数理解析研究所講究録
第1802巻 2012年 71-77

動的計画法を用いた多次元ナップサック問題の変数縮小法

防衛大学校・理工学研究科 山下 美里 (Misato Yamashita)
Graduate School of Science and Engineering,
National Defense Academy
防衛大学校・情報工学科 片岡 靖詞 (Seiji Kataoka)
Department of Computer Science,
National Defense Academy

1 はじめに

ナップサック問題 (KP: Knapsack Problem) とは，価値と重量を持ったアイテムを重量制限のあるナップサックに入れる際，どのような入れ方をすれば総価値を最大にすることができるのかという問題である．KPには様々なバリエーションがあるが，ここでは，ナップサック制約が複数ある**多次元ナップサック問題** (MDKP: Multidimensional Knapsack Problem) を扱う．これは，一般の0-1計画問題と同一視される傾向があり，古典的なMartello-Toth[4]には索引にさえ登場していない．また比較的新しいKellerer-Pferschy-Pisinger[3]には1つの章が割かれているものの，詳細な議論は2次元に限定した場合だけである．

また，KPへのアプローチのひとつとして**動的計画法** (DP: Dynamic Programming) が知られている．これは，限られたアイテム数における最適解の情報をもとに，アイテム数を徐々に増やしながら，最終的に全アイテムの最適解を得る方法である．メモリを浪費する欠点はあるものの，多くの有用な情報を生成している．そして，最終的に最適解に至る解は，途中段階でもトップの僅かな状態から派生していることが多い．

本研究では，この性質を利用し，高速に安定した精度の近似解を得る．さらに，提案する手法を変数の縮小に活用することも考える．

2 多次元ナップサック問題

次元数（制約数）をmとすると，MDKPは以下のように定式化できる．

$$\text{(MDKP)} \quad \max \quad \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} w_{ij} x_j \leq c_i \quad i = 1, \ldots, m \tag{2}$$

$$x_j \in \{0, 1\} \tag{3}$$

MDKPの僅かな特徴としては，w_{ij}が非負であることと，制約行列要素のほとんどが非ゼロであり，扱いやすくなる特殊構造を持っていないことが挙げられる．このことが，よりMDKPを難しくしている．

通常のKPが$\mathcal{NP}$-困難であることから，MDKPが$\mathcal{NP}$-困難であることも自明である．さらに通常のKPが完全多項式オーダーの近似解法が存在するのに対し，MDKPでは，2次元でさえも完全多項式の近似解法が（$\mathcal{P} \neq \mathcal{NP}$の仮定のもとでは）存在しないことが証明されている．また，ヒューリスティック解法においても，精度が安定したアルゴリズムが少なく，適当な下界値を得ることも困難を要する問題であることが知られている．

3 多次元ナップサック問題の動的計画法

$f(k, \boldsymbol{b})$ ($\boldsymbol{b} = (b_1, \ldots, b_m)$) を$k$個めまでのアイテムで，重量制限が$b_i$のときの最適値とする．このときMDKPに対するDPの漸化式は以下のように記述できる．

$$f(k, \boldsymbol{b}) = \begin{cases} f(k-1, \boldsymbol{b}) \\ \qquad \text{if } b_i < w_{ik} \text{ for some } i \\ \max \begin{cases} f(k-1, \boldsymbol{b}) \\ f(k-1, b_1 - w_{1k}, b_2 - w_{2k}, \cdots, b_m - w_{mk}) + p_k \end{cases} \\ \qquad \text{if } b_i \geq w_{ik} \text{ for all } i \end{cases}$$

考え方は通常のKP（m=1）に準じており難しくない．しかし，このやり方では$n \cdot c_1 \cdot \cdots \cdot c_m$のメモリを消費してしまうので実際的ではない．

そこで，状態の変化するところだけに注目したリスト表記をMDKPに適用する．これは，以下のように記述できる．ここでPは累加価値，$\boldsymbol{W}$は累加重量ベクトル，$\boldsymbol{w}_k$は制約行列の第k列ベクトルである．

Algorithm DPforMDKP

```
1: L_M := {(0, 0)}
2: for k := 1 to n do
3:     L_0 := L_M
4:     L_1 := ∅
5:     for (P, W) ∈ L_0
6:         if W + w_k ≤ c    then L_1 := L_1 ∪ (P + p_k, W + w_k)
7:     end for
8:     L_M := L_0 ∪ L_1
9: end for
```

