{n_1}$ is chosen before the realization of uncertainty, and in the second stage, a recourse decision $y \in \mathbb{R}^{n_2}$ is made after observing the realization of the uncertain parameter ξ. The two-stage stochastic program can be written as $\min_{x \in X} c^\mathsf{T} x + \mathbb{E}_\xi[Q(x, \xi)]$, where $c \in \mathbb{R}^{n_1}$ is the first-stage cost vector, and $\mathbb{E}_\xi[\cdot]$ denotes the expectation with respect to the distribution of ξ. Here, given the first-stage decision x and uncertainty realization ξ, the second-stage recourse function $Q(x, \xi)$ is defined as $Q(x, \xi) := \min_y \{q^\mathsf{T} y \mid Wy \geq h - Tx\}$, where the uncertain parameter ξ includes the data (W, h, T, q).

We assume that the uncertain parameter ξ can take values from a finite scenario set Ω, where each scenario $\omega \in \Omega$ corresponds to a realization $\xi^\omega = (W^\omega, h^\omega, T^\omega, q^\omega)$ with probability p^ω, satisfying $\sum_{\omega \in \Omega} p^\omega = 1$. Then, a sample average approximation (SAA) to the two-stage stochastic program can be written as

$$\min_{x \in X} \; c^\mathsf{T} x + \sum_{\omega \in \Omega} p^\omega Q^\omega(x), \tag{1}$$

where

$$Q^\omega(x) := \min_y \; (q^\omega)^\mathsf{T} y \quad \text{s.t.} \quad W^\omega y \geq h^\omega - T^\omega x. \tag{2}$$

To ensure a well-defined problem, we further assume that the second-stage problem (2) is always feasible (i.e., satisfies relatively complete recourse) and that the recourse function is finite, i.e., $|Q^\omega(x)| < +\infty$ for all $x \in X$ and $\omega \in \Omega$. Our goal is to develop an efficient solution approach for the SAA problem (1).

3.2 Benders Decomposition

The extensive-form SAA (1) becomes computationally challenging when the number of scenarios increases. BD exploits its separable structure by decomposing it into a master problem and a set of scenario-wise subproblems. At each iteration t, we first solve the following master problem (MP_t):

$$\begin{aligned} (\text{MP}_t) \quad \min_{x \in X, \theta} \quad & c^\mathsf{T} x + \sum_{\omega \in \Omega} p^\omega \theta^\omega \\ \text{s.t.} \quad & \theta^\omega \geq (\pi_\tau^\omega)^\mathsf{T} (h^\omega - T^\omega x), \quad \forall \tau \in [t-1],\ \omega \in \Omega, \end{aligned} \tag{3}$$

where $\theta^\omega \geq (\pi_\tau^\omega)^\mathsf{T}(h^\omega - T^\omega x),\ \forall \tau \in [t-1],\ \omega \in \Omega$ denote the current optimality cuts to approximate the recourse function $Q^\omega(x)$ from below, and no cuts are present when $t = 1$. The problem (MP_t) is solved to obtain an optimal first-stage solution $(\hat{x}_t, \hat{\theta}_t)$ together with a lower bound

to the original problem (1). Given $\hat{x}_t$, we construct the following dual of the subproblem for each scenario $\omega \in \Omega$:

$$(\text{SP}_t^\omega) \quad \max_{\pi^\omega \geq 0} (\pi^\omega)^\mathsf{T}(h^\omega - T^\omega \hat{x}_t) \quad \text{s.t.} \quad (W^\omega)^\mathsf{T}\pi^\omega = q^\omega. \tag{4}$$

We denote the optimal solution of (4) as π_t^ω. By strong duality, $Q^\omega(\hat{x}_t) = (\pi_t^\omega)^\mathsf{T}(h^\omega - T^\omega \hat{x}_t)$, and since $\hat{x}_t$ is a feasible first-stage solution, $c^\mathsf{T}\hat{x}_t + \sum_{\omega\in\Omega} p^\omega Q^\omega(\hat{x}_t)$ provides an upper bound for (1). The optimal dual solutions $\{\pi_t^\omega\}_{\omega\in\Omega}$ are then used to generate Benders optimality cuts of the following form

$$\theta^\omega \geq (\pi_t^\omega)^\mathsf{T}(h^\omega - T^\omega x), \quad \forall \omega \in \Omega, \tag{5}$$

which provide valid lower approximations of the recourse function. If the current (MP_t)'s optimal solution $(\hat{x}_t, \hat{\theta}_t)$ satisfies the newly generated cuts (5), then we terminate the algorithm and return $(\hat{x}_t, \hat{\theta}_t)$ as the optimal solution. Otherwise, we add the optimality cuts (5) to the master problem and proceed to the next iteration. This iterative process converges to a globally optimal solution in a finite number of iterations under mild conditions, as shown in Van Slyke and Wets [1969].

Despite its effectiveness, classical BD often suffers from slow convergence. In particular, the number of generated cuts can grow rapidly, leading to computationally expensive master problems. Motivated by this computational challenge, we develop an RL framework to speed up the BD algorithm next.

4 Reinforcement Learning for Benders Decomposition

In this section, we propose Reinforcement Learning for Benders Decomposition (RLBD), a framework that adaptively selects Benders cuts to improve computational efficiency. Unlike classical BD, which either adds all scenario-wise cuts or aggregates them into a single cut at each iteration, RLBD formulates cut selection as a sequential decision-making problem and learns a policy to select the most informative cuts at each iteration (see Figure 1 for an illustration). We first introduce the MDP formulation in Section 4.1 and the neural network policy parameterization in Section 4.2, respectively. The RLBD framework consists of two phases: (i) **Training**, where the stochastic cut-selection policy is learned using a policy gradient method via REINFORCE (Section 4.3), and (ii) **Testing**, where the trained policy is applied in a greedy manner to new problem instances via policy rollout (Section 4.4).

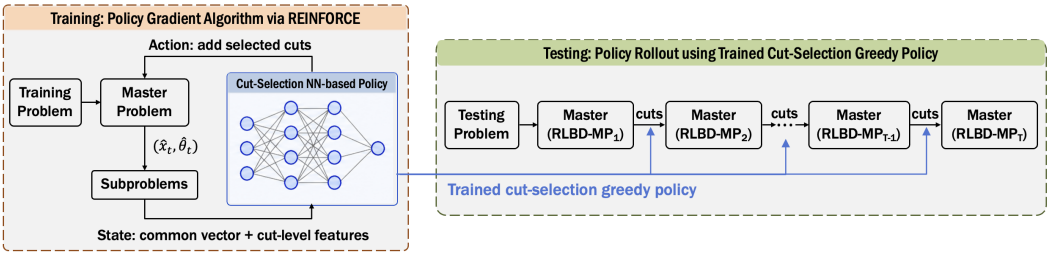


Figure 1: The RLBD Framework.

