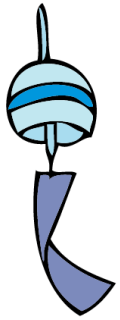
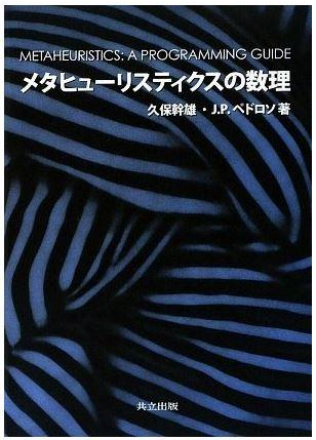


メタヒューリスティクスの数理

～第2章 代表的なメタヒューリスティクス～

局所探索法
多出発局所探索法
反復局所探索法
模擬焼きなまし法
禁断探索法
誘導局所探索法



メタヒューリスティクス　～夏の祭典～
2013/07/10(水)　M1 今泉孝章

最適化問題

●定義

特定の集合上で定義されたある
実数(or整数)関数についてその値が
最大(最小)となる状態を解析する問題

↓

実行可能領域 F が与えられたとき関数 $f(x)$ を
最大(最小)とする x を求める問題

ここでは組み合わせ最適化問題を主に扱う

↓

最適解の集合が離散的である問題

最適化問題

機械スケジューリング問題

各ジョブにおける処理時間p_iと納期d_iが分かっているとき
納期遅れの合計値が最も小さくなるジョブの処理順番を求める

ジョブ	i	1	2	3	4
処理時間	p_i	1	2	3	4
納期	d_i	5	9	6	4

F →ジョブの全ての処理順番の組み合わせで4!通り(離散的)

$f(x)$ →納期遅れの合計値 $\sum_{i=1}^{4}\left[\sum_{x(j)\le x(j)} p_j - d_i\right]$ ($x(i)$はジョブiの処理順番を表す)

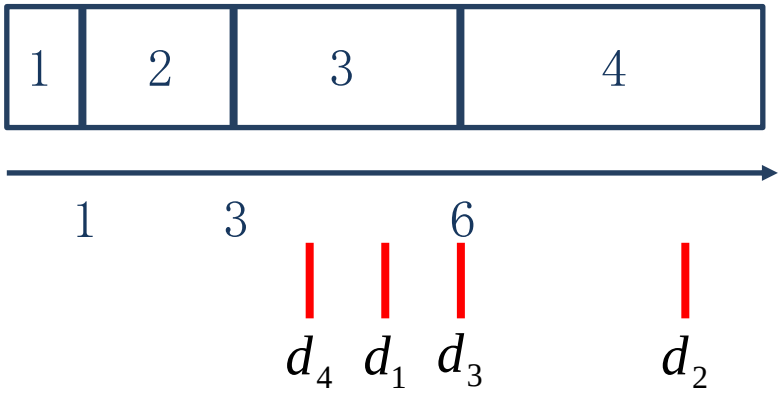
x →ジョブのある処理順番(例：1→4→3→2)

最適化問題

機械スケジューリング問題

ジョブ	i	1	2	3	4
処理時間	p_i	1	2	3	4
納期	d_i	5	9	6	4

解が1→2→3→4の場合



ジョブ4が納期4に対し
処理終了時刻が10である

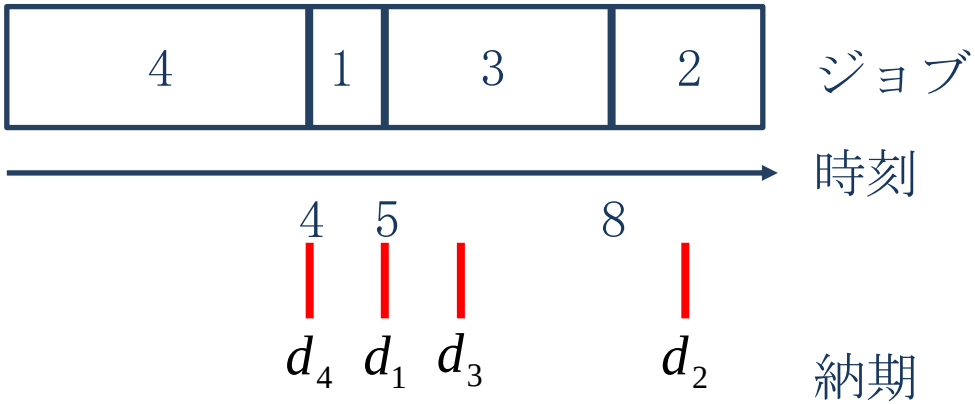
→ $f(x)=6$

最適化問題

機械スケジューリング問題

ジョブ	i	1	2	3	4
処理時間	p_i	1	2	3	4
納期	d_i	5	9	6	4

解が4→1→3→2の場合



ジョブ3が納期6に対し
処理終了時刻が8である
+
ジョブ2が納期9に対し
処理終了時刻が10である

→ $f(x)=3$

メタヒューリスティクスとは?

- **厳密解法（Exact Algorithm）**
  最適性の保証された解を求める手法

↕

- **近似解法（Approximate Algorithm）**
  一番良い解（厳密解）ではなくある程度良い解（近似解）を求めることが保証されている手法

↑

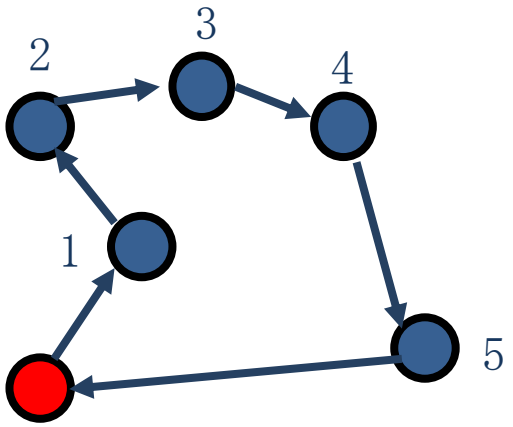
- **発見的解法（Heuristics）**
  求められた解に精度の保証はないが、経験的に近似解が求められるとわかっている手法

メタヒューリスティクスとは?

- **発見的解法（Heuristics）**
  - **構築法（Constructive Method）**

    何もない状態からあるルールに基づき解を構築する手法

巡回セールスマン問題



- **最近近傍探索（Nearest Neghibor Search）**

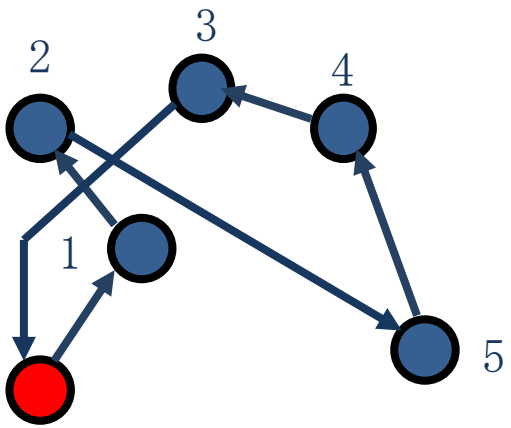
  最も近い都市を順につなぎ、全ての都市を訪問し終えたら、出発地に戻る

メタヒューリスティクスとは?

- 発見的解法（Heuristics）
  - 改善法（Improvement Method）

    適当な実行可能初期解からあるルールに基づき
    解を改善していく手法

巡回セールスマン問題



初期解

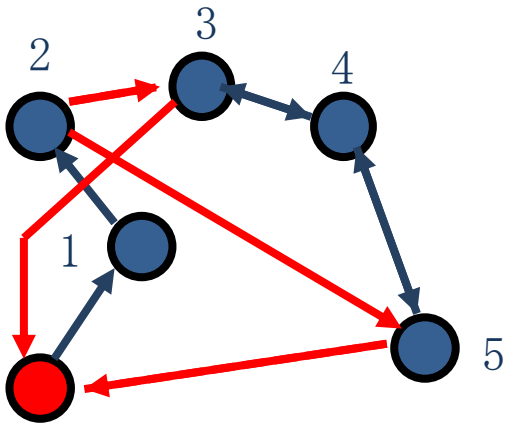
出発地

メタヒューリスティクスとは?

- 発見的解法（Heuristics）
  - 改善法（Improvement Method）

適当な実行可能初期解からあるルールに基づき
解を改善していく手法

巡回セールスマン問題



出発地

改善解

(3−出発地)+(2−5)→(2−3)+(出発地−5)

- 2−opt法

適当な巡回路の2本の枝
(i,j)、(k,l)で $d_{ik}+d_{jl}<d_{ij}+d_{kl}$
となる枝があれば (i,k)、(j,l)に
繋ぎかえる

メタヒューリスティクスとは?

- **メタヒューリスティクス（Metaheuristics）**

  ヒューリスティクスに幾つかのパラメータを追加し
  問題を巧く解くテクニックのこと

～例～

- 改善の方法を設定する
- 改善を終了し、最終的な解とする基準を設定する
- 改善する解を探索する方法を設定する

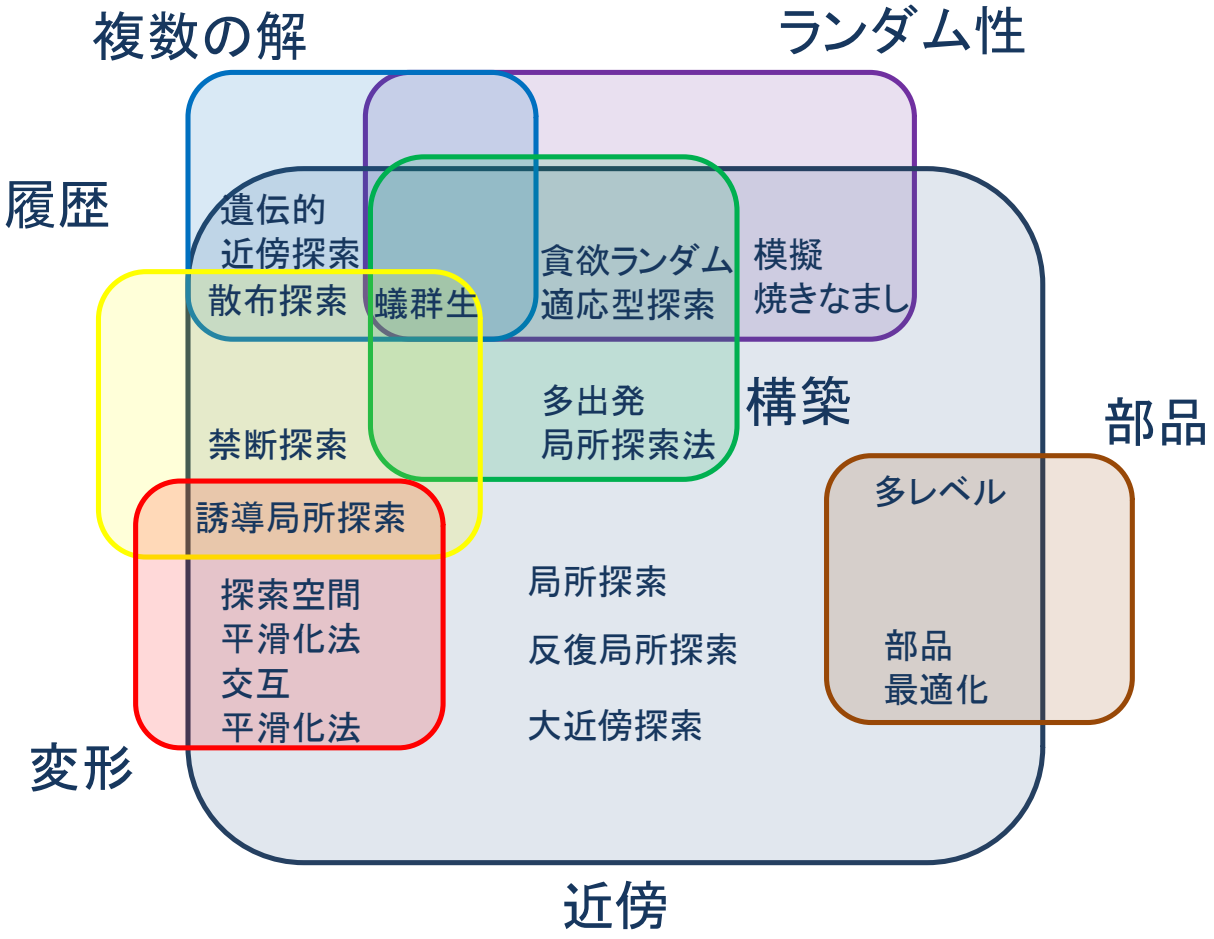
↓

効率的にヒューリスティクスを行うための戦略
計算時間に応じた良好な結果を得られる

メタヒューリスティクスとは?

メタヒューリスティクスの基本戦略

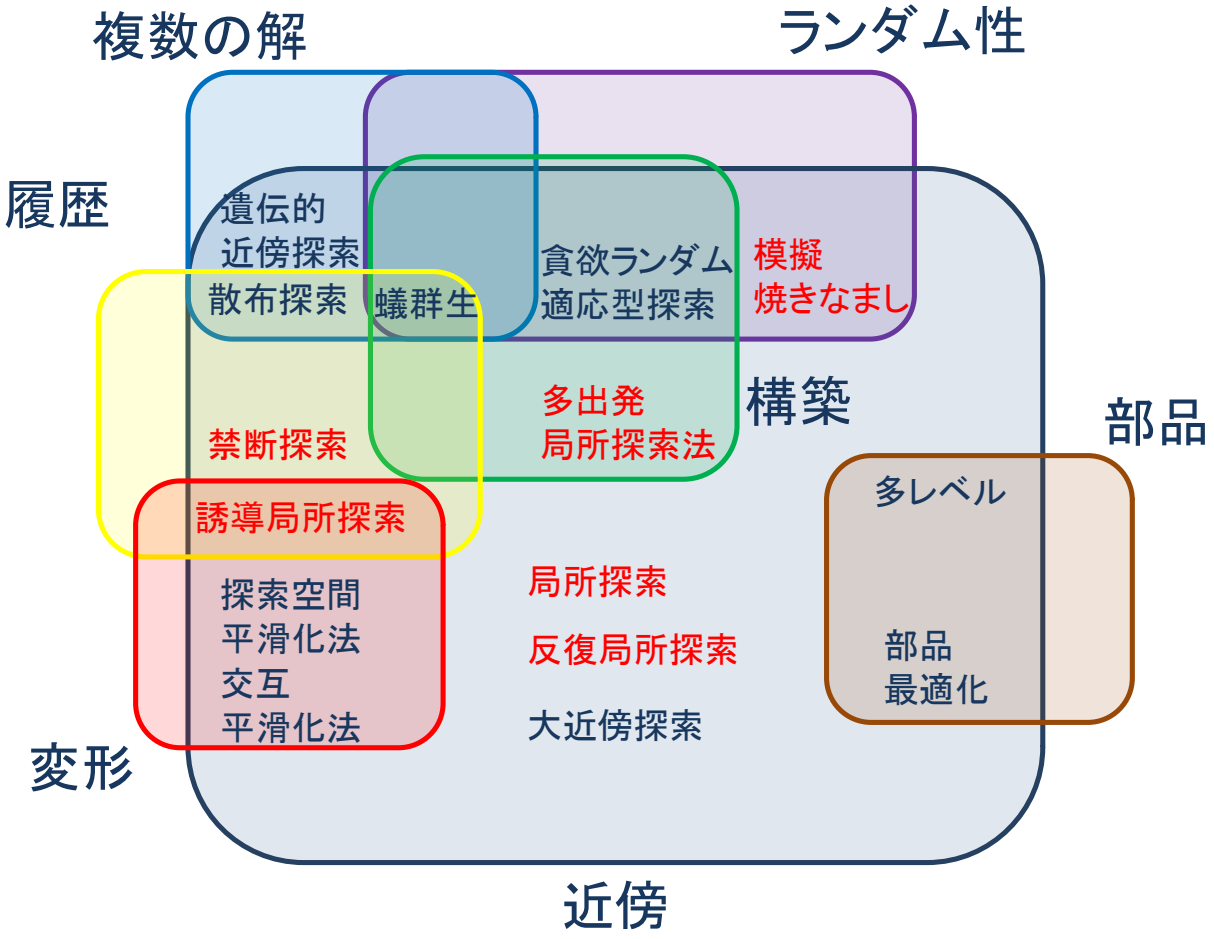
- 近傍の利用
- 構築法の利用
- 部品の利用
- 複数解の保持
- ランダム性の利用
- 問題の変形
- 探索履歴の利用



メタヒューリスティクスとは?

