数理計画法 第6回

塩浦昭義

情報科学研究科 准教授

shioura@dais.is.tohoku.ac.jp

http://www.dais.is.tohoku.ac.jp/~shioura/teaching

第5章 組合せ計画

§5.2 分枝限定法

組合せ計画問題

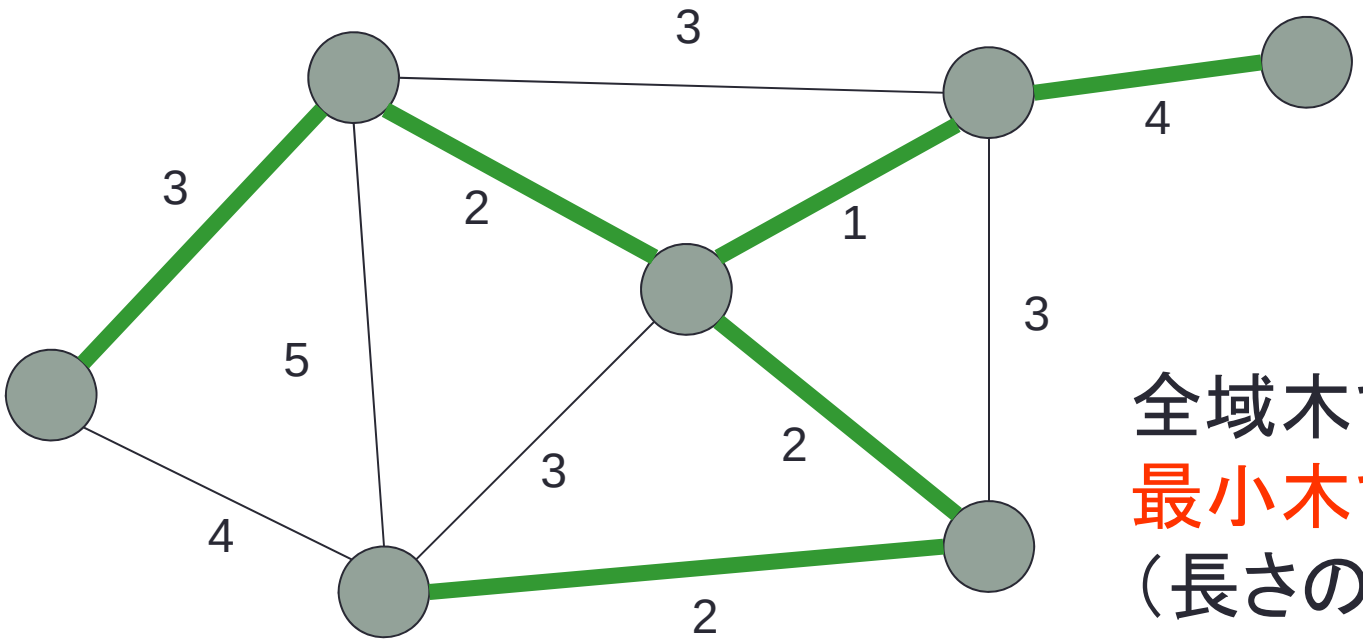
- 組合せ計画問題とは:
  - 有限個の「もの」の組合せの中から，目的関数を最小または最大にする組合せを見つける問題
  - 例1:整数計画問題全般（整数の組合せ）
  - 例2:グラフの最小木問題，最短路問題，（グラフの枝の組合せ）
  - 例3:巡回セールスマン問題（都市の順列）

- 解きやすい問題と解きにくい問題
  - 解きやすい問題≒多項式時間で解ける問題
  - 解きにくい問題≒NP困難な問題
    （多項式時間で解けないと信じられている問題）

※注意:組合せ計画問題の解は有限個➔有限時間で必ず解ける！

最小木問題

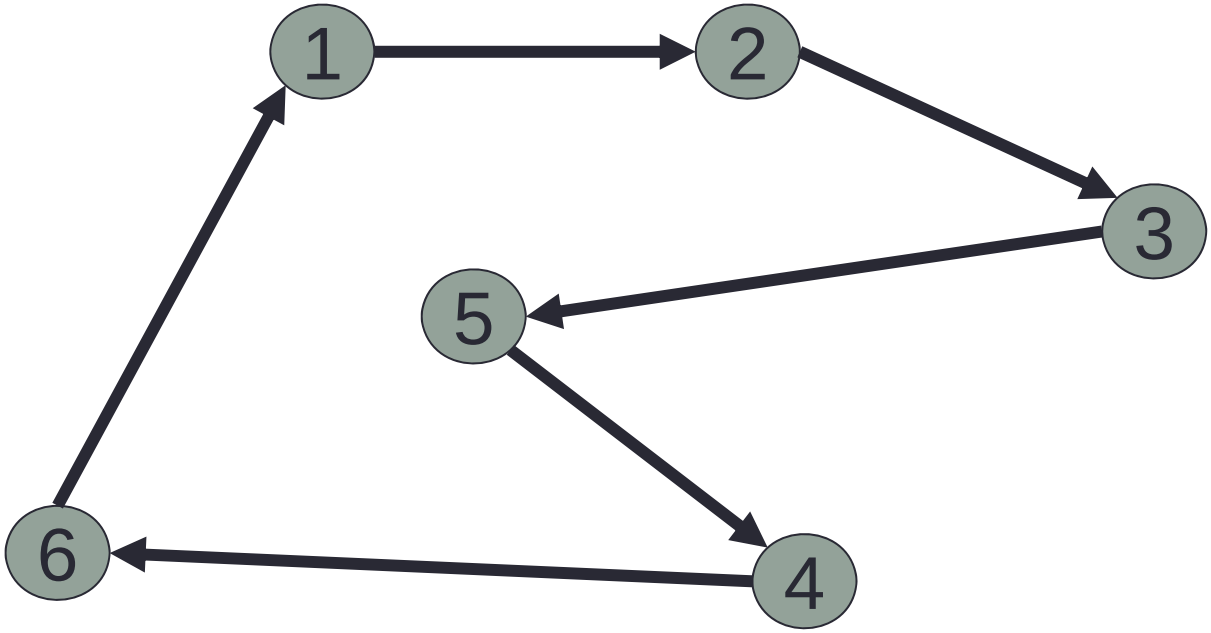
- 入力：無向グラフG=(V,E)，各枝の長さd(e)（e∈E）
- 出力：Gの最小木（Gの全域木で，枝の長さの和が最小のもの）



