A 2-approximation algorithm for the minimum knapsack problem with a forcing graph

Yotaro Takazawa* and Mizuno Shinji†

June 29, 2016

Abstract

Carnes and Shmoys [2] presented a 2-approximation algorithm for the minimum knapsack problem. We extend their algorithm to the minimum knapsack problem with a forcing graph (MKPFG), which has a forcing constraint for each edge in the graph. The forcing constraint means that at least one item (vertex) of the edge must be packed in the knapsack. The problem is strongly NP-hard, since it includes the vertex cover problem as a special case. Generalizing the proposed algorithm, we also present an approximation algorithm for the covering integer program with 0-1 variables.

keywords: Approximation algorithms, Minimum knapsack problem, Forcing graph, Covering integer program

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

*Depertment of Industrial Engineering and Economics, School of Engineering, Tokyo Institute of Technology, 2-12-1-W9-58, Oo-Okayama, Meguro-ku, Tokyo, 152-8552, Japan. E-mail: takazawa.y.ab@m.titech.ac.jp

†Depertment of Industrial Engineering and Economics, School of Engineering, Tokyo Institute of Technology, 2-12-1-W9-58, Oo-Okayama, Meguro-ku, Tokyo, 152-8552, Japan. Tel.: +81-3-5734-2816, Fax: +81-3-5734-2947, E-mail: mizuno.s.ab@m.titech.ac.jp.

times the optimal value. Carnes and Shmoys [2] presented a 2-approximation algorithm for the following minimum knapsack problem:

$$\begin{array}{ll} \min & \displaystyle\sum_{j\in V} c_j x_j \\ \text{s.t.} & \displaystyle\sum_{j\in V} a_j x_j \geq b, \\ & x_j \in \{0,1\}, \quad \forall j \in V = \{1,\cdots,n\}, \end{array} \tag{1}$$

where V is a set of n items, $a_j,\ c_j \geq 0$ $(j \in V)$, and $b > 0$. Without loss of generality, we assume $\sum_{j\in V} a_j \geq b$ so that the problem is feasible.

In this paper, we propose a 2-approximation algorithm for the minimum knapsack problem with a forcing graph:

$$\text{MKPFG} \left| \begin{array}{lll} \min & \displaystyle\sum_{j\in V} c_j x_j & \\ \text{s.t.} & \displaystyle\sum_{j\in V} a_j x_j \geq b, & \\ & x_i + x_j \geq 1, & \forall \{i,j\} \in E, \\ & x_j \in \{0,1\}, & \forall j \in V = \{1,\cdots,n\}, \end{array} \right. \tag{2}$$

by extending the algorithm of Carnes and Shmoys [2], where E is a set of edges $\{i,j\} \in V \times V$. The constraint $x_i + x_j \geq 1$ means that either i or j must be chosen. It is called a forcing constraint and the graph $G = (V,E)$ is called a forcing graph.

The problem MKPFG (2) includes the minimum weight vertex cover problem (VCP) as a special case. It is known that VCP is a strongly NP-hard problem and has inapproximability such that the problem is hard to approximate within any constant factor better than 1.36 unless $P = NP$ [5] and 2 under unique games conjecture [9]. It follows that MKPFG is strongly NP-hard and has at least the same inapproximability as VCP. Bar-Yehuda and Even [1] proposed a 2-approximation algorithm for VCP, so we also extend their result.

The maximum version of MKPFG is known as the knapsack problem with a conflict graph (KPCG). KPCG is the maximum knapsack problem with disjunctive constraints for pairs of items which cannot be packed simultaneously in the knapsack. KPCG is also referred to as the disjunctively constrained knapsack problem. Exact and heuristic algorithms for KPCG were studied by [6, 7, 12] and approximation algorithms were proposed by

[10, 11]. Any exact algorithm for KPCG can solve MKPFG since MKPFG can be transformed into KPCG by complementing the variables. However, the approach of converting MKPFG into KPCG cannot be used in general when we consider the performance guarantee of approximation algorithms. To our knowledge, no approximation algorithms for MKPFG are presented so far.

In section3, we generalize our algorithm to the covering integer program with 0-1 variables (CIP), which is also referred to as the capacitated covering problem.