メタヒューリスティクスの基本戦略

- 近傍の利用
- 構築法の利用
- 部品の利用
- 複数解の保持
- ランダム性の利用
- 問題の変形
- 探索履歴の利用

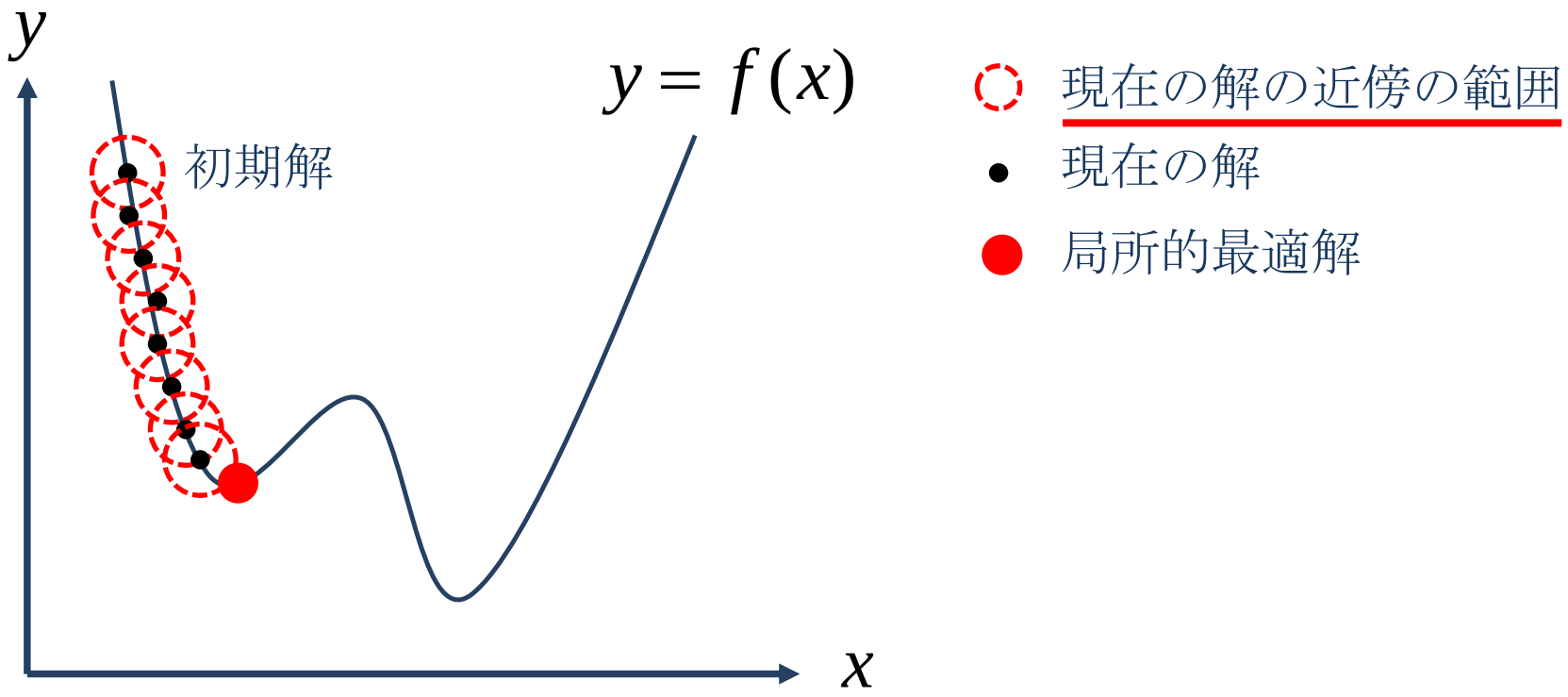


局所探索法

●局所探索法（Local Search）

現在の解の近傍が目的関数の値を改善する場合はその近傍を新たな解入れ替え、そうでなければ現在の解を維持する

※別名： 丘登り法（Hill Climbing Method）

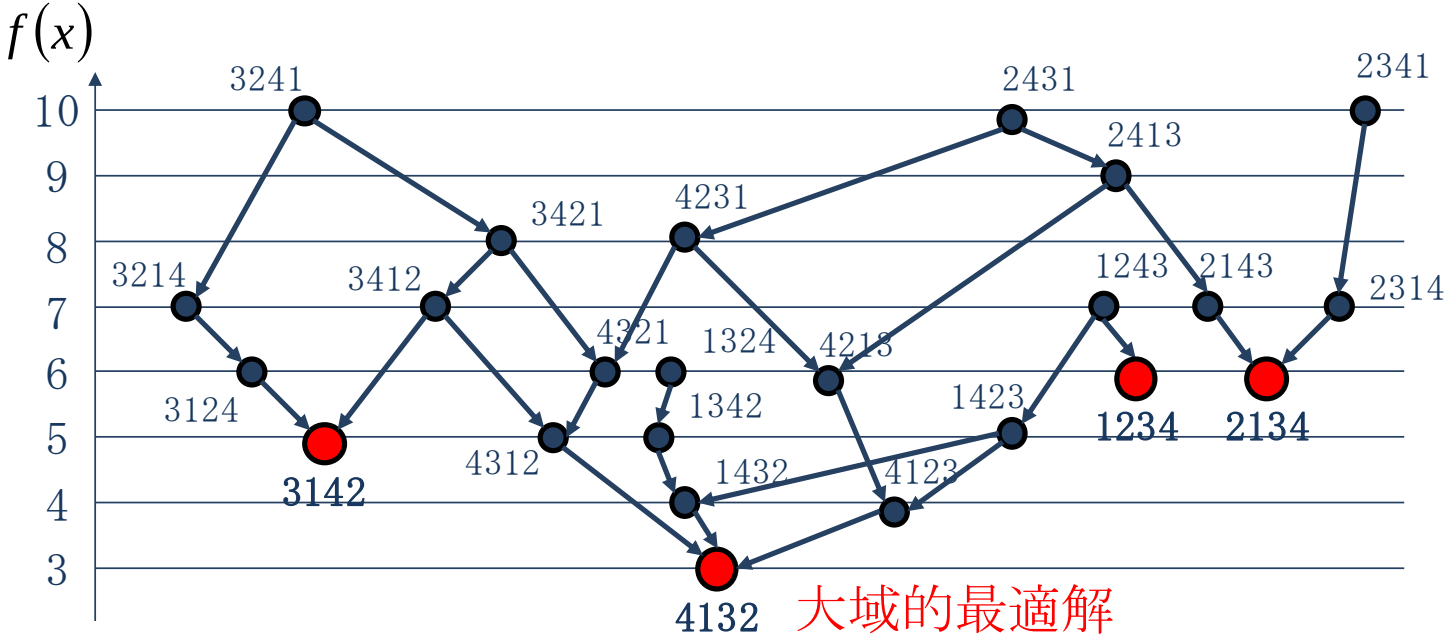


局所探索法

●近傍（Neighborhood）

現在の解を少し摂動させて得られる解の集合

～機械スケジューリング問題での例～



局所最適解

それ以外

枝で結ばれた点が
解 x の近傍の集合
$N(x)$：それぞれの解における近傍となる

局所探索法

$$improve(x) = \begin{cases} f(x') < f(x) \text{を満たす適当な } x' \in N(x) & if \quad x' \text{が存在} \\ \phi & \text{それ以外} \end{cases}$$

アルゴリズム

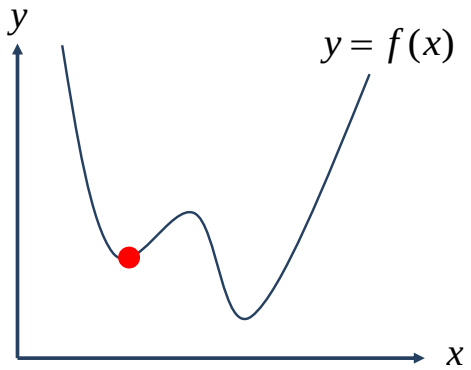
```
1  x := 適当な初期解
2  while improve(x) ≠ φ do
3      x := improve(x)
4  return x
```

→初期解の設定

→改善する解が近傍に存在する限り実行

→ xを改善する解に更新

→ xを出力



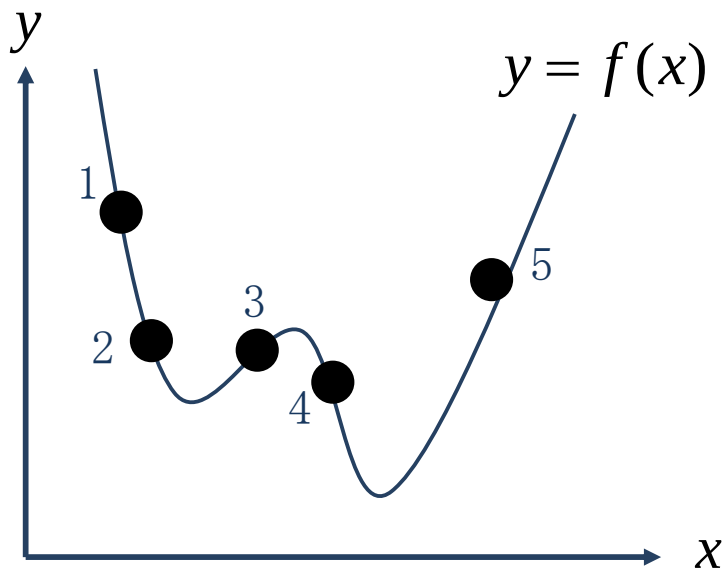
他のメタヒューリスティクスの基礎となる手法
→局所探索法を改良したものが多くの
メタヒューリスティクスとなっている

多出発局所探索法

●多出発局所探索法 (Multiple Start Local Search)

初期解を変更し繰り返し局所探索法を実行する手法
問題が大渓谷性を持つときに有効

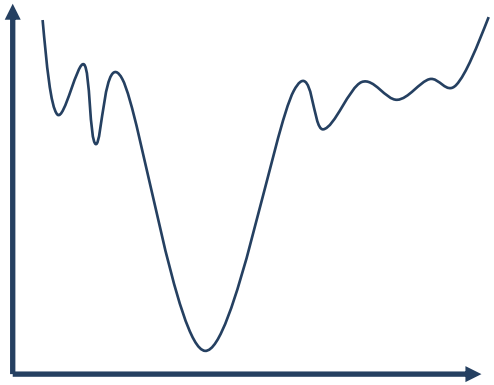
初期解によって局所最適解にとどまるか
大域的最適解に到達できるか異なる



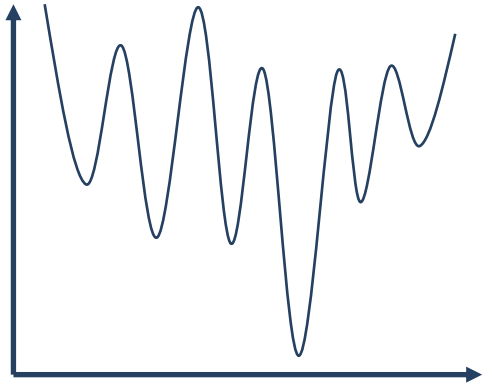
1→局所解
2→局所解
3→局所解
4→大域解
5→大域解

多出発局所探索法

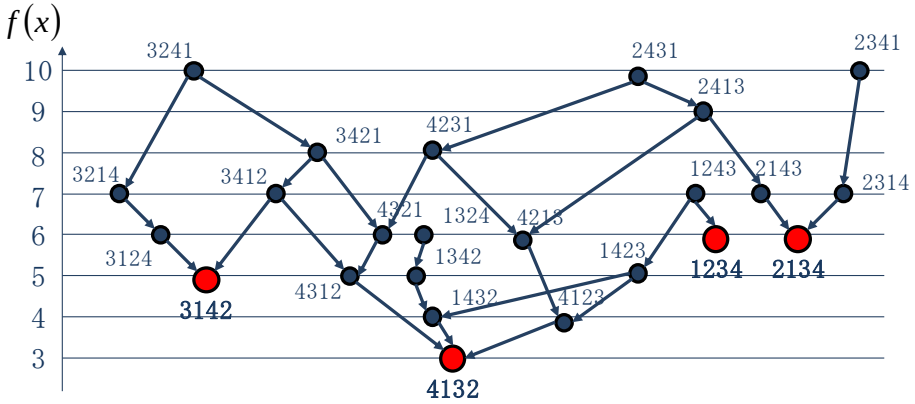
大渓谷性（Big Valley Property）



大渓谷性を持つ



大渓谷性を持たない



多出発局所探索法

アルゴリズム

```
1  while 終了基準が満たされていない do
2      x := 適当な初期解
3      while improve(x) ≠ φ do
4          x := improve(x)
5      x* :=現時点での最良解
6  return x*
```

局所探索法

短所

同じ局所解を繰り返し獲得してしまう可能性がある

→初期解になるべく異なるものを選択することで回避

→**多様化（Diversification）**

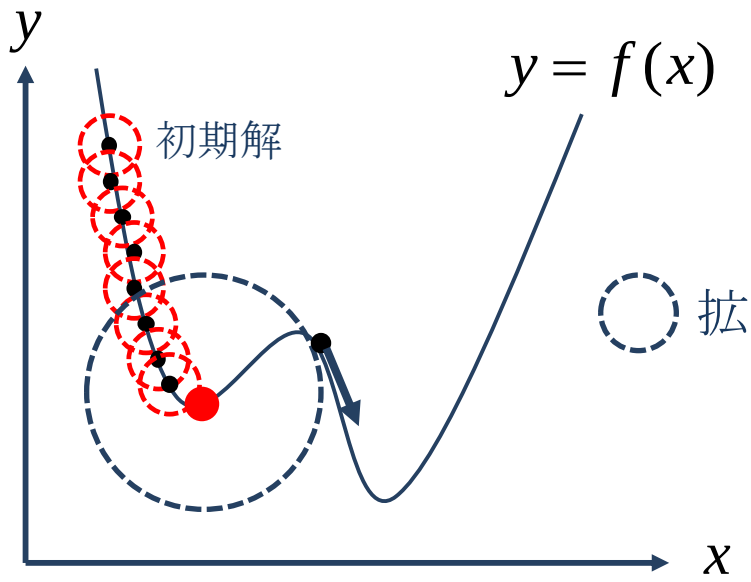
⇔**集中化（intensification）**

以前の試行で獲得した局所解と共通部分を多く持つ解を利用

反復局所探索法

●反復局所探索法（Iterated Local Search）

局所探索法を繰り返すときに初期解からやり直すのではなく局所最適解の近傍からやり直す手法



局所最適解から抜け出すために
拡張された近傍 $N'(x)$ を導入する

※局所解からの脱出を目的とするので解が必ずしも改善するとは限らない

反復局所探索法

$kick(x) =$ 適当な $x' \in N'(x)$

アルゴリズム

```
1  x := 適当な初期解
2  while 終了基準が満たされていない do
3      while improve(x) ≠ φ do
4          x := improve(x)
5      x* :=現時点での最良解
6      x := kick(x または x*)
7  return x*
```