全域木であり，
最小木である
（長さの和＝14）

巡回セールスマン問題

- セールスマンが n 都市をちょうど一回ずつ巡回する
- 都市 i から j への距離は c_{ij}（平面上の距離で与えられるケースも多い）
- 目的：都市を巡回する際の総距離を最小にする



組合せ計画問題に対するアプローチ

- 組合せ計画問題をどのように解くか？
- 解きやすい問題の場合
  - 多項式時間アルゴリズムを構築（教科書§5.1）➔より高速な解法へ
- 解きにくい問題の場合
  - 絶対に最適解が必要な場合➔厳密解法
    - 分枝限定法（§5.2，授業で説明）←現在の主流
    - 動的計画法（§5.3，「アルゴリズムとデータ構造」）
  - ある程度良い解であれば十分という場合
    - 精度保証付き近似アルゴリズム
      （解の良さに対する理論保証あり）
    - ヒューリスティックス（解の良さは実験的に証明）（§5.4, 5.5）

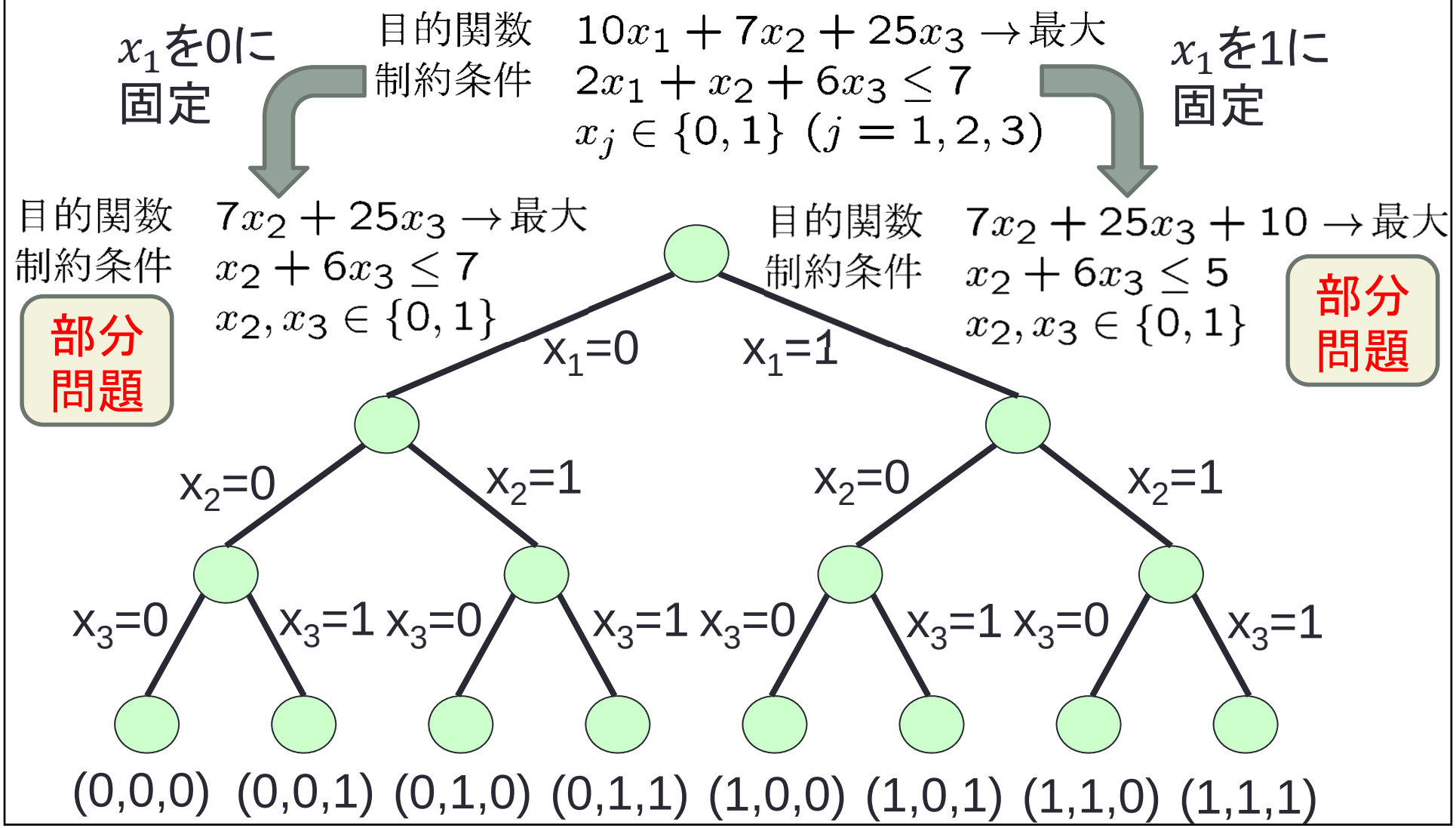
組合せ計画問題に対する厳密解法

- 組合せ計画問題は解を全列挙すれば解ける
- しかし，計算時間が膨大で現実には不可能
  - ➔ 解の全列挙における無駄を出来るだけ省く
  - 動的計画法：同一の部分問題を繰り返し解かない
  - 分枝限定法：ある部分問題から最適解が得られないことがわかったら，その部分問題は無視する

