Reinforcement Learning for Best Subset Optimization

Kameel Dossal

Brown University, 2024

Honors Thesis

Submitted in fulfillment of the requirements for Honors in Statistics from Brown University

This thesis by Kameel Dossal is accepted in its present form
by the Brown University Department of Biostatistics as satisfying the thesis
requirements for the degree of Bachelors of Science with Honors.

Date 4/30/24 [illegible]

Alice Paul, Advisor

Date 04/30/2024 Ying Ma

Ying Ma, Reader

Date ______ ______________________

Roee Gutman, Director of Undergraduate Studies

Acknowledgements

I owe a huge thanks to Professor Alice Paul for her guidance and insights throughout my thesis journey. Her feedback and advice weren't just invaluable to the completion of this work, but also to shaping my path as a researcher. Professor Paul provided me with the opportunity to develop my skills steering me towards my passion in the field.

I would like to thank my family and partner for their unwavering love and support. Their sacrifices have not gone unnoticed and have been a driving force behind my academic pursuits. To my sisters and cousins, thank you for your encouragement and for always believing in me, you motivate me to continue my path.

I'm also grateful to my friends and peers who have provided a supportive community, especially those I've met along my journey at Brown University. Your company has been a source of reassurance in times of stress. Thank you all for your part in this journey.

Abstract

Background: Traditional methods for variable selection in high-dimensional data analysis, such as Stepwise Selection and Lasso, are computationally efficient but provide only approximate solutions. This thesis investigates the application of Reinforcement Learning to optimize the Branch and Bound algorithm used in solving Best Subset Selection, aiming to overcome the computational intensity associated with traditional heuristic branching rules.

Methodology: We developed a Deep Q-Learning model to generate RL-based branching rules, evaluating its performance against traditional methods across synthetically generated datasets designed to simulate real-world complexities. Several training methodologies were implemented and evaluated to provide a deeper understanding of model generalizability and efficiency in navigating the solution space of the B&B algorithm.

Results: Our RL-based branching rule demonstrated superior performance to traditional rules regarding convergence rates and reduced variance across different datasets. Feature importance analysis revealed that the model builds on heuristic rules and relies on refining global bounds and dataset-specific statistics for branching decisions.

Conclusion: This study highlights the potential of integrating RL with the B&B algorithm to improve best subset variable selection, challenging conventional branching rules. Future research directions include expanding the dataset size and complexity for training and exploring the scalability of the RL model to higher-dimensional datasets.

Contents

1 Introduction

Optimal variable selection, particularly in the context of sparse linear regression, has long been a focal point of statistical research. Traditional approaches, such as Stepwise Selection and Lasso (L_1 regularization), have been widely adopted for their computational efficiency, despite these methods only providing an approximation of the best subset. Lasso specifically uses a convex relaxation of best subset selection (L_0 regularization), to perform both regularization and variable selection, substituting the non-convex optimizations required for best subset. This non-convex nature of best subset leads to challenges from high dimensionality, which have previously rendered it impractical for datasets exceeding 30 variables [8]. However, thanks to recent advancements in computational strategies for tackling non-convex optimization problems, best subset selection has resurfaced as a viable and potentially superior alternative [7]. These innovations are largely attributed to the reframing of best subset as a Mixed Integer Quadratic Optimization (MIQO), stemming from developments of efficient methods for solving MIQOs [3].

The conventional method for solving MIQOs, the Branch and Bound (B&B) algorithm, iteratively divides the optimization problem into sub-problems, a process that demands heuristic-guided decisions. These decisions, known as branching rules, are crucial for determining which sub-problem to explore next and which variable within that sub-problem to branch on. Notably, the computational intensity of B&B methods greatly depends on the rule used, and traditional heuristic rules such as Strong Branching present serious challenges regarding scalability and efficiency [4]. In this context, Balcan et al. highlight the potential of machine learning in enhancing these traditional rules.

Within this paradigm, Etheve et al. introduce a novel Reinforcement Learning (RL) methodology, FMSTS, that processes each step of the B&B algorithm into states and generates a branching rule utilizing an RL model. This marked the first application of RL in fully optimizing branching strategies and was shown to be both effective and generalizable across different problem instances, showcasing the potential for RL to discover new and efficient branching policies. Building on this, Parsonson et al. propose an innovative approach, retro-branching, addressing challenges of sparse rewards and complex decision spaces and demonstrate the potential for RL to outperform traditional rules. This thesis further extends RL's application for refining the branching process, by introducing updated state and reward definitions and proposing an alternative retro-branching structure.

To test our design, we generate three collections of synthetic datasets. The first, designed for training, utilizes static parameters—to ensure consistency—across three variable size configurations: low ($p = 10$), medium ($p = 50$), and high ($p = 100$). We train four distinct models, each on a specific configuration and one encompassing all configurations. The second collection, designed for validation, compares these models and provides a benchmark using the same static configurations. The last collection, reserved for testing, utilizes dynamic parameters to simulate real-world data complexity. We then compare our best RL model against traditional Max Fraction Branching and Strong Branching decision rules. Through this methodology, the study aims to demonstrate RL's potential for improving the B&B algorithm and advancing methods for variable selection.

We first formalize best subset selection as an MIQO problem and detail the Branch and Bound (B&B) algorithm in section 2. We then detail our RL model in section 3, which provides background on Deep Q-Learning and details the construction of our agent, covering model architecture, algorithmic design, and the rationale for hyperparameter selection. Section , Methodology, outlines the experimental design, including data generation, model training, validation, and evaluation against traditional rules. Lastly, we present experimental findings in section 5, Results and Analysis section, focusing on the performance, generalizability, and feature importance of our RL model.