2 An Algorithm and Analysis

Carnes and Shmoys [2] used the following LP relaxation of the minimum knapsack problem (1), which was constructed by Carr et al. [3]:

$$\begin{array}{lll} \min & \displaystyle\sum_{j\in V} c_j x_j & \\ \text{s.t.} & \displaystyle\sum_{j\in V\setminus A} a_j(A)x_j \geq b(A), & \forall A \subseteq V, \\ & x_j \geq 0, & \forall j \in V, \end{array} \tag{3}$$

where

$$\begin{array}{rcll} b(A) & = & \max\{0, b - \sum_{j\in A} a_j\}, & \forall A \subseteq V, \\ a_j(A) & = & \min\{a_j, b(A)\}, & \forall A \subseteq V, \forall j \in V\setminus A. \end{array} \tag{4}$$

It is known that any feasible 0-1 solution of (3) is feasible for (1).

Similarly, we use the following LP relaxation of MKPFG (2):

$$\begin{array}{lll} \min & \displaystyle\sum_{j\in V} c_j x_j & \\ \text{s.t.} & \displaystyle\sum_{j\in V\setminus A} a_j(A)x_j \geq b(A), & \forall A \subseteq V, \\ & x_i + x_j \geq 1, & \forall \{i,j\} \in E, \\ & x_j \geq 0, & \forall j \in V. \end{array} \tag{5}$$

The dual of (5) is represented as

$$\begin{array}{lll} \max & \displaystyle\sum_{A\subseteq V} b(A)y(A) + \sum_{\{i,j\}\in E} z_{\{i,j\}} & \\ \text{s.t.} & \displaystyle\sum_{A\subseteq V: j\notin A} a_j(A)y(A) + \sum_{k:\{j,k\}\in E} z_{\{j,k\}} \le c_j, & \forall j \in V, \\ & y(A) \ge 0, & \forall A \subseteq V, \\ & z_{\{i,j\}} \ge 0, & \forall \{i,j\} \in E, \end{array} \tag{6}$$

where each dual variable $y(A)$ corresponds to the inequality $\sum_{j\in V\setminus A} a_j(A)x_j \ge b(A)$ and $z_{\{i,j\}}$ corresponds to the forcing constraint for the edge $\{i,j\}$.

Now we introduce a well-known result for a primal-dual pair of linear programming [4].

Lemma 2.1. *Let $\bar{\boldsymbol{x}}$ and $\bar{\boldsymbol{y}}$ be feasible solutions for the following primal and dual linear programming problems:*

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

If the conditions

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

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 2.1 to the problems (5) and (6), we have the following lemma and corollary.

Lemma 2.2. *Let $\boldsymbol{x}$ and $(\boldsymbol{y},\boldsymbol{z})$ be feasible solutions for (5) and (6), respectively. If these solutions satisfy*

$$\begin{array}{ll} \textit{(a):} & \forall j \in V, x_j > 0 \Rightarrow \sum_{A\subseteq V: j\notin A} a_j(A)y(A) + \sum_{k:\{j,k\}\in E} z_{\{j,k\}} = c_j, \\ \textit{(b-1):} & \forall \{i,j\} \in E, z_{\{i,j\}} > 0 \Rightarrow x_i + x_j \le 2, \\ \textit{(b-2):} & \forall A \subseteq V, y(A) > 0 \Rightarrow \sum_{j\in V\setminus A} a_j(A)x_j \le 2b(A), \end{array} \tag{7}$$

then $\boldsymbol{x}$ is a solution within a factor of 2 of the optimal solution of (5).

Corollary 2.1. *Let $\boldsymbol{x}$ be a feasible 0-1 solution of (5) and $(\boldsymbol{y}, \boldsymbol{z})$ be a feasible solution of (6). If these solutions satisfy (7), $\boldsymbol{x}$ is a solution within a factor of 2 of the optimal solution of (2).*

We propose a polynomial algorithm for calculating $\boldsymbol{x}$ and $(\boldsymbol{y}, \boldsymbol{z})$ which satisfy the conditions in Corollary 2.1. The algorithm generates a sequence of points $\boldsymbol{x}$ and $(\boldsymbol{y}, \boldsymbol{z})$ which always satisfy the following conditions:

- $\boldsymbol{x} \in \{0,1\}^n$.
- $(\boldsymbol{y}, \boldsymbol{z})$ is feasible for (6).
- $\boldsymbol{x}$ and $(\boldsymbol{y}, \boldsymbol{z})$ satisfy (7).

All the forcing constraints in (5) are satisfied in Step 1 and the other constraints in (5) are met in Step 2. For the points $\boldsymbol{x}$ and $(\boldsymbol{y}, \boldsymbol{z})$ at each step, we use symbols $S = \{j \in V \mid x_j = 1\}$, $\bar{b} = b - \sum_{j \in V} a_j x_j$, and $\bar{c_j} = c_j - (\sum_{A \subseteq V : j \notin A} a_j(A) y(A) + \sum_{k:\{j,k\} \in E} z_{\{j,k\}})$ for $j \in V$. $\bar{E} \subseteq E$ denotes a set of unchecked edges in Step 1. Now we state our algorithm.

