数理解析研究所講究録
第1802巻 2012年 64-70

繰越可能な容量制約付きの多期間ナップサック問題への動的計画法

DP solution algorithms for the Multi-period Knapsack Problem with Carry-over Capacities

防衛大学校情報工学科
柳 秉俊, 山田 武夫

YOU Byungjun, YAMADA Takeo
{g48095, yamada}@nda.ac.jp
Department of Computer Science, The National Defense Academy
Yokosuka, Kanagawa 239-8686, Japan

1 はじめに

本報告は多期間ナップサック問題 (MPKP: multi-period knapsack problem, [5]) を変形し, 未使用の容量が次期以降に繰越し可能な多期間ナップサック問題を考察する. T 期間で, 各期間 t には n_t 個の商品を持つ部分集合 N_t とナップサックの容量 c_t があるとする. 商品 $j \in N_t$ の利得 (重量) は p_j^t (w_j^t) とし, このような商品を総利得が最大となるようにナップサックに詰めるのが問題である. 本報告では, $n_t = |N_t|$ $(t = 1, 2, \cdots, T)$ で, $n = \sum_{t=1}^{T} n_t$ とする.

しかし, 本報告で注目するのは, 未使用の容量が次期以降に繰越し可能な場合である. 例えば, 期間 1 で $c_1 - \sum_{j \in N_1} w_j^1 x_j^1$ だけの容量が残るが, これは期間 2 に繰越されるということである. したがって, 期間 2 での容量制約は次のようになる.

$$\sum_{j \in N_2} w_j^2 x_j^2 \leq c_2 + (c_1 - \sum_{j \in N_1} w_j^1 x_j^1).$$

全ての期間をこのように考えると, 次のような問題が得られる.

$$P_1\text{: maximize} \quad \sum_{t=1}^{T} \sum_{j \in N_t} p_j^t x_j^t \tag{1}$$

$$\text{subject to} \quad \sum_{\tau=1}^{t} \sum_{j \in N_\tau} w_j^\tau x_j^\tau \leq C_t, \quad t = 1, \cdots, T, \tag{2}$$

$$x_j^t \in \{0, 1\}, \quad \forall j \in N_t,\ t = 1, \cdots, T. \tag{3}$$

ここで, $C_t = \sum_{\tau=1}^{t} c_\tau$ は累積容量であり, $C_1 < C_2 < \cdots < C_T$ と仮定する.

P_1 に加えて, 各期間の商品の部分集合 N_t から, それぞれたかだか一個の商品しか採択できないという多肢選択型問題 [14] を紹介する.

$$P_2\text{: maximize} \quad (1)$$

$$\text{subject to} \quad (2),\ (3),$$

$$\sum_{j \in N_t} x_j^t \leq 1, \quad t = 1, 2, \cdots, T. \tag{4}$$

本報告では, P_1 と P_2 を「繰越可能な容量制約付きの多期間ナップサック問題」(MPKPC: multi-period knapsack problem with carry-pver capacities) と呼ぶ. これらの問題は NP-困難 [6] だが, その理由は MPKPC の部分問題であるナップサック問題 (KP: Knapsack problem, [10, 13, 4]) がすでに NP-困難だからである.

P_1 は Dudziński et al. [3] が最初定式化し, 変数縮小法を提案した. Faaland [5] は P_1 の整数計画バージョンを定式化し, 分枝限定法によって 1000 個の変数を持つ規模までの例題を解いている. Lin と Wu[11] は P_2 の変形問題に分枝限定法を提案した. [12] で, Lin と Chen は動的計画法を提案し, APL2 言語 [7, 9] でアルゴリズムを実装した. しかし, APL2 はインタープリターであるため, 小規模の例題しか最適に解けない.

MPKPC は線形 0-1 計画問題なので, 小規模の例題は CPLEX [8] とか GUROBI のような混合整数計画問題 (MIP: mixed integer programming, [15]) ソルバーを用いれば解けるが, 大規模な例題は解けない. 本報告では, これに対処できる動的計画法を紹介する. 提案解法の性能を評価するために, 様々な例題について数値実験を行った. その結果, 本論文の手法が MIP ソルバーを直接用いるよりも優れていることが分かった.

