8　組合せ最適化・整数計画問題

前の章では，最短路問題がグラフの頂点や枝の数によって与えられる問題規模の多項式程度の基本演算で解の得られることを学びましたが，このようなアルゴリズムを**多項式時間アルゴリズム** (polynomial-time algorithm) といいます．最大流問題もダイクストラ法で増量可能路を求めることで多項式時間のうちに解の得られることが証明できます．一方，ネットワーク流問題よりずっと単純に見えても，多項式時間アルゴリズムの存在が絶望視されている問題もあります．その代表が**ナップサック問題** (knapsack problem) です．ここでは，多項式アルゴリズムのない場合によく用いられる 2 つのアルゴリズムのメカニズムをナップサック問題を例に紹介します.

8.1　0-1 ナップサック問題

例 8.1.　品物 G_1, G_2, G_3, G_4 をナップサックに詰めてハイキングに出かけたい．品物の体積の総和がナップサックの容積を越えてしまうとき，どの品物をナップサックに詰めればよいか ？

ナップサックの容積を $b = 12$ (ℓ)，4 つの品物それぞれの体積を $a_1 = 3$, $a_2 = 6$, $a_3 = 4$, $a_4 = 5$ (ℓ) としましょう．この場合，品物の体積は $\sum_{j=1}^{4} a_j = 18$ (ℓ) となってナップサックに入りません．しかし，このハイカーにとっての各品物の価値が $c_1 = 7$, $c_2 = 9$, $c_3 = 5$, $c_4 = 5$ のように数値化されていれば，この問題は数理計画問題として定式化することができます.

品物が G_1, G_2, …, G_n の n 個の場合，それぞれに変数 x_1, x_2, …, x_n を設定し，

$$x_j = \begin{cases} 1, & \text{品物 } G_j \text{ をナップサックに詰めるとき} \\ 0, & \text{品物 } G_j \text{ をナップサックに詰めないとき} \end{cases}$$

と定めます．これにより，問題は次のように定式化できます:

$$\left|\begin{array}{lllllllll} \text{最大化} & c_1x_1 & + & c_2x_2 & + & \cdots & + & c_nx_n & \\ \text{条　件} & a_1x_1 & + & a_2x_2 & + & \cdots & + & a_nx_n & \leq \quad b \\ & & & x_j \in \{0,\ 1\}, & & & & j = 1,\ 2,\ \ldots,\ n & \end{array}\right. \tag{8.17}$$

ただし，係数 $a_1, \ldots, a_n, b, c_1, \ldots, c_n$ はすべて正で，

$$a_1 + a_2 + \cdots + a_n > b \tag{8.18}$$

であるものとします．この問題の最大の特徴は，各変数 x_j が 0 または 1 という整数値しか取れない点にあります．この種の問題を **0-1 整数計画問題** (0-1 integer programming problem) とよび，一般には

$$\left|\begin{array}{lllllllll} \text{最大化} & c_1x_1 & + & c_2x_2 & + & \cdots & + & c_nx_n & \\ \text{条　件} & a_{i1}x_1 & + & a_{i2}x_2 & + & \cdots & + & a_{in}x_n & \leq \ b, \quad i = 1, 2, \ldots, m \\ & & & & & & & x_j \in \{0, 1\}, & j = 1, 2, \ldots, n \end{array}\right. \tag{8.19}$$

のように定式化され，係数が正であることは必ずしも仮定されません．

問題 (8.17) のもう一つの特徴は，一般の 0-1 計画問題 (8.25) と比較すればわかるように，0-1 条件 $x_j \in \{0, 1\}, j = 1, 2, \ldots, n$ 以外の制約条件が 1 本しか存在しないことです．そのため，0-1 条件を

$$0 \leq x_j \leq 1, \quad j = 1, 2, \ldots, n$$

に**連続緩和** (continuous relaxation) すれば，(8.17) からは多項式時間で答の得られる線形計画問題

$$\left|\begin{array}{lllllllll} \text{最大化} & c_1x_1 & + & c_2x_2 & + & \cdots & + & c_nx_n & \\ \text{条　件} & a_1x_1 & + & a_2x_2 & + & \cdots & + & a_nx_n & \leq \ b \\ & & & 0 \leq x_j \leq 1, & & & & & j = 1, 2, \ldots, n \end{array}\right. \tag{8.20}$$

