情報システム評価学
―整数計画法―

第7回目：分枝限定法
近似アルゴリズム

塩浦昭義（東北大学 大学院情報科学研究科 准教授）

分枝限定法の考え方

- 整数計画問題(組合せ最適化問題)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

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）
- 解いても無駄な部分問題の例
  - 最適解がすでに得られた部
  - 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）
- 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

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
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，
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？

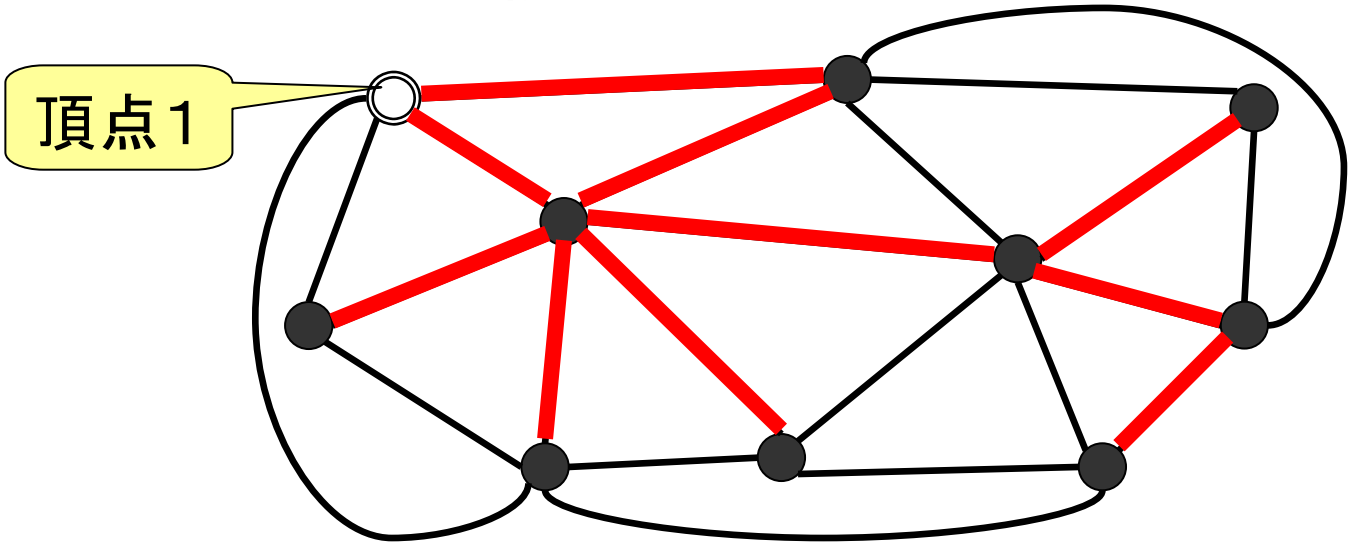
どのような緩和問題を解くか？

ステップ３（分枝操作）：
P'を場合分けによって$P'_1, P'_2, \ldots, P'_k$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？

