数理解析研究所講究録
第747巻 1991年 160-171

160

多重選択ナップザック問題の高速厳密解法

岡山理科大学工学部 仲川 勇二 *(Yuji Nakagawa)*

1. まえがき

離散最適化問題の解法として，新しくモジュラ法[1] が提案された．本稿においては，モジュラ法を多重選択ナップザック問題へ適用する．この問題は非線形（一次元）ナップザック問題[2] とも呼ばれる．

モジュラ法は，原問題の変数をモジュールに対応させている．モジュラ法は，モジュール（変数）を統合により減らし，最終的に単一のモジュール（変数）にして原問題の最適解を求める．モジュラ法は，Marstenら[2] の分枝限定法と動的計画法のハイブリッド利用のアイデアを拡張したものである． Marstenらの解法は幅優先探索を用いた分枝限定法と本質的に同じである．

多重選択ナップザック問題の厳密解法で最も高速と考えられいるのは，現在においてもSinhaら[3] のアルゴリズムである．このアルゴリズムは，分枝限定法に基づいていて，深さ優先探索を利用すると共に線形緩和問題の高速な解法を用いている．この緩和問題の解法アルゴリズムは結果として貪欲法と同じものであった．一方，Marstenら[2] は，彼らのハイブリッド法を多次元の非線形ナップザック問題へ適用している．Sinha-Zoltner法とMarsten-Morin法を多重選択ナップザック問題へ適用した場合の優劣は，本質的には深さ優先探索（Sinhaら）と幅優先探索（Marstenら）の優劣と考えられる．幅優先探索の有利な面としては，幅方向に同一レベルの部分問題列に対して

(1)優越操作が効果的に利用できること，

(2)線形緩和問題の系統的かつ高速な解法の開発が容易であること，

である．劣った面としては

(1)問題によってはきわめて大きな作業領域を必要とすること，

(2)原問題の下限値として利用するための暫定解（本稿では準最適解）の更新に特別な工夫（Marstenらはヒューリスティック解法を利用）を必要とすること，

である．

本稿で提案する解法は，幅優先探索の利点を効果的に利用するとともに，その欠点を(1)に関しては，統合すべきモジュールの選定政策を工夫することと，計算コード（C言語で作成）での同一メモリーの重複使用等で緩和した．(2)に関しては，部分問題の線形緩和を貪欲法で解く際に得られた貪欲解を用いて，準最適解（暫定解）の更新を行い，この欠点を解決している．

本アルゴリズムが，1000クラスで各クラス50変数，合計 5万変数の整数優越条件適用済み多重選択ナップザック問題25問を，EWS (CPUはMC68030)で，最小42秒，最大3686秒，平均1254秒で解いたことを報告する．このテスト問題は Sinhaらと同じ方法で生成された．また，この問題は，一様乱数で係数を生成した場合と比べると，55万変数以上の規模の問題に相当している[3]．

2. 多重選択ナップザックと非線形ナップザック問題

i^U 個のクラスと各クラス i に対して a_i^U 個の変数をもつ多重選択ナップザック問題は次のように書ける.

$$
\begin{aligned}
(K):\quad & max \sum_{i\in I}\sum_{a\in A_i} c_{ia}\xi_{ia} \\
& s.t. \sum_{i\in I}\sum_{a\in A_i} d_{ia}\xi_{ia} \le b \\
& \sum_{a\in A_i} \xi_{ia} = 1 \quad (i\in I) \\
& \xi_{ia}\ge 0,\ integer \quad (a\in A_i,\ i\in I).
\end{aligned}
$$

但し, $I=\{1,2,\ldots,i^U\}$, $A_i=\{1,2,\ldots,a_i^U\}$ $(i\in I)$.

また, この問題と等価で i^U 個の変数と a_i^U 個の代替案をもつ非線形ナップザック問題は

$$
\begin{aligned}
(P):\quad & max\ f(\mathbf{x}) = \sum_{i\in I} f_i(x_i) \\
& s.t.\ g(\mathbf{x}) = \sum_{i\in I} g_i(x_i) \le b \\
& x_i \in A_i \quad (i\in I).
\end{aligned}
$$

となる. 但し, $f_i(a)=c_{ia}$, $g_i(a)=d_{ia}$ $(a\in A_i,\ i\in I)$ である.

