生物配列解析アルゴリズム

渋谷

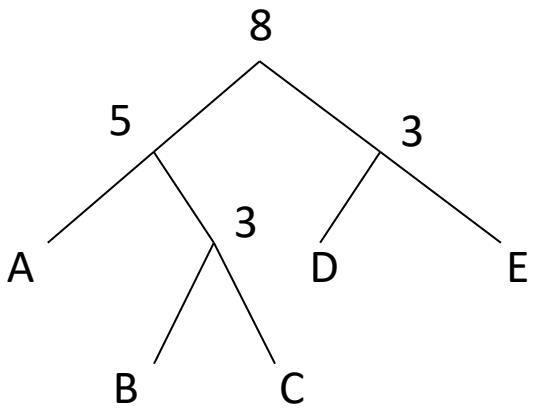
東京大学医科学研究所ヒトゲノム解析センター
（兼）情報理工学系研究科コンピュータ科学専攻
http://www.hgc.jp/~tshibuya

今回の話題

- 系統樹（続き）
- ゲノム・アセンブリ

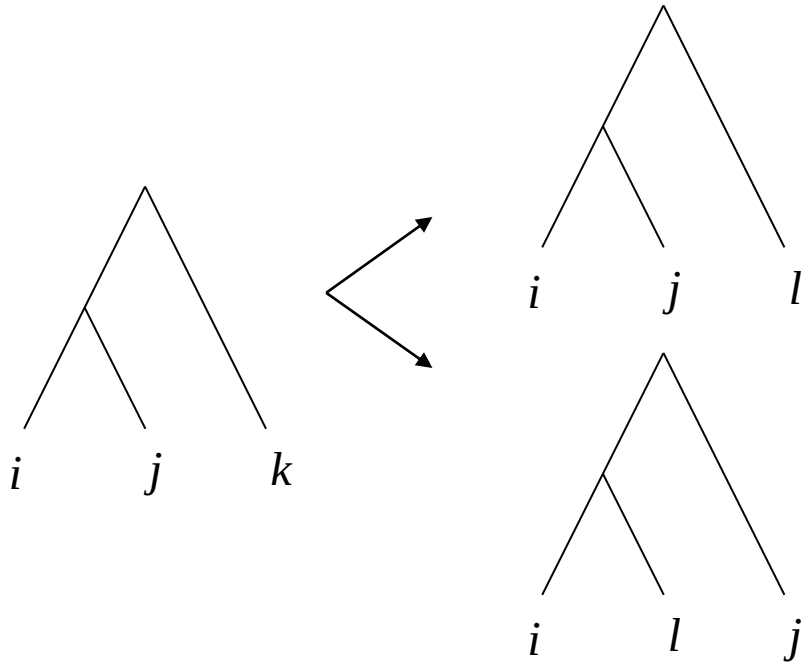
Ultrametric Tree (1)

- Ultrametric tree/matrix
  - 葉の間の距離を次のように定義
    - internal nodeのラベルは数字（下に行くほど小さい・正）
    - 葉間の距離：　$D[i,j] = \mathrm{label}(\mathrm{LCA}(i,j))$
  - *cf.* min-ultrametric tree/matrix
    - ラベルが下に行くほど大きい
- 理想的な系統樹はultrametric tree
  - 生物学的には、進化速度が一定である、という条件



Ultrametric Tree (2)

- 行列Dがultrametricである必要十分条件
  - 対称行列（対角は0、それ以外は正）
  - すべての3つ組i,j,kに対して
    - Max $(D[i, j], D[j, k], D[k, i])$ をとる組は2つ
- 行列→木は一意



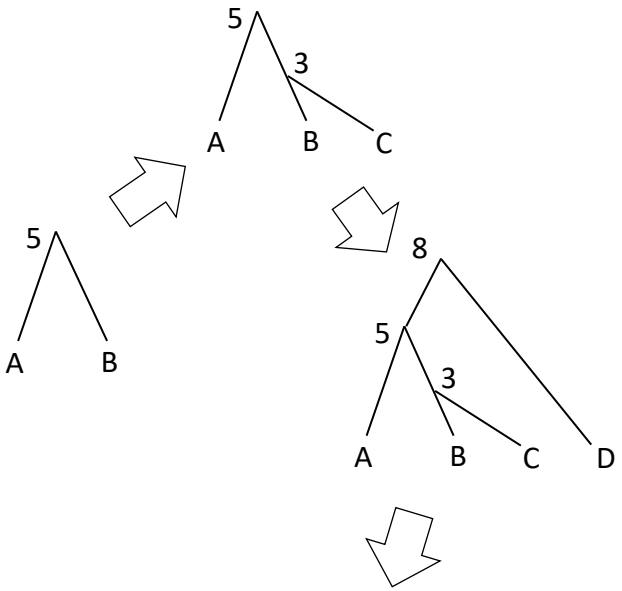
i,l,kの関係によって木の形が**一意に**定まるが、それはj,l,kの関係と矛盾しない

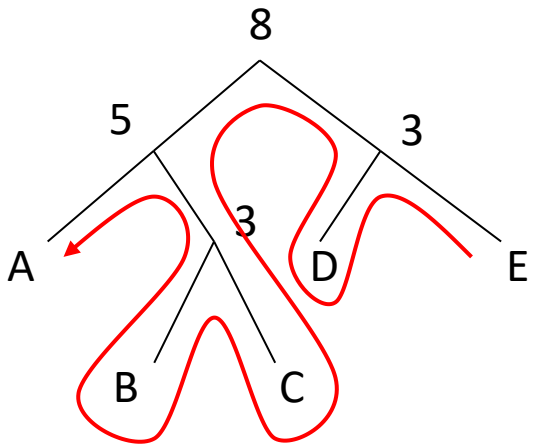
木構造は**一意**であるがl,k,とiまたはjとの関係は定義から矛盾しない

再帰的に証明する

Ultrametric Tree (3)

- 木の構成アルゴリズム
  - 単に一つずつ加えていけばよい
    - 木の左から順にチェックしていく
  - 計算時間は$O(n)$
    - nは葉数：行列サイズに関して劣線形時間アルゴリズム（行列の全値を見ない）
- 判定は$O(n^2)$
  - とにかく木を作成してみる
    - 新しい葉を作成できなければだめ
  - それぞれの葉に関して自分より左にある葉との距離をチェックする
    - 巡回するだけでチェックできるので、LCA等の難しいアルゴリズムは不要





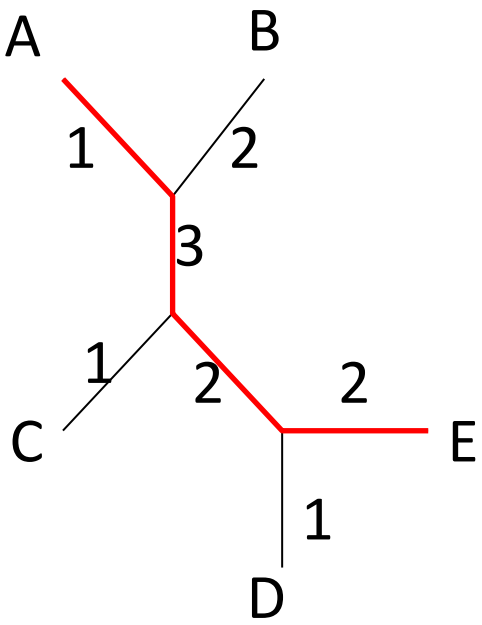
Eの正当性チェック

Ultrametric Tree (4)

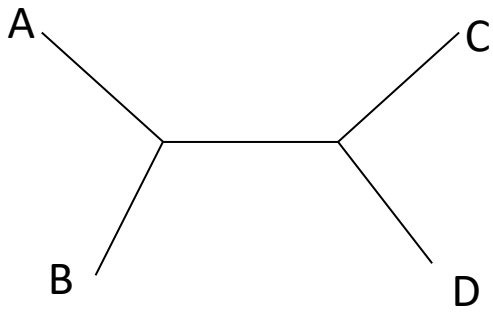
- データの集め方
  - タンパク質・DNA等の配列の変異率から推測する
  - 配列がわからない場合にはDNAの二重鎖を1本にして、どれほど結合しやすいか（結合温度で）計測する、なんていう実験的手法も
- それがultrametric でない場合は？
  - あきらめて別の方法を使う
  - 値を少しいじってultrametricにする
- Ultrametric行列の簡単な（強引な）作り方
  - Dを距離行列だと思ったグラフを考える
  - Minimum Spanning Treeを作成する
    - $O(m+n \log n)$ $(=O(n^2))$ by Prim's algorithm
  - $D'[i,j]$ = path i–j上の最大長の枝長、とする
  - このとき、
    - D'はultrametric
    - $D' \leq D$ （全ての要素が小さくなっている）
    - $D'' \leq D$となっているすべてのultrametric 行列D''に対して$D'' \leq D'$

Additive Tree (1)

- 木上のパスに相当する距離行列⇔木
  - 枝長は正とする
  - 内部ノードも使ってよい
- 性質
  - ultrametric distance は additive
    - 逆は成り立たない
  - ルート（根）がどこかは不明
- 生物学的には
  - 進化速度が異なることを意味している