巡回セールスマン問題に対する分枝限定法の例

- 限定操作のやり方ー1木による緩和を利用
  - 頂点1にちょうど2本の枝が接している
  - 頂点1に接している枝を1本削除すると全域木になる



1木の例

最小1木は，そのグラフの巡回路の長さの下界値

∴ある部分問題において，最小1木の長さ≧暫定解の長さ

➔ その部分問題は無視してよい

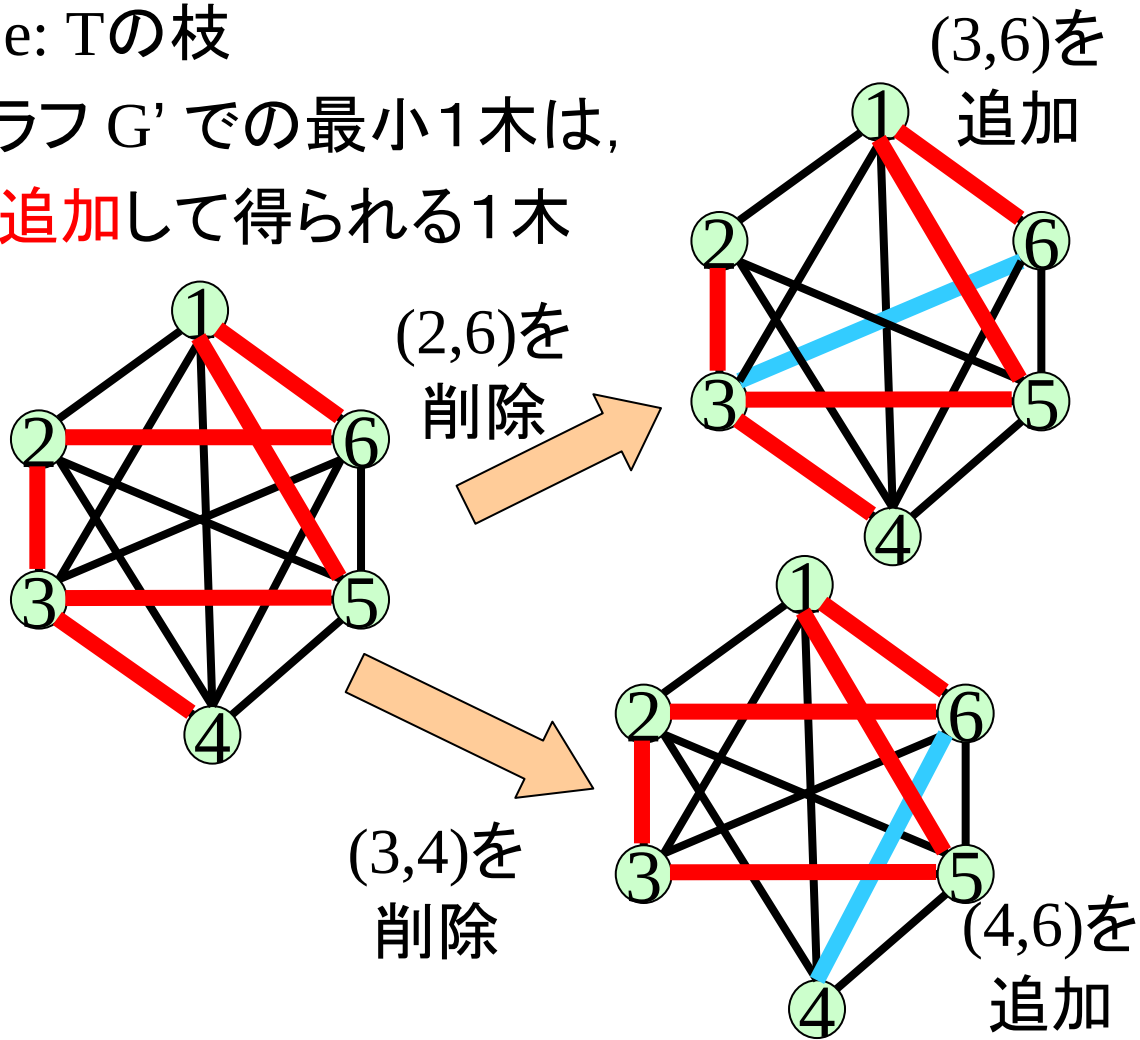
最小1木の性質

- T:グラフGの最小1木，e: Tの枝

➔ Gから枝 e を削除したグラフ G’ での最小1木は，
T – {e} にある枝を一本追加して得られる1木

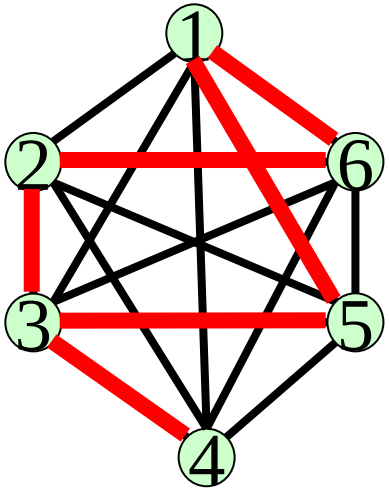
2	3	4	5	6	
99	58	57	18	53	1
	42	52	88	30	2
		30	35	45	3
			95	46	4
				85	5