分枝限定法の考え方

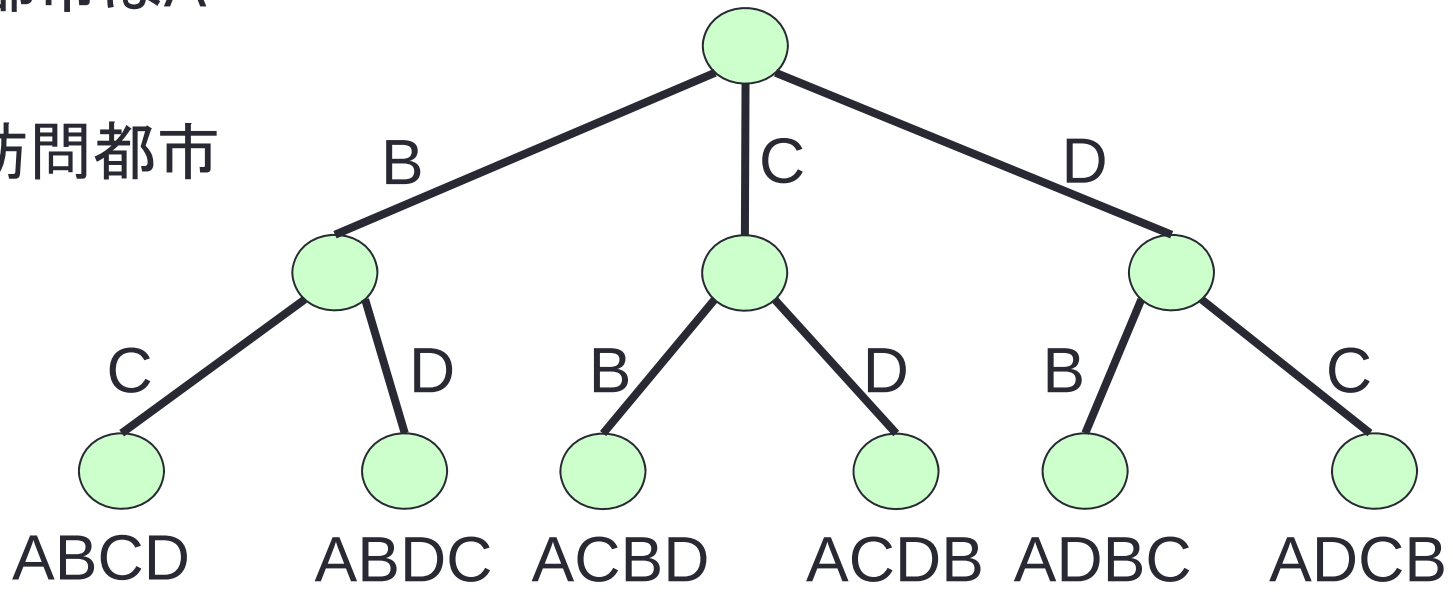
- 組合せ計画問題を，場合分けによって部分問題に分解（分枝操作）
  - 0–1ナップサック問題：各変数について 0 の場合と 1 の場合に分ける
  - 巡回セールスマン問題：次に訪問する都市によって場合分け
- 分枝の進行の様子は探索木により表現可能

0-1ナップサック問題の探索木



巡回セールスマン問題の探索木

- 4都市{A,B,C,D}の対称巡回セールスマン問題の場合
- 最初の訪問都市はA



分枝限定法の考え方

- 分枝操作により，たくさんの部分問題が生成される
- 解く必要のない（解いても無駄な）部分問題が検出されたら，
  さらなる分枝操作をストップ（限定操作）
- 暫定解の保持と緩和問題の利用により，無駄をチェック
  - 暫定解：分枝限定法のそれまでの計算により
    得られている最良の許容解
  - 緩和問題：元の数理計画問題の制約条件の一部を緩和して
    得られる問題
- 解く必要のない部分問題の例
  - 最適解がすでに得られた
  - 現在の暫定解より良い許容解を得られる可能性がなくなった
  - 許容解が存在しない（実行不可能）

緩和問題

- 緩和問題：元の数理計画問題の制約条件の一部を緩和して得られる問題

目的関数：$\sum_{i=1}^{n} c_i x_i$ → 最大
制約条件：$\sum_{i=1}^{n} a_i x_i \leq b$
$x_1, x_2, \ldots, x_n \in \{0,1\}$

0-1ナップサック問題

$x_j \in \{0,1\}$を$0 \leq x_j \leq 1$に緩和

目的関数：$\sum_{i=1}^{n} c_i x_i$ → 最大
制約条件：$\sum_{i=1}^{n} a_i x_i \leq b$
$0 \leq x_j \leq 1 \ (j = 1,2, \ldots, n)$

連続ナップサック問題

連続ナップサック問題は線形計画問題➔多項式時間で解ける
O(n)時間アルゴリズムが存在（「アルゴリズムとデータ構造」）

緩和問題の性質

- 緩和問題は元の問題より解きやすい（簡単に解ける）ことが多い
  - 通常，簡単に解ける問題を緩和問題として選ぶ
- 緩和問題は元の問題の条件を緩和した問題

  ➔緩和問題の実行可能解集合は，元の問題の実行可能解集合を含む

  ➔緩和問題の最大値≧元の問題の最大値

  ∴緩和問題の最適値（最大値）は，元の問題の最適値の上界
- 緩和問題の最適解を修正することにより，元の問題の実行可能解を作ることが可能なケースが多い
  - 緩和問題の最適解が元の問題の実行可能解

