Article

Semi-supervised Learning Approach to Efficient Cut Selection in the Branch-and-Cut Framework

Jia He Sun [1]*, and Salimur Choudhury [2]

[1] Lakehead University, Department of Computer Science; jsun16@lakeheadu.ca
[2] Queen's University, School of Computing; s.choudhury1@queensu.ca
* Correspondence: jsun16@lakeheadu.ca

Abstract: Mixed integer programming (MIP) is an extremely versatile subclass of mathematical optimization problems. Applications of MIP are ubiquitous in our world today, ranging from scheduling to network design to production planning. Owing to its integrality constraints, MIP problems can be extremely difficult to solve efficiently, especially at large scales. The standard approach in state-of-the-art commercial solvers is called branch-and-cut. The branch-and-cut framework recursively reduces the solution solution space by splitting the original MIP problem into subproblems (branching). At each of these subproblems, cutting planes are added to further reduce the solution space (cutting). The selection of these cuts is an integral part of the branch-and-cut process as high quality cuts can greatly increase solving efficiency. Currently, cut selection is decided by heuristics that both require expert knowledge and lack generalizability. In this paper, we propose an efficient and highly generalizable cut selection scheme based on semi-supervised learning. First, we design a cut evaluation metric that labels cuts based on whether they are efficient or not. Then, we train a deep learning classification model with unsupervised pre-training as a ranking function for cuts. In our evaluation, the proposed model outperforms standard heuristics and is comparable to existing machine learning approaches. Furthermore, the model is shown to be generalizable over both problem size and problem class.

Keywords: machine learning; semi-supervised learning; mixed integer programming; cutting planes

Citation: Sun, J.H.; Choudhury, S. Semi-supervised Learning Approach to Efficient Cut Selection in the Branch-and-Cut Framework. *Journal Not Specified* **2022**, *1*, 0. https://doi.org/

Received:
Accepted:
Published:

1. Introduction

MIP problems are linear programming (LP) problems with integrality constraints. That is, some or all of the solution variables must take integer values. This particular subclass of optimization problems can be applied to a plethora of industry applications including but not limited to: scheduling [1], network design [2], and production planning [3]. However, due to the non-convexity of its feasible region, a characteristic enforced by its integrality constraints, MIP problems are extremely difficult to solve efficiently.

Modern commercial MIP solvers take the branch-and-cut approach which is a combination of the branch-and-bound technique and the cutting planes technique [4]. The branch-and-bound technique recursively separates the solution space into smaller subspaces (branches) while keeping track on the best solution found so far to eliminate future branches (bounds). The cutting planes technique aims to reduce the size of a solution space by adding linear inequalities (cuts) as additional constraints. The branch-and-cut method applies the cutting planes technique for each branch in the branch-and-bound process. However, the selection of solution variable for the branching and the selection of cuts are key decisions with huge impact on the overall efficiency of the branch-and-cut algorithm [4]. Currently, problem specific heuristics are used to make these decisions. These heuristics are often manually designed and lack the generalizability to be deployed on a large class of problems.

To combat the aforementioned issues, machine learning (ML) techniques have been implemented to produce efficient and generalizable MIP solving techniques. Having an effective cut selection algorithm is imperative to an efficient MIP solver as high quality cuts can significantly reduce the feasible set which leads to a reduction in the number of nodes in the branch-and-bound search tree. With this motivation, we propose a generalizable and efficient cut selection scheme for the branch-and-cut framework. This selection scheme includes a cut classification system that differentiates efficient cuts from inefficient cuts and a semi-supervised machine learning model that learns to do the same.

The key contributions of this paper can be summarized as follows:

- we propose a novel cut classification scheme using a multiple instance learning (MIL) approach
- we implement a supervised classification deep learning model augmented by unsupervised pre-training
- we evaluate the generalizability of our model in terms of both problem size and problem class against existing heuristics and proposed ML models

2. Related Works

There are two main approaches of applying machine learning techniques to solving optimization problems. The first approach aims to design a pure machine learning model that solve an optimization problem in a black box style [5]. These have been shown to be effective empirically but they lack theoretical guarantees.

The second approach, the approach that this paper has taken, aims to augment existing algorithms with the integration of machine learning techniques [5]. This approach typically maintains the theoretical guarantees of the existing algorithm while improving some aspects of the algorithm that may be heuristic-based.

In particular to the branch-and-cut algorithm employed in MIP solvers, the branching process and the cutting process are two key areas where machine learning techniques can be implemented to improve upon human designed heuristics for higher efficiency and higher generalizability. Recently, there has been a surge in interest in augmenting the branch-and-cut framework with machine learning techniques, however it has been mostly focused on the branching process. He et al. designed an imitation learning model which can learn an adaptive node searching strategy in the branch-and-bound process that performs better than modern commercial solvers on the MIP problem class [6]. Khalil et al. proposed a model that learns to mimic the strong branching strategy, a time consuming process that significantly reduces the size of the branch-and-bound search tree [7]. Khalil et al. proposed further improvements to the branch-and-bound process by designing a machine learning model that selects which node in the search tree to make progress on by predicting whether or not a heuristic will succeed at a given node [8]. These works and many more are detailed in Huang et al.'s survey for this research cluster [9].

