Improving Learning to Branch via Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

1 Introduction

Mixed Integer Programming (MIP) has been applied widely in many real-world problems [1, 2]. Branch and Bound (B&B) is a general and widely used paradigm for solving MIP problems [3]. B&B recursively partitions the solution space into a search tree and compute relaxation bounds along the way to prune subtrees that provably can not contain an optimal solution. This iterative process requires sequential decision makings: *node selection*: selecting the next solution space to evaluate, *variable selection*: selecting the variable by which to partition the solution space [4]. In this work, we focus on learning a variable selection strategy, which is the core of the B&B algorithm [5].

In this work, we argue that strong branching is not a good expert to imitate (More discussion in section B). To obtain effective and non-myopic policies, we use reinforcement learning (RL) and model the variable selection process as a Markov Decision Process (MDP). We design a policy network inspired by primal-dual iteration and employing a novelty search evolutionary strategy (NS-ES) to improve the policy. For efficiency-effectiveness trade-off, the primal-dual policy ignores the redundant information and makes high-quality decisions on the fly. For reducing variance, the ES algorithm is an attractive choice as its gradient estimation is independent of the trajectory length [6]. For exploration, we introduce a new representation of the B&B solving process employed by novelty search [7] to encourage visiting new states.

We evaluate our RL trained agent over a range of problems (namely, set covering, maximum independent set, capacitated facility location). The experiments show that our approach significantly outperforms state-of-the-art human-designed heuristics [4] as well as imitation based learning methods [8, 9]. In the ablation study, we compare our primal-dual policy net with GCN [9], our novelty based ES with vanilla ES [6]. The results confirm that both our policy network and the novelty search evolutionary strategy are indispensable for the success of the RL agent. In summary, our main contributions are the followings:

- We point out the long-term overestimation of strong branching and suggest that methods other than imitating strong branching are needed to find better variable selection policy.
- We model the variable selection process as MDP and design a novel policy net based on primal-dual iteration over reduced LP relaxation.
- We introduce a novel set representation and optimal transport distance for the branching process associated with a policy, based on which we train our RL agent using novelty search evolution strategy and obtain substantial improvements in empirical evaluation.

2 Method

Due to the pruning in B&B, a good variable selection policy can significantly improve solving efficiency. To illustrate how to improve variable selection policy, we organize this section in three parts. First, we present our formulation of the variable selection process as a RL problem. Next, we introduce the primal-dual based policy network. Then, we introduce our branching process representation and the corresponding NS-ES training algorithm.

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

2.1 RL Formulation

Let the B&B algorithm and problem distribution $\mathcal{D}$ be the environment. The sequential decision making of variable selection can be formulated as a Markov decision process. We specify state space $\mathcal{S}$, action space $\mathcal{A}$, transition $\mathcal{P}$ and reward r as follows

- **State Space**. At iteration t, node selection policy will pop out a LP relaxation P_{LP} from the problem set S. We set the representation of the state to $s_t = \{P_{LP}, J, S\}$, where J is the index set of integer variables.
- **Action Space**. At iteration t, the action space is the index set of non-fixed integer variables determined by the relaxation: $\mathcal{A}(s_t) = \{j \in J : \boldsymbol{\ell}_j < \boldsymbol{u}_j\}$.
- **Transition**. Given state s_t and action a_t, the new state is determined by the node selection policy.
- **Reward**. As our target is solving the problem faster, we set the reward $r_t = -1$ with discount $\gamma = 1$. Maximizing the cumulative reward encourages the agent solving problems with less steps.

2.2 Primal Dual Policy Net

Reduced LP. In the solving process, the variable bounds keep changing due to branching. Thus, we obtain our reduced LP relaxation by the following two steps: 1) remove fixed variables x_j, where $\ell_j = u_j$, and plug their values into the constraints; 2) remove trivial constraints, where $\max_{\boldsymbol{\ell} \le \mathbf{x} \le \boldsymbol{u}} \sum_j A_{ij} x_j \le b_i$. In the view of primal-dual iteration, the LP relaxation has Lagrangian form:

$$\min_{\mathbf{x}} \max_{\boldsymbol{\lambda}} \mathbf{c}^T \mathbf{x} + \boldsymbol{\lambda}^T (A\mathbf{x} - \boldsymbol{b}), \quad \text{s.t.} \;\; \boldsymbol{\ell} \le \mathbf{x} \le \boldsymbol{u}, \mathbf{0} \le \boldsymbol{\lambda} \tag{1}$$

where variables and constraints naturally form a bipartite graph. Fixed variables and trivial constraints are nodes having no interaction with other nodes on the graph. A more accurate description of the state can be obtained after removing these redundant nodes.