問題 (K), (P) において, 全解空間からある最適解を見つけるのに必要となる情報量はそれぞれ

$$
\begin{aligned}
H^K &= \log_2(2^{\sum_{i\in I} a_i^U}) \\
&= \sum_{i\in I} a_i^U \quad (bit), \\
H^P &= \log_2\left(\prod_{i\in I} a_i^U\right) \\
&= \sum_{i\in I} \log_2 a_i^U \quad (bit)
\end{aligned}
$$

となる. 例えば, $i^U=1000$, $a_i^U=50$ のとき, それぞれ $H^K=50000\ (bit)$,

$H^P = 5644\ (bit)$ となる．これは非線形ナップザックの方が曖昧さの少ない簡潔な定式化であることを示している．本アルゴリズムにおいては非線形ナップザック問題の形で原問題を取り扱う．

Sinhaら[3]は問題 (K) に対して次の二優越条件を使った．

整数優越 問題 (K) において $r, s \in A_i$のとき $c_{ir} \geq c_{is}$ かつ $d_{ir} \leq d_{is}$ であるならば，$\xi_{is}=0$ となる最適解が存在する．

ＬＰ優越 $d_{ir} < d_{is} < d_{it}$ かつ $c_{ir} \leq c_{is} \leq c_{it}$ なる $r, s, t \in A_i$ に対して，

$$(c_{is}-c_{ir})/(d_{is}-d_{ir}) \leq (c_{it}-c_{is})/(d_{it}-d_{is})$$

ならば，問題 (K) の線形緩和問題は $\xi_{is}=0$ となる最適解を持つ．

この二つの優越条件を用いると，問題 (K) に対する線形緩和問題として縮小ＬＰ問題 (trancated linear program) が得られる．Sinhaらはこの縮小ＬＰ問題の高速な解法アルゴリズムを提案している．しかし，このアルゴリズムは結果としてFox[6]の貪欲法と同じものであった．文献(3)では，最小化問題であることと，Algorithm I の Step 3 に明かなミスプリントが含まれていることに注意する必要がある．更に，Sinhaらは貪欲解を利用して，ＬＰ解よりも強力でかつ計算の簡単な限界値(Penalty) を提案している．本アルゴリズムの計算コードでは，これらの技法を有効に利用している．

3. 非線形ナップザック問題のためのモジュラ法

本解法は，モジュラ法[1] に基づいており，データの抽象化を考慮したModula-2風疑似コードを用いて書くと図1のようになる．

DATADEF はデータの抽象的定義を行う場所で，C言語では構造体，C++ ではクラス，Pascal および Modula-2 ではレコード型を用いて実現できる．

データ列 $\ll A \gg$，$\ll f \gg$，$\ll g \gg$，$\ll P \gg$ は，原問題 (P) に関連したデータの集まりである．

データ列 $\ll NEAR \gg$，$\ll OPT \gg$ は，それぞれ原問題 (P) の準最適解と厳密解を記録するためのものである．

データ列 $\ll Py \gg$ は，この解法で実行すべき各種政策を表わす．

政策$\ll CIM \gg$は，次に統合すべきモジュールを選定するためのもので，本アルゴリズムでは，代替案数最小と最大の二つのモジュールを選定することとした．

政策 $\ll F \gg$ は，どのような深測技術を用いるか（政策 $\ll FT \gg$）と，どのモジュールに対して深測操作を実施すべきか（政策 $\ll CFM \gg$）と，どのような準最適解更新法を用いるか（政策 $\ll NT \gg$）を決めるために使われる．本アルゴリズムの政策 $\ll FT \gg$ では，限界値操作のみを取扱い，Sinha等と同様ＬＰ優越条件を適用後に貪欲法を用い，更にSinha等の限界値計算法を用いた．

