03

第3章

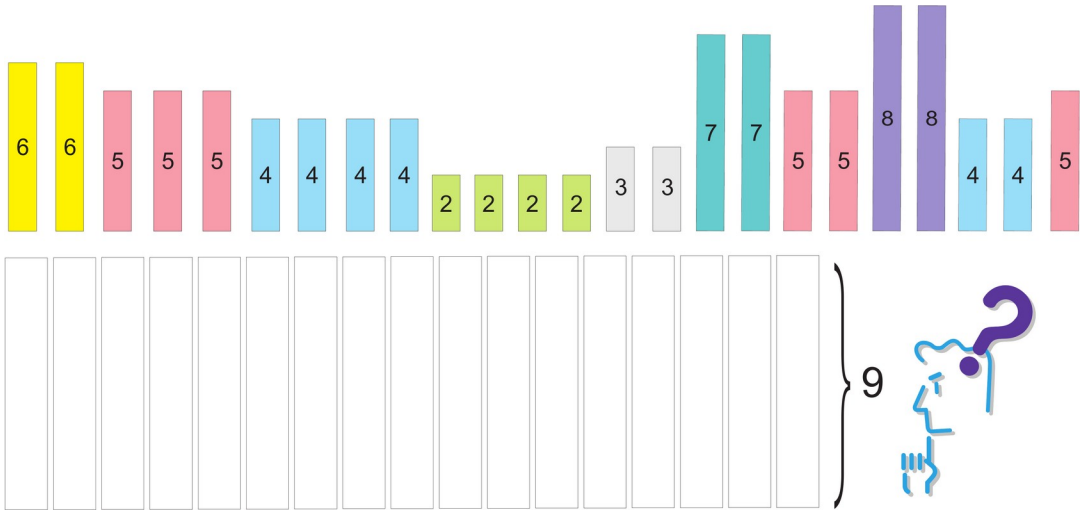
箱詰め問題と切断問題

箱詰め問題の定式化と、切断問題に対する列生成法

この章の内容

1. 箱詰め問題と切断問題
2. 箱詰め問題の定式化
3. 切断問題に対する列生成法

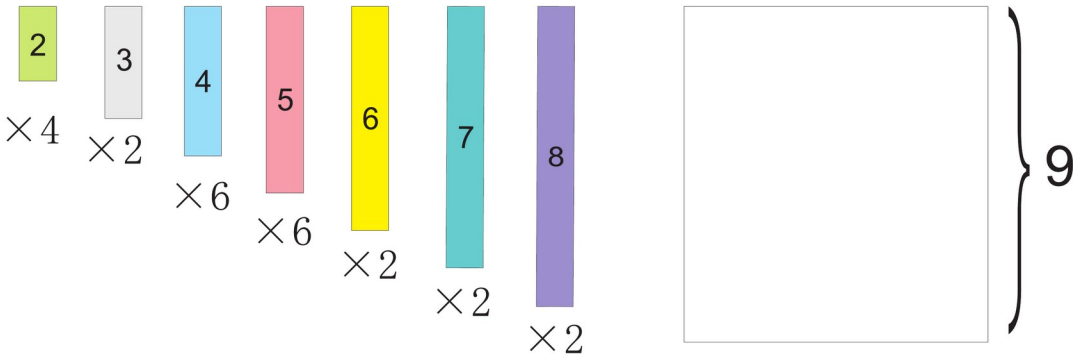
例題：荷物の箱詰め



箱詰め問題の問題例

- 1つの箱に詰められるのは 9 kg まで
- 24 個の荷物の重さは分かっている
- 容積は気にしなくてよい
- 使う箱の数をなるべく少なくしたい

例題：原紙からの切り出し



切断問題の問題例

- 原紙の幅は 9 メートル
- 注文は 7 件
  - 幅 2,3,4,5,6,7,8 メートル
  - 枚数 4,2,6,6,2,2,2 枚
- 使う原紙の枚数を最小にしたい