局所探索法

問題が近接最適性を満たす時には有効

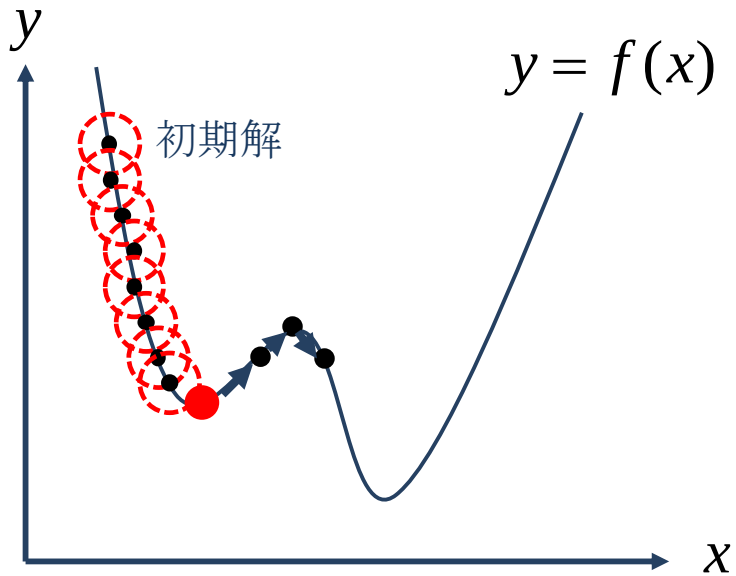
近接最適性（Proximate Optimality Property）

良い解に近いところには良い解が存在する

模擬焼きなまし法

●模擬焼きなまし法 （Simulated Annealing Method）

ランダム性を用いて局所最適解からの脱出を試みる手法



局所最適解から抜け出すために
確率的に解の改悪を許す

x： 現在の解

y： x の近傍 $y \in N(x)$

$\Delta = f(y) - f(x)$

T: 温度パラメータ

$\Delta < 0$

→ 確率 $\exp(-\Delta/T)$ で改悪を許す

模擬焼きなまし法

基本アルゴリズム

1 $x_1 :=$ 適当な実行可能解
2 $for\ \ k = 1\ to\ \infty\ do$
3 y を$N(x_k)$からランダムに選択
4 確率 $\exp\{-[f(y)-f(x_k)]^+ / T_k\}$で $x_{k+1} := y$（y を受理）
5 それ以外のときは $x_{k+1} := x_k$（ yを棄却）

反復回数k のときの温度パラメータ T_k
→非負の値をとる減少数列→ $T_1 \geq T_2 \geq \cdots \geq T_\infty = 0$

↓

試行が進むにつれ受理確率が小さくなり解に最適解に収束する
→多くの手法と異なり理論的に大域的最適解に収束する

模擬焼きなまし法

設定パラメータ

INTPROB　　：初期温度設定のためのパラメータ

TEMPFACTOR：温度冷却の速さを調節するパラメータ

SIZEFACTOR　：解を生成した回数に関する

CUTOFF　　：解を受理した回数に関するパラメータ

MINPERCENT：解の受理確率に関するパラメータ

FREEZELIM　：収束判定に関するパラメータ

↓

それぞれ予備実験により妥当な値を見つける
それぞれはどのような役割をするのか?

模擬焼きなまし法

●温度冷却方法 ～幾何冷却～

同一温度で均衡に達するまで(ある一定の回数まで)移動を反復し、温度をパラメータTEMPFACTOR(0～1の数)倍することで温度を下げる

●初期温度

ランダムな初期解からランダムな近傍を選んだときの目的関数値の増加量 Δ の平均値 $\overline{\Delta}$ とパラメータINTPROBを使って

$$T = -\frac{\overline{\Delta}}{\ln INTPROB}$$ で定める

模擬焼きなまし法

●同一温度内での試行反復回数

近傍の大きさの平均$\overline{N}$ 、解を生成した回数(移動しようとした回数)trials、生成した解が受理された回数changesとする.

このとき同一温度での反復試行は
パラメータSIZEFACTOR$\times \overline{N} <$ trialsかつ
パラメータCUTOFF$\times \overline{N} <$ changesが満たされる限り継続する

●収束判定

カウント変数freezeは受理確率changes/trialsがパラメータMINPERCENT以下になったら1増え、現在までの解が更新されたら0になるとする.

このときfreezeがパラメータFREEZELIMに達したら試行は終了

模擬焼きなまし法

実践的アルゴリズム

1 $x :=$ 適当な実行可能解
2 $f^* :=$ 適当な大きな値(上界)
3 $\overline{N} :=$ 近傍の大きさの平均値
4 初期温度 $T > 0$ を $T = -\dfrac{\overline{\Delta}}{\ln INTPROB}$ により決定

5 *while* $freeze < FREEZELIM$ *do*
6 $changes := 0;\ trials := 0$
7 *while* $trials < SIZEFACTOR \cdot \overline{N}$ and $changes < CUTOFF \cdot \overline{N}$ *do*
8 $trials := trials + 1$
9 解 x のランダムな近傍解 y を選択
10 $\Delta = f(y) - f(x)$
11 *if* $\Delta \le 0$ *then* (受理)
12 $changes := changes + 1;\ x = y$
13 *if* $f(x) < f^*$ *then* $f^* := f(x);\ x^* := 0$

模擬焼きなまし法

実践的アルゴリズム～続き～

```
14          if Δ > 0 then
15            [0,1] の一様乱数 r を選択
16            if r ≤ e^{-Δ/T} then      （受理）
17                changes := changes + 1; x = y
18      T := T · TEMPFACRTOR
19      if 7行目からの while ループで f* が更新された then freeze := 0
20      if changes/trials < MINPERCENT then freeze := freeze + 1
21  return x*
```

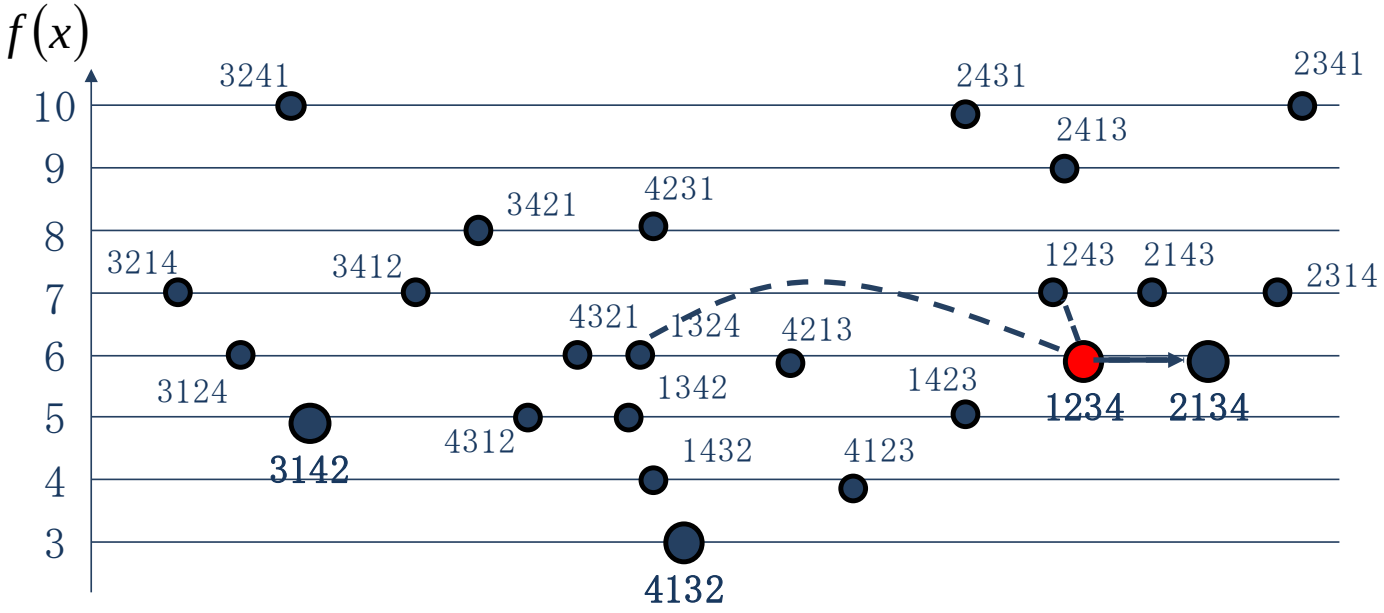
生成した解の受理する方法で別の方法が提案されている

- **閾値受理法**: Δ があらかじめ設定した値より小さければ受理
- **大洪水法**: 近傍解 $f(y)$ が現在の温度T_k よりも小さければ受理
- **旅行記録法**: 近傍解 $f(y)$ が現在の最良解 f^* と比べてそれほど悪くなければ、すなわち$f(y) < f^* + \theta$ であれば受理

禁断探索法

●禁断探索法（Tabu Search）

前に通過した地点に戻ることを避けるために禁断リスト（Tabu List）に保持し、解を更新する手法.
近傍 $N(x)$ の中から目的関数の値を最も改善する x' を選択する.（改善解がなくても最も改悪しないものを選択）

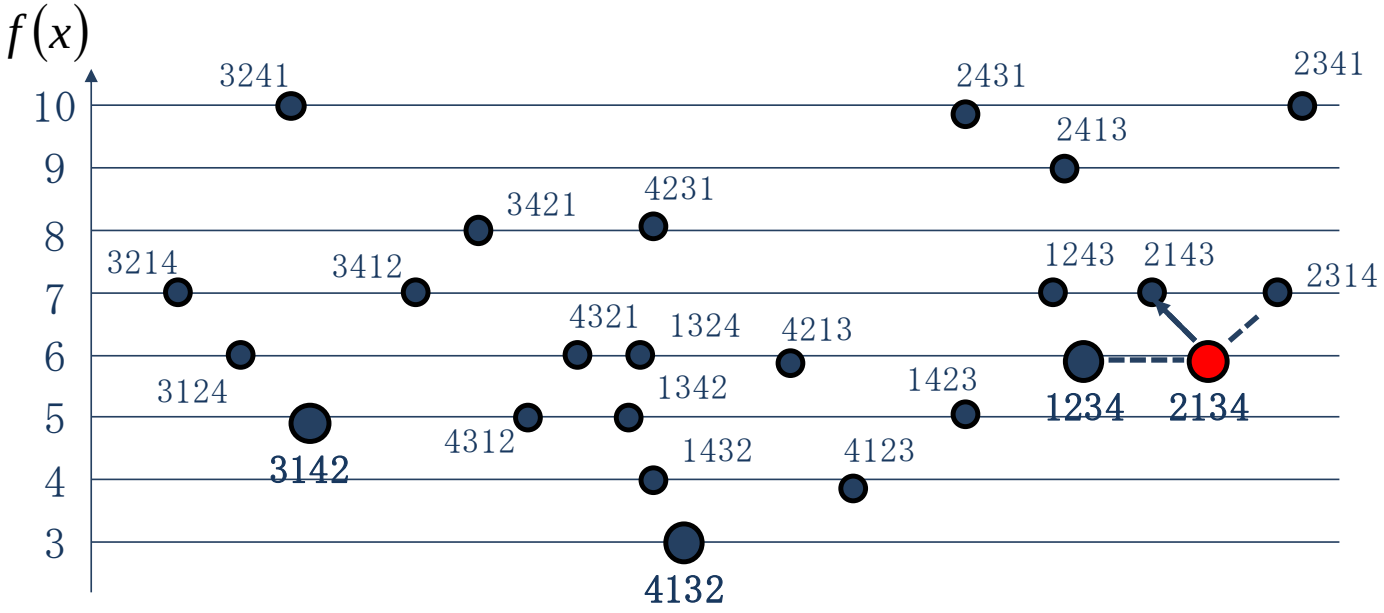


禁断リスト
1、2の入れ替え

禁断探索法

●禁断探索法（Tabu Search）

前に通過した地点に戻ることを避けるために禁断リスト（Tabu List）に保持し、解を更新する手法.
近傍 $N(x)$ の中から目的関数の値を最も改善する x' を選択する.（改善解がなくても最も改悪しないものを選択）

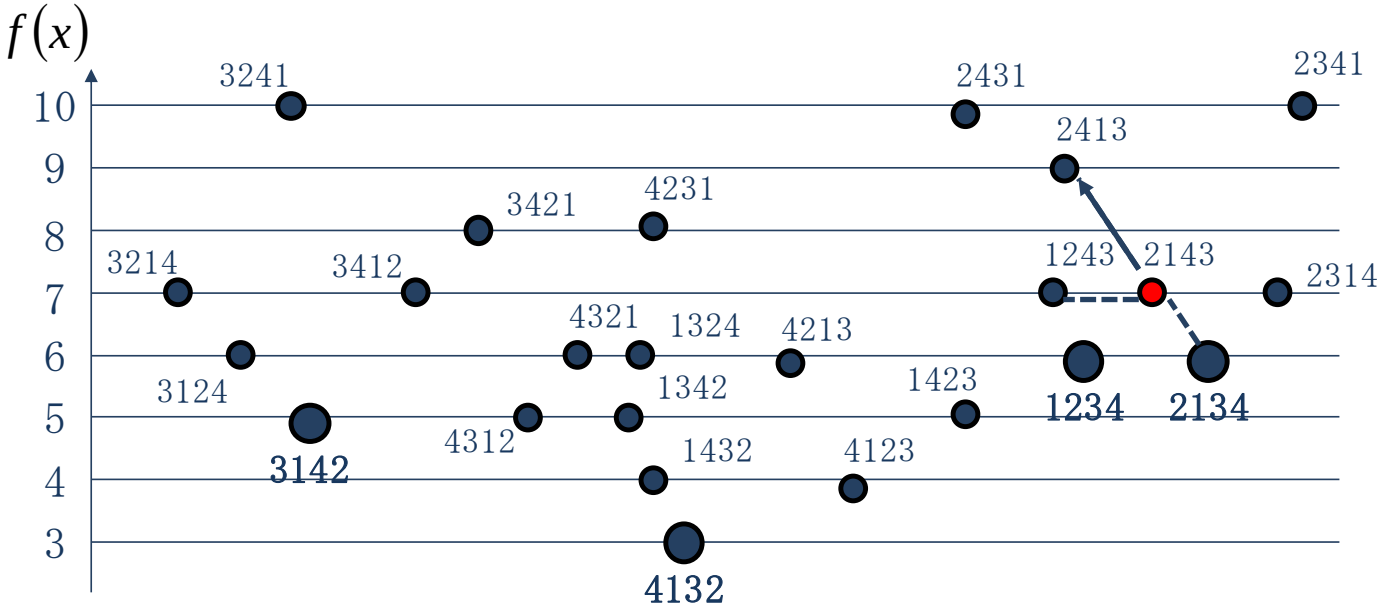


禁断リスト
1、2の入れ替え
3、4の入れ替え

禁断探索法

●禁断探索法（Tabu Search）

前に通過した地点に戻ることを避けるために禁断リスト（Tabu List）に保持し、解を更新する手法.
近傍 $N(x)$ の中から目的関数の値を最も改善する x' を選択する.（改善解がなくても最も改悪しないものを選択）

