Efficient Primal Heuristics for Mixed-Integer Linear Programs

Akang Wang, Linxin Yang, Sha Lai, Xiaodong Luo
Shenzhen Research Institute of Big Data
Shenzhen, China
wangakang@sribd.cn, yanglinxin, laisha, xiaodongluo@cuhk.edu.cn

Xiang Zhou, Haohan Huang, Shengcheng Shao, Yuanming Zhu, Dong Zhang, Tao Quan
Huawei GTS, China
zhouxiang60, huanghaohan, shaoshengcheng, zhuyuanming5@huawei.com
zhangdong48, quantao@huawei.com

Abstract

This paper is a short report about our work for the primal task in the Machine Learning for Combinatorial Optimization NeurIPS 2021 Competition. For each dataset of our interest in the competition, we propose customized primal heuristic methods to efficiently identify high-quality feasible solutions. The computational studies demonstrate the superiority of our proposed approaches over the competitors'.

1 Introduction

Many combinatorial optimization problems can be modeled and solved via *Mixed-integer linear programming* (MILP). As shown by (1), the MILP problem seeks for an assignment to continuous and discrete decision variables (x and y, respectively) that are subject to linear constraints, with the goal of minimizing a linear objective. A wide range of important applications in transportation, manufacturing, and many other industrial domains can be formulated as MILP models. However, solving those models to guaranteed optimality is generally $\mathcal{NP}$-hard.

$$
\begin{aligned}
\min_{x,y} \quad & c^\top x + d^\top y \\
\text{s.t.} \quad & Ax + By \leq h \\
& x \in \mathbb{R}^n, y \in \mathbb{Z}^m
\end{aligned} \tag{1}
$$

Various general-purpose primal heuristics have been developed to produce feasible solutions to MILP problems. Among them, the simplest one is *rounding heuristic*, which rounds each fractional element in a solution to its nearest integer. Although the rounding method is computationally cheap, it often fails to identify feasible points. A more sophisticated idea is called *feasibility pump* [5, 2]. Briefly speaking, feasibility pump iteratively calls a rounding step and a projection step to ensure that the returned solutions satisfy integrality constraints and linear constraints, respectively. Usually, the aforementioned heuristics could not produce solutions of high quality and thus are often employed to construct the very first feasible solutions, while RINS [4], RENS [3], and other computationally expensive primal heuristics are used for improving the incumbent solutions. These improvement heuristics search for new incumbents within a neighborhood of some given feasible solutions.

In this work, we tackle a few problem-specific MILP formulations of our interest by utilizing classic primal heuristics but in a more judicious manner, by exploiting the problem structures. We provide problem definitions in Section 2 and our solution approaches in Section 3. Computational results are summarized in Section 4. Finally, we conclude our work in Section 5.

2 Problem definitions

The Machine Learning for Combinatorial Optimization (ML4CO) NeurIPS 2021 Competition [1] focuses on the design of application-specific algorithms for solving MILP models. Specifically, the primal task aims to identify high-quality incumbent solutions as fast as possible. For an MILP instance of the form (1), as a primal algorithm proceeds, new improved solutions will be identified and thus primal bounds are updated, as shown in Figure 1. The primal integral (the area of the shaded region) is then used as the metric for evaluating different primal algorithms. Clearly, an algorithm performing well will exhibit a small primal integral.
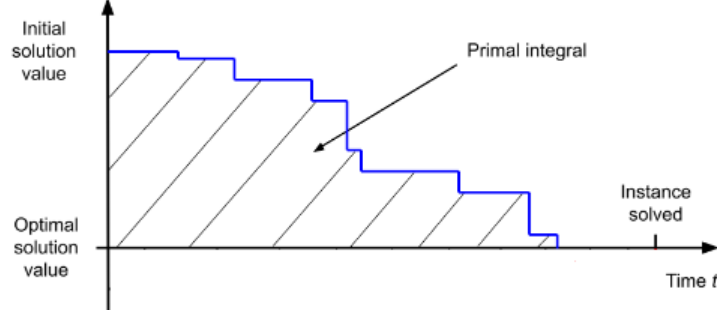

Figure 1: Primal bounds evolution versus solving time (adapted from [1])

There are three problem benchmarks in the ML4CO competition: *item placement*, *load balancing*, and *anonymous*. The first two problem benchmarks are inspired by real-life applications of large-scale systems at Google, while the third benchmark is an anonymous problem inspired by a real-world, large-scale industrial application. There are $10,000$ instances for the first two benchmarks, respectively, and 118 instances in the anonymous dataset, as training data. The competition encourages both machine learning (ML) based algorithms and non-ML algorithms.