2 動的計画法 (DP: dynamic programming)

P_1 の部分問題として, 期間 1 から期間 t の商品 j までを対象とした問題で, 容量が b であるものを次のように導入する.

$$
\begin{aligned}
P^1_{t,j}(b): \quad \text{maximize} \quad & \sum_{\tau=1}^{t-1}\sum_{j\in N_\tau} p^\tau_j x^\tau_j + \sum_{i=1}^{j} p^t_i x^t_i \\
\text{subject to} \quad & \sum_{l=1}^{\tau}\sum_{i\in N_l} w^l_i x^l_i \le C_\tau, \quad \tau = 1,\cdots,t-1, \\
& \sum_{l=1}^{t-1}\sum_{i\in N_l} w^l_i x^l_i + \sum_{i=1}^{j} w^t_i x^t_i \le b, \\
& x^\tau_j \in \{0,1\}, \quad \forall j \in N_\tau,\ \tau = 1,\cdots,t,
\end{aligned}
$$

ここで, この問題の最適目的関数値を $\bar{z}^1_{t,j}(b)$ とすると, 最適性の原理 [1, 2] により, $\bar{z}^1_{t,j}(b)$ は次の再帰方程式によって計算可能である.

$$
\bar{z}^1_{t,j}(b) = \begin{cases} -\infty, & b < 0, \\ \max\{\bar{z}^1_{t,j-1}(b), \bar{z}^1_{t,j-1}(b-w^t_j) + p^t_j\}, & b \in [0, C_t], \\ \bar{z}^1_{t,j}(C_t), & b > C_t. \end{cases} \tag{5}
$$

この式の初期値を

$$
\bar{z}^1_{t,0}(b) = \begin{cases} \bar{z}^1_{t-1,n_{t-1}}(b), & t \ge 1, \\ 0, & t = 0, \end{cases} \tag{6}
$$

とすると, 次の条件によって, 最適な商品決定が可能である.

$$
x^\star_{t,j}(b) = \begin{cases} 1, & \text{if } \bar{z}^1_{t,j-1}(b-w^t_j) + p^t_j > \bar{z}^1_{t,j-1}(b), \\ 0, & \text{その他.} \end{cases} \tag{7}
$$

P_1 の最適目的関数値は $z^\star = \bar{z}^1_{T,n_T}(C_T)$ で得られ, この DP アルゴリズムの計算複雑さは $O(\sum_{t=1}^{T} n_t C_t)$ である.

P_2 の部分問題としては, 部分容量 b で期間 t では N_t の中からたかだか一個の商品の採否を問う問題を次のように導入する.

$$
\begin{aligned}
P_t^2(b): \quad \text{maximize} \quad & \sum_{\tau=1}^{t}\sum_{j\in N_\tau} p_j^\tau x_j^\tau \\
\text{subject to} \quad & \sum_{l=1}^{\tau}\sum_{j\in N_l} w_j^l x_j^l \le C_\tau, \quad \tau = 1,2,\cdots,t-1, \\
& \sum_{l=1}^{t}\sum_{j\in N_l} w_j^l x_j^l \le b, \\
& \sum_{j\in N_\tau} x_j^\tau \le 1, \quad \tau = 1,2,\cdots,t, \\
& x_j^\tau \in \{0,1\}, \quad \forall j \in N_\tau,\ \tau = 1,2,\cdots,t,
\end{aligned}
$$

ここで, この部分問題の最適目的関数値を $\bar{z}_t(b)$ とすると, 最適性の原理により, $\bar{z}_t(b)$ は次の再帰方程式によって計算可能である.

$$
\bar{z}_t(b) = \max[\bar{z}_{t-1}(b), \max_{j\in N_t}\{\bar{z}_{t-1}(b-w_j^t) + p_j^t\}] \tag{8}
$$

これは Lin [12] の再帰方程式と本質的に同一である. しかし, 計算を明示するため, (8) の代案を次のように導入する.

$$
\bar{z}_{t,j}^2(b) = \max[\bar{z}_{t-1}(b), \max_{1\le i\le j}\{\bar{z}_{t-1}(b-w_i^t) + p_i^t\}] \tag{9}
$$

すべての $b \in [0, C_t]$ で, $\bar{z}_t(b) \equiv \bar{z}_{t,n_t}^2(b)$ であるので, (9) から (5) と類似した次の再帰方程式が得られる.

$$
\bar{z}_{t,j}^2(b) = \begin{cases} -\infty, & b < 0, \\ \max\{\bar{z}_{t,j-1}^2(b), \bar{z}_{t-1,n_{t-1}}^2(b-w_j^t) + p_j^t\}, & b \in [0, C_t], \\ \bar{z}_{t,j}^2(C_t), & b > C_t. \end{cases} \tag{10}
$$