Algorithm 1

Step 0: Let $\boldsymbol{x} = \boldsymbol{0}$ and $(\boldsymbol{y}, \boldsymbol{z}) = (\boldsymbol{0}, \boldsymbol{0})$ be initial solutions. Set $S = \emptyset$, $V' = \{j \in V \mid a_j > 0\}$, $\bar{E} = E$, $\bar{b} = b$, and $\bar{c}_j = c_j$ for $j \in V$.

Step 1: If $\bar{E} = \emptyset$, then go to Step 2. Otherwise choose an edge $e = \{i, j\} \in \bar{E}$. If $x_i + x_j \geq 1$, then update $\bar{E} = \bar{E} \backslash \{e\}$ and go back to the top of Step 1. If $x_i + x_j = 0$, increase $z_{\{i,j\}}$ as much as possible while maintaining feasibility for (6). Since $z_{\{i,j\}}$ appears in only two constraints of (6) corresponding to the vertices i and j, we see that

$$z_{\{i,j\}} = \bar{c}_s \quad \text{for} \quad s = \arg\min\{\bar{c}_i,\ \bar{c}_j\}.$$

Update $x_s = 1, S = S \cup \{s\}, \bar{E} = \bar{E} \backslash \{e\}, \bar{c}_i = \bar{c}_i - z_{\{i,j\}}, \bar{c}_j = \bar{c}_j - z_{\{i,j\}}$, and $\bar{b} = \bar{b} - a_s$. Go back to the top of Step 1.

Step 2: If $\bar{b} \leq 0$, then output $\tilde{\boldsymbol{x}} = \boldsymbol{x}$ and $(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{z}}) = (\boldsymbol{y}, \boldsymbol{z})$ and stop. Otherwise calculate $a_j(S)$ for all $j \in V' \backslash S$ by (4), where $b(S) = \bar{b}$. Increase $y(S)$ as much as possible while maintaining feasibility for (6). Since $y(S)$

appears in constraints of (6) for $j \in V' \backslash S$ so that $a_j(S) > 0$, we see that

$$y(S) = \frac{\bar{c}_s}{a_s(S)} \quad \text{for} \quad s = \arg\min_{j \in V' \backslash S} \left\{ \frac{\bar{c}_j}{a_j(S)} \right\}.$$

Update $x_s = 1$, $S = S \cup \{s\}$, $\bar{c}_j = \bar{c}_j - a_j(S)y(S)$ for any $j \in V' \backslash S$, and $\bar{b} = \bar{b} - a_s$. Go back to the top of Step 2.

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

Lemma 2.3. *$\tilde{\boldsymbol{x}}$ is a feasible 0-1 solution of (5) and $(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{z}})$ is a feasible solution of (6).*

Proof. By the assumption that MKPFG (2) is feasible, $\boldsymbol{x} = (1, \cdots, 1)$ is feasible for the LP relaxation problem (5). Algorithm 1 starts form $\boldsymbol{x} = \boldsymbol{0}$ and updates an variable x_j from 0 to 1 at each iteration until satisfying all the constraints in (5). Hence $\tilde{\boldsymbol{x}}$ is a feasible 0-1 solution of (5).

Algorithm 1 starts from the dual feasible solution $(\boldsymbol{y}, \boldsymbol{z}) = (\boldsymbol{0}, \boldsymbol{0})$ and maintains dual feasibility throughout the algorithm. Hence $(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{z}})$ is feasible for (6). □

Lemma 2.4. *$\tilde{\boldsymbol{x}}$ and $(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{z}})$ satisfy (7).*

Proof. Since $\boldsymbol{x} = \boldsymbol{0}$ at the beginning and the algorithm sets $x_j = 1$ only if the j-th constraint in (6) becomes tight, (a) of (7) is satisfied. (b-1) of (7) follows from $\tilde{\boldsymbol{x}} \in \{0, 1\}^n$. Thus it suffices to show that (b-2) holds. We consider two cases, whether or not the algorithm stops at the first iteration of Step 2.