2 Background

2.1 Best Subset as an MIQO

The best subset problem is defined as minimizing the sum of squared residuals subject to a constraint on the number of non-zero coefficients, represented by the L_0 norm ($\|\cdot\|_0$), to a predefined subset size, k. The problem is traditionally formalized as:

$$\underset{\beta}{\text{minimize}} \ \frac{1}{2}\|Y - X\beta\|_2^2 \quad \text{s.t.} \quad \|\beta\|_0 \leq k \tag{1}$$

where Y signifies the response vector, X a matrix of p predictors by n observations, and β the coefficient vector. This can also be rewritten in Lagrangian form as:

$$\underset{\beta}{\text{minimize}} \ \frac{1}{2}\|Y - X\beta\|_2^2 \quad + \lambda_0\|\beta\|_0 \tag{2}$$

Bertsimas et al. redefine this approach by viewing best subset selection as a series of MIQOs. By solving the minimization of a quadratic objective function under linear constraints, with a subset of variables required to take on integer values, explicitly 0 for excluded and 1 for selected, variable selection can be reworked as an MIQO problem. The simplest formulation of this from their work is the 'Big M' formulation, which introduces a novel way to bound the coefficient search space with a constant, M_u, ensuring M_u exceeds the absolute values of the optimal coefficients. The MIQO problem is thus defined as:

$$\begin{aligned}
&\underset{\beta,Z}{\text{minimize}} \ \frac{1}{2}\|Y - \beta X\|_2^2 \\
&- M_u z_i \leq \beta_i \leq M_u z_i \quad \forall i = 1, 2, ..., p \\
&z_i \in \{0, 1\} \quad \forall i = 1, 2, ..., p \\
&\sum_{n=1}^{p} z_i \leq k
\end{aligned} \tag{3}$$

Where each z_i is a binary variable indicating the inclusion ($z_i = 1$) or exclusion ($z_i = 0$) of the i^{th} variable in the optimal subset. This formulation translates the original best subset selection problem into an MIQO framework, enabling the application of problem-specific optimization methods for a task previously deemed computationally intractable. These methods utilize a solver to find solutions to relaxations of the original optimization problem, without the integer constraint on Z values, to explore the search space effectively.

2.2 Branch and Bound

Currently, the most prominent solvers used for this framing of best subset selection is the commercial software l1cd [14], which finds the set of values of β and Z that are potential solutions to the relaxation. This set is called the feasible region, with each element being a feasible solution and those also satisfying the original integer constraints on Z being referred to as integer feasible solutions. This method of dividing the solution space is known as cutting planes. Notably, the relaxation imposes interval constraints on Z, ensuring they remain within the bounds of 0 and 1, thus reducing the solution space for optimization at each step. The challenge of this algorithm stems from adhering to these constraints while seeking optimal solutions by successfully minimizing the 'Big M' formulation.

The cutting plane method is then continuously used when relaxation solutions fail to meet the integer constraints on Z. This iteratively narrows the feasible region by segregating it into two: one inclusive of the relaxation's optimal solution and another containing integer feasible solutions. The former then becomes the updated feasible region and the process repeats until solved. This method, while effective for linear objectives, encounters limitations with quadratic functions due to the difficulty in isolating integer solutions using linear constraints, particularly under the specified $\{0, 1\}$ constraints on Z. This scenario necessitates the use of the B&B algorithm as a more adaptable solution.

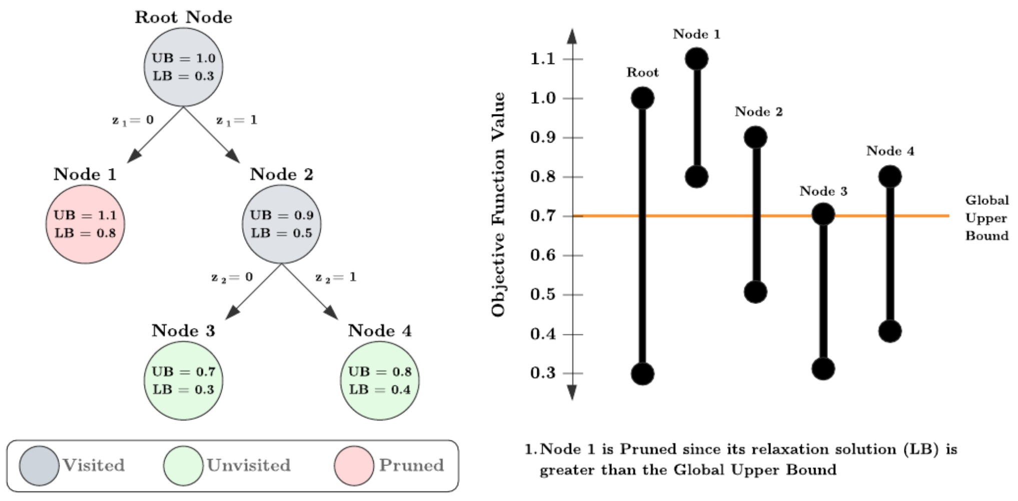
Diverging from cutting planes, B&B facilitates the division of the initial feasible region into two regions that may each contain integer feasible solutions. This process, referred to as branching, allows the algorithm to be visualized as a decision tree. The root of this tree represents the entire feasible region, and each child node is an updated feasible region. The relaxation is then solved, providing a lower bound (LB), the best possible relaxation value, on the optimal integer feasible solutions in each region. Once an integer feasible solution is found, either directly by solving or using a heuristic to find one from the relaxation solution, it becomes the current upper bound (UB). The UB is then updated to the best current integer solution as more integer-feasible solutions are found, and all regions with an LB greater are eliminated from consideration, this is called pruning.

This process, through iterative branching and solving relaxations, systematically narrows down the search space. Progression of B&B is tracked through the optimality gap:

$$\text{optimality gap} = \frac{min(\text{UB}) - min(\text{LB})}{min(\text{LB})} \tag{4}$$

which is gradually minimized as the algorithm iterates towards convergence, where the minimum upper bound matches the minimum lower bound, to a specified threshold, signaling the identification of an optimal integer solution. All B&B algorithms shown in this paper utilize a threshold of $1e^{-4}$. The B&B process is illustrated in Figure 1.

Figure 1: B&B Algorithm



Hazimeh and Mazumder developed the 'L0BnB' package, which uses coordinate descent and local combinatorial optimization to quickly approximate solutions to the relaxation. In our setting, the relaxation is solved using this package, the lower and possibly upper bounds are determined, and any necessary nodes in the tree are pruned. Subsequently, the algorithm selects the next node-variable pair for branching, dividing the original problem into two more tractable sub-problems, each representing a binary decision on the inclusion or exclusion of variables in the optimal subset.

Importantly, the decision on which variable and node to branch on is decided using heuristic branching rules, which significantly impact the depth of the B&B tree, and therefore the computational intensity of the algorithm. Among the prevalent methods, max fraction branching targets variables with z values furthest from being integers, aiming for immediate resolution of ambiguity. In contrast, pseudo-cost branching estimates the impact of each variable's inclusion or exclusion based on past branching decisions, calculating an average change in the objective value to guide future decisions [1]. Despite their utility, some of the most effective branching strategies, notably strong branching, entail substantial computational resources. Strong branching exhaustively evaluates a subset of branching variables by solving two child nodes for each and selecting the variable that leads to the largest expected improvement in the optimality gap. Although this process leads to a more efficient search through the solution space, it also necessitates a comprehensive computation that drastically escalates with the problem's dimensionality [4].

