数理解析研究所講究録
第2027巻 2017年 10-14

An approximation algorithm for the covering 0-1 integer program

Yotaro Takazawa * Shinji Mizuno †

Abstract

The covering 0-1 integer program is a generalization of fundamental combinatorial optimization problems such as the vertex cover problem, the set cover problem, and the minimum knapsack problem. In this article, extending a 2-approximation algorithm for the minimum knapsack problem by Carnes and Shmoys (2015), we propose a Δ_2-approximation algorithm, where Δ_2 is the second largest number of non-zero coefficients in the constraints.

1 Introduction

For a given minimization problem having an optimal solution, an algorithm is called an α-approximation algorithm if it runs in polynomial time and produces a feasible solution whose objective value is less than or equal to α times the optimal value. We study the covering 0-1 integer program (CIP), which is formulated as follows:

$$\text{CIP} \left| \begin{array}{ll} \min & \displaystyle\sum_{j\in N} c_j x_j \\ \text{s.t.} & \displaystyle\sum_{j\in N} a_{ij} x_j \geq b_i, \forall i \in M = \{1, \cdots, m\}, \\ & x_j \in \{0,1\}, \forall j \in N = \{1, \cdots, n\}. \end{array} \right. \tag{1}$$

where b_i, a_{ij}, and c_j $(i \in M,\ j \in N)$ are nonnegative. Assume that $\sum_{j\in N} a_{ij} \geq b_i$ for any $i \in M$, so that the problem is feasible. Let Δ_i be the number of non-zero coefficients in the i-th constraint $\sum_{j\in N} a_{ij} x_j \geq b_i$. Without loss of generality, we assume that $\Delta_1 \geq \Delta_2 \geq \cdots \geq \Delta_m$ and $\Delta_2 \geq 2$.

CIP is a generalization of fundamental combinatorial optimization problems such as the vertex cover problem, the set cover problem, and the minimum knapsack problem. There are some Δ_1-approximation algorithms for CIP, see Koufogiannakis and Young [4] and references therein.

*Department of Industrial Engineering and Management, Tokyo Institute of Technology
†Department of Industrial Engineering and Economics, Tokyo Institute of Technology

In this article, we propose a Δ_2-approximation algorithm for CIP. Our algorithm is an extension of a 2-approximation algorithm for the minimum knapsack problem which is a special case of CIP where $m = 1$ by Carnes and Shmoys [1]. Part of this article is included in Takazawa and Mizuno [5].

2 An algorithm and its analysis

Carnes and Shmoys [1] used an LP relaxation of the minimum knapsack problem, which was presented by Carr et al. [2]. We also use the following LP relaxation of (1):

$$\begin{array}{lll} \min & \sum_{j\in N} c_j x_j & \\ \text{s.t.} & \sum_{j\in N\setminus A} a_{ij}(A)x_j \geq b_i(A), & \forall A \subseteq N,\ \forall i \in M, \\ & x_j \geq 0, & \forall j \in N, \end{array} \tag{2}$$

where

$$\begin{array}{rcl} b_i(A) & = & \max\{0, b_i - \sum_{j\in A} a_{ij}\},\ \forall i \in M, \forall A \subseteq N, \\ a_{ij}(A) & = & \min\{a_{ij}, b_i(A)\},\ \forall i \in M, \forall A \subseteq N, \forall j \in N\setminus A. \end{array} \tag{3}$$

Carr et al. [2] show that any feasible 0-1 solution of (2) is feasible for (1). The dual problem of (2) can be stated as

$$\begin{array}{lll} \max & \sum_{i\in M}\sum_{A\subseteq N} b_i(A)y_i(A) & \\ \text{s.t.} & \sum_{i\in M}\sum_{A\subseteq N: j\notin A} a_{ij}(A)y_i(A) \leq c_j, & \forall j \in N, \\ & y_i(A) \geq 0, & \forall A \subseteq N,\ \forall i \in M. \end{array} \tag{4}$$

Now we introduce a well-known result for a primal-dual pair of linear programming [3].

Lemma 1. *Let $\bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{y}}$ be feasible solutions for the following primal and dual linear programming problems:*

$$\min\left\{\boldsymbol{c}^T\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b},\ \boldsymbol{x} \geq \boldsymbol{0}\right\} \quad \textit{and} \quad \max\left\{\boldsymbol{b}^T\boldsymbol{y} \mid \boldsymbol{A}^T\boldsymbol{y} \leq \boldsymbol{c},\ \boldsymbol{y} \geq \boldsymbol{0}\right\}.$$

If the conditions

$$\begin{array}{ll} \textit{(a):} & \forall j \in \{1, \cdots, n\}, \bar{x}_j > 0 \Rightarrow \sum_{i=1}^m a_{ij}\bar{y}_i = c_j, \\ \textit{(b):} & \forall i \in \{1, \cdots, m\}, \bar{y}_i > 0 \Rightarrow \sum_{j=1}^n a_{ij}\bar{x}_j \leq \alpha b_i \end{array}$$

hold, then $\bar{\boldsymbol{x}}$ is a solution within a factor of α of the optimal solution, that is, the primal objective value $\boldsymbol{c}^T\bar{\boldsymbol{x}}$ is less than or equal to α times the optimal value. (Note that the primal problem has an optimal solution because both the primal and dual problems are feasible.).

