ナップサック問題およびその発展問題の統一的解法

Solving a Class of Knapsack Problems on Recursive Data Structures

篠埜 功[†] Isao SASANO　　胡 振江[†] Zhenjiang HU　　武市 正人[†] Masato TAKEICHI　　小川 瑞史[††] Mizuhito OGAWA

†東京大学大学院工学系研究科情報工学専攻
Department of Information Engineering, University of Tokyo
({sasano,hu,takeichi}@ipl.t.u-tokyo.ac.jp)
††NTT コミュニケーション科学基礎研究所
NTT Communication Science Laboratories
(mizuhito@theory.brl.ntt.co.jp)

概 要

組合せ最適化問題の代表的問題の１つであるナップサック問題は、 NP 困難であるが、重みが整数のみの場合には動的計画法による多項式時間アルゴリズムが知られている。しかし、入力データにリスト、木等の構造を導入して構造に関する制約条件（たとえば互いに隣合わない等）を加えると、効率のよいアルゴリズムを考えるのは容易ではない。本論文では、入力データが再帰構造を持つナップサック問題を統一的に扱い、仕様からの効率のよいアルゴリズム導出法を提案する。

1 はじめに

ナップサック問題とは、価値、重みを持つ N 個の荷物と、容量 C の袋が入力として与えられたときに、重み和が C を越えないような荷物の選び方の中で、価値の和が最大の選び方を求める問題である。この問題は組合せ最適化問題の代表的問題の１つであり、 NP 困難であるが、重みが整数のみの場合には動的計画法による多項式時間アルゴリズムが知られている [1]。しかし、入力データにリスト、木等の構造を導入して構造に関する制約条件（たとえば互いに隣合わない等）を加えると、効率のよいアルゴリズムを考えるのは容易ではない [2]。

本論文では、ナップサック問題を最大重み和問題 [3]としてとらえ直し、ナップサック問題のさまざまな発展問題に対する効率のよいアルゴリズムの統一的導出法を提案する。この導出法のポイントは、再帰関数で記述された制約条件から、最適化定理 [3]を適用することによりアルゴリズムが導出されるところにある。本論文では、関数型言語 Haskell [4]に近い記法を用いる。

2 最大重み和問題および最適化定理

最大重み和問題とは、「要素に重みの与えられたある再帰データ x が与えられたときに、性質 p を満たす x 中の要素の部分集合のうちで重み和が最大のものを求める」という問題である。筆者らは [3]において、再帰関数により記述された性質記述から、最大重み和問題を解く線形時間アルゴリズムを導出する手法を提案した。本節では、その結果のみを示す。詳しくは、 [3]を参照されたい。導出においては、次の最適化定理が重要な役割を果たす。

Theorem 1 (最適化定理)
性質 p が有限の値域をもつ関数 $f_i (i = 1, \ldots, n)$ と mutumorphisms をなしているならば、性質 p に関

$$
\begin{array}{l}
opt\ accept\ \phi_1 \ldots \phi_k\ x = getdata\ (\uparrow_{snd}\ /\ [\,(c,w,r^*) \mid (c,w,r^*) \leftarrow (\![\psi_1,\ldots,\psi_k]\!)_D\ x,\ accept\ c\,]\,) \\
\quad \textbf{where}\ \psi_i\ (e, cand_1, \ldots, cand_{n_i}) = \\
\qquad eachmax\,[\,(\phi_i\ (e^*, c_1, \ldots, c_{n_i}), \\
\qquad\qquad (\textbf{if}\ marked\ e^*\ \textbf{then}\ weight\ e^*\ \textbf{else}\ 0) + w_1 + \ldots + w_{n_i}, \\
\qquad\qquad C_i\ (e^*, r_1^*, \ldots, r_{n_i}^*)) \mid \\
\qquad\qquad\qquad e^* \leftarrow [mark\ e, unmark\ e], \\
\qquad\qquad\qquad (c_1, w_1, r_1^*) \leftarrow cand_1, \cdots, (c_{n_i}, w_{n_i}, r_{n_i}^*) \leftarrow cand_{n_i}] \qquad (i = 1, \ldots, k) \\
eachmax\ xs = foldl\ f\ [\,]\ xs \\
\quad \textbf{where}\ f\ [\,]\ (c, w, cand) = [(c, w, cand)] \\
\qquad\quad f\ ((c, w, cand) : opts)\ (c', w', cand') = \textbf{if}\ c = c'\ \textbf{then} \\
\qquad\qquad\qquad\qquad \textbf{if}\ w > w'\ \textbf{then}\ (c, w, cand) : opts \\
\qquad\qquad\qquad\qquad \textbf{else}\ opts +\!\!+ [(c', w', cand')] \\
\qquad\qquad\qquad \textbf{else}\ (c, w, cand) : f\ opts\ (c', w', cand') \\
getdata\ (\_, \_, x) = x
\end{array}
$$

