Adaptive Constraint Partition Based Optimization Framework for Large-Scale Integer Linear Programming (Student Abstract)

Huigen Ye[1,2], Hongyan Wang[1], Hua Xu[1*], Chengming Wang[3], Yu Jiang[3]

[1] State Key Laboratory of Intelligent Technology and Systems, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

[2] School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China

[3] Meituan Inc., Block F&G, Wangjing International R&D Park, No.6 Wang Jing East Rd, Chaoyang District, Beijing, 100102, China

yehg5@mail2.sysu.edu.cn, why17@mails.tsinghua.edu.cn, xuhua@tsinghua.edu.cn, wangchengming02@meituan.com, jiangyu31@meituan.com

Abstract

Integer programming problems (IPs) are challenging to be solved efficiently due to the NP-hardness, especially for large-scale IPs. To solve this type of IPs, Large neighborhood search (LNS) uses an initial feasible solution and iteratively improves it by searching a large neighborhood around the current solution. However, LNS easily steps into local optima and ignores the correlation between variables to be optimized, leading to compromised performance. This paper presents a general adaptive constraint partition-based optimization framework (ACP) for large-scale IPs that can efficiently use any existing optimization solver as a subroutine. Specifically, ACP first randomly partitions the constraints into blocks, where the number of blocks is adaptively adjusted to avoid local optima. Then, ACP uses a subroutine solver to optimize the decision variables in a randomly selected block of constraints to enhance the variable correlation. ACP is compared with LNS framework with different subroutine solvers on four IPs and a real-world IP. The experimental results demonstrate that in specified wall-clock time ACP shows better performance than SCIP and Gurobi.

Introduction

Integer programming problems (IPs) are usually NP-hard, and their algorithm design is challenging and research-worthy. Even with advances, the performance of traditional tree algorithms for IPs, such as branch-and-bound (Zarpellon, Jo et al. 2021) and branch-and-cut (Huang, Wang et al. 2022), decrease severely in high-dimensional search spaces. Therefore, Large neighborhood search (LNS) (Shaw 1998; Pisinger and Ropke 2010; Song et al. 2020), defining neighborhoods and optimizing blocks that contain a subset of decision variables, has been widely used and achieved good results for many real-world large-scale IPs(Sonnerat, Wang et al. 2021; Li, Chen et al. 2022). However, for IPs with millions of decision variables, the traditional LNS frameworks ignore the correlation between variables and easily step into local optima . This paper proposes ACP to address the above two issues. The key idea of ACP is that it adaptively updates the number of blocks to avoid local optima, and uses a two-step variable selection method to enhance the correlation between variables to be optimized. Results on four large-scale IPs and a real-world IP verify the effectiveness of ACP.

Algorithm 1: The framework of ACP

Input: An IP P with constraints C and variables X, an initial feasible solution S_X, a subroutine solver F
Output: A solution S_X

1: **while** Specified wall-clock time not met **do**
2: $C = C_1 \cup C_2 \cup \cdots \cup C_{k-1} \cup C_k$ ▷ *Constraint Partition*
3: $X_{sub} \leftarrow$ variables selected in a random block C_i
4: $X_{opt} \leftarrow S_X$
5: $S_X \leftarrow$ FIX_OPTIMIZE(P, S_X, X_{sub}, F) ▷ *Optimization*
6: **if** $f(S_X) - f(X_{opt}) < \epsilon f(X_{opt})$ **then** ▷ *Block Update*
7: $k \leftarrow k - 1$
8: **end if**
9: **end while**
10: **return** S_X

Method

As shown in Algorithm 1, ACP starts with an initial feasible solution that is artificially constructed, and iteratively improves the current solution. Then, constraints are randomly partitioned into disjoint blocks (Step 2). Each iteration only considers one block and the variables in this block are optimized, while other variables are fixed with the value of the current optimal feasible solution (Step 3-5). The number of blocks is adaptively updated according to the objective value improvement of recent two iterations with a pre-set threshold (Step 6-7). Based on ACP, the ACP2 framework is derived that uses the subroutine solver instead of artificially constructing to generate an initial feasible solution.

Constraint Partition. For an IP P with the decision variable set X and the constraint set C, ACP randomly divides C into k disjoint blocks $C_1, C_2, \ldots, C_k$ where $C = C_1 \cup C_2 \cup \cdots \cup C_k$. k is a hyperparameter that is different for different IPs.

Optimization. At each iteration, one block $C_i (i \in [1, \ldots, k])$ is randomly selected, and C_i differs at different iterations. Given a feasible solution S_x of the current IP P, all decision variables X_{sub} in the randomly selected subset C_i are treated as the local neighborhood of the search.

*Corresponding author

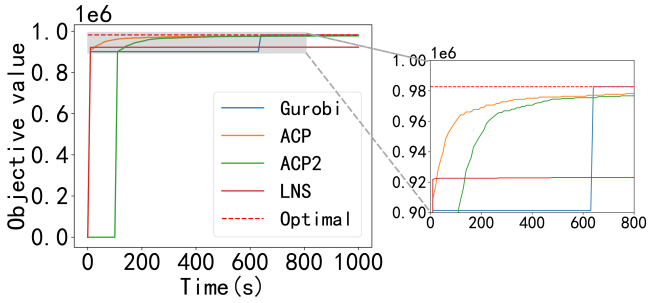

Figure 1: The objective value variation for real-world IP.