PD policy net. We parameterize our policy network $\pi_\theta(a_t|s_t)$ as a primal-dual iteration over the reduced LP relaxation by message passing

$$\mathbf{Y}_i \leftarrow \mathbf{f}_{\mathcal{C}}\Big(\mathbf{Y}_i, \sum\nolimits_j \mathbf{A}_{ij}\, \mathbf{m}_{\mathcal{C}}\left(\mathbf{X}_j\right)\Big), \quad \mathbf{X}_j \leftarrow \mathbf{f}_{\mathcal{V}}\Big(\mathbf{X}_j, \sum\nolimits_i \mathbf{A}_{ij}\, \mathbf{m}_{\mathcal{V}}\left(\mathbf{Y}_i\right)\Big) \tag{2}$$

where $f_{\mathcal{C}}, f_{\mathcal{C}}$ are two-layers neural networks, $m_{\mathcal{C}}, m_{\mathcal{V}}$ are one layer neural networks, A_{ij} is the entry in the reduced constraint matrix A and $\mathbf{X}, \mathbf{Y}$ are the embedding for variables and constraints initialized by P_{LP} and J. After two iterations of Equation 2, the variable embedding $\mathbf{X}$ is passed to a two layer neural network score function $f_{\mathcal{S}}$ and the output is the final score for each variable. Since the state reduction and message passing are both inspired by primal-dual iteration, we call it PD policy. A more detailed discussion and comparison with GCN [9] can be found at section C.2.

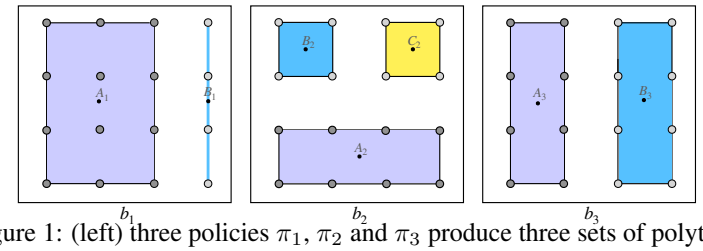
2.3 Set Representation for Policy and Optimal Transport Distance

We train the RL agent using evolution strategy similar to NSR-ES [7] and we need to define the novelty score for B&B process. In the general B&B algorithm, the solving process can be represented by a search tree, where each leaf is a solved subproblem. Given a branch policy π and an instance Q, we define our representation $b(\pi, Q) = \{R_1, \cdots, R_H\}$ as the collection of those leaf subproblems. Focusing on MIP, a subproblem R_i is a LP relaxation which can be represented by its feasible region, a polytope. For example, in Figure 1, b_1, b_2 and b_3 are the set of polytopes produced by three different policies π_1, π_2 and π_3 respectively. And $b_1 = \{A_1, B_1\}$ is a set of two polytopes (leaf subproblems), $b_2 = \{A_2, B_2, C_2\}$ is a set of three polytopes, and $b_3 = \{A_3, B_3\}$ is a set of two polytopes. For computational efficiency, we ignore the constraints and only consider variable bounds such that every polytope is a box. For each polytope R_i (leaf subproblem), we define the weight function $w(\cdot)$ and distance function $d(\cdot, \cdot)$ between two polytopes R_i and R_j as

- $w(R_i) := \#\{x \in R_i : x \text{ is the feasible solution for } Q\}$.
- $d(R_i, R_j) := \|g_i - g_j\|_1$, where g_i and g_j are the center of mass for R_i and R_j respectively.

For example, in Figure 1, we have $w(A_1) = 12, d(A_1, A_2) = \frac{3}{2}$. Then we can map the representation $b = \{R_1, \cdots, R_H\}$ to a simplex $p(b) \in \Delta^{H-1}$ by normalizing the weights $p(R_j) = w(R_j) / \sum_{i=1}^{H} w(R_i)$, and compute a cost matrix $W_{ij} = d(R_i, R_j)$ (See Figure 1 for examples). Then, we can define the metric D between two representations as the Wasserstein distance (or optimal transport distance) [10, 11]:

$$D(b_1, b_2) = \min_{\Gamma} \sum\nolimits_{i,j} \Gamma_{ij} W_{ij}(b_1, b_2), \quad \text{s.t.} \;\; \Gamma \mathbf{1} = p(b_1),\; \Gamma^T \mathbf{1} = p(b_2) \tag{3}$$



$$W(b_1, b_2) = \begin{bmatrix} 3/2 & 3/2 & 5/2 \\ 5/2 & 7/2 & 3/2 \end{bmatrix}$$

$$W(b_1, b_3) = \begin{bmatrix} 1/2 & 3/2 \\ 5/2 & 1/2 \end{bmatrix}$$

$$\Gamma(b_1, b_2) = \begin{bmatrix} 1/2 & 1/4 & 0 \\ 0 & 0 & 1/4 \end{bmatrix}$$

$$\Gamma(b_1, b_3) = \begin{bmatrix} 1/2 & 1/4 \\ 0 & 1/4 \end{bmatrix}$$

Figure 1: (left) three policies π_1, π_2 and π_3 produce three sets of polytopes b_1, b_2 and b_3 respectively for the same problem Q, (right) example cost matrix W and transportation matrix Γ.

