情報システム評価学
ー整数計画法ー

第８回目：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）

今日の話の流れ

- 近似精度が理論的に保証されたアルゴリズム
  - 0-1ナップサック問題に対する0．5近似
  - 整数ナップサック問題に対する0．5近似
  - 平面上の巡回セールスマン問題に対する2近似
  - 平面上の巡回セールスマン問題に対する1．5近似
- メタヒューリスティックス(metaheuristics)
  - タブー探索法(tabu search)
  - （シミュレーテッド）アニーリング法(simulated annealing)
  - 遺伝アルゴリズム(genetic algorithm)

今日の話の流れ

- 近似精度が理論的に保証されたアルゴリズム
  - 0-1ナップサック問題に対する0．５近似
  - 整数ナップサック問題に対する0．５近似
  - 平面上の巡回セールスマン問題に対する2近似
  - 平面上の巡回セールスマン問題に対する1．５近似
- メタヒューリスティックス(metaheuristics)
  - タブー探索法(tabu search)
  - （シミュレーテッド）アニーリング法(simulated annealing)
  - 遺伝アルゴリズム(genetic algorithm)

近似解の近似比

- 近似解の近似比＝近似解の目的関数値／最適値
  - 近似比＝1←→解は最適
- 最大化問題の場合：
  - 常に近似比≦1
  - α近似アルゴリズム（α≦1）

    ←→ 近似比≧αの近似解を常に求める
- 最小化問題の場合：
  - 常に近似比≧1
  - α近似アルゴリズム（α≧1）

    ←→ 近似比≦αの近似解を常に求める

0-1ナップサック問題に対する0.5近似アルゴリズム

最大化　$c_1x_1 + c_2x_2 + \cdots + c_nx_n$

条件　$a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b$

$x_j \in \{0, 1\}\ (j = 1, \ldots, n)$

ただし $c_1/a_1 \geq c_2/a_2 \geq \cdots \geq c_n/a_n$ と仮定

整数 $k\ (1 \leq k \leq n)$ は

$a_1 + \cdots + a_{k-1} < b \leq a_1 + \cdots + a_{k-1} + a_k$

を満たす

⇩

アイテム集合 $\{1, 2, \ldots, k-1\}$ と $\{k\}$ のうち,

良い方を選ぶと0.5近似解

(近似解の目的関数値$= \max\{\sum_{j=1}^{k-1} c_j, c_k\} \geq 0.5\times$ 最適値)

0-1ナップサック問題に対する0.5近似アルゴリズム

近似比の証明:LP緩和解との比較

整数 k $(1 \leq k \leq n)$ は

$a_1 + \cdots + a_{k-1} < b \leq a_1 + \cdots + a_{k-1} + a_k$ を満たす

⇒ LP緩和解は $\left(1, 1, ..., 1, \underbrace{\frac{b - \sum_{j=1}^{k-1} a_j}{a_k}}_{\text{k番目の要素}}, 0, .., 0\right)$

$$\text{LP緩和解の目的関数値} = \sum_{j=1}^{k-1} c_j + c_k \times \frac{b - \sum_{j=1}^{k-1} a_j}{a_k} \geq \text{元問題の最適値}$$

$$\sum_{j=1}^{k-1} c_j + c_k \times \frac{b - \sum_{j=1}^{k-1} a_j}{a_k} \leq 2\max\{\sum_{j=1}^{k-1} c_j, c_k\} = 2 \times \text{近似解の目的関数値}$$

⇒ $\text{近似解の目的関数値} = \max\{\sum_{j=1}^{k-1} c_j, c_k\} \geq 0.5 \times \text{最適値}$

整数ナップサック問題に対する0.5近似アルゴリズム

最大化 $c_1x_1 + c_2x_2 + \cdots + c_nx_n$

条件 $a_1x_1 + a_2x_2 + \cdots + a_nx_n \le b$

x_jは非負整数 $(j = 1, \ldots, n)$

ただし $c_1/a_1 \ge c_2/a_2 \ge \cdots \ge c_n/a_n$, $a_j \le b$ と仮定

$x_1 = \lfloor b/a_1 \rfloor$, $x_j = 0$ $(j = 2, ..., n)$ は0.5近似解

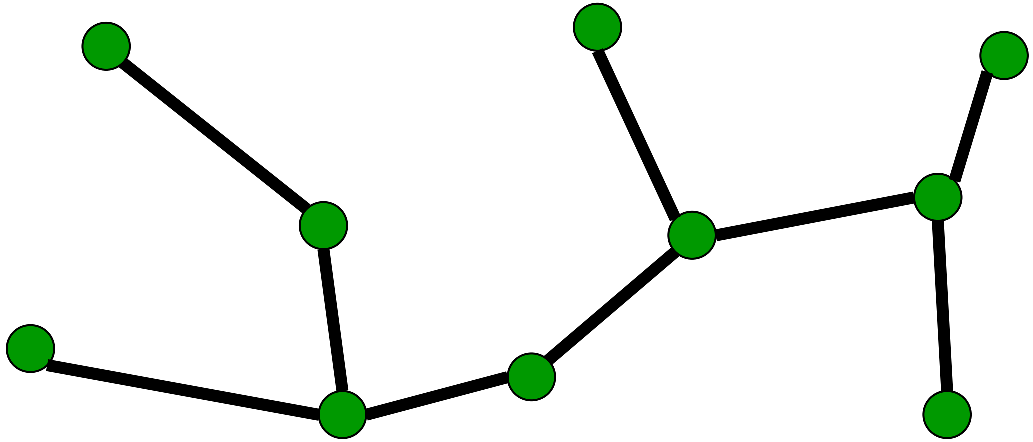
(近似解の目的関数値 $= c_1\lfloor b/a_1 \rfloor \ge 0.5\times$ 最適値)

LP緩和解は $x_1 = b/a_1$, $x_j = 0$ $(j = 2, ..., n)$

$$\underbrace{c_1 \times \lfloor b/a_1 \rfloor}_{\text{近似解の目的関数値}} \le \underbrace{c_1 \times b/a_1}_{\text{LP緩和の目的関数値}} \le c_1 \times (\lfloor b/a_1 \rfloor + 1) \le 2c_1 \times \lfloor b/a_1 \rfloor$$