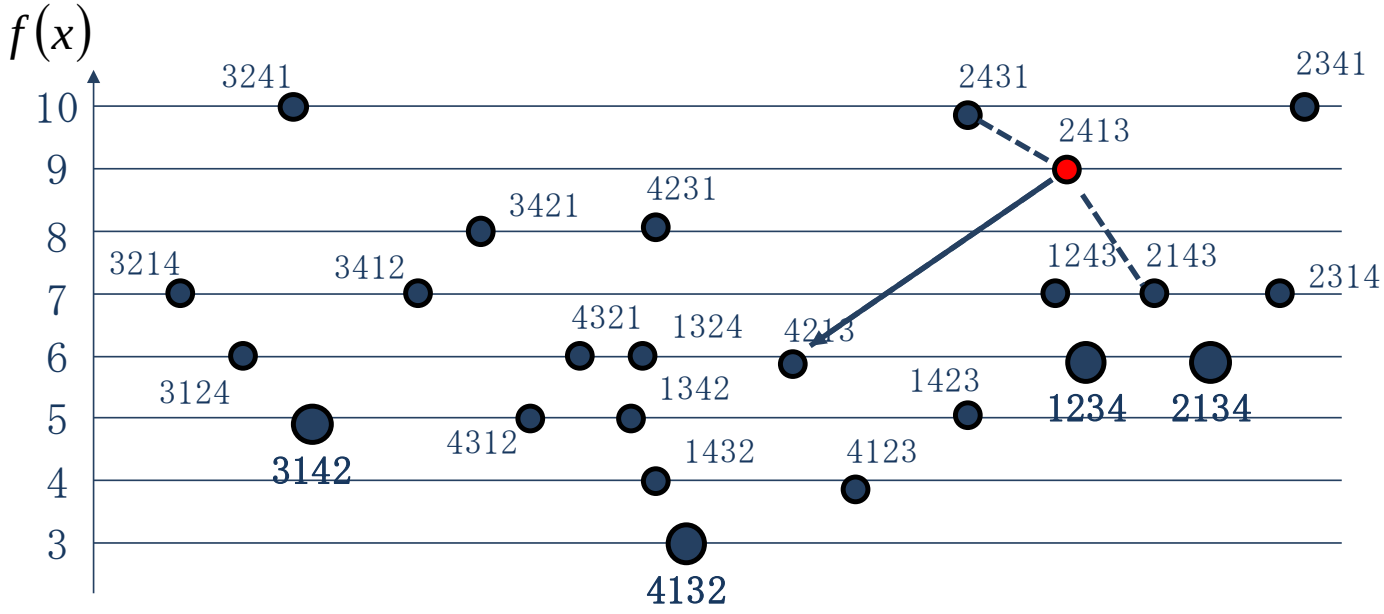


禁断リスト
3、2の入れ替え
3、4の入れ替え

禁断探索法

●禁断探索法（Tabu Search）

前に通過した地点に戻ることを避けるために禁断リスト（Tabu List）に保持し、解を更新する手法.
近傍 $N(x)$ の中から目的関数の値を最も改善する x' を選択する.（改善解がなくても最も改悪しないものを選択）

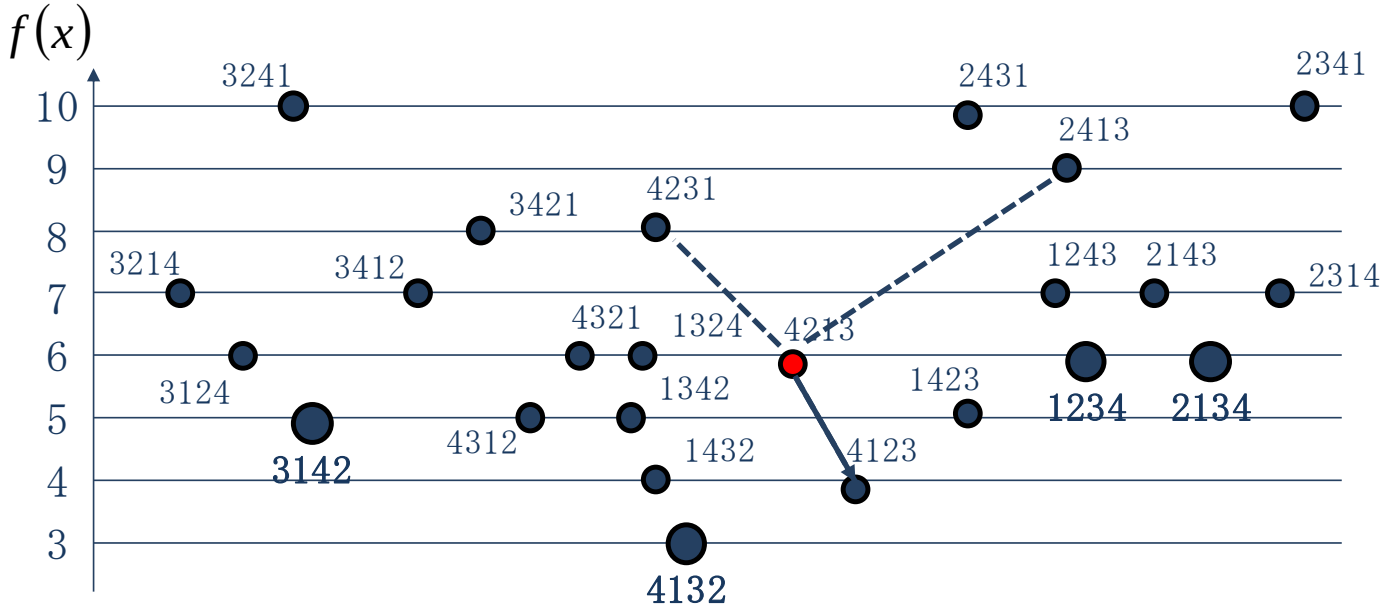


禁断リスト
3、4の入れ替え
2、4の入れ替え

禁断探索法

●禁断探索法（Tabu Search）

前に通過した地点に戻ることを避けるために禁断リスト（Tabu List）に保持し、解を更新する手法.
近傍 $N(x)$ の中から目的関数の値を最も改善する x' を選択する.（改善解がなくても最も改悪しないものを選択）

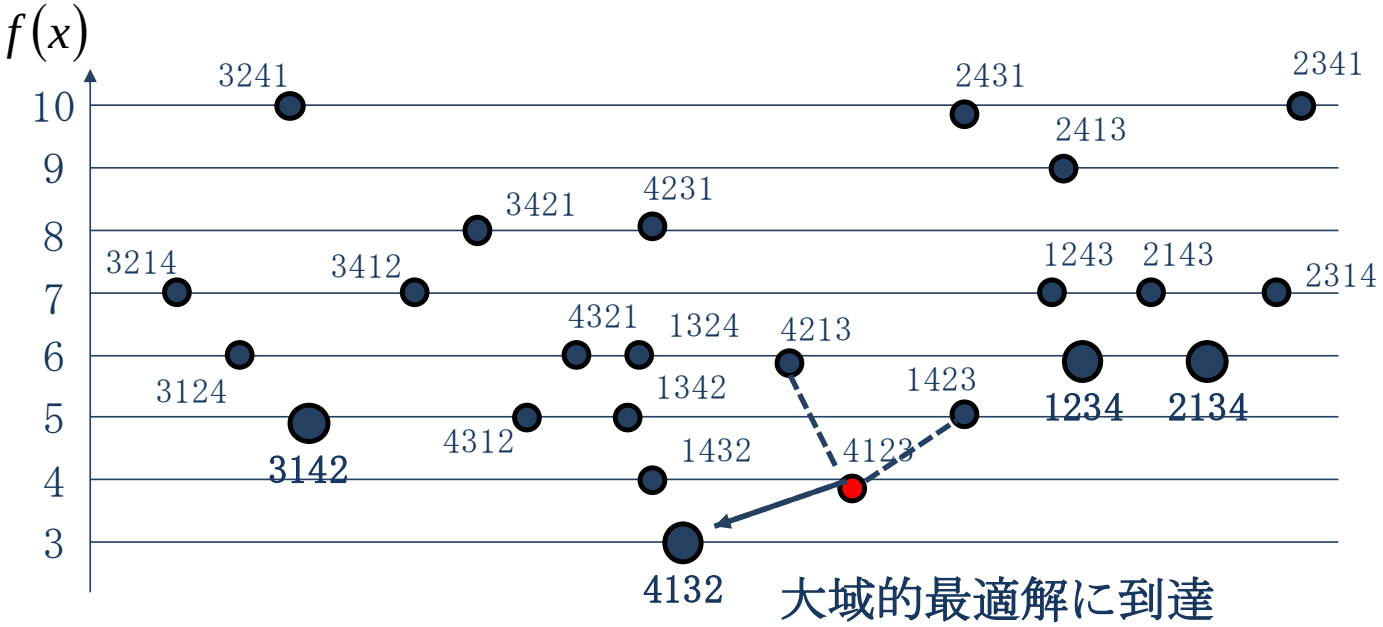


禁断リスト
2、4の入れ替え
2、4の入れ替え

禁断探索法

●禁断探索法（Tabu Search）

前に通過した地点に戻ることを避けるために禁断リスト（Tabu List）に保持し、解を更新する手法.
近傍 $N(x)$ の中から目的関数の値を最も改善する x' を選択する.（改善解がなくても最も改悪しないものを選択）



禁断リスト
2、4の入れ替え
1、2の入れ替え

禁断探索法

$$move(x) = \underset{x' \in N(x) \setminus Tabu}{\arg\min} \; f(x')$$

→近傍のうちタブーリスト
に入っているものを除いた中の
最小のものを選択する

アルゴリズム

1 $t := 0$
2 $x_0 :=$ 適当な初期解
3 $Tabu := \phi$ →禁断リストの初期化
4 $TL :=$ 正の定数(パラメータ)　(TL:リストの長さ)
5 *while* 終了基準が満たされていない *do*
6 $x_{t+1} := move(x_t)$
7 $Tabu := Tabu \cup \{x_t\} \setminus \{x_{t-TL}\}$ →選択された数列のうちリスト長分だけ保存する
8 $t := t+1$
9 $x^* :=$ 現在までの最良解
10 *return* x^*

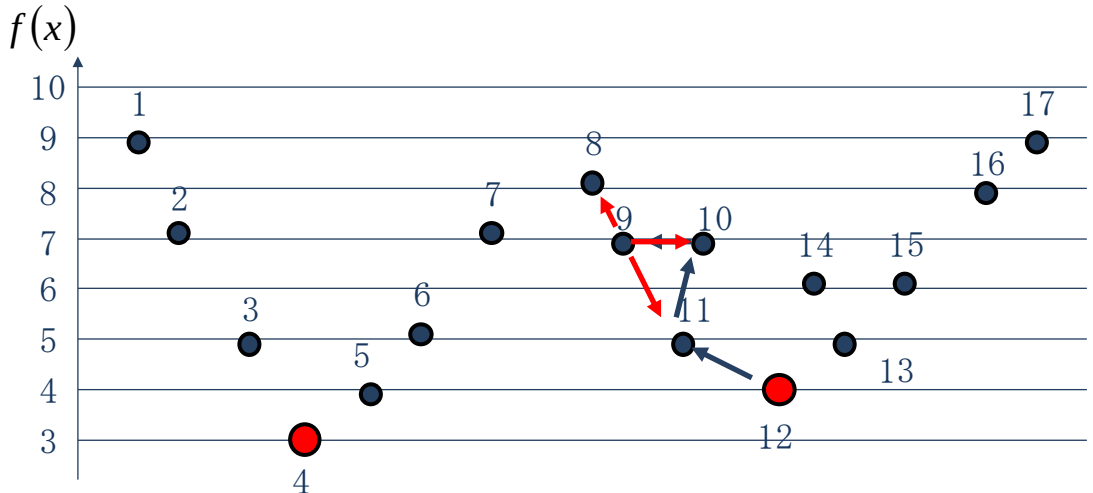
禁断探索法

- 禁断リストの長さTL(タブー長：Tabu Length)、リストに保存する属性(例：交換したジョブの組)などを調節する.

- TLが長すぎると行き場を失う可能性がある

→なんらかの基準を満たせば禁断リストに入っていても移動を許すことが必要

→希求レベル：目的関数値が改善されるなら移動を許すなど



TL=3の場合

禁断リスト

12⇔11の移動

(9⇔11の移動と同じ性質とする)

11⇔10の移動

(8⇔9の移動と同じ性質とする)

9⇔10の移動

禁断探索法

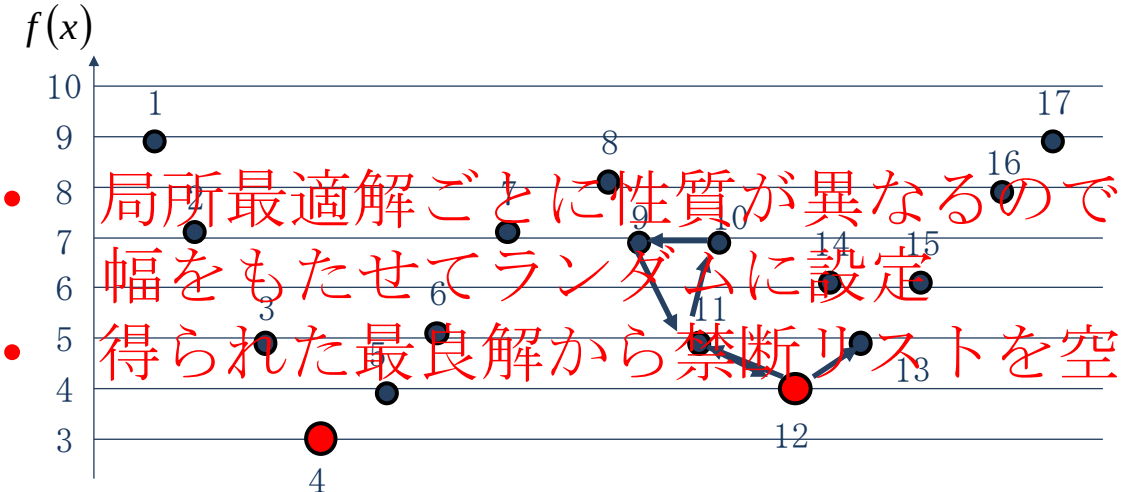
- 禁断リストの長さ**TL(タブー長：Tabu Length)**、リストに保存する**属性**(例：交換したジョブの組)などを調節する.