が導かれます．いま，c_j/a_j が

$$c_1/a_1 \geq c_2/a_2 \geq \cdots \geq c_n/a_n \tag{8.21}$$

の順になっているものとしましょう (必要ならば，ソートして添字を付け替えます)．品物を G_1 から $G_2, G_3, \ldots$ の順にナップサックに詰めていき，スペースがあって丸ごと入れば，それを 1 個いれます．もしも最後に入れようとする品物 G_p がスペース不足で丸ごと入らなけれ

ば，その一部 (スペース分) だけを入れます．こうして得られる解

$$\overline{x}_j = \begin{cases} 1, & j = 1,\ \ldots,\ p-1 \\ \left(b - \sum_{i=1}^{p-1} a_i \right) / a_p, & j = p \\ 0, & j = p+1,\ \ldots,\ n \end{cases}$$

が，実は (8.17) の連続緩和問題 (8.29) の最適解です．

連続緩和問題の最適解 $\overline{\mathbf{x}} = (\overline{x}_1,\ \ldots,\ \overline{x}_n)$ と元の 0-1 整数計画問題の最適解 $\mathbf{x}^* = (x_1^*,\ \ldots,\ x_n^*)$ との間には，品物の詰め方からもわかるように

$$\sum_{j=1}^{n} c_j \overline{x}_j \geq \sum_{j=1}^{n} c_j x_j^* \tag{8.22}$$

の関係があります．しかし，最後の品物 G_p がちょうど 1 個入るとは限らないので，$\overline{\mathbf{x}}$ は必ずしも (8.17) の最適解とはなりません．

8.2 動的計画法

0-1 ナップサック問題 (8.17) の係数 $a_1, \cdots, a_n, b$ は整数であるものと仮定しましょう．計算機上では，すべての数が有理数として扱われるので，この仮定によって不都合は生じません．

ここでは次のような関数を考えます：

$V_k(y)$ ： 容積 y の中に詰める品物の候補を $G_1, \ldots, G_k$ に限定したときに
得られる最大の価値．

したがって，(8.17) の最適解は目的関数値が $V_n(b)$ の実行可能解ということになります．この値を求めるため，$V_k(y)$ が満たす次の関係に注目しましょう：

$$V_k(y) = \max\{V_{k-1}(y),\ V_{k-1}(y - a_k) + c_k\} \tag{8.23}$$

ここでは $k = 1,\ 2,\ \ldots,\ n$; $y = 0,\ 1,\ \ldots,\ b$ ですが，

$$\left.\begin{aligned} V_0(y) &= 0, \qquad y \geq 0 \text{ の場合} \\ V_k(y) &= -\infty, \quad y < 0 \text{ の場合} \end{aligned}\right\} \tag{8.24}$$

とします．式 (8.23) を**動的計画法** (dynamic programming: DP) の**漸化式** (recursion fomula) とよびますが，「容積 y の中に詰める候補を $G_1, \ldots, G_k$ とするときの最大価値 $V_k(y)$ は，

- y の中に詰める候補を $G_1, \ldots, G_{k-1}$ とするときの最大価値 $V_{k-1}(y)$,
- y の中に，まず G_k を詰め，残った容積 $y - a_k$ に詰める候補を $G_1, \ldots, G_{k-1}$ とするときの最大価値 $V_{k-1}(y - a_k) + c_k$

のどちらか大きいほうに等しい」ことを意味します.