If the algorithm stops at the first iteration of Step 2, we obtain a primal feasible solution in Step 1. Then (b-2) holds since $\tilde{y}(A) = 0$ for any $A \subseteq V$. Otherwise, the algorithm does not obtain a primal feasible solution in Step 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, $\tilde{y}(A) > 0$ implies

$$A \subseteq \tilde{S} \backslash \{\ell\}. \tag{8}$$

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

$$\sum_{j \in \tilde{S} \backslash \{\ell\}} a_j < b. \tag{9}$$

By (8) and (9), we observe that

$$\sum_{j\in(\tilde{S}\setminus\{\ell\})\setminus A} a_j(A) \le \sum_{j\in(\tilde{S}\setminus\{\ell\})\setminus A} a_j = \sum_{j\in\tilde{S}\setminus\{\ell\}} a_j - \sum_{j\in A} a_j < b - \sum_{j\in A} a_j \le b(A),$$

where the first and last inequality follows from the definitions (4) of $a_j(A)$ and $b(A)$. Thus, we have that

$$\sum_{j\in V\setminus A} a_j(A)\tilde{x}_j = \sum_{j\in\tilde{S}\setminus A} a_j(A) = \sum_{j\in(\tilde{S}\setminus\{\ell\})\setminus A} a_j(A) + a_\ell(A) < 2b(A),$$

where the last inequality follows from $a_\ell(A) \le b(A)$. □

Lemma 2.5. *The running time of Algorithm 1 is $O(|E| + |V'|^2)$, where $V' = \{j \in V \mid a_j > 0\}$.*

Proof. The running time of one iteration of Step 1 is $O(1)$ and the number of iterations in Step 1 is at most $|E|$. The running time of one iteration of Step 2 is $O(|V'|)$ and the number of iterations in Step 2 is at most $|V'|$. Therefore the running time of the algorithm is $O(|E| + |V'|^2)$. □

The following result follows from Corollary 2.1 and Lemmas 2.3, 2.4, and 2.5.

Theorem 2.1. *Algorithm 1 is a 2-approximation algorithm for MKPFG (2).*

3 Generalization to a Covering Integer Program with 0-1 Variables

In this section, we generalize Algorithm 1 to a covering integer program with 0-1 variables (CIP), which is represented as

$$\text{CIP}\left|\begin{array}{lll} \min & \displaystyle\sum_{j\in N} c_j x_j & \\ \text{s.t.} & \displaystyle\sum_{j\in N} a_{ij} x_j \ge b_i, & \forall i \in M = \{1,\cdots,m\}, \\ & x_j \in \{0,1\}, & \forall j \in N = \{1,\cdots,n\}, \end{array}\right. \tag{10}$$

where b_i, a_{ij}, and c_j ($i \in M$, $j \in N$) are nonnegative. Assume that $\sum_{j\in N} a_{ij} \ge b_i$ 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 $\Delta_2 \geq 2$. There are some Δ_1-approximation algorithms for CIP, see Koufogiannakis and Young [8] and references therein. We propose a Δ_2-approximation algorithm. The minimum knapsack problem with a forcing graph (2) is a special case of CIP for which $\Delta_2 = 2$.

We introduce a LP relaxation problem of CIP constructed by Carr et al. [3]. The relaxation problem is represented as

$$\begin{array}{lll} \min & \displaystyle\sum_{j\in N} c_j x_j & \\ \text{s.t.} & \displaystyle\sum_{j\in N\setminus A} a_{ij}(A)x_j \geq b_i(A), & \forall A \subseteq N,\ \forall i \in M, \\ & x_j \geq 0, & \forall j \in N, \end{array} \tag{11}$$

where

$$\begin{array}{rcl} b_i(A) & = & \max\{0, b_i - \sum_{j\in A} a_{ij}\},\ \forall i \in M, \forall A \subseteq N, \\ a_{ij}(A) & = & \min\{a_{ij}, b_i(A)\},\ \forall i \in M, \forall A \subseteq N, \forall j \in N\setminus A. \end{array} \tag{12}$$

Carr et al. [3] show that any feasible 0-1 solution of (11) is feasible for (10). The dual problem of (11) can be stated as

$$\begin{array}{lll} \max & \displaystyle\sum_{i\in M}\sum_{A\subseteq N} b_i(A)y_i(A) & \\ \text{s.t.} & \displaystyle\sum_{i\in M}\sum_{A\subseteq N: j\notin A} a_{ij}(A)y_i(A) \leq c_j, & \forall j \in N, \\ & y_i(A) \geq 0, & \forall A \subseteq N,\ \forall i \in M. \end{array} \tag{13}$$

By applying Lemma 2.1 to the LP problems (11) and (13), we have the following result.