通常のKPでは，8行目において無駄な状態を削除することができるため，リストのサイズを抑えることが可能である．しかしながら，MDKPではmが大きくなると優越性判定はほとんど成立せず，事実上すべての状態を残すことになる．したがって，8行目におけるL_Mのサイズは2^kの勢いで大きくなり，アイテム数20くらいで計算時間を圧迫する．

4 近視眼的動的計画法

通常のKPでは，あらかじめアイテムを価値と重量とで効率の高い順に並べておくことを仮定していることが多い．DPはアイテムの並んでいる順序には関係なく

適用できるが，効率の良い順に並べると扱いやすくなることが知られている．

また，DP の特徴として，考慮したアイテムの部分問題では最適な解が得られており，その延長線上に最終的な最適解があるはずである．その最適解に至る状態は，DP の途中であっても，比較的総価値の高い辺りに位置しており，後半に大逆転ということはあまりないことを経験している．

まず通常の KP に DP を適用したときに現れる状態を観察し，そこで得られた性質を **Algorithm** DPforMDKP に適用する．

4.1 通常のナップサック問題の場合

通常の KP に DP を適用したときに現れる状態を観察するため，以下の例題を考える．

- アイテム数 n=6，重量制限 c=190 とする．それぞれの価値と重量が

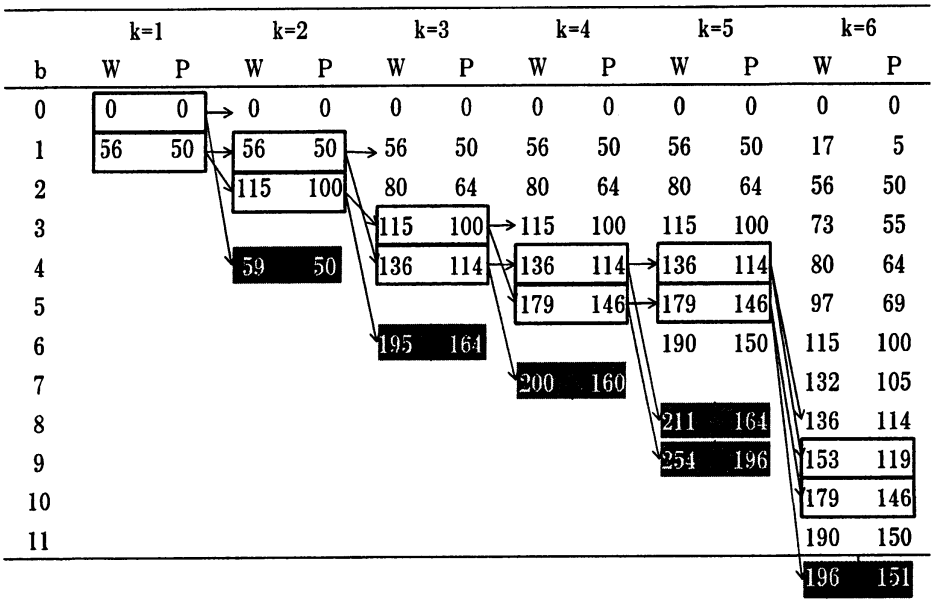
$$\begin{aligned} p_j &= (50, 50, 64, 46, 50, 5) \\ w_j &= (56, 59, 80, 64, 75, 17) \end{aligned}$$

である DP を考える．この例題ではアイテムを効率の高い順に並べている．

	k=1		k=2		k=3		k=4		k=5		k=6	
b	W	P	W	P	W	P	W	P	W	P	W	P
0	0	0	0	0	0	0	0	0	0	0	0	0
1	56	50	56	50	56	50	56	50	56	50	17	5
2			115	100	80	64	80	64	80	64	56	50
3					115	100	115	100	115	100	73	55
4					136	114	136	114	136	114	80	64
5							179	146	179	146	97	69
6									190	150	115	100
7											132	105
8											136	114
9											153	119
10											179	146
11											190	150