    ➔元の問題の最適解になっている！

0-1ナップサック問題の緩和問題：例

緩和問題の最適解は，c_j/a_j の大きい方から順に変数を増やしていけば得られる

	1	2	3	4	5	6	7	8
c_j	15	100	90	60	40	15	10	1
a_j	2	20	20	30	40	30	60	10

b=102

a_jの合計72≦b

(1,1,1,1,(102-72)/40,0,0,0)は最適解
目的関数値＝295
≧0-1ナップサック問題の最適値

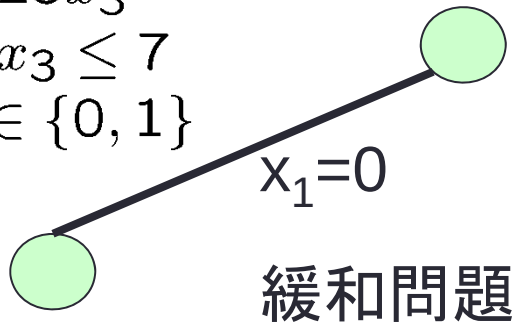
$x_5 = 0$に置き換えた解(1,1,1,1,0,0,0,0)は元問題の実行可能解
目的関数値＝265
$x_4 = 0$にして$x_5 = 1$とした解(1,1,1,0,1,0,0,0)も元問題の実行可能解
目的関数値＝245

解く必要のない部分問題の例：最適解が得られた部分問題

$$\begin{aligned} &\text{最大化} && 10x_1 + 7x_2 + 25x_3 \\ &\text{条件} && 2x_1 + x_2 + 6x_3 \leq 7 \\ & && x_j \in \{0,1\}\ (j = 1,2,3) \end{aligned}$$

$$\begin{aligned} &\text{最大化} && 7x_2 + 25x_3 \\ &\text{条件} && x_2 + 6x_3 \leq 7 \\ & && x_2, x_3 \in \{0,1\} \end{aligned}$$

緩和



$$\begin{aligned} &\text{最大化} && 7x_2 + 25x_3 \\ &\text{条件} && x_2 + 6x_3 \leq 7 \\ & && 0 \leq x_2, x_3 \leq 1 \end{aligned}$$

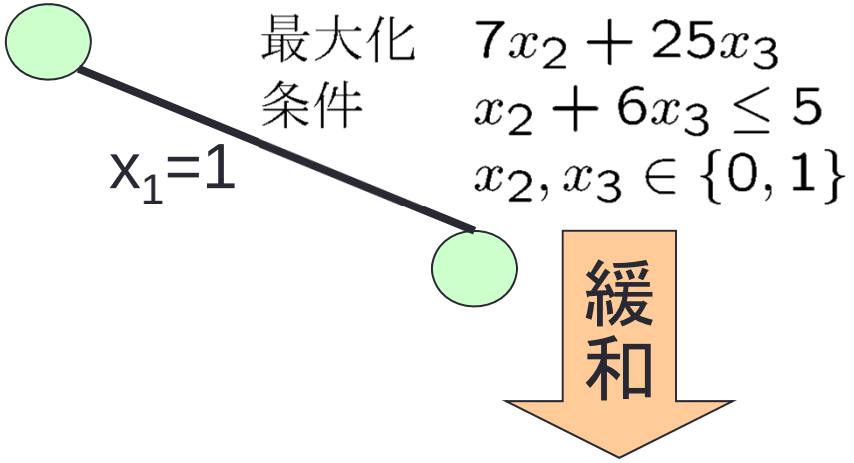
緩和問題の最適解は
$(x_2, x_3) = (1, 1)$（整数解）
➔元の**部分問題の最適解**

この部分問題をさらに調べても，
より良い解は得られない
➔この部分問題の**探索をストップ**

$(x_1,x_2,x_3)=(0,1,1)$ は暫定解，目的関数値＝32

解く必要のない部分問題の例：より良い解が得られない部分問題

$(x_1,x_2,x_3)=(0,1,1)$
は暫定解，
目的関数値＝32



最大化　$7x_2 + 25x_3$
条件　$x_2 + 6x_3 \leq 5$
$x_2, x_3 \in \{0, 1\}$

最大化　$7x_2 + 25x_3$
条件　$x_2 + 6x_3 \leq 5$
$0 \leq x_2, x_3 \leq 1$

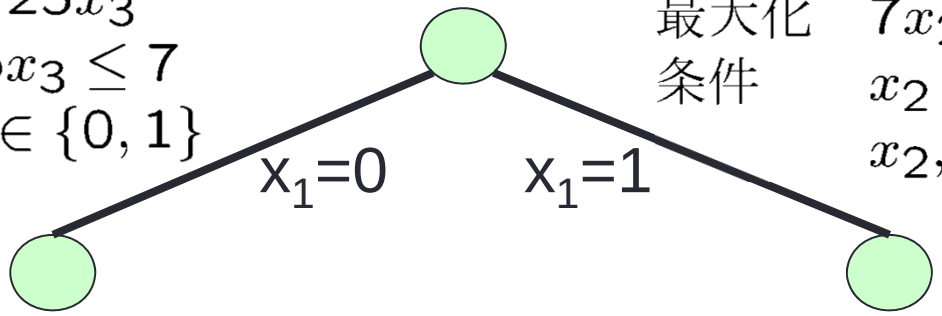
緩和問題の最適解は
$(x_2, x_3) = (1, 2/3)$
目的関数値＝23.6666…
これは部分問題の上界値，
　≦ 暫定解の目的関数値32

この部分問題をさらに調べても，
暫定解より良い解は得られない
➔この部分問題の探索をストップ

解く必要のない部分問題の例: 実行不可能な部分問題

最大化 $10x_1 + 7x_2 + 25x_3$
条件 $8x_1 + x_2 + 6x_3 \leq 7$
$x_1, x_2, x_3 \in \{0, 1\}$