箱詰め問題と切断問題

箱詰め問題（ビンパッキング）

- n 個のアイテム
- アイテム i のサイズ s[i]
- サイズ B のビンが無限個
- 必要なビンの数を最小化

切断問題

- m 種類の注文
- 注文 i の幅 w[i] と注文数 q[i]
- 幅 B の原紙
- 必要な原紙の数を最小化

3.1 箱詰め問題の定式化

アイテムをどのビンに入れるか、ビンを使うか否かを0-1変数で表す

箱詰め問題の定式化

X[i,j]：アイテム i をビン j に詰めるとき 1、Y[j]：ビン j を使うとき 1

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j=1}^{U} Y_j \\
\text{subject to} \quad & \sum_{j=1}^{U} X_{ij} = 1 && \forall i = 1, \ldots, n \\
& \sum_{i=1}^{n} s_i X_{ij} \leq B Y_j && \forall j = 1, \ldots, U \\
& X_{ij} \leq Y_j && \forall i, j \\
& X_{ij} \in \{0, 1\} && \forall i, j \\
& Y_j \in \{0, 1\} && \forall j = 1, \ldots, U
\end{aligned}
$$

- 目的：使うビンの数の最小化
- 第1制約：各アイテムをいずれかのビンへ
- 第2制約：ビンのサイズ上限と、未使用ビンには入れられないこと
- 第3制約：定式化の強化

ビン数の上限：FFD

bpp.py

```
def FFD(s, B):
    remain = [B]
    sol = [[]]
    for item in sorted(s, reverse=True):
        for (j,free) in enumerate(remain):
            if free >= item:
                remain[j] -= item
                sol[j].append(item)
                break
        else:
            sol.append([item])
            remain.append(B-item)
    return sol
```

- 大きいアイテムから順に、入る最初のビンに入れる
- remain：各ビンの残りサイズ
- 入るビンがなければ新しいビンを作る
- len(sol) がビン数の上限 U

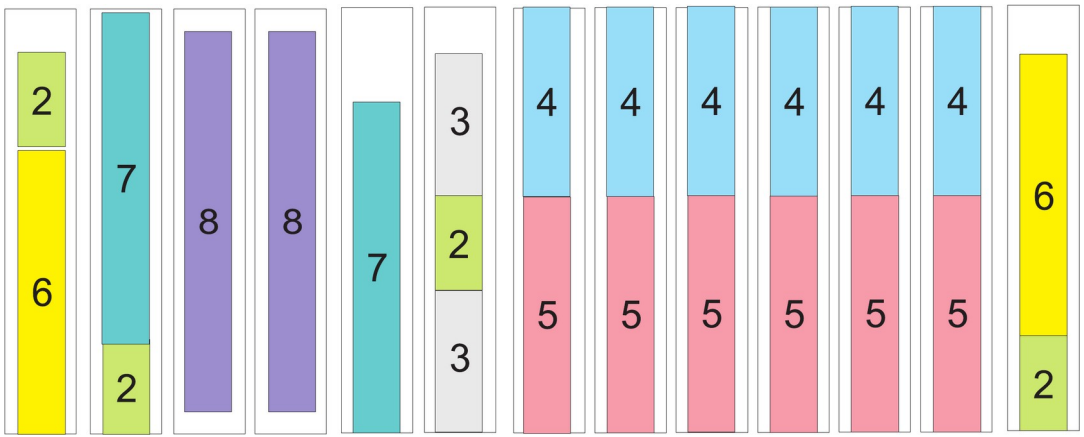
AMPLによるモデル

bpp.mod

```
set I;                    # アイテムの集合
set J;                    # ビンの集合

param s {I} > 0;          # アイテム i のサイズ
param B > 0;              # ビンのサイズ

var x {I,J} binary;
var y {J} binary;

minimize Bins:
   sum {j in J} y[j];

subject to Assign {i in I}:
   sum {j in J} x[i,j] = 1;

subject to Capacity {j in J}:
   sum {i in I} s[i] * x[i,j] <= B * y[j];

subject to Strong {i in I, j in J}:
   x[i,j] <= y[j];
```

- J は FFD で得たビン数 U から作る
- リスト s は辞書に直して渡す
  - {i: s[i] for i in range(n)}
- パラメータは集合の要素で添字付け