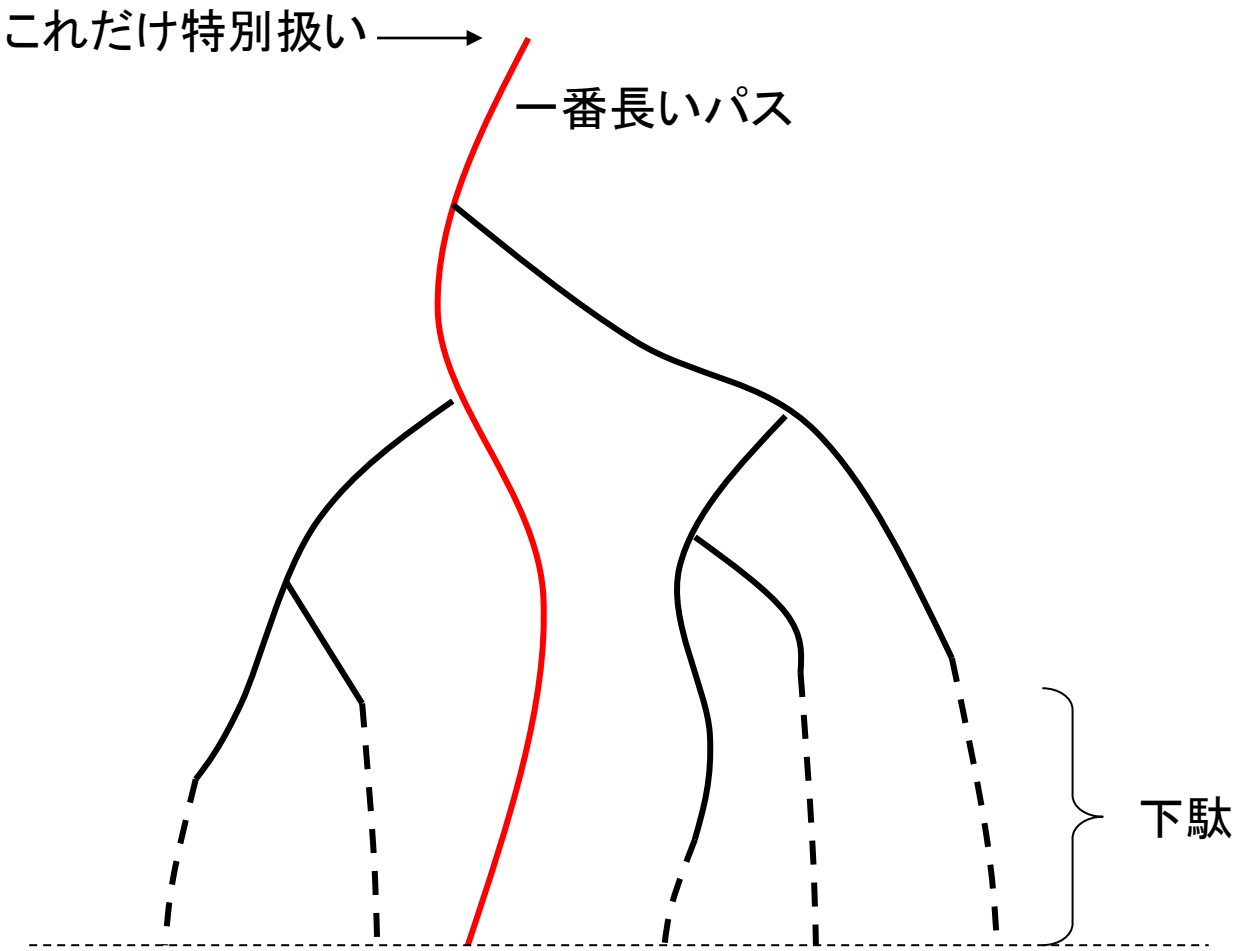
Additive Tree (2)



- 必要十分条件
  - 対称行列
    - 対角線は0、それ以外は正
  - for all i, j, k, l
    - $D[i,j]+D[k,l] \leq \max\,(D[i,k]+D[i,l], D[i,l]+D[j,k])$
    - *i.e.*, これらの3つの値のうち、2つ（以上）が同じ最大値をとる
  - 再帰的に証明
- 内部ノードもすべて距離行列の対象になっている場合のアルゴリズムは簡単
  - すなわちadditive treeがn点から成っている場合
  - Minimum Spanning Treeを求めるだけ
    - $O(m + n \log n)$ $(= O(n^2))$ by Prim's Algorithm

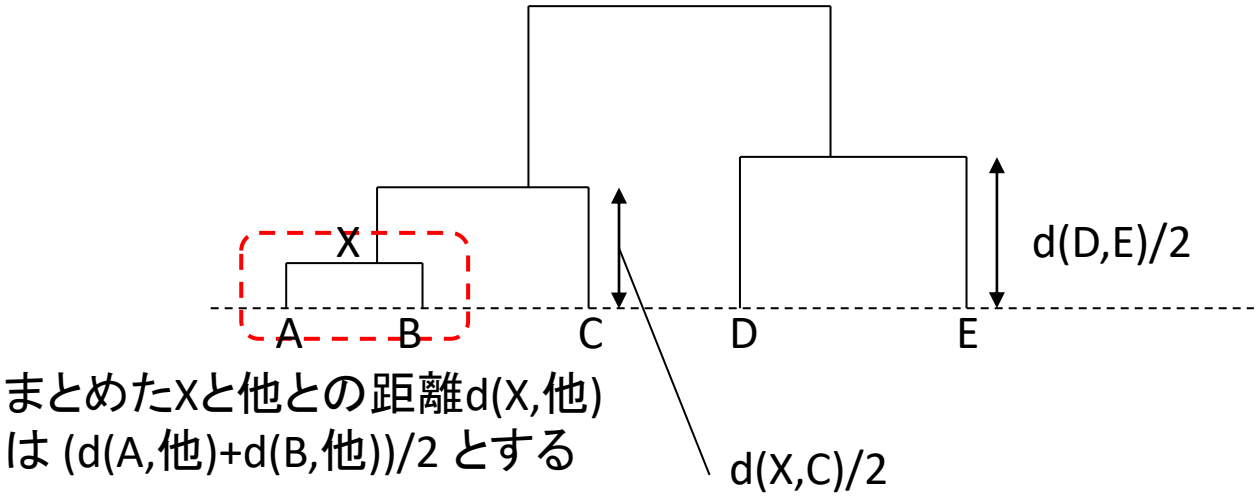
Additive Trees ← Ultrametric Trees

- うまく下駄を履かせるだけでultrametric treesの問題に帰着できる



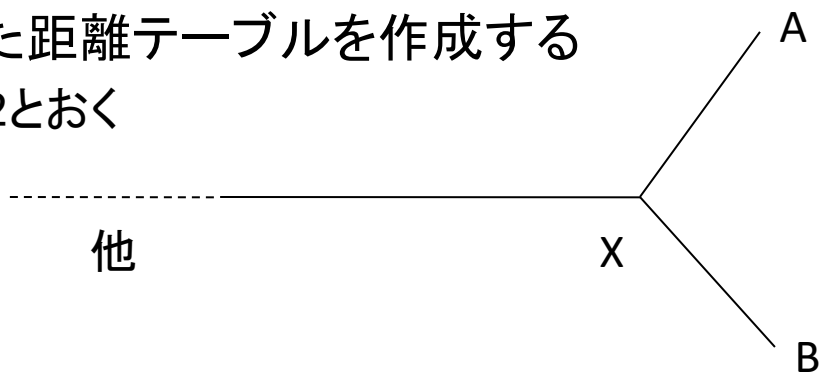
UPGMA (Unweighted Pair-Group Method using Arithmetic averages)

- 簡単なヒューリスティック [Sokal & Michener '58]
  - 近いものから順番にまとめる（ボトムアップに作成）
  - ultrametricな行列であれば、正確に構成することができる
    - そうでないものでも動く（何か出力することはする）



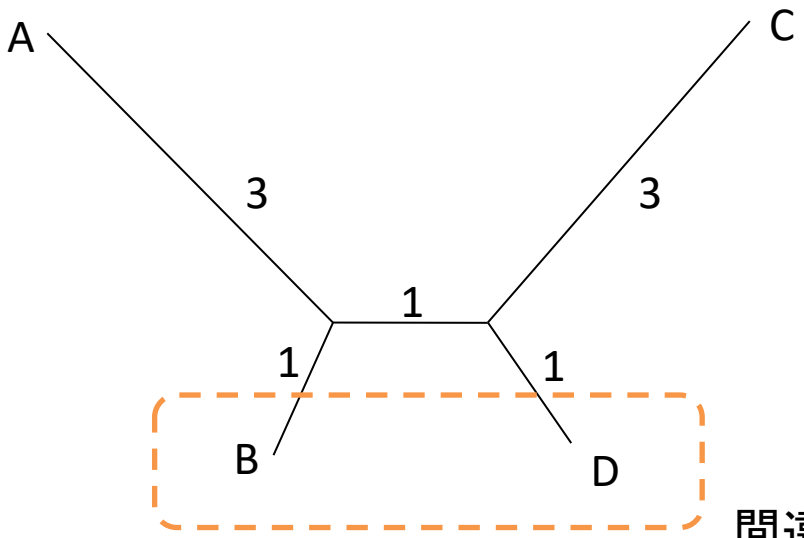
Fitch and Margoliash '68

- UPGMAはあくまでultrametric treeに対するアルゴリズム
  - 通常の行列に対する計算法としてはあまりよくない
- Fitch and Margoliash ’68
  - 近いものをまとめる （ここはUPGMAと同じ）
    - 近い組み合わせA,Bに対して親Xを作る
  - XとA,Bの距離を以下のように定義する
    - d(A, X)= (a+x−b)/2
    - d(B, X)= (b+x−a)/2
      - a: AからBを除く他のノードへの平均距離
      - b: BからAを除く他のノードへの平均距離
      - x: AB間の距離
  - A,Bを取り除き、代わりにXをいれた距離テーブルを作成する
    - Xと他の点の間の距離は、(a+b)/2とおく