最大化 $7x_2 + 25x_3$
条件 $x_2 + 6x_3 \leq 7$
$x_2, x_3 \in \{0, 1\}$



最大化 $7x_2 + 25x_3$
条件 $x_2 + 6x_3 \leq -1$
$x_2, x_3 \in \{0, 1\}$

この部分問題は実行不可能
➔この部分問題の探索をストップ

分枝限定法の流れ

記号 L: 部分問題のリスト,
　　x*: 暫定解, z: 暫定値（暫定解の目的関数値）
ステップ0：L={元問題}, z=－∞, x*は未定義とする.
ステップ1（探索）:
　　Lが空ならば計算終了. 現在の x* が最適解.
　　Lが非空ならば, Lから部分問題P'を選び, 削除.
ステップ2（限定操作）:
　2-a: P' が実行不可能であることがわかったら, ステップ1へ.
　2-b: P'の最適解が得られたら, 必要に応じて x*, z を更新して
　　ステップ1へ.
　2-c: P' の緩和問題を解いた結果, 暫定解より良い許容解が
　　得られないことがわかったらステップ1へ.

分枝限定法の流れ

ステップ2（限定操作）:

2−a: P’ が実行不可能であることがわかったら，ステップ1へ.

2−b: P’の最適解が得られたら，必要に応じて x^*, z を更新して
ステップ1へ.

2−c: P’ の緩和問題を解いた結果，暫定解より良い許容解が
得られないことがわかったらステップ1へ.

ステップ3（分枝操作）:

P’を場合分けによってP'_1, P'_2, …, P'_kに分解.

L にこれらの問題を入れ，ステップ1へ.

分枝限定法の実装における検討事項

分枝限定法の性能は，各々のステップを
如何に実現するかによって左右される

どのような順番で
部分問題を選ぶか？
（部分問題の探索法）

ステップ１（探索）：
Lが空ならば計算終了．現在の x* が最適解．
Lが非空ならば，Lから部分問題P’を選び，削除．

ステップ２（限定操作）：
2−a: P’ が実行不可能であることがわか
2−b: P’の最適解が得られたら，必要に応じて x*, z* を更新して
ステップ１へ．
2−c: P’ の緩和問題を解いた結果，暫定
得られないことがわかったらステップ

どのように実行不可能
性を判定するか？

どのような緩和問題を
解くか？

ステップ３（分枝操作）：
P’を場合分けによってP’$_1$, P’$_2$, …, P’$_k$に分解．
L にこれらの問題を入れ，ステップ１へ．

どのように問題を
分解するか？

分枝限定法の実装における検討事項

- 部分問題の探索法
  - どのような順番で部分問題を選ぶか？
- 限定操作のやり方
  - どのように実行不可能性を判定するか？
  - どのような緩和問題を解くか？
- 分枝操作のやり方
  - どのように問題を分解するか？

部分問題の探索法

- 最良優先探索と深さ優先探索が一般的
- 最良優先探索
  - 最も良い許容解が得られそうな部分問題を優先的に選ぶ
  - 緩和問題から得られる上界値などを部分問題の評価値として使用する
  - 利点：計算終了までに解く部分問題の数が少ない
    ➔ 計算時間が少ない
- 深さ優先探索
  - 部分問題の深さがより大きい問題を優先的に選ぶ
  - 利点：実装が簡単，メモリ使用量が少ない，
    暫定解を早く得やすい

深さ優先探索を用いた分枝限定法の例

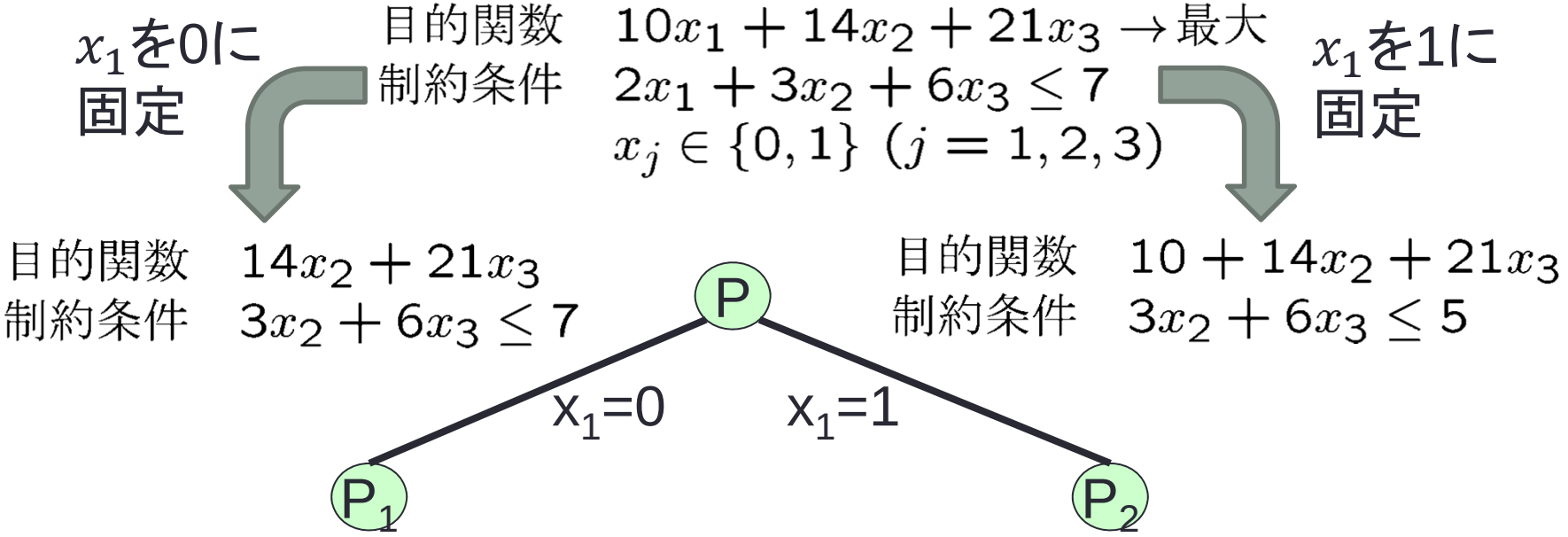
0-1ナップサック問題を例として:

初期の暫定解　x*=未定義，　暫定解の目的関数値＝－ ∞ とおく.