例 8.2. 次の例題を動的計画法を使って実際に解いてみましょう：

$$\left|\begin{array}{lllllllll} \text{最大化} & 7x_1 & + & 9x_2 & + & 5x_3 & + & 5x_4 & \\ \text{条　件} & 3x_1 & + & 6x_2 & + & 4x_3 & + & 5x_n & \leq \ 12 \\ & & & x_j \in \{0,\ 1\}, & & & & j = 1,\ 2,\ 3,\ 4. & \end{array}\right.$$

まず，$V_1(1) = 0, V_1(2) = 0, V_1(3) = \cdots = V_1(12) = 7$ は明らかでしょう．それぞれに対応する x_1 の値を付けて下のような表を作ります：

k / y	1	2	3	4	5	6	7	8	9	10	11	12
$V_1(y)$	0	0	7	7	7	7	7	7	7	7	7	7
x_1	0	0	1	1	1	1	1	1	1	1	1	1

次に $a_2 = 6$ ですから，

$$V_2(y) = V_1(y), \qquad y \leq 5 \text{ のとき}$$

$$V_2(6) = \max\{V_1(6),\ V_1(6-6) + c_2\} = \max\{7,\ 9\} = 9$$

$$V_2(7) = \max\{7,\ 9\} = 9, \quad V_2(8) = \max\{7,\ 9\} = 9$$

$$V_2(9) = \max\{V_1(9),\ V_1(9-6) + c_2\} = \max\{7,\ 7+9\} = 16.$$

以下同様にして $V_4(12)$ まで計算し，それと同時に max を与える x_j の値も求めます：

k / y	1	2	3	4	5	6	7	8	9	10	11	12
$V_2(y)$	0	0	7	7	7	9	9	9	16	16	16	16
x_2	0	0	0	0	0	1	1	1	1	1	1	1
$V_3(y)$	0	0	7	7	7	9	12	12	16	16	16	16
x_2	0	0	0	0	0	0	1	1	0	0	0	0
$V_2(y)$	0	0	7	7	7	9	9	9	16	16	16	17
x_2	0	0	0	0	0	0	0	0	0	0	0	1

さて，最適解 $\mathbf{x}^* = (x_1^*,\ x_2^*,\ x_3^*,\ x_4^*)$ ですが，次の手順で求めます： まず，

$$V_4(12) = 17, \qquad\qquad x_4^* = 1.$$

ここから逆向きに計算していき，

$$\begin{aligned} V_3(12 - a_4) = V_3(7) &= 12, \quad & x_3^* &= 1 \\ V_2(7 - a_3) = V_2(3) &= 7, & x_2^* &= 0 \\ V_1(3) &= 7, & x_1^* &= 1 \end{aligned}$$

となります. ■

ここで取りあげた 0-1 ナップサック問題では品物 G_j がそれぞれ 1 個しか用意されていないとしましたが，いくつも用意されている場合も同様に考えることができます．これを定式化すれば，

$$\left|\begin{array}{llllllll} \text{最大化} & c_1x_1 & + & c_2x_2 & + & \cdots & + & c_nx_n \\ \text{条　件} & a_1x_1 & + & a_2x_2 & + & \cdots & + & a_nx_n \ \leq\ b \\ & & & x_j \in \mathbf{Z}_+, & & j = 1,\ 2,\ \ldots,\ n & & \end{array}\right. \tag{8.25}$$

となりますが，ここで $\mathbf{Z}_+$ は非負の整数全体の集合を表し，また係数はすべて正で条件 (8.18) を満たすものと仮定します．問題 (8.25) は「0-1」を付けずに単に**ナップサック問題**とよばれますが，これも漸化式

$$V_k(y) = \max\{V_{k-1}(y),\ V_k(y - a_k) + c_k\} \tag{8.26}$$

を用いて動的計画法で解くことができます (なぜか？ 演習問題 8.2).

ナップサック問題だけでなく，動的計画法は多くの**組合せ最適化問題** (combinatorial optimization problem) に適用できますが，問題規模が大きい場合には次に紹介する分枝限定法の方が効率的でしょう．また，より一般的な 0-1 整数計画問題 (8.19) を厳密に解く場合にも，現在のところ分枝限定法を用いるより手がないようです．それでは，分枝限定法のベースとなる単純列挙法から説明しましょう.