しかし、UPGMAやFitch-Margoliashでは、、

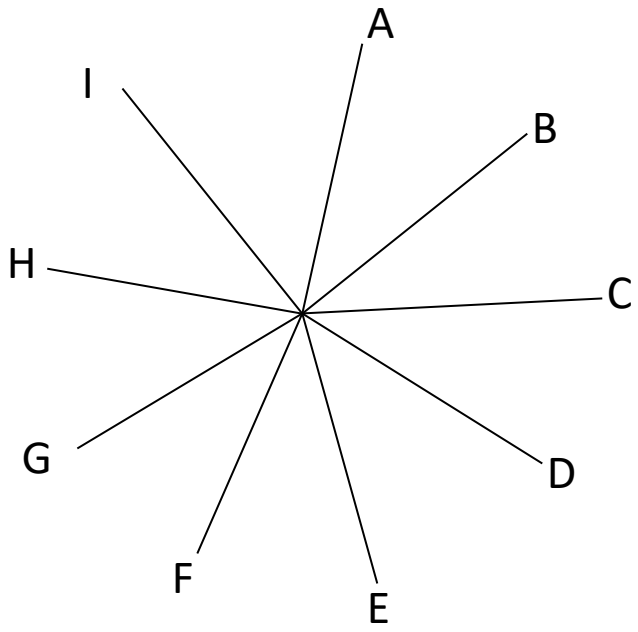
- additive treeから作成された行列を、正確に木に再構成できるとは限らない
  - 何も考えずに、近いペアをまとめてしまうため
- ただし、先にも述べたとおり、理想的なadditive tree相手ならば下駄をはかせたUPGMAで解ける。
- しかし、additive treeでない一般行列（ultrametricにしようにも下駄がよくわからない）に対して強引に適用しても、（経験的に）あまりよい系統樹にならないことがある



間違えて纏めてしまう

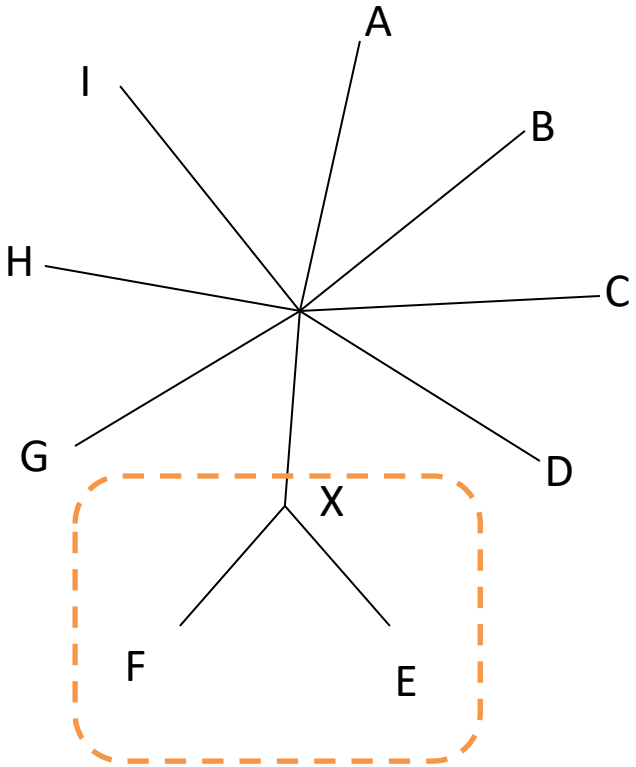
Neighbor Joining (Saitou, Nei '87) (1)

- 同じく「近い」ものから順番にまとめるが、、
  - additiveな行列を、正確に再構成することが可能
  - そうでない距離行列に対しても「うまく」動く（実験的検証）
- まず星型の木を考える
  - 特定の点から「他のすべての点を代表する点」への距離をルートからの枝長とする
    - $c_i = \sum_j d(i,j) / (n-1)$
      - $d(i, i) = 0$ に注意
    - たとえば、Fにとっては、d(F, A), d(F, B),...,d(F,I)の平均的距離に相当
    - 木の枝長の総和は　$\sum_{i<j} d(i,j) / (n-1)$



Neighbor Joining (Saitou, Nei '87)　(2)

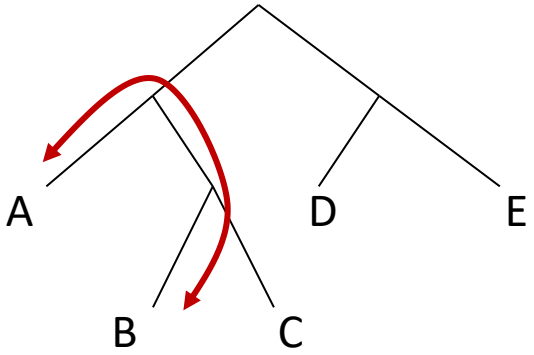
- 例えば、E,Fの親ノードとして新たなノードXを入れた場合、木の枝長の総和は以下の3つの値の和となっているべきと考えられる
  - $\sum_{i\in\{E,F\},\, j\neq E,F} d(i,j)/2(n-2)$
  - $d(E,F)/2$
  - $\sum_{i,j\neq E,F} d(i,j)/(n-2)$
- この総枝長が最小となる2つ組を探す
  - ${}_nC_2$通り
  - こうして選ばれた組は、必ずadditive treeな系統樹上であれば隣り合ったノードであることが証明できる
- その2つ組を削除し、そのかわりにその親を入れた星型木に対して同じことを繰り返す
  - ここで、2点E,Fの親Xを作った場合、XからAへの距離は
    - $(d(E,A)+d(F,A)-d(E,F))/2$
- 総計算時間は$O(n^3)$　[Studier & Keppler ’88]



まとめてXを作ることを考える

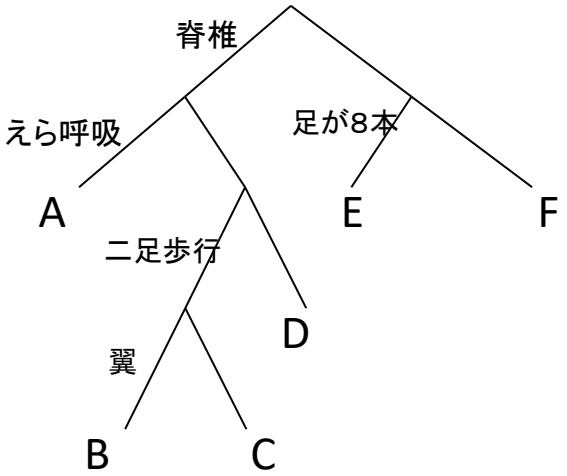
より一般的な行列に対する理想的な定式化

- 誤差の最小化
  - Given: $d(i,j)$
  - Output: The tree that minimizes $\sum (d(i,j)-D(i,j))^2$
    - $D(i,j)$: 作成した木上での距離
  - しかしこれは、NP困難
    - というわけで、結局よく用いられるのは Neighbor Joining



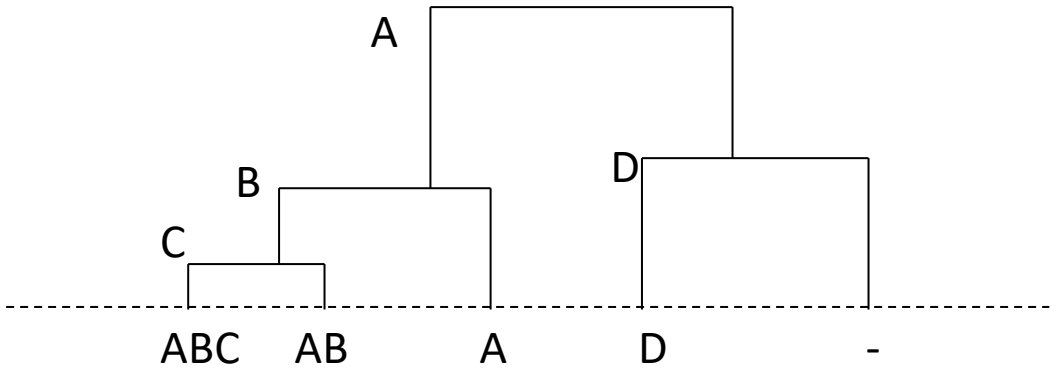
Character-based Perfect Phylogeny

- 様々な特徴から系統樹を作成するには？
  - 脊椎を持っている・翼を持っている・etc
  - DNA・たんぱく質が特定の部分配列・特徴的パタンなどを持つ
  - DNA（たんぱく質）のある位置の塩基がAである
    - SNPs（一塩基多型）、haplotype（多型＝SNPsの組み合わせ）
- 理想的な特徴
  - その特徴があるのはある枝の子孫のみ
    - 適応集中のような状況は考えない
    - 配列の塩基の話であれば、それが1度変わってまた戻ることはないものとする
  - 一部の枝に（uniqueに）特徴が割り当てられる
- そのような特徴にあった系統樹を作成するには？