2.1 Item placement

This item placement problem deals with spreading items across multiple containers utilizing them evenly. Let I denote the set of items and J denote the set of containers. Let a binary variable x_{ij} be 1 if item i is placed in container j and 0 otherwise. Each item will be placed in exactly a single container, as shown by constraints (3). Let K represent the set of dimensions. For dimension $k \in K$ of container $j \in J$, knapsack constraints (4) represent some physical considerations while (5) and (6) properly account for the placement unevenness, which is penalized in the objective (2). The goal is to identify an assignment for x, y, z such that constraints (3) – (8) are satisfied and the objective (2) is minimized.

$$\min_{x,y,z} \quad \sum_{j \in J} \sum_{k \in K} \alpha_k y_{jk} + \sum_{k \in K} \beta_k z_k \tag{2}$$

$$\text{s.t.} \quad \sum_{j \in J} x_{ij} = 1 \qquad\qquad \forall i \in I \tag{3}$$

$$\sum_{i \in I} a_{ik} x_{ij} \leq b_k \qquad\qquad \forall j \in J, \forall k \in K \tag{4}$$

$$\sum_{i \in I} d_{ik} x_{ij} + y_{jk} \geq 1 \qquad\qquad \forall j \in J, \forall k \in K \tag{5}$$

$$y_{jk} \leq z_k \qquad\qquad \forall j \in J, \forall k \in K \tag{6}$$

$$x_{ij} \in \{0, 1\} \qquad\qquad \forall i \in I, \forall j \in J \tag{7}$$

$$y_{jk} \geq 0 \qquad\qquad \forall j \in J, \forall k \in K \tag{8}$$

2.2 Load balancing

This problem deals with apportioning workloads across as few machines as possible. Let M and N denote the set of tasks and the set of machines, respectively. For task $i \in M$, only a subset of machines, denoted by $N^i \subseteq N$, are accessible. Let a binary variable y_j be 1 if machine j is used and 0 otherwise. Let x_{ij} denote the amount of workload from task i to machine j, as defined in constraints (10). Constraints (11) enforce the capacity requirement for each machine. The apportionment is required to be robust to any one machine's failure, as indicated by constraints (12). The goal is to identify an assignment for x, y such that constraints (10) – (14) are satisfied and the objective (9) is minimized.

$$\min_{x,y} \quad \sum_{j \in N} y_j \tag{9}$$

$$\text{s.t.} \quad x_{ij} \leq a_i y_j \qquad \forall i \in M, \forall j \in N^i \tag{10}$$

$$\sum_{i \in M : j \in N^i} x_{ij} \leq b_j \qquad \forall j \in N \tag{11}$$

$$\sum_{j \in N^i \setminus \{j'\}} x_{ij} \geq a_i \qquad \forall i \in M, \forall j' \in N^i \tag{12}$$

$$y_j \in \{0, 1\} \qquad \forall j \in N \tag{13}$$

$$0 \leq x_{ij} \leq b_j \qquad \forall i \in M, \forall j \in N^i \tag{14}$$

2.3 Anonymous

The concrete MILP formulation corresponding to this benchmark is not provided in the ML4CO competition.

3 Solution approaches

3.1 Item placement

We analyze $10,000$ item placement instances and find out that

- $|I| = 105, |J| = 10$;
- a_{ik}, d_{ik} values of 5 items are significantly bigger than those of others.

Based on this empirical finding, we place the 5 big items into the first five containers respectively before any primal heuristics are applied.

Meta-heuristics   The item placement problem is simply a multi-dimensional multi-knapsack instance. The knapsack problem is one of the most well-studied combinatorial optimization problems, and tremendous efforts have been devoted to the development of efficient heuristic algorithms, from which we adopt some ideas for item placement. In particular, we first apply a greedy method in which items are first sorted based on their sizes and then assigned to containers. This will produce the very first feasible solution. After that, a *large neighborhood search* idea is considered as an improvement heuristic approach. We select one or two items respectively from two containers and swap them if this leads to a better incumbent.

Math-heuristics   Math-heuristic methods are based on solving mathematical models. We consider a construction method and an improvement method. The construction method consists of two steps: (i) first assign items to the first five containers by solving an assignment model; (ii) then assign the remaining items to the last five containers by solving a sub-MIP. The assignment model is defined as follows:

- (a) combine small items to generate candidates that might be placed in the first five containers;
- (b) assume all items that are placed at the remaining containers have "continuous" sizes (rather than "discrete");

(c) add an SOS1 constraint for each small item (to ensure that each item will be placed in at most one container).

Again, the large neighborhood search method is utilized for solution improvement. Specifically, we properly choose two out of the last five containers, and then solve a sub-MIP to reassign items within those two containers optimally.