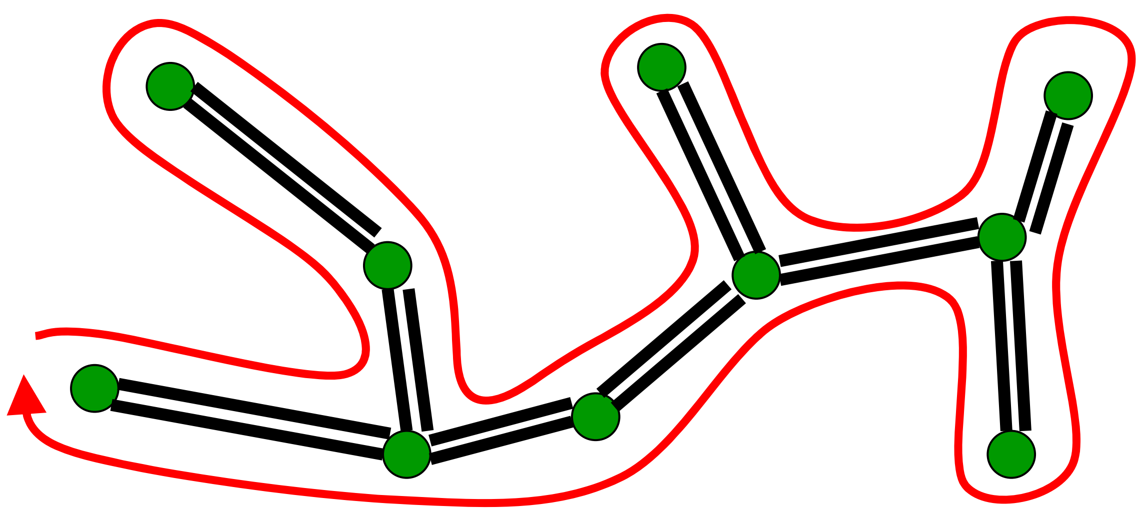
平面上の巡回セールスマン問題に対する2近似アルゴリズム

- 最小全域木を使った近似アルゴリズム
  - 平面上の点に対する最小全域木を求める（枝数＝n-1）



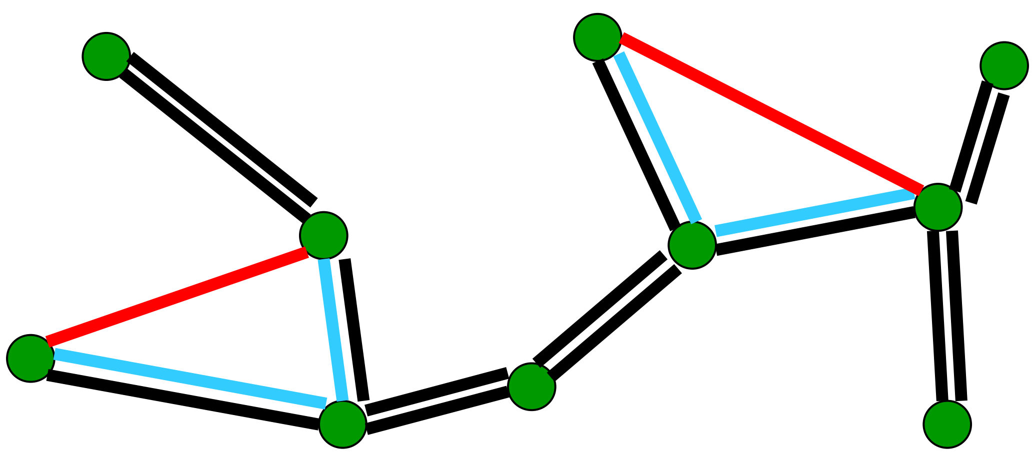
平面上の巡回セールスマン問題に対する2近似アルゴリズム

- 最小全域木を使った近似アルゴリズム
  - 平面上の点に対する最小全域木を求める（枝数＝n-1）
  - 全域木の各枝を2本の枝に置き換え，全頂点を通過する閉路をつくる（枝数＝2（n-1））



平面上の巡回セールスマン問題に対する2近似アルゴリズム

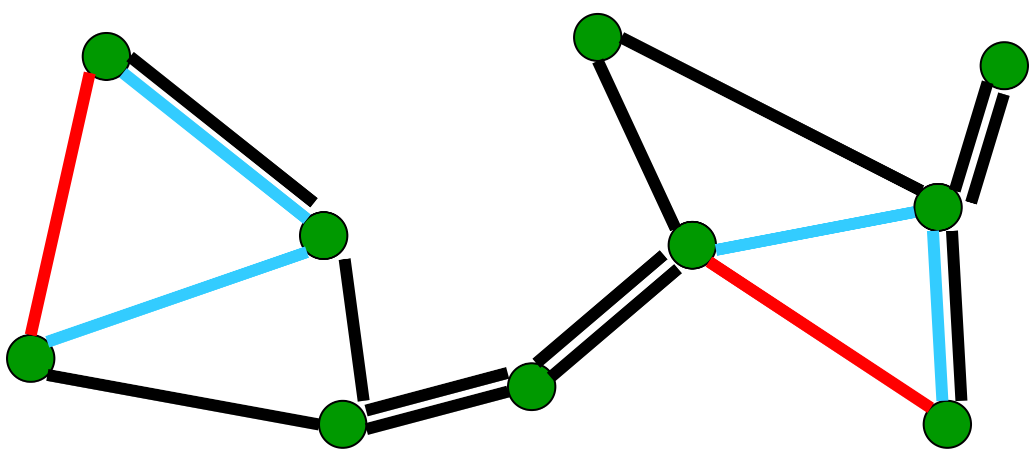
- 最小全域木を使った近似アルゴリズム
  - 平面上の点に対する最小全域木を求める（枝数＝n-1）
  - 全域木の各枝を2本の枝に置き換え，全頂点を通過する閉路をつくる（枝数＝2（n-1））
  - 閉路の無駄な部分をショートカットして，巡回路をつくる



ショーカット1回に付き，枝数が1減少

平面上の巡回セールスマン問題に対する2近似アルゴリズム

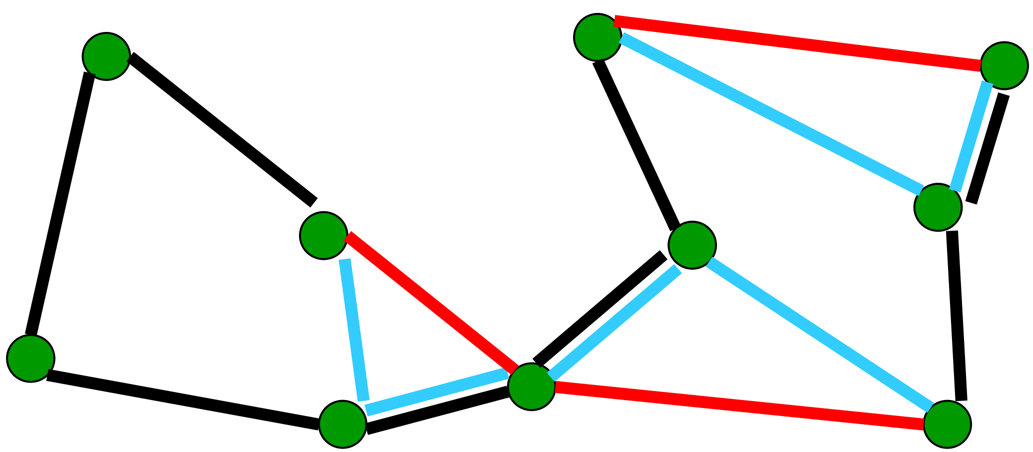
- 最小全域木を使った近似アルゴリズム
  - 平面上の点に対する最小全域木を求める（枝数＝n-1）
  - 全域木の各枝を2本の枝に置き換え，全頂点を通過する閉路をつくる（枝数＝2（n-1））
  - 閉路の無駄な部分をショートカットして，巡回路をつくる



