Learning Primal Heuristic Frequency in Mixed Integer Programming

Justin Dumouchelle
McGill: ID 260954626
justin.dumouchelle@mail.mcgill.ca

April 2020

1 Introduction

1.1 Problem Description

Mixed Integer Linear Programming (MILP or MIP) is a methodology in mathematical optimization which has been used in a wide variety of practical problems such as vehicle routing [1], kidney exchange [2], and airline crew scheduling [3]. In recent years primal heuristics have played a significant role in the success of modern MIP solvers [4]. Primal heuristics find feasible solutions to a MIP instance, which can reduce the size of the branch-and-bound tree by pruning nodes which are known to lead to sub-optimal solutions. Despite the effectiveness of primal heuristics, the open question of when to run primal heuristics still remains. In this work, we examine the use of reinforcement learning as a methodology to decide how frequently a primal heuristic should be run in branch-and-bound. Specifically, we frame this as a contextual bandit problem where given a MIP instance we set a hyperparameter of SCIP [5], an open source MIP solver, which decides how frequently a heuristic should be run.

1.2 Background

1.2.1 Mixed Integer Programming

A MIP problem is a constrained linear optimization problem where we require a subset of the variable in the optimal solution to be integer. In general a MIP problem can be described as:

$$\min\{c^T x : Ax \leq b, x \geq 0, x_j \in \mathbb{Z}, \forall j \in \mathcal{I}\}$$

where $x \in \mathbb{R}^n$ denotes the variables, $c \in \mathbb{R}^n$ denotes the linear cost function, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ define the constraints, and $\mathcal{I} \subseteq \{1, ..., n\}$ is the subset of variables which require integrality. In general, MIP problems are NP-Hard, however research over the past several decades has led to viable techniques to solve a wide variety of practical problems.

1.2.2 Branch-and-bound

Modern state-of-the-art MIP solvers are based on the branch-and-bound algorithm run in conjunction with several other algorithmic components which improve efficiency [6]. Branch-and-bound is

tree based search method which is used to explore the search space and provide optimal solutions to MIP instances. At each node we solve a linear programming (LP) relaxation which gives a solution x. If x satisfies the integrality constraints defined by $\mathcal{I}$, then this defines a leaf node as we cannot branch further. Similarly, if we have a solution $\bar{x}$ which satisfies integrality constraints and $c^T\bar{x} \leq c^T x$, then we can prune this node as exploring further will not result in any better solutions. If the x does not satisfy either of the previous two conditions, then we need to choose a fractional variable $j \in \mathcal{I}$ and define two sub-problems by adding the constraint to the LP relaxation bounding $x_j \leq \lfloor a \rfloor$ and $x_j \geq \lceil a \rceil$, where a is the value which x_j took in the nodes LP relaxation. It can also be the case the LP relaxation at a node is infeasible, in which case the node can be pruned as we do not further need to explore the sub-tree. This process is repeated until every node is a leaf or pruned.

1.2.3 Primal Heuristics

Primal heuristics are used to compute feasible solutions to MIP instances throughout the branch-and-bound process. One main objective of primal heuristics is to generate incumbent solutions (minimal cost feasible solutions so far), which can be used to prune nodes in the branch-and-bound tree.

1.2.4 SCIP

SCIP is a state-of-the-art open-source MIP solver. SCIP implements a wide variety of primal heuristics and the decision of when they should be run is based on the priority and frequency. Without user specification these values remain fixed regardless of the problem instance.

2 Methodology

In this we project explore the use of a contextual bandit as a methodology to learn the frequency of a heuristics in SCIP. We specifically focus on the heuristic vectorlength diving as it provides quality solutions at a high computation cost [7]. We also disable all other heuristics in order to view the effectiveness of vectorlength diving in isolation. In the remainder of this section we will describe the observation space, actions, rewards, and the contextual bandit algorithm used.

2.1 Observations

The observation is defined using basic statistics of the MIP instance. Specifically, we take the features to be defined by (1) the number of variables, (2) the number of constraints, (3) the number of continuous variables, (4) the number of integer variables, and (5) the number of binary variables. This results in a 5-tuple to define an observation.

2.2 Actions

In the most general case the frequency of a heuristics can be any integer in the range $[-1, 65534]$, with -1 indicating not to run the heuristic, 0 indicating to only be run once (at some specified node), and 1 to 65534 indicating a decreasing frequency to run the heuristic (i.e. 1 is the most frequent and 65534 is the least frequent. In this project we simplify this action space by only

considering a small set of 3 actions, which include not running the heuristic (-1), running the heuristics at the highest frequency (1), and running the heuristic at the SCIP default frequency (10).