The cut selection process has seen less focus from researchers aiming to integrate machine learning into the branch-and-cut framework. Tang et al. proposed a deep reinforcement learning (RL) formulation for intelligent adaptive cut selection for the cutting planes method, a MIP solving scheme that relies purely on cuts [10]. However, this work focuses purely on only one type of cuts (Gomory) and aims to reduce the total number of cuts added. Paulus et al. proposed an imitation based learning model called "NeuralCut" based on a lookahead expert that aims to close the integrality gap as much as possible at the root node of a MIP [11]. While these two works are in the same domain as our work, they have different in terms of target evaluation. The model proposed in this paper aims to improve the efficiency of the branch-and-cut framework as whole and is evaluated as such via run time which is different than the two aforementioned papers.

Huang et al. designed a multiple instance supervised machine learning model for cut selection in the branch-and-cut framework called "Cut Ranking" which includes a cut labelling system as well as a trained scoring function [12]. This model has been deployed in an industrial setting and has outperformed the existing commercial solver by an average speedup ratio of 12.42%. Huang et al.'s work is most similar to the work proposed in this paper as "Cut Ranking" also aims to reduce the overall efficiency of the branch-and-cut process. However, not only do we propose a different labelling system for generating labelled cut data, we also take a semi-supervised approach to the machine learning model as opposed to "Cut Ranking"'s completely supervised approach.

Semi-supervised learning is a machine learning technique that utilizes both unlabelled and labelled data. It is especially useful for scenarios where data labelling is an expensive or difficult task, such as the cut labelling process in this paper [13]. The semi-supervised approach taken in this paper, unsupervised pre-training (such as auto-encoders), has been successfully implemented into deep neural networks since 2007 [14,15]. Details regarding semi-supervised learning can be found in Erhan et al.'s survey [13].

3. Cut Classification

For every MIP problem, and for each node of the branch-and-bound search tree, existing MIP solvers can generate a set of candidate cuts. The goal of our model is to select the most efficient cuts from this candidate set. In our work, the approach taken is MIL.

3.1. Multiple Instance Learning

The proposed cut classification system is based on MIL where the training data is generated based on bags of instances. This approach is chosen because individual cuts will have little measurable effect on the overall efficiency of the branch-and-cut framework, thus, cuts are grouped into bags and are evaluated at the bag level. Then, labels are assigned at the bag level. "Cut Ranking" takes the same approach for data generation [12].

Consider a MIP problem P of the form:

$$max\{c^T x : Ax \leq b, x_j \in \mathbb{Z}, \forall j \in N_I\} \tag{1}$$

where $c, x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $N_I \subseteq N = \{1, ..., n\}$. Let x_{LP} be an optimal solution to P's corresponding LP relaxation and let C be the candidate cut set generated by a solver. For each cut $c_i \in C$, it is of the form:

$$\alpha_i^T x \leq \beta_i \tag{2}$$

Let $f_{c_i} \in \mathbb{R}^l$ denote the feature vector of c_i. Let $B = \{B_1, ..., B_k\} \subseteq C$ be all bags of cuts sampled from C. Then, the feature vector of a bag B_u, denoted by f_{B_u}, is the average of the feature vectors f_{c_i} for all $c_i \in B_u$. That is, the feature vector of a bag of cuts is the average of the feature vectors of the cuts in the bag. Furthermore, $|B_u| \geq 0.1 \cdot |C|, \forall j \in \{1, ..., k\}$. In other words, the size of each sampled bag of cuts must be at least 10% of the size of the candidate cut set. This is to ensure that we do not have samples with not enough cuts to make a measurable difference in run time.

For each cut c_i, the features extracted are as follows:

1. cut coefficients features (4): maximum, minimum, mean, and standard deviation of cut coefficients α_i
2. objective function coefficients features (4): maximum, minimum, mean, and standard deviation of objective function coefficients that correspond to the non-zero cut coefficients
3. support: proportion of variables with non-zero cut coefficients to all variables
4. integral support: proportion of integer variables with non-zero cut coefficients to all variables with non-zero cut coefficients
5. relative violation: violation of the cut against an optimal solution of the LP relaxation normalized against the right hand side (if the right hand side is 0, then it is not normalized):

$$\frac{\alpha^T x_{LP} - \beta}{|\beta|} \tag{3}$$

6. distance: euclidean distance between an optimal solution of the LP relaxation and the hyperplane imposed by the cut:
$$\frac{\alpha^T x_{LP} - \beta}{||\alpha||} \tag{4}$$
7. objective function parallelism: measure of linear dependence between the cutting plane and the objective function:
$$\frac{\alpha^T c}{||\alpha|| ||c||} \tag{5}$$
8. expected improvement: approximation of the improvement of the optimal objective value of the LP relaxation after adding the cut:
$$\frac{\alpha^T x_{LP} - \beta}{||\alpha||} \cdot \frac{\alpha^T c}{||\alpha||} \tag{6}$$

The first two features are basic structural data of the cut. The next two features are support features of the cut with regard to the integrality constraints of the problem. The rest of the features are the main metrics put forward by Wesselmann and Suhl that aim to measure the quality of cuts [16].