政策 $\ll CFM \gg$ としては，最初と準最適解の更新があったときに，す

べてのモジュールを深測操作の対象とし選定し，その他のときは最も最近統合により生成されたモジュールを選定することとした．また，政策 $\ll NT \gg$ では上限値を求める際に得られた貪欲解で，そのときの準最適解よりも良い解があれば，準最適解を更新することとした．この政策はSinha等の暫定解更新法と実質的に同じものである．

データ列 $\ll M \gg$ は，次に統合すべきモジュール（変数）の添字番号を蓄えるための器である．

データ列 $\ll T \gg$ は更新された現在の問題 (P) の解から原問題に対応した解をたどるのに必要な情報を蓄える．初期値として空データ列を代入する．

関数 $Fathom()$ は政策 $\ll F \gg$ を現在の問題 (P) に対して実施するためのもので，関数の入力としてデータ列 $\ll P \gg$, $\ll T \gg$, $\ll F \gg$, $\ll NEAR \gg$ を使う．関数のリターンとしては，縮小された決定空間をもつ更新された問題 (P) と，その問題の解から原問題の解をたどるのに必要なデータ列 $\ll T \gg$ を戻す．

式 $Copy(\ll NEAR \gg, \ll OPT \gg)$ はデータ列 $\ll NEAR \gg$ をデータ列 $\ll OPT \gg$ に複写することを意味する．

関数 $ChoiceIM()$ は政策 $\ll CIM \gg$ 実施するためのもので，関数のリターンは選定された n^M 個のモジュールの番号集合 $\ll M \gg$ である．実際の計算機コードでは $nM = 2$ としている．

関数 *Integrate*() はモジュール（変数） $x_{m_1}, \ldots, x_{m_{n_M}}$ を単一のモジュールに統合し更新された問題 (P) とデータ列 $\ll T \gg$ を求め，それをリターンとする．本計算機コードでは作業領域を小さくするために，深測操作の一部（実行可能性と優越操作）をこの関数の中で用いている．

関数 *FindOptimalSolution*() は，単一のモジュールからなる問題 (P) の最適解（計算機実験によると，f^{NEAR} より悪い解となることがあることに注意）を求める関数．リターンは最適解 f^{OPT}, $\mathbf{x}^{OPT}$.

Sinhaらと本アルゴリズムの重要な違いは，Sinhaらは深さ優先探索を用いているので，深さ方向の部分問題列に対して系統的な限界値の再計算法 (reoptimization) を用いたが，本アルゴリズムでは，幅方向の（問題 (P) で b の値のみ異なった）部分問題列に対して系統的な再計算法を用いた．深さ方向の再計算による効率化は，最大限工夫しても変数の個数分程度の部分問題を対象とした向上であるが，幅方向はアルゴリズムが同方向に列挙した膨大な部分問題を対象とする．したがって，幅方向（本アルゴリズム）は対象とする部分問題の数が多いことと共に，系統的な再計算の工夫が容易であることから，深さ方向（Sinhaら）よりも格段有利であると言える．

また， *Fathom*() で最初のモジュラ問題に対して用いた政策 $\ll CFM \gg$ は，仲川ら[4] においては各変数に対する上下限の計算を， Mathurら[5] においてはフェーズ 1 (reduction process) を一般化した政策である．

4. 計算機実験

本アルゴリズムの計算機コードを作成するために，C言語を用いた．C言語は既に広範囲の分野で利用されており，科学技術計算の分野での利用が最も遅れていると言われている．しかし，現時点では科学技術計算の分野においては，C言語はいくつかの問題を残しており，C言語がFORTRANより全面的に優れているとは言い難い．本敲において問題の残るC言語を用いた理由は，FORTRAN　ではアルゴリズムの構造的表現が困難であることに加え，データ構造の抽象化が殆どできないために，バグのない完全なコードの作成が困難であったためである．本計算機コードの開発にはC言語用インタープリタである　C-terpを用いた．計算コード作成に際して，メモリーの重複利用等計算速度は多少犠牲になるが，メモリーを有効に利用するための工夫を行った．しかし計算速度を向上させるための特別な工夫は，現在のところ全く行ってはいない．