図 1: 通常の KP に DP を適用したときに現れる状態

図 1 において，右下の (190,150) が最適な状態であり，矢印が最適な状態に至る路である．結果論ではあるが，この例題では各段階で累加価値の大きい方から 3 つのみ見ていれば最適解に至っていることがわかる．また，(0,0) という状態あるいは周辺の累加価値の大きくない状態は，後半になればほとんど無用なものと言っても差し支えない．累加価値の大きい方から 3 つというのは，後からわかることであるが，仮に大きい方から 2 つのみに着目したときは，図 2 に示したように状態の変化を観察することができる．



b	k=1 W	k=1 P	k=2 W	k=2 P	k=3 W	k=3 P	k=4 W	k=4 P	k=5 W	k=5 P	k=6 W	k=6 P
0	0	0	0	0	0	0	0	0	0	0	0	0
1	56	50	56	50	56	50	56	50	56	50	17	5
2			115	100	80	64	80	64	80	64	56	50
3					115	100	115	100	115	100	73	55
4			59	50	136	114	136	114	136	114	80	64
5							179	146	179	146	97	69
6					195	164			190	150	115	100
7							200	160			132	105
8									211	164	136	114
9									254	196	153	119
10											179	146
11											190	150
											196	151

図 2: 累加価値の大きい方から 2 つに着目した場合

最適な状態には至らないものの，2 番目に良い解である近似解 146 に至ることがわかる．このことから，アイテムが適切な順序に並んでいれば，リストのごくわずかな部分だけに注目しているだけで，最適解あるいはかなり高精度の近似解が得られることがわかる．

4.2 多次元ナップサック問題の場合

一般的な KP の例題から，DP において最終的に最適解となる状態は，途中の段階でもリストの中で，累加価値の高い，ごくわずかな位置にあることが観察される．この性質を，**Algorithm** DPforMDKP にも適用することを考える．**Algorithm** DPforMDKP の 8 行目は，

$$L_M := L_0 \cup L_1$$

であった．このとき，各リストを P の大きい順に作り，L_0 と L_1 をマージするときに

$$L_M := \{L_0 \cup L_1 | \text{総価値の大きい方から有限個 } (S \text{ 個})\}$$

という DP を考える．これは途中の段階でもリストの中からトップの高々 S 個の状態しか見ていないことを意味する．このアイデアを導入した DP を**近視眼的動的計画法** (MyoDP) と呼ぶことにする．ここで考慮しなければならないのは，

① アイテムをどのような順番に並べ換えるのか

② リストに残す有限個の状態数 (S 個) をいくつにするのか

の 2 点である．それを評価するために，次に実験を行う．

アイテム数を 100 に限定し，制約数 m を 20,40,80，最大リストサイズ (②) を 10,100, 1000,10000 にして実験を行った．アイテムの順序付け (①) としてランダム (rnd) の他に，できるだけ簡単に得られる情報を用いるため，最初に線形計画問題 (LP: Linear

Programming) を解き，1になるもの，0になるもの，小数 (f) になるものに分類し，1f0, 10f, 0f1, 01f, f10, f01 と並べ換えた6種；効率 $p_j/\sum_i w_{ij}$ の大きい順 (nio) と小さい順 (ndo)；そして，LP の双対最適値 π_i を用いて，$p_j/\sum_i(\pi_i w_{ij})$ の大きい順 (pii) と小さい順 (pid) について比較した．最適値は CPLEX で求め，得られた近似値の精度を% で示している．DP の計算時間は，ほとんどが1秒に満たず計算することができたので詳細は省く．

アイテムの順序は通常効率のよい順 (nio) がよいとされているが，図3の結果を見ると，事前に LP を解き 1f0 に並べ換えたものが一番よかった．LP において0になるものを後回しにし，前半でよい解を固めてしまうのが得策であることがわかる．図4は，並べ換えを一番精度のよかった 1f0 として，最大リストサイズに注目したときの精度を表したものである．$S = 10$ というのは，かなり極端な実験だが，それでも 99.5% を超える精度の解が得られたことは特筆すべきことである．

