# Partition 制約付き最小化ナップサック問題に対する 3-近似アルゴリズム

電気通信大学　情報理工学部 情報・通信工学科 情報数理工学コース　1411070　木村 優貴　村松 正和研究室

## 1　はじめに

本研究では NP 困難な問題に対するアプローチの中でも代表的手法である主双対法を用いて近似アルゴリズムの設計を行った．

近似アルゴリズムとは，最適値に十分に近い目的関数値が得られるような解を求めるアルゴリズムである．近似アルゴリズムの中でも最適化問題の全ての入力に対して最適値の $\alpha$ 倍以内の値を持つ解を返すアルゴリズムをその最適化問題に対する $\alpha$-近似アルゴリズムと呼ぶ [1, 2]．この時の $\alpha$ を近似率と呼ぶ．最小化問題に対しては $\alpha > 1$ であり，最大化問題に対しては $\alpha < 1$ となる．

最小化ナップサック問題は，品物をナップサックに価値の総和をある需要以上に保ちつつ，詰め込む品物の重さの総和を最小化することを目的とする問題である [1]．この問題は NP 困難な問題であることが知られており [3]，最適解の導出や近似アルゴリズムの設計が広く行われている．

本研究は，この最小化ナップサック問題に対し新たな制約を加え，拡張した問題に対して制度保証のある近似アルゴリズムの提案を行う．

## 2　Partition 制約付き最小化ナップサック問題

集合 $\mathcal{P}$ を

$$\begin{aligned} &\mathcal{P} = \{P_1, \dots, P_m\} \\ &\mathcal{P} \subseteq 2^V, |P_j| \geq 2 \quad \forall j \in \{1, \dots, m\} \\ &P_i \cap P_j = \emptyset \quad \forall i, j \in \{1, \dots, m\} (i \neq j) \end{aligned}$$

と定義し，以下の問題を考える:

$$\min \sum_{j \in V} c_j x_j \tag{2.1a}$$

$$\text{s.t.} \sum_{j \in V} a_j x_j \geq b \tag{2.1b}$$

$$\sum_{j \in P} x_j \geq 1 \qquad \forall P \in \mathcal{P} \tag{2.1c}$$

$$x_j \in \{0, 1\} \qquad \forall j \in V \tag{2.1d}$$

制約 (2.1c) は任意の $P \in \mathcal{P}$ に含まれる品物のうち少なくとも一つは選択することを意味する．この制約を Partition 制約と呼ぶことにし，問題 (2.1) を Partition 制約付き最小化ナップサック問題 (Minumum Knapsack Problem with Partition Constraints:MKPPC) と呼ぶことにする．

MKPPC の近似アルゴリズムを構築するために 2 つの部分問題

$$\begin{aligned} \min \quad & \sum_{j \in V} c_j x_j \\ \text{s.t.} \quad & \sum_{j \in P} x_j \geq 1 \quad \forall P \in \mathcal{P} \\ & x_j \in \{0, 1\} \quad \forall j \in V \end{aligned} \tag{2.2}$$

と

$$\begin{aligned} \min \quad & \sum_{j \in V} c_j x_j \\ \text{s.t.} \quad & \sum_{j \in V} a_j x_j \geq b \\ & x_j \in \{0, 1\} \quad \forall j \in V \end{aligned} \tag{2.3}$$

を考える．

問題 (2.2) は Partition 制約 (2.1c) を持った問題であり，厳密解法を考えることができる．また，問題 (2.3) はナップサック制約 (2.1b) を持った問題であり，2-近似アルゴリズムが存在する [1]．

問題 (2.2) に対する厳密解法を示す．

**Algorithm 1**

**Input:** $V$: 品物の集合，$\mathcal{P}$: 品物を分割した集合，$\boldsymbol{c}$: 品物の重さのベクトル

**Output:** $\tilde{\boldsymbol{x}}$: 問題 (2.2) の解

**Step0:** $\boldsymbol{x} = \boldsymbol{0}$ を初期解として与える．$\bar{\mathcal{P}} = \mathcal{P}$ を初期値として与える．

**Step1:** $P \in \bar{\mathcal{P}}$ を 1 つ選択し，$\bar{\mathcal{P}} = \bar{\mathcal{P}} \setminus \{P\}$ とする．

**Step2:** $s = \arg\min_{j \in P} c_j$ を計算し，$x_s = 1$ とする．

**Step3:** $\bar{\mathcal{P}} = \emptyset$ ならば $\tilde{\boldsymbol{x}} = \boldsymbol{x}$ とし，アルゴリズムを終了する．そうでないならば Step1 へ戻る．

更に，MKPPC に対する 3-近似アルゴリズムを提案する．

**Algorithm 2**

**Input:** $V$: 品物の集合，$\mathcal{P}$: 品物を分割した集合，$\boldsymbol{a}$: 品物の価値のベクトル，$b$: ナップサック内に保ちたい価値，$\boldsymbol{c}$: 品物の重さのベクトル

**Output:** $\tilde{S}$: 選択した品物の集合

**Step0:** $S_1 = S_2 = \emptyset$ とする．

**Step1:** 問題 (2.2) を満たすように Algorithm 1 を適用．$S_1 = \{j \mid x_j = 1\}$ とする．また，$\bar{b} = b - \sum_{j \in S_1} a_j$ とする．

※ Step1 の結果がナップサック制約を満たしているなら Step3 へ

**Step2:** 残った品物 $V' = V \setminus S_1$ と $\bar{b}$ で問題 (2.3) を構成．最小化ナップサック問題に対する 2-近似アルゴリズムを適用．$S_2 = \{j \mid x_j = 1\}$ とする．