Transitioning from heuristic branching strategies, RL introduces an adaptive and problem-specific branching rule. By applying RL, the agent utilizes iterative learning from the outcomes of previous branching decisions to develop a policy that can anticipate the most effective paths without exhaustive computations. This approach offers a scalable solution to the computational difficulty in high-dimensional variable selection.

3 RL Model Development

Etheve et al. highlight RL's suitability for the B&B algorithm, thanks to its iterative nature providing an ideal learning environment for agents to interact and learn from. The construction of our RL agent utilizes Deep Q-Learning [6], a temporal difference RL algorithm that iteratively interacts with an environment, collecting state information, selecting actions, and receiving feedback through rewards. This model, devised to minimize node exploration, utilizes a single action space, which is used to select the optimal node-variable pair from all viable combinations at each step during the B&B decision tree. This section covers the basics of Deep Q-learning and our implementation.

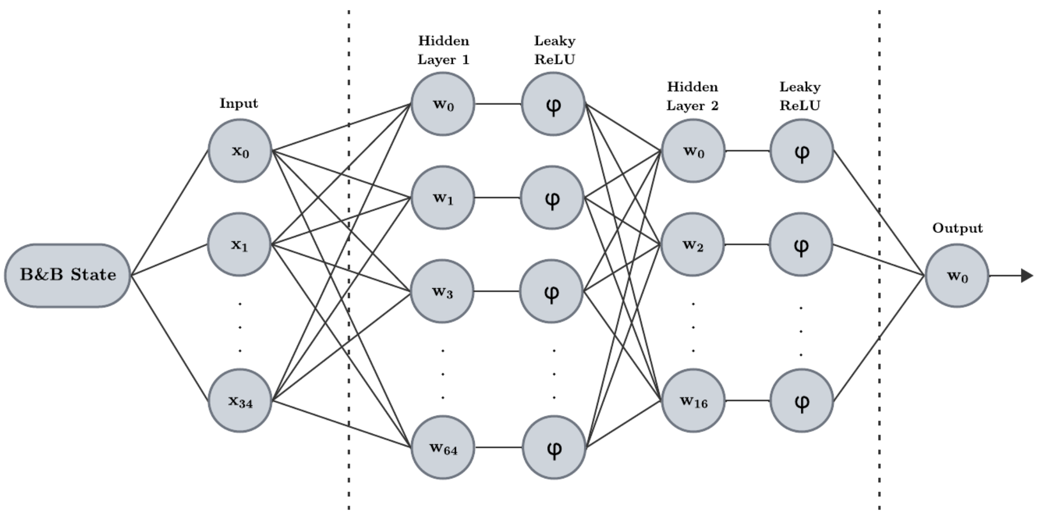
3.1 Q-Learning

Q-learning, the foundation for our model's strategy, is typically used to optimize decision-making within an MDP (Markov Decision Process) framework. This algorithm's core lies in learning the action-value function, or Q-function, which estimates the expected reward of taking a given action in a given state and following a certain policy thereafter. The goal of Q-Learning is to discover the policy that maximizes the total reward over the long term, effectively guiding the agent through the decision space towards optimal outcomes. The Q-learning algorithm is formalized as follows:

$$Q(s_t, u_t) \leftarrow Q(s_t, u_t) + \alpha[r_{t+1} + \gamma \max_{u_{t+1}} Q(s_{t+1}, u_{t+1}) - Q(s_t, u_t)]^2 \tag{5}$$

In the Q-learning formula, $Q(s, u)$ denotes the action-value function that estimates the total expected reward for taking an action u in a state s, with s_t and u_t representing the state and action at a specific timestep t, respectively. The learning rate, α, determines the degree to which the newly acquired information will influence the existing knowledge, essentially balancing the update of the Q-value with new versus old observations. The reward received after taking an action u_t in state s_t is represented by r_{t+1}, serving as the immediate payoff for the agent's action. The discount factor, γ, quantifies the value of future rewards, indicating how much importance is placed on immediate versus subsequent rewards. Lastly, $max_{u_{t+1}}Q(s_{t+1}, u_{t+1})$ signifies the highest reward that is predicted to be attainable in the subsequent state, s_{t+1}, providing a forward-looking measure that influences the current Q-value update, ensuring the model is guided towards decisions that are expected to yield the highest cumulative reward in the long run.

Figure 2: Deep Q-Network Architecture



Note: Upon initialization of our agent, both the Policy and Target networks are created using the above architecture with randomly generated weights.

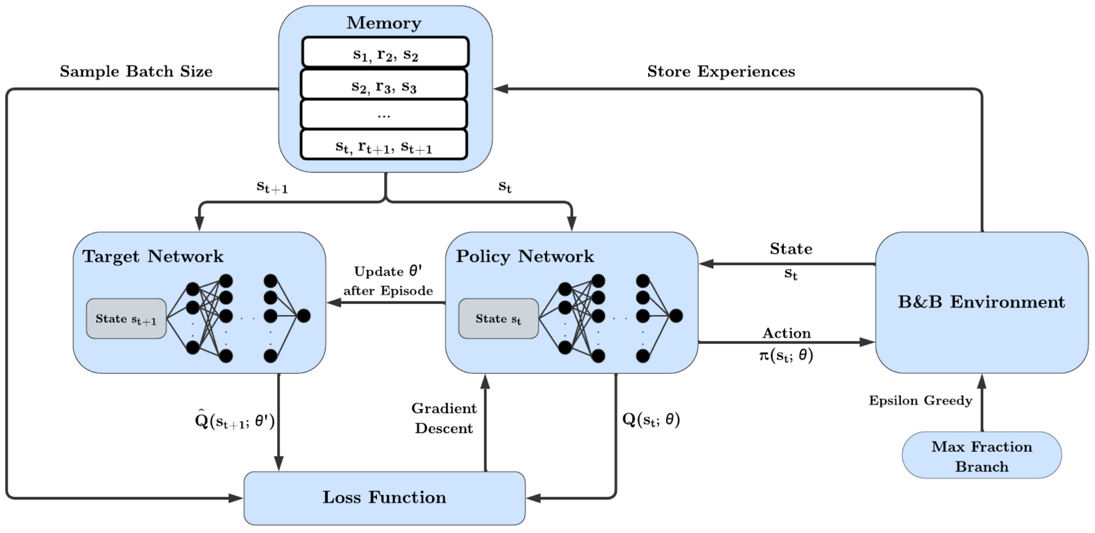
Our model's architecture, a Deep Q-Network (DQN), extends traditional Q-learning by incorporating deep learning to approximate the Q-function. The Deep Q-learning agent comprises a policy network and a target network, respectively tasked with generating Q-value predictions for action selection and stabilizing the learning process. As depicted in Figure 2, the network takes in an input reflective of the B&B process's current state and outputs an estimated Q-value, for each potential node-variable state. Given the number of node-variable pairs is dynamic, our model is designed to estimate Q-values based directly on state inputs. This approach simplifies our notation by eliminating the need to explicitly specify which actions, u, were taken.