4.1 Markov Decision Process Formulation

We model the cut-selection problem in BD as a finite-horizon discounted MDP. In this formulation, each Benders iteration corresponds to one decision step. At iteration t, the agent observes the current master problem (RLBD-MP_t):

$$\begin{aligned}(\text{RLBD-MP}_t) \quad \min_{x\in X,\theta} \quad & c^\mathsf{T}x + \sum_{\omega\in\Omega} p^\omega \theta^\omega \\ \text{s.t.} \quad & \theta^\omega \geq (\pi_\tau^\omega)^\mathsf{T}(h^\omega - T^\omega x), \quad \forall \tau \in [t-1],\ \omega \in A_\tau. \end{aligned} \tag{6}$$

Unlike (MP_t) in classical BD (3), where all scenario-wise cuts $\omega \in \Omega$ are included, we instead maintain only a subset $A_\tau \subset \Omega$ for all $\tau \in [t-1]$. Moreover, the selected subset A_τ may vary across iterations. The agent solves (RLBD-MP_t) to obtain an optimal solution $(\hat{x}_t, \hat{\theta}_t)$ and an optimal objective value, which serves as a lower bound for problem (1), denoted by LB_t. Based on $(\hat{x}_t, \hat{\theta}_t)$, the agent then solves the same subproblem (SP_t^ω) in (4) and generates scenario-wise cuts of the form (5)

for all $\omega \in \Omega$. Since the first-stage solution $\hat{x}_t$ is feasible for (1) and $Q^\omega(\hat{x}_t) = (\pi_t^\omega)^\mathsf{T}(h^\omega - T^\omega \hat{x}_t)$, the agent constructs an upper bound for problem (1), defined by $\text{UB}_t := c^\mathsf{T}\hat{x}_t + \sum_{\omega\in\Omega} p^\omega (\pi_t^\omega)^\mathsf{T}(h^\omega - T^\omega \hat{x}_t)$. Then, the agent selects a subset of candidate cuts $A_t \subset \Omega$, and the environment advances to the next iteration $t+1$ by adding the selected cuts A_t to (6).

Formally, the finite-horizon discounted MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, f, r, \gamma, T)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ represents the transition dynamics induced by the Benders solver, $r(s, a)$ is the step-wise reward function, $\gamma \in (0, 1]$ is the discount factor, and T is the number of decision stages. Next, we introduce each component in detail.

4.1.1 State Space

At each iteration t, the state representation contains both global information about the current convergence statistics, master problem and subproblem features, shared by all candidate cuts, and local information about each candidate cut.

Convergence statistics. The state includes the current iteration index t, lower bound LB_t, upper bound UB_t, and the current optimality gap, defined by $\text{Gap}_t := (\text{UB}_t - \text{LB}_t)/(|\text{UB}_t| + \epsilon)$. It also includes changes in the lower/upper bounds and the optimality gap, defined by $\Delta\text{LB}_t := \text{LB}_t - \text{LB}_{t-1}$, $\Delta\text{UB}_t := \text{UB}_{t-1} - \text{UB}_t$, and $\Delta\text{Gap}_t := \text{Gap}_{t-1} - \text{Gap}_t$, respectively. Based on them, the state further includes the gap reduction rate $\rho_t^{\text{Gap}} := \Delta\text{Gap}_t/(\text{Gap}_{t-1} + \epsilon)$, lower-bound improvement rate $\rho_t^{\text{LB}} := \Delta\text{LB}_t/(|\text{LB}_{t-1}| + \epsilon)$, and upper-bound improvement rate $\rho_t^{\text{UB}} := \Delta\text{UB}_t/(|\text{UB}_{t-1}| + \epsilon)$, which characterize the convergence statistics of the Benders procedure across iterations. Here, $\epsilon > 0$ is a small positive constant, which is added in the denominator to avoid numerical instability. For each candidate cut of the form (5) generated at iteration t, we denote the cut-level violation at the current master solution $(\hat{x}_t, \hat{\theta}_t)$ as $v_t^\omega := (\pi_t^\omega)^\mathsf{T}(h^\omega - T^\omega \hat{x}_t) - \hat{\theta}_t^\omega$ for all $\omega \in \Omega$. We then record the mean violation $\bar{v}_t := \sum_{\omega\in\Omega} p^\omega v_t^\omega$ and the maximum violation $v_t^{\max} := \max_{\omega\in\Omega} v_t^\omega$, which reflect the approximation quality of the cuts. Note that $\bar{v}_t$ and $v_t^{\max}$ represent the global information, and we will include the cut-level violation v_t^ω in the local state component, introduced later.

Master problem features. The state includes the number of cuts added in the previous iteration K, the cumulative number of cuts added C_t^{cum}, and the solve time for (RLBD-MP_t), denoted by T_t^{MP}. These features reflect the size and computational burden of the master problem.

Subproblem features. After solving the scenario-wise subproblems (4), the state records summary statistics of the scenario-wise recourse costs. Let Q_t^ω denote the optimal objective value for (SP_t^ω) in (4) under the current first-stage solution $(\hat{x}_t, \hat{\theta}_t)$. We include the average recourse cost $\bar{Q}_t = \sum_{\omega\in\Omega} p^\omega Q_t^\omega$, the maximum recourse cost $Q_t^{\max} = \max_{\omega\in\Omega} Q_t^\omega$, the minimum recourse cost $Q_t^{\min} = \min_{\omega\in\Omega} Q_t^\omega$, and the standard deviation of the scenario-wise recourse costs $\{Q_t^\omega\}_{\omega\in\Omega}$.

Cut-level features. The cut-level state is given by $(v_t^\omega, \|\pi_t^\omega\|_2, |(\pi_t^\omega)^\mathsf{T} h^\omega|, \|(\pi_t^\omega)^\mathsf{T} T^\omega\|_2, \text{NC}_t^\omega)$ for each $\omega \in \Omega$, where v_t^ω is the cut-level violation, π_t^ω is the optimal dual variable used to generate the cut, $(\pi_t^\omega)^\mathsf{T} h^\omega$ and $(\pi_t^\omega)^\mathsf{T} T^\omega$ represent the intercept and coefficient of the cut, and $\text{NC}_t^\omega := \sum_{\tau=1}^{t-1} \mathbf{1}\{\omega \in A_\tau\}$ records the number of cuts generated by the same scenario ω in previous iterations, respectively. These local features capture the quality of each candidate cut, while the global features track the current convergence status.