2.3 Rewards

The rewards are defined by the primal integral [8]. The primal integral was chosen as it reflects the effectiveness of a primal heuristic throughout the entire solving process, favors quality solutions early in the solving process (pruning a maximal number of nodes), and can be computed with a fixed time limit for the solving process. As we want to minimize the primal integral, we consider the reward to be defined by the negative primal integral to formulate this as a maximization of reward. Note that for simplicity in this project we consider an approximation of this quantity at each second, rather than at the time when a new incumbent is found.

2.4 Modelling

Using the state, action, and reward, we implement this task as a one-step decision making process. As we have a one-step decision making process with an observation, we view this as a contextual bandit problem. We consider LinUCB [9] with disjoint linear model for each arm to model the problem. LinUCB is a contextual bandit approach which defines a linear model for each arm which maps the observation space to the reward space. In [9], the application is with respect to new article recommendation, which provides a set of features for both the users and actions. This results in them having a context vector which is defined by features of the user and actions. In our problem setting there is no clear set of features with respect to the arm, so we consider our context vector to be defined completely by the observations with respect to the MIP instance.

3 Results

3.1 Data

The data was collected from a subset of the easy MIP instances from MIPLIB2010 [10] and MIPLIB2017 [11]. Solving the MIP instances to completion can be problematic as the time to solve for some instances and actions can be prohibitively expensive. To resolve this we consider the problem of minimizing the primal integral over a finite time horizon of 5 minutes. The data was then generated offline by solving each MIP instance with every action over 5 seeds with the specified timeouts. From these solved instances, we then excluded two problems. (1) Problems where no primal bound was found for any action within the specified time limit, and (2) Instances which were trivially solved by SCIP in less than 5 seconds. We then sample from this data in an online fashion while learning with LinUCB. The offline data generation was done to ensure reputability of the LinUCB results and reduce compute time when searching for hyperparameters. It was also done in parallel which greatly reduces the compute time.

3.2 Evaluation

To evaluate the performance we split the set of MIP instances into a train and test set with 80% and 20% of the instances respectively. We use the set of train instances and sample from their offline data to train our model to provide a realistic online training simulation.

To evaluate the performance of our model, we evaluate the model at each episode. Since we have the data which was generated offline, we can compute the performance of our model by greedily selecting the action which maximizes expected reward for each instance and averaging the result over seeds. We can then plot this reward averaged over instances at each time step to see the performance across the entire set of training instances.

3.3 Hyperparameter search

In LinUCB, the hyperparameter α, defines the magnitude at which we explore vs. exploit. Higher values of α will dominate the expected reward causing the model prefer exploration. We perform a hyperparameter search across $\alpha \in \{0.1, 0.5, 1.0, 5.0, 10.0\}$. The models are trained for 1000 episodes and averaged over 10 runs. Figure 1 shows the mean reward averaged over all training instances, seeds, and runs based on the greedy action by the bandit.

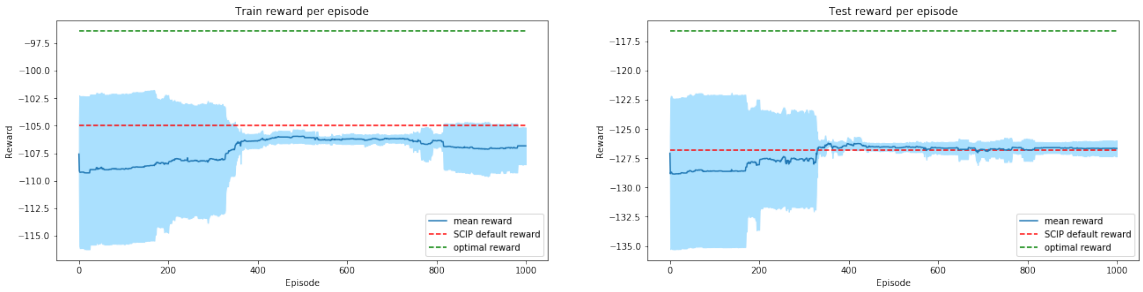
Figure 1: Training reward for $\alpha \in \{0.1, 0.5, 1.0, 5.0, 10.0\}$

From Figure 1 we can see that $\alpha = 1.0$ appears to learn the best. For $\alpha < 1$ the bandit performance remains consistent indicating that the bandit may be exploiting actions which it deems to be optimal early in the training process. For $\alpha > 1$ we can see variation between episodes, but there does not appear to be much in terms of learning. One reason for this could be for the allowance of too much exploration. For these reasons we will continue with the use of $\alpha = 1.0$ in the following section.