本計算コードは，一般的な実数係数の多重選択問題を解くために開発した．したがって問題の係数は実数値であると仮定している．また，すべての実数計算は倍精度で行った．問題の係数が整数であるときは，本計算コードは倍精度実数計算の大半の部分が整数計算となりかなり高速となる．

限界値操作が効果的に働くためには，良い準最適解が必要である．本計算コードでは，N-N法[7] を改良した方法を用いた．

テスト問題はSinhaらと同様の方法で，一様乱数を用いて生成した．ここで使用した問題は，Sinhaらで最も困難な場合（$0 \leq d_{ik},\ c_{ik} \leq 8a_i^U$）で，１０００クラス（$i^U=1000$）各クラス５０変数（$a_i^U=50$）の２５問である．本実

験では，SONY NEWS NWS-1850 (CPU MC68030 (25MHz)) を用いた．表1にテスト問題25問に対して，原問題の上限値を貪欲法[3] で求めるのに必要な計算時間，準最適解をスマートグリーディ法[8] で求めるための時間，本アルゴリズムを用いて許容誤差 0 で最適値1を求めるための時間，本アルゴリズムを用いて許容誤差 0.999 で最適値2を求めるための時間を示す．このテスト問題の場合，最適値が整数であるので最適値1，最適値2はともに厳密な意味で最適値である．

表1の上限値を求めるための計算時間は，他の計算時間を判断するための参考として示した．実際には，上限値は準最適解を求める際にほとんど余分な時間を使わずに求めることができる．最適値を求めるのに必要な計算時間は，準最適解を求めるための時間と最適解を求めるための時間の合計である．たとえば，最適値が整数であるという情報を使わなかった場合，最適値を求めるための計算時間の25問の平均は 43.0+4940.8秒となる．最適値が整数であるという情報（すなわち許容誤差 0.999）を使った場合には，43.0+1211.0秒となる．本実験により，実用的に十分な大きさの問題がEWSを用いて容易に解き得ることが分かった．

5. むすび

本論文では，モジュラ法に基づいた多重選択ナップザック問題の解法アルゴリズムの概要を説明した．本アルゴリズムを用いると，従来解かれたことのない大規模な問題がEWSでも解けることを示した．今後は本計算コードを利用して，モジュラー法で使える各種政策および個々の技法の有効性について詳細な研究を行いたい．

文献

(1) 仲川: "離散最適化問題のための新解法", 信学論（A）, **J73-A** , 3, pp. 550-556 (平2-3).

(2) R. E. Marsten and T. L. Morin : "A hybrid approach to discrete mathematical programming", Math. program., **14** , pp. 21-40 (1977).

(3) P. Sinha and A. A. Zoltners: "The multiple-choice knapsack problem", Oper. Res., **27**, 3, pp.503-515 (Jan. 1979).

(4) 仲川, 松下, 疋田: "多次元ナップザック問題の解法", 信学論（A）, **J65-A** , 6, pp.529-534, (昭57-6).

(5) K. Mathur, H. M. Salkin and S. Morito: "An efficient algorithm for the general multiple-choice knapsack problem(GMKP)", Ann. of Oper. Res., **4** , pp. 253-283 (1985/6).

(6) B. Fox: "Discrete optimization via marginal analysis", Manage. Sci., **13** , pp. 210-216 (Nov. 1966).

(7) Y. Nakagawa and K. Nakashima: "A heuristic method for determining optimal reliability allocation", IEEE Trans. Reliab., **R-26** , 3, pp. 156-161 (Aug. 1977).