Character-based Perfect Phylogeny ← Ultrametric Tree

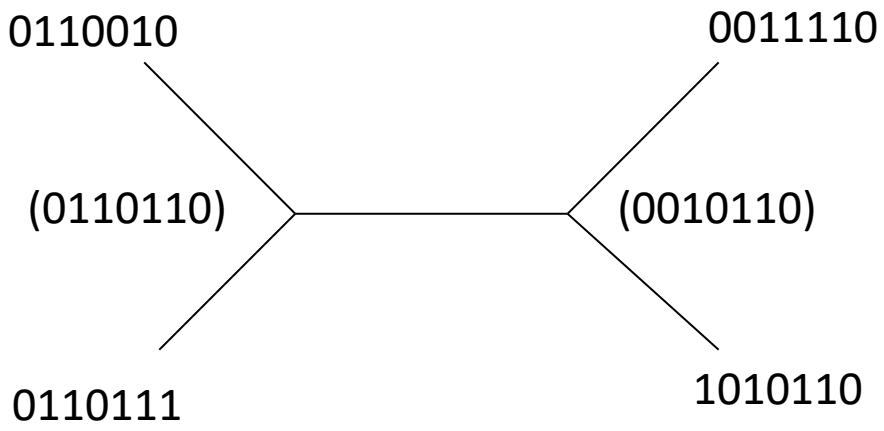
- $D[i,j] = i, j$ に共通な特徴の数
  - とするとmin-ultrametric tree となる
  - 距離を（特徴数）ー（共通な特徴な数）とすればultrametric tree
  - 特徴に重み付けがあってもよい



ただし、「ある特徴をもっていない」のも共通な特徴数に数える

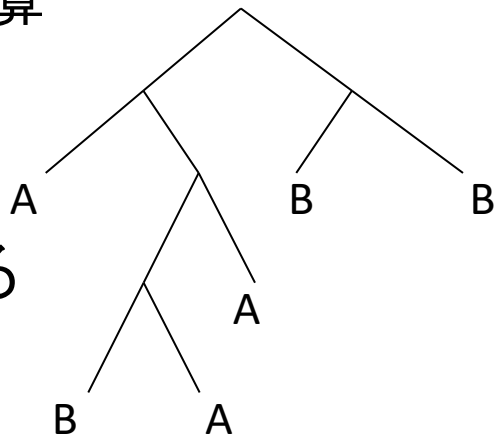
Maximum Parsimony

- PerfectではないPhylogenyの問題
  - サイズs^dの（拡張）ハイパーキューブにおける Minimum Steiner Treeを解く問題
    - d: 文字数、s: アルファベットサイズ
- 難しさは？
  - NP-hard
  - 当然ながら、（ギャップを考えて）アラインメントするのと同時に系統樹も作成、といったのはさらに難しい



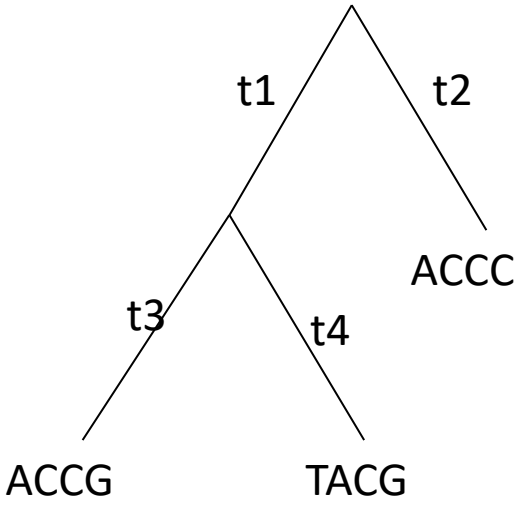
やさしいサブ問題： Fitch-Hartigan '71

- 系統樹の形と葉のラベル（１文字）が与えられていた場合に内部ノードにラベル付けをする問題
- 仮定
  - 葉に一文字だけのラベルが与えられている
  - できるだけ枝の両端点は同じ文字になるようにする
- アルゴリズム
  - 葉から順番にDP
    - そのノードのラベルがxであった時の最高点を計算
    - $O(s^2 n)$（s: アルファベットサイズ）
- 応用
  - 変異確率をより正確に計算することができる
    - PAM等のスコア表のより正確な計算に有用



類似品：Felsenstein '81

- 与えた系統樹が与えた配列データを出力する確率を求める
  - 入力：木の形と各枝長（進化時間）、葉の配列データ
  - 内部ラベル
    - わからないので、EMを用いて予測
  - 枝長を推定するための変異モデル
    - Jukes-Cantor Model
      - 単位進化時間あたり、別の塩基に変異する確率を α
    - Kimura Model
      - A-G,C-T間は α
      - それ以外（4通り）は β
- Felsenstein '81（最尤法）
  - その確率の最も高い木を系統樹として出力
  - しかし、すべての木を列挙するのはかなり大変
    - $(2n-3)!!$通りもある（!!は!の一個飛ばし）
  - なんらかのヒューリスティック
    - 分割統治・GA・グリーディー等々

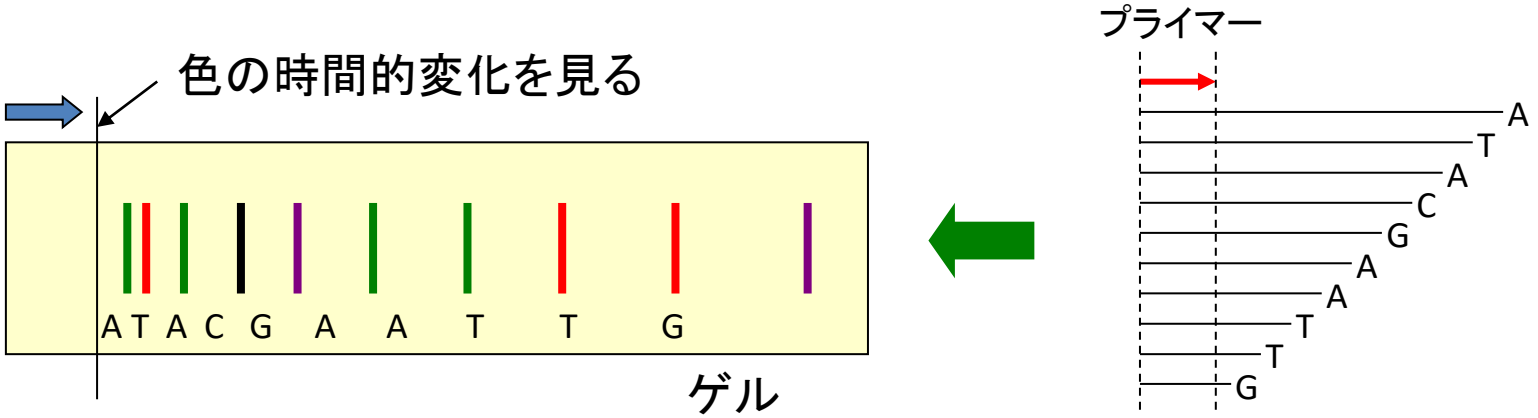


作成した系統樹の評価

- 定性的な評価は難しい
- Bootstrappingによる推定
  - 系統樹の各分岐に対し変異率を与えると、系統樹から（ランダムな）標本を生成できる
  - 生成した標本から系統樹を作成する
  - これを繰り返して、同じ系統樹を得る確率が高ければその系統樹の確度（安定性）は高い、という評価、とする評価法

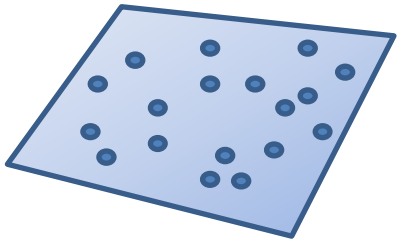
DNA Sequencer (昔)

- Sanger Sequencing
  - 高精度に（比較的）長く読む目的では現在でも現役
  - それでも読めるのはせいぜい1000塩基程度
- やりかた
  - primerから先の様々な長さの部分配列を作成
  - 終端がA,T,C,Gのどれであるかを判定できるようなシグナル付き塩基に終端を置換
  - 電気泳動をする
    - 分子量によってゲル上を動く速度が異なる

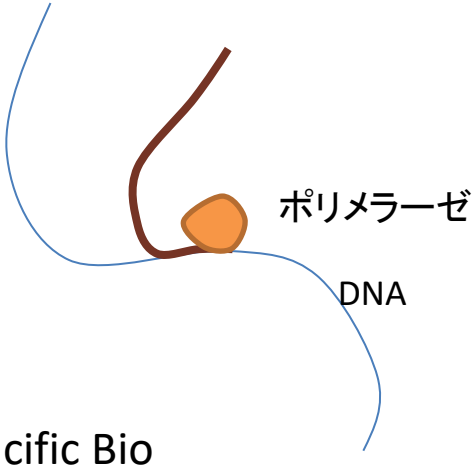


DNA Sequencer (数年前～数年後)