The initial state s_1 is obtained from solving the master problem (RLBD-MP_1) without any cuts. The initial LB_1, UB_1, Gap_1, cut violation statistics, and recourse-related statistics are computed from solving the initial master problem and subproblems. The history-dependent features, such as gap reduction and bound improvement rates, are initialized to zero because no previous iteration exists.

4.1.2 Action Space, Transition Dynamics, and Reward Function

Action Space. At each iteration t, we use $\{1, 2, \ldots, |\Omega|\}$ to encode all candidate cuts, each corresponding to one optimality cut of the form (5) with coefficients obtained from solving the subproblem (4) for scenario $\omega \in \Omega$. The action corresponds to selecting a subset of K cuts from $\{1, 2, \ldots, |\Omega|\}$ to be added to the master problem. To this end, we model the action space as $\mathcal{A} = \{A \subset \{1, 2, \ldots, |\Omega|\} : |A| = K\}$. Note that the same cut index $a_t \in \Omega$ may correspond to different cuts in different iterations t, because even for the same scenario, the cut coefficients obtained from solving the subproblem (4) can vary depending on the first-stage solution.

We use $\pi_\theta(a_t|s_t)$ to denote the probability of selecting cut $a_t \in \{1, 2, \ldots, |\Omega|\}$ given state s_t. Note that since the iteration index t is included in state s_t, this policy $\pi_\theta(a_t|s_t)$ is naturally time-dependent. Given state s_t, we sample an ordered sequence $A_t = (a_t^{(1)}, a_t^{(2)}, \ldots, a_t^{(K)})$ following $\pi_\theta(\cdot|s_t)$ without replacement. The sampling is performed sequentially: after one cut is selected, it is masked out, and the remaining probabilities are re-normalized before the next cut is sampled. The joint probability of action $A_t \in \mathcal{A}$ is thus given by $P_\theta(A_t|s_t) = \prod_{i=1}^{K} \frac{\pi_\theta(a_t^{(i)}|s_t)}{1-\sum_{j<i}\pi_\theta(a_t^{(j)}|s_t)}$.

This stochastic cut-selection policy naturally introduces exploration through the policy distribution. Unlike deterministic selection rules, the use of a stochastic policy allows the agent to sample actions according to their relative importance, assigning higher probability to more promising cuts while retaining the ability to explore alternative candidates. We will introduce a neural network architecture to parameterize $\pi_\theta(a_t|s_t)$ based on both global information and cut-level features in Section 4.2.

Transition Dynamics. Given the current state s_t and action A_t, the environment advances to the next state $s_{t+1} = f(s_t, A_t)$ by adding the selected cuts A_t to the master problem (6). The updated master problem (RLBD-MP$_{t+1}$) is re-optimized to obtain a new first-stage solution. Based on this solution, we solve the subproblems (SP$^\omega_{t+1}$) for all scenarios $\omega \in \Omega$ and generate a new set of candidate cuts. The resulting information from the master problem and subproblems, including updated bounds, optimality gap, violation, and other summary statistics, is used to construct the next state s_{t+1}.

Reward Function. The step-wise reward is defined as $r(s_t, A_t) := \alpha \cdot F_t + \beta \cdot L_t - \lambda$, where F_t denotes the gap-based reward term, L_t denotes the reward associated with the master problem solve time, and $-\lambda$ is a per-step penalty. First of all, the gap-based reward is defined by $F_t := \log(\text{Gap}_{t-1}) - \log(\text{Gap}_t)$, which is scale-invariant and designed to reflect the convergence progress of BD. Second, $L_t := -\frac{T_t^{\text{MP}}}{T_{\text{ref}}}$, where T_t^{MP} is the master problem solve time at iteration t, and T_{ref} is a normalization constant chosen based on the problem scale. Finally, the per-step penalty $-\lambda$ is introduced to encourage convergence within fewer iterations, which accumulates over time and imposes an increasing penalty on prolonged solution trajectories. Note that the parameters α, β and λ are user-defined weights that balance these components and will be tuned in Section 5.2.

Combining all components above, the value function of this MDP takes the following form:

$$\max_\theta V_\theta(s_1) := \mathbb{E}\left[\sum_{t=1}^{T} \gamma^t r(s_t, A_t)\right], \text{ where } A_t \sim P_\theta(\cdot|s_t),\ s_{t+1} = f(s_t, A_t),\ \forall t \in [T].$$

4.2 Neural Network Policy Parameterization

We parameterize the cut-selection policy $\pi_\theta(a_t|s_t)$ using a shared feedforward neural network (see Figure 2). The purpose of the network is to assign a scalar score to each candidate cut, which is later converted into a probability distribution for stochastic cut selection. Specifically, at iteration t, for each candidate cut $\omega \in \Omega$, we construct a cut-specific state representation $s_t^\omega \in \mathbb{R}^{28}$. The first 23 entries of s_t^ω include the global state information shared by all candidate cuts, and the last 5 entries describe the local properties of candidate cut ω, given by $(v_t^\omega, \|\pi_t^\omega\|_2, |(\pi_t^\omega)^\mathsf{T} h^\omega|, \|(\pi_t^\omega)^\mathsf{T} T^\omega\|_2, \text{NC}_t^\omega)$.

Each input vector s_t^ω is passed through a multilayer perceptron with two hidden layers and ReLU activations: $z_\omega = \text{NN}_\theta(s_t^\omega) = W_3 \cdot \text{ReLU}(W_2 \cdot \text{ReLU}(W_1 s_t^\omega + b_1) + b_2) + b_3$, for all $\omega \in \Omega$, where $\theta = (W_l, b_l)_{l=1}^3$ denotes the network parameters, and $z_\omega \in \mathbb{R}$ is the output logit assigned to candidate cut ω. Given the logits produced by the neural network, we apply a softmax function to transform them into a probability distribution: $\pi_\theta(\cdot|s_t) = \text{Softmax}(z_1, z_2, \ldots, z_{|\Omega|})$. Here, $\pi_\theta(\omega|s_t)$ denotes the probability of selecting cut ω, which is used to construct the joint action probability $P_\theta(A_t|s_t)$.