For example, in Figure 1, the distance $D(b_1, b_2) = \frac{3}{2}$, $D(b_1, b_3) = \frac{3}{4}$ meaning b_3 is closer to b_1 than b_2. Hence the corresponding policy π_3 is closer to π_1 than π_2.

2.4 Novelty Search Evolutionary Strategy

Equipped with metric D between representations, we can define the novelty score following [7]. Given a policy memory M (a collection of older policies) and an instance Q sampled from the problem distribution $\mathcal{D}$, novelty score is computed as:

$$N(\theta, Q, M) = \frac{1}{k} \sum_{\pi_j \in k\text{NN}(M,\theta)} D(b(\pi_\theta, Q), b(\pi_j, Q)) \tag{4}$$

where $k\text{NN}(M, \theta)$ is the k nearest neighbor of π_θ in M. Back to Algorithm 1, B&B algorithm recursively splits the feasible region and obtains a set of polytopes when finishing solving an instance. Notice that a polytope in the set representation is invariant with the generating order, i.e. branching x_1 then x_2 will give the same polytope with branching x_2 then x_1. As a result, our metric D and novelty score N is mostly determined by the pruning behavior during the solving process. Put everything together, we summarize the training algorithm in section D.

3 Experiments

We now present comparative experiments against two competing machine learning approaches and three SCIP's branching rules to assess the value of our RL agent, as well as an ablation study to validate our choice of policy representation and training algorithm.

3.1 Setup

We evaluate the policies on three data set: set covering, maximum independent set, capacitated facility loacation. We compare against: Reliability Pseudocost Branch (RPB) [4]; Full Strong Branching (FSB) [12]; Vanilla Full Strong Branching (VFS) [12]; Support Vector Machine (SVM) rank [8] and GCN approach [9]. We denote our method as RL, which is the primal-dual net trained by NS-ES. More details can be found in section E.1.

3.2 Decision Quality

We evaluate the variable selection quality by solving 100 test instances under *clean* setting. We say a method wins in this experiment if it results in the least number of solving nodes. As FSB and RPB benefit a lot from branching LP information (section B), we do not include them when counting Wins. Table. 1 shows our RL agent leads the win times on all datasets and the average solving nodes on set covering, and independent set are significantly better than other methods. Our method can also generalize to larger unseen instances. The result can be found in section E.2

3.3 Improvement Analysis

Having seen the improvements brought by RL, we would like to ask what kind of decisions our agent learns. We answer this question in two aspects: finding lower primal bound c^* and obtaining higher dual value $\hat{c}$ that allows pruning in line 5 Algorithm 1. We compare our RL agent with GCN, SVM, VFC on 100 maximum independent set test instances under *clean* setting.

We first examine primal bound c^*. Figure 2 plots the feasible solutions founded during the solving process. A point (n, y) means we find a feasible solution $c^* = y$ in a subproblem containing n branch constraints. Figure 2 shows that our RL agent is able to detect small c^* at the early stage. Hence, it can prune more subproblems and solve the MIP faster.

Table 1: Policy evaluation on test instances. Wins are counted by the number of times a method results in least number of solving nodes

Method	T_{avg}	N_{avg}	Wins	T_{avg}	N_{avg}	Wins	T_{avg}	N_{avg}	Wins
FSB	99.73	146	na/100	19.19	140	na/100	27.16	964	na/100
RPB	12.64	1292	na/100	3.06	250	na/100	21.39	**1449**	na/100
VFS	1935.35	1518	5/ 75	244.14	1304	0/100	173.50	1848	31/100
SVM	21.19	1581	1/100	10.83	498	1/100	29.64	2096	17/100
GCN	10.37	1104	28/100	1.56	418	2/100	26.31	1752	13/100
RL	**7.91**	**820**	**66**/100	**1.26**	**200**	**97**/100	**20.85**	1640	**39**/100
	Set Covering			Independent Set			Facility Location		

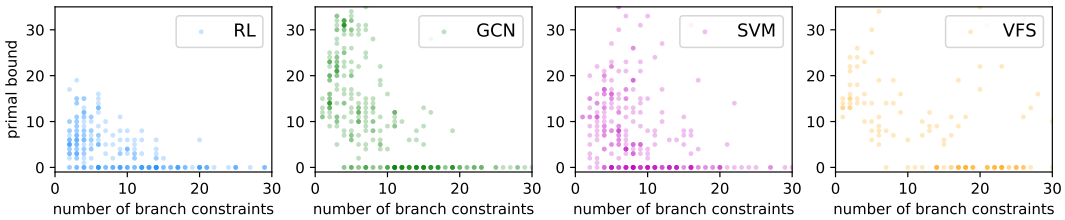


Figure 2: Primal bounds versus the depth in search tree (number of branch constraints) they are found

Then, we check dual value $\hat{c}$. To eliminate the influence in primal bound c^* changing, we initialize $c^* = c_{\text{opt}}$ with the optimal value like [8]. We plot the curve of average width versus the depth in Figure 3. The area under the curve equals the average number of solving nodes, and we report it in the legend. Figure 3 shows that although our RL agent has the worst width in the beginning, it has the lowest peak and leads the overall performance. This means our RL agent successfully employs a non-myopic policy to maximize $\hat{c}$ in the long term.