3.2. Cut Evaluation and Data Labelling

For each MIP problem P, after we have sampled k bags from the generated candidate cut set C, every sampled bag B_u is evaluated by adding all cuts in B_u to P and running the solver. To evaluate the performance of each bag, the metric used in our scheme is normalized run time. Run time is chosen over other metrics such as number of cuts added and number of nodes visited because our main goal is to improve the overall efficiency of the cut-and-branch framework currently used in MIP solvers and run time is the most accurate reflection of this. The run time recorded for each bag of cuts is normalized as some MIP problems will naturally take longer to run than others due to problem size.

Let r_j be the run time of problem P with appended bag B_j for all $j \in \{1, ..., k\}$. Without loss of generality, assume B_v to be the bag with shortest run time and B_w be the bag with the longest run time. Then, the evaluation value assigned to each sampled bag B_u, normalized run time, is defined by:
$$r_j^* = 1 - \frac{r_j - r_m}{r_n - r_m} \tag{7}$$

In this format, for each MIP problem, the best performing bag will always be evaluated as 1 and the worst performing bag will always be evaluated as 0. The rest of the bags will have an evaluation of some value in $[0, 1]$.

After each bag of cuts has been evaluated, it will be given a discrete label. In our scheme, we will assign 1 to bags with normalized run time over λ_1 and assign 0 to bags with normalized run time under λ_2. λ_1 and λ_2 are both hyperparameters between 0 and 1 and $\lambda_1 > \lambda_2$. All other bags will not be labelled and consequently will not be used in the supervised training portion of the model.

This labelling system is chosen because it is consistent over all possible distributions of sample performances. Consider a naive labelling system where samples are labelled 1 if they are in the top 50 percentile and 0 otherwise. And consider the proposed labelling system where samples are labelled 1 if their normalized run time is over 0.5 and 0 otherwise (0.5 is an arbitrarily chosen threshold for this example). Consider Fig. 1 and Fig. 2 which both display the sample performance distribution of the same two MIP problems taken from our data set. We can see that, in Fig. 1, the naive labelling scheme is very inconsistent when there are samples whose distributions are skewed towards either end of the scale. Meanwhile, our proposed labelling scheme, in Fig. 2, remains consistent over all types of distributions while still allowing us to control the number of positive samples by tuning the hyperparameters.