To optimize the model using gradient descent, the Q-learning equation is reworked into a loss function defined as:

$$\text{Loss} = [r_{t+1} + \gamma \hat{Q}(s_{t+1}; \theta') - Q(s_t; \theta)]^2 \tag{6}$$

In this formulation, $\hat{Q}$ represents the Q-value predictions from the target network (with weights θ'), and Q corresponds to those from the policy network (with weights θ).

Figure 3: Deep Q-Learning Overview



The algorithmic process visualized in Figure 3 involves the network iteratively interacting with the B&B environment. This interaction includes receiving state information, s_t, and selecting node-variable pairs for branching based on the policy $\pi(s;\theta)$ until the B&B algorithm converges. After completing a round of variable selection, referred to as an episode, the model retroactively records state pairs and rewards into its memory. Details on action selection and memory storage are discussed later in this section.

Following an episode, the model samples from its memory to update the policy network using the loss function. It then sets the target network weights (θ') to those of the policy network (θ), aiding in training stability. An ϵ-greedy strategy is also utilized to balance between exploration and exploitation, initially selecting actions based on the

Max Fraction Branching rule with a probability ϵ. As training progresses and ϵ decreases, the model increasingly favors actions that maximize expected rewards based on current Q-value estimates, shifting towards exploitation with a probability of $1-\epsilon$. This approach ensures a comprehensive search to determine an optimal policy.

Table 1: DQN Model Hyperparameters

Hyperparameter	**Value**
Batch Size	64
Epsilon Decay	0.995
Epsilon Minimum	0.05
Gamma (γ)	0.8

Lastly, model hyperparameters, shown in Table 1, were selected and fine-tuned through experimentation. This involved training models across training configurations and evaluating their ability to discover an optimal decision rule and the number of episodes it took to do so. Through continuously adjusting hyperparameters and recording their impact on the model's performance, we identified a set that showed overall effectiveness across all configurations.

During training, after each B&B tree is completed the RL agent utilizes memory replay [12] to optimize the model's weights. A batch size of 64 was selected, ensuring the model learns from a diverse set of state pairs without drastically increasing training times. The epsilon value, begins at 1 and decays to 0.05, with the decay rate calibrated to transition smoothly over approximately 100 episodes of training. The γ value of 0.8 was chosen to moderate the emphasis on future rewards. In challenging variable selection problems, B&B trees can become drastically larger, introducing a risk of the model becoming overly focused on distant future rewards at the expense of practical, near-term decisions. A lower gamma helps mitigate this, encouraging the model to value immediate rewards more equally and adapt more flexibly to various problem complexities.

3.2 Actions

Our agent utilizes a single action space, estimating Q-values for each available node-variable state using the policy network, $Q(s_t)$. After estimating rewards, the agent uses the policy $\pi(s_t;\theta)$ to select the state that maximizes the estimated reward for branching.

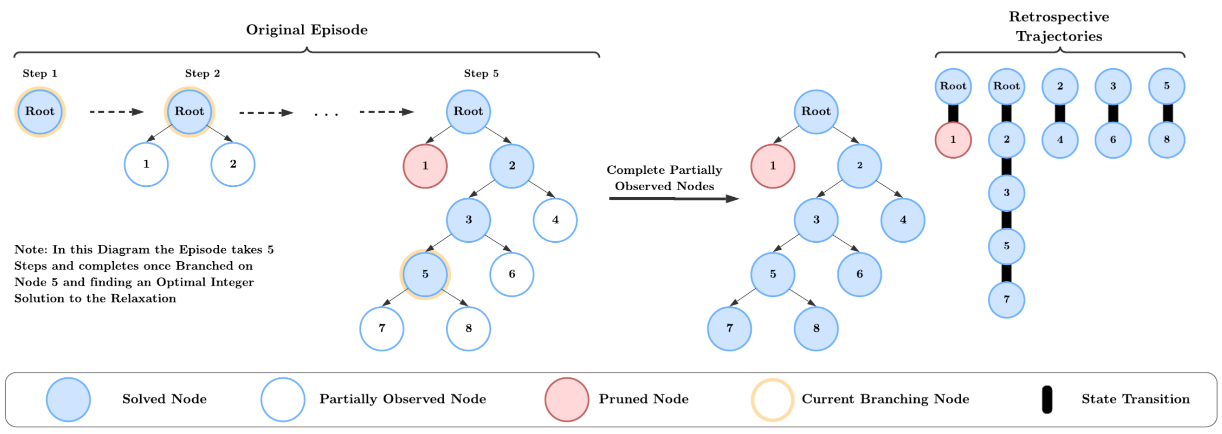
3.3 States

During each step of the B&B algorithm, we record 34 statistics, segmented into four distinct categories: characteristics of the variable selection problem, the current state of the tree, individual node attributes, and specifics of the variables within each active node. This approach is built upon Khalil et al., with the addition of specific statistics to provide a comprehensive view of the variable selection process. These statistics are compiled into a state vector, serving as the input for our DQN.

Table 2: Statistics used Describe a State of B&B

Statistic Type	**Statistic Name**
Problem Statistics (12)	5 quantiles of covariances of X 5 quantiles of $X \cdot y$ Variable count Initial optimality gap
Tree Statistics (6)	Steps taken Number of active nodes Length of candidate solution variables Global lower bound Global upper bound Current optimality gap
Node Statistics (7)	Support length of the relaxation solution Whether a node has the global lower bound Whether a node has the global upper bound Node level z_{ub} length z_{lb} length Node specific lower bound
Variable Statistics (9)	5 quantiles of x_i covariance with all other x_j $x_i \cdot y$ Lower z_i value z_i value Upper z_i value

Figure 4: Updated Retro-Branching


Original Episode
Step 1
Step 2
Step 5
Root
1
2
3
4
5
6
7
8
Note: In this Diagram the Episode takes 5 Steps and completes once Branched on Node 5 and finding an Optimal Integer Solution to the Relaxation
Complete Partially Observed Nodes
Retrospective Trajectories
Solved Node
Partially Observed Node
Pruned Node
Current Branching Node
State Transition