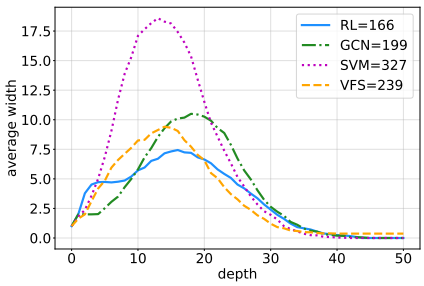


Figure 3: Average width versus depth

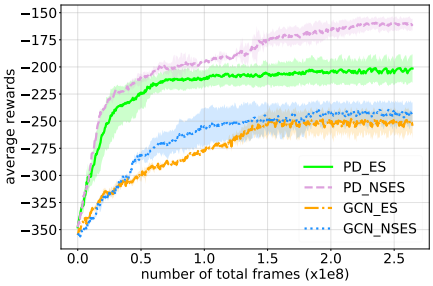


Figure 4: Comparison of 4 RL agents

3.4 Ablation Study

We present an ablation study of our method on maximum independent set problem by comparing four types of RL agents: (1) PD policy + ES; (2) PD policy + NS-ES; (3) GCN + ES; (4) GCN + NS-ES. We sample $V = 200$ instances as our validation set in and plot the average number of solving nodes on the validation set during the training process for five random seeds. The results are plotted in Figure 4. (1) and (2) having larger rewards than (3) and (4) shows that PD policy can obtain more improvement than GCN. Also, (2) and (4) having larger rewards than (1) and (3) shows that novelty search helps to find better policies. The results suggest that both PD policy and NS-ES are indispensable in the success of RL agent.

4 Conclusion

We present an NS-ES framework to automatically learn the variable selection policy for MIP. Central to our approach is the primal-dual policy network and the set representation of the B&B process. We demonstrate our RL agent makes high-quality variable selection across different problems types and sizes. Our results suggest that with carefully designed policy networks and learning algorithms, reinforcement learning has the potential to advance algorithms for solving MIP.

References

[1] Cynthia Barnhart, Amy M Cohn, Ellis L Johnson, Diego Klabjan, George L Nemhauser, and Pamela H Vance. Airline crew scheduling. In *Handbook of transportation science*, pages 517–560. Springer, 2003.

[2] Rafael A Melo and Laurence A Wolsey. Mip formulations and heuristics for two-level production-transportation problems. *Computers & Operations Research*, 39(11):2776–2786, 2012.

[3] Laurence A Wolsey and George L Nemhauser. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons, 1999.

[4] Tobias Achterberg and Timo Berthold. Hybrid branching. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 309–311. Springer, 2009.

[5] Tobias Achterberg and Roland Wunderling. Mixed integer programming: Analyzing 12 years of progress. In *Facets of combinatorial optimization*, pages 449–481. Springer, 2013.

[6] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[7] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In *Advances in neural information processing systems*, pages 5027–5038, 2018.

[8] Elias Boutros Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[9] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 15580–15592, 2019.

[10] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

[11] Gabriel Peyré, Marco Cuturi, et al. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019.

[12] Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, et al. The scip optimization suite 7.0. 2020.

[13] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[14] Ingo Rechenberg. Evolutionsstrategien. In *Simulationsmethoden in der Medizin und Biologie*, pages 83–114. Springer, 1978.

[15] Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. Natural evolution strategies. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 3381–3387. IEEE, 2008.

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

[17] Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017.

[18] Christoph Hansknecht, Imke Joormann, and Sebastian Stiller. Cuts, primal heuristics, and learning to branch for the time-dependent traveling salesman problem. *arXiv preprint arXiv:1805.01415*, 2018.

[19] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, et al. Deep graph library: Towards efficient and scalable deep learning on graphs. *arXiv preprint arXiv:1909.01315*, 2019.

[20] Egon Balas and Andrew Ho. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. In *Combinatorial Optimization*, pages 37–60. Springer, 1980.

[21] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[22] Gérard Cornuéjols, Ranjani Sridharan, and Jean-Michel Thizy. A comparison of heuristics and relaxations for the capacitated plant location problem. *European journal of operational research*, 50(3):280–297, 1991.