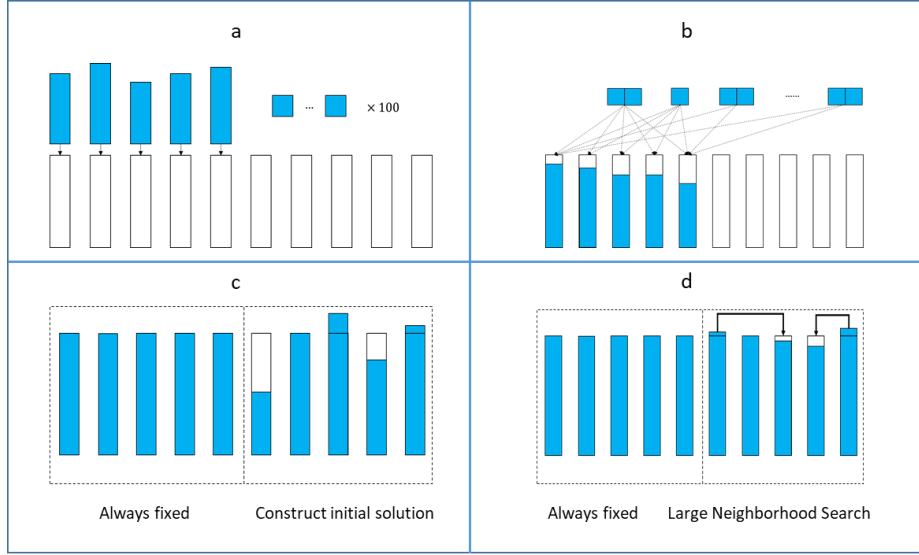A pictorial representation of the math-heuristic method is given in Figure 2.

Figure 2: Math-heuristics for item placement

3.2 Load balancing

Observation 1 *Rounding up a solution to the linear programming relaxation of model (9) – (14) would produce a feasible solution to (9) – (14).*

Observation 1 is clearly valid since rounding up the fractional element y_j with a non-zero value will not affect the satisfaction of linear constraints in the model but produce an integral solution. To further exploit the possibility of rounding a fractional solution towards a new incumbent, in our implementation, we adaptively select the rounding threshold η for y_j. In another word, we choose η in an adaptive manner and round up y_j only if it exceeds η. The large η values would lead to solutions with smaller objective values but might cause returned solutions to be infeasible, hence some care has been taken when choosing η. Specifically, we first select a target objective value based on the current primal and dual bounds and then determine η via quantile selection such that after rounding with η the objective matches the predetermined value. The initial primal bound corresponds to the feasible solution returned by rounding up every fractional element y_j with a non-zero value, while the initial dual bound is approximated by amplifying the root node lower bound with a certain factor. If the rounding step produces a new incumbent solution, we then update the primal bound; otherwise, we set the dual bound to the corresponding objective value. We iterate this process until the primal-dual gap falls below a certain threshold.

Observation 2 *In model (9) – (14), constraints (11) can be tightened as follows:*

$$\sum_{i \in M: j \in N^i} x_{ij} \le b_j y_j \quad \forall j \in N. \tag{15}$$

Due to constraints (10), $y_j = 0$ implies $x_{ij} = 0$, hence Observation 2 is valid. Furthermore, across $10,000$ instances, we also observe that $a_i < b_j$ for $i \in M, j \in N^i$. As a result, constraints (10) are

dominated by (15) and thus eliminated from the model. Now we call (9), (12) – (15) as the "tightened model".

In our experience, the tightened model has a much better root node bound. Thus, the root node linear programming (LP) solution is likely to be a good candidate for the rounding heuristics we discussed previously. In our implementation, we first apply the rounding heuristic method to the root LP solution to the original model and then to the optimal LP solution to tightened model. To further improve the incumbent, we then adopt RINS. In particular, we define and solve a sub-MIP based on the tightened model, using its LP solution and the current incumbent as a guide to fix part of the binary variables.

3.3 Anonymous

Though the concrete MILP formulation is not available, we can discover some pattern from MILP instances in the LP file format.