箱詰め問題の最適解



箱詰め問題の最適解

- HiGHS で解くと、ビン数 13 の解が得られる
- データは切断問題の例題から作った

3.2 切断問題に対する列生成法

必要な列（変数）だけを、双対変数の情報を使って順次追加する

列生成法の考え方

- 制約の係数行列で、行は制約、列は変数に対応
- 多くの変数を含む問題で、必要な列を順次追加する
- 原紙の幅を超えない切り出し方を切断パターンと呼ぶ
- 初期パターン：1つの注文をできるだけ多く切る
  - 注文1なら $\lfloor 9/2 \rfloor = 4$ 枚 → $(4,0,0,0,0,0,0)$

! モデリングのコツ

変数の数が非常に多いときには列生成法を使え。子問題を解いて必要なパターンだけを生成する。

初期切断パターンと線形緩和の解

初期パターンだけで解いた線形緩和の最適値は $16\ 2/3$

パターン	切り出す注文	枚数	最適解 x	双対変数 λ
1	注文1（幅2）	4	1	1/4
2	注文2（幅3）	3	2/3	1/3
3	注文3（幅4）	2	3	1/2
4〜7	注文4〜7（幅5〜8）	1	6, 2, 2, 2	1

λ は各注文の価値。$\lambda_1 = 1/4$ は「注文1の価値は原紙の $1/4$」と解釈できる

子問題：整数ナップサック問題

原紙の幅を超えない範囲で、最も価値の高いパターンを求める（y_i：注文 i を切り出す数）

$$
\begin{aligned}
\text{maximize} \quad & \tfrac{1}{4}y_1 + \tfrac{1}{3}y_2 + \tfrac{1}{2}y_3 + y_4 + y_5 + y_6 + y_7 \\
\text{subject to} \quad & 2y_1 + 3y_2 + 4y_3 + 5y_4 + 6y_5 + 7y_6 + 8y_7 \le 9 \\
& y_1, \dots, y_7 \text{ は非負の整数}
\end{aligned}
$$

- 最適解 y=(2,0,0,1,0,0,0)、最適値 1.5
- 被約費用は 1−(2λ1+λ4) = −0.5
- この列を追加すると原紙 0.5 枚分の利益
- 被約費用がすべて非負になるまで繰り返す

主問題と子問題（一般形）

t_i^k：k 番目の切断パターンで注文 i を切り出す数、x_k：パターン k の使用回数

$$
\begin{aligned}
&\text{主問題} \quad \text{minimize} && \sum_{k=1}^{K} x_k \\
&\qquad\quad \text{subject to} && \sum_{k=1}^{K} t_i^k x_k \geq q_i \quad \forall i = 1, \ldots, m \\
&&& x_k \text{ は非負の整数} \quad \forall k = 1, \ldots, K \\
&\text{子問題} \quad \text{maximize} && \sum_{i=1}^{m} \lambda_i y_i \\
&\qquad\quad \text{subject to} && \sum_{i=1}^{m} w_i y_i \leq B \\
&&& y_i \text{ は非負の整数} \quad \forall i = 1, \ldots, m
\end{aligned}
$$

- λ：主問題の線形緩和の最適双対変数
- 子問題は被約費用が負の列を求める
- 実行可能なパターンは Σ w[i] t[i] ≤ B を満たす

主問題のモデル

csp_master.mod

```
set ORDERS;                  # 注文の種類
param nPat > 0 integer;      # 現在のパターン数
set PATTERNS = 1..nPat;      # 切断パターンの集合

param q {ORDERS} >= 0;       # 注文 i の注文数
param t {PATTERNS, ORDERS} >= 0 default 0;

var x {PATTERNS} integer >= 0;    # 使用回数

minimize Rolls:
   sum {k in PATTERNS} x[k];

subject to Orders {i in ORDERS}:
   sum {k in PATTERNS} t[k,i] * x[k] >= q[i];
```

- nPat を 1 増やすだけで集合が伸びる
- default 0：値を与えない要素は 0
  - 新パターンの 0 をいちいち書かなくてよい

子問題のモデル

knapsack.mod

```
set ORDERS;
param B > 0;                          # 原紙の幅
param w {ORDERS} > 0;                 # 注文 i の幅
param q {ORDERS} >= 0;                # 注文 i の注文数
param price {ORDERS} >= 0 default 0;   # 双対変数

var y {i in ORDERS} integer >= 0, <= q[i];

maximize Reduced:
   sum {i in ORDERS} price[i] * y[i];

subject to Width:
   sum {i in ORDERS} w[i] * y[i] <= B;
```