A Background

Mixed Integer Programming. MIP is an optimization problem, which is typically formulated as

$$\min_{\mathbf{x}\in\mathbb{R}^n} \ \{\mathbf{c}^T\mathbf{x} : A\mathbf{x} \leq \boldsymbol{b}, \boldsymbol{\ell} \leq \mathbf{x} \leq \boldsymbol{u}, x_j \in \mathbb{Z}, \ \forall j \in J\} \tag{5}$$

where $\mathbf{c} \in \mathbb{R}^n$ is the objective vector, $A \in \mathbb{R}^{m\times n}$ is the constraint coefficient matrix, $\boldsymbol{b} \in \mathbb{R}^m$ is the constraint vector, $\boldsymbol{\ell}, \boldsymbol{u} \in \mathbb{R}^n$ are the variable bounds. The set $J \subseteq \{1, \cdots, n\}$ is an index set for integer variables. We denote the feasible region of x as $\mathcal{X}$.

Linear Programming Relaxation. LP relaxation is an important building block for solving MIP problems, where the integer constraints are removed:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \ \{\mathbf{c}^T\mathbf{x} : A\mathbf{x} \leq \boldsymbol{b}, \boldsymbol{\ell} \leq \mathbf{x} \leq \boldsymbol{u}\}. \tag{6}$$

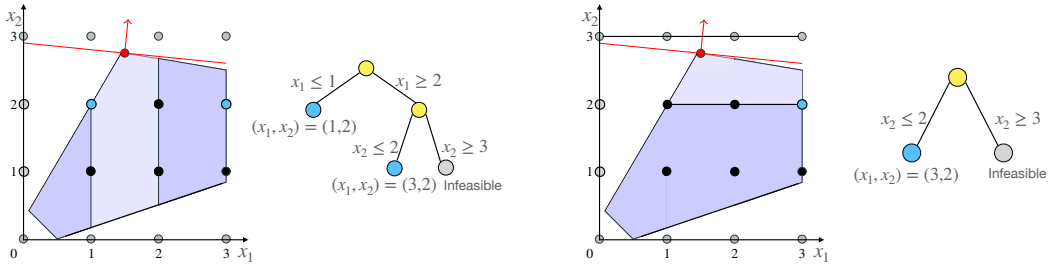
Branch and Bound. LP based B&B is the most successful method in solving MIP. A typical LP based B&B algorithm for solving MIP looks as Algorithm 1 [13]. It consists of two major decisions: *node selection*, in line 2, and *variable selection*, in line 7. In this paper, we will focus on the *variable selection*. Given a LP relaxation and its optimal solution $\hat{x}$, the *variable selection* means selecting an index j. Then, branching splits the current problem into two subproblems, each representing the original LP relaxation with a new constraint $x_j \leq \lfloor \hat{x}_j \rfloor$ for Q_j^- and $x_j \geq \lceil \hat{x}_j \rceil$ for Q_j^+ respectively. This procedure can be visualized by a binary tree, which is commonly called search tree.

Algorithm 1: Branch and Bound

Input: A MIP P in form Equation 5
Output: An optimal solution set x^* and optimal value c^*

1 Initialize the problem set $S := \{P_{LP}\}$. where P_{LP} is in form Equation 6. Set $x^* = \phi, c^* = \infty$;
2 If $S = \phi$, exit by returning x^* and c^* ;
3 Select and pop a LP relaxation $Q \in S$;
4 Solve Q with optimal solution $\hat{x}$ and optimal value $\hat{c}$;
5 If $\hat{c} \geq c^*$, go to 2 ;
6 If $\hat{x} \in \mathcal{X}$, set $x^* = \hat{x}$, $c^* = \hat{c}$, go to 2 ;
7 Select variable j, split Q into two subproblems Q_j^+ and Q_j^-, add them to S and go to 3 ;

Here we gives a simple illustration of B&B algorithm in Figure 5. Given the LP relaxation, the polytope represents the feasible region of the LP relaxation and the red arrow represents the objective vector. We first solve the LP relaxation and obtain the solution $\hat{x}$ as the red point. Noticing it is not feasible for MIP, we branch the LP relaxation into two subproblems. In (a) we select to split variable x_1 and in (b) we select to split variable x_2. The subproblems obtained after branching are displayed by the shaded purple regions. After finishing solve these two MIPs, we obtain the search trees t_1 and t_2. We can see that a wise selection of variable x_2 can solve the problem faster.



(a) split on x_1 and search tree t_1 (b) split on x_2 and search tree t_2

Figure 5: Illustration of splitting in B&B and the corresponding search tree

Evolution Strategy. Evolution Strategies (ES) is a class of black box optimization algorithm [14]. In this work, we refer to the definition in Natural Evolution Strategies (NES) [15]. NES represents the population as a distribution of parameter vectors θ characterized by parameters $\phi : p_\phi(\theta)$. Under a fitness function, $f(\theta)$, NES optimizes ϕ to maximize the average fitness of the population $\mathbb{E}_{\theta\sim p_\phi}[f(\theta)]$. In recent work, [6] outlines a version of NES applied to standard RL benchmark problems. In their work, θ parameterizes the policy π_θ, $p_\phi(\theta)$ is a Gaussian distribution $\mathcal{N}(\theta_t, \sigma^2 I)$

and $f(\theta)$ is defined as the cumulative reward $R(\theta)$ over a full agent interaction. At every iteration, [6] apply n additive Gaussian noises to the current parameter and update the population as

$$\theta_{t+1} = \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} f(\theta_t + \sigma\epsilon_i)\epsilon_i \tag{7}$$

To encourage exploration, [7] propose Novelty Search Evolution Strategy (NS-ES). In NS-ES, the fitness function $f(\theta) = \lambda N(\theta) + (1-\lambda)R(\theta)$ is selected as a combination of domain specific novelty score N and cumulative reward R, where λ is the balancing weight.