ショーカット1回に付き，枝数が1減少

平面上の巡回セールスマン問題に対する2近似アルゴリズム

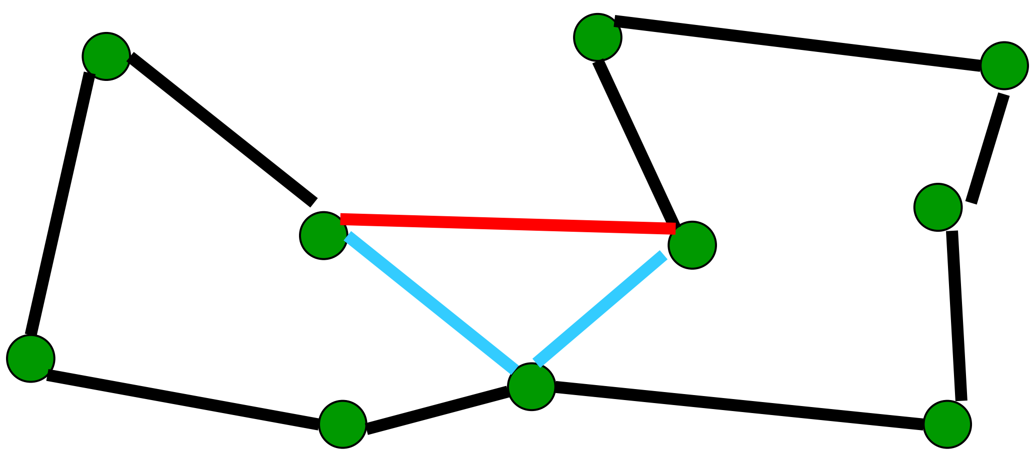
- 最小全域木を使った近似アルゴリズム
  - 平面上の点に対する最小全域木を求める（枝数＝n-1）
  - 全域木の各枝を2本の枝に置き換え，全頂点を通過する閉路をつくる（枝数＝2（n-1））
  - 閉路の無駄な部分をショートカットして，巡回路をつくる



ショーカット1回に付き，枝数が1減少

平面上の巡回セールスマン問題に対する2近似アルゴリズム

- 最小全域木を使った近似アルゴリズム
  - 平面上の点に対する最小全域木を求める（枝数＝n-1）
  - 全域木の各枝を2本の枝に置き換え，全頂点を通過する閉路をつくる（枝数＝２（n-1））
  - 閉路の無駄な部分をショートカットして，巡回路をつくる

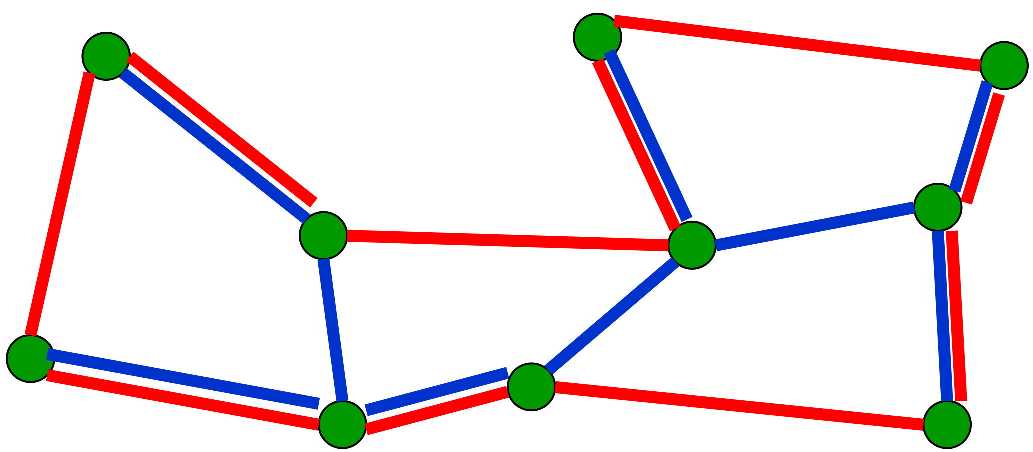


ショーカット１回に付き，枝数が１減少

平面上の巡回セールスマン問題に対する2近似アルゴリズム

- 近似比の証明：示したいこと
  - 最適解の長さ ≧ 最小全域木の長さ
  - 近似解の長さ ≦ 最小全域木の長さ×２

    ➔ 近似解の長さ ≦ 最適解の長さ×２



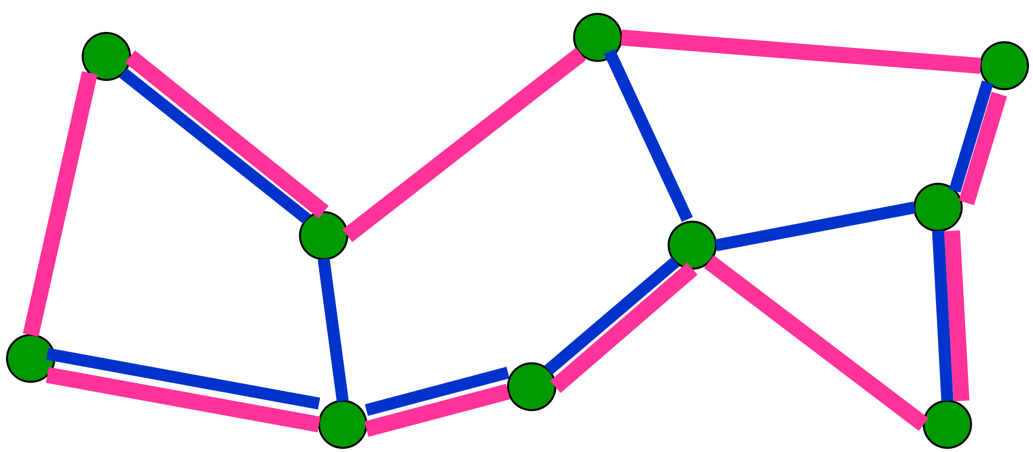
平面上の巡回セールスマン問題に対する2近似アルゴリズム

- 示したいこと: 最適解の長さ ≧ 最小全域木の長さ
  - 巡回路から枝を1本取り除くと全域木

∴ 近似解（巡回路）の長さ

≧ 近似解から枝を1本取り除いたものの長さ

≧ 最小全域木の長さ



平面上の巡回セールスマン問題に対する2近似アルゴリズム

- 示したいこと: 近似解の長さ ≦ 最小全域木の長さ×２
- 近似アルゴリズムの流れ
  - 最小全域木の各枝を2本の枝に置き換え，全頂点を通過する閉路をつくる