- TLが**長すぎる**と行き場を失う可能性がある

→なんらかの基準を満たせば禁断リストに入っていても移動を許すことが必要

→**希求レベル**：目的関数値が改善されるなら移動を許すなど

- TLが**短すぎる**と局所最適解から抜け出せないことがある



TL=1の場合

禁断リスト

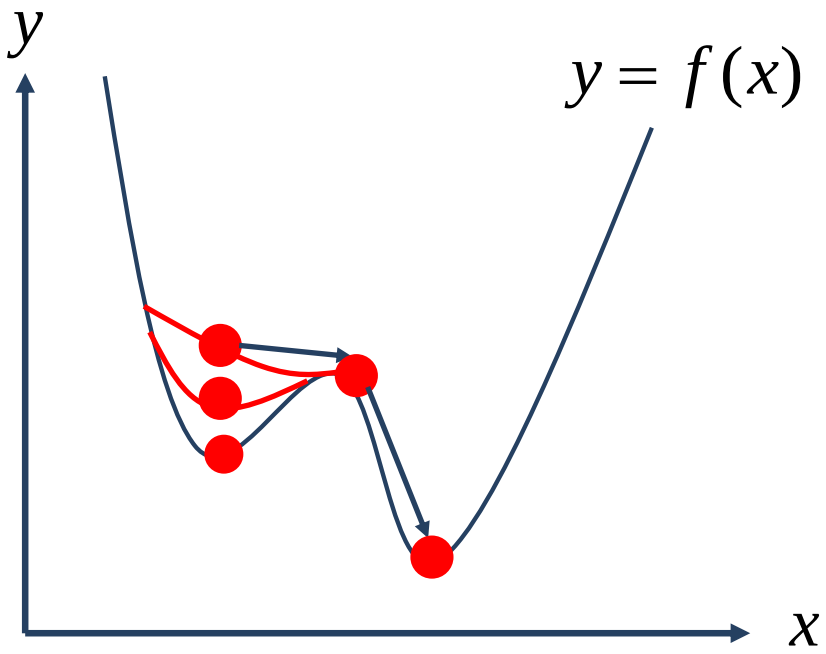
1⇔10の移動

- 局所最適解ごとに性質が異なるので幅をもたせてランダムに設定
- 得られた最良解から禁断リストを空にする

誘導局所探索法

●誘導局所探索法（Guided Local Search）

反復局所探索法の一種. 局所最適解に陥ったとき目的関数値に影響を及ぼす「属性」にペナルティを課すことで局所最適解から脱出する.



もとの目的関数にある「属性」についてペナルティを課した拡張目的関数を用いる

誘導局所探索法

目的関数が各「属性」ごとの線形和で表されるとする

↓

要素の集合を U 、その要素を u 、解 x が要素 u を持つとき1, 持たないとき0の変数を x_u 、要素 u に対するコストを $c(u)$ としたとき、$f(x)$ が $f(x)=\sum_{u\in U} c(u)x_u$ で表される

↓

機械スケジューリング問題で具体的に要素 u やコスト $c(u)$ について考える

誘導局所探索法

ジョブ1が目的関数に与える影響(遅れ)を考えてみる

ジョブ1

処理順番	1	2	3	4
行われたジョブ	なし	2|3|4	23|34|42	234
遅れ	0	0|0|0	1|3|2	5

- ジョブ1が一番目

遅れ→0
要素→ジョブ1が一番目

誘導局所探索法

ジョブ1が目的関数に与える影響(遅れ)を考えてみる

ジョブ1

処理順番	1	2	3	4
行われたジョブ	なし	2|3|4	23|34|42	234
遅れ	0	0|0|0	1|3|2	5

- ジョブ1が二番目、ジョブ2が一番目

  遅れ→0
  要素→ジョブ1が二番目、ジョブ2が二番目

誘導局所探索法

ジョブ1が目的関数に与える影響(遅れ)を考えてみる

ジョブ1

処理順番	1	2	3	4
行われたジョブ	なし	2|3|4	23|34|42	234
遅れ	0	0|0|0	1|3|2	5

- ジョブ1が二番目、ジョブ3が一番目

  遅れ→0
  要素→ジョブ1が二番目、ジョブ2が三番目

誘導局所探索法

ジョブ1が目的関数に与える影響(遅れ)を考えてみる

ジョブ1

処理順番	1	2	3	4
行われたジョブ	なし	2|3|4	23|34|42	234
遅れ	0	0|0|0	1|3|2	5

- ジョブ1が二番目、ジョブ4が一番目

  遅れ→0
  要素→ジョブ1が二番目、ジョブ2が四番目

誘導局所探索法

ジョブ1が目的関数に与える影響(遅れ)を考えてみる

ジョブ1

処理順番	1	2	3	4
行われたジョブ	なし	2|3|4	23|34|42	234
遅れ	0	0|0|0	1|3|2	5

- ジョブ1が三番目、ジョブ2、3が一or二番目

遅れ→1
要素→ジョブ1が三番目、ジョブ2、3が一or二番目

誘導局所探索法

ジョブ1が目的関数に与える影響(遅れ)を考えてみる

ジョブ1

処理順番	1	2	3	4
行われたジョブ	なし	2|3|4	23|34|42	234
遅れ	0	0|0|0	1|3|2	5

- ジョブ1が三番目、ジョブ3、4が一or二番目

  遅れ→3
  要素→ジョブ1が三番目、ジョブ3、4が一or二番目

誘導局所探索法

ジョブ1が目的関数に与える影響(遅れ)を考えてみる

ジョブ1

処理順番	1	2	3		4
行われたジョブ	なし	2|3|4	23|34	42	234
遅れ	0	0|0|0	1|3|2		5

- ジョブ1が三番目、ジョブ4、2が一or二番目

  遅れ→2
  要素→ジョブ1が三番目、ジョブ4、2が一or二番目

誘導局所探索法

ジョブ1が目的関数に与える影響(遅れ)を考えてみる

ジョブ1

処理順番	1	2	3	4
行われたジョブ	なし	2|3|4	23|34|42	234
遅れ	0	0|0|0	1|3|2	5

- ジョブ1が四番目

  遅れ→5
  要素→ジョブ1が四番目

要素 u →ジョブの実行順番と先に行われたジョブの組み

コスト$c(u)$→一つのジョブがもたらす遅れ

誘導局所探索法

ジョブ1

処理順番	1	2	3	4
行われたジョブ	なし	2|3|4	23|34|42	234
遅れ	0	0|0|0	1|3|2	5

ジョブ2

処理順番	1	2	3	4
行われたジョブ	なし	1|3|4	13|34|41	134
遅れ	0	0|0|0	0|0|0	1

ジョブ3

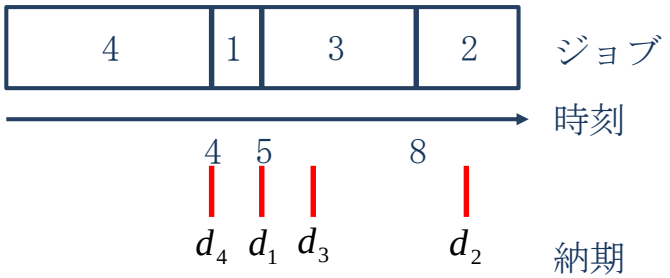
処理順番	1	2	3	4
行われたジョブ	なし	1|2|4	12|24|41	124
遅れ	0	0|0|1	0|3|2	4

ジョブ4

処理順番	1	2	3	4
行われたジョブ	なし	1|2|3	12|23|31	123
遅れ	0	1|2|3	3|5|4	6

□ → $x_u = 1$

それ以外 → $x_u = 0$



$f(x) = 3$

$$f(x) = \sum_{u \in U} c(u) x_u$$

$$= 0 \cdot 1 + 0 \cdot 1 + 2 \cdot 1 + 1 \cdot 1$$

$$= 3$$

誘導局所探索法

要素集合 U 上に長期メモリLMT(Long Term Memory)を定義することで各要素にペナルティを課す

$$\bar{f}(x,\lambda)=\sum_{u\in U}\{c(u)x_u+\lambda\cdot LTM(u)\}$$

$\bar{f}(x,\lambda)$: 拡張目的関数

λ : パラメータ

長期メモリは初期値0で以下のルールに従って増加していく

$$u'=\arg\max_{u\in U}\frac{c(u)}{1+LTM(u)}$$

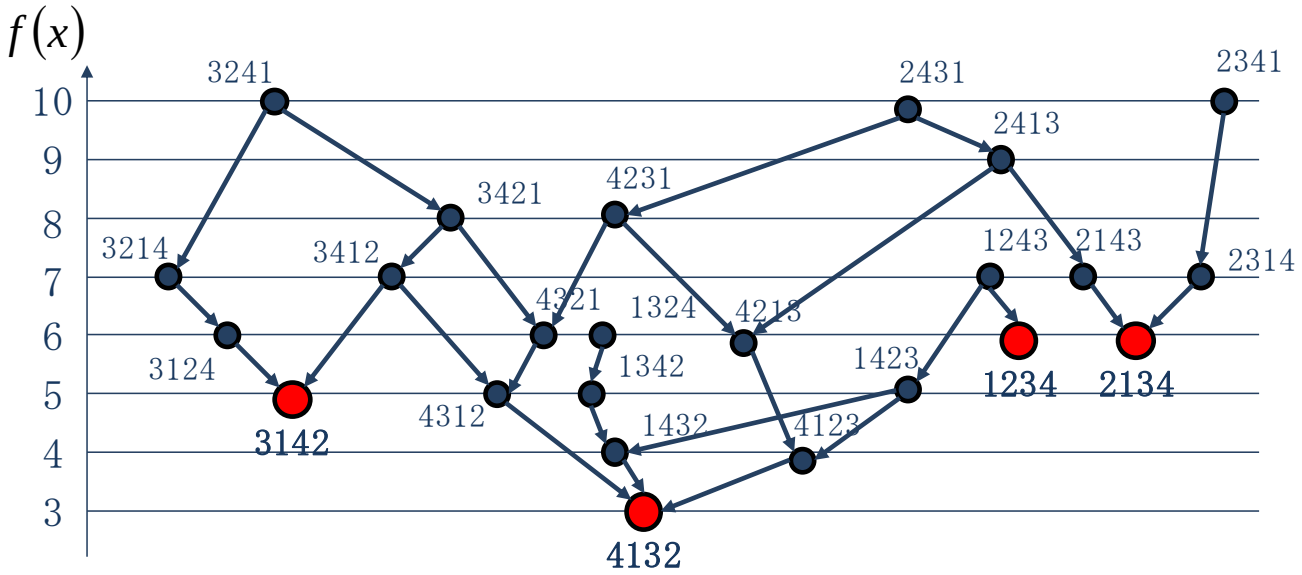
→ 初期段階では最もコストが大きい要素が選択されることになる

↓

$$LTM(u'):=LTM(u')+1$$

→ 初期段階では最もコストが大きい要素にペナルティがまず課されることになる

誘導局所探索法



2341からスタート→局所最適解2134に到達

$$f(x)=\sum_{u\in U} c(u)x_u$$

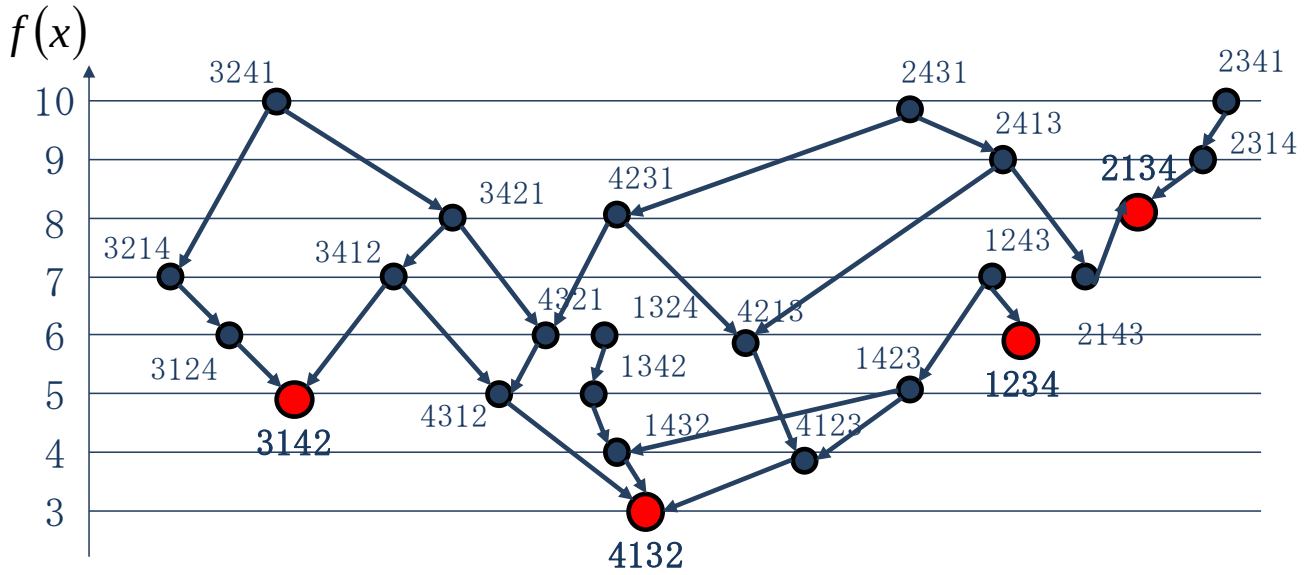
$$=0\cdot 1+0\cdot 1+0\cdot 1+\boxed{6\cdot 1}$$

$$=6$$

$$u'=\arg\max_{u\in U}\frac{c(u)}{1+LTM(u)} \rightarrow u=$$ ジョブ4が四番目

$$LTM(\text{ジョブ4が四番目})\coloneqq LTM(u')+1=1 \qquad \lambda=2$$

誘導局所探索法



$$\bar{f}(x,\lambda)=\sum_{u\in U}\{c(u)x_u+\lambda\cdot LTM(u)\}$$

$$=\{0\cdot 1\}+\{0\cdot 1\}+\{0\cdot 1\}+\{6\cdot 1+2\cdot 1\}$$

λ　$LTM(u)$

ペナルティを与えることで局所最適解でなくなった

誘導局所探索法

$$\lambda = \alpha \times \left\{ \frac{\sum_{u \in U^+} c(u)}{|U^+|} \right\}$$

U^+ : 要素の集合 U のうち $c(u) > 0$ の部分集合

α : 0～1の値をとる平準化パラメータ
近傍が大きい→小さめ(0.1程度)
近傍が小さい→大きめ(0.5程度)

$$improve(x, \lambda) = \begin{cases} \bar{f}(x', \lambda) < \bar{f}(x, \lambda) \text{ を満たす適当な } x' \in N(x) & if \quad x' \text{ が存在} \\ \phi & \text{それ以外} \end{cases}$$

アルゴリズム

```
1  x := 適当な初期解
2  最良解  x* := x
3  長期メモリ LTM(u) ∀u∈U を0に初期化
4  while 終了基準が満たされていない do
5      while improve(x, λ) ≠ φ do
6          x := improve(x, λ)
```