By applying Lemma 1 to the LP problems (2) and (4), we have the following result.

Lemma 2. *Let $\boldsymbol{x}$ and $\boldsymbol{y}$ be feasilbe solutions for (2) and (4), respectively. If these solutions satisfy*

$$\begin{array}{ll} (a): & \forall j \in N, x_j > 0 \Rightarrow \sum_{i \in M} \sum_{A \subseteq N: j \notin A} a_{ij}(A) y_i(A) = c_j, \\ (b): & \forall i \in M,\ \forall A \subseteq N, y_i(A) > 0 \Rightarrow \sum_{j \in N \setminus A} a_{ij}(A) x_j \leq \Delta_2 b(A), \end{array} \tag{5}$$

then $\boldsymbol{x}$ is a solution within a factor of Δ_2 of the optimal solution of (1).

Corollary 1. *Let $\boldsymbol{x}$ be a feasible 0-1 solution of (2) and $\boldsymbol{y}$ be a feasible solution of (4). If these solutions satisfy (5), $\boldsymbol{x}$ is a solution within a factor of Δ_2 of the optimal solution of (1).*

Our algorithm is presented in Algorithm 1 below. The goal is to find $\boldsymbol{x}$ and $\boldsymbol{y}$ which satisfy the conditions in Corollary 1. The algorithm generates a sequence of points $\boldsymbol{x}$ and $\boldsymbol{y}$ which always satisfy the following conditions:

- $\boldsymbol{x} \in \{0,1\}^n$.
- $\boldsymbol{y}$ is feasible for (4).
- $\boldsymbol{x}$ and $\boldsymbol{y}$ satisfy (5).

In Algorithm 1, we use the symbols $S = \{j \in N \mid x_j = 1\}$, $b_i(S) = \max\{0, b_i - \sum_{j \in S} a_{ij}\}$ for $i \in M$, and $\bar{c}_j = c_j - \sum_{i \in M} \sum_{A \subseteq N: j \notin A} a_{ij}(A) y_i(A)$ for $j \in N$.

Algorithm 1

Input: M, N, a_{ij}, b_i and c_j $(i \in M,\ j \in N)$.

Output: $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$.

Step 0: Set $\boldsymbol{x} = \boldsymbol{0}$, $\boldsymbol{y} = \boldsymbol{0}$, and $S = \emptyset$. Let $N_i' = \{j \in N \mid a_{ij} > 0\}$ for $i \in M$, $\bar{c}_j = c_j$ for $j \in N$, and $i = m$.

Step 1: If $i = 0$, then output $\tilde{\boldsymbol{x}} = \boldsymbol{x}$ and $\tilde{\boldsymbol{y}} = \boldsymbol{y}$ and stop. Otherwise set $b_i(S) = \max\{0, b_i - \sum_{j \in S} a_{ij}\}$ and go to Step 2.

Step 2: If $b_i(S) = 0$, then update $i = i - 1$ and go to Step 1. Otherwise calculate $a_{ij}(S)$ for any $j \in N_i' \setminus S$ by (3). Increase $y_i(S)$ while maintaining dual feasibility until at least one constraint $s \in N_i' \setminus S$ is tight. Namely set

$$y_i(S) = \frac{\bar{c}_s}{a_{is}(S)} \quad \text{for} \quad s = \arg\min_{j \in N_i' \setminus S} \left\{ \frac{\bar{c}_j}{a_{ij}(S)} \right\}.$$

Update $\bar{c}_j = \bar{c}_j - a_{ij}(S) y_i(S)$ for $j \in N' \setminus S$, $x_s = 1$, $S = S \cup \{s\}$, and $b_i(S) = \max\{0, b_i(S) - a_{is}\}$. Go back to the top of Step 2.

For the outputs $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$ of Algorithm 1, we have the following results.

Lemma 3. *$\tilde{\boldsymbol{x}}$ is a feasible 0-1 solution of (2) and $\tilde{\boldsymbol{y}}$ is a feasible solution of (4).*

Proof. By the assumption that (1) is feasible, $\boldsymbol{x} = (1, \cdots, 1)$ is feasible for the LP relaxation problem (2). Algorithm 1 starts from $\boldsymbol{x} = \boldsymbol{0}$ and updates a variable x_j from 0 to 1 at each iteration until each constraint in (2) is satisfied. Hence $\tilde{\boldsymbol{x}}$ is a feasible 0-1 solution of (2).