Then in function FIX_OPTIMIZE, a subroutine solver F (SCIP, Gurobi) is used to search the optimal solution of a sub-IP with decision variable set X_{sub}. All integer variables in $X \setminus X_{sub}$ are fixed with the value of the current optimal feasible solution X_{opt}. The new solution S_X is obtained by recombining X_{sub} and $X \setminus X_{sub}$.

Block update. To avoid getting stuck in local optima, ACP updates the number of blocks k with an optimization threshold of objective value improvement ϵ. If the improvement rate $f(S_X) - f(X_{opt})$ of the objective function $f(S_X)$ after one iteration is less than ϵ, the current solution is likely to be a local optimum. In this case, ACP reduces the number of blocks k to expand the neighborhood to jump out of the local optimum. Additional details such as the initial block number k and optimization threshold ϵ are given in the appendix.

Experiments

Experimental Settings. With different subroutine solvers, we compare ACP with LNS framework (Song et al. 2020) and the original subroutine solver on maximum independent set (IS), minimum vertex cover (MVC), maximum cut (MAXCUT), minimum set covering(SC) and one real-world large-scale IP in the internet domain. For IS, MVC and MAXCUT, we use random graphs of 1,000,000 points and 3,000,000 edges. For SC, we use random problem of 1,000,000 items and 1,000,000 sets. For the real-world large-scale IP, it has more than 800,000 decision variables and 50,000 constraints. All experiments are repeated 5 times and the average of the metric is recorded.

Results and Analysis. Table 1 shows the results of all the related methods on five IPs, and the best results are in bold fonts. Compared with SCIP and Gurobi, ACP obviously achieves much better performance on all the five IPs, especially for MAXCUT, SC and the real-world IP. It can also be seen that with constraint's block partition and adaptive update of block number, ACP reliably offers remarkable improvements over LNS framework with different subroutine solvers and the original subroutine solver. For clearer comparison, we further show the objective value variation with running time for the real-world large-scale IP in Figure 1. It can be found that ACP exhibits noteworthy advantages within limited wall-clock time, which can find the approximate optimal solution faster than other methods, even for the fastest commercial solver Gurobi. Due to the length limitation, we show other comparison results of all the related methods on small, medium and large-scale IPs in the appendix. The results also demonstrate the obvious advantages of ACP over other baseline methods.

Problem	Method	SCIP-based Obj. value	Gurobi-based Obj, value
IS (Maximize)	Solver	7866.55	215365.65
	LNS	194733.92	220368.34
	ACP	**207901.54**	227559.62
	ACP2	196742.1	**227636.51**
MVC (Minimize)	Solver	490857.44	283170.59
	LNS	304860.39	277870.09
	ACP	**291226.0**	271163.92
	ACP2	301836.42	**271087.22**
MAXCUT (Maximize)	Solver	9.02	971138.1
	LNS	553840.66	910662.36
	ACP	**829597.17**	1050400.97
	ACP2	747434.19	**1053583.50**
SC (Minimize)	Solver	919264.06	320034.15
	LNS	584210.54	224435.73
	ACP	**392082.53**	208062.73
	ACP2	432368.51	**201034.73**
Read-world IP (Maximize)	Solver	0.0	903119.36
	LNS	904387.43	924886.13
	ACP	**909113.41**	**948461.41**
	ACP2	907155.48	942235.76

Table 1: Comparison results for ACP and LNS with different subroutine solvers on different benchmark IPs.

Conclusion

This paper presents ACP for large-scale IPs that can efficiently use any existing optimization solver as a subroutine. The results and analysis show that the ACP framework is obviously superior to the mainstream method in specified wall-clock time. In the future, we will study the combination of LNS and graph neural networks for large-scale IPs.

Acknowledgements

This research was supported by Meituan.

References

Huang, Z.; Wang, K.; et al. 2022. Learning to select cuts for efficient mixed-integer programming. *Pattern Recognit.*, 123: 108353.

Li, J.; Chen, Z.; et al. 2022. MAPF-LNS2: Fast Repairing for Multi-Agent Path Finding via Large Neighborhood Search. In *AAAI'22*, volume 36, 10256–10265.

Pisinger, D.; and Ropke, S. 2010. Large neighborhood search. In *Handbook of metaheuristics*, 399–419.

Shaw, P. 1998. Using constraint programming and local search methods to solve vehicle routing problems. In *CP'98*, 417–431.

Song, J.; Yue, Y.; Dilkina, B.; et al. 2020. A General Large Neighborhood Search Framework for Solving Integer Linear Programs. In *NIPS'20*, volume 33, 20012–20023.

Sonnerat, N.; Wang, P.; et al. 2021. Learning a large neighborhood search algorithm for mixed integer programs. *arXiv:2107.10201*.

Zarpellon, G.; Jo, J.; et al. 2021. Parameterizing branch-and-bound search trees to learn branching policies. In *AAAI'21*, volume 35, 3931–3939.