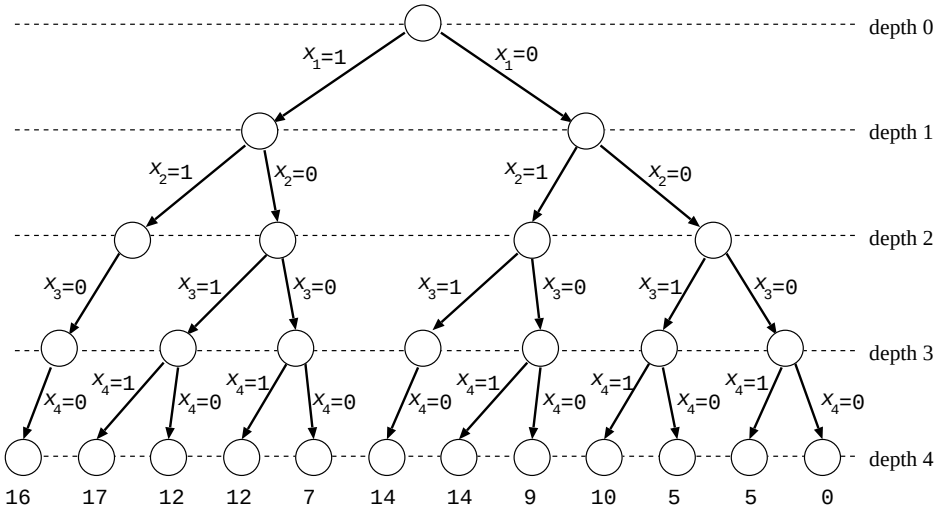

図 8.6: 単純列挙による分枝木

8.3 単純列挙法

0-1 ナップサック問題 (8.17) を解く最も素朴な方法は実行可能解を数え上げることでしょう．これを以下の手順で体系的に行い，例 8.1 の問題に適用してみましょう:

(a) 品物を G_1 から添字の小さい順にナップサックに詰める．

(b) ナップサックに品物が入らなければ，添字の最も大きい品物 G_k をナップサックから出し，G_{k+1} から再び添字の小さい順に詰める．

この手順に従って実行可能解を列挙していくと，図 8.6 に示す木が最上段の頂点を始点として，左上から右下に向かって**行きがけ順** (preorder) に生成されます．

生成された木を**分枝木** (branching tree) とよび，最下段に並んでいる葉はそれぞれ実行可能解に対応し，その下に付けられた数字が目的関数値を表します．したがって例 8.1 の問題には，12 の実行可能解が存在し，左から 2 番目の葉に対応する $\mathbf{x} = (1,\ 0,\ 1,\ 1)$ が最適値 17 を与えることがわかります．しかし，この方法では，すべての解の組合せである 2^n 個もの葉が生成される可能性があり，効率がよいとはいえません．最適解を与える見込みのない葉の生成を省略することはできないでしょうか ？

8.4 分枝限定法

分枝木を生成する過程で，一杯になったナップサックから添字の最も大きい G_k を取り出すところ，つまり手順 (b) の最初の状態を想定しましょう．分枝木の根の深さを 0 として深さ $k-1$ の頂点 (これを i とします) から枝分かれし，深さ n の葉まで枝を伸ばす操作を開始します．品物 G_k までの詰め方 $(x'_1,\ \ldots,\ x'_k)$ $(x'_k = 0)$ はそのままで，ナップサックの残りのスペースに G_{k+1} から品物を詰めていくわけですから，このとき生成される分枝木の葉は (8.17) と同じ形の問題:

$$\left|\begin{array}{ll} \text{最大化} & \displaystyle\sum_{j=k+1}^{n} c_j x_j + \sum_{j=1}^{k} c_j x'_j \\ \text{条　件} & \displaystyle\sum_{j=k+1}^{n} a_j x_j \le b - \sum_{j=1}^{k} a_j x'_j \\ & x_j \in \{0,\ 1\},\quad j = k+1,\ \ldots,\ n, \end{array}\right. \tag{8.27}$$