    ➔ 閉路の長さ＝最小全域木の長さ×２
  - 閉路の無駄な部分をショートカットして，巡回路をつくる

    ➔ 三角不等式より，ショートカットしても閉路の長さは増えない

    ➔ 近似解（巡回路）の長さ ≦ 閉路の長さ

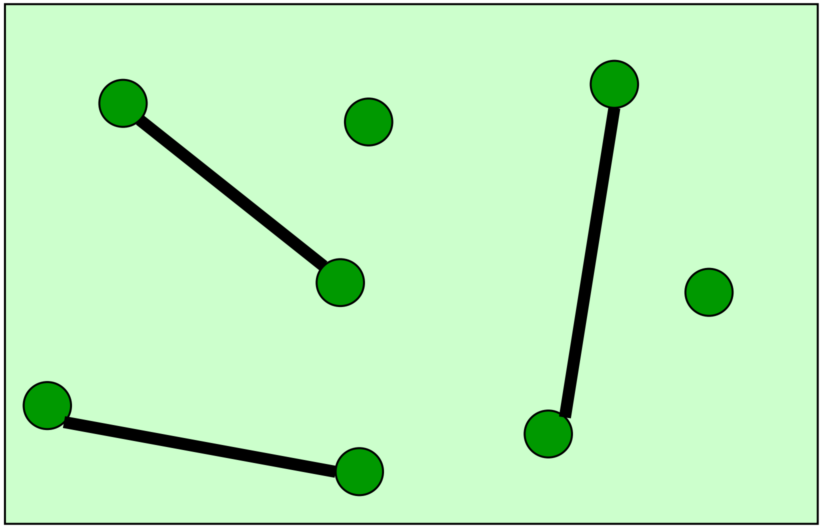
平面上の巡回セールスマン問題に対する1．5近似アルゴリズム

- 最小全域木＋マッチングを使った近似アルゴリズム
- 無向グラフの枝集合Mはマッチング

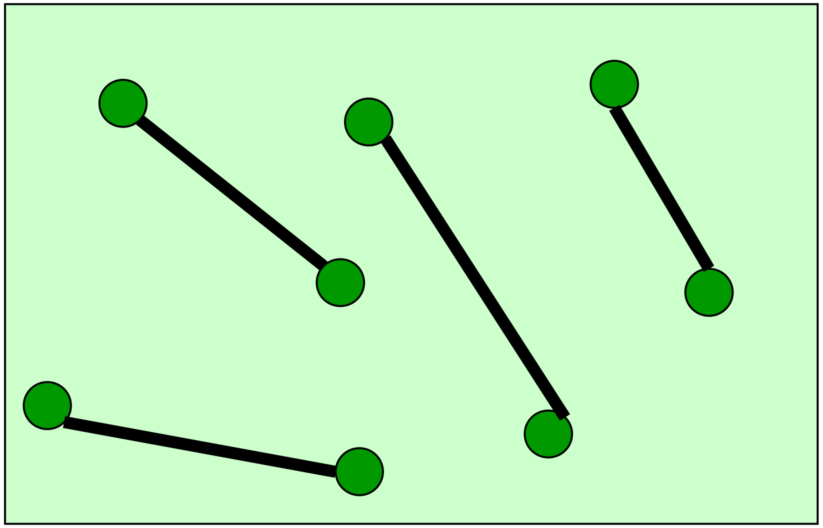
←→ 各頂点に接続する枝は高々1本

- Mは完全マッチング

←→ 各頂点に接続する枝はちょうど一本



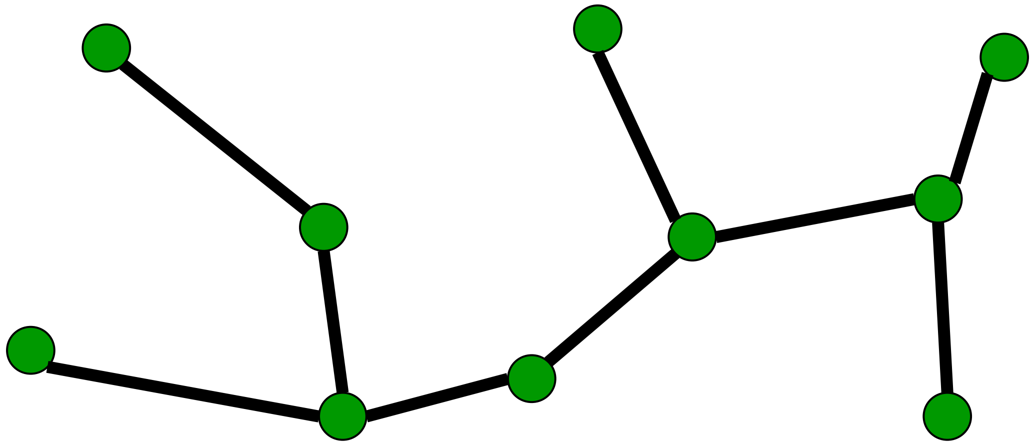
マッチングの例



完全マッチングの例

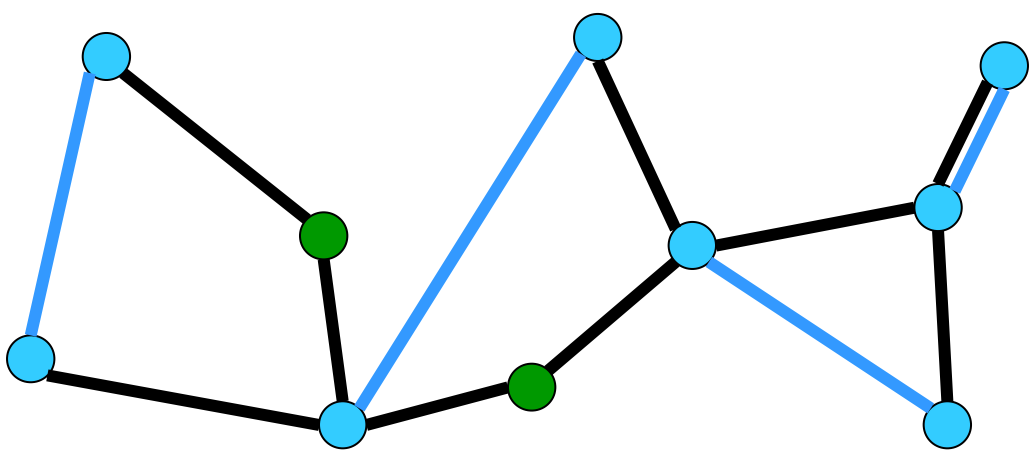
平面上の巡回セールスマン問題に対する1．5近似アルゴリズム

- 最小全域木＋マッチングを使った近似アルゴリズム
  - 平面上の点に対する最小全域木を求める



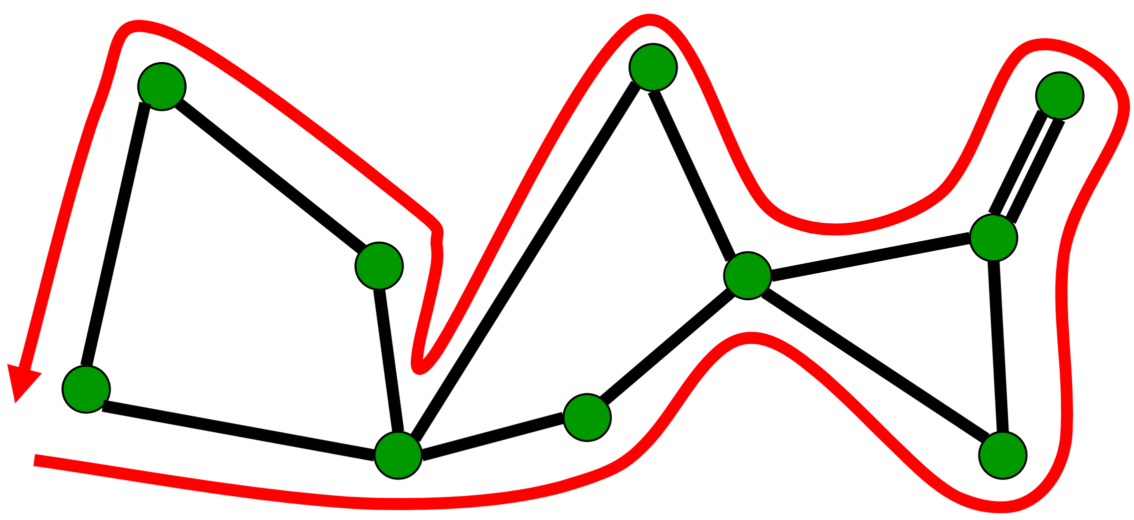
平面上の巡回セールスマン問題に対する1．5近似アルゴリズム

- 最小全域木＋マッチングを使った近似アルゴリズム
  - 平面上の点に対する最小全域木を求める
  - 次数が奇数の頂点の集合に対し，長さ最小のマッチングを求める（注意：次数が奇数の頂点の数は偶数）