Observation 3 *One can define a planning horizon and associate with each discrete variable a time period.*

The details of Observation 3 can be deduced from the constraint hypergraph [9] in which every node represents a discrete variable and every edge joins a pair of variables if they occur together in a constraint. Let H denote the planning horizon and $h \in H$ denote a time period, as shown in Figure 3.
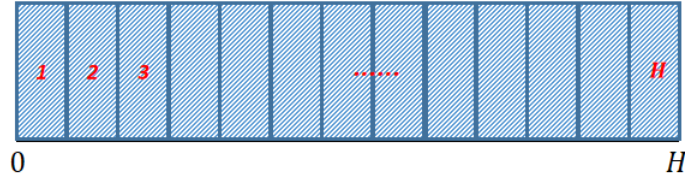

Figure 3: Planning horizon

Based on Observation 3, we propose our first primal heuristic called a "rolling-horizon" method (Figure 4), which consists of the following steps:

   (i) ignore constraints involving discrete variables with their h values greater than $\widetilde{H}$;

  (ii) relax the integrality constraints on variables with their h values greater than $\overline{H}$;

 (iii) fix those discrete variables with their h values less than $\widehat{H}$ at the optimal solution from a previous run;

 (iv) solve the sub-MIP;

  (v) increase $\widehat{H}, \overline{H}$ and $\widetilde{H}$ and then iterate steps (i) - (iv) until $\widehat{H} = H$.
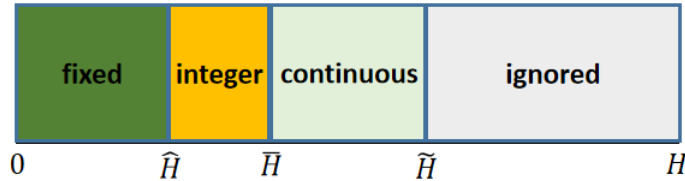

Figure 4: The rolling-horizon method

Parameters $\widehat{H}, \overline{H}$ and $\widetilde{H}$ are selected adaptively to balance feasibility, optimality, and the computational cost of solving sub-MIPs. We remark that the rolling-horizon method might be generally

5

applicable to optimization models with discrete variables spanning consecutive time periods, for example, inventory management models.

The rolling-horizon method is often quite time-consuming, hence we first utilize some computationally favorable methods to generate the very first solutions. In particular, we implement a variant of feasibility pump that combines a rounding step and a projection step, using weak and strong perturbation to escape cycle or stalling. After obtaining the first incumbent, we then call RENS. Specifically, the RENS model is a sub-MIP defined by fixing those discrete variables with their h values less than $0.9H$ at the incumbent.

4 Computational Results

We implement our code in Python except that the meta-heuristic algorithm is implemented in C++. The evaluation is conducted via Ecole 0.7.3 [8], using the PySCIPOpt 3.3.0 [7] interface to call SCIP 7.0.3 [6] as the LP and MILP solver. The test set includes 100 instances for both item placement and load balancing, 20 anonymous instances (run each instance with 5 different seeds). The time limit is 300 seconds for each instance and the average primal integral is reported as the performance metric.

We adopt the final results from [1] and provide the synopsized version in Table 1. The columns "Item placement", "Load balancing", and "Anonymous" report the average primal integral values across 100 runs, respectively. Compared with the second places (2nd in Table 1) in item placement, load balancing and anonymous benchmarks, our proposed approaches produce 67.5%, 51.8%, and 7.7% smaller primal integral values, respectively. Clearly, our customized approaches significantly outperform those from other teams in the competition.

Table 1: Computational results in the ML4CO competition

Team	Primal integral		
	Item placement	Load balancing	Anonymous
Our work	358.43	1145.31	35220755.00
2nd	1102.19	2374.55	38163724.79

5 Conclusions

In this work, we propose specialized primal algorithms for tackling item placement, load balancing and anonymous benchmarks in the ML4CO competition. Our approaches are mainly based on classic primal heuristics in the MILP literature. However, we exploit the problem structures and employ those heuristics in a much more sophisticated manner. As a result, our proposed algorithms perform significantly better than the other approaches in the ML4CO competition.

References

[1] https://www.ecole.ai/2021/ml4co-competition/, accessed on January 22, 2022.

[2] Livio Bertacco, Matteo Fischetti, and Andrea Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 4(1):63–76, 2007.

[3] Timo Berthold. Rens. *Mathematical Programming Computation*, 6(1):33–54, 2014.

[4] Emilie Danna, Edward Rothberg, and Claude Le Pape. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90, 2005.

[5] Matteo Fischetti, Fred Glover, and Andrea Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.

[6] 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.

[7] Stephen Maher, Matthias Miltenberger, Joao Pedro Pedroso, Daniel Rehfeldt, Robert Schwarz, and Felipe Serrano. Pyscipopt: Mathematical programming in python with the scip optimization suite. In *International Congress on Mathematical Software*, pages 301–307. Springer, 2016.

[8] Antoine Prouvost, Justin Dumouchelle, Lara Scavuzzo, Maxime Gasse, Didier Chételat, and Andrea Lodi. Ecole: A gym-like library for machine learning in combinatorial optimization solvers. *arXiv preprint arXiv:2011.06069*, 2020.

[9] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming*. Elsevier, 2006.