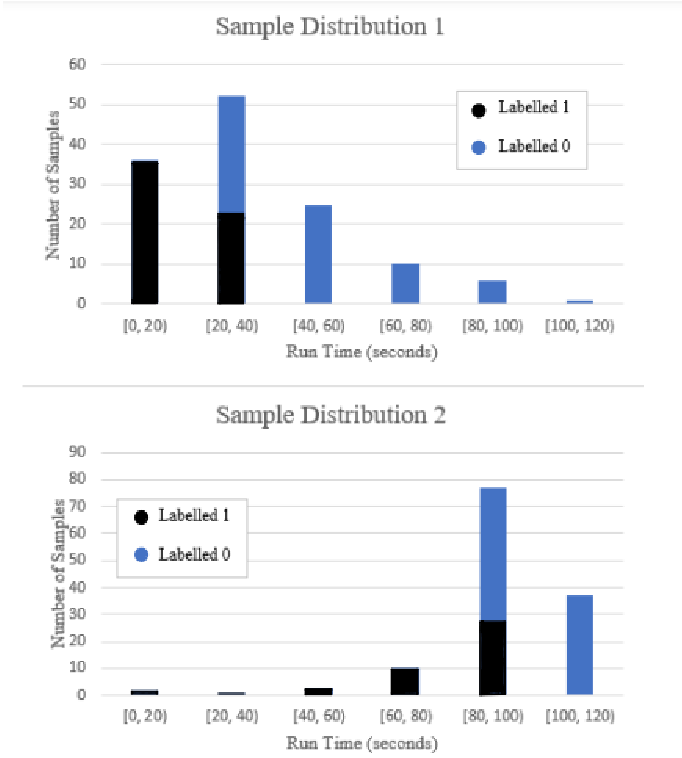

Figure 1. Naive labelling example

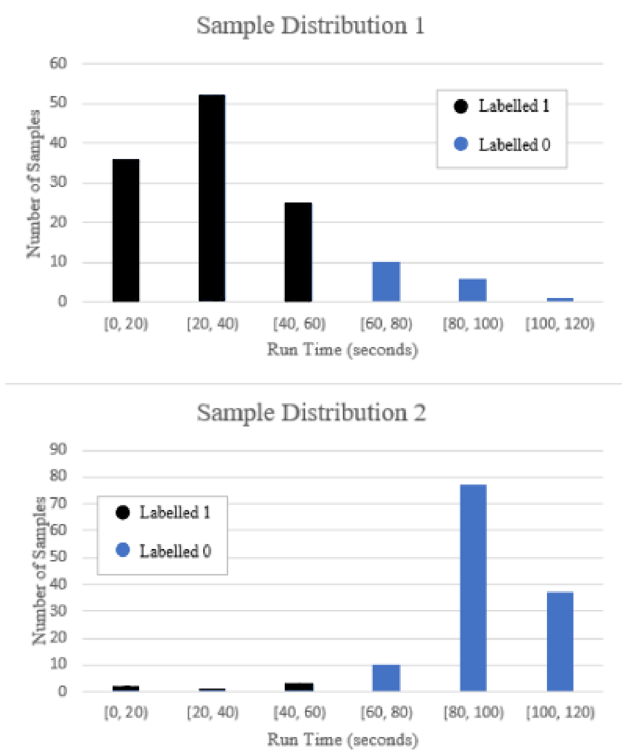

Figure 2. Proposed labelling example

We choose allow some data points to remain unlabelled because it may be difficult for the machine learning model to learn from samples with very similar performances but are labelled differently. Again, consider the naive labelling system in Fig. 1, there are samples with very similar run times but some are labelled 1 and some are labelled 0. The machine learning model may then learn to differentiate between these two samples when there may not be any meaningful difference between them, they just happened to be on the threshold determined by the labelling system. Intuitively, we are labelling the samples we know are good as 1 and the samples we know are bad as 0. The samples in the middle that could be good or could be bad are not labelled. During our hyperparameter tuning phase, we experimented with labelling all samples and found that it performs worse than the proposed method.

4. Semi-supervised Learning Model

The proposed machine learning model is a semi-supervised deep learning model for tabular data. The reason that unsupervised pre-training is employed in this model is mainly due to the nature of the cut classification scheme described in the previous section. Our proposed classification scheme only labels a portion of all generated data points (the proportion of labelled to unlabelled depends on hyperparameters). Thus, we have an abundance of unlabelled data at our disposal. Furthermore, the data generation process in the MIP setting, while offline, is quite time extensive. Therefore, unsupervised pre-training is employed to not only make use of all generated data, but also to offset the consequences of having time extensive labelled data generation. The success of unsupervised pre-training is well documented, especially in the case of auto-encoders, the most popular type of unsupervised pre-training [13]. However, auto-encoders are more suited to settings such as computer vision and voice recognition as opposed to our structured tabular data. With that in mind, we implement the pre-training model used in TabNet, a deep tabular data learning model [17]. TabNet's pre-training model, similar to a denoising auto-encoder, is designed to predict missing feature values from corrupted feature input based on observed interdependencies.The model includes feature transformers and fully connected layers at each decision step with the output as the reconstructed features. Fig. 3 is a visual representation of how the unlabelled data is used in the pre-training phase. More details on the unsupervised pre-training model can be found in Arık and Pfister's paper [17].

cutcoeffmean	cutcoeffmax	cutcoeffmin	cutcoeffstdev	objcoeffmean	objcoeffmax	objcoeffmin	objcoeffstdev
-0.951506737	?	-5.634705266	1.057157164	0.218474975	84.42622951	0	?
0.067979603	8.080977324	-4.693085502	?	0	?	0	0
-0.136057101	1.375	-1.291666667	0.301423796	2276.834972	5005.982414	428.1134468	1363.277017
?	2.166666667	-1.75	?	2229.410388	7415.333333	419.3333333	?
?	7.178208738	-8.389608017	0.770533116	0	0	0	0
-0.136038114	7.039868599	?	0.676022566	0	0	0	0
?	1	-2269341.523	75813.37949	28.125	?	0	79.54951288
-2532.842225	1	-2269433.634	?	28.125	225	0	79.54951288

cutcoeffmean	cutcoeffmax	cutcoeffmin	cutcoeffstdev	objcoeffmean	objcoeffmax	objcoeffmin	objcoeffstdev
	2.7640838						**2.72384259**
			0.518479531		**0**		
-0.00576976			**0.543557416**				**673.771471**
-0.15555153							
		-7.6023609					
-2532.73942					**225**		
			75816.45668				