4.3 Training: Policy Gradient Algorithm via REINFORCE

In this section, we leverage policy gradient methods to train the stochastic cut-selection policy π_θ via the REINFORCE algorithm [Williams, 1992] (see Algorithm 1). One training episode corresponds to one complete BD trajectory. During each episode, at each iteration t, the algorithm records the current state s_t, the selected cuts A_t, their log-probabilities $\log P_\theta(A_t|s_t)$, and the stepwise reward r_t. After the episode terminates (either BD converges or hits the maximum iterations $T_{\max}$), the return-to-go is computed for each step: $G_t = \sum_{\ell=t}^{T} \gamma^{\ell-t} r_\ell$. According to Williams [1992], the gradient

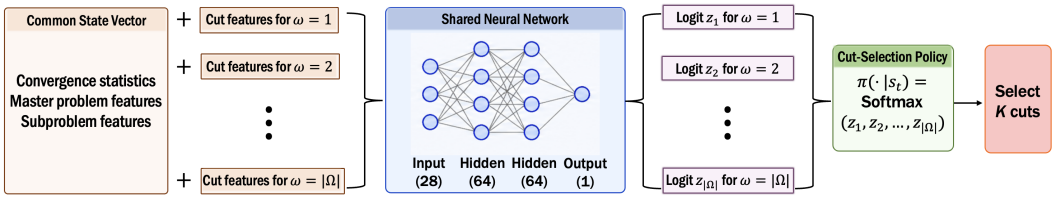


Figure 2: The shared neural network is applied to each candidate cut using the concatenation of common state vector and local cut features, producing one logit per candidate cut.

Algorithm 1: Training Cut-Selection Policy via REINFORCE

Input : Training problem, initial policy network parameter θ, discount factor γ, number of cuts added per iteration K, maximum iterations $T_{\max}$, number of episodes E, learning rate η, optimality gap tolerance ϵ_{tol}, reward weights $\alpha,\ \beta,\ \lambda$, reference master problem solve time T_{ref}

Output : Trained cut-selection policy π_θ

for $e = 1, \ldots, E$ **do**
  Initialize the master problem (RLBD-MP$_1$) based on the training problem without any cuts;
  for $t = 1, \ldots, T_{\max}$ **do**
    Solve (RLBD-MP$_t$) to obtain an optimal solution $(\hat{x}_t, \hat{\theta}_t)$ and a lower bound LB$_t$;
    Based on $(\hat{x}_t, \hat{\theta}_t)$, solve (SP$_t^\omega$) to construct candidate cuts (5) and obtain an upper bound UB$_t$;
    Construct state s_t according to the global information and cut-level features;
    Compute policy distribution $\pi_\theta(\cdot|s_t) = \text{Softmax}(\text{NN}_\theta(s_t^1), \ldots, \text{NN}_\theta(s_t^{|\Omega|}))$;
    Sample K cuts following $\pi_\theta(\cdot|s_t)$ without replacement and set the selected subset as action A_t;
    Compute the stepwise reward $r_t = r(s_t, A_t) = \alpha \cdot F_t + \beta \cdot L_t - \lambda$;
    Add the selected cuts A_t to the master problem and advance to the next iteration;
    if $(UB_t - LB_t)/|UB_t| < \epsilon_{tol}$ **then**
      break;
  Compute the return-to-go backward: $G_T = 0$ and $G_t = r_t + \gamma G_{t+1}$ for $t = T-1, \ldots, 0$;
  Update the policy parameter: $\theta \leftarrow \theta + \eta \sum_{t=1}^{T} \nabla_\theta \log P_\theta(A_t \mid s_t) \cdot G_t$;
return π_θ.

of the value function $V_\theta(s_1)$ can be estimated as $\nabla_\theta V_\theta(s_1) \approx \mathbb{E}[\sum_{t=1}^{T} \nabla_\theta \log P_\theta(A_t|s_t) \cdot G_t]$, where $\nabla_\theta \log P_\theta(A_t|s_t) = \sum_{i=1}^{K} \nabla_\theta \log(\frac{\pi_\theta(a_t^{(i)}|s_t)}{1-\sum_{j<i}\pi_\theta(a_t^{(j)}|s_t)})$. At the end of each episode, we apply backpropagation to update the policy network parameter θ in the gradient ascent direction.

4.4 Testing: Policy Rollout using the Trained Cut-Selection Greedy Policy

Once the cut-selection policy π_θ is trained, it can be applied to solve problems with similar structures but different inputs. We consider a setting in which a decision maker needs to repeatedly solve a collection of two-stage stochastic programs that share the same problem structure but differ in their realizations of uncertain parameters and/or dimensions of the decision variables. This setting naturally arises in applications where similar decision problems must be solved under different scenarios or operating conditions. For example, in supply chain management, one may solve a sequence of stochastic facility location problems that share the same objective function and constraints but differ in demand realizations or decision dimensions. Since the policy network π_θ is designed to be independent of the problem size, it can naturally generalize to such instances with different demand realizations and varying numbers of candidate facilities and customer sites. For a given test problem, we then implement BD with the trained cut-selection policy in a greedy manner: at each iteration, we select the top-K cuts with the largest logits from the policy network to add to the master problem.

5 Numerical Results

We test our RLBD on a two-stage stochastic electric vehicle (EV) charging station location problem and compare it with existing benchmarks. We first introduce our experimental setup in Section 5.1, report the hyperparameter tuning results in Section 5.2, and compare our RLBD with other

benchmarks in Section 5.3, respectively. Finally, we provide some managerial insights from the trained cut-selection policy in Section 5.4.

5.1 Experimental Setup

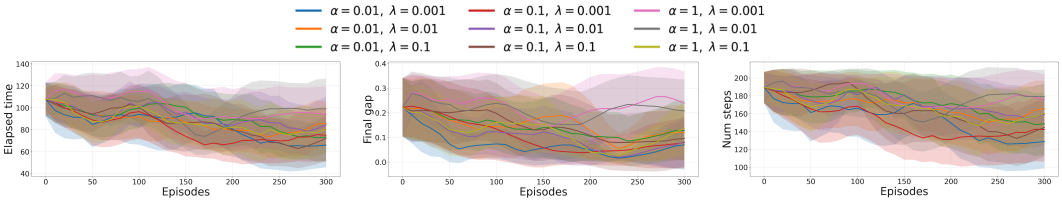
We consider a two-stage stochastic EV charging station location problem [Sun et al., 2026], where we have a set $\mathcal{I}$ of candidate charging stations and a set $\mathcal{J}$ of customer sites with uncertain demand $d_j,\ j \in \mathcal{J}$. In the first stage, a subset of charging stations is selected to open, and their capacities are determined before demand realization. In the second stage, demand is realized and fulfilled by the open stations. The objective is to minimize first-stage investment costs and second-stage transportation and unmet-demand penalties, while maximizing total revenue. In the training phase, we generate $|\mathcal{I}| = 8$ candidate charging stations and $|\mathcal{J}| = 12$ customer sites. For each customer site $j \in \mathcal{J}$, the demand d_j follows a Normal distribution. We first sample μ_j uniformly from $[20, 100]$ and set $\sigma_j = 0.1\mu_j$ and then generate $|\Omega| = 100$ demand scenarios $\{d_j^\omega\}_{\omega\in\Omega}$ following $\mathcal{N}(\mu_j, \sigma_j^2)$. This forms our training problem. Please refer to (7)–(13) in Appendix A for the full model formulation and parameter setup.