局所探索法

誘導局所探索法

アルゴリズム～続き～

7　得られた局所最適解 x を用いて長期メモリ $LTM(u)$ を更新

8

$$u' = \arg\max_{u \in U} \frac{c(u)}{1 + LTM(u)}$$

$$LTM(u') := LTM(u') + 1$$

9　ペナルティ用パラメータ λ の更新

10　*if* $f(x) < f(x^*)$ *then* $x^* := x$

11 *return* x^*

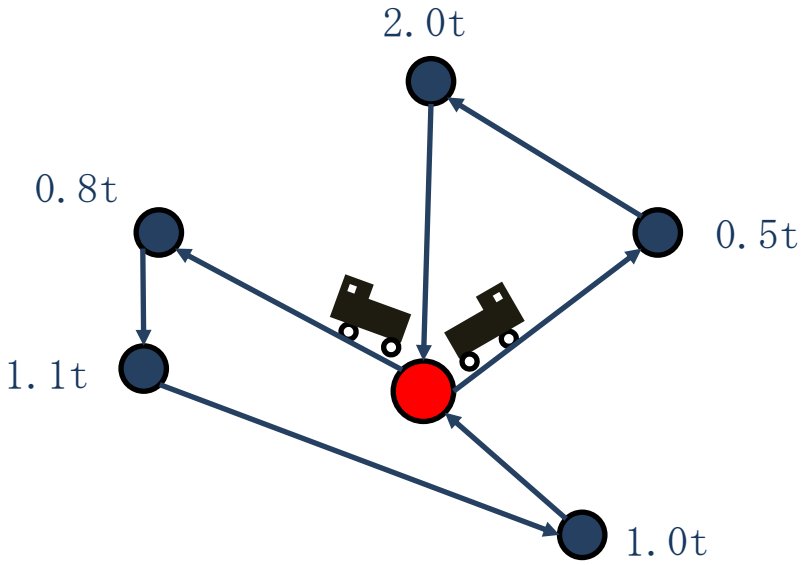
タブーサーチを用いた配送計画システムと配送計画への応用(誘導局所探索法)

- **配送計画問題（Vehicle Routing Problem）**

  デポ、配送先、車両、各配送先の需要が所与のとき最もコストが小さくなるように、配送先割り付け、配送順を決定する問題


×2

1台で3t運べるとする



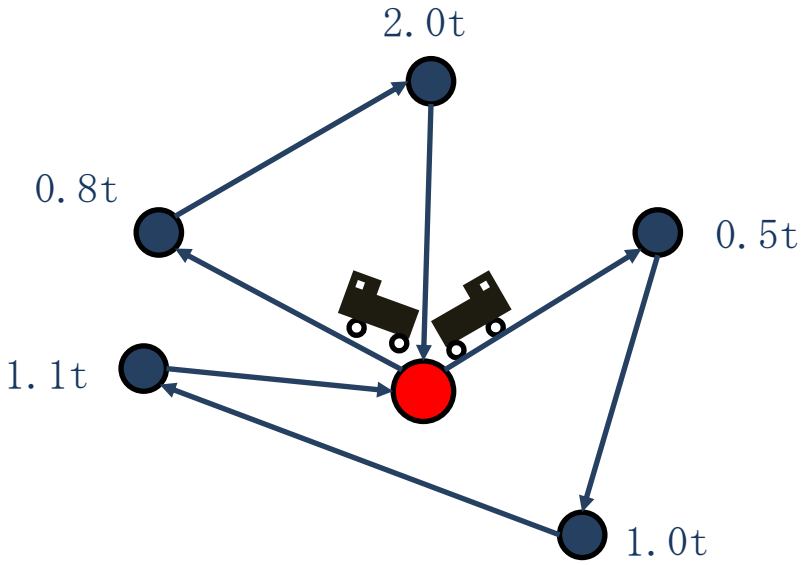
タブーサーチを用いた配送計画システムと配送計画への応用(誘導局所探索法)

●配送計画問題（VRP： Vehicle Routing Problem）

デポ、配送先、車両、各配送先の需要が所与のとき最もコストが小さくなるように、配送先割り付け、配送順を決定する問題


×2

1台で3t運べるとする



その他制約

- 時間制約
- 通行制限

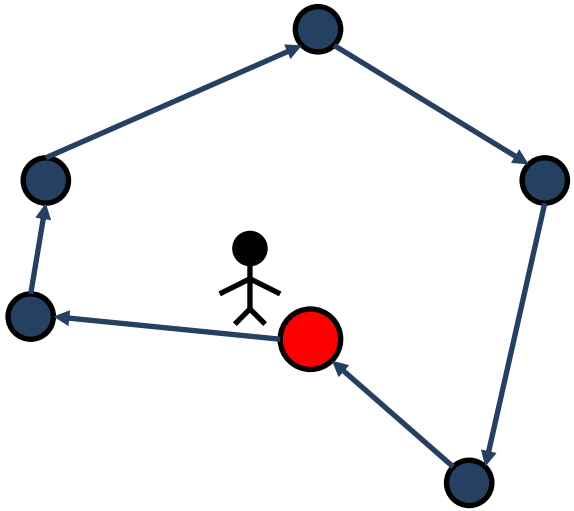
タブーサーチを用いた配送計画システムと配送計画への応用(誘導局所探索法)

●巡回セールスマン問題（TSP： Tranveling Salesman Problem）

訪問する都市が与えられたとき、それらをそれぞれ一回ずつ訪問して出発地に戻る最短ルートを決定する問題

VRPにおいて目的関数が総移動距離、車両が1台配送時刻、配送物などの制約がなくなったもの

↓



改善法における初期解の設定が難しい

タブーサーチを用いた配送計画システムと配送計画への応用(誘導局所探索法)

局所探索法アルゴリズム

```
1  x := 適当な初期解
2  while improve(x) ≠ φ do
3      x := improve(x)
4  return x
```

→初期解の作成

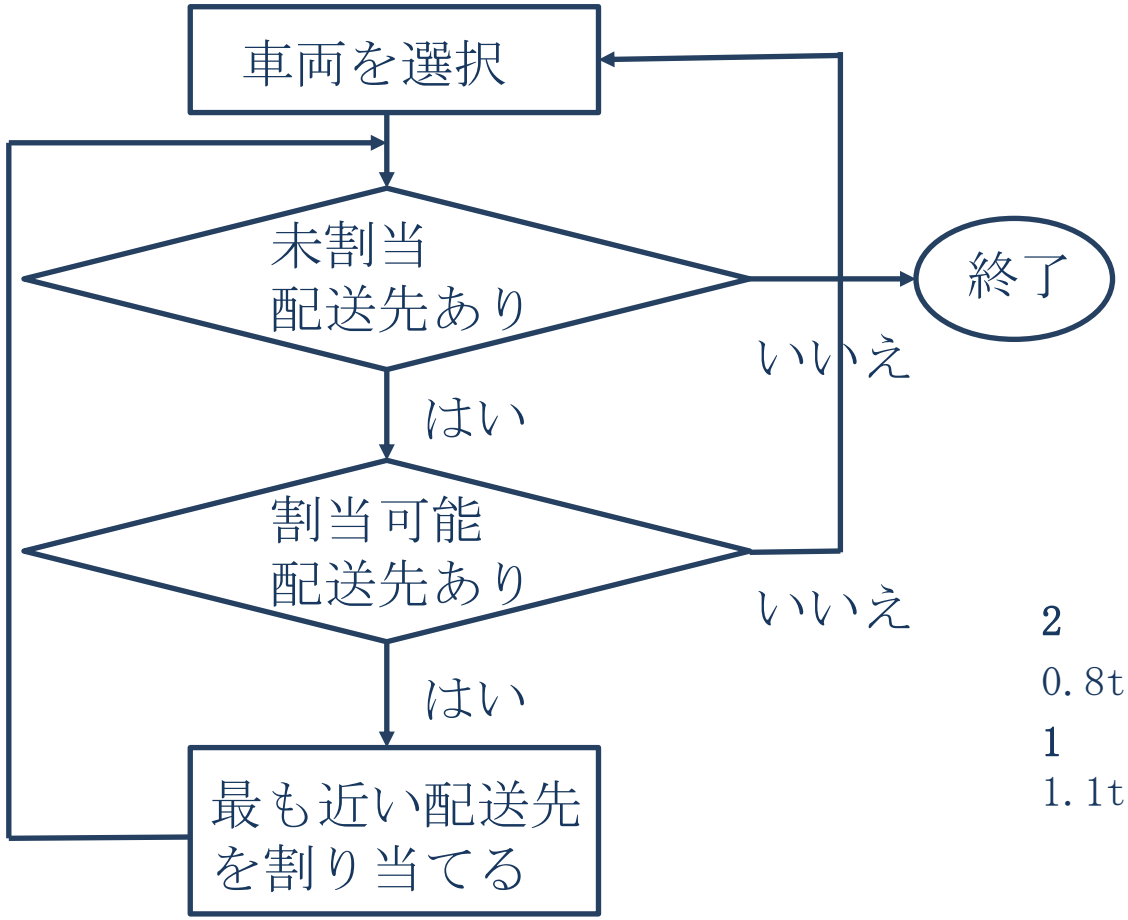
→近傍の構成

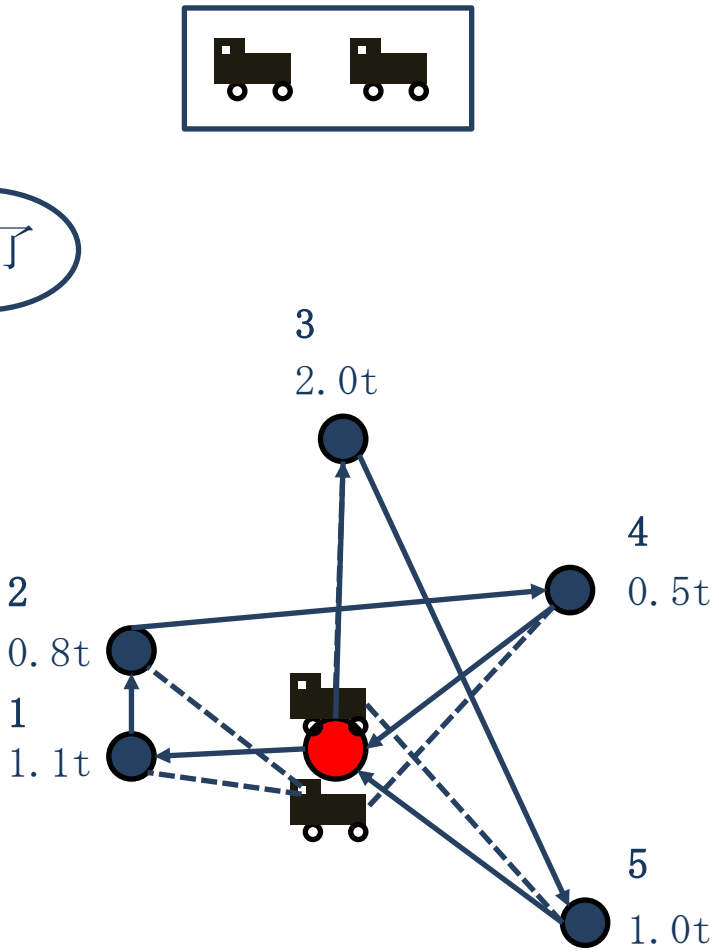
→解の評価

それぞれの要素が配送計画問題でどのように扱われているか説明する

タブーサーチを用いた配送計画システムと配送計画への応用(誘導局所探索法)

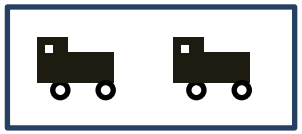
初期解の作成～その1～





タブーサーチを用いた配送計画システムと配送計画への応用(誘導局所探索法)

初期解の作成～その2 セービング法～



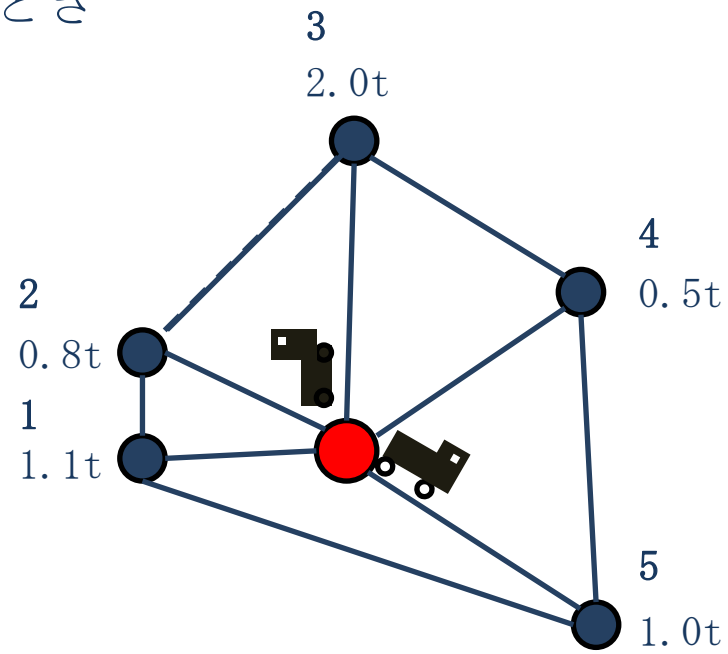
1:配送先と同じ台数の車両があるとしすべての配送先との往復ルートをつくる

↓

2:需要を超過しない配送先同士をつないだときに最も距離が小さいものからつなぐ

↓

3:車両台数になるまで継続する



タブーサーチを用いた配送計画システムと配送計画への応用(誘導局所探索法)

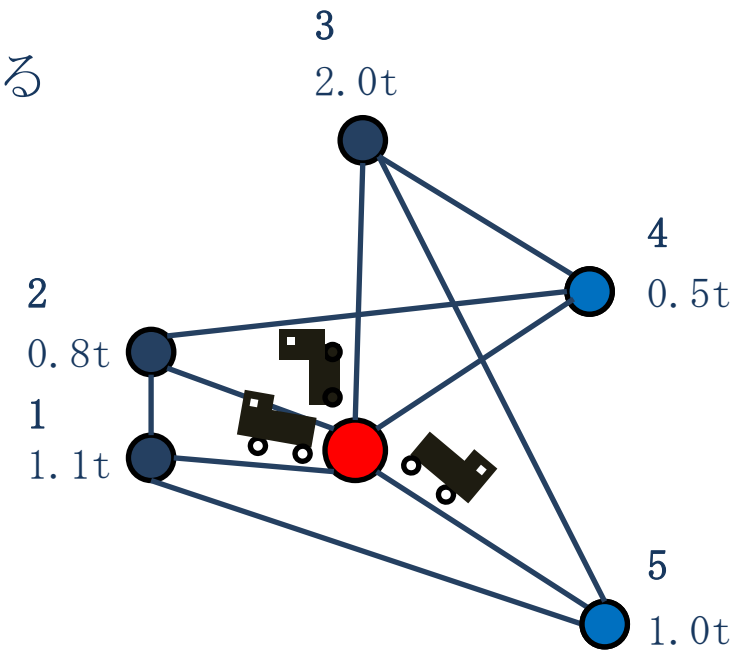
近傍の構成

初期解の一部を変更することで解の改善を行う→近傍に移動

1. 車両割り当て変更

(a)配送先の交換

二つの車両間で配送先を交換する



タブーサーチを用いた配送計画システムと配送計画への応用(誘導局所探索法)