DATADEF

$\ll A \gg = \{i^U, \{a_1^U, a_2^U, \ldots, a_{i^U}^U\}\};$

$\ll f \gg = \{f_1(1), \ldots, f_1(a_1^U), \ldots, f_{i^U}(1), \ldots, f_{i^U}(a_{i^U}^U)\}$

$\ll g \gg = \{\{g_1(1), \ldots, g_1(a_1^U), \ldots, g_{i^U}(1), \ldots, g_{i^U}(a_{i^U}^U)\}, b\}$

$\ll P \gg = \{\ll A \gg, \ll f \gg, \ll g \gg\}$

$\ll NEAR \gg = \{f^{NEAR}, \{x_1^{NEAR}, \ldots, x_{i^U}^{NEAR}\}\};$

$\ll OPT \gg = \{f^{OPT}, \{x_1^{NEAR}, \ldots, x_{i^U}^{NEAR}\}\};$

$\ll Py \gg = \{\ll CIM \gg, \ll F \gg\};$

$\ll M \gg = \{n^M, \{m_1, m_2, \ldots, m_{n^M}\}\};$

ENDDEF

FUNCTION ModularApproach()

INPUT 問題 $\ll P \gg$, 政策 $\ll Py \gg$, 準最適解 $\ll NEAR \gg$;

BEGIN

$Yes \leftarrow 1$; $No \leftarrow 0$; $IsNearsolOptimal \leftarrow No$;

$\ll T \gg \leftarrow \{\emptyset\}$;

WHILE $i^U \geq 2$ **DO**

$\{\ll P \gg, \ll T \gg, \ll NEAR \gg\} \leftarrow$ Fathom($\ll P \gg, \ll T \gg, \ll F \gg, \ll NEAR \gg$);

IF exist $i \in \{1, \ldots, i^U\}$ such that $a_i^U = 0$ **THEN**

$IsNearsolOptimal \leftarrow Yes$;

$Copy(\ll NEAR \gg, \ll OPT \gg)$;

EXIT this while loop;

ENDIF

$\ll M \gg \leftarrow$ ChoiceIM($\ll P \gg, \ll CIM \gg$);

$\{\ll P \gg, \ll T \gg\} \leftarrow$ Integrate($\ll M \gg, \ll P \gg, \ll T \gg$);

ENDWHILE

IF $IsNearsolOptimal = No$ **THEN**

$\ll OPT \gg \leftarrow$ FindOptimalSolution($\ll P \gg, \ll T \gg$);

IF $f^{OPT} < f^{NEAR}$ **THEN**

$Copy(\ll NEAR \gg, \ll OPT \gg)$;

ENDIF

ENDIF

RETURN 最適解 $\ll OPT \gg$;

END /* ModularApproach */

図1．非線形ナップザック問題のモジュラ法アルゴリズム

表1　テスト問題の上限値，準最適値，最適値1，
最適値2を求めるための計算時間（sec.）

問題	上限値	準最適値	最適値1	最適値2
1	7.0	43.9	4992.2	3164.0
2	6.9	43.7	3852.5	0.0
3	6.9	44.8	3200.5	0.0
4	6.9	41.1	5406.5	3161.2
5	6.9	42.1	4549.9	3097.5
6	6.8	42.3	3712.7	694.8
7	6.8	41.7	3981.7	0.0
8	6.9	43.6	3943.7	1778.7
9	7.1	43.4	3529.4	0.0
10	6.9	41.3	3081.8	0.0
11	6.9	42.1	3217.4	0.0
12	7.0	42.6	4114.9	0.0
13	6.9	41.5	3354.2	0.0
14	6.9	42.2	7122.6	3245.6
15	6.7	42.7	0.0	0.0
16	6.9	43.2	7430.3	3250.5
17	7.1	46.2	3375.7	0.0
18	7.0	41.0	7561.7	3280.4
19	7.0	43.4	3330.1	0.0
20	7.0	44.2	15655.2	445.1
21	7.0	45.3	9119.1	314.2
22	6.9	43.8	5727.4	3166.4
23	6.9	42.6	3922.3	1032.2
24	6.9	42.6	5983.4	3643.1
25	7.1	45.1	3354.2	0.0
平均	6.9	43.0	4940.8	1211.0
最小	6.7	41.0	0.0	0.0
最大	7.1	46.2	15655.2	3643.1