Lemma 3.1. *Let $\boldsymbol{x}$ be a feasible 0-1 solution of (11) and $\boldsymbol{y}$ be a feasible solution of (13). If these solutions satisfy*

$$\begin{array}{ll} \textit{(a):} & \forall j \in N, x_j > 0 \Rightarrow \sum_{i\in M}\sum_{A\subseteq N: j\notin A} a_{ij}(A)y_i(A) = c_j, \\ \textit{(b):} & \forall i \in M,\ \forall A \subseteq N, y_i(A) > 0 \Rightarrow \sum_{j\in N\setminus A} a_{ij}(A)x_j \leq \Delta_2 b(A), \end{array} \tag{14}$$

then $\boldsymbol{x}$ is a solution within a factor of Δ_2 of the optimal solution of (10).

Our algorithm is presented in Algorithm 2 below. The goal is to find $\boldsymbol{x}$ and $\boldsymbol{y}$ which satisfy the conditions in Lemma 3.1. The algorithm generates a sequence of points $\boldsymbol{x}$ and $\boldsymbol{y}$. Throughout the algorithm, the conditions $\boldsymbol{x} \in \{0,1\}^n$, constraints in (13), and (14) are satisfied. The constraints in (11) are satisfied at Step 2. In Algorithm 2, 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 $\bar{c}_j = c_j - \sum_{i\in M} \sum_{A\subseteq N: j\notin N} a_{ij}(A) y_i(A)$ for $j \in N$.

Algorithm 2

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 $j \in N_i' \backslash S$ by (12). Increase $y_i(S)$ while maintaining dual feasibility until at least one constraint $s \in N_i' \backslash S$ is tight. Namely set

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

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

In the same way as the proof of Lemma 2.3, we have the following result for the outputs $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$ of Algorithm 2.

Lemma 3.2. *$\tilde{\boldsymbol{x}}$ is a 0-1 feasible solution of (11) and $\tilde{\boldsymbol{y}}$ is a feasible solution of (13).*

The next lemma is similarly proved as Lemma 2.4.

Lemma 3.3. *$\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{y}}$ satisfy (14).*

Proof. All the conditions in (a) of (14) 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 (12) and the i-th constraint has Δ_i non-zero coefficients. Then, we consider the case of $i = 1$. In the similar way of the proof in Lemma 2.4, for any subset $A \subseteq N$ such that $\tilde{y}_1(A) > 0$, we have

$$\sum_{j \in N \setminus A} a_{1j}(A)\tilde{x}_j \leq 2b_1(A) \leq \Delta_2 b_1(A).$$

□

Lemma 3.4. *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 3.1. *Algorithm 2 is a Δ_2-approximation algorithm for CIP (10).*

4 Conclusion

We proposed a 2-approximation algorithm for the minimum knapsack problem with a forcing graph. The approximability of the algorithm is the same as that of the algorithms for the minimum knapsack problem presented by Carnes and Shmoys [2] and for the minimum vertex cover problem by Bar-Yehuda and Even [1]. Then we generalize the algorithm to the covering integer program with 0-1 variables and proposed a Δ_2-approximation algorithm, where Δ_2 is the second largerst 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] R. Bar-Yehuda and S. Even: A linear-time approximation algorithm for the weighted vertex cover problem, *Journal of Algorithms*, **2** (1981), 198-203.

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

[3] R. D. Carr, L. Fleischer, V. J. Leung, 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.

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

[5] I. Dinur and S. Safra: On the hardness of approximating minimum vertex cover, *Annals of Mathematics*, **162** (2005), 439-485.

[6] M. Hifi and N. Otmani: An algorithm for the disjunctively constrained knapsack problem. *International Journal of Operational Research*, **13** (2012), 22-43.

[7] M. Hifi, S.Saleh, L.Wu: A fast large neighborhood search for disjunctively constrained knapsack problems, *Proceedings of 3rd International Symposium on Combinatorial Optimization, ISCO 2014, volume 8596 of Lecture Notes in Computer Science*, (Springer, 2014), 396-407.

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

[9] S. Khot and O. Regev: Vertex cover might be hard to approximate to within 2-ϵ, *Journal of Computer and System Sciences*, **74** (2008), 335-349.

[10] U. Pferschy and J. Schauer: The knapsack problem with conflict graphs, *Journal of Graph Algorithms and Applications*, **13** (2009), 233-249.

[11] U. Pferschy and J. Schauer: Approximation of knapsack problems with conflict and forcing graphs, *Journal of Combinatorial Optimization* (2016).

[12] T. Yamada, S. Kataoka, and K. Watanabe: Heuristic and exact algorithms for the disjunctively constrained knapsack problem, *Information Processing Society of Japan Journal*, **43** (2002), 2864-2870.