We train the policy network using the `Adam` optimizer, with learning rate 10^{-3}, and discount factor $\gamma = 0.99$. All experiments are conducted on the Pitzer cluster at the Ohio Supercomputer Center with 16 CPU cores. We use Gurobi 12.0.0 coded in Python 3.10 for solving all optimization problems.

5.2 Training: Hyperparameter Tuning

We first tune the hyperparameters in defining the reward function. We fix $\beta = 0.001$ and $T_{\text{ref}} = 0.1$ and perform a grid search over the other weight parameters $\alpha \in \{0.01, 0.1, 1.0\}$ and $\lambda \in \{0.001, 0.01, 0.1\}$, resulting in nine reward configurations. For each configuration, we train the policy network in five independent runs and evaluate the learned cut-selection policies in a greedy manner on the same training problem. The evaluation results are summarized in Figure 3, where we plot the episodic elapsed time, final optimality gap, and number of Benders iterations across different reward configurations. The shaded area represents the 90% confidence interval among the five independent runs. Based on these curves, $\alpha = 0.01$ and $\lambda = 0.001$ perform the best in all three metrics, and we fix $\alpha = 0.01$ and $\lambda = 0.001$ in the following experiments.

After fixing the reward setting, we now tune the number of cuts added per iteration K from $\{10, 20, 30\}$. The average evaluation results over five independent runs are shown in Figure 4, where we report the overall trends of elapsed time, final optimality gap, and the number of Benders iterations under different values of K. Note that in these experiments, even under the same initialized policy, different values of K lead to different BD performance, and therefore, the three curves do not start from the same initial point. We also present the total rewards for $K \in \{10, 20, 30\}$ and close-up comparisons of total reward and elapsed time in Figure 6 in Appendix B. In later training episodes, $K = 10$ achieves higher total rewards and lower elapsed times. Overall, $K = 10$ achieves the most stable convergence behavior and computational efficiency, and we adopt $K = 10$ in the next benchmark experiments.

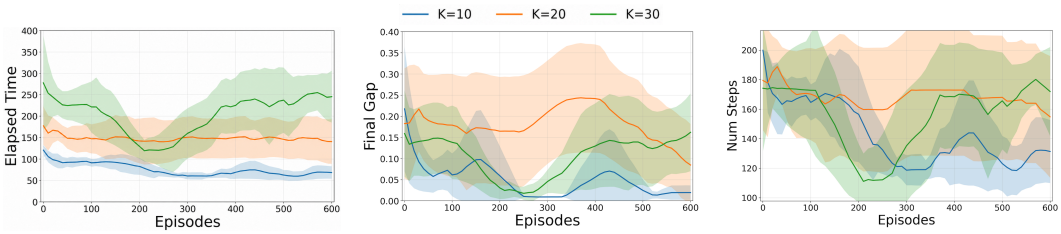


(a) Elapsed Time (b) Final Optimality Gap (c) Number of Steps

Figure 3: Hyperparameter tuning results under different combinations of α and λ.

5.3 Testing: Comparison with Existing Benchmarks

Once the cut-selection policy π_θ is trained in our RLBD framework, we apply it to solve testing instances with various demand distributions and problem sizes and compare it with the following



(a) Elapsed Time (b) Final Optimality Gap (c) Number of Steps

Figure 4: Evaluation results under different numbers of cuts added per iteration K.

benchmarks: (i) vanilla BD presented in Section 3.2; (ii) LearnBD proposed in Jia and Shen [2021], which trains a support vector machine classifier on the same training problem; (iii) Random-K, which uniformly samples K candidate cuts without replacement at each iteration; and (iv) Top-K, which selects the top K candidate cuts that maximize the cut-level violation v_t^ω. Note that Random-K, Top-K and our RLBD share the same cut budget per iteration. We also consider several demand distribution settings in which we sample $|\Omega| = 100$ testing demand scenarios from the same Normal distribution as the training problem, as well as from left-skewed and right-skewed Normal distributions by considering potential distribution shifts.

Table 1 summarizes the comparison between BD, LearnBD, Random-K, Top-K, and RLBD on different testing instances, where we report the average solution time (Time), master problem solve time (Master), number of iterations (Iter.), and the final optimality gap (Gap) over five independent testing datasets. The time limit is set to one hour and the optimality gap tolerance is set to $\epsilon_{\text{tol}} = 1\%$ for all instances. From Table 1, when $|\mathcal{I}| \times |\mathcal{J}| = 8 \times 12$ or 10×15, all methods solve the problem to the prescribed optimality gap tolerance, while RLBD achieves approximately a fivefold reduction in computational time over BD and LearnBD and also requires less computational time than Random-K and Top-K. This improvement is primarily due to more efficient master problems with adaptive cut selection (see the "Master" column). When the problem size increases to $|\mathcal{I}| \times |\mathcal{J}| = 15 \times 20$, both BD and LearnBD cannot solve the problem within one hour and produce large optimality gaps; however, RLBD reduces the final optimality gap to within 1% in around 1,000 seconds.

Table 1: Performance comparison between BD, LearnBD, and RLBD on testing instances with various demand distributions and problem sizes