図 1　一般的最適化関数 opt

する最大重み和問題には線形時間アルゴリズムが存在し、それを機械的に導出できる。 □

Mutumorphisms [5] は、単一の構造再帰である catamorphism を、相互再帰的に定義された関数群に一般化したものである。再帰的データ型は、一般的に次のように定義することができる。

$$
\begin{array}{rcl}
D\ \alpha & = & C_1\ (\alpha, D_1, \ldots, D_{n_1}) \\
& | & C_2\ (\alpha, D_1, \ldots, D_{n_2}) \\
& | & \cdots \\
& | & C_k\ (\alpha, D_1, \ldots, D_{n_k})
\end{array}
$$

D_i は $D\ \alpha$ を表し、 C_i は構成子を表す。リスト、木などの再帰データ型はこの形で表現できる。

この再帰データ上で mutumorphisms は次のように定義される。

Definition 1 (Mutumorphisms)

再帰的データ $D\ \ \alpha$ 上の mutumorphisms とは、次のような形で相互再帰的に定義された関数群 $f_1, f_2, \ldots, f_n$ を指す。 $i \in \{1, 2, \ldots, n\}$ に対して、

$$
\begin{array}{l}
f_i\ (C_1\ e\ x_1 \ldots x_{n_1}) = \phi_{i1}\ e\ (h\ x_1) \ldots (h\ x_{n_1}) \\
f_i\ (C_2\ e\ x_1 \ldots x_{n_2}) = \phi_{i2}\ e\ (h\ x_1) \ldots (h\ x_{n_2}) \\
\quad \vdots \\
f_i\ (C_k\ e\ x_1 \ldots x_{n_k}) = \phi_{ik}\ e\ (h\ x_1) \ldots (h\ x_{n_k}) \\
\textbf{where} \\
\qquad h\ x = (f_1\ x, f_2\ x, \ldots, f_n\ x)
\end{array}
$$
□

以下では、有限の値域を持つ関数からなる mutumorphisms を有限 mutumorphisms と呼ぶ。

性質 p が有限 mutumorphisms で定義されているとき、 p は、有限の定義域を持つ述語 $accept$ と有限の値域を持つ catamorphism に次のように分解することができる。

$$p = accept \circ (\![\phi_1, \phi_2, \ldots, \phi_k]\!)$$

$(\![\_]\!)$ は、データ型 $D\ \alpha$ 上の catamorphism を表す。引数として入力データを受けとり、性質 p に関する最大重み和問題の 1 つの解を返す関数 mws_p は、上記の関数 $accept, \phi_1, \phi_2, \ldots, \phi_k$ を用いて、

$$mws_p\ x = opt\ accept\ \phi_1 \ldots \phi_k\ x$$

と記述される。関数 opt の定義は、図 1に与えられており、この関数は線形時間関数である。関数 opt の正しさ、線形性は、 [3]に示されている。

3　通常のナップサック問題

ここでは、ナップサック問題を、最大重み和問題としてとらえ、最適化定理を適用することにより多項式時間アルゴリズムを導出する方法を示す。

3.1　仕　　様

荷物の集合は、重みと価値の対を要素とする non-empty リスト型 $[(Weight, Value)]$ で表す。ナップサック問題は、荷物の価値を重みとする最大重み和問題であり、満たすべき性質は、選ばれた荷物の重み和が袋の容量 C を越えないという性質 $knap$ であ