3.4 Retro-Branching.

Using conventional B&B solvers, nodes are solved through branching, contingent on a variable within a node being selected. This approach, however, leads to leaf nodes being partially explored, due to not being branched on and solved, preventing the generation of states from these nodes. To address this, we revisit partially observed nodes, selecting variables within each to solve. This ensures the derivation of complete, albeit non-optimal, states for nodes previously left unresolved, as each node is limited to finding a solution via the variables present in its support set. Utilizing the completed tree, we then implement an updated version of the retro-branching methodology, building off of 13. Retro-branching significantly augments our model's learning capability by retrospectively constructing trajectories that are linked to the state of previous nodes. This facilitates focused learning from states with high relevance to each other. This learning strategy, depicted in Figure 4, showcases the construction of said trajectories. Where each connection between nodes captures a state s_t, next state s_{t+1}, and reward r_{t+1}.

Notably, to mitigate the impact of variance in tree lengths—which could disproportionately influence the RL agent's memory due to episodes of varying lengths stemming from inconsistent performance—a cap of 128 state pairs is sampled from each episode. This approach ensures a balanced representation in the memory, preventing episodes associated with suboptimal branching from dominating the learning dataset.

3.5 Rewards

Our model's reward system integrates both a binary and continuous structure to evaluate the efficacy of each branching decision. The binary component assigns a reward based on whether a branching action leads to a terminal (leaf) node within the Branch and Bound (B&B) tree. Specifically, a penalty, $r_1 = -1$, is applied for actions that do not result in a terminal node, while achieving a terminal node halts further branching at that path, warranting a $r_1 = 0$ reward. This aspect of the reward structure incentivizes the agent towards actions that efficiently prune the search space, aiming for solutions that minimize the tree's breadth and depth, reflecting an approach to expedite the attainment of an optimal solution [13].

The continuous reward categorizes changes in the optimality gap (opt_gap), into three distinct buckets. Given the diminishing magnitude of opt_gap changes as the B&B process unfolds, these categories are defined as follows:

$$r_2 = \begin{cases} 1 & \text{if } 0 < \text{opt\_gap} \leq 0.05 \\ 2 & \text{if } 0.05 < \text{opt\_gap} \leq 0.15 \\ 3 & \text{if opt\_gap} > 0.15 \\ 0 & \text{otherwise, if the opt\_gap was not reduced} \end{cases}$$

To compute the overall reward linked to each branching action, we aggregate r_1 and r_2. This composite reward structure, drawing inspiration from the retro-branching methodology and the importance of changes in the optimality gap in variable selection, aims to drive the RL agent towards branching choices that not only shorten the B&B tree but also expedite the convergence to an optimal solution.

3.6 Algorithmic design

Utilizing our reward structure, action space, collected state statistics, and DQN architecture we can outline the structure of our RL model within the B&B framework for best subset variable selection. The algorithms presented below describe a single episode's progression and training.

Algorithm 1 Algorithm for a Single Episode

```
1: Initialize episode: $X, y, \lambda_0, \epsilon$, Threshold
2: $M = 1.5 \cdot \max\left(\left|X^\intercal X\right|^{-1} X^\intercal y\right)$
3: Solve relaxation
4: Calculate initial opt_gap
5: while opt_gap > Threshold do
6:     $k \sim \text{Uniform}(0, 1)$
7:     if $k \leq \epsilon$ then
8:         Select $(\text{node}, x_j)$ via max fraction branching
9:     else
10:        for Available node, $x_j$ pairs do
11:            Generate state $s_t$
12:            Estimate Q-value using $Q(s_t; \theta)$
13:        end for
14:        Use policy $\pi(s_t; \theta)$ to select $(\text{node}, x_j)$ with largest Q-value
15:    end if
16:    Branch on selected $(\text{node}, x_j)$
17:    Solve relaxation
18:    Update opt_gap
19:    Prune nodes if needed
20: end while
21: if Training then
22:     Perform Memory Replay
23: end if
24: return solution coefficients $\beta$
```

Algorithm 2 Memory Replay Algorithm

```
1: Decay $\epsilon$
2: for Leaf node in B&B tree do
3:     Select $x_j$ from leaf node using max fraction branching
4:     Solve relaxation
5: end for
6: Retro-branch to fill Memory with $(s_t, r_{t+1}, s_{t+1})$ from completed tree
7: Sample batch size from Memory
8: Optimize $\theta$ via gradient descent on $\left[r_{t+1} + \gamma \hat{Q}(s_{t+1}; \theta') - Q(s_t; \theta)\right]^2$
9: Set $\theta' = \theta$
```

4 Methodology

4.1 Data Generation

To examine the effectiveness of RL in variable selection, we generated synthetic data, simulating various settings to assess the performance of our RL models across distinct scenarios. Our approach divided the datasets into three distinct compilations: a training and validation compilation, both under static conditions, and a testing compilation, designed to mimic more dynamic realistic data for evaluating our final model.

Each training and validation dataset contains an observation matrix X with $n = 1000$ entries and a predictor count $p \in \{10, 50, 100\}$, designed to represent low, medium, and high-dimensional settings, respectively. To simplify training, we omitted binary predictors; however, these were incorporated within the test datasets, by rounding generated predictors, to assess the model's robustness in handling diverse data types. The construction of each predictor followed a multivariate normal distribution, centered at zero, mathematically represented as:

$$x_i \sim N(0, \Sigma) \tag{7}$$

where Σ encapsulates an exponential correlation with coefficient ρ, formalized by:

$$\Sigma_{ij} = \rho^{|i-j|} \tag{8}$$

The response variable y was constructed as a linear combination of a selected 10% of the predictors, plus a normally distributed error term ϵ:

$$y = \beta_0 X + \epsilon \tag{9}$$

where $\epsilon \sim N(0, \sigma^2)$ and β_0 is a vector of length p with every tenth entry set to 1 and the remainder to 0. After y is then centered about zero and scaled to have unit L2-norm. The signal-to-noise ratio (SNR), a critical measure of dataset quality, was also defined as:

$$\text{SNR} = \frac{\text{var}(X\beta_0)}{\sigma^2} \tag{10}$$

Table 3: Parameters used for Data Generation

Collection	**Setting**	n	p	**SNR**	ρ	**Binary %**
Training & Validation	low	1000	10	5	0.5	0
	medium	1000	50	5	0.5	0
	high	1000	100	5	0.5	0
Testing	—	500-1500	10-150	2-5	0.5-0.9	0-100

Table 3 presents the parameters for generating our synthetic datasets, dividing them into training, validation, and testing collections. To perform L_0 regularization, a λ_0 value must be specified, which directly influences the sparsity of the solution. Because of this, the training settings—categorized under low, medium, and high—employed static parameters. This was a strategic decision, as an optimal λ_0 value could be found for each setting beforehand through experimentation. This was essential as suboptimal λ_0 values could result in episodes remaining unsolved with the default exploration strategy of max fraction branching, leading to the training on unsolved episodes—which is not preferred. Additionally, poor λ_0 values could foster adverse behaviors where the model might prematurely 'give up' in challenging variable selection scenarios, expecting them to be unsolvable. This methodology ensured that the RL agent could efficiently learn and complete episodes, regardless of problem complexities.