Figure 3. Unsupervised pre-training example

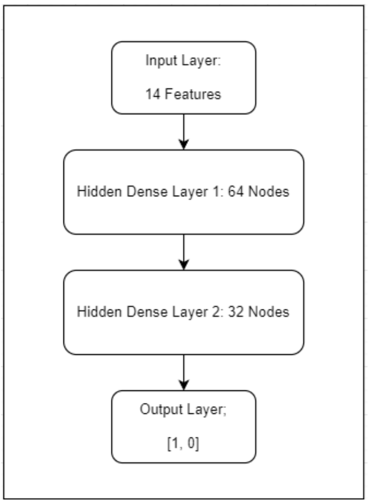

Figure 4. Diagram of Model Architecture

4.1. Model Architecture

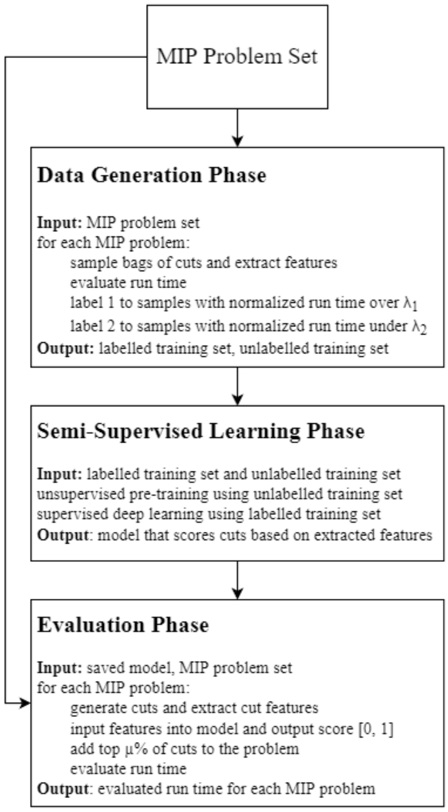

Figure 5. High-level diagram of the proposed cut selection scheme

Following pre-training, the data will be pushed through a supervised classification model. The model consists of 4 fully connected layers with an input layer, an output layer, and 2 hidden dense layers of size 64 and 32 respectively. Since the proposed model is a binary classification model we chose to use binary cross entropy as our loss function:

$$-\frac{1}{N}\sum_{i=1}^{N} y\log(p) + (1-y)\log(1-p) \tag{8}$$

where N is output size, y is target value, and p is model output. For the same reasoning, we also choose to use sigmoid as our activation function:

$$\sigma(z) = \frac{1}{1 + e^{-x}} \tag{9}$$

A regression model was not chosen because, in our internal experiments, the binary classification model consistently outperformed it. We hypothesize that this is due to the relatively small amount of labelled data and the existence of many outliers. A multi-class classification model was also tested with little success. This may be due to the classification thresholds are abstract thresholds enforced by hyperparameters as opposed to actual existing structural differences. For each cut, the output of model will be a continuous value between $[0, 1]$ and the top τ% of cuts will be added to the model, τ is the cut selection threshold hyperparameter. A visual of our implemented architecture is given in Fig. 4.

To summarize the entire cut selection scheme proposed in our work, we provide both a high-level diagram in Fig. 5 and a detailed algorithm description in Algorithm 1.

Algorithm 1 Proposed Cut Selection Algorithm

Data Generation Phase
Input: MIP problem set
Output: labelled training set, unlabelled training set

for each MIP problem **do**
  Do some action.
  generate set of candidate cuts
  sample bags of cuts from candidate cut set
  for each bag of cuts **do**
    construct features
    evaluate run time
  end for
  normalize run time across all bags of cuts
  label 1 to samples with normalized run time $\geq \lambda_1$
  label 2 to samples with normalized run time $< \lambda_2$
  other samples remain unlabelled
  put labelled samples into labelled training set
  put unlabelled samples into unlabelled training set
end for
return labelled training set, unlabelled training set

Semi-supervised Learning Phase
Input: labelled training set, unlabelled training set
Output: model that scores cuts based on extracted features

initialize autoencoder
for epoch in pre-training epochs **do**
  train autoencoder using unlabelled training set
end for
save weights
initialize deep classification model
load weights
for epoch in training epochs **do**
  train model using unlabelled training set
  loss function: binary cross entropy
end for
return model

Evaluation Phase
Input: saved model, MIP problem set
Output: run time (of each MIP problem)

for each MIP problem **do**
  generate set of candidate cuts
  for cut in candidate cut set **do**
    construct features
    input features into saved model and receive score in [0, 1]
  end for
  add top μ% of cuts to MIP problem
  evaluate run time
end for
return run times for each MIP problem

5. Experimentation

In our experimentation, the cut selection scheme is implemented only at the root node of the search tree. In other words, cuts are only being added to the original MIP problem. Since each node of the branching search tree can be considered its own MIP problem, we believe that the results of our experimentation extends to the all nodes of the branching search tree.