解いていない部分問題は元の問題Pのみ➔Pを選ぶ

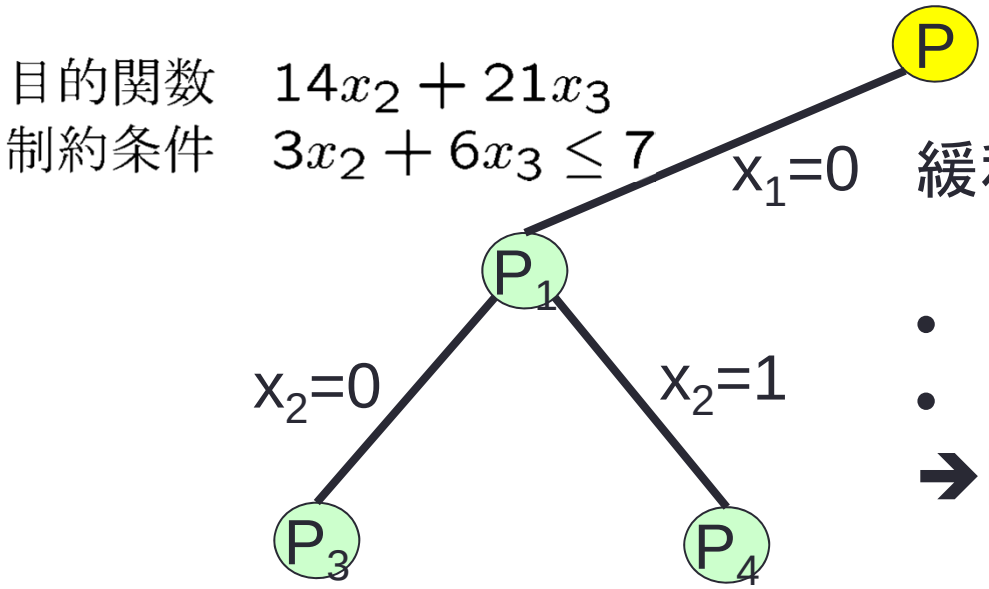
変数x_1を選択，0に固定した部分問題P_1

および 1に固定した部分問題P_2を作る.



深さ優先探索を用いた分枝限定法の例

解いていない部分問題はP_1, P_2, 2つとも深さは同じ
➔ 0に固定した部分問題P_1を先に選ぶ



緩和問題の最適解

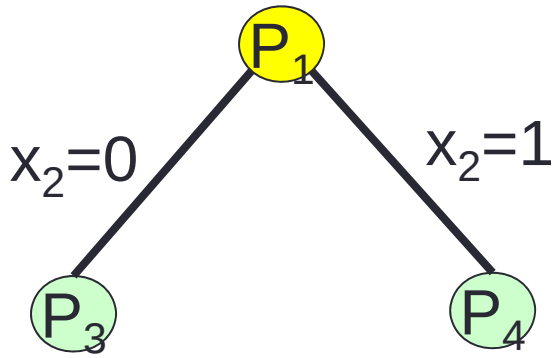
$$(x_1, x_2, x_3) = (0,1,2/3)$$

- 元の問題の実行可能解ではない
- 目的関数値は暫定解より大きい

➔限定操作はできないので，
分枝操作を行う

深さ優先探索を用いた分枝限定法の例

解いていない部分問題はP_2, P_3, P_4
➔深さ優先探索を使っているので，もっとも深いところにある問題 P_3, P_4 を優先的に選ぶ
今回は 0に固定した部分問題P_3を先に選ぶ

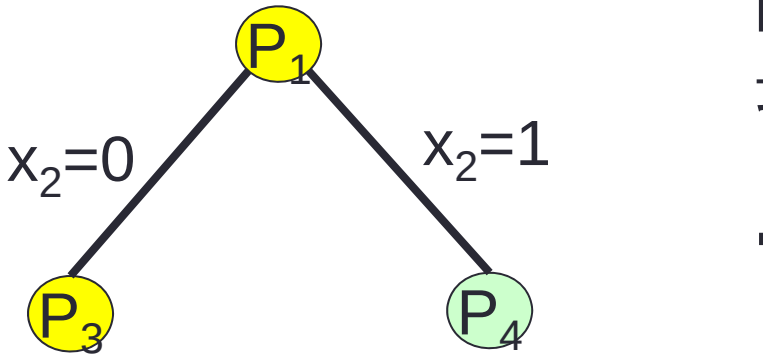


目的関数　$21x_3$
制約条件　$6x_3 \leq 7$

P_3は変数1個なので簡単に解ける
最適解は $(x_1, x_2, x_3) = (0,0,1)$
目的関数値＝21
➔暫定解$(x_1^*, x_2^*, x_3^*) = (0,0,1)$
　とする

深さ優先探索を用いた分枝限定法の例

暫定解$(x_1^*, x_2^*, x_3^*) = (0,0,1)$，目的関数値＝21
解いていない部分問題はP_2, P_4
➔深さ優先探索を使っているので，もっとも深いところにある問題P_4を優先的に選ぶ