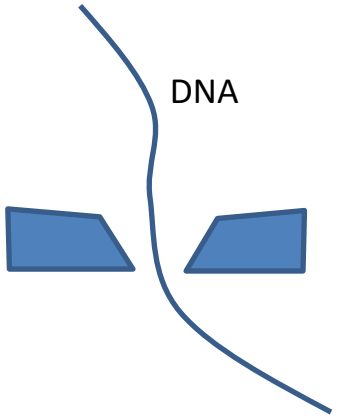
- 革命的な進展
  - 様々な技術が開発され、とてつもない速度で解読されている
    - 次世代シーケンサーと総称される
  - 人一人のゲノムが10万円以下、数時間で読める時代
    - ただ、やはり（まだ）長いDNAは読めないものが多い



Illumina Genome Analyzer
プレート上で、大量のDNA断片を並列でコピー・伸長させ、それを1塩基伸長させるごとに観察する



Pacific Bio
大量のDNA断片に対し、実際のコピーをミクロレベルでリアルタイムに観察する

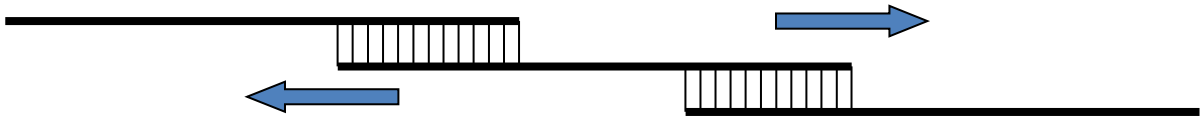


IonProton
半導体テクノロジーの利用

より長い配列の解読はどうすればよいか？

- やりかた
  - 解読できるサイズに分割
  - それらをつなげていく
- つなげ方
  - Shotgun sequencing & DNA Assembly
    - 大量のランダムに切って作成した小さなDNA配列（リード）を読む
    - それらをつなげていく（リピート・エラーに注意）
  - Walking
    - それでもわからない解読領域を前後に延ばしていく
  - Physical mapping
    - 解読する前に大きな配列の順番を決定する方法
    - 最近はあまり使われない（コストがかかるため）

Walking / Assembly

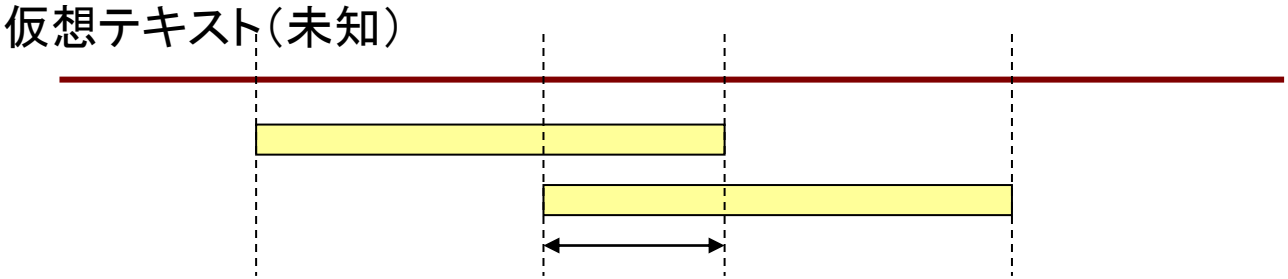


古典的なDNA Assemblyの解法

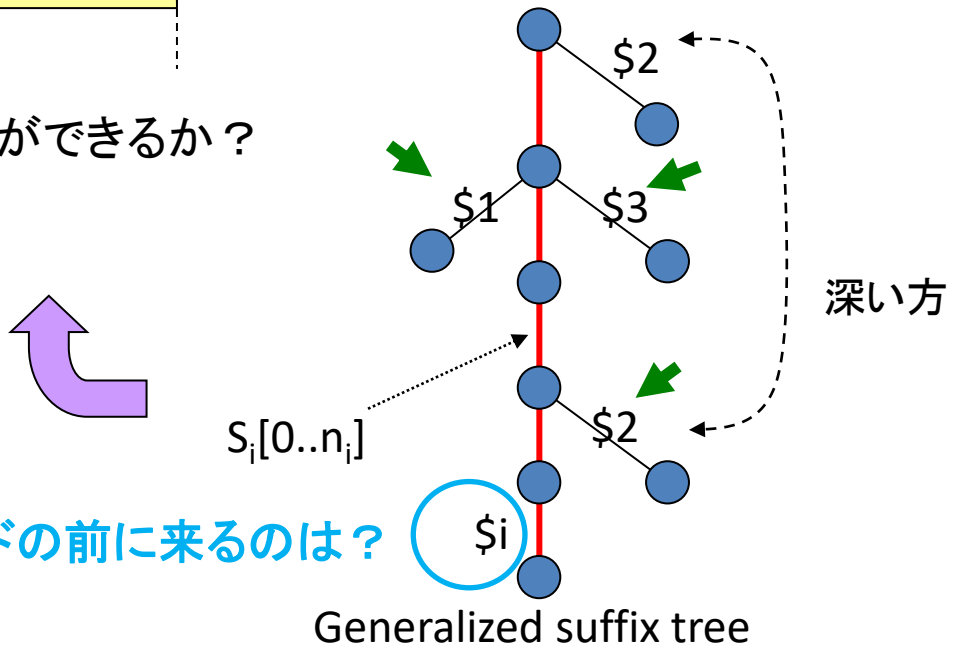
- 一般的に使われる解法
  - すべての組み合わせに関してどれくらいoverlapがあるかをチェックする
  - その結果から適当に順番を決め、全体を構成する
  - 基本的にはヒューリスティック
- 注意する点
  - リピートがたくさんあるかもしれない
  - エラーがあるかもしれない
  - 逆方向の配列も存在する
  - コンタミネーションがあるかもしれない

All-pairs Suffix-Prefix Matching (復習)

- ある2つ（以上）の文字列がどれくらい前後でオーバーラップしているかを知るのは一般化接尾辞木を用いて線形時間

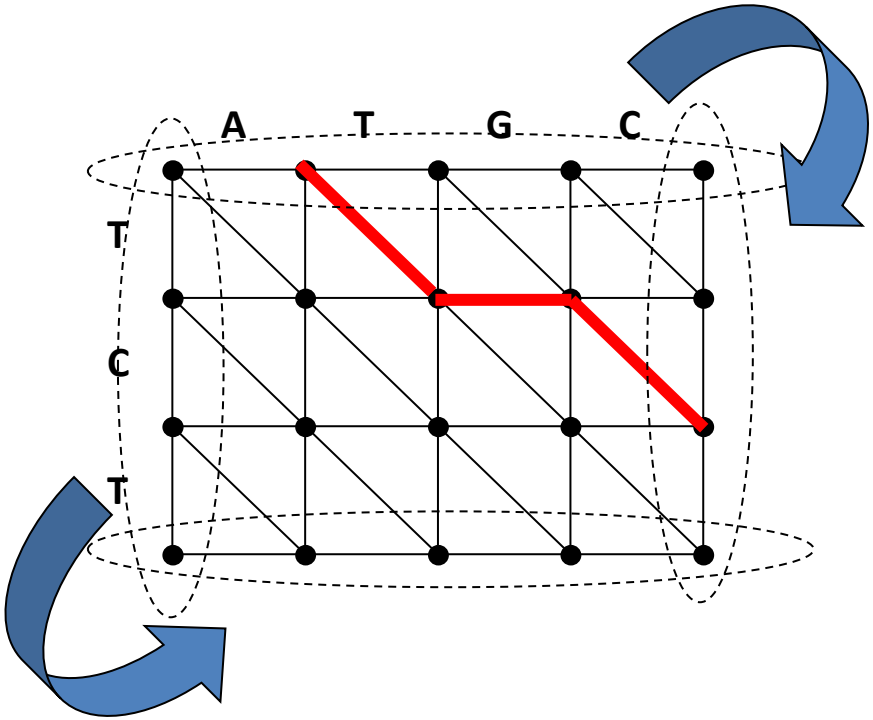


どれほど長く一致させることができるか？



というようなのは実際にはやはり使えなくて、、、

- 普通のDPを使う
  - 適当なBLAST的ハッシュでDPを行う場合を限定することも多い
  - external gapの扱いに注意
  - 全対全でやるため、計算時間は膨大