5.1. Data Sets

To train our model, we procured a data set consisting of 80 real-world set partitioning problems from the Mixed Integer Programming Library (MIPLIB2017) [18]. Set partitioning is chosen as it is one of the most widely applied mathematical optimization problems, especially in the fields of transportation systems, communication systems, scheduling, resource allocation, and industrial planning systems [19]. The set partitioning problem can be stated as follows: for a given finite set G and a set P of n subsets X_j associated with costs C_j, find a partition of G with minimum cost [19]. That is, a cost minimizing subset of P where all elements are disjoint of each other and the union of the elements is G. The problems in our chosen problem set are all similar in terms of difficulty as they all take less than 30 minutes to solve. For each problem, 135 samples were extracted from the candidate cut set which included the following types of cuts: probing, Gomory, Gomory mixed integer, reduce and split, flow cover, mixed integer rounding, two-step mixed integer rounding, lift and project, residual capacity, zero half, clique, odd wheel, and knapsack cover [20]. After deleting duplicate cut samples, the total number of data points generated for training is 10,602.

For comparison, we implement the follow evaluation baselines:

1. **random**: cuts are added randomly
2. **relative violation**: cuts with the highest violation relative to its right hand side are added
3. **objective function parallelism**: cuts that are the closest to being parallel with the objective function are added
4. **distance**: cuts that have the highest euclidean distance between an optimal solution of the LP relaxation and the hyperplane imposed by the cut are added
5. **expected improvement**: cuts that have the highest approximation of objective improvement
6. **"Cut Ranking"** [12]
7. proposed model but **without unsupervised pre-training**

Baselines 1-5 are common heuristics for cut selection [16]. Baseline 6 is the model proposed by Huang et al. that also focuses on run time [12]. Baseline 7 is to confirm the effects of using unlabelled data. The work of Tang et al. and Paulus et al. are not included since they are designed based on other metrics. Tang et al.'s work aims to minimize the number of cuts in the cutting planes method while Paulus et al's work aims to maximize the integrality gap closed per cut added at the root node [10,11].

For evaluation, we perform experiments on the following real-world data sets:

1. 50 small set partitioning problems (different problems than the ones used in training) [18]
2. 50 large set partitioning problems [18]
3. 50 mixed integer knapsack problems [21]
4. 50 lot sizing problems [22]
5. 50 general MIP problems [18]

Data sets 1-2 are used to evaluate the performance of the model on similar problems that it was trained on as well as how well it generalizes in terms of problem size. Data sets 3-5 are used to evaluate how well the model generalizes to different types of MIP problems.

Problem Set	Proposed Model	Cut Ranking	Relative Violation	Distance	Parallelism	Random	Expected Improvement
Set Partitioning (small)	**0.764**	0.515	0.476	0.372	0.351	0.321	0.395
Set Partitioning (large)	**0.749**	0.621	0.579	0.475	0.397	0.216	0.227
Mixed Integer Knapsack	0.991	**0.998**	0.974	0.762	0.764	0.762	0
Lot Sizing	**0.795**	0.698	0.667	0.197	0.740	0.773	0.205
General MIP	0.695	**0.729**	0.562	0.424	0.392	0.163	0.199

Table 1. Average normalized run time evaluation between proposed model, "Cut Ranking", and various heuristics (higher is better)

5.2. Hyperparameters

After tuning, the hyperparameters for data generation are $\lambda_1 = 0.7$ and $\lambda_2 = 0.45$. That is, cut samples with normalized run time over 0.7 are labelled 1 and cut samples with normalized run time under 0.45 are labelled 0. With these hyperparameters, the labelled samples total to 5,020 and the unlabelled samples total to 5,582. The cut selection threshold hyperparameter τ is set to be 0.7, that is, the top 30% of cuts are added to the model. This is consistent with all the evaluation baselines to ensure fair evaluation.

For model specific hyperparameters, dropout is set to be 0.01, learning rate is set to be 0.0001, batch size is set to be 32, unsupervised pre-training is ran for 512 epochs, and supervised training is also ran for 512 epochs.

5.3. Implementation

The MIP solver used is the Coin-or Cut-and-Branch Solver [20]. The model is implemented in python using the Tensorflow library [23]. The hardware specifications used are an Intel(R) Core i5-9400 CPU and a NVIDIA GeForce GTX 1650.

5.4. Results

Similar to the data generation phase, the evaluation metric we use to is normalized run time. To reiterate the intuition of this metric, for each MIP instance, the best performing algorithm (the algorithm with the lowest run time) will be evaluated as 1 and the worst performing algorithm (the algorithm with the highest run time) will be evaluated as 0. The rest of the algorithms will have an evaluation of some value in $[0, 1]$ depending on where they lie on the distribution of run times. Eq. 7 is the exact formula for normalized run time. Then, in each of MIP data sets used in evaluation, the normalized run time of each MIP instance is averaged for all the algorithms.