2頂点間の枝の長さ



巡回セールスマン問題に対する分枝限定法の例

- 解きたい問題：6頂点の巡回セールスマン問題



最小1木の長さ＝208
巡回路ではないので分枝する

2	3	4	5	6	
99	58	57	18	53	1
	42	52	88	30	2
		30	35	45	3
			95	46	4
				85	5

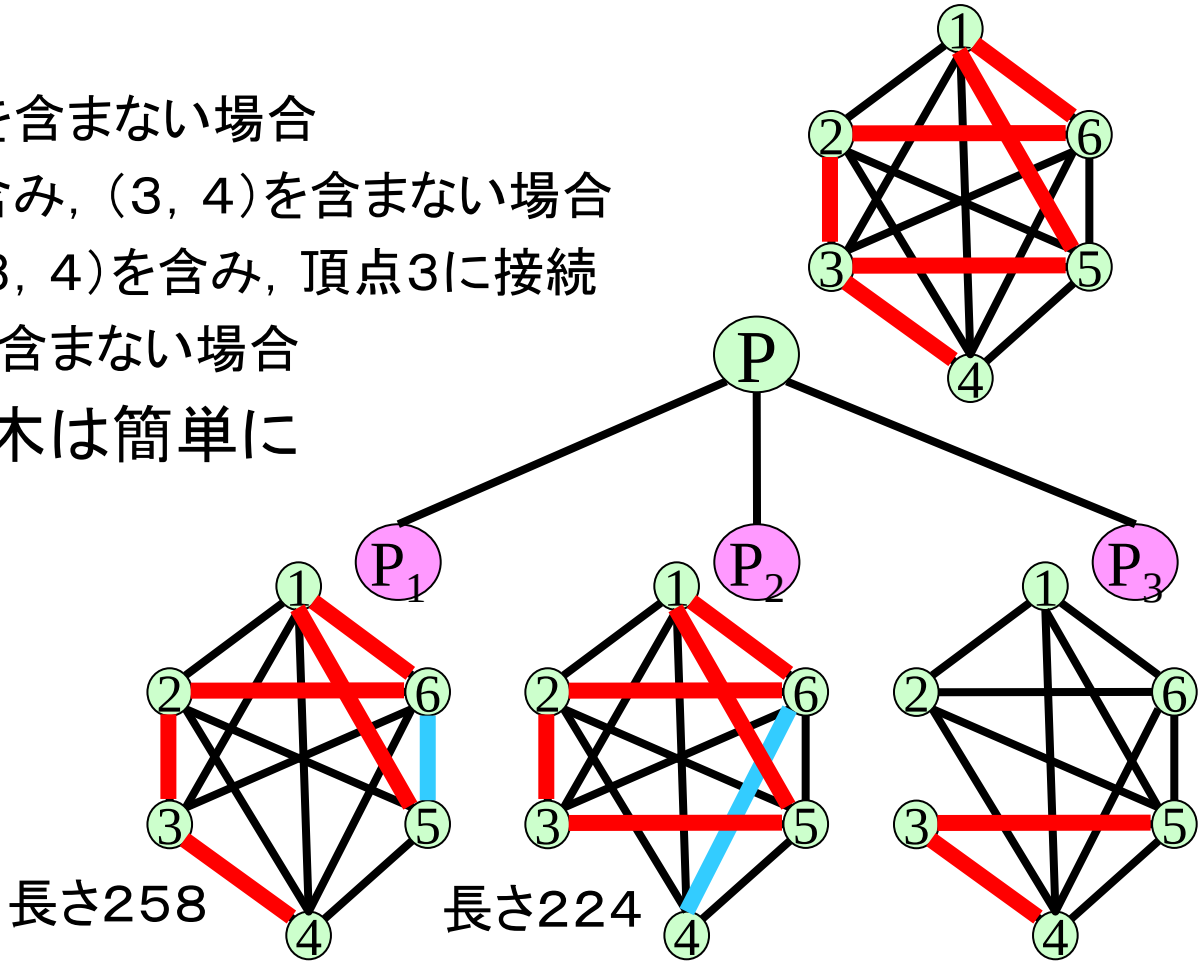
2頂点間の枝の長さ

- 最小1木において，頂点3の次数＞2

➔ 頂点3に接続する枝（3，5），（3，4）を使って，3つの部分問題を生成

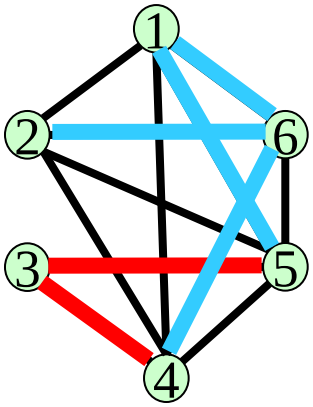
巡回セールスマン問題に対する分枝限定法の例

- ➔ 頂点3に接続する枝(3, 5), (3, 4)を使って, 3つの部分問題を生成
  - P_1: 枝(3, 5)を含まない場合