簡単なつなげ方＆全体の構成

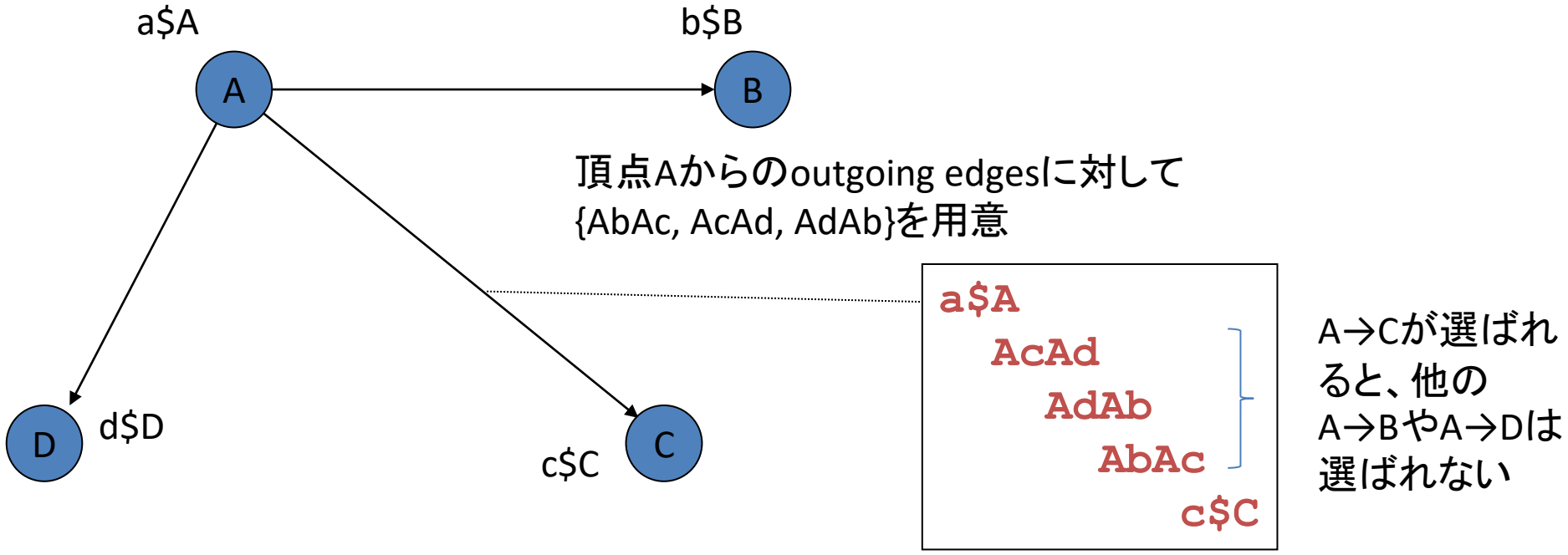
- グリーディーにつなげる
  - つなげた時のマッチスコアが高いものから順につなげていく
    - リピートなどはないと仮定
      - cf. Superstring問題, SBH（後述）
    - 十分なコピー数のシークエンシングを行ってからマルチプル・アラインメントをするのが理想だが、高コスト
- 構成
  - 理想的なデータでは特に留意点なし
  - 一致しない場所
    - 単に多数決
    - すべて出力
    - 連続して曖昧な箇所はマルチプル・アラインメントが要

Shortest Superstring Problem (SSP)

- 問題
  - 入力:　k 本の文字列
  - 出力:　それらを部分文字列として含む最小の文字列
- 難しい
  - 判定問題がNP−complete
    - ハミルトニアンパス問題からの還元で証明
- 理想的なエラーの発生のないようなShotgun sequencingの解析に相当する
  - 実際には
    - エラーは多数
    - 最小のものかどうかは不明
    - 逆方向の配列も存在
    - リードより長りリピートも多数存在

ハミルトニアンパスからSSPへの還元

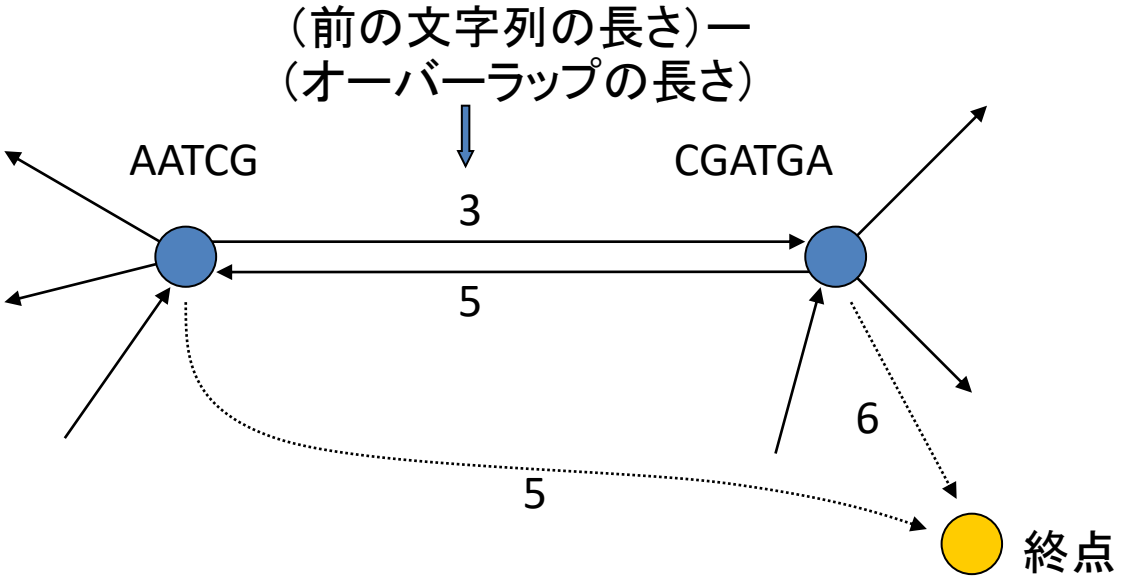
- ハミルトニアンパス（HP）問題
  - 枝長 1 で、長さ n（＝点数）のTSPの有無
  - NP完全
- HPの辺を文字列でうまく表現してSSPソルバーで解く（還元）
  - $3n+2m+1$以下のsuperstringの有無



このSSPをとけば、ハミルトニアンパスが解けてしまう

SSP→有向TSP

- よって、SSPの判定問題がNPに属することは明らかなので、SSPはNP完全
- ヒューリスティック解法
  - SSP→有向TSP（すなわち有向TSPでSSPを解く）も可能で、それを利用して、TSPのヒューリスティックを使うことが可能
  - 他に定数OPTの解法もあり（Cycle cover問題を利用）



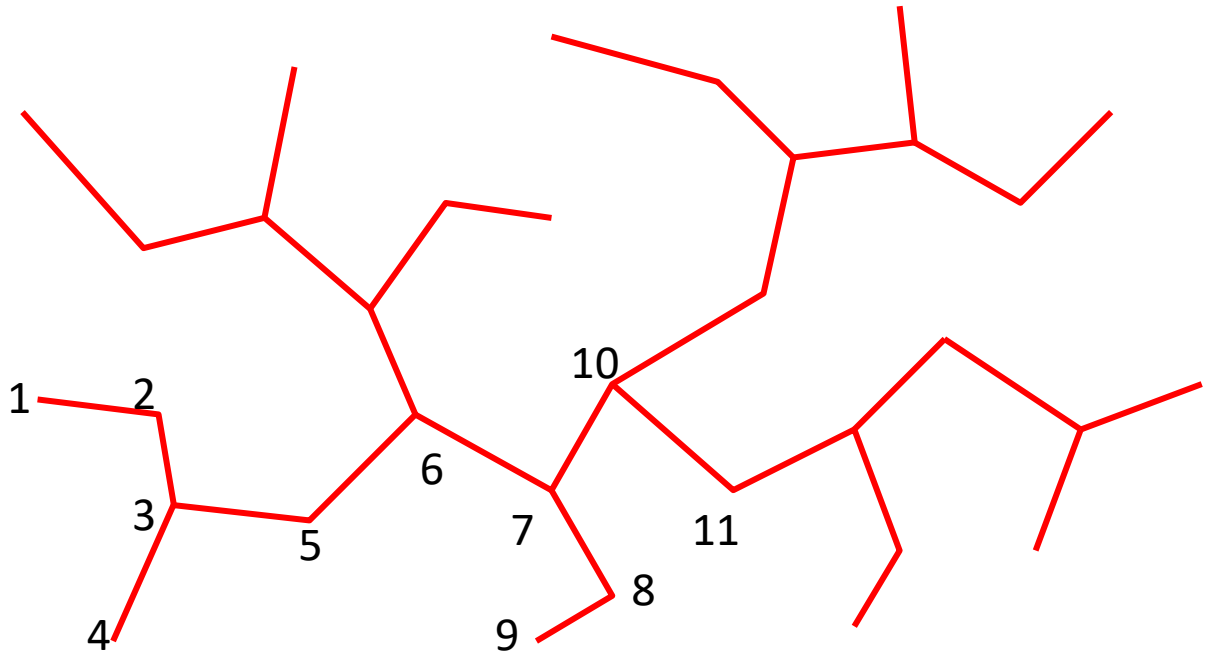
このようなグラフを
Overlap graph とよぶ

巡回商人問題 (TSP)

- すべての点を巡って戻ってくる最短の閉路を求める問題
  - 判定問題（ある値よりよい解があるかどうかを判定する問題）はNP完全（＝難しい、ということ）
  - 無向グラフでは最適解の定数倍以内が保証されるようなアルゴリズムが存在(metric distance)
  - Euclidean TSPに対してはPTASが存在 [Arora '96]
  - simulated annealing などのヒューリスティックも有効