Table 1 displays the evaluation results of our proposed model compared against the selected baselines. First and foremost, the proposed model significantly outperforms all evaluated baselines on the data set, set partitioning (small), achieving an average normalized run time of almost 50% higher than the next highest baseline. This is to be expected as this is the data set that our model was trained on.

As for the other data sets, our proposed model is at worst comparable to both "Cut Ranking", the ML model, and relative violation, the best performing heuristic. For the set partitioning (large) and the lot sizing data sets, our model outperforms all of the baselines by a comfortable margin. For mixed integer knapsack problems, the proposed model performs slightly worse than "Cut Ranking" and slightly better than relative violation, but can be considered comparable. For the general MIP data set, our model is less efficient than "Cut Ranking" by a slight margin but outperforms the other baselines.

From the results detailed in Table 1, we can conclude that, on the data set it was trained on, the proposed model performs better than existing heuristics and similar ML models.

Problem Set	With Pre-Training	Without Pre-Training
Set Partitioning (small)	**0.764**	0.676
Set Partitioning (large)	**0.749**	0.612
Mixed Integer Knapsack	**0.991**	0.975
Lot Sizing	**0.795**	0.596
General MIP	**0.695**	0.612

Table 2. Average normalized run time evaluation between proposed model and proposed model without unsupervised pre-training (higher is better)

Furthermore, it generalizes well in terms of both problem class and problem size as it is, at worst, competitive with the evaluated baselines on the other data sets.

To confirm that the unsupervised pre-training portion of our model is indeed improving the performance, we also evaluated our proposed model without pre-training. These evaluation results are shown in Table 2. We can see that, other than the mixed integer knapsack data set, our proposed model consistently performed its no pre-training counterpart by a sizable margin. Even in the mixed integer knapsack data set, the proposed model still performed better, though only by a slight margin. On average, the model with pre-training performs around 13% higher in terms of normalized run time. Therefore, we can conclude that the unsupervised pre-training portion of our proposed model is indeed beneficial to the overall performance.

6. Conclusion

The backbone of modern state-of-the-art MIP solvers is the branch-and-cut framework. The selection of cutting planes to be implemented at each node of the branching search tree is an important task and, to tackle this, we proposed a semi-supervised deep learning based cut selection scheme. In this paper, we defined a novel MIL cut classification scheme that evaluates cuts based on normalized run time. Furthermore, due to the difficult and expensive cut labelling process, we propose a semi-supervised deep learning model that can train on both unlabelled data as well as labelled data. An unsupervised pre-training model is trained to reconstruct features based on inter-feature dependencies using unlabelled data. Then, the labelled data are trained upon using a standard binary classification approach. Overall, we designed a machine learning model that can be used to evaluate and rank cuts in a branch-and-cut framework. From our experiments on real-world MIP problem sets, we found that our model outperforms existing frameworks and is comparable to other proposed machine learning based approaches. Furthermore, after testing on five different types of MIP problems, we found that our model is generalizable over both problem size and problem class. Lastly, we confirmed that the unsupervised pre-training portion of our proposed model is indeed a beneficial inclusion.

Due to the heuristic nature of the cut selection problem, machine learning appears to be a suitable approach. However, in this problem, generating accurate evaluations of cut quality can be a difficult and expensive task. In this paper, we attempted to bypass that using semi-supervised learning. However, there are certainly other approaches to this problem such as imitation learning and transfer learning. Furthermore, when the entire branch-and-cut framework is considered as a whole, not only is the evaluation of cut quality a problem, determining the quantity of cuts to be added is also an interesting problem. Currently, both the branching process and the cutting process have received attention from the machine learning community. However, they are often considered completely separately. It can be interesting and fruitful to study the dependencies between variable/node selection in the branching process and cut selection in the cutting process.

Funding: This research received no external funding.

Data Availability Statement: The MIP datasets used in training data generation and evaluation can be found at the following links:

- Mixed Integer Programming Library (MIPLIB2017). [18]
- A. Atamtürk's personal website. [21] [22]

The training data used in this study are openly available in Mendeley Data at 10.17632/thtz8h894m.1 and also available for download here.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

MDPI	Multidisciplinary Digital Publishing Institute
MIP	Mixed integer programming
LP	Linear programming
ML	Machine learning
MIL	Multiple instance learning
RL	Reinforcement learning