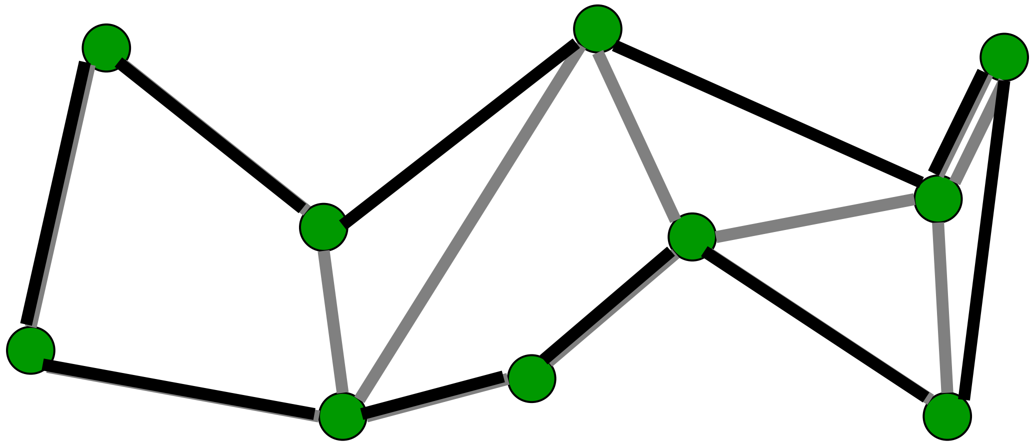
平面上の巡回セールスマン問題に対する1．5近似アルゴリズム

- 最小全域木＋マッチングを使った近似アルゴリズム
  - 平面上の点に対する最小全域木を求める
  - 次数が奇数の頂点の集合に対し，長さ最小のマッチングを求める
  - 最小全域木＋マッチングに対し，各頂点の次数は偶数
    ➔ 全ての辺を通る閉路（オイラー閉路）が存在



平面上の巡回セールスマン問題に対する1．5近似アルゴリズム

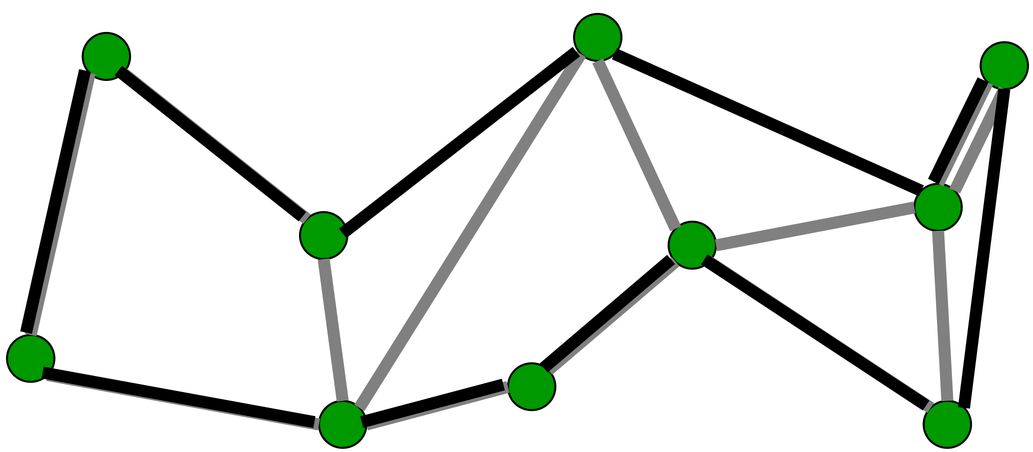
- 平面上の点に対する最小全域木を求める
- 次数が奇数の頂点の集合に対し，長さ最小のマッチングを求める
- 最小全域木＋マッチングに対し，各頂点の次数は偶数
  ➔ 全ての辺を通る閉路（オイラー閉路）が存在
- 閉路の無駄な部分をショートカットして，巡回路をつくる



平面上の巡回セールスマン問題に対する1．5近似アルゴリズム

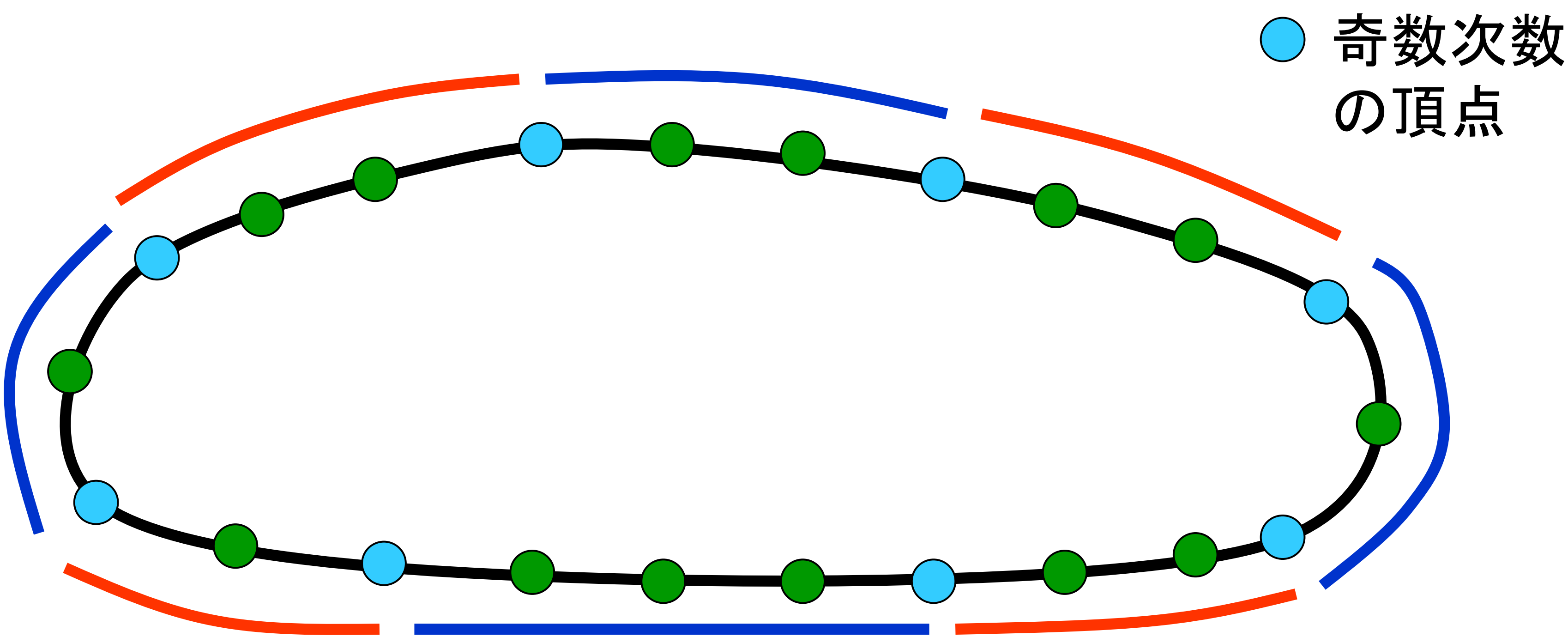
- 近似比の証明：示したいこと
  - 近似解の長さ ≦ 最小全域木の長さ＋最小マッチングの長さ
  - 最適解の長さ ≧ 最小全域木の長さ
  - 最適解の長さ ≧ 最小マッチングの長さ×２