B Why Imitating Strong Branching is Not Good

Strong branching is a human-designed heuristic, which solves all possible branch LPs Q_j^+, Q_j^- ahead of branching. As strong branching usually produces the smallest B&B search trees [16], many learning-based variable selection policy are trained by mimicking strong branching [9, 8, 17, 18]. However, we claim that strong branching is not a good expert: the reason strong branching can produce a small search tree is the reduction obtained in solving branch LP, rather than its decision quality. Specifically, (i) Strong branching can check lines 5, 6 in Algorithm 1 before branching. If the pruning condition is satisfied, strong branching does not need to add the subproblem into the problem set S. (ii) Strong branching can strengthen other LP relaxations in the problem set S. For example, if strong branching finds $x_1 \geq 1$ and $x_2 \geq 1$ can be pruned during solving branch LP, then any other LP relaxations containing $x_1 \geq 1$ can be strengthened by adding $x_2 \leq 0$. These two reductions are the direct consequence of solving branch LP, and they can not be learned by a variable selection policy.

To examine the decision quality of strong branching, we employ vanilla full strong branching [12], which takes the same decision as full strong branching, while the side-effect of solving branch LP is switched off. Experiments in Section 3.2 show that vanilla full strong branching has poor decision quality. Hence, imitating strong branching is not a wise choice for learning variable selection policy.

C Implementation

C.1 Hardware

All the experiments were run at a Ubuntu 18.04 machine with Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz, 256 GB Memory and Nvidia RTX 2080Ti graphic cards.

C.2 PD Policy

Comparison. PD policy is similar to the GCN in [9] but has two major differences. First, we use a dynamic reduced graph where fixed variables and trivial constraints are removed due to the variable bounds changing during the solving process while [9] do not consider it. The reduced graph can not only save computation, but also give a more accurate description of the solving state by ignoring the redundant information. The ablation in Section 3.4 shows it is indispensable in the success of RL. Second, we use a simple matrix multiplication in our PD policy while [9] use a complicated edge embedding in GCN. In some sense, GCN can be seen as an overparameterized version of our method. And our success reveals that message passing on the LP relaxation is the true helpful structure.

detail. We implement our primal dual policy net using dgl [19], with hidden dimension $h = 64$ and ReLU activation. The feature $\mathbf{X}$ for variable is a 17 dimension vector and feature $\mathbf{Y}$ for constraint is a 5 dimension vector. We list the detail of feature in Table. 2

C.3 Baseline

FSB. We use the implementation in SCIP [12]

VFS. We use the implementation in SCIP [12]

RPB. We use the implementation in SCIP [12]

GCN. We tried to implement GCN in dgl [19], however, it is significantly slower than the original implementation in [9]. Hence, we still use the implementation in [9].

Tensor	Name	Description
$\mathbf{X}$	type	a one-hot encoding for (binary, integer, implicit, continuous)
	coef	objective coefficient
	lb	variable lower bound
	ub	variable upper bound
	at-lb	indicator whether solution value equals lower bound
	at-ub	indicator whether solution value equals upper bound
	sol-frac	solution value fractionality
	basis-status	a one-hot encoding for simplex basis status (lower, basic, upper, zero)
	red	reduced cost
	age	normalized LP age
	sol-val	solution value
$\mathbf{Y}$	obj-sim	cosine similarity with objective
	bias	bias value
	is-tight	tightness indicator in LP solution
	dualsol-val	dual solution value
	age	normalized LP age

Table 2: Feature $\mathbf{X}$ for variable and feature $\mathbf{Y}$ for constraint

SVM. We use the implementation in [9].

D Training

We have two settings *clean*, *default*. In experiments, we always train and test under the same setting.

Imitation Learning. We initialize our PD policy using imitation learning similar to [9]. The difference is we only use 10000 training samples, 2000 validation samples and 10 training epochs as a warm start. In our setting, a policy from scratch can hardly solve an instance in a reasonable time, hence a warm start is necessary.

Novelty Search Evolution Strategy. We improve our RL agent using Algorithm 2. The parameters are set as $\alpha = 1e-4$, $\sigma = 1e-2$, $n = 40$, $V = 200$, $w = 0.25$, $\beta = 0.99$, $T = 1000$, $k = 10$.