る。

$$knap\ xs = weightsum\ xs \leq C$$

3.2　導　　出

性質 $knap$ を non-empty リスト上の有限 mutumorphisms で表すために、以下のような関数 cut を導入する。

$$\begin{array}{lcl} cut & : & Weight \to Weight \\ cut\ w & = & \mathbf{if}\ w \leq C\ \mathbf{then}\ w\ \mathbf{else}\ C+1 \end{array}$$

この関数 cut は、重みを引数にとり、重みが C より大きければ $C + 1$ を返し、そうでなければその重みをそのまま返す。この関数 cut を用いると、性質 $knap$ は以下のように表すことができる。

$$\begin{array}{l} knap\ xs = sumw\ xs \leq C \\ sumw\ [x] = cut\ (\mathbf{if}\ marked\ x\ \mathbf{then}\ weight\ x \\ \qquad\qquad\qquad\qquad \mathbf{else}\ 0) \\ sumw\ (x:xs) = \\ \qquad cut\ ((\mathbf{if}\ marked\ x\ \mathbf{then}\ weight\ x \\ \qquad\qquad \mathbf{else}\ 0) + sumw\ xs) \end{array}$$

関数 $marked$ は、引数に与えられた要素が選ばれている (マーク付けられている) ならば $True$、そうでないならば $False$ を返す関数である。 $knap$ の取り得る値は $True, False$, $sumw$ の取り得る値は $0, 1, \ldots, C + 1$ であるので、関数群 $knap, sumw$ は、non-empty リスト上の有限 mutumorphisms をなす。よって、最適化定理を適用することにより、次のアルゴリズムを得る。

$$\begin{array}{l} knapsack\ x = opt\ accept\ \phi_1\ \phi_2\ x \\ \quad \mathbf{where} \\ \qquad accept\ x = x \leq C \\ \qquad \phi_1\ x = cut\ (\mathbf{if}\ marked\ x\ \mathbf{then}\ weight\ x \\ \qquad\qquad\qquad\qquad \mathbf{else}\ 0) \\ \qquad \phi_2\ (x,y) = cut\ ((\mathbf{if}\ marked\ x\ \mathbf{then} \\ \qquad\qquad\qquad\qquad\qquad weight\ x\ \mathbf{else}\ 0) + y) \end{array}$$

このアルゴリズムの計算量は、 $O(CN)$ である。

4　リスト上のナップサック問題

前章では通常の、構造のないナップサック問題を扱った。この章では、荷物がリスト状に並んでいる場合を考え、隣り合った荷物は選ぶことができないという制約条件が加わった問題を解くアルゴリズムを最適化定理により導出する。

4.1　仕　　様

まず、隣り合った荷物は選ばれないという性質 $indep$ は、次のように定義できる。

$$\begin{array}{l} indep\ [x] = True \\ indep\ (x:xs) = \\ \qquad \mathbf{if}\ marked\ x\ \mathbf{then} \\ \qquad\quad not\ (marked\ (hd\ xs))\ \wedge\ indep\ xs \\ \qquad \mathbf{else}\ indep\ xs \\ hd\ [x] = x \\ hd\ (x:xs) = x \end{array}$$