  - P_2: (3, 5)を含み, (3, 4)を含まない場合
  - P_3: (3, 5), (3, 4)を含み, 頂点3に接続する他の枝を含まない場合
- P_1, P_2 の最小1木は簡単に計算できる

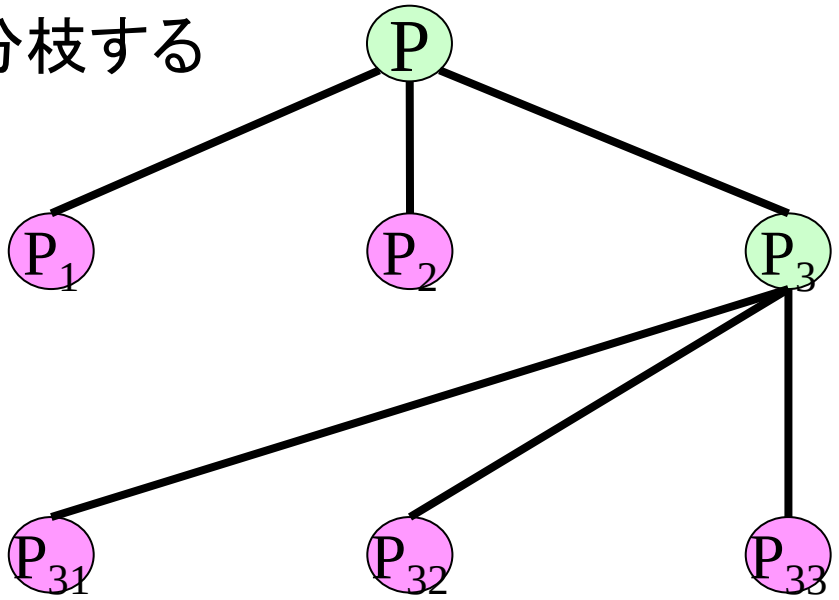


巡回セールスマン問題に対する分枝限定法の例

- 問題 P_3 を選ぶ



最小1木の長さ＝212
巡回路ではないので分枝する



頂点6の次数＞2
➔枝(2，6)，(4，6)を使って
部分問題を生成

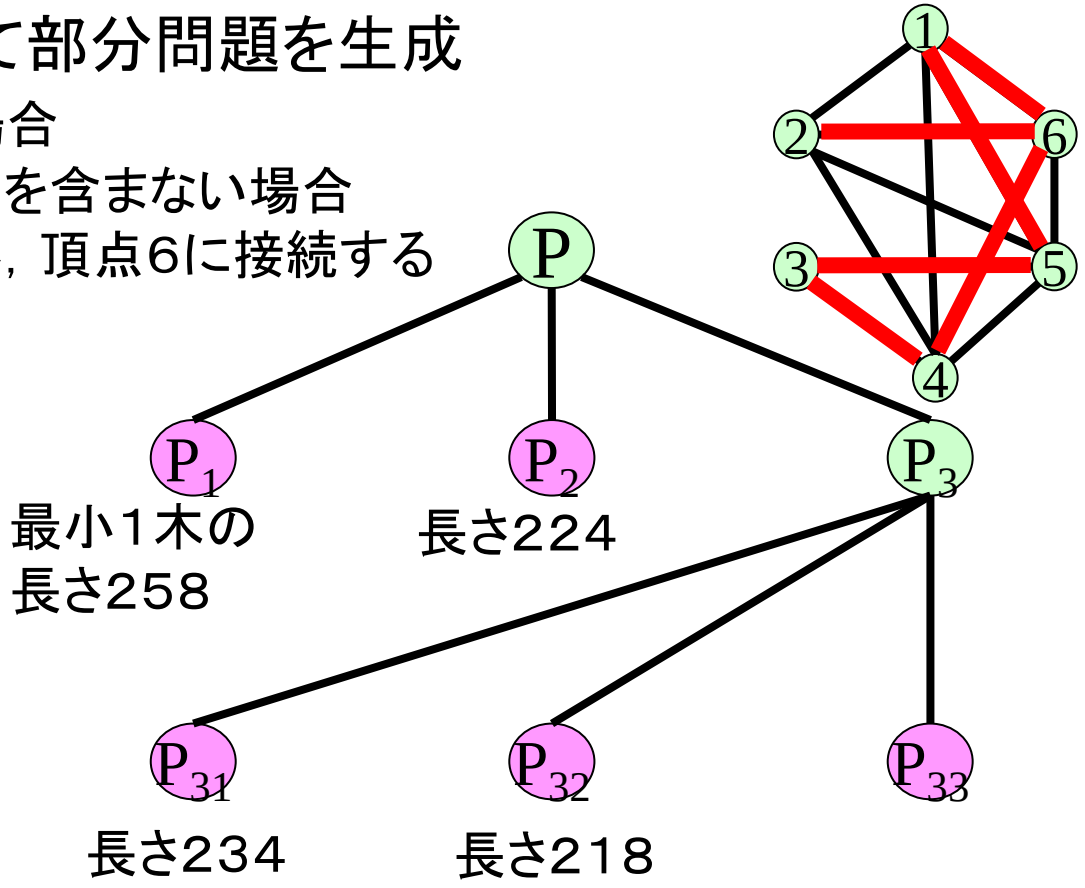
- P_{31}: 枝(2，6)を含まない場合
- P_{32}: (2，6)を含み，(4，6)を含まない場合
- P_{33}: (2，6)，(4，6)を含み，頂点6に接続する他の枝を含まない場合