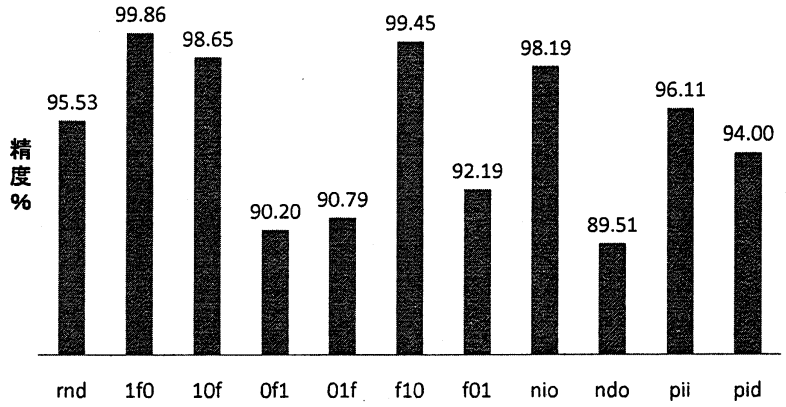


図 3: 並べ換えによる精度（1000）

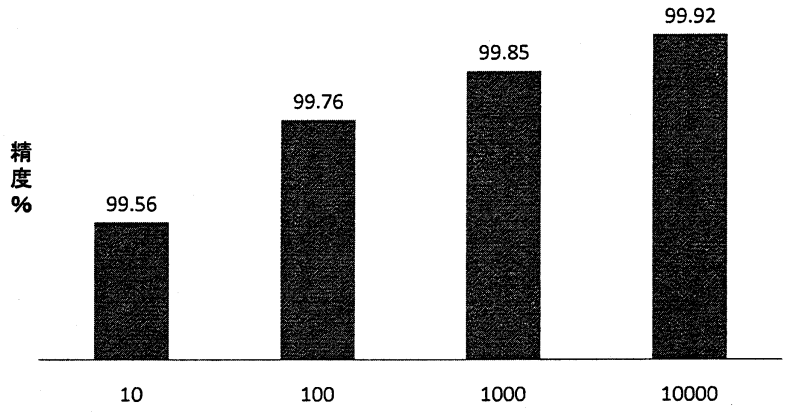


図 4: 最大リストサイズによる精度（1f0）

次に，MyoDP のパラメータを，並べ換え 1f0，最大リストサイズ 1000 として，Chu-Beasley[2] のベンチマーク問題に適用した実験結果を示す．表1の Problem において，n はアイテム数，m は制約数，α は総重量に対する重量制限の割合を表す．比較に，CPLEX と Balev ら [1] の近似解法を用いる．Balev らの方法は，まず LP で0と1になったものを0と1に固定する．そして，残った小数の部分を CPLEX で厳

密に解く方法である．結果のCPLEX timeにある∞は，実行時間が1時間以上かかっても厳密解が出なかったものである．そのときの最適値は，今までのところ知られている一番良い値を最適値として結果を出している．

ベンチマーク問題の計算機実験の結果を見ると，MyoDPは高精度かつ時間も一瞬で安定していることが確認できる．Balevらの方法は，精度は良いが，問題が大きくなると時間がかかり始めている．

表 1: ベンチマーク問題

Problem			No. of	CPLEX	1f0(1000)	1f0(1000)	Balev	Balev
n	m	α	Problems	time	time	value	time	value
100	5	0.25	10	4.17	0.01	99.78	0.01	97.80
100	5	0.50	10	4.09	0.01	99.87	0.01	99.13
100	5	0.75	10	12.43	0.01	99.91	0.44	99.56
100	10	0.25	10	37.20	0.01	99.41	3.59	98.71
100	10	0.50	10	35.22	0.01	99.72	2.02	99.39
100	10	0.75	10	20.70	0.02	99.87	2.30	99.67
100	30	0.25	10	727.33	0.03	98.07	5.52	99.33
100	30	0.50	10	660.73	0.03	99.11	2.99	99.80
100	30	0.75	10	127.63	0.04	99.66	4.60	99.87
250	5	0.25	10	115.87	0.021	99.80	1.11	99.23
250	5	0.50	10	116.77	0.02	99.90	1.11	99.50
250	5	0.75	10	56.74	0.02	99.95	0.66	99.80
250	10	0.25	10	∞	0.03	99.56	1.02	99.35
250	10	0.50	10	∞	0.03	99.82	0.04	99.71
250	10	0.75	10	∞	0.04	99.89	0.03	99.81
250	30	0.25	10	∞	0.07	99.01	2.09	99.79
250	30	0.50	10	∞	0.09	99.58	1.96	99.87
250	30	0.75	10	∞	0.11	99.75	0.93	99.93
500	5	0.25	10	1421.52	0.03	99.91	0.01	99.57
500	5	0.50	10	688.80	0.04	99.96	0.01	99.80
500	5	0.75	10	239.29	0.05	99.98	0.01	99.86
500	10	0.25	10	∞	0.06	99.83	0.04	99.64
500	10	0.50	10	∞	0.07	99.92	0.04	99.82
500	10	0.75	10	∞	0.09	99.95	0.04	99.91
500	30	0.25	10	∞	0.14	99.42	4.53	99.87
500	30	0.50	10	∞	0.18	99.74	2.61	99.94
500	30	0.75	10	∞	0.23	99.85	3.21	99.97