- 双対変数 λ は子問題から見れば定数
  - パラメータ price として宣言
- y の上限に注文数 q[i] を付ける
  - 注文数より多く切っても意味がない

メインループの要点

主問題と子問題の間で双対情報と新しい列をやりとりする

1 線形緩和を解く

relax_integrality を 1 にすると、整数条件を外して主問題を解ける

2 双対変数を取り出す

制約 Orders の dual を price として子問題に渡す

3 終了判定

ナップサック問題の最適値が 1 より小さければ、被約費用がすべて非負なので終了

4 列を追加する

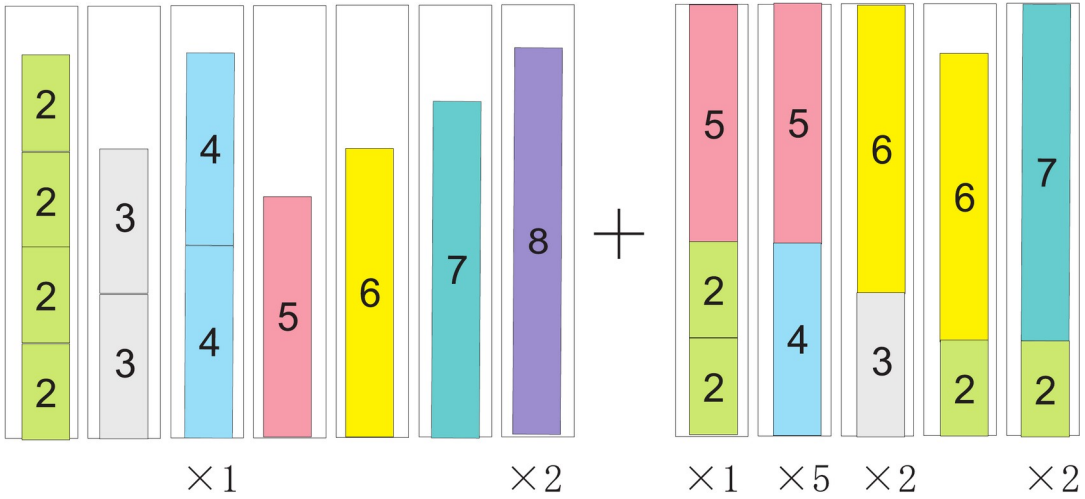
nPat を 1 増やし、新しいパターンの中身だけを t に追加する

列生成法のメインループ

csp.py

```
while True:
    master.option["relax_integrality"] = 1
    master.solve(verbose=False)
    assert master.solve_result == "solved"
    duals = master.get_constraint("Orders").get_values().to_pandas()
    price = {i: duals.loc[i, "Orders.dual"] for i in ORDERS}

    knapsack.param["price"] = price
    knapsack.solve(verbose=False)
    assert knapsack.solve_result == "solved"
    if knapsack.get_objective("Reduced").value() < 1 + EPS:
        break

    ys = knapsack.get_variable("y").get_values().to_pandas()
    pat = [int(ys.loc[i, "y.val"] + 0.5) for i in ORDERS]
    t.append(pat)
    K += 1
    master.param["nPat"] = K
    master.param["t"] = {(K, i): pat[i] for i in ORDERS}
```

生成されたパターンと最適解



- 5 つのパターンを追加して列生成が終了
- 整数条件を付けて解くと原紙 13 枚
- 箱詰め問題として解いた結果と一致
- 生成される列はソルバーによって変わりうる

左側が初期パターン、右側が生成されたパターン、数字は使用回数

まとめ

- ✓ 箱詰め問題と切断問題は同じ問題である
- ✓ 箱詰め問題は0-1変数 X, Y で定式化し、FFD で U を得る
- ✓ 切断問題は切断パターンを列とする主問題で表す
- ✓ 子問題の整数ナップサック問題で新しい列を生成する
- ✓ AMPL では nPat を増やすだけで列を追加できる
- ✓ 例題の最適値は原紙 13 枚