E Experiments

E.1 Setup

Benchmarks: We consider three classes of instances, Set Covering [20], Maximum Independent Set [21] and Capacitated facility location [22], those are not only challenging for state-of-the-art solvers, but also representative for problems encountered in practice. For each class, we set up a backbone based on which we randomly generate the dataset as many real-world problems also share the same backbone. For example, a logistics company frequently solves instances on very similar transportation networks with different customer demands. We generate set covering instances using 1000 columns. We train on instances with 500 rows and evaluate on instances with 500 rows (test), 1000 rows (medium transfer), 1500 rows (hard transfer). We train maximum independent set on graphs with 400 nodes and evaluate on graphs with 400 nodes (test), 1000 nodes (medium transfer), and 1500 nodes (hard transfer). We generate capacitated facility location with 100 facilities. We train on instances with 40 customers (test) and evaluate on instances with 40 customers (test), 200 customers (medium transfer), and 400 customers (hard transfer). More details are provided in the section E.3

Settings: Throughout all experiments, we use SCIP 7.0.1 as the backend solver, with a time limit of 1 hour. For SCIP parameters, we have two settings: *clean*, a pure B&B algorithm with deep-first-search as the node-selection policy and all other functions are disabled so as to eliminate the interference between variable selection and other components of the solver; *default*, the default setting in SCIP where all components are kept to default. It is a state-of-the-art solver (environment) that is able to solve challenging problems.

Algorithm 2: Evolutionary Strategy with Novelty Score.

Input: Learning rate α, Noise std σ, number of workers n, Validation size V, Batch size M, Initial weight λ, Weight decay rate β, Iterations T, Parameter θ_0, Policy memory M, Instance distribution $\mathcal{D}$

, Neighborhood size k. **Output:** Best parameter θ_{best}

Sample validation instances $Q_1, \cdots, Q_V \sim \mathcal{D}$

Set $R_{\text{best}} = \frac{1}{V}\sum_{j=1}^{V} R(\theta_0, Q_j), \theta_{\text{best}} = \theta_0$

for *t=0* **to** *T* **do**

  Sample instances $P_1, \cdots, P_M \sim \mathcal{D}$

  Sample $\epsilon_1, \cdots, \epsilon_n \sim \mathcal{N}(0, I)$ and compute $\theta_t^i = \theta_t + \sigma\epsilon_i$

  Set $M = \{\theta_t^1, \cdots, \theta_t^n\}$

  for *i=1* **to** *n* **do**

    Compute $R_i = \frac{1}{m}\sum_{m=1}^{M} R(\theta_t^i, P_m)$

    Compute $N_i = \frac{1}{m}\sum_{m=1}^{M} N(\theta_t^i, P_m, M)$

  end

  Set $\theta_{t+1} = \theta_t + \alpha\frac{1}{n\sigma}\sum_{i=1}^{n} \lambda \cdot N_i\epsilon_i + (1-\lambda)\cdot R_i\epsilon_i$

  Compute $R^{(t+1)} = \frac{1}{V}\sum_{j=1}^{V} R(\theta_{t+1}, Q_j)$

  if $R^{(t+1)} > R_{best}$ **then**

    Set $R_{\text{best}} = R^{(t+1)}$, $\theta_{\text{best}} = \theta_{t+1}$, $\lambda = \beta * \lambda$

  end

end

Baselines: We compare against: Reliability Pseudocost Branch (RPB) [4], the human-designed state-of-the-art branching rule, which computes strong branching in the beginning and gradually switches to simpler heuristics; Full Strong Branching (FSB), a full version of strong branching; Vanilla Full Strong Branching (VFS), strong branching with branch LP information muted [12]; and two recent machine learning policies support vector machine (SVM) rank approach [8] and GCN approach [9] [1]. We denote our method as RL, which is the primal-dual net trained by NS-ES.

Metrics. To minimize the expected solving cost, metrics are selected as the average solving times (T_{avg}) and average solving nodes (N_{avg}). Since MIP instances could vary a lot in difficulty, we count the number of times each method leads the performance over the number of times each method solves the instance within timelimit (Wins) as a third robust metric.

Implementation. The detail of implementation is provided in section C

E.2 Generalization to Larger Instances

It is very important for RL agents to transfer to larger unseen instances as training on large instances is very expensive in the real world. We investigate the generalization ability of our RL agent by solving 100 transfer instances under *default* setting. To meet the needs in practice, we say a method wins in this experiment if it results in the fastest solving time. As VFS is not able to solve any transfer instance in time limit, we do not list its results in Table. 3. We can see, except for RPB and SVM having comparable performance on hard set covering and hard facility location, respectively, the RL agent leads the performance.

[1] The source code has been released in [9].

Table 3: Policy evaluation on transfer instances. Wins are counted by the number of times a method results in fastest solving time.