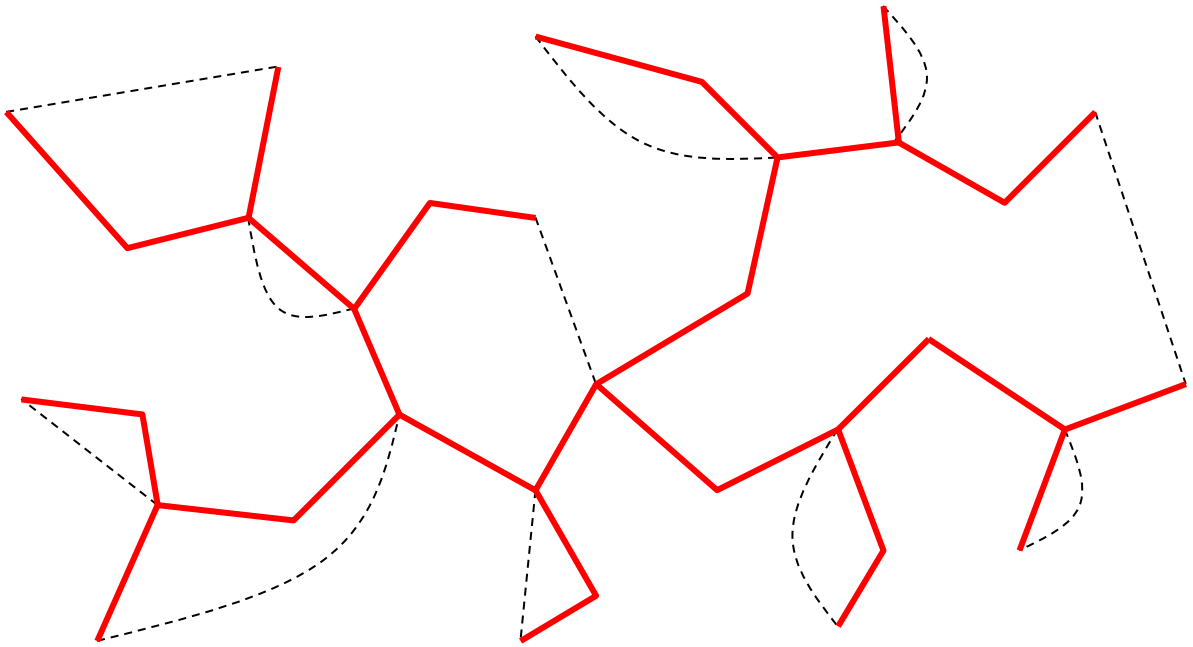
（無向）TSPの2倍近似多項式時間アルゴリズム

- Minimum Spanning Treeを作る
  - 例えばPrimのアルゴリズムで、$O(m + n \log n)$
- 適当にその上をDFS順に辿るだけで2倍近似を達成
  - MSTの長さはTSPより短い
  - MSTの長さの2倍よりは短い閉路を作成することが可能



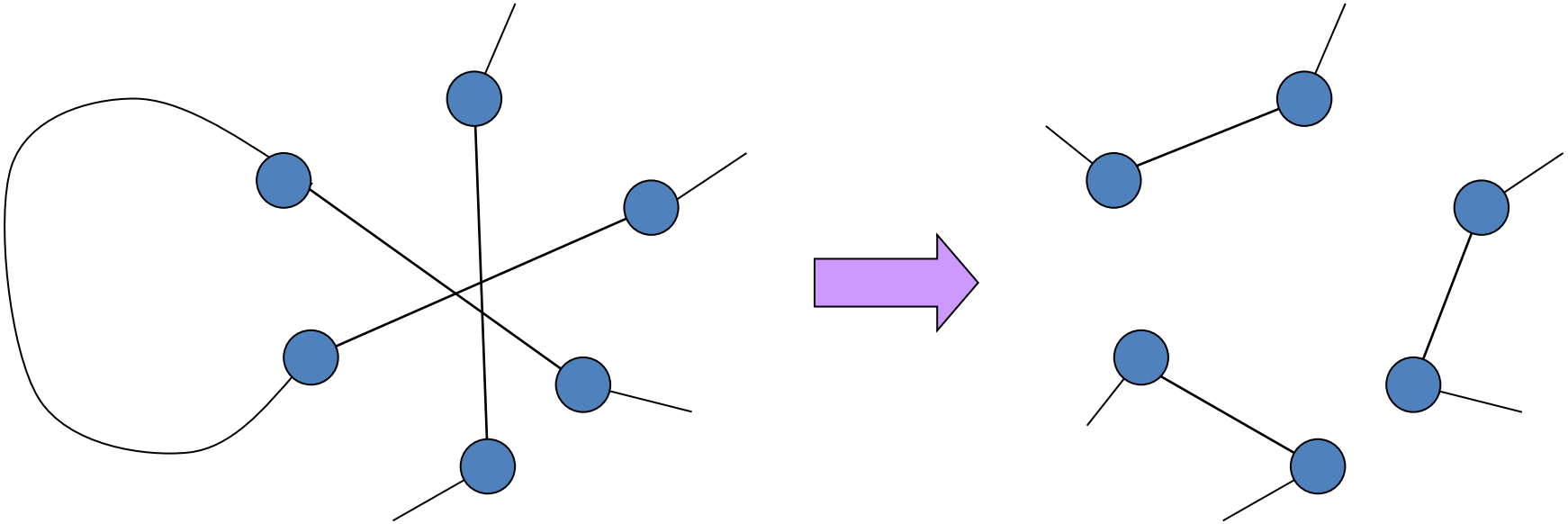
（無向）TSPの1.5倍近似多項式時間アルゴリズム [Christofides '76]

- MSTに加えて次数が奇数の点の最小重みマッチングの枝を加える
  - $O(n(m+n \log n))$ [Gabow '90] （少し難しい）
- この上でオイラー閉路の順番で巡回すればよい！
- すると1.5倍近似におさまる（これはtight）



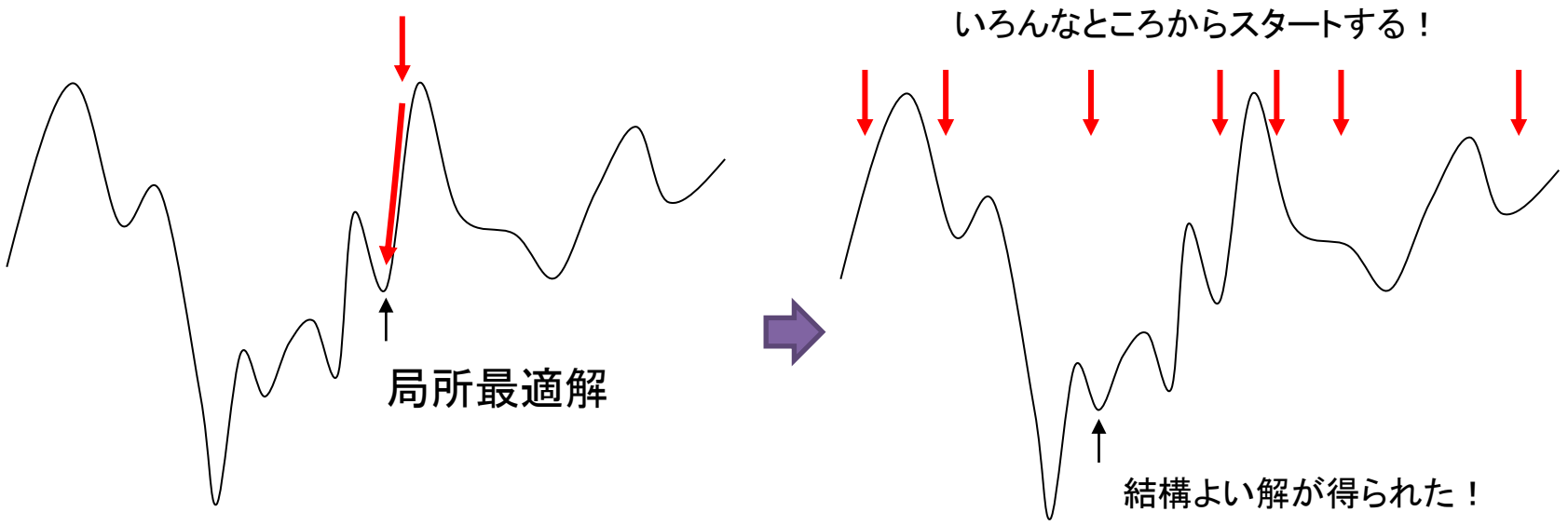
代表的なTSPのヒューリスティック

- k-opt最適化
  - 適当にk本の枝をとってきて、適当にその端点を入れ替えることで解がよりよくなるならば交換する
    - ただし、閉路が2つに分かれてしまうような交換は除外
  - そのような枝の組がなくなるまで繰り返す
  - 通常k=2,3程度のみを用いる（kが大だとオーバーヘッドが大）
  - 二種類の戦略： First 改善・Best改善