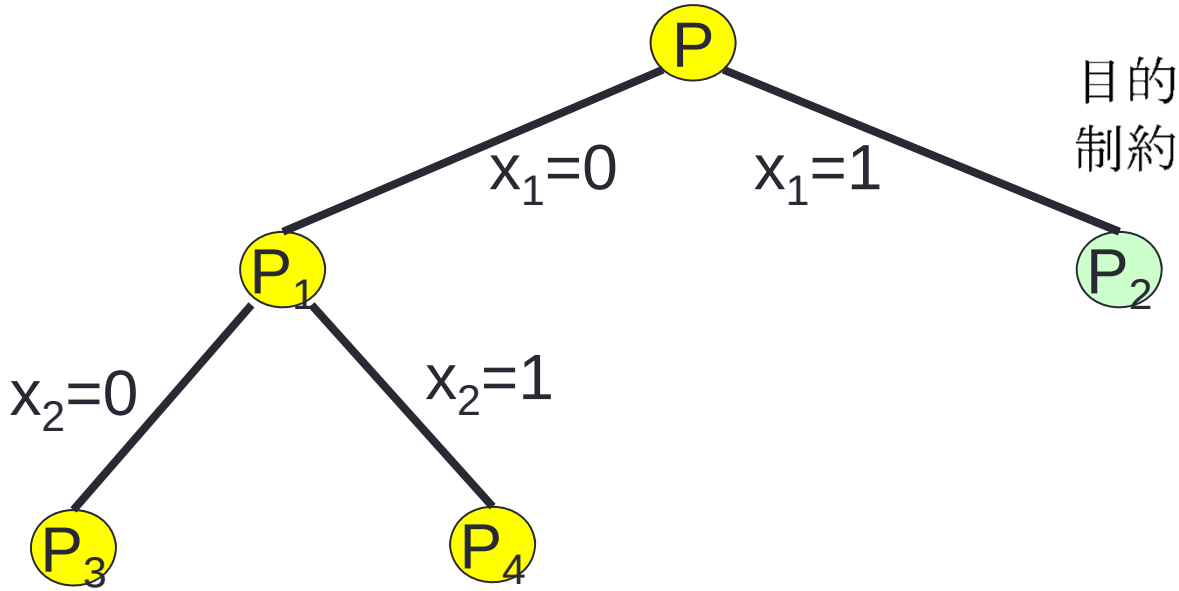


目的関数　$14 + 21x_3$
制約条件　$6x_3 \leq 4$

P_4は変数1個なので簡単に解ける
最適解は $(x_1, x_2, x_3) = (0,1,0)$
目的関数値＝14
➔暫定解の目的関数値より悪いので
　暫定解は変更しない

深さ優先探索を用いた分枝限定法の例

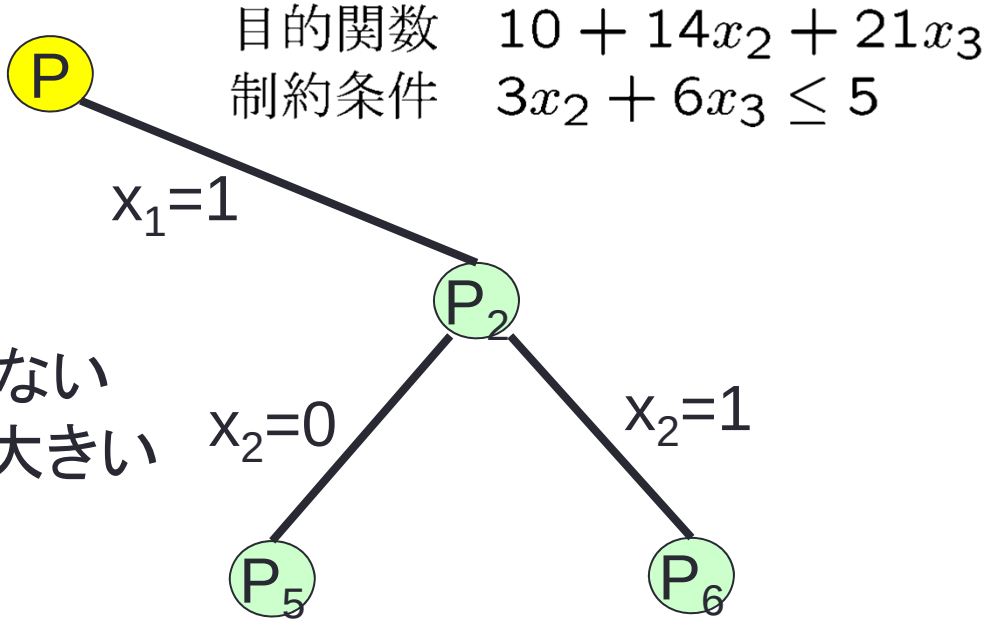
暫定解$(x_1^*, x_2^*, x_3^*) = (0,0,1)$，目的関数値＝21
解いていない部分問題はP_2 ➔P_2を選ぶ