Our validation datasets were generated using the same static parameters from training. This allowed us to benchmark our models for hyperparameter tuning, evaluating the policies models developed during training. Contrastingly, the testing datasets were constructed by randomly selecting a combination of parameters derived from the variable ranges outlined. This dynamic approach was designed to mirror the complexity and challenges of real-world data, offering a robust assessment of our RL agent's performance and generalizability relative to conventional branching rules.

4.2 Experimental Design

To evaluate the development and generalizability of our RL models, we trained four distinct models under different settings: low, medium, high, and a mixed model that encapsulated all settings. Each model underwent 100 episodes of training—corresponding to 100 uniquely generated synthetic datasets for each setting. This decision was based on empirical observations as 100 episodes was the minimum number for our RL agent to develop effective strategies in each setting. Following training, a validation set of 25 episodes was created for each setting. Using this set we compared the different RL models, to discern the efficacy of different training approaches, and established a baseline against traditional branching rules.

Table 4: Parameters used for Data Generation

Setting	$\boldsymbol{\lambda_0}$	$\boldsymbol{M}$
low	~ 0.01	~ 1.35
medium	~ 0.065	~ 0.60
high	~ 0.017	~ 0.50

Prior to training, we determined optimal M values, as outlined by Hazimeh et al., and calculated the maximum λ_0 for each dataset within the respective settings using the following formula.

$$\lambda_{0\text{ max}} = \frac{\max(|X \cdot y)|)}{2} \tag{11}$$

The initial $\lambda_{0\text{ max}}$ was then fine-tuned to balance between model performance and solution sparsity. The approximate λ_0 and M values used for each setting are shown in Table 4. Post-training, the datasets and policy networks were archived for hyperparameter tuning, model comparisons, and to ensure reproducibility.

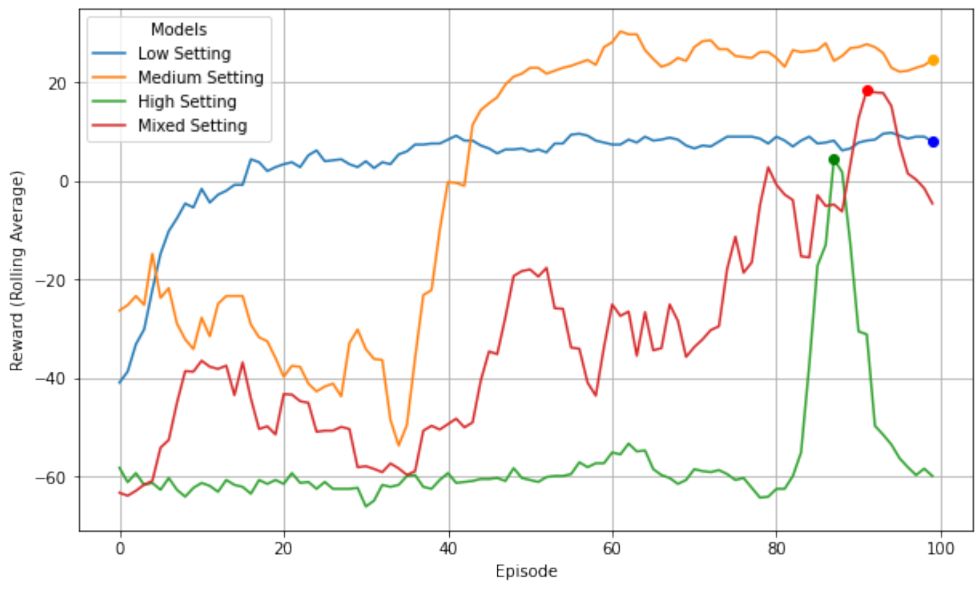
For testing, we dynamically generated 25 datasets, comparing the top-performing RL model against traditional branching methods. This phase aimed to test the model's real-world applicability. While the process for determining an optimal M mirrored that of training, the random nature of dataset generation necessitated a different approach for determining λ_0 values. After identifying the maximum $\lambda_{0\,\max}$, we ran a series of experiments using traditional heuristic branching rules on the testing datasets measuring the convergence frequency. This process involved applying a large range of fractions of the maximum λ_0, and gradually narrowing this range to identify a set that yielded the highest convergence rates with traditional rules. This led us to a set of fractions of this maximum—$[0.3, 0.15, 0.1, 0.05]$—which reliably converged across the randomly generated datasets, establishing an equitable comparison between the traditional and RL branching rules. This approach, applying four different λ_0 values to each dataset, culminated in 100 episodes which were compared for our final results.

5 Results and Analysis

5.1 Model Performance during Training

During training, we assessed the performance of four distinct models, each tailored to a specific setting—low, medium, high, and mixed—over 100 episodes. The analysis of model performance, depicted in Figure 5, reveals significant variations in the range of rewards across these settings. This variability can be attributed to our reward calculation methodology. In the low setting, the relatively fewer number of iterations leads to fewer negative rewards tied to branches failing to reduce the optimality gap. Conversely, in the high setting, the high number of iterations resulted in lower rewards due to the increased likelihood of branches not impacting the optimality gap. However, as the difficulty of the setting increases, the opportunity to make larger changes in the optimality gap also increases. The medium setting presents the highest rewards by balancing the frequency of iterations with opportunities to greatly reduce the optimality gap.