の最適値よりも大きな目的関数値を与えることはありません．この (8.27) を元の問題 (8.17) の**部分問題** (subproblem) と呼び，その最適解を $(x'_{k+1},\ \ldots\ x'_n)$ で表すことにします．

さて，この時点までに得られた実行可能解の中で $\mathbf{x}^* = (x^*_1,\ \ldots,\ x^*_n)$ が最も大きな目的関数値

$$V = \sum_{j=1}^{n} c_j x^*_j$$

を与えているものとしましょう．この $\mathbf{x}^*$ を問題 (8.17) の**暫定解** (incumbent) とよびますが，もしも (8.27) の最適解 $(x'_{k+1},\ \ldots,\ x'_n)$ が

$$\sum_{j=k+1}^{n} c_j x'_j + \sum_{j=1}^{k} c_j x'_j \le v \tag{8.28}$$

を満たせば，G_k までの詰め方を $(x'_1,\ \ldots,\ x'_k)$ に固定するかぎり，$\mathbf{x}^*$ よりも良い実行可能解は得られないことがわかります．したがって (8.28) が成り立てば，頂点 i から枝分かれして分枝木の葉を生成する作業は無駄です．ところが，部分問題 (8.27) 自体が 0-1 ナップサック問題であり，頂点 i からの枝分かれが無駄かどうかの判定に，いちいちこれを解くのは時間的

に高価です．そこで，(8.27) の連続緩和問題を考えます:

$$\left|\begin{array}{ll} \text{最大化} & \displaystyle\sum_{j=k+1}^{n} c_j x_j + \sum_{j=1}^{k} c_j x'_j \\ \text{条　件} & \displaystyle\sum_{j=k+1}^{n} a_j x_j \le b - \sum_{j=1}^{k} a_j x'_j \\ & 0 \le x_j \le 1, \quad j = k+1, \ \ldots, \ n. \end{array}\right. \tag{8.29}$$

第 8.1 節でも示したとおり，(8.29) は (8.27) よりはるかに簡単に解くことができます．その最適解を $(\overline{x}_{k+1}, \ \ldots, \ \overline{x}_n)$ で表すとき，

$$\sum_{j=k+1}^{n} c_j \overline{x}_j \ge \sum_{j=k+1}^{n} c_j x'_j$$

が成り立ちますから，もしも

$$L \equiv \sum_{j=k+1}^{n} c_j \overline{x}_j + \sum_{j=1}^{k} c_j x'_j \le V \tag{8.30}$$

が成り立てば，たとえ (8.27) を解いても $\mathbf{x}^*$ よりも良い解が得られることはなく，したがって頂点 i からの枝分かれを省略できます．この (8.30) を使って分枝木の枝生成を省く操作を**下界値** (lower bound) L による**限定操作** (bounding operation) といいます．

分枝限定法 (branch-and-bound method) は，限定操作と添字の最も大きな品物をナップサックから取り出すことに相当する**分枝操作** (branching operation) とによって構成されますが，これを例 8.1 の問題に適用した結果が図 8.7 の分枝木です．

algorithm BRANCH&BOUND

変数の添字を (8.21) の順にソートする; $V := 0$; $b' := b$; $k := 1$;

(a) : while $k \le n$ do begin

　if $a_j \le b'$ then begin $x'_j := 1$; $b' := b' - a_j$ end

　else $x'_j := 0$;

　$k := k + 1$

end;

if $\sum_{j=1}^{n} c_j x'_j > V$ then begin $V := \sum_{j=1}^{n} c_j x'_j$; $\mathbf{x}^* := (x'_1, \ \ldots, \ x'_n)$ end;

(b) : repeat $k := k - 1$ until $k < 1$ or $x'_k = 1$;

if $k \geq 1$ then begin

$x'_k := 0$; $b' := b' + a'_k$;