$indep$ の引数が $[x]$ の場合は、荷物が一つの場合に対応し、それを選んでも選ばなくても隣り合った荷物は選ばれないので、 $True$ である。 $indep$ の引数が $(x : xs)$ の場合は、荷物が複数の場合に対応し、先頭の荷物 x が選ばれた場合には、残った荷物 xs から、先頭 $hd\,xs$ を選ばずに、かつ性質 $indep$ を満たすように選ばなければならない。 hd は、 non-empty リストを引数にとり、その先頭要素を返す関数である。この性質 $indep$ を用いることにより、この問題の性質 p は次のように記述できる。

$$p\ xs \quad = \quad indep\ xs\ \wedge\ knap\ xs$$

4.2　導　　出

この性質 p は、関数 hd の値域が (一般には) 有限でないため、有限 mutumorphisms では表されていない。性質 p を有限 mutumorphisms で表すために、関数 $indep$ 中の $not\ (marked\ (hd\ xs))$ の部分を p_1 とおく。融合変換により、 p_1 は、

$$\begin{array}{lcl} p_1\ [x] & = & not\ (marked\ x) \\ p_1\ (x:xs) & = & not\ (marked\ x) \end{array}$$

と変換される。よって、性質 $indep$ は、

$$\begin{array}{l} indep\ [x] = True \\ indep\ (x:xs) = \mathbf{if}\ marked\ x \\ \qquad\qquad\qquad\qquad \mathbf{then}\ p_1\ xs\ \wedge\ indep\ xs \\ \qquad\qquad\qquad\qquad \mathbf{else}\ indep\ xs \end{array}$$

となり、関数群 $p, indep, p_1, knap, sumw$ は non-empty リスト上の有限 mutumorphisms をなす。よって、結果は省略するが、最適化定理により $O(CN)$ のアルゴリズムが得られる。

5　木上のナップサック問題

この章では、荷物が木状になっている場合を考え、選ぶ荷物はすべて枝でつながっていなければならないという制約条件が加わった問題を解くアルゴ

リズムを最適化定理により導出する。[2]においては、関係を用いることによりこの問題に類似した問題を解くアルゴリズムを導出しているが、導出時に適用する規則の条件が複雑であり、規則が適用しにくい。それに対し、最適化定理は条件が簡潔であるので適用しやすく、導出の自動化も可能である。

5.1 仕　　様

木は次のように定義される 2 分木であるとする。

$$Tree\ \alpha ::= Leaf\ \alpha \mid Bin\ \alpha\ (Tree\ \alpha)\ (Tree\ \alpha)$$

まず、和が C を越えないという性質 tk は、

$$\begin{aligned}
&tk\ t = tws\ t \leq C\\
&tws\ (Leaf\ x) =\\
&\quad cut\ (\mathbf{if}\ marked\ x\ \mathbf{then}\ weight\ x\ \mathbf{else}\ 0)\\
&tws\ (Bin\ x\ t_1\ t_2) =\\
&\quad cut\ ((\mathbf{if}\ marked\ x\ \mathbf{then}\ weight\ x\ \mathbf{else}\ 0)\\
&\qquad + tws\ t_1 + tws\ t_2)
\end{aligned}$$

のように記述できる。次に、選んだ荷物が枝でつながっているという性質 tc は次のように書ける。

$$\begin{aligned}
&tc\ (Leaf\ x) = True\\
&tc\ (Bin\ x\ t_1\ t_2) =\\
&\quad \mathbf{if}\ marked\ x\ \mathbf{then}\ tc'\ t_1\ \wedge\ tc'\ t_2\\
&\quad \mathbf{else}\ (nm\ t_1\ \wedge\ tc\ t_2)\ \vee\ (tc\ t_1\ \wedge\ nm\ t_2)
\end{aligned}$$

木が、1 つの葉のみからなる場合には、性質 tc は満たされる。木が、$(Bin\ x\ t_1\ t_2)$ である場合には、x が選ばれていた場合には、t_1 と t_2 は、選ばれた頂点が根の側に集まっている (すなわち、ある頂点が選ばれていたならば、その親の頂点も選ばれている) という性質 tc' を共に満たしているかどうかチェックし、x が選ばれていなかった場合には、t_1, t_2 の一方には選ばれた頂点が存在してはならず、もう一方は性質 tc を満たしていなければならない。木に選ばれた頂点が存在しないという性質が nm であり、

$$\begin{aligned}
&nm\ (Leaf\ x) = not\ (marked\ x)\\
&nm\ (Bin\ x\ t_1\ t_2) =\\
&\quad not\ (marked\ x)\ \wedge\ nm\ t_1\ \wedge\ nm\ t_2
\end{aligned}$$