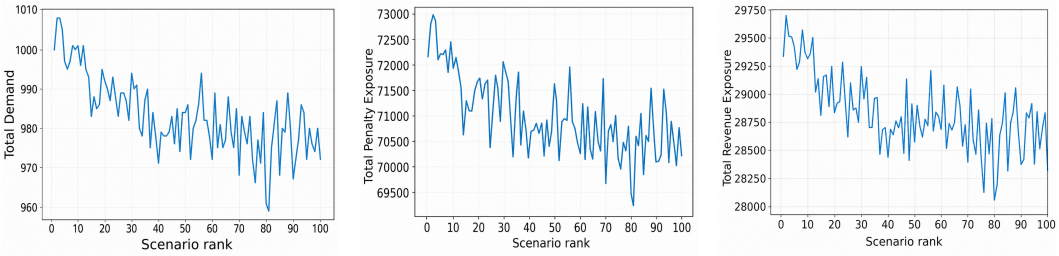
$\lvert\mathcal{I}\rvert \times \lvert\mathcal{J}\rvert$	Method	Normal				Left-skewed Normal				Right-skewed Normal			
		Time (s)	Master (s)	Iter.	Gap (%)	Time (s)	Master (s)	Iter.	Gap (%)	Time (s)	Master (s)	Iter.	Gap (%)
8×12	BD	237.63	222.03	90.2	0.55	244.52	227.63	90.6	0.57	252.60	234.63	90.4	0.56
	LearnBD	234.77	212.56	124.2	0.55	239.75	215.14	125.0	0.57	246.73	221.81	124.6	0.56
	Random-K	63.26	35.75	122.2	0.87	67.39	38.90	126.2	0.95	64.17	36.58	124.2	0.86
	Top-K	55.02	30.42	107.6	0.91	54.14	29.69	108.6	0.85	54.57	30.19	108.0	0.86
	RLBD	**44.43**	**25.62**	106.6	0.95	**46.00**	**26.39**	108.0	0.90	**48.32**	**27.16**	106.6	0.87
10×15	BD	446.70	418.17	121.4	0.69	436.67	404.55	118.4	0.87	454.84	423.42	122.2	0.82
	LearnBD	448.74	408.87	163.8	0.71	440.99	396.62	161.2	0.87	453.92	410.03	164.2	0.82
	Random-K	129.55	78.21	162.2	0.89	133.92	82.02	167.2	0.85	131.56	79.42	164.2	0.79
	Top-K	121.61	73.13	157.8	0.94	130.61	77.61	160.6	0.71	119.29	70.59	157.4	0.69
	RLBD	**97.37**	**60.25**	154.4	0.86	**102.26**	**62.00**	151.4	0.97	**102.77**	**62.66**	150.8	0.79
15×20	BD†	3600.00	3506.94	254.8	23.49	3600.00	3501.10	257.8	22.99	3600.00	3501.31	251.8	26.58
	LearnBD†	3600.00	3489.76	297.6	24.02	3600.00	3496.69	298.6	16.86	3600.00	3486.71	293.4	27.56
	Random-K	1231.44	1052.13	334.6	0.91	1409.39	1221.68	346.8	0.95	1432.20	1245.39	342.4	0.94
	Top-K	1234.08	1087.46	347.0	0.93	1195.01	1049.06	347.8	0.84	1207.88	1057.62	352.0	0.87
	RLBD	**1067.33**	**926.72**	331.6	0.86	**1068.45**	**929.41**	334.4	0.99	**1090.37**	**951.19**	336.6	0.90

† indicates that the one-hour time limit was reached before the prescribed 1% optimality gap tolerance was met.

5.4 Managerial Insights from the Learned Cut-Selection Policy

To understand the relationship between the behavior of the trained cut-selection policy π_θ in RLBD and the characteristics of the input dataset, we rank the scenarios based on the number of cuts generated by each scenario across all iterations (NC_T^ω) and plot their corresponding total demand $\sum_{j\in\mathcal{J}} d_j^\omega$, total penalty exposure $\sum_{j\in\mathcal{J}} p_j d_j^\omega$, and total revenue exposure $\sum_{j\in\mathcal{J}} r_j d_j^\omega$ in Figure 5. On the x-axis, scenarios are sorted by the selected count NC_T^ω in descending order, where Scenario #1 corresponds to the most frequently selected scenario. We focus on the testing instance with right-skewed Normal distribution and $|\mathcal{I}| \times |\mathcal{J}| = 10 \times 15$. From Figure 5, the trained policy tends

to select cuts from scenarios with higher total demand, penalty exposure, and revenue exposure, suggesting that high-load demand realizations correspond to more critical scenarios and are more likely to generate informative cuts.



(a) Total Demand (b) Total Penalty Exposure (c) Total Revenue Exposure

Figure 5: Scenario-level exposure metrics sorted by scenario rank.

6 Conclusions

In this work, we proposed an RLBD framework for adaptively selecting Benders cuts to improve the efficiency of solving two-stage stochastic programs. We modeled the cut-selection process as an MDP, where each Benders iteration corresponds to one decision step, and the state includes current convergence statistics, master problem and subproblem features, and cut-level features. The reward accounts for optimality-gap reduction, master problem solution time, and the number of iterations. We parameterized the stochastic cut-selection policy using a neural network and trained it with the REINFORCE algorithm. The learned policy can be deployed on testing instances with varying uncertainty distributions and problem sizes. Numerical experiments on a two-stage stochastic EV charging station location problem demonstrated that RLBD generalizes effectively to problems with similar structures but different input data (including distribution shifts) and decision variable dimensions. Compared with existing benchmark methods, including vanilla BD, Random-K, Top-K, and LearnBD, RLBD consistently reduces computational time across all test settings and achieves approximately fivefold speedups over vanilla BD and LearnBD on small and medium-sized testing instances.

The proposed RLBD framework is currently limited to two-stage stochastic mixed-integer linear programs with continuous second-stage recourse variables. In addition, the cut-selection policy is restricted to a neural network representation. There are several future directions worth investigating. First, the framework could be extended to two-stage stochastic programs with integer recourse by learning a policy that selects among alternative classes of cuts, such as integer optimality cuts, Lagrangian cuts, or strengthened Benders cuts. Second, it would be worthwhile to investigate more expressive policy representations, such as graph neural networks, which can naturally capture the structural relationships among variables, constraints, and cuts and may further improve the policy's generalization ability.

7 Acknowledgments

This work is supported in part by NSF under grant 2331782.

References

Yossiri Adulyasak, Jean-François Cordeau, and Raf Jans. Benders decomposition for production routing under demand uncertainty. *Operations Research*, 63(4):851–867, 2015.

N. Alguacil and A. J. Conejo. Multiperiod optimal power flow using benders decomposition. *IEEE Transactions on Power Systems*, 15(1):196–201, 2000. doi: 10.1109/59.852121.

Beste Basciftci, Xian Yu, and Siqian Shen. Resource distribution under spatiotemporal uncertainty of disease spread: Stochastic versus robust approaches. *Computers & Operations Research*, 149: 106028, 2023.

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *International Conference on Learning Representations (ICLR),Workshop Track*, 2017.

J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962. doi: 10.1007/BF01386316. URL `https://doi.org/10.1007/BF01386316`.

Yoshua Bengio, Emma Frejinger, Andrea Lodi, Rahul Patel, and Sriram Sankaranarayanan. A learning-based algorithm to quickly compute good primal solutions for stochastic integer programs. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, volume 12296 of *Lecture Notes in Computer Science*, pages 99–111. Springer, 2020. doi: 10.1007/978-3-030-58942-4_7.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

John R. Birge and François Louveaux. *Introduction to Stochastic Programming*. Springer, 2011.