この式の初期値を

$$
\bar{z}_{t,0}^2(b) = \begin{cases} \bar{z}_{t-1,n_{t-1}}^2(b), & t \ge 1, \\ 0, & t = 0, \end{cases} \tag{11}
$$

とすると, 次の条件によって, 最適な商品決定が可能である.

$$
x_{t,j}^\star(b) = \begin{cases} 1, & \text{if } \bar{z}_{t-1,n_{t-1}}^2(b-w_j^t) + p_j^t > \bar{z}_{t,j-1}^2(b), \\ 0, & \text{otherwise.} \end{cases} \tag{12}
$$

P_2 の最適目的関数値は $z^\star = \bar{z}_T(C_T)$, または $z^\star = \bar{z}_{T,n_T}^2(C_T)$ で得られ, 計算複雑さは $O(\sum_{t=1}^{T} n_t C_t)$ である.

3 数値実験

本章では, MPKPC に動的計画法を適用した場合の計算機実験を行い, その性能を評価する. アルゴリズムを ANSI C 言語により実装し, Dell Precision T7400 workstation (CPU: Xeon X5482 Quad-Core×2, 3.20GHz, RAM: 64GB) 上で実験を行った.

3.1 例題の生成

w_j^t と p_j^t は以下の相関タイプによりランダムに発生した.

- 無相関 (UNCOR)
  w_j^t : [1, 100] 間の一様乱数
  p_j^t : [1, 100] 間の一様乱数, w_j^t と独立
- 弱相関 (CORR)
  w_j^t : [1, 100] 間の一様乱数
  p_j^t : $[0.8 \cdot w_j^t, 0.8 \cdot w_j^t + 20]$ 間の一様乱数, w_j^t と独立

ナップサック容量は $C_t = 25t$ とし, t に比例すると仮定する.

3.2 P_1 の計算結果

本節では P_1 に動的計画法を適用したときの性能を評価する. 比較対象としては CPLEX12.2 による直接解法を用いた. 表 1 と 2 には様々な例題を解いたときの最適値 (z^*) と計算時間 (秒) を要約した. これらは, すべてランダムに生成された 10 例題についての平均で, 計算は 1800 秒で打ち切っている. 表 1 では, T を 100 に固定して n_t を 100 から 1600 に変えながら実験を行っており, 表 2 では $n_t = 100$ に固定した. これらの表から以下の所見が得られる.

- CPLEX12.2 による直接解法に比べ, 動的計画法を適用すると, 計算時間が短縮される. 特に, T の増加は CPLEX の計算時間を急激に大きくし, $T = 1600$ では CPLEX による直接解法では解けない例題がある.
- 相関に関係なく, 最適目的関数値 (z^*) は T とともにほぼ線形に増加する. n_t に関する成長率は線形より遅い.
- 相関は z^* を小さくする. これは自然なことで, 相関がある場合には小さい重量と大きい利得を持つ商品の割合が, 相関がない場合と比べ, 少ないからである.

3.3 P_2 の計算結果

同じく, 表 3 と 4 は P_2 に動的計画法を適用した場合の結果である. 表 3 は T を 100 に固定した場合で, 表 4 は n_t を 100 に固定した場合である. これらの表から以下の所見が得られる.

- 動的計画法は CPLEX による直接解法より計算時間が早い. CPLEX による直接解法の場合, 大きい T では 1800 秒以内で厳密解を得ることにしばしば失敗している.
- 動的計画法の計算時間は n_t にほぼ比例する. 一方, 計算時間は T の増加の割合よりも, もっと急激に増加する. これらのことは無相関と弱相関の場合でぼほ同様である.
- 最適目的関数値は T と共に線形に増加するが, $n_t \geq 100$ ではほぼ飽和している.

表 1: 計算結果 ($P_1, T = 100$).

相関	n_t	$z^\star$	計算時間 (秒)	
			動的計画法	CPLEX による直接解法
無	100	38684.6	0.02	1.37
	200	52799.1	0.05	2.34
	400	71888.5	0.12	4.19
	800	96537.7	0.25	7.79
	1600	126765.0	0.50	11.60
弱	100	9216.9	0.02	0.72
	200	11825.4	0.05	1.44
	400	15364.5	0.12	1.91
	800	19835.0	0.24	4.63
	1600	25327.2	0.50	9.45

表 2: 計算結果 ($P_1, n_t = 100$).