5 変数縮小法

Balevらは，次のような手順の変数縮小法を提案している．

1. 暫定解$\tilde{\boldsymbol{x}}$をもとに，$x_j = 1 - \tilde{x}_j$に固定した状態で上界値U_jを求める．

2. アイテムをU_jの大きい順に並べDPを適用する.

3. DPのk回めにおいて，残りを$\tilde{x}_{k+1}$〜$\tilde{x}_n$にした実行可能解(下界値L_k)を求める.

4. このとき，$L_k \geq U_k$になれば，最適解$\boldsymbol{x}^*$において$x_j^* = \tilde{x}_j (j = k+1, \cdots, n)$に固定できる.

このときのDPをMyoDPに変えた変数縮小法を考える．MyoDPのk番目において，アイテムが入り，$k+1$番目以降を暫定解どおりにすると重量制限を超える場合がある．このk番目の状態を除外するならば，$\tilde{\boldsymbol{x}}$より良い解が出る可能性が失われてしまう．逆に残すと，$\tilde{\boldsymbol{x}}$より良い解が出る可能性はあるが，次第に限られたリストの中が全て実行不可能解で一杯になる危険性が出てくる．そこで，MyoDPでk番目を計算する際，より良い解が出る可能性のために，残重量制限の1〜2%程度のオーバーを許し進めることにする.

比較対象のアルゴリズムとして，Balev[1]の近似解法と変数縮小法を用いる．実験に使う問題は，MDKPのベンチマーク問題とし，次の3パターンの実験を行う．暫定解Balev+変数縮小Balev(BB)，暫定解MyoDP+変数縮小Balev(MB)，暫定解MyoDP+変数縮小MyoDP(MM)である．また，CPLEXにおいて分枝限定法に入るまでにどのくらい縮小されたのかを比較に使う.

図5の結果は，アイテム数(100,250,500)別による変数縮小後の変数の割合である．MMはアイテム数が多くなるほど効果があり，多くの変数を固定することができたが，一番結果のよかったのはMBであった．MyoDPは，短時間に高精度の解を得る点においては，優れていることが確認できた．しかし，変数縮小におけるMyoDPは時間的にも変数縮小の観点からも改善の余地があると言える.

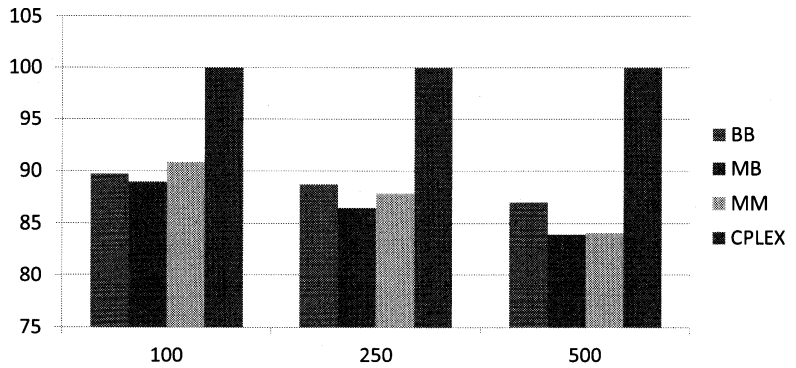


図 5: 縮小後の変数の割合(%)

参考文献

[1] S. Balev, N. Yanev, A. Fréville and R. Andonov: A dynamic programming based reduction procedure for the multidimensional 0-1 knapsack problem. *European Journal of Operational Research*, 2008, 63-76.
[2] P.C. Chu and J.E. Beasley: A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 1998, 63-86.
[3] H. Kellerer, U. Pferschy and D. Pisinger: *Knapsack Problems*, (2004) Springer.
[4] S. Martello and P. Toth: *Knapsack Problems*, (1990) Wiley.