Merve Bodur and James R Luedtke. Mixed-integer rounding enhanced benders decomposition for multiclass service-system staffing and scheduling with arrival rate uncertainty. *Management Science*, 63(7):2073–2091, 2017.

Cheng Chi, Amine Mohamed Aboussalah, Elias B. Khalil, Juyoung Wang, and Zoha Sherkat-Masoumi. A deep reinforcement learning framework for column generation. In *Advances in Neural Information Processing Systems*, 2022.

Seung Jin Choi, Kimiya Jozani, Joshua F Cooper, and I Esra Buyuktahtakin. Learning to optimize at scale: A Benders decomposition-transformers framework for stochastic combinatorial optimization. In *NeurIPS 2025 Workshop MLxOR: Mathematical Foundations and Operational Integration of Machine Learning for Uncertainty-Aware Decision-Making*, 2025.

Hanjun Dai, Yuan Xue, Zia Syed, Dale Schuurmans, and Bo Dai. Neural stochastic dual dynamic programming. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=aisKPsMM3fg`.

Mohammadreza Daneshvar, Behnam Mohammadi-Ivatloo, Kazem Zare, and Somayeh Asadi. Two-stage stochastic programming model for optimal scheduling of the wind-thermal-hydropower-pumped storage system considering the flexibility assessment. *Energy*, 193:116657, 2020.

Justin Dumouchelle, Rahul Patel, Elias B. Khalil, and Merve Bodur. Neur2SP: Neural two-stage stochastic programming. In *Advances in Neural Information Processing Systems*, volume 35, pages 23992–24005. Curran Associates, Inc., 2022. doi: 10.52202/068431-1742.

Gopalakrishnan Easwaran and Halit Üster. Tabu search and Benders decomposition approaches for a capacitated closed-loop supply chain network design problem. *Transportation Science*, 43(3): 301–320, 2009.

Matteo Fischetti, Domenico Salvagnin, and Arrigo Zanette. A note on the selection of Benders' cuts. *Mathematical Programming*, 124(1-2):175–182, 2010.

Matteo Fischetti, Ivana Ljubić, and Markus Sinnl. Benders decomposition without separability: A computational study for capacitated facility location problems. *European Journal of Operational Research*, 253(3):557–569, 2016.

Matteo Fischetti, Ivana Ljubić, and Markus Sinnl. Redesigning Benders decomposition for large-scale facility location. *Management Science*, 63(7):2146–2162, 2017.

Arthur M Geoffrion and Glenn W Graves. Multicommodity distribution system design by benders decomposition. *Management Science*, 20(5):822–844, 1974.

Huiwen Jia and Siqian Shen. Benders cut classification via support vector machines for solving two-stage stochastic programs. *INFORMS Journal on Optimization*, 3(3):278–297, 2021.

Esmaeil Keyvanshokooh, Sarah M Ryan, and Elnaz Kabir. Hybrid robust and stochastic optimization for closed-loop supply chain network design using accelerated Benders decomposition. *European Journal of Operational Research*, 249(1):76–92, 2016.

Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems*, 30, 2017.

Zhe Li, Bernard T. Agyeman, Ilias Mitrai, and Prodromos Daoutidis. Learning to control inexact Benders decomposition via reinforcement learning. *arXiv preprint arXiv:2508.06700*, 2025.

Zhuwen Li, Qifeng Chen, and Vladlen Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. *Advances in Neural Information Processing Systems*, 31, 2018.

Thomas L. Magnanti and Richard T. Wong. Accelerating Benders decomposition: Algorithmic enhancement and model selection criteria. *Operations Research*, 29(3):464–484, 1981.

Arthur Maheo, Philip Kilby, and Pascal Van Hentenryck. Benders decomposition for the design of a hub and shuttle public transit system. *Transportation Science*, 53(1):77–88, 2019.

Kyle Mana, Stephen Mak, Parisa Zehtabi, Michael Cashmore, Daniele Magazzeni, and Manuela Veloso. Accelerating Benders decomposition via reinforcement learning surrogate models. In *ICAPS 2023 Workshop on Planning and Scheduling for Financial Services (FinPlan)*, 2023.

Vinod Nair, Krishnamurthy Dvijotham, Iain Dunning, and Oriol Vinyals. Learning fast optimizers for contextual stochastic integer programs. In *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence*, pages 591–600. AUAI Press, 2018. URL `https://auai.org/uai2018/proceedings/papers/217.pdf`.

Chandra A Poojari and John E Beasley. Improving Benders decomposition using a genetic algorithm. *European Journal of Operational Research*, 199(1):89–97, 2009.

Cristian Ramírez-Pico, Ivana Ljubić, and Eduardo Moreno. Benders adaptive-cuts method for two-stage stochastic programs. *Transportation Science*, 57(5):1252–1275, 2023.

Huangrong Sun, Xian Yu, and Güzin Bayraksan. Contextual stochastic optimization for determining electric vehicle charging station locations with decision-dependent demand learning. *Transportation Research Part B: Methodological*, 211:103524, 2026.

Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *International Conference on Machine Learning*, pages 9367–9376. PMLR, 2020.

Richard M Van Slyke and Roger Wets. L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17(4):638–663, 1969.

Qianfan Wang, Yongpei Guan, and Jianhui Wang. A chance-constrained two-stage stochastic program for unit commitment with uncertain wind power output. *IEEE Transactions on Power Systems*, 27 (1):206–215, 2012.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

Xian Yu, Siqian Shen, and Huizhu Wang. Integrated vehicle routing and service scheduling under time and cancellation uncertainties with application in nonemergency medical transportation. *Service Science*, 13(3):172–191, 2021.

Xian Yu, Siqian Shen, Babak Badri-Koohi, and Haitham Seada. Time window optimization for attended home service delivery under multiple sources of uncertainties. *Computers & Operations Research*, 150:106045, 2023.

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 clearly state the paper's contributions and experimental scope. The claimed improvements are supported by the benchmark results and are limited to the two-stage stochastic EV charging station location settings. The paper does not make unsupported claims about performance beyond the evaluated instances.

   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: The paper discusses the scope and limitations of RLBD, including its problem setting, evaluation procedure, generalization scope, and computational scaling.

   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: [N/A]

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: The paper provides the problem formulation, algorithmic details, training and testing procedures, hyperparameter settings, benchmark setup, and evaluation metrics needed to reproduce the main experimental results.

Guidelines:

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

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

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

   Answer: [Yes]

   Justification: Our code is available at `https://github.com/HaochenCai/RLBD`, which includes the generated data, instructions for reproducing the main experiments, the training, evaluation, and benchmark procedures. This link is included at the end of the Abstract.