巡回セールスマン問題に対する分枝限定法の例

➔枝(2, 6), (4, 6)を使って部分問題を生成

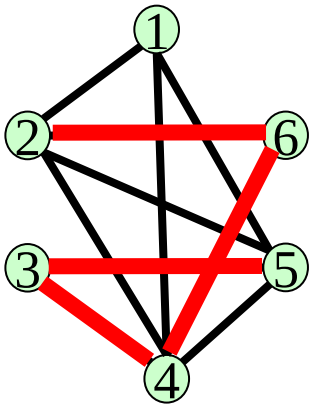
- P_{31}: 枝(2, 6)を含まない場合
- P_{32}: (2, 6)を含み, (4, 6)を含まない場合
- P_{33}: (2, 6), (4, 6)を含み, 頂点6に接続する他の枝を含まない場合

- P_{31}, P_{32} の最小1木は簡単に計算できる

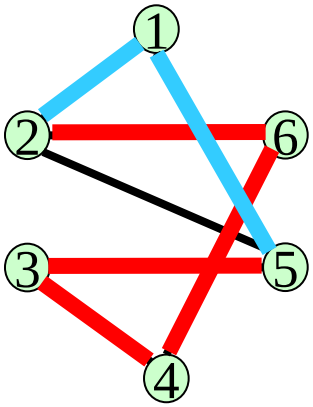


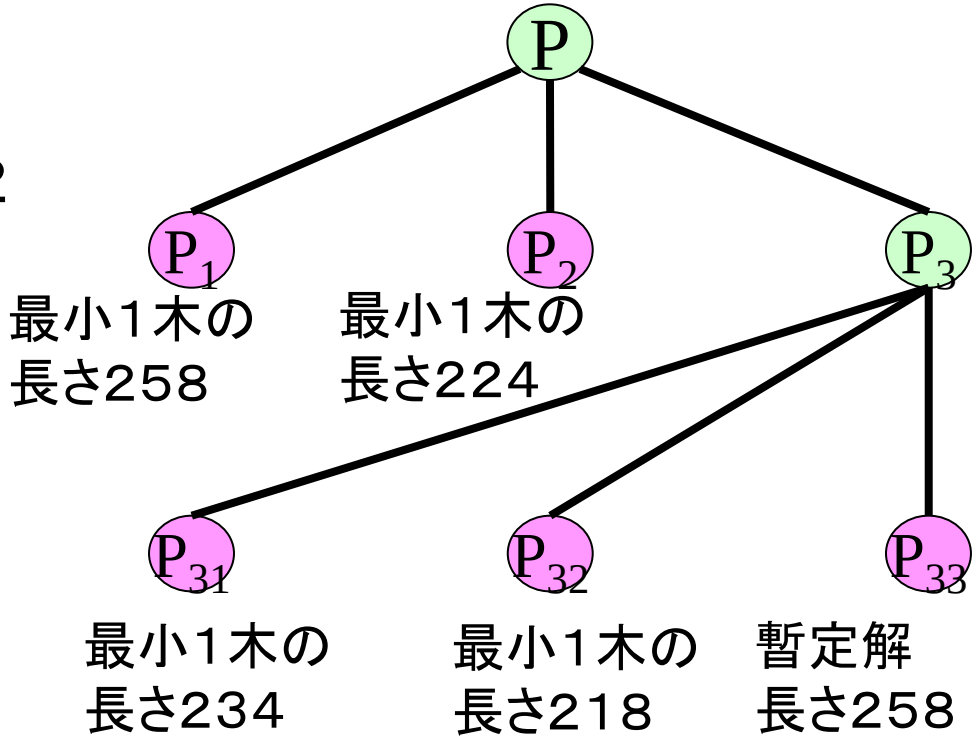
巡回セールスマン問題に対する分枝限定法の例

- 次に問題P_{33}を選ぶ



頂点4は既に次数2
➔他の枝(1,4),
(2,4), (4,5) を削除

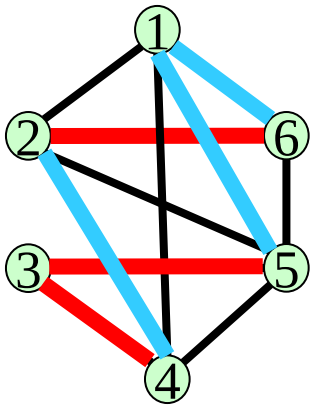


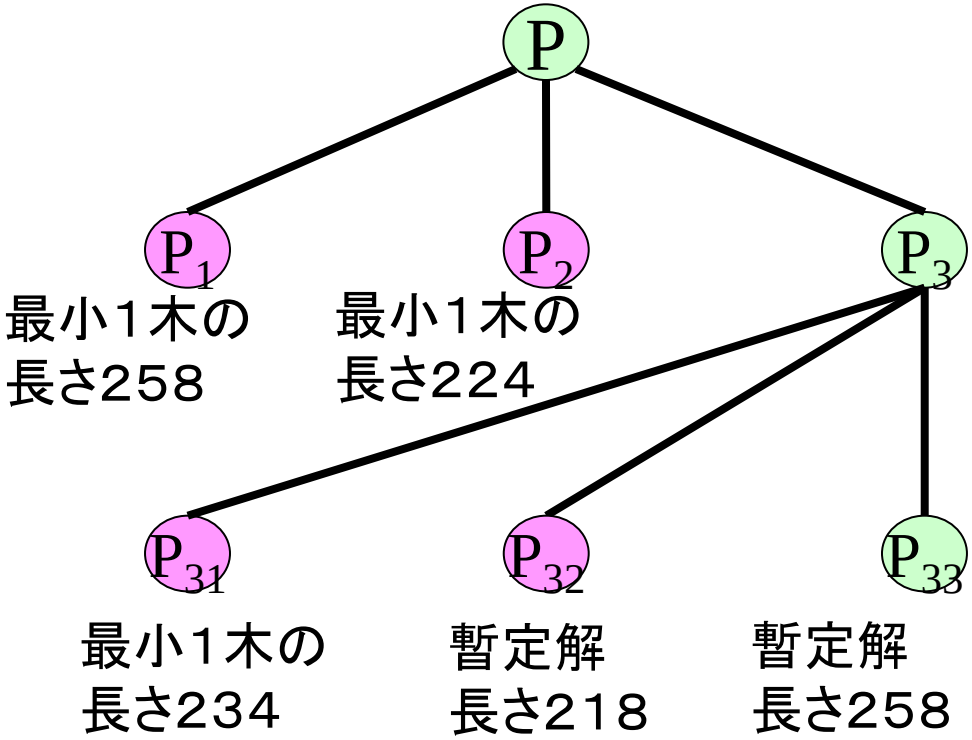


枝を削除したグラフでの最小1木の長さ＝258
これは巡回路➔暫定解とする

巡回セールスマン問題に対する分枝限定法の例

- 次に問題P_{32}を選ぶ





最小1木の長さ＝218
これは巡回路➔暫定解とする

まだ解いていない部分問題において，

最小1木の長さ≧暫定解の長さ 218

➔ 他の部分問題は無視して良い ➔ 分枝限定法の計算終了

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

近似アルゴリズム

- 良い許容解を短時間で求めるアルゴリズム
- 直感，発見，経験に基づき構築されたアルゴリズム（ヒューリスティックス）を指すことが多い
- 近似アルゴリズムを使う理由
  - 短時間で許容解が欲しい
  - 問題が複雑/大規模であり，整数計画問題として定式化することが難しい
  - 分枝限定法を使って最適解を求めることが困難
  - ある種の問題に対しては，問題の構造を利用したアルゴリズムを使った方が効果的

近似アルゴリズム

- 近似アルゴリズムの評価尺度
  - 近似精度 --- 理論的な評価/実験による評価
  - 計算時間

- 解の構築方法による分類
  - 許容解を一つのみ構築 --- 貪欲アルゴリズムなど
  - 許容解を繰り返し改善 --- 近傍探索（ローカルサーチ）

貪欲アルゴリズム(greedy algorithm)

- ある基準に関して最良のアイテム，枝，頂点などを順々に追加していき，許容解を作るアルゴリズム
- 直感的には，良い許容解を得ることが出来そう
- 最小木などの問題に対しては最適解を得る（➔マトロイド構造）
- 問題によっては悪い解を出力する可能性も

貪欲アルゴリズム(greedy algorithm)

- 例1:0-1ナップサック問題

$$
\begin{aligned}
&\text{最大化} \quad 12x_1 + 8x_2 + 17x_3 + 11x_4 + 6x_5 + 2x_6 + 2x_7 \\
&\text{条件} \quad 4x_1 + 3x_2 + 7x_3 + 5x_4 + 3x_5 + 2x_6 + 3x_7 \leq 9 \\
&\qquad x_j \in \{0, 1\} \ (j = 1, \ldots, 7)
\end{aligned}
$$

アイテムの価値

アイテムの重量

- 各アイテムに対して(価値／重量)を計算，
  大きい順に並べる

①	②	~~3~~	~~4~~	~~5~~	⑥	~~7~~
3	2.67	2.43	2.2	2	1	0.67

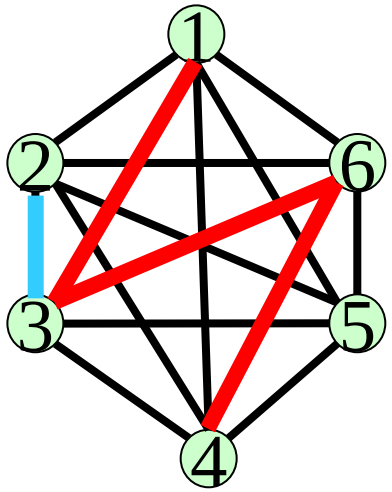
- この順番にアイテムを追加する．重量オーバーとなる
  アイテムは除外する ➔{1, 2, 6}, 目的関数値32

貪欲アルゴリズム(greedy algorithm)

- 例2 : 対称巡回セールスマン問題

- 短い枝から順に追加する.
- ある頂点の次数が3以上になる場合は除外する.