近傍の構成

初期解の一部を変更することで解の改善を行う→近傍に移動

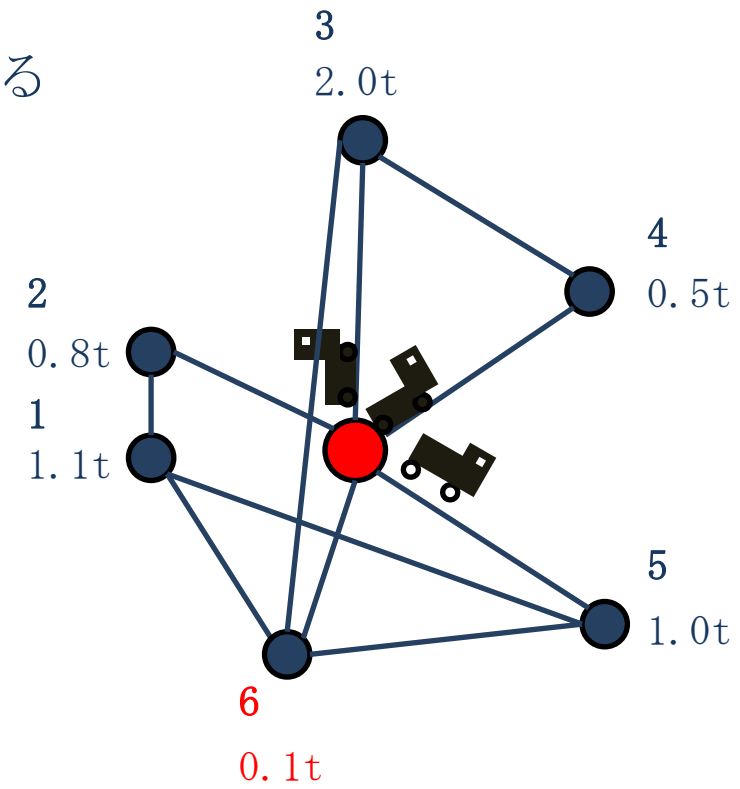
1. 車両割り当て変更

(a)配送先の交換

二つの車両間で配送先を交換する

(b)配送先の移動(削除・追加)

車両の配送先を一つ削除し別の車両に追加する



タブーサーチを用いた配送計画システムと配送計画への応用(誘導局所探索法)

近傍の構成

初期解の一部を変更することで解の改善を行う→近傍に移動

1. 車両割り当て変更

   (a)配送先の交換

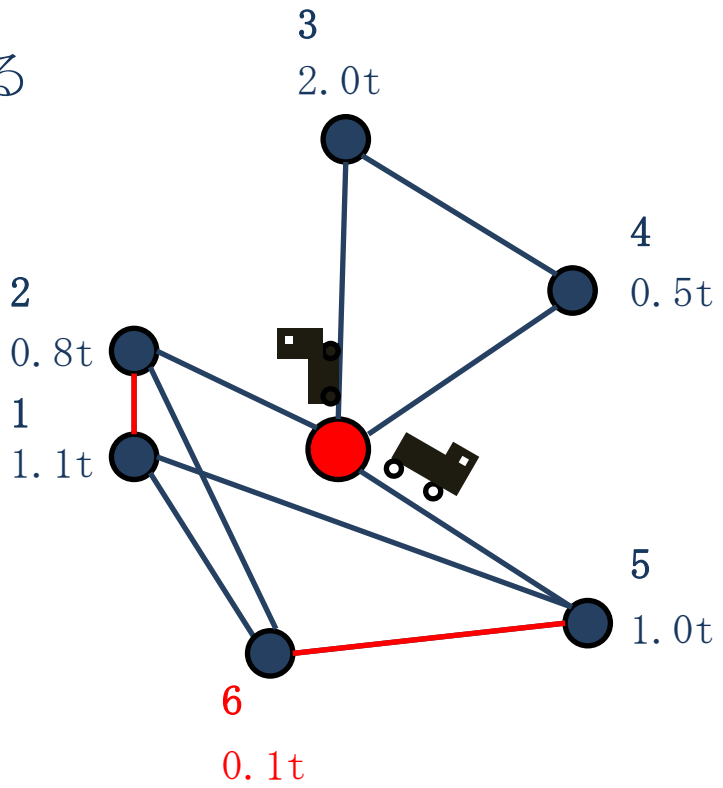
   二つの車両間で配送先を交換する

   (b)配送先の移動(削除・追加)

   車両の配送先を一つ削除し別の車両に追加する

2. 配送順序の変更

   2-opt法などを用いる



1、2のいづれかもしくは両方を用いて近傍を構成する

タブーサーチを用いた配送計画システムと配送計画への応用(誘導局所探索法)

近傍解の評価(改善か改悪か)

評価指標

- 使用車両台数
- 車両の総走行距離
- 業務時間(最後の車両が配送センタに到着する時間)
- 配送時間が違反されているかのペナルティ値
- 複数の評価指標の総和

厳密に問題を定義し誘導局所探索法によってVRPの解法を示していく

タブーサーチを用いた配送計画システムと配送計画への応用(誘導局所探索法)

定数・変数の定義

$K = \{0,1,\cdots,m\}$: 車両の集合

$N = \{0,1,\cdots,n\}$: 配送先、デポの集合(デポはN=0)

$d_{ij} \quad (i, j \in N \ and \ i \neq j)$: 点iと点jとの距離→配送先、デポ間の距離

u_k^d : 車両kの距離単価

$c_{ij}^k \ (k \in K \ and \ i \neq j)$: 車両kが点ij間を移動する際のコスト
一般に $c_{ij}^k = d_{ij} u_k^d$

$x_{ij}^k = \begin{cases} 1 \\ 0 \end{cases}$ 車両kが点iから点jへ移動する場合1,それ以外0

タブーサーチを用いた配送計画システムと配送計画への応用(誘導局所探索法)

目的関数

$$\sum_{i,j\in N,i\neq j,k\in K} c_{ij}x_{ij}^{k} = \sum_{i,j\in N,i\neq j,k\in K} d_{ij}u_{k}^{d}x_{ij}^{k}$$

→ルートとして実際に利用したものについてのコストの総和

制約条件

$$\sum_{i\in N,i\neq j,k\in K} x_{ij}^{k} = 1 \quad for \quad {}^{\forall} j \in N$$

$$\sum_{j\in N,i\neq j,k\in K} x_{ij}^{k} = 1 \quad for \quad {}^{\forall} i \in N$$

→どの配送先も1台の車両によって配送される

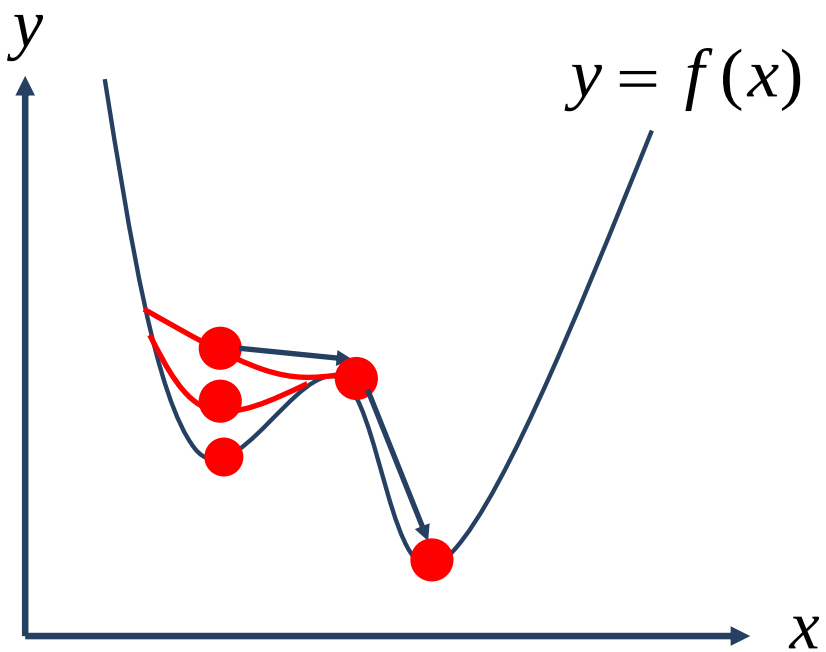
$$\sum_{i\in N,k\in K} x_{i0}^{k} = m$$ デポへの到着

$$\sum_{j\in N,k\in K} x_{0j}^{k} = m$$ デポから出発

→デポへは車両がm回出入りする

タブーサーチを用いた配送計画システムと配送計画への応用(誘導局所探索法)

誘導局所探索法を用いたVRP



ペナルティ更新規則

$$u' = \arg\max_{u \in U} \frac{c(u)}{1 + LTM(u)}$$

$$LTM(u') := LTM(u') + 1$$

もとの目的関数 $f(x)$ が「要素」によって分解されると仮定

↓

要素の集合を U とし $u \in U$ のある要素 u に関するコストを $c(u)$ とする

↓

$$f(x) = \sum_{u \in U} c(u) x_u$$

コストの大きい要素にペナルティ $LTM(u)$ を課し目的関数を拡張

↓

$$\bar{f}(x, \lambda) = \sum_{u \in U} \{c(u) x_u + \lambda \cdot LTM(u)\}$$

タブーサーチを用いた配送計画システムと配送計画への応用(誘導局所探索法)

誘導局所探索法を用いたVRP

VRPにおける要素、要素に関するコストとは?

機械スケジューリング問題の場合
要素: ジョブの処理順
　　とすでに行われたジョブ
コスト: 遅れ

ジョブ1

処理順番	1	2	3	4
行われたジョブ	なし	2|3|4	23|34|42	234
遅れ	0	0|0|0	1|3|2	5

ジョブ2

処理順番	1	2	3	4
行われたジョブ	なし	1|3|4	13|34|41	134
遅れ	0	0|0|0	0|0|0	1

ジョブ3

処理順番	1	2	3	4
行われたジョブ	なし	1|2|4	12|24|41	124
遅れ	0	0|0|1	0|3|2	4

ジョブ4

処理順番	1	2	3	4
行われたジョブ	なし	1|2|3	12|23|31	123
遅れ	0	1|2|3	3|5|4	6

タブーサーチを用いた配送計画システムと配送計画への応用(誘導局所探索法)

VRPにおける要素、要素に関するコストとは?

目的関数 $\sum_{i,j\in N, i\neq j, k\in K} c_{ij}x_{ij}^{k} = \sum_{i,j\in N, i\neq j, k\in K} d_{ij}u_{k}^{d}x_{ij}^{k}$ →関連するインデックスは i、j、k

従って要素集合 U は i 、j 、k の組み合わせの集合となりその要素 u は $u=(i,j,k)$ で表される

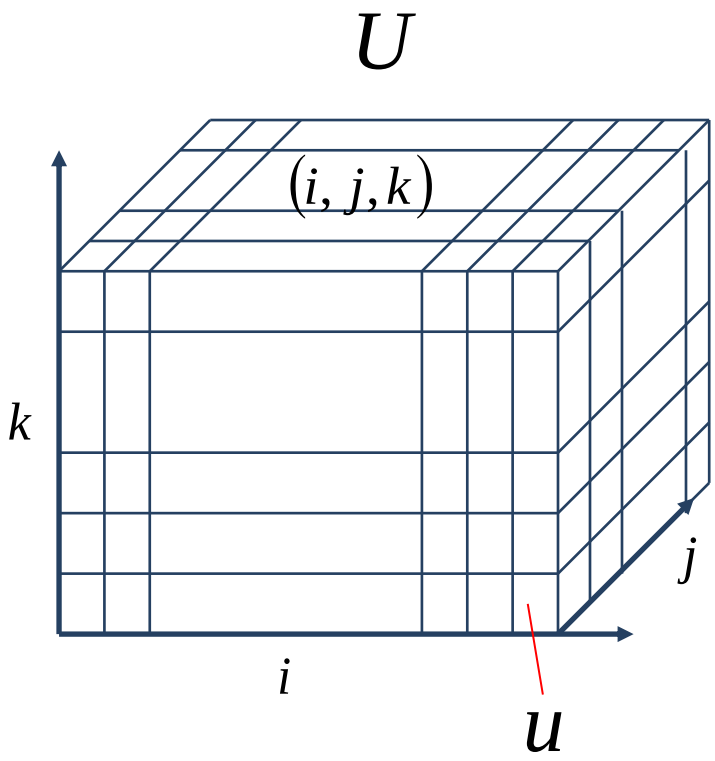
→車両 k で点 ij 間を移動するということが要素になる

コストは車両 k で点 ij 間を移動したときのコストになる→ $d_{ij}u_{k}^{d}$

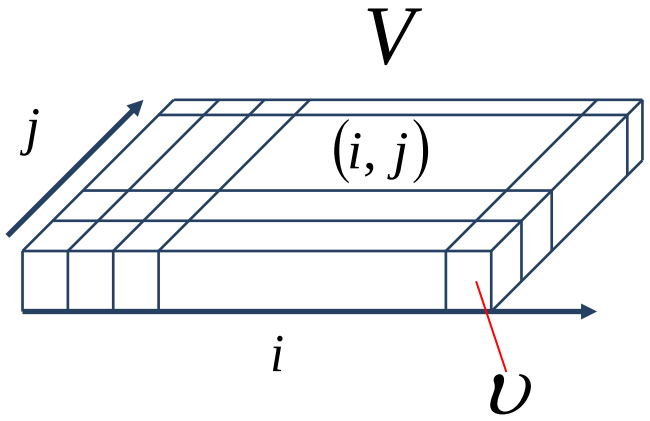
$$f(x)=\sum_{u\in U} c(u)x_{u} = \sum_{i,j\in N, i\neq j, k\in K} d_{ij}u_{k}^{d}x_{ij}$$

実際には要素をさらに分割する

タブーサーチを用いた配送計画システムと配送計画への応用(誘導局所探索法)



要素を分解して
新しい要素をつくる→



車両 k を同一視して都市間の
移動 (i, j) のみを要素とする

新しい要素集合 を V 、その要素を υ 、要素 υ に関するコストを $c(\upsilon)$ 、
解が要素 υ を持つとき1、持たないとき0の決定変数を x_υ とする

$$f(x)=\sum_{u\in U} c(u)x_u = \sum_{i,j\in N, i\neq j, k\in K} d_{ij}u_k^d x_{ij} = \sum_{\upsilon\in V} c(\upsilon)x_\upsilon$$

タブーサーチを用いた配送計画システムと配送計画への応用(誘導局所探索法)

$$LTM(u) \rightarrow LTM'(\upsilon)$$ ペナルティを課す要素が変化

$$\bar{f}(x,\lambda) = \sum_{u \in U}\{c(u)x_u + \lambda \cdot LTM(u)\} \rightarrow \sum_{\upsilon \in V}\{c(\upsilon)x_\upsilon + \lambda \cdot LTM'(\upsilon)\}$$