緩和問題 (8.29) の最適解 $(\overline{x}_{k+1}, \ldots, \overline{x}_n)$ を求める; $L := \sum_{j=k+1}^{n} c_j \overline{x}_j + \sum_{j=1}^{k} c_j x'_j$;

if $L \leq V$ then goto (b)

else begin $k := k + 1$; goto (a) end

end

end;

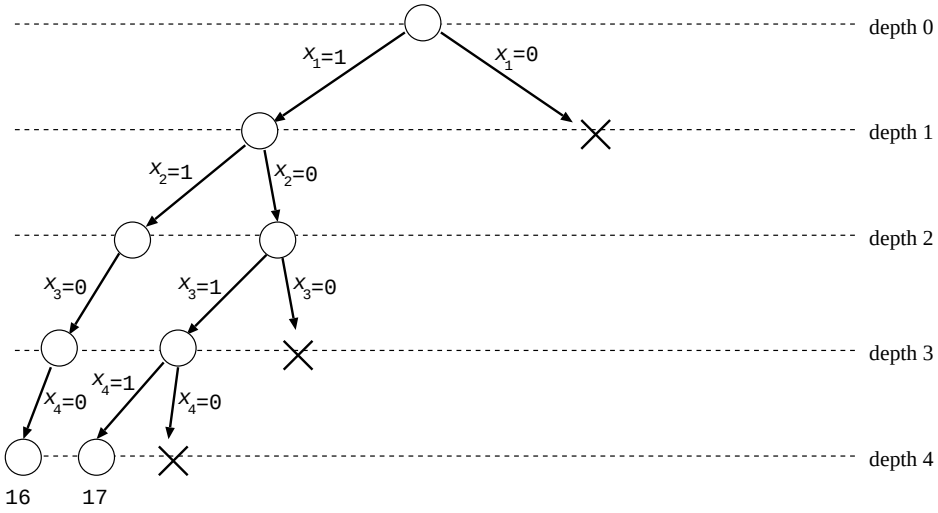

図 8.7: 分枝限定法による分枝木

演習問題

8.1 0-1 ナップサック問題 (8.17) の最適解 $\mathbf{x}^*$ と，その連続緩和問題の最適解 $\overline{\mathbf{x}}$ の間に (8.22) の関係が成り立つことを，２つの問題の実行可能領域の包含関係に着目して証明しなさい．

8.2 容積 y のナップサックに品物 $G_1, \ldots, G_k$ を詰める場合の価値の総和の最大値を $f(k,\ y)$ で表したとき，

$$f(k,\ y) = \max\{f(k-1,\ y),\ f(k-1,\ y-a_k) + c_k\} \tag{8.31}$$

の関係が成り立つ．0-1 ナップサック問題の最適値は，

$$\begin{cases} f(k,\ y) = -\infty, & y < 0 \text{ のとき} \\ f(0,\ y) = 0, & y \geq 0 \text{ のとき}, \end{cases}$$

と定めて動的計画法を使えば，$f(n,\ b)$ によって与えられる．

(a) 関係式 (8.31) を説明しなさい．

(b) 動的計画法を使って例 8.1 の問題を解きなさい．

8.3 0-1 ナップサック問題では各品物 G_j は 1 つしか存在しなかったが，複数存在する問題を単に**ナップサック問題**とよび，以下のように定式化される:

$$\left|\begin{array}{lllllllll} \text{最大化} & c_1x_1 & + & c_2x_2 & + & \cdots & + & c_nx_n & \\ \text{条　件} & a_1x_1 & + & a_2x_2 & + & \cdots & + & a_nx_n & \leq\ b \\ & & & 0 \leq x_j \leq u_j, & & x_j \in \mathbf{Z}, & & j = 1,\ 2,\ \ldots,\ n. & \end{array}\right.$$

ただし，$\mathbf{Z}$ は整数全体の集合を表し，u_j は品物 G_j の個数を示す正の整数である．この問題を，動的計画法，あるいは分枝限定法によって解く方法を工夫しなさい．

8.4 土浦から上野まで最も割高に行く方法を提案しなさい．