➔ 近似解の長さ ≦ 最適解の長さ×１．５



平面上の巡回セールスマン問題に対する1．5近似アルゴリズム

- 示したいこと: 最適解の長さ ≧ 最小マッチングの長さ×２
  - 奇数次数の頂点を 1, 2, …, k（kは偶数）と仮定
  - 頂点1, 2, …, k を使って，最適解（巡回路）を幾つかのパスに分解（パスの長さの合計＝最適値）



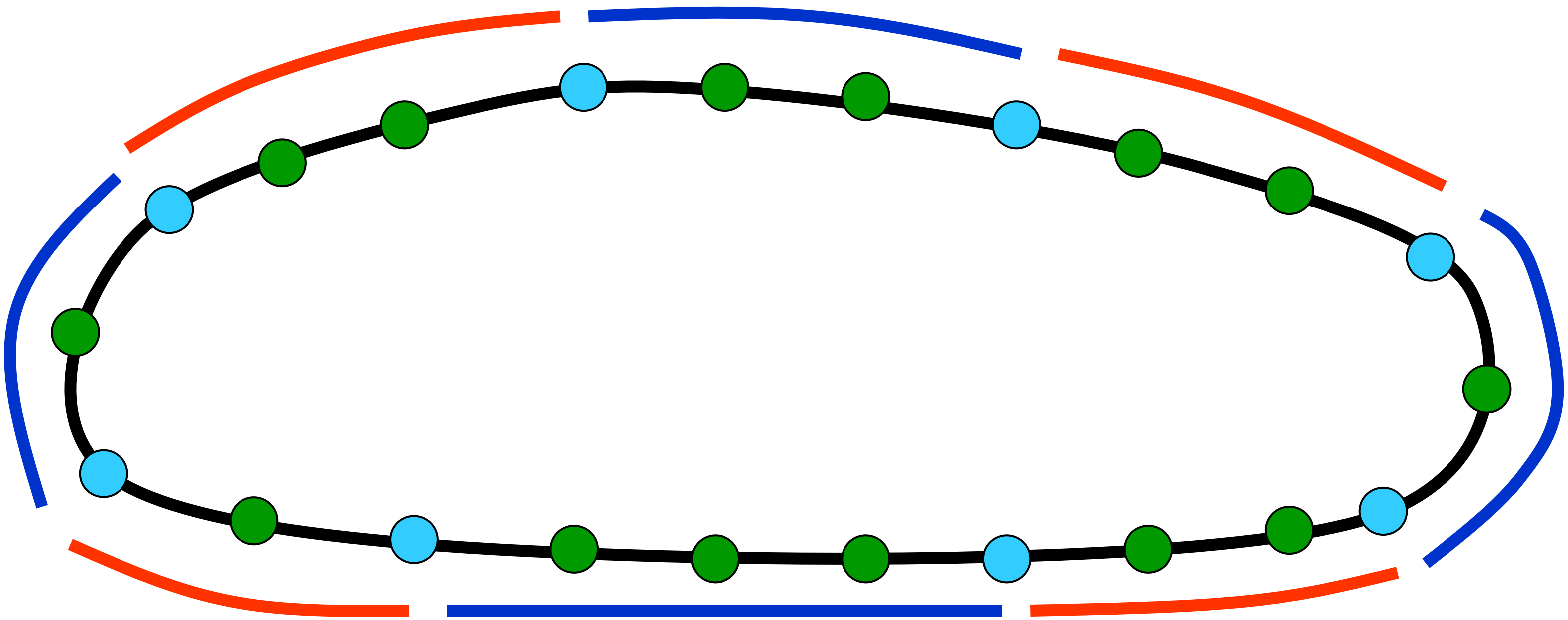
平面上の巡回セールスマン問題に対する1．５近似アルゴリズム

- 奇数番目のパスと偶数番目のパスに分ける
- 示したいこと:

奇数番目のパスの長さの和 ≧ 最小マッチングの長さ

偶数番目のパスの長さの和 ≧ 最小マッチングの長さ

➔ 最適値 ≧ 最小マッチングの長さ×２



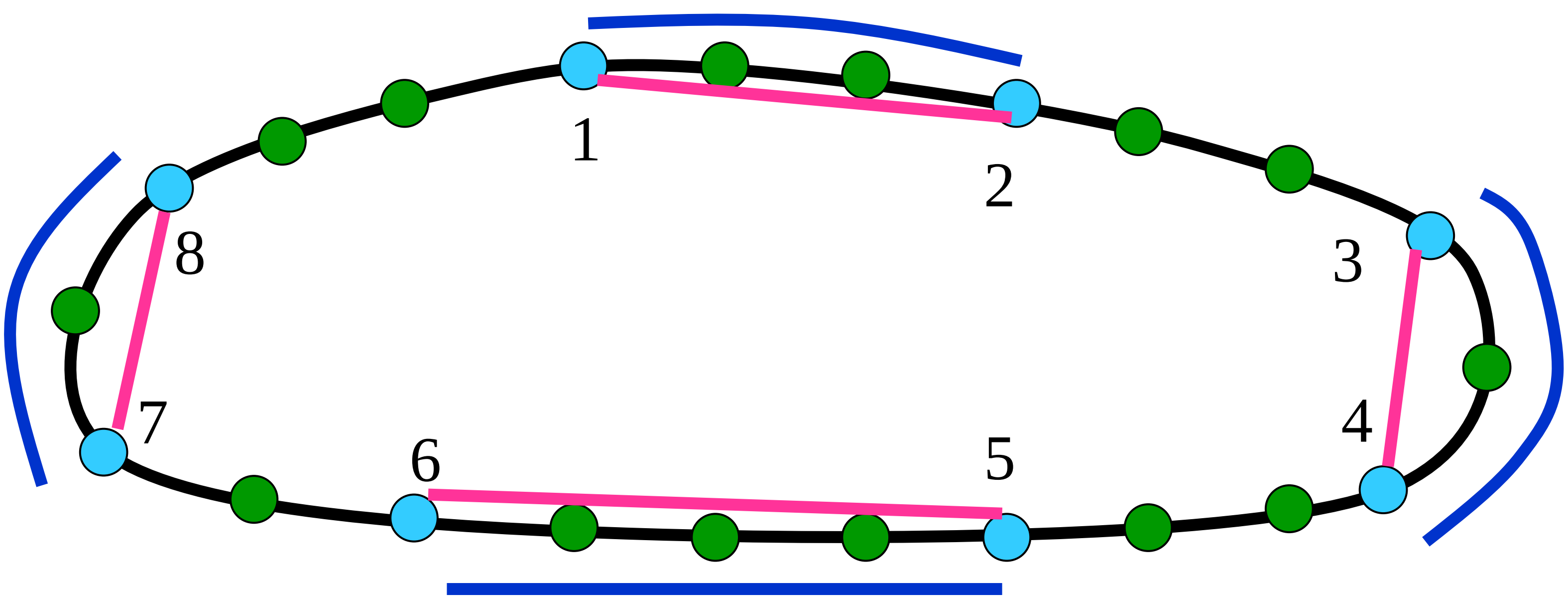
平面上の巡回セールスマン問題に対する1．5近似アルゴリズム

- 示したいこと: 奇数番目のパスの長さの和 ≧ 最小マッチングの長さ
- 三角不等式より,

  頂点 i と i+1を結ぶパスの長さ ≧ 枝（i, i+1）の長さ

➔ 奇数番目のパスの長さの和

≧ 枝 (1, 2), (3, 4), …, (k-1, k) の長さの和



平面上の巡回セールスマン問題に対する1．5近似アルゴリズム

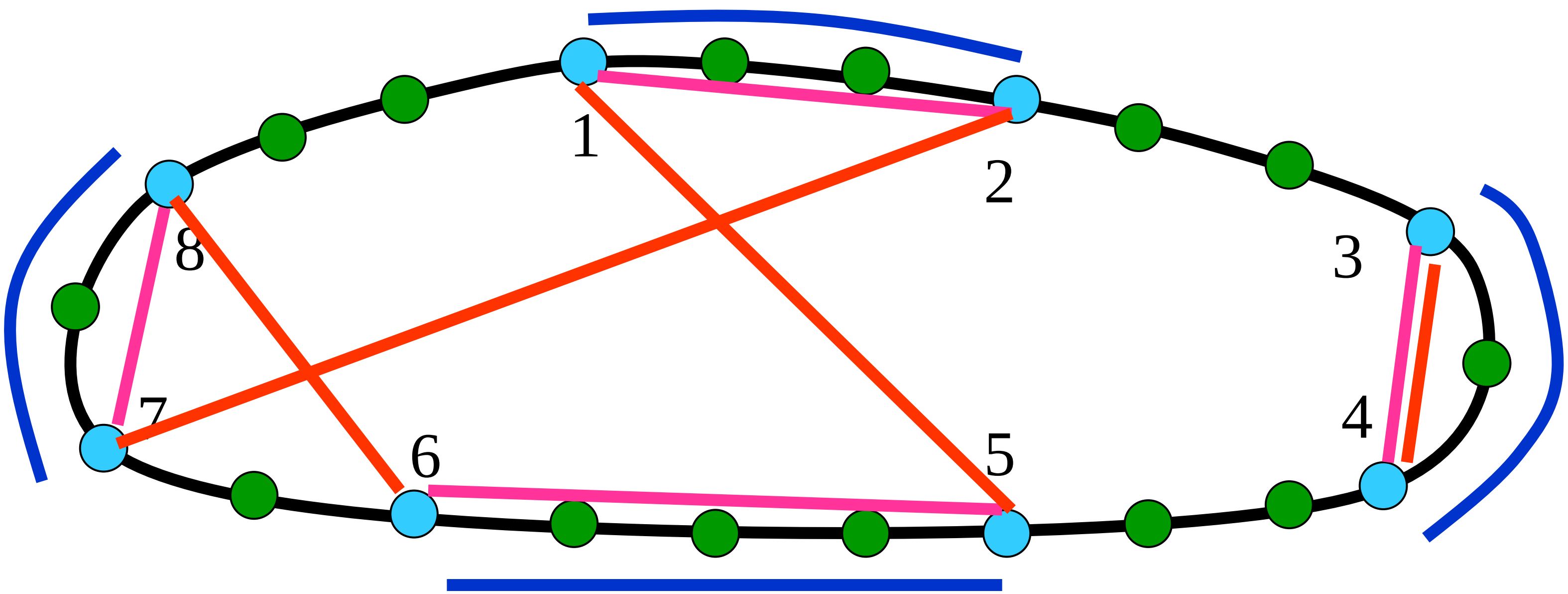
- 示したいこと: 奇数番目のパスの長さの和 ≧ 最小マッチングの長さ

➔ 奇数番目のパスの長さの和

≧ 枝 (1, 2), (3, 4), …, (k-1, k) の長さの和

枝 (1, 2), (3, 4), …, (k-1, k) は奇数次数頂点に対するマッチング

➔枝 (1, 2), (3, 4), …, (k-1, k) の長さの和 ≧ 最小マッチングの長さ

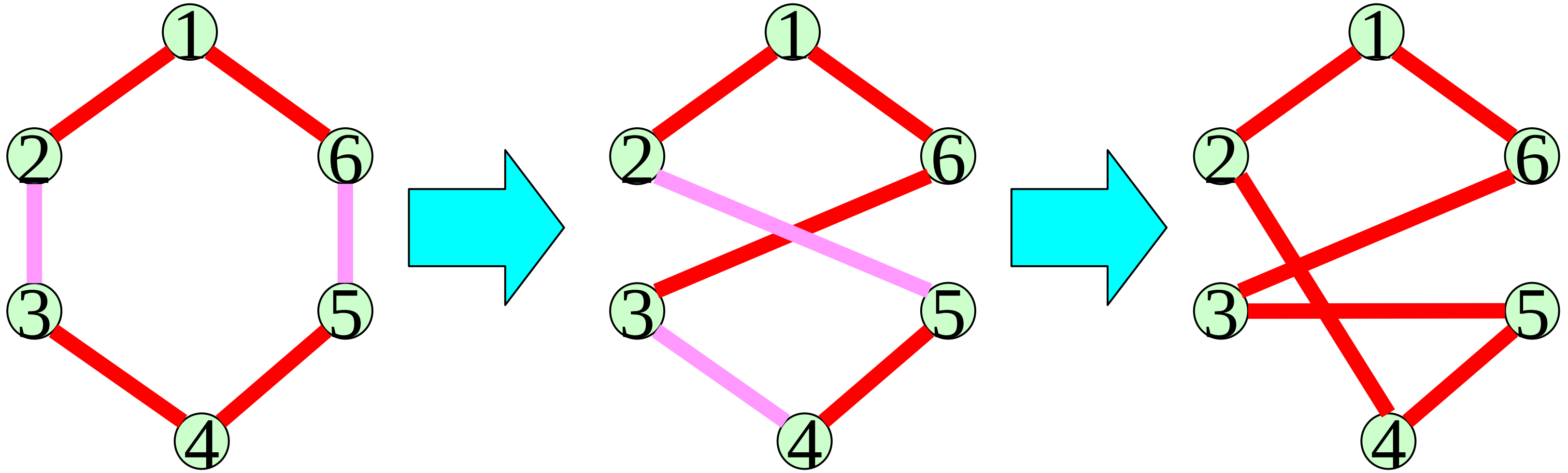


今日の話の流れ

- 近似精度が理論的に保証されたアルゴリズム
  - 0-1ナップサック問題に対する0．5近似
  - 整数ナップサック問題に対する0．5近似
  - 平面上の巡回セールスマン問題に対する2近似
  - 平面上の巡回セールスマン問題に対する1．5近似
- メタヒューリスティックス(metaheuristics)
  - タブー探索法(tabu search)
  - (シミュレーテッド)アニーリング法(simulated annealing)
  - 遺伝アルゴリズム(genetic algorithm)