相関	T	$z^\star$	計算時間 (秒)	
			動的計画法	CPLEX による直接解法
無	100	38684.6	0.02	1.37
	200	76872.2	0.12	4.52
	400	153674.8	0.48	20.76
	800	306620.0	1.94	117.42
	1600	613381.2	7.80	726.74[5]
弱	100	9216.9	0.02	0.72
	200	18340.8	0.11	3.15
	400	36652.7	0.50	15.90
	800	73177.7	2.01	86.51
	1600	146385.7	8.11	688.72[9]

[5] 1800 秒以内に, 10 例題中 5 例題が解けた.

[9] 同じく, 9 例題が解けた.

表 3: 計算結果 ($P_2, T = 100$).

相関	n_t	$z^\star$	計算時間 (秒)	
			動的計画法	CPLEX による直接解法
無	100	9780.9	0.02	0.69
	200	9861.0	0.05	1.16
	400	9895.9	0.09	1.74
	800	9899.5	0.19	4.75
	1600	9900.0	0.37	10.56
弱	100	3875.9	0.02	1.52
	200	3890.9	0.04	3.60
	400	3898.3	0.09	6.27
	800	3899.7	0.19	7.61
	1600	3900.0	0.38	18.24

表 4: 計算結果 ($P_2, n_t = 100$).

相関	T	$z^\star$	計算時間 (秒)	
			動的計画法	CPLEX による直接解法
無	100	9780.9	0.02	0.69
	200	19565.4	0.10	2.58
	400	39154.2	0.42	6.78
	800	78316.5	1.64	30.11
	1600	156641.7	6.50	713.66[1]
弱	100	3875.9	0.02	1.53
	200	7751.0	0.10	6.15
	400	15501.2	0.39	34.31
	800	31007.2	1.59	148.75
	1600	62021.0	6.37	-

[1] 1800 秒以内に, 10 例題中 1 例題が解けた.

- どの例題も解けなかった.

4 結論

本報告では, 繰越可能な多期間ナップサック問題 (MPKPC: multi-period knapsack problem with carry-over capacities) を扱った. この問題は線形 0-1 計画問題であるため, 市販の汎用最適化ソルバーを用いれば, 解けることが知られているが, 問題の規模が大きくなると, このようなアプローチでは解き難い. このような問題を解決するために, 動的計画法による解法を提案した. その結果, 汎用ソルバーでは解けない問題が解けるようになった. 2 値の決定変数が最大 160000 個の例題まで厳密に解くことに成功している.

参考文献

[1] Bellman, R. (1957). *Dynamic Programming*. Princeton, New Jersey: Princeton University Press.

[2] Cormen, T. H., Leiserson C. E., Rivest, R., & Stein, C. (2009). *Introduction to Algorithms*. (3rd ed.). Massachusetts: MIT Press.

[3] Dudziński, K., & Walukiewicz, S. (1985). On the multiperiod binary knapsack problem. *Methods of Operations Research*, 49, 223-232.

[4] Dudziński, K., & Walukiewicz, S. (1987). Exact methods for the knapsack problem and its generalizations. *European Journal of Operational Research*, 28, 3-21.

[5] Faaland, B. H. (1981). The multiperiod knapsack problem. *Operations Research*, 29, 612-616.

[6] Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: a guide to the theory of NP-completeness*. New York: WH Freeman and Company.

[7] IBM APL2. Available from:http://www-01.ibm.com/software/awdtools/apl, 2011.

[8] IBM ILOG CPLEX Optimizer. Available from: http://www-01.ibm.com/software/integration/optimization /cplex-optimizer, 2011.

[9] Iverson, K. E. (1962). *A Programming Language*, New York: APL Press.

[10] Kellerer, H., Pferschy, U., & Pisinger, D. (2004). *Knapsack problems*. Berlin: Springer.

[11] Lin, E. Y. H., & Wu, C. (2004). The multiple-choice multi-period knapsack problem. *Journal of Operational Research Society*, 55, 187-197.

[12] Lin, E. Y. H., & Chen, M. C. (2010). A dynamic programming approach to the multiple-choice multi-period knapsack problem and the recursive APL2 code. *Journal of Information & Optimization Sciences*, 31, 289-303.

[13] Martello, S., & Toth, P. (1990). *Knapsack problems: Algorithms and computer implementations*. New York: John Wiley & Sons.

[14] Sinha, A., & Zoltners, A. A. (1979). The multiple-choice knapsack problem. *Operations Research*, 27, 503-515.

[15] Wolsey L. A. (1998). *Integer programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. New York: Wiley-Interscience Publication.