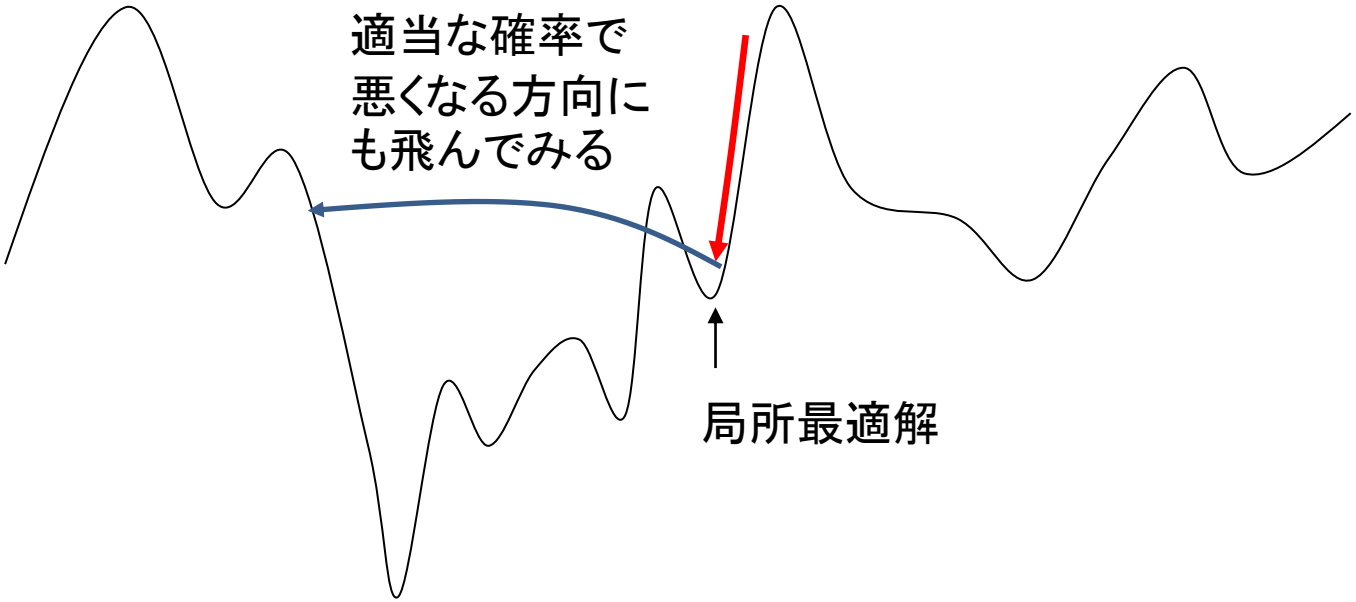
Multiple Start

- k−optは局所最適解に陥り易い
- 最も頭を使わない簡単な対処法：
  - マルチプルスタート



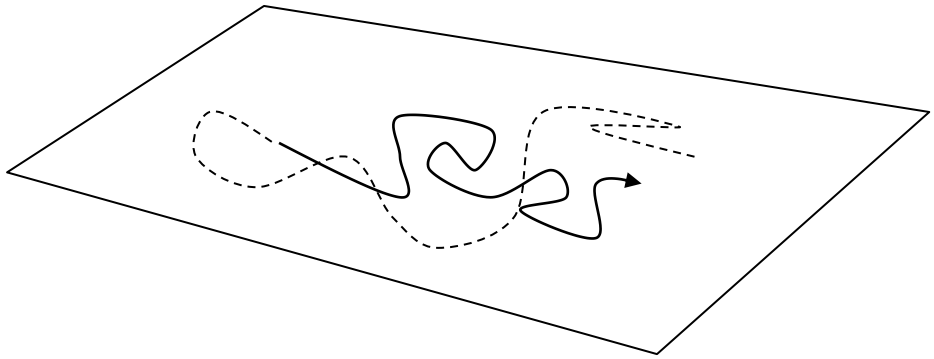
Simulated Annealing

- 適当な確率で、たとえ解が悪くなっても端点の交換を許す
  - 悪い度合いによって移る確率も変えることが多い
- その確率はだんだん下げていく
  - 分子運動の「熱なまし」的に



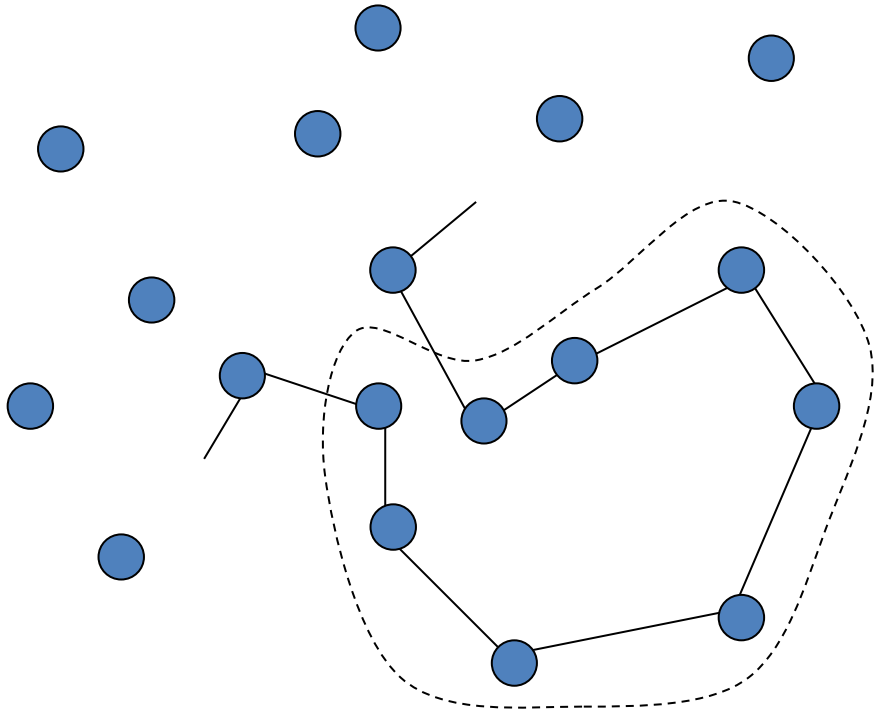
Tabu Search

- 過去の挙動を適当な長さだけ覚えておき、少なくともその間に探索したものは禁止する
  - 「解そのもの」ではなく、解の特徴を禁止することで非常にうまく計算できる場合がある。
  - TSP系の問題の場合は例えば2-optの「ある枝の交換」をしばらく禁止する、など。



Genetic Algorithm

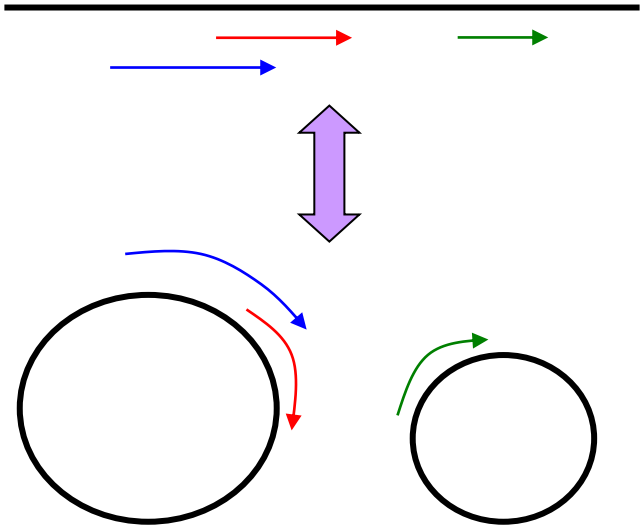
- 複数の解から、組み合わせて新しい解を作成する
  - k−optの拡張と見ることもできる
  - 部分解のとり方は様々

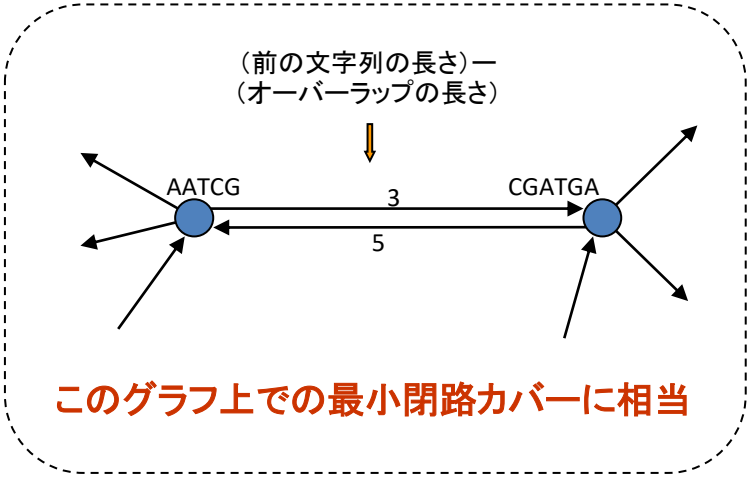


部分解を別の解の一部と交換して
新しい解を作成する

Cycle Cover vs SSP

- Cycle cover problem
  - 与えられた文字列を含むCyclic stringの集合
    - bcabcabcabcはcyclic string「abc」に含まれるとする
  - そのうち、長さの和が最小のものを求める問題（SCCP）
- すると明らかにSSPの解≥SCCPの解
  - SSPの解から最小かどうかわからない同じ長さのCCPの例を作成できる