References

1. Pan, C.H. A study of integer programming formulations for scheduling problems. *International Journal of Systems Science* **1997**, *28*, 33–41. https://doi.org/10.1080/00207729708929360.
2. Guihaire, V.; Hao, J.K. Transit network design and scheduling: A global review. *Transportation Research Part A: Policy and Practice* **2008**, *42*, 1251–1273. https://doi.org/https://doi.org/10.1016/j.tra.2008.03.011.
3. Díaz-Madroñero, M.; Mula, J.; Peidro, D. A review of discrete-time optimization models for tactical production planning. *International Journal of Production Research* **2014**, *52*, 5171–5205. https://doi.org/10.1080/00207543.2014.899721.
4. Mitchell, J.E. Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization* **2002**, *1*, 65–77.
5. Bengio, Y.; Lodi, A.; Prouvost, A. Machine Learning for Combinatorial Optimization: a Methodological Tour d'Horizon. *CoRR* **2018**, *abs/1811.06128*, [1811.06128].
6. He, H.; Daume III, H.; Eisner, J.M. Learning to Search in Branch and Bound Algorithms. In Proceedings of the Advances in Neural Information Processing Systems; Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N.; Weinberger, K., Eds. Curran Associates, Inc., 2014, Vol. 27.
7. Khalil, E.; Le Bodic, P.; Song, L.; Nemhauser, G.; Dilkina, B. Learning to Branch in Mixed Integer Programming. *Proceedings of the AAAI Conference on Artificial Intelligence* **2016**, *30*. https://doi.org/10.1609/aaai.v30i1.10080.
8. Khalil, E.; Dilkina, B.; Nemhauser, G.; Ahmed, S.; Shao, Y. Learning to Run Heuristics in Tree Search. 2017, pp. 659–666. https://doi.org/10.24963/ijcai.2017/92.
9. Huang, L.; Chen, X.; Huo, W.; Wang, J.; Zhang, F.; Bai, B.; Shi, L. Branch and Bound in Mixed Integer Linear Programming Problems: A Survey of Techniques and Trends, 2021. https://doi.org/10.48550/ARXIV.2111.06257.
10. Tang, Y.; Agrawal, S.; Faenza, Y. Reinforcement Learning for Integer Programming: Learning to Cut. In Proceedings of the Proceedings of the 37th International Conference on Machine Learning; III, H.D.; Singh, A., Eds. PMLR, 2020, Vol. 119, *Proceedings of Machine Learning Research*, pp. 9367–9376.
11. Paulus, M.B.; Zarpellon, G.; Krause, A.; Charlin, L.; Maddison, C. Learning to Cut by Looking Ahead: Cutting Plane Selection via Imitation Learning. In Proceedings of the International Conference on Machine Learning. PMLR, 2022, pp. 17584–17600.
12. Huang, Z.; Wang, K.; Liu, F.; Zhen, H.L.; Zhang, W.; Yuan, M.; Hao, J.; Yu, Y.; Wang, J. Learning to Select Cuts for Efficient Mixed-Integer Programming. *Pattern Recogn.* **2022**, *123*. https://doi.org/10.1016/j.patcog.2021.108353.
13. Erhan, D.; Courville, A.; Bengio, Y.; Vincent, P. Why does unsupervised pre-training help deep learning? In Proceedings of the Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, 2010, pp. 201–208.
14. Ranzato, M.; Huang, F.J.; Boureau, Y.L.; LeCun, Y. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In Proceedings of the 2007 IEEE conference on computer vision and pattern recognition. IEEE, 2007, pp. 1–8.
15. Bengio, Y.; Lamblin, P.; Popovici, D.; Larochelle, H. Greedy layer-wise training of deep networks. *Advances in neural information processing systems* **2006**, *19*.
16. Wesselmann, F.; Stuhl, U. Implementing cutting plane management and selection techniques. In *Technical Report*; University of Paderborn, 2012.

17. Arık, S.Ö.; Pfister, T. Tabnet: Attentive interpretable tabular learning. In Proceedings of the Proceedings of the AAAI Conference on Artificial Intelligence, 2021, Vol. 35, pp. 6679–6687.
18. Gleixner, A.; Hendel, G.; Gamrath, G.; Achterberg, T.; Bastubbe, M.; Berthold, T.; Christophel, P.M.; Jarck, K.; Koch, T.; Linderoth, J.; et al. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation* **2021**. https://doi.org/10.1007/s12532-020-00194-3.
19. Diaby, M. Linear programming formulation of the set partitioning problem. *Int. J. Operational Research Int. J. Operational Research* **2010**, *8*, 399–427. https://doi.org/10.1504/IJOR.2010.034067.
20. Forrest, J.; Ralphs, T.; Santos, H.G.; Vigerske, S.; Forrest, J.; Hafer, L.; Kristjansson, B.; jpfasano.; EdwinStraver.; Lubin, M.; et al. coin-or/Cbc: Release releases/2.10.8 **2022**. https://doi.org/10.5281/zenodo.6522795.
21. Atamtürk, A. On the Facets of the Mixed–Integer Knapsack Polyhedron. *Mathematical Programming* **2003**, *98*, 145–175.
22. Atamtürk, A.; oz, J.C.M. A Study of the Lot-Sizing Polytope. *Mathematical Programming* **2004**, *99*, 443–465.
23. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A.; Dean, J.; Devin, M.; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from tensorflow.org.