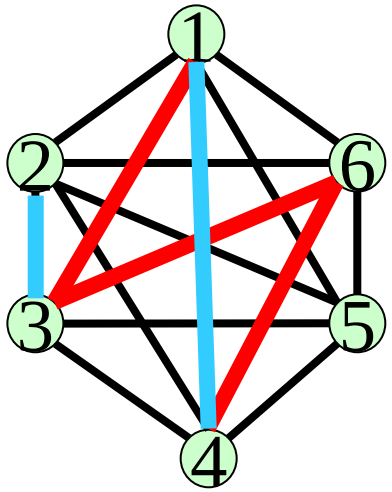
2	3	4	5	6	
9	○2	8	12	11	1
	7×	19	10	32	2
		29	18	○6	3
			24	○3	4
				19	5



貪欲アルゴリズム(greedy algorithm)

- 例2:対称巡回セールスマン問題

- 短い枝から順に追加する.
- ある頂点の次数が3以上になる場合は除外する.
- 部分巡回路が出来てしまう場合は除外する

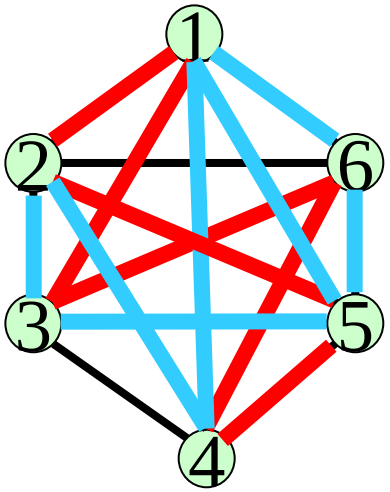


2	3	4	5	6	
9	○2	8×	12	11	1
	7×	19	10	32	2
		29	18	○6	3
			24	○3	4
				19	5

貪欲アルゴリズム(greedy algorithm)

- 例2:対称巡回セールスマン問題

- 短い枝から順に追加する.
- ある頂点の次数が3以上になる場合は除外する.
- 部分巡回路が出来てしまう場合は除外する



2	3	4	5	6	
(9)	(2)	8 ×	12 ×	11 ×	1
	7 ×	19 ×	(10)	32	2
		29	18 ×	(6)	3
			(24)	(3)	4
				19 ×	5

長さ54

貪欲アルゴリズムの近似精度

- 必ずしも最適解を出力するとは限らない
- 近似精度が非常に大きくなる問題例も存在
  - 紹介した2つの貪欲アルゴリズムの近似精度は無限大

貪欲アルゴリズムの近似精度

- 例1：0-1ナップサック問題（k は任意の正整数）

$$
\begin{aligned}
&\text{最大化} \quad x_1 + (k-1)x_2 \\
&\text{条件} \quad x_1 + kx_2 \leq k \\
&\qquad\quad x_1, x_2 \in \{0, 1\}
\end{aligned}
$$

  - 得られる近似解：アイテム1のみ，目的関数値1
  - 最適解：アイテム2のみ，目的関数値 $k-1$
  - 近似解の近似精度：k-1/1

    --- k を大きくすると近似精度は任意に大きくなる
  - でも，貪欲アルゴリズムをちょっと改良すると近似精度を1／2にすることが可能

貪欲アルゴリズムの近似精度

- 例2: 対称巡回セールスマン問題

Bad News

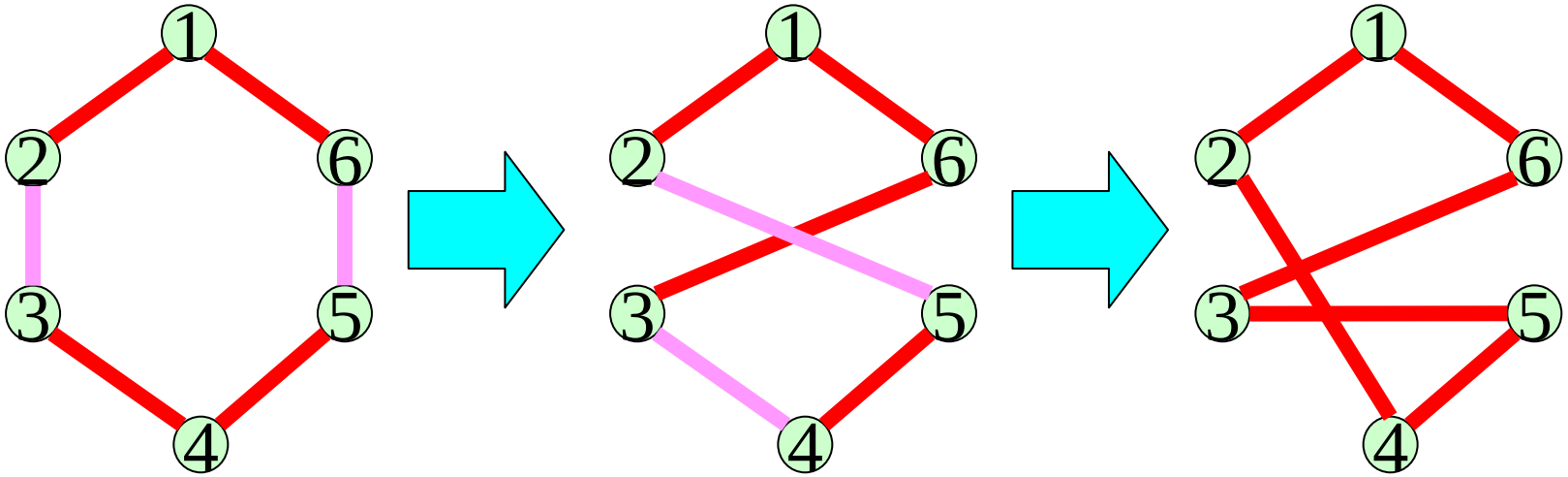
都市数 n に対して，近似比 $\geq \dfrac{\log n}{\log \log n}$ となる問題例が存在

Good News

枝の長さが三角不等式 $c_{ij} \leq c_{ik} + c_{kj}$ を満たす場合，
都市数 n に対して，常に近似比 $= O(\log n)$ となる

局所探索(local search)

- ある許容解から別の許容解を得るための基本的な操作を定義（要素の交換など）
- 基本的な操作で得られる許容解の集合＝近傍
- 基本的な操作を使って，現在の許容解を繰り返し修正し，より良い解を構築する --- 得られた解は局所最適解
- 例：巡回セールスマン問題 --- 基本的な操作：2本の枝の交換