3.4 Train and Test results

With the best hyperparameters found in the previous section we will again plot the mean curves over the train results along with their standard deviation. In addition we can use the offline data to compute both the optimal action/reward for each instance and the action/reward for the default SCIP parameter. We also evaluate the performance of our model on the set of test instances which have not been seen during training. See Figures 2a and 2b for the train and test plots respectively.

From Figure 2, we can see similar patterns in improvement in both the training and testing results. This can be indicative of some generalization between the two settings. Unfortunately, we do not out-preform the default SCIP setting in the case of the training instances, which is equivalent having a bandit choose the same arm regardless of context. From the results it is difficult to draw further conclusions since the learning is not particularly strong, but overall they suggest potential of learning heuristic frequency in SCIP.



(a) Training reward

(b) Test reward

Figure 2: Train and test results in comparison to SCIP and optimal

4 Conclusion

This project gave me the opportunity to explore a new area of reinforcement learning, contextual bandits. In addition it gave me the opportunity to work with SCIP and investigate the use of reinforcement learning as a methodology for learning in MIP solving. Although the results were not particularly strong, they do indicate some learning, making this a potential area to explore more rigorously. A future direction could be to improve the observation space and investigate the use of other contextual bandit models. Another potential direction for future work could be to consider the task of deciding if we should run a heuristic at a given node in the branch-and-bound tree. This would provide a much richer feature space as we have access to node specific information as well as more control over the MIP solving process. In fact, in [12], they show improvements in MIP solving by running heuristics at a given node when they predict success via a classification approach. Developing an environment within MIP solvers is a much more difficult task, however it appears to be an appropriate direction given the potential to improve upon state-of-the-art MIP solvers.

5 Notes

A link to the github repo containing all the code can be found here:
https://github.com/jdumouchelle/scip_bandit

A link to the to the Youtube presentation can be found here:
https://youtu.be/mAjXY_HI-9k

References

[1] R. Kulkarni and P. R. Bhave, "Integer programming formulations of vehicle routing problems," *European Journal of Operational Research*, vol. 20, no. 1, pp. 58–67, 1985.

[2] D. J. Abraham, A. Blum, and T. Sandholm, "Clearing algorithms for barter exchange markets: Enabling nationwide kidney exchanges," in *Proceedings of the 8th ACM conference on Electronic commerce*. ACM, 2007, pp. 295–304.

[3] D. Wedelin, "An algorithm for large scale 0–1 integer programming with application to airline crew scheduling," *Annals of operations research*, vol. 57, no. 1, pp. 283–301, 1995.

[4] T. Achterberg and R. Wunderling, "Mixed integer programming: Analyzing 12 years of progress," in *Facets of combinatorial optimization*. Springer, 2013, pp. 449–481.

[5] T. Achterberg, "Scip: solving constraint integer programs," *Mathematical Programming Computation*, vol. 1, no. 1, pp. 1–41, 2009.

[6] A. Atamtürk and M. W. Savelsbergh, "Integer-programming software systems," *Annals of operations research*, vol. 140, no. 1, pp. 67–124, 2005.

[7] T. Berthold, "Primal heuristics for mixed integer programs," 2006.

[8] T. Achterberg, T. Berthold, and G. Hendel, "Rounding and propagation heuristics for mixed integer programming," in *Operations research proceedings 2011*. Springer, 2012, pp. 71–76.

[9] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 661–670.

[10] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz *et al.*, "Miplib 2010," *Mathematical Programming Computation*, vol. 3, no. 2, p. 103, 2011.

[11] A. Gleixner, G. Hendel, G. Gamrath, T. Achterberg, M. Bastubbe, T. Berthold, P. M. Christophel, K. Jarck, T. Koch, J. Linderoth *et al.*, "Miplib 2017: Data-driven compilation of the 6th mixed-integer programming library," *Optimization online preprint: http://www. optimization-online. org/DB_HTML/2019/07/7285. html. Submitted to Mathematical Programming Computation*, 2019.

[12] E. B. Khalil, B. Dilkina, G. L. Nemhauser, S. Ahmed, and Y. Shao, "Learning to run heuristics in tree search." in *IJCAI*, 2017, pp. 659–666.