**Step3:** $\tilde{S} = S_1 \cup S_2$ とし，アルゴリズムを停止する．

**定理 2.1** Algorithm 2 は MKPPC に対する 3-近似アルゴリズムである．

**証明**　問題 (2.1) の最適値を $\theta$，問題 (2.2) の最適値を $\theta_1$，問題 (2.3) の最適値を $\theta_2$ とおく．問題 (2.1) は問題 (2.2) にナップサック制約を加えたものなので，$\theta_1 \leq \theta$．同様にして $\theta_2 \leq \theta$ なので，

$$\begin{aligned}\sum_{j\in\tilde{S}} c_j &= \sum_{j\in S_1} c_j + \sum_{j\in S_2} c_j \\ &\leq \theta_1 + 2\theta_2 \\ &\leq \theta + 2\theta \\ &= 3\theta\end{aligned}$$

となる．したがって $\sum_{j\in\tilde{S}} c_j \leq 3\theta$ を満たす．　□

## 2.1　数値実験

提案アルゴリズムを C++で実装したものと，数理計画ソルバー gurobi を比較する数値実験をそれぞれ MKPPC と MKPFH に対して行なった．実験環境は次の表 1 の通りである．

表 1: 実行環境

| CPU | | OS |
|---|---|---|
| 4GHz Intel Core i7 | | macOS 10.12.5 |
| 開発言語 | ソルバー | メモリ |
| C++14 | gurobi7.0.2 | 16GB |

また，全実験に共通する各入力パラメータの値は

$$\begin{aligned} a_j &\in [1, 20] && (\forall j \in V) \\ b &\in \left[\frac{4}{5}\sum_{j\in V} a_j, \sum_{j\in V} a_j\right] \\ c_j &\in [1, 20] && (\forall j \in V)\end{aligned}$$

とした．MKPPC における $\mathcal{P}$ は乱数で $[2, \frac{|V|}{2}]$ から選択した．$|V| = 1000$ から $|V| = 5000$ まで，頂点数を 100 ずつ増加させ，各頂点数について問題を 20 問生成し，gurobi と Algorithm 2 との実行時間の平均を取り，その比較を行った．実験結果を図 1 にまとめた．

図 1 より，頂点数が増加するにつれて計算時間が二次関数的に増加していることがわかる．gurobi との実行時間の比較としては，$|V| = 3500$ までは Algorithm 2 の方が早く終了していることがわかる．

また，gurobi により得られた解を厳密解とし，Algorithm 2 により得られる近似解の近似率の確認を生成した 820 問に対して行った．その結果を図 2 にまとめた．

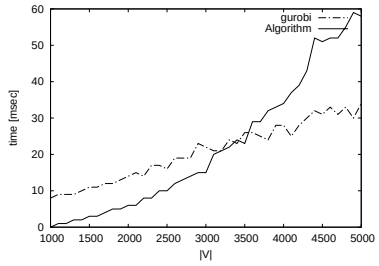


図 1: 実行時間の比較

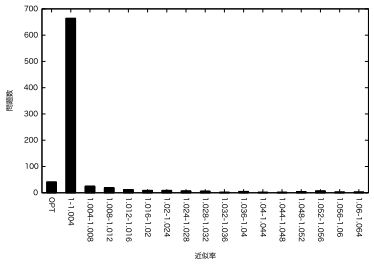


図 2: MKPPC に対する近似率

実験結果としては 820 問の問題全てが最適値の 1.064 倍以内の目的関数値が得られていることがわかる．また，多くの問題について最適値の 1.004 倍以内の目的関数値が得られており，高い精度で近似解が得られていることがわかる．

# 3　Forcing Hypergraph 付き最小化ナップサック問題

Forcing Hypergraph $H = (V, \mathcal{E})$ を以下のように定義する: $V = \{1, \ldots, n\}$，任意の $E \in \mathcal{E}$ について $|E| \geq 2$．

問題 (2.1) の制約 (2.1c) を

$$\sum_{j\in E} x_j \geq 1 \qquad \forall E \in \mathcal{E}$$

に変更した問題を考える．この制約は任意の $E \in \mathcal{E}$ に含まれる頂点のうちどれか一つは必ず選択しなければならないということを意味する．この制約を Forcing 制約と呼ぶことにする．更にこの問題を Forcing Hypergraph 付き最小化ナップサック問題 (Minimum Knapsack Problem with Forcing Hypergraph:MKPFH) と呼ぶことにする．

任意の 2 つの要素 $E_1, E_2 \in \mathcal{E}(E_1 \neq E_2)$ が互いに素である時，MKPFH は MKPPC に等しい．

MKPFH に対する $k-$ 近似アルゴリズムの提案する．ただし，$k = \max_{E\in\mathcal{E}} |E|$ とする．

## 参考文献


[1] David P. Williamson，David B. Shmoys 著，浅野孝夫 訳: 近似アルゴリズムデザイン，共立出版 (2015)

[2] V. V. ヴァジラーニ 著，浅野孝夫 訳: 近似アルゴリズム，シュプリンガージャパン (2002)

[3] Michael R. Garey，David S. Jphnson: COMPUTERS AND INTRACTABILITY A Guide to the Theory of NP-Completeness，Freedman (1979)