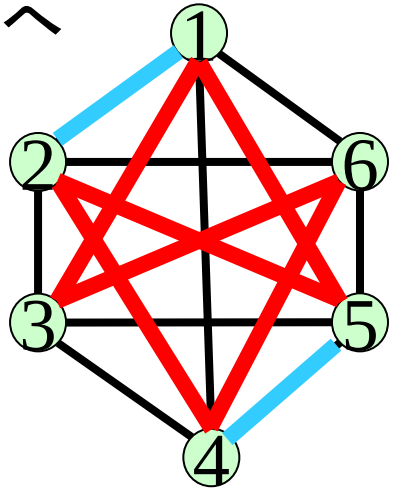
巡回セールスマン問題に対する局所探索

- 基本的な操作：2本の枝の交換
  - 現在の巡回路から枝2本(a, b), (c, d) を削除
  - 新たな枝 (a, c), (b, d) または (a, d), (b, c) を追加

枝(1,2), (4,5) を削除（長さ33）

枝(1,5), (2,4) を追加（長さ31）

➔ 合計の長さ54から52へ



2	3	4	5	6	
9	2	8	12	11	1
	7	19	10	32	2
		29	18	6	3
			24	3	4
				19	5

巡回セールスマン問題に対する局所探索

- 基本的な操作：2本の枝の交換
  - 現在の巡回路から枝2本(a, b), (c, d) を削除
  - 新たな枝 (a, c), (b, d) または (a, d), (b, c) を追加

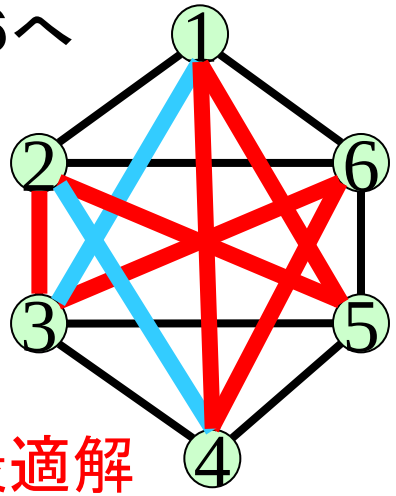
枝(1,3), (2,4) を削除（長さ21）

枝(1,4), (2,3) を追加（長さ15）

➔ 合計の長さ52から46へ

枝2本の交換では
これ以上巡回路の
長さを短くすることは
不可能

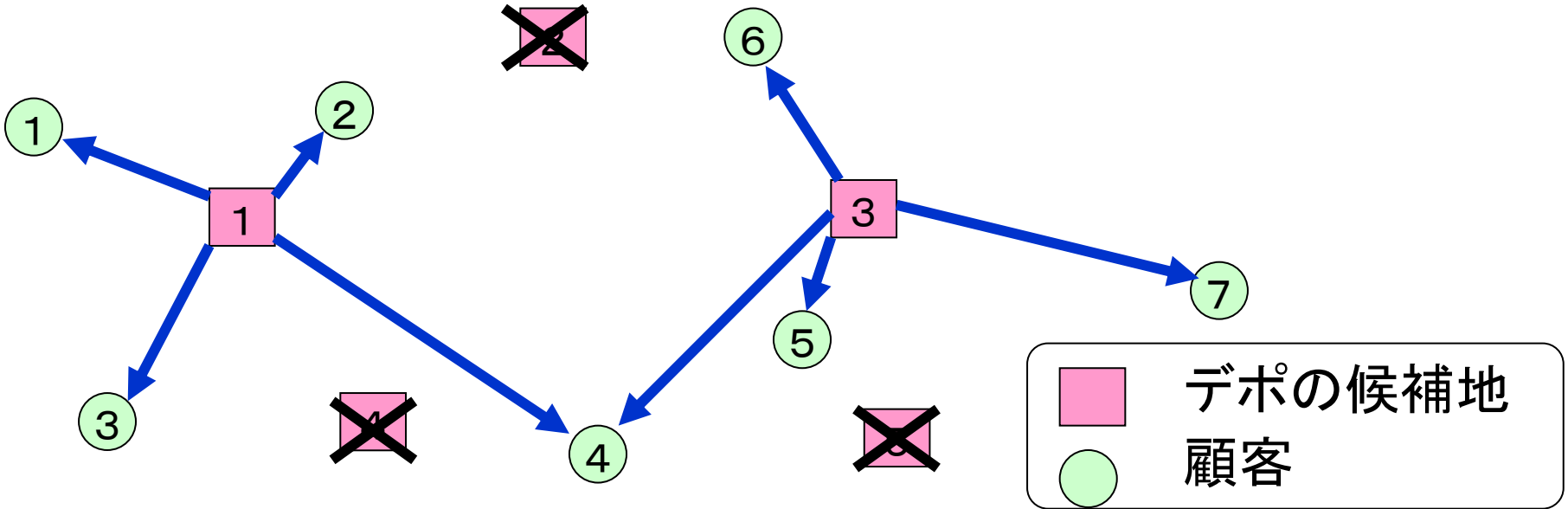
➔ 現在の巡回路は局所最適解



2	3	4	5	6	
9	2	8	12	11	1
	7	19	10	32	2
		29	18	6	3
			24	3	4
				19	5

容量なし施設配置問題

- デポの候補地 $N=\{1, 2, \ldots, n\}$, 顧客 $M=\{1, 2, \ldots, m\}$
- デポの幾つかを設置（デポ i の設置コスト f_i）
- 全ての顧客をいずれかのデポに割り当て
  （顧客 j をデポ i に割り当て➔コスト c_{ij}）
- 目的：デポの設置コスト＋顧客の割当コストの最小化