拡張目的関数

ペナルティ更新規則

$$\upsilon' = \arg\max_{\upsilon \in V} \frac{c(\upsilon)}{1 + LTM'(\upsilon)}$$

$$LTM(\upsilon') := LTM(\upsilon') + 1$$

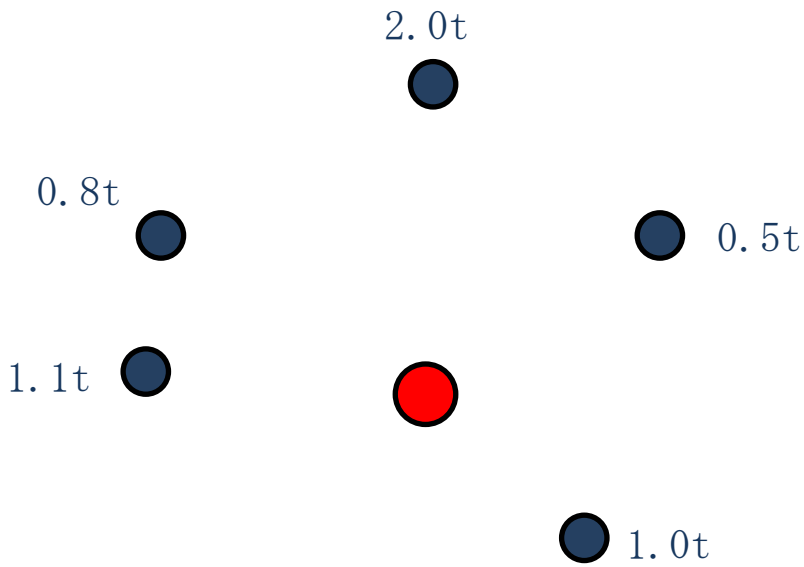
パラメータ更新則

$$\lambda = \alpha \times \left\{ \frac{\sum_{\upsilon \in V^+} c(\upsilon)}{|V^+|} \right\}$$

V^+ : 要素の集合 V のうち $c(\upsilon) > 0$ の部分集合

α : 0～1の値をとる平準化パラメータ

得られた解の要素の中で配送先間の距離が大きいものから優先的にペナルティが課される


2.0t
0.8t
0.5t
1.1t
1.0t

ジョブ1

処理順番	1	2	3	4
行われたジョブ	なし	2|3|4	23|34|42	234
遅れ	0	0|0|0	1|3|2	5

ジョブ2

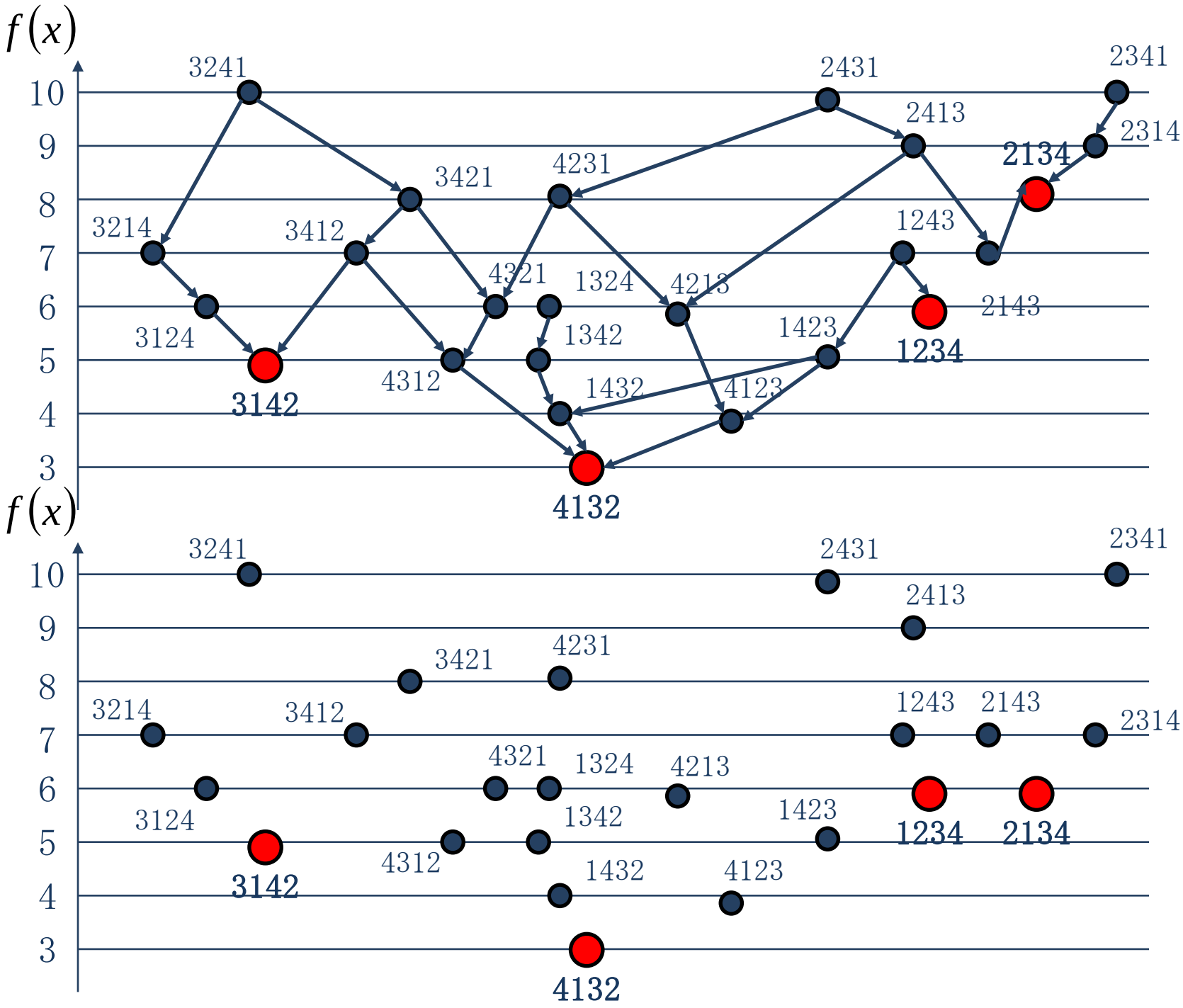
処理順番	1	2	3	4
行われたジョブ	なし	1|3|4	13|34|41	134
遅れ	0	0|0|0	0|0|0	1

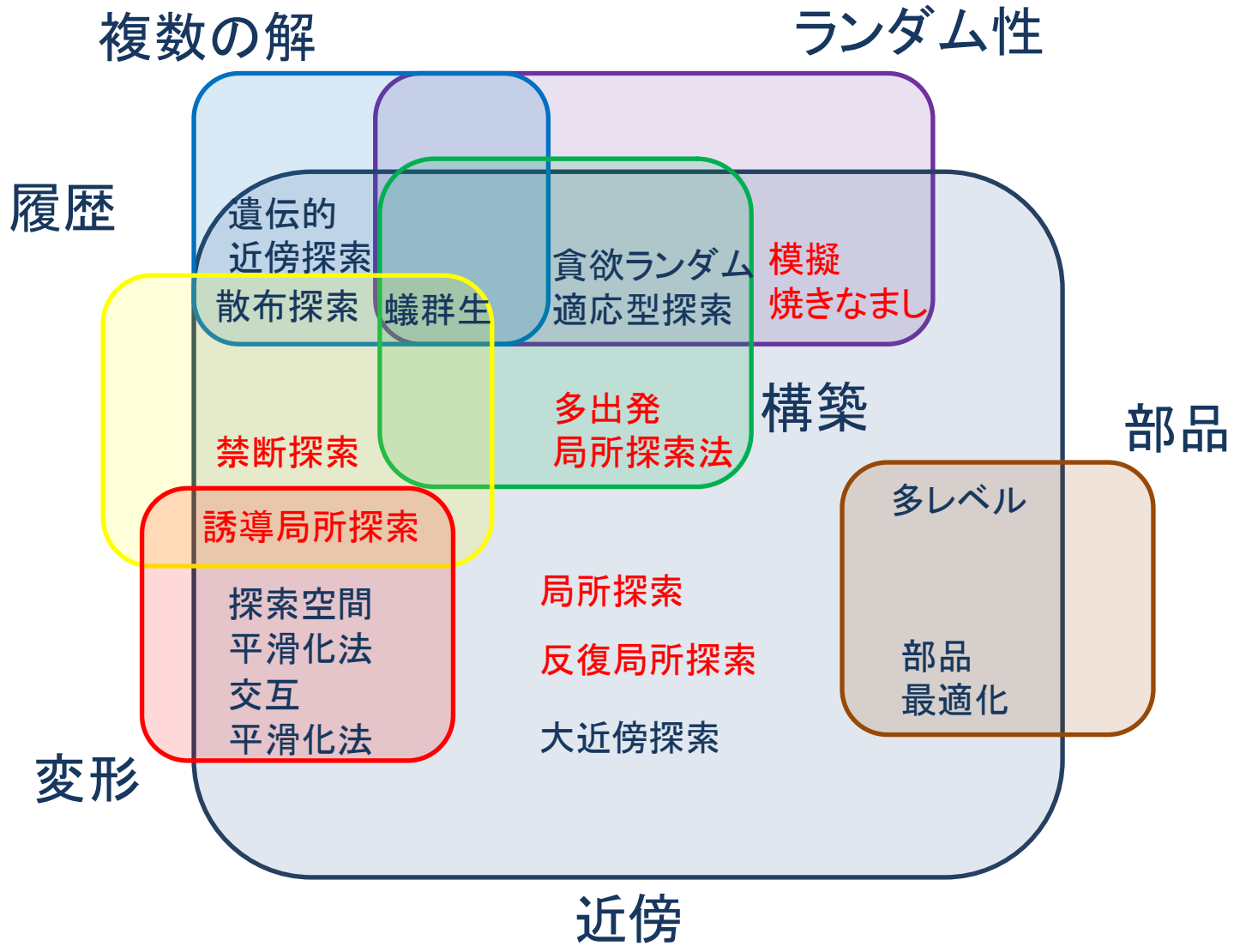
ジョブ3

処理順番	1	2	3	4
行われたジョブ	なし	1|2|4	12|24|41	124
遅れ	0	0|0|1	0|3|2	4

ジョブ4

処理順番	1	2	3	4
行われたジョブ	なし	1|2|3	12|23|31	123
遅れ	0	1|2|3	3|5|4	6


f(x)
10
9
8
7
6
5
4
3
3241
2431
2341
2413
2314
2134
4231
3421
3214
3412
1243
4321
1324
4213
2143
3124
1342
1423
1234
4312
1432
4123
3142
4132
f(x)
10
9
8
7
6
5
4
3
3241
2431
2341
2413
4231
3421
3214
3412
1243
2143
2314
4321
1324
4213
3124
1342
1423
1234
2134
4312
1432
4123
3142
4132


複数の解
ランダム性
履歴
遺伝的
近傍探索
散布探索
蟻群生
貪欲ランダム
適応型探索
模擬
焼きなまし
構築
多出発
局所探索法
禁断探索
部品
多レベル
誘導局所探索
局所探索
探索空間
平滑化法
交互
平滑化法
反復局所探索
部品
最適化
大近傍探索
変形
近傍

模擬焼きなまし法

最適解への収束することの証明

生成された数列 $\{x_k\}$ はマルコフ連鎖 M を導く

※マルコフ連鎖：未来の挙動が現在の値だけで決定され、過去の挙動と無関係である

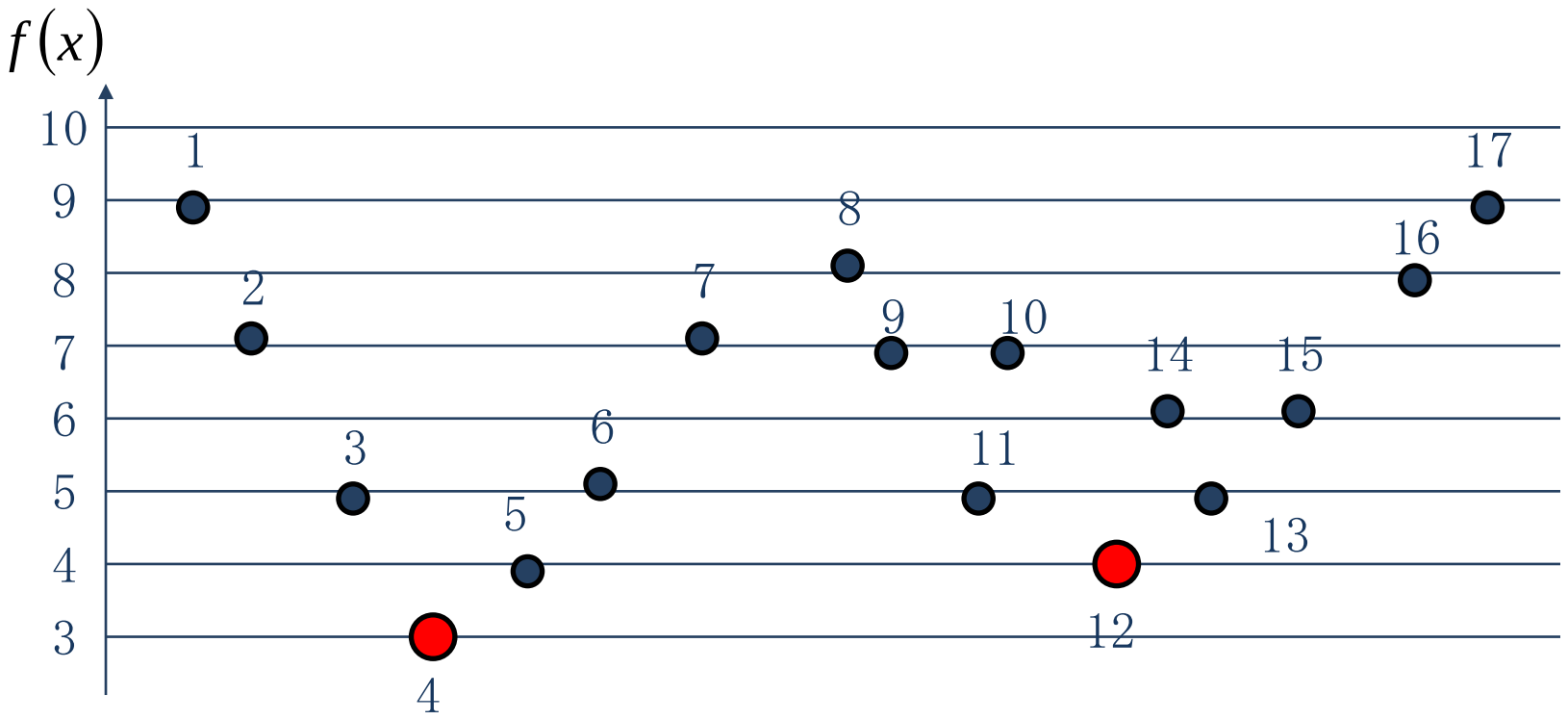
近傍 $N(x)$ の中からランダムに y を選択する確率→ $R(x,y)$
すべての近傍が等確率で選択される
→ $R(x,y)=1/|N(x)|$（但し$|\ |$は中の要素の個数を表す）

マルコフ連鎖 M の推移確率

$$P_k(x,y)=\begin{cases}0\\ R(x,y)\exp\left\{-[f(y)-f(x)]^+/T_k\right\}=1/|N(x)|\exp\left\{-[f(y)-f(x)]^+/T_k\right\}\\ 1-\sum_{x\neq x'}P_k(x,x')\end{cases}$$

模擬焼きなまし法

最適解への収束することの証明



模擬焼きなまし法

実践的模擬焼きなまし法