Figure 5: Model Rewards During Training

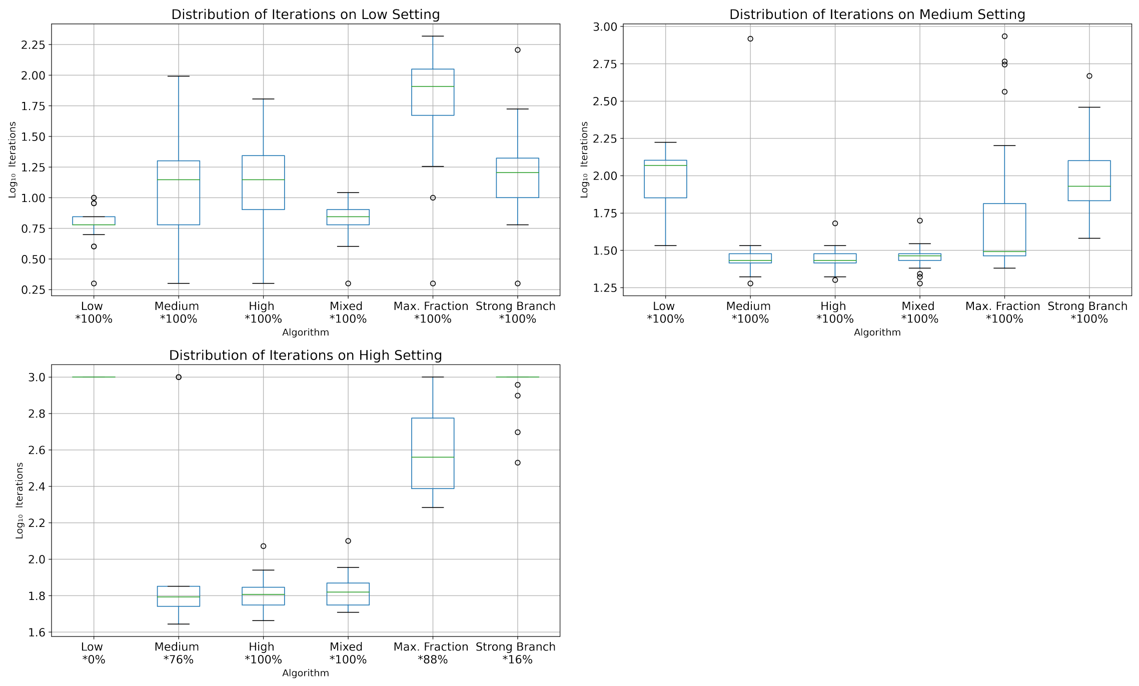


Note: This figure illustrates the rolling average total reward per episode across different models. The points indicate when training was ended and the model was saved.

As expected, the complexity of the setting directly influenced the RL agent's learning trajectory and the duration needed to devise an optimal strategy. Stability becomes an increasing concern as setting difficulty rises, as seen with the high-setting model trends, making training very erratic. Interestingly, the mixed-setting model exhibited a far more progressive learning curve, sequentially mastering the challenges presented by the low, medium, and then high-difficulty datasets.

Catastrophic forgetting, a phenomenon akin to overfitting in Reinforcement Learning, where a model abruptly forgets previously learned strategies, is a key sign of training instability. This necessitated the implementation of early stopping for the models in the High and Mixed settings, to ensure the preservation of peak model performance before overfitting. Both models showed a fleeting grasp of an effective policy for high-setting datasets before experiencing forgetting. Despite this, the mixed model had continued success in less complex settings hinting at the necessity for distinct strategies across different problem sizes. Following the training of these models, the next phase involved evaluating their generalizability across settings through a set of validation datasets.

Figure 6: Validation Performance



Note: * Indicates the percentage of Episodes that converged before 1,000 Iterations. Episodes that reached this limit were terminated, this was done to prevent excessive run times.

Figure 6 presents the comparative performance of our four trained RL models against two conventional branching rules: strong and max fraction. Each model excelled in the validation settings corresponding to their training environments. Despite varied performance across different settings, the specialized models were competitive with conventional branching rules, showcasing satisfactory generalizability.

Notably, the mixed-setting model matched the performance of the specialized models across all settings, indicating a more adaptable policy was found. This model, in particular, demonstrated superior efficiency and consistency compared to the traditional rules, consistently finding solutions within 1,000 iterations. This is best shown in the High setting, where the High and Mixed Models achieved solutions roughly tenfold faster than Max Fraction and Strong Branching. Moreover, these models attained 100% convergence, a stark contrast to the 88% for max fraction and 16% for strong branching. Overall, the mixed model showed exceptional promise in outperforming heuristic methods, positioning it as the primary candidate for the subsequent testing phase.

5.2 Evaluating RL Branching Rule

Table 5: Branching Rule Performance on Testing Data

Branching Rule	Completed Episodes %	Solved	Unsolved
		Iterations[2]	Optimality Gap[3]
RL Mixed Model	50%	42.3 (43.8)	0.35 (0.23)
Max Fraction	19%	85.8 (121.73)	0.41 (0.28)
Strong Branch	21%	113.3 (177.47)	0.34 (0.22)

[1]Mean and SD of the number of iterations for episodes completed within 1,000 iterations.

[2]Mean and SD of the final Optimality Gap reached at 1,000 iterations.

Utilizing the testing set of dynamically generated datasets and the range of λ_0 values, we evaluated the performance of our RL Mixed Model branching rule against the traditional max fraction and strong branching rules. These results, as detailed in Table 5, compare the capabilities of each branching rule across 100 Episodes resembling real-world complexities. Our findings reveal significant differences in the ability to solve episodes within the 1,000 iteration threshold. Notably, the RL branching rule demonstrated a remarkable capacity, solving more than twice the number of episodes and reaching solutions within fewer iterations than the traditional branching rules. To assess the instances that remained unsolved by 1,000 iterations we analyzed the final optimality gaps of each episode. Here we see comparable performance between our RL model and Strong Branching, with both marginally outperforming Max Fraction. This outcome suggests that in difficult scenarios characterized by large predictors p and low SNR, the challenge posed to each branching rule was similarly formidable. Interestingly, our RL approach surpasses the trade-offs of traditional methods, outperforming strong branching in solving more episodes and max fraction in terms of efficiency, effectively combining the strengths of both methods without their compromises.

5.3 Feature Importance

To assess the significance of features input to our model, we randomly shuffled observed values of each feature, i, across states, S, while maintaining all other feature values constant. This effectively decouples the specific signal contributed by the shuffled variable, without altering its value distribution. This aims to discern the impact of each variable's signal on the model's decision-making ability. For each feature, we calculate a Mean Square Error (MSE), between the expected Q-values of the original state (s_t) and that of the modified state (s'_t), where feature i undergoes permutation, this is formalized as:

$$MSE_i = \frac{1}{n}\sum_{t=1}^{n}(Q(s_t) - Q(s'_t))^2 \tag{12}$$

$$\text{where } s'_{t,i} = random(i \in S_i)$$

We followed this procedure for the 34 input features, calculating the average MSE across 30 random shuffles to ensure robustness, drawing on a collection of states from five episodes under each training setting. Due to traditional and RL branching rules selecting optimal states for branching, the value ranges for many features were limited, which would underestimate feature importance. To rectify this, a branching rule that randomly selects node-variable pairs to branch on was used to provide a broad set of states for evaluating features' impacts.