目的関数　$10 + 14x_2 + 21x_3$
制約条件　$3x_2 + 6x_3 \leq 5$

深さ優先探索を用いた分枝限定法の例

暫定解$(x_1^*, x_2^*, x_3^*) = (0,0,1)$, 目的関数値＝21
解いていない部分問題はP_2 ➔P_2を選ぶ



緩和問題の最適解

$$(x_1, x_2, x_3) = (1,1,1/3)$$

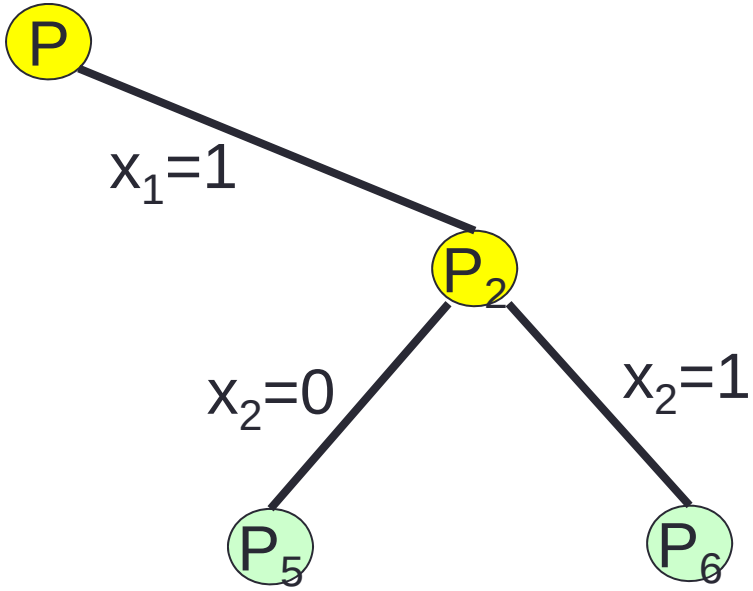
- 元の問題の実行可能解ではない
- 目的関数値31は暫定解より大きい

➔限定操作はできないので,
分枝操作を行う

深さ優先探索を用いた分枝限定法の例

暫定解$(x_1^*, x_2^*, x_3^*) = (0,0,1)$，目的関数値＝21
解いていない部分問題はP_5, P_6 ➔P_5を選ぶ

P_5は変数1個なので簡単に解ける
最適解は $(x_1, x_2, x_3) = (1,0,0)$
目的関数値＝10
➔暫定解の目的関数値より悪いので
暫定解は変更しない



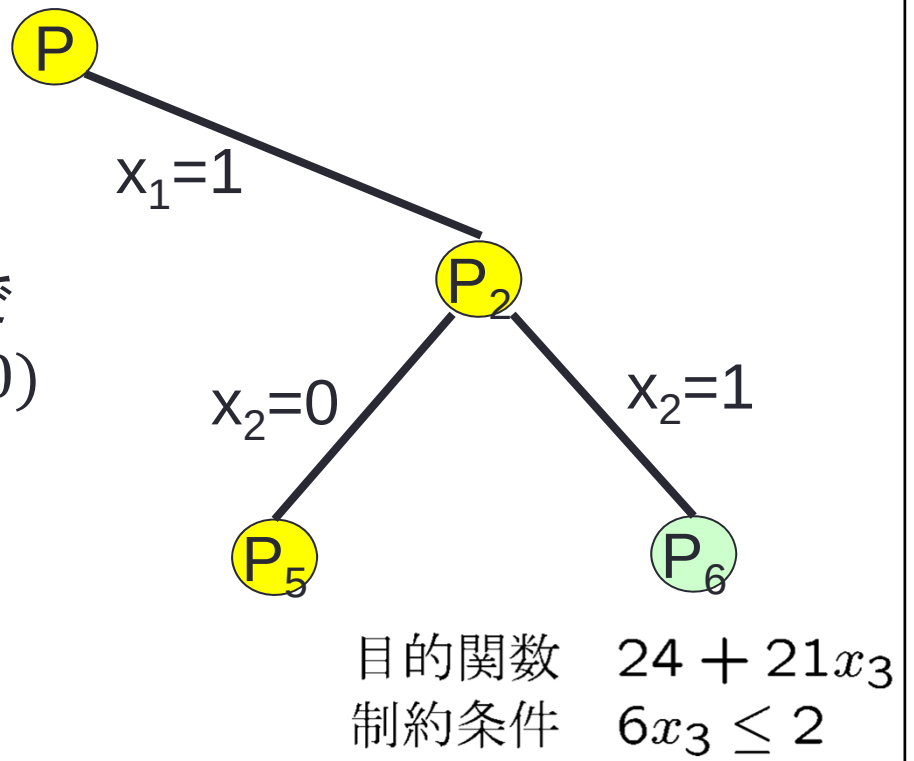
目的関数　$10 + 21x_3$
制約条件　$6x_3 \leq 5$

深さ優先探索を用いた分枝限定法の例

暫定解$(x_1^*, x_2^*, x_3^*) = (0,0,1)$，目的関数値＝21
解いていない部分問題はP_6 ➔P_6を選ぶ

P_6は変数1個なので簡単に解ける
最適解は $(x_1, x_2, x_3) = (1,1,0)$
目的関数値＝24
➔暫定解の目的関数値より良いので
　暫定解を変更 $(x_1^*, x_2^*, x_3^*) = (1,1,0)$

解いていない部分問題がなくなった
➔分枝限定法は終了，
　現在の暫定解が最適解



分枝限定法の高速化のための工夫

- より良い上界値（緩和問題）を使う
- 良い許容解（近似解）をあらかじめ計算し，暫定解として使う
  - 無駄な部分問題を早く削除できる
  - 良い上界値・許容解の計算に時間を使いすぎると逆効果
- 分枝の工夫
  - $x_1 + x_2 + \ldots + x_k = 1$ と言う制約がある場合には，$x_j = 1$ と言う制約を付加した k 個の部分問題に分割
- 前処理：事前に問題の規模を出来るだけ小さくする
  - 値を固定できる変数を事前に検出
  - より強い条件式の導出
  - 無駄な条件式の削除

レポート問題（締切：次回講義13：05）

下記のナップサック問題を分枝限定法で解きなさい。
ただし，「ナップサックの容量＝10」とする
注意事項：

- 深さ優先探索のやり方で部分問題を解くこと
- 変数をx_A, x_B, x_C, x_Dとおいたとき，
  - x_A, x_B, x_C, x_Dの順に変数を0または1に固定して部分問題を生成すること
  - 変数を0に固定した問題を優先して解くこと

品物	A	B	C	D
価値	25	9	11	17
重さ	8	3	4	7