のように定義できる。また、性質 tc' は、

$$\begin{aligned}
&tc'\ (Leaf\ x) = True\\
&tc'\ (Bin\ x\ t_1\ t_2) =\\
&\quad \mathbf{if}\ marked\ x\ \mathbf{then}\ tc'\ t_1\ \wedge\ tc'\ t_2\\
&\quad \mathbf{else}\ nm\ t_1\ \wedge\ nm\ t_2
\end{aligned}$$

のように定義することができる。

これら 2 つの性質 tk, tc を用いることにより、この問題の性質 p は、次のように記述できる。

$$p\ t = tk\ t\ \wedge\ tc\ t$$

5.2 導　　出

関数群 p, tk, tc, tws, nm, tc' は木上の有限 mutumorphisms をなすので、最適化定理を適用することにより、次のアルゴリズムを得る。

$$\begin{aligned}
&knaptree\ t = opt\ accept\ \phi_1\ \phi_2\ t\\
&\ \ \mathbf{where}\\
&\quad accept\ (a,b,c,d) = a \leq C\ \wedge\ b\\
&\quad \phi_1\ x = (cut\ (\mathbf{if}\ marked\ x\ \mathbf{then}\ weight\ x\\
&\qquad\qquad \mathbf{else}\ 0),\\
&\qquad\quad True, not\ (marked\ x), True)\\
&\quad \phi_2\ (x, (a_1,b_1,c_1,d_1)\ (a_2,b_2,c_2,d_2)) =\\
&\qquad (cut\ (\mathbf{if}\ marked\ x\ \mathbf{then}\ weight\ x\\
&\qquad\qquad \mathbf{else}\ 0) + a_1 + a_2),\\
&\qquad\ \mathbf{if}\ marked\ x\ \mathbf{then}\ d_1\ \wedge\ d_2\\
&\qquad\ \mathbf{else}\ (c_1\ \wedge\ b_2)\ \vee\ (b_1\ \wedge\ c_2),\\
&\qquad\ not\ (marked\ x)\ \wedge\ c_1\ \wedge\ c_2,\\
&\qquad\ \mathbf{if}\ marked\ x\ \mathbf{then}\ d_1\ \wedge\ d_2\\
&\qquad\ \mathbf{else}\ c_1\ \wedge\ c_2)
\end{aligned}$$

このアルゴリズムの計算量は、$O(C^2N)$ である。

6 結　　論

本論文では、重みが整数の各種ナップサック問題を、最大重み和問題とみることによって統一的に扱い、多項式時間アルゴリズムの導出法を提案した。

参考文献

[1] Silvano Martello and Paolo Toth. *Knapsack Problems : Algorithms and Computer Implementations.* Wiley-Interscience series in discrete mathematics and optimization. John Wiley & Sons Ltd., 1990.

[2] Oege de Moor. A generic program for sequential decision processes. In *Proceedings of 7th International Symposium on Programming Languages : Implementations, Logics, and Programs (PLILP'95)*, LNCS 982, pages 1–23, Utrecht, the Netherlands, September 1995.

[3] Isao Sasano, Zhenjiang Hu, Masato Takeichi, and Mizuhito Ogawa. Make it practical: A generic linear-time algorithm for solving maximum-weightsum problems. In *The 2000 ACM SIGPLAN International Conference on Functional Programming (ICFP2000)*, Montreal, Canada, September 2000. ACM Press. to appear.

[4] Richard Bird. *Introduction to Functional Programming using Haskell (second edition).* Prentice Hall, 1998.

[5] E. Meijer, M. Fokkinga, and R. Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In *Proc. Conference on Functional Programming Languages and Computer Architecture*, LNCS 523, pages 124–144, Cambridge, Massachusetts, August 1991.