Method	T_{avg}	N_{avg}	Wins	T_{avg}	N_{avg}	Wins	T_{avg}	N_{avg}	Wins
					Medium				
FSB	1806	987	0/85	2835	109	0/57	506	208	3/100
RPB	360	16142	6/100	228	**1607**	8/100	311	**562**	4/100
SVM	1003	14734	0/92	1353	6351	8/81	314	1079	17/100
GCN	351	18518	7/100	1654	32951	0/76	332	1231	7/100
RL	**295**	**12134**	**87**/100	**148**	2775	**84**/100	**258**	1032	**69**/100
					Hard				
FSB	3361	902	0/15	3566	45	0/3	1046	107	0/90
RPB	**1608**	87462	30/80	1743	**13139**	21/82	763	**355**	5/99
SVM	2928	77734	0/30	3243	14927	0/20	**606**	702	37/100
GCN	1979	93082	0/71	2973	18206	0/26	891	987	5/97
RL	1628	**60174**	**50**/80	**1497**	13522	**62**/79	624	754	**53**/100
	Set Covering			Independent Set			Facility Location		

E.3 Data Set

Set Covering. We generate a weighted set covering problem following [20]. The problem is formulated as the following ILP.

$$\begin{aligned} \min \sum_{S \in \mathcal{S}} w_S x_S \\ \text{subject to} \sum_{S: e \in S} X_S \geq 1, \ \forall e \in \mathcal{U} \\ x_S \in \{0, 1\}, \ \forall S \in \mathcal{S} \end{aligned}$$

where $\mathcal{U}$ is the universe of elements, $\mathcal{S}$ is the universe of the sets, w is a weight vector. For any $e \in \mathcal{U}$ and $S \in \mathcal{S}$, $e \in S$ with probability 0.05. And we guarantee that for any e, it is contained by at least two sets in $\mathcal{S}$. Each w_S is uniformly sampled from integer from 1 to 100.

We first generate a set covering problem with $\mathcal{U}_0 = \{e_1, \cdots, e_{400}\}$ and $\mathcal{S}_0 = \{S_1, \cdots, S_{1000}\}$ and set it as our backbone. Then, every time we want to generate a new problem with m elements, we let $\mathcal{U} = \mathcal{U}_0 \cup \{e_{401}, e_{402}, \cdots, e_m\}$ add new e_i into $S \in \mathcal{S}$ following the pipeline mentioned above.

Maximum Independent Set. We generate maximum independent set problem using Barabasi-Albert [21] graphs. The problem is formulated as the following ILP.

$$\begin{aligned} \max \sum_{v \in V} x_v \\ \text{subject to} \quad x_u + x_v \leq 1, \ \forall e_{uv} \in E \\ x_v \in \{0, 1\}, \ \forall v \in V \end{aligned}$$

where V is the set of vertices and E is the set of edges. We generate the BA graph using a preferential attachment with affinity coefficient 4.

We first generate a BA graph G_0 with 350 nodes. Then, every time we want to generate a new problem with n variables, we expand G_0 using preferential attachment.

Capacitated Facility Location. We generate the capacitated facility location problem following [22]. The problem with m customers and n facilities is formulated as the following MIP.

$$
\begin{aligned}
\min \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} d_j y_{ij} + \sum_{i=1}^{n} f_i x_i \\
\text{subject to} \sum_{i=1}^{n} y_{ij} = 1,\ \forall j = 1, \cdots, m \\
\sum_{j=1}^{m} d_j y_{ij} \leq u_i x_i,\ \forall i = 1 \cdots, n \\
y_{ij} \geq 0,\ \forall i = 1, \cdots, n \text{ and } j = 1, \cdots, m \\
x_i \in \{0, 1\},\ \forall i = 1, \cdots, n
\end{aligned}
$$

where $x_i = 1$ indicates facility i is open, and $x_i = 0$ otherwise; f_i is the fixed cost if facility i is open; d_j is the demand for customer j; c_{ij} is the transportation cost between facility j and customer i; y_{ij} is the fraction of the demand of customer j filled by facility i. Following [22], where we first sample the location of facility and customers on a 2 dimension map. Then $c_i j$ is determined by the Euclidean distance between facility i and customer j and other parameters are sampled from the distribtuion given in [22].

We first generate the location of 100 facilities and 40 customers as our backbone. Then, every time we want to generate a new problem with m customers, we generate new $m - 40$ locations for customers and follow the pipeline mentioned above.

F Discussion

An interesting phenomenon is that GCN can easily beat VFS after imitation learning (Or our PD policy can obtain similar result). One possible explanation is that the primal-dual message passing is a principle structure that naturally learns the good decisions and ignores the noise brought by strong branching. Another possible reason is the biased sampling. To keep the diversity of the sample, [9] employs a mixed policy of RPB and VFS to sample the training data. It is possible that VFS is a powerful expert on the state distribution determined by the mixed policy. More studies are needed before we can answer this question.

Another point is our set representation is compatible with general B&B algorithm. Once the weight function w and distance function d are defined, we can compute the distance between two B&B solving processes.