Additionally, variables linked to a dataset's dimensionality were intentionally omitted. This exclusion was grounded in the fact that increases in dimensionality negatively skews the DQN's predicted rewards due to the increased episode length. This methodology highlights the features with the most pronounced impact when deciding which node-variable pair to branch on each iteration within an episode. Table 6 presents the top 10 features according to their influence on the MSE. For comparative clarity, the MSE values have been normalized relative to the highest observed MSE.

Table 6: Most Important Features in Mixed Model

Statistic Name	**Norm. MSE**
Whether a Node Has the Global Lower Bound	1.000
Whether a Node Has the Global Upper Bound	0.2564
Current Optimality Gap	0.0742
X Covariance Quantile 50%	0.0164
X Covariance Quantile 75%	0.0144
$X \cdot y$ Quantile 100%	0.0142
X_i Covariance Quantile 50%	0.0088
X_i Covariance Quantile 75%	0.0086
Z value	0.0076
$X_i \cdot y$	0.0067

In examining the impact of features on our Mixed Model branching rule, we observed that the RL policy discovered prominently values branching on nodes having the Global Lower and Upper bounds. Interestingly this is a very different strategy than Max Fraction or Strong Branching as it relies on refining the global bounds to effectively identify branches that can be pruned reducing the number of nodes needed to be explored. The algorithm also relies on various static statistics of the dataset, showcasing that RL can lead to a more data-driven branching rule that adapts to the datasets given.

The model's valuation of the current optimality gap and Z values align with expectations, given their relationship with model rewards and established strategies such as max fraction respectively. Unsurprisingly, features related to a dataset's dimensionality emerged as crucial predictors of future rewards (see Appendix for details). This association likely reflects the RL model's generalizability and understanding of problem complexity across different settings. This analysis reveals that our RL agent has developed adaptable, data-driven policies for variable selection, capable of adapting its approach to the unique demands of each dataset.

6 Conclusion

This thesis presents a step forward in the application of RL to enhance the B&B algorithm for solving Best Subset Variable Selection. Our implementation of RL-based branch rules presents promising potential and highlights the opportunity for future research in optimizing heuristic algorithms using machine learning. Our findings reveal that our RL agent demonstrated remarkable efficacy, outperforming traditional branching rules across synthetic datasets that closely resemble real-world complexities. This was evidenced by faster convergence rates and reduced variance in performance across varied datasets.

However, the study faced limitations regarding computational resources, which confined our exploration to problem instances with a maximum predictor size in the hundreds. Optimizing GPU utilization could mitigate these limitations, allowing for more efficient training of our DQN on larger problem instances. Despite these constraints, we found our RL Mixed Model, trained across a diverse collection of datasets, presented the most generalizable model. This suggests a promising avenue for future research in expanding the datasets' size and complexity used during training to further improve the generalizability and effectiveness of an RL branching rule. Additionally, the scalability of our RL agent to very high-dimensional datasets ($n << p$) was not examined. The assessment of RL-based branching rules to such datasets represents a critical area for future exploration.

Feature importance analysis revealed that the RL Mixed Model built off existing strategies by additionally focusing on refining global bounds and utilizing dataset-specific statistics to guide branching decisions. This indicates a broader, more adaptable strategy that could significantly influence future developments in B&B branching rules. Overall, the successful application of RL for variable selection within the B&B algorithm challenges traditional branching rules and carves a path for further research in the uses of machine learning in optimization and algorithmic design.

References

[1] Tobias Achterberg and Roland Wunderling. Mixed integer programming: Analyzing 12 years of progress. 2013. doi: 10.1007/978-3-642-38189-8_18.

[2] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. *arXiv:1803.10150v2*, 2018.

[3] Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best subset selection via a modern optimization lens. 2016. doi: 10.1214/15-AOS1388.

[4] Santanu S. Dey, Yatharth Dubey, Marco Molinaro, and Prachi Shah. A theoretical and computational analysis of full strong-branching. 2021. doi: arXiv:2110.10754.

[5] Marc Etheve, Zacharie Alès, Côme Bissuel, Olivier Juan, and Safia Kedad-Sidhoum. Reinforcement learning for variable selection in a branch and bound algorithm. 2020. doi: 10.1007/978-3-030-58942-4_12.

[6] Vincent Francois-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. 2018. doi: 10.1561/2200000071.

[7] Trevor Hastie, Robert Tibshirani, and Ryan Tibshirani. Best subset, forward stepwise or lasso? analysis and recommendations based on extensive comparisons. doi: 10.1214/19-STS733.

[8] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* 2009.

[9] Hussein Hazimeh and Rahul Mazumder. Fast best subset selection: Coordinate descent and local combinatorial optimization algorithms. 2021. doi: arXiv:1803.01454.

[10] Hussein Hazimeh, Rahul Mazumder, and Ali Saab. Sparse regression at scale: Branch-and-bound rooted in first-order optimization. 2021. doi: arXiv:2004.06152.

[11] Elias B. Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. *Learning to Branch in Mixed Integer Programming.* 2016.

[12] Ruishan Liu and James Zou. The effects of memory replay in reinforcement learning. 2017. doi: 10.48550/arXiv.1710.06574.

[13] Christopher W. F. Parsonson, Alexandre Laterre, and Thomas D. Barrett. Reinforcement learning for branch-and-bound optimisation using retrospective trajectories. 2022. doi: arXiv:2205.14345.

[14] Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. *arXiv:1906.04859v3*, 2019.

7 Appendix

Table 7: Feature Importance in Mixed Model

Statistic Name	**MSE**
Steps Taken	6.2719
Number of Active Nodes	6.1843
Variable Count	0.0166
Node Level	0.0008
Support Length of the Relaxation Solution	0.0006
Whether a Node Has the Global Lower Bound	0.0005
Length of Candidate Solution Variables	0.0004
z_{ub} Length	0.0002
Whether a Node Has the Global Upper Bound	0.0001
z_{lb} Length	0.0002

*Most important features including those impacted by the dimensionality of a dataset.

Thesis Github Repository: https://github.com/Kdossal/RL-VS