容量なし施設配置問題に対する局所探索

- 基本的な操作：
  デポを一つ追加，一つ削除
- 顧客は常に最良のデポに割り当てる

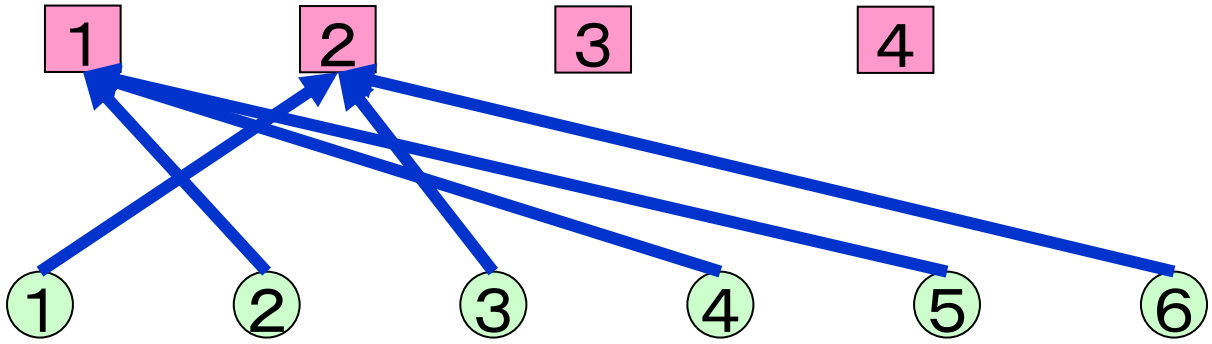
1	2	3	4
21	16	11	24

デポの
設置コスト

	1	2	3	4
1	6	2	3	4
2	1	9	4	11
3	15	2	6	3
4	9	11	4	8
5	7	23	2	9
6	4	3	1	5

顧客のデポへの
割当コスト

コスト37＋24＝61



容量なし施設配置問題に対する局所探索

- 基本的な操作：
  デポを一つ追加，一つ削除
- 顧客は常に最良のデポに割り当てる

デポ3を新たに設置

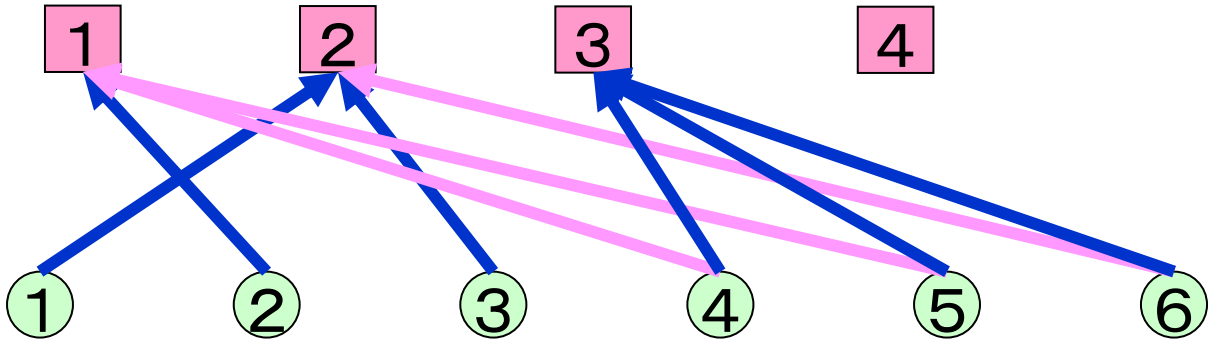
1	2	3	4
21	16	11	24

デポの設置コスト

	1	2	3	4
1	6	2	3	4
2	1	9	4	11
3	15	2	6	3
4	9	11	4	8
5	7	23	2	9
6	4	3	1	5

顧客のデポへの割当コスト

コスト37＋24＝61➔48＋12＝60



容量なし施設配置問題に対する局所探索

- 基本的な操作：
  デポを一つ追加，一つ削除
- 顧客は常に最良のデポに割り当てる

1	2	3	4
21	16	11	24

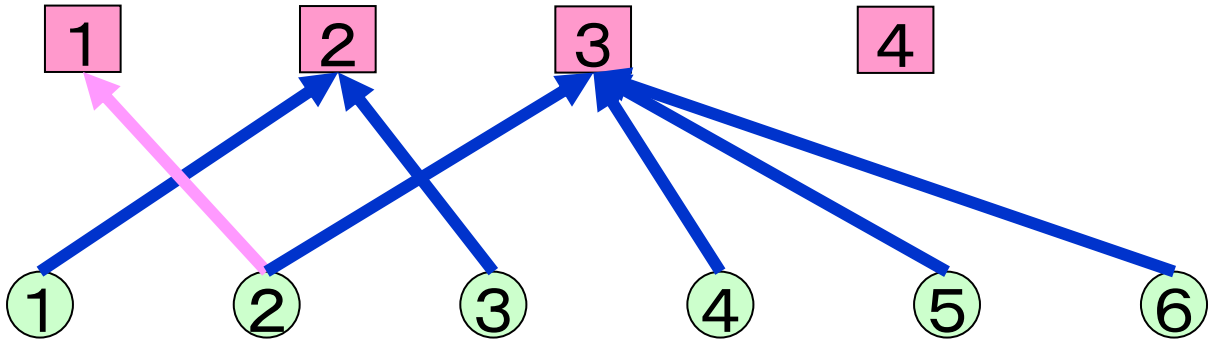
デポの設置コスト

	1	2	3	4
1	6	2	3	4
2	1	9	4	11
3	15	2	6	3
4	9	11	4	8
5	7	23	2	9
6	4	3	1	5

顧客のデポへの割当コスト

デポ1を廃止

コスト48＋12＝60➔27＋15＝42



容量なし施設配置問題に対する局所探索

- 基本的な操作：
  デポを一つ追加，一つ削除
- 顧客は常に最良のデポに割り当てる

1	2	3	4
21	16	11	24

デポの
設置コスト

	1	2	3	4
1	6	2	3	4
2	1	9	4	11
3	15	2	6	3
4	9	11	4	8
5	7	23	2	9
6	4	3	1	5

顧客のデポへの
割当コスト

デポ2を廃止➔局所最適解

コスト27＋15＝42➔11＋20＝31