   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: The paper specifies the experimental setup, including the problem instances, training and testing procedures, hyperparameter choices, baselines, and evaluation metrics.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: The paper reports 90% confidence intervals for the main experimental curves and averages benchmark results over multiple testing seeds, with the variability source described in the text.

   Guidelines:

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

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: The paper reports the computing environment and experimental time limits used for the benchmark experiments, including the use of CPU-based optimization runs and a one-hour time limit for each test instance.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The research follows the NeurIPS Code of Ethics. It is a computational optimization study using synthetic benchmark data and does not involve human subjects, private data, or sensitive attributes.

   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: The paper discusses positive impacts through more efficient stochastic optimization and potential negative impacts related to over-reliance on automated optimization decisions without proper oversight.

Guidelines:

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

11. **Safeguards**

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

Answer: [N/A]

Justification: The work does not release high-risk models, scraped datasets, or generative systems. The experiments use optimization benchmark and synthetic data, so these safeguard concerns are not applicable.

Guidelines:

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

12. **Licenses for existing assets**

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

Answer: [Yes]

Justification: The paper cites the original sources of the existing benchmark datasets and prior methods used for comparison. It also distinguishes publicly available benchmark instances from the synthetic instances generated in this work.

Guidelines:

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

13. **New assets**

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

    Answer: [Yes]

    Justification: The paper documents the new experimental assets introduced in the work, including the synthetic instance generation procedure, benchmark settings, and implementation details needed to reproduce the experiments.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing, human-subject experiments, user studies, or participant data collection. The work is based on computational optimization experiments.

    Guidelines:

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

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

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

Answer: [N/A]

Justification: The paper does not involve human subjects, crowdsourcing, user studies, or participant data collection. Therefore, IRB approval or equivalent review is not applicable.

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: The research does not use LLMs as part of the core methodology, experiments, or scientific contribution. Any LLM use was limited to writing or editing assistance.

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.

Appendix

A Two-stage Stochastic EV Charging Station Location problem

Let $\mathcal{I}$, $\mathcal{J}$, and Ω denote the set of candidate charging stations, customer sites, and demand scenarios, respectively. Define the first-stage decision variables as $y_i \in \{0,1\}$ and $z_i \in \mathbb{Z}_+$ for all $i \in \mathcal{I}$, where $y_i = 1$ if station i is selected to open and z_i denotes the number of chargers installed at station i. Define the second-stage decision variables x_{ij} and s_j as the amount of demand at customer site $j \in \mathcal{J}$ fulfilled by charging station i and the unmet demand at customer site j for all $i \in \mathcal{I},\ j \in \mathcal{J}$, respectively. Parameters $f_i, b_i, c_{ij}, p_j, r_j, C_i$ denote the investment cost for opening station $i \in I$, unit cost for installing one charger at station $i \in I$, unit transportation cost from station $i \in I$ to customer site $j \in J$, unit penalty for unmet demand at customer site $j \in J$, unit revenue for demand fulfillment at site $j \in J$, and the maximum number of EV demand each charger can fulfill at location $i \in I$, respectively.

Following Sun et al. [2026], the two-stage stochastic EV charging station location model is given by

$$\min_{y,z} \quad \sum_{i\in\mathcal{I}} f_i y_i + \sum_{i\in\mathcal{I}} b_i z_i + \frac{1}{|\Omega|}\sum_{\omega\in\Omega} Q^\omega(z) \tag{7}$$

$$\text{s.t.} \quad z_i \le M_i y_i,\ \forall i \in \mathcal{I} \tag{8}$$

$$y_i \in \{0,1\},\ z_i \in \mathbb{Z}_+,\ \forall i \in \mathcal{I} \tag{9}$$

where constraints (8) enforce that chargers can be installed at location $i \in \mathcal{I}$ only if a charging station is opened there, and that the number of installed chargers at each location i is bounded above by M_i. For each scenario $\omega \in \Omega$, the recourse function is defined as

$$Q^\omega(z) = \min_{x,u} \quad \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} c_{ij}x_{ij} + \sum_{j\in\mathcal{J}} p_j u_j - \sum_{j\in\mathcal{J}} r_j d_j^\omega \tag{10}$$

$$\text{s.t.} \quad \sum_{i\in\mathcal{I}} x_{ij} + u_j = d_j^\omega,\ \forall j \in \mathcal{J} \tag{11}$$

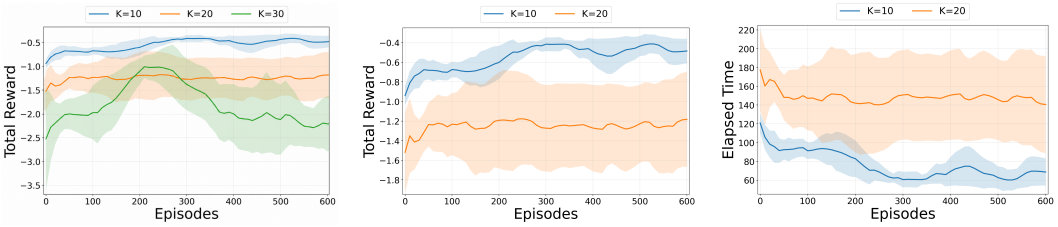
$$\sum_{j\in\mathcal{J}} x_{ij} \le C_i z_i,\ \forall i \in \mathcal{I} \tag{12}$$

$$x_{ij} \ge 0,\ u_j \ge 0,\ \forall i \in \mathcal{I},\ j \in \mathcal{J} \tag{13}$$

where constraints (11) ensure that the demand at each customer site $j \in \mathcal{J}$ is either satisfied by an open charging station or accounted for as unmet demand. Constraints (12) enforce capacity limits by requiring that the total demand served at each open station $i \in \mathcal{I}$ does not exceed the product of the number of installed chargers z_i and the per-charger capacity C_i. Finally, constraints (13) impose non-negativity on both satisfied and unmet demand variables.

In our experiments, we set $|\mathcal{I}| = 8$ and $|\mathcal{J}| = 12$. The other parameters are uniformly sampled within the following ranges: $f_i \in [80, 300]$, $b_i \in [5, 40]$, $c_{ij} \in [1, 80]$, $p_j \in [30, 120]$, $r_j \in [5, 60]$, $C_i \in [40, 120]$, and $M_i \in [10, 80]$.

B Additional Numerical Results



(a) Total Reward (b) Total Reward (Close-up) (c) Elapsed Time (Close-up)

Figure 6: Total rewards for $K \in \{10, 20, 30\}$ and close-up comparisons of total reward and elapsed time for $K \in \{10, 20\}$.