Algorithm 1 starts from the dual feasible solution $\boldsymbol{y} = \boldsymbol{0}$ and maintains dual feasibility throughout the algorithm. Hence $\tilde{\boldsymbol{y}}$ is feasible for (4). □

Lemma 4. *$\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$ satisfy (5).*

Proof. All the conditions in (a) of (5) are naturally satisfied by the way the algorithm updates primal variables. It suffices to show that all the conditions in (b) are satisfied. For any $i \in \{2, \cdots, m\}$ and any subset $A \subseteq N$ such that $\tilde{y}_i(A) > 0$, we obtain that

$$\sum_{j \in N \setminus A} a_{ij}(A)\tilde{x}_j \leq \Delta_i b_i(A) \leq \Delta_2 b_i(A),$$

since $a_{ij}(A) \leq b_i(A)$ by the definition (3) and the i-th constraint has Δ_i non-zero coefficients.

Then, we consider the case of $i = 1$. Define $\tilde{S} = \{j \in V \mid \tilde{x}_j = 1\}$. Let $\tilde{x}_\ell$ be the variable which becomes 1 from 0 at the last iteration of Step 2. From Step 2, $\tilde{y}_1(A) > 0$ implies

$$A \subseteq \tilde{S} \setminus \{\ell\}. \tag{6}$$

Since the algorithm does not stop just before setting $\tilde{x}_\ell = 1$, we have

$$\sum_{j \in \tilde{S} \setminus \{\ell\}} a_{1j} < b_1. \tag{7}$$

By (6) and (7), we observe that for any subset $A \subseteq N$ such that $\tilde{y}_1(A) > 0$

$$\sum_{j \in (\tilde{S} \setminus \{\ell\}) \setminus A} a_{1j}(A) \leq \sum_{j \in (\tilde{S} \setminus \{\ell\}) \setminus A} a_{1j} = \sum_{j \in \tilde{S} \setminus \{\ell\}} a_{1j} - \sum_{j \in A} a_{1j} < b_1 - \sum_{j \in A} a_{1j} \leq b_1(A),$$

where the first and last inequality follows from the definitions (3) of $a_j(A)$ and $b_i(A)$. Thus, we have that for any subset $A \subseteq N$ such that $\tilde{y}_1(A) > 0$

$$\sum_{j \in V \setminus A} a_{1j}(A)\tilde{x}_j = \sum_{j \in \tilde{S} \setminus A} a_{1j}(A) = \sum_{j \in (\tilde{S} \setminus \{\ell\}) \setminus A} a_{1j}(A) + a_{1\ell}(A) \leq \Delta_2 b_1(A),$$

where the last inequality follows from $a_{1\ell}(A) \leq b_1(A)$ and $\Delta_2 \geq 2$. □

Lemma 5. *The running time of Algorithm 2 is $O(\Delta_1(m+n))$.*

Proof. The running time of one iteration of Step 1 is $O(\Delta_1)$ and the number of iterations in Step 1 is at most m. On the other hand, the running time of one iteration of Step 2 is $O(\Delta_1)$ and the number of iterations in Step 2 is at most $m+n$. Therefore the total running time of the algorithm is $O(\Delta_1 m) + O(\Delta_1(m+n)) = O(\Delta_1(m+n))$. □

From the results above, we can obtain the next theorem.

Theorem 1. *Algorithm 2 is a Δ_2-approximation algorithm for CIP.*

3 Conclusion

The covering 0-1 integer program (CIP) is a generalization of fundamental combinatorial optimization problems. There are some Δ_1-approximation algorithms for CIP, where Δ_1 is the largest number of non-zero coefficients in the constraints. In this article, we extend a 2-approximation algorithm for the minimum knapsack problem by Carnes and Shmoys [1] to CIP and propse a Δ_2-approximation algorithm, where the second largest number of non-zero coefficients in the constraints.

Acknowledgment

This research is supported in part by Grant-in-Aid for Science Research (A) 26242027 of Japan Society for the Promotion of Science.

References

[1] T. Carnes and D. Shmoys: Primal-dual schema for capacitated covering problems, *Mathematical Programming*, **153** (2015), 289-308.

[2] R. D. Carr, L. Fleischer, V. J. Leung and C. A. Phillips: Strengthening integrality gaps for capacitated network design and covering problems, *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms* (2000), 106-115.

[3] D. Du, K. Ko and X. Hu: Design and Analysis of Approximation Algorithms, (*Springer Optimization and Its Applications*, 2011), 297-303.

[4] C. Koufogiannakis and N.E. Young: Greedy δ-approximation algorithm for covering with arbitrary constraints and submodular cost, Algorithmica, **66** (2013), 113-152.

[5] Y. Takazawa and S. Mizuno: A 2-approximation algorithm for the minimum knapsack problem with a forcing graph, to appear Journal of Operations Research Research of Japan (2017).

Yotaro Takazawa
Department of Industrial Engineering and Management
Tokyo Institute of Technology
2-12-1 Ohokayama
Meguro-ku Tokyo 152-8552, Japan
E-mail: takazawa.y.ab@m.titech.ac.jp