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

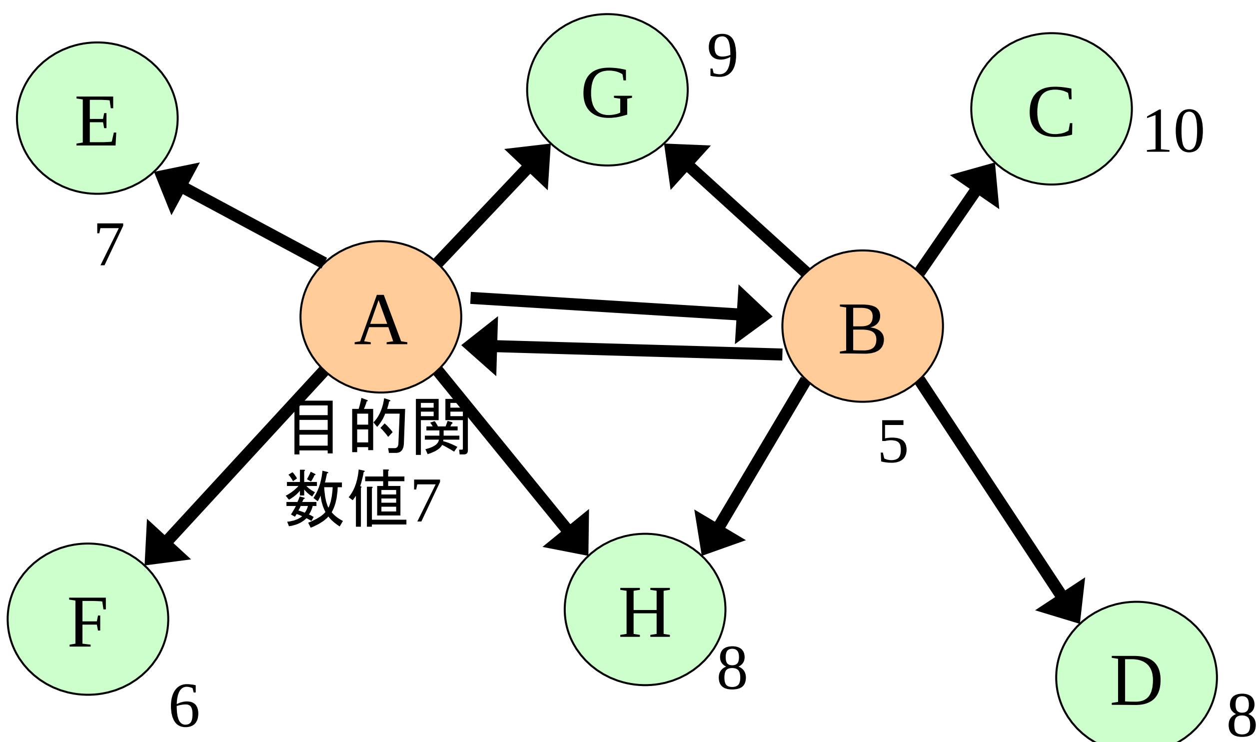


メタヒューリスティックス

- 局所探索の改良版
  - 局所探索では，近傍内に存在する，より良い解に更新することを繰り返す ➔ 局所最適解から抜け出せないことも
  - メタヒューリスティックスでは，より悪い解に更新することを許す ➔ 局所最適解からの脱出
- メタヒューリステックスの例
  - タブー探索
  - アニーリング法
  - 遺伝アルゴリズム
  - アント（蟻）アルゴリズム，などなど
  - 解きたい問題との相性により，各アルゴリズムは得手不得手がある

タブー探索法

- 局所最適解からの脱出方法の一案
  - 常に近傍内で最も良い解に更新する
  - 問題点：おなじ解の間を行ったり来たりする可能性



タブー探索法

- 局所最適解からの脱出方法の一案
  - 常に近傍内で最も良い解に行く
  - 問題点：おなじ解の間を行ったり来たりする可能性
  - 回避案１：過去に訪問した解には二度と行かない
    ➔ 膨大な記憶容量と計算時間が必要
  - 回避案２：最近訪れた解の幾つかを記憶（タブーリスト），それらの解には行かない

タブー探索法

- タブーリストについて
  - 最近訪れた解をｔ個記憶する
  - ｔの値は大きすぎても小さすぎても良くない．予備実験等によりチューニングする必要有り
- 終了条件について
  - 例１: 反復回数が一定数に達する
  - 例２: 一定の反復回数の間，解の改善が得られない

タブー探索法の大まかな手順

１．タブーリストを初期化

２．初期解Sを求める

３．終了条件が成り立つまで，下記の手順を繰り返す

３－１：Sの近傍に入っていて，かつタブーリストに含まれていない解のうち，最も良いもの S' を求める

３－２：現在の解を S' に更新．タブーリストを更新

４．これまで求めた解のうち，一番良いものを出力

アニーリング法

- 物理現象の「焼きなまし(annealing)」からアイディアを得た方法
  - 温度が高い➔原子は激しく動く➔安定な状態に移りやすい➔温度を下げる➔安定な状態が得られる
- アニーリング法は「温度」というパラメータをもつ
  - 温度が高い➔より悪い解への移動が起こりやすい
    ➔大域的最適解に到達しやすくなる
  - 徐々に温度を下げる➔良い解に徐々に収束する

アニーリング法の大まかな手順

1．初期解Sを求める
2．初期温度T，温度の減少率 r（0＜r＜1）を決める
3．温度が十分小さくなるまで，下記の手順を繰り返す
　3－1：以下の手順を L 回繰り返す
　　3－1： Sの近傍内の解S' をランダムに選ぶ
　　3－2：SとS' の目的関数値の差をΔとおく
　　3－3：S' がSより良い解ならば，SをS' に更新
　　3－4：S' がSより悪い解ならば，確率 $e^{-\Delta/T}$でSをS' に更新
　3－2：温度 T を rTに下げる
4．これまで求めた解のうち，最良のものを出力

遺伝アルゴリズム

- 遺伝子の進化からアイディアを得た方法
  - 遺伝子(染色体)の交叉や突然変異によって新しい世代が形成される
  - 弱いものが淘汰され，強いものが生き残る
- 遺伝アルゴリズムは複数の(許容とは限らない)解を複数個もっていて(集団と呼ぶ)，これらを繰り返し更新

集団に対する基本操作

- 評価
  - 与えられた解の良さを評価する
  - 目的関数値が大きいほどよい（最大化の場合）
  - 許容解に近いほどよい
- 両親の選択
  - 評価値をふまえて解のペア（両親）を選ぶ
  - 評価値の良いものほど親に選ばれやすい

集団に対する基本操作

- 交叉
  - 両親である解をうまく組み合わせて，幾つかの新しい解をつくる
- 整数計画問題に対する交叉の例
  - 両親$(x_1, x_2, \ldots, x_n)$, $(y_1, y_2, \ldots, y_n)$
  - １点交叉 ➔ $(x_1, \ldots, x_k, y_{k+1}, \ldots, y_n)$, $(y_1, \ldots, y_k, x_{k+1}, \ldots, x_n)$
  - ２点交叉 ➔ $(x_1, \ldots, x_k, y_{k+1}, \ldots, y_h, x_{h+1}, \ldots, x_n)$,
    $(y_1, \ldots, y_k, x_{k+1}, \ldots, x_h, y_{h+1}, \ldots, y_n)$
  - 一様交叉➔各成分を $\{x_j, y_j\}$ からランダムに選択
  - 交叉により無意味な解が出てこないように，交叉の方法を検討する必要がある

集団に対する基本操作

- 突然変異
  - 交叉により得られた解をランダムに修正する
- 淘汰
  - 解の評価値に基づき，現在の集団の中の解を一定数以下に減らす

- 以上の操作を適当な順番で繰り返す
- 終了条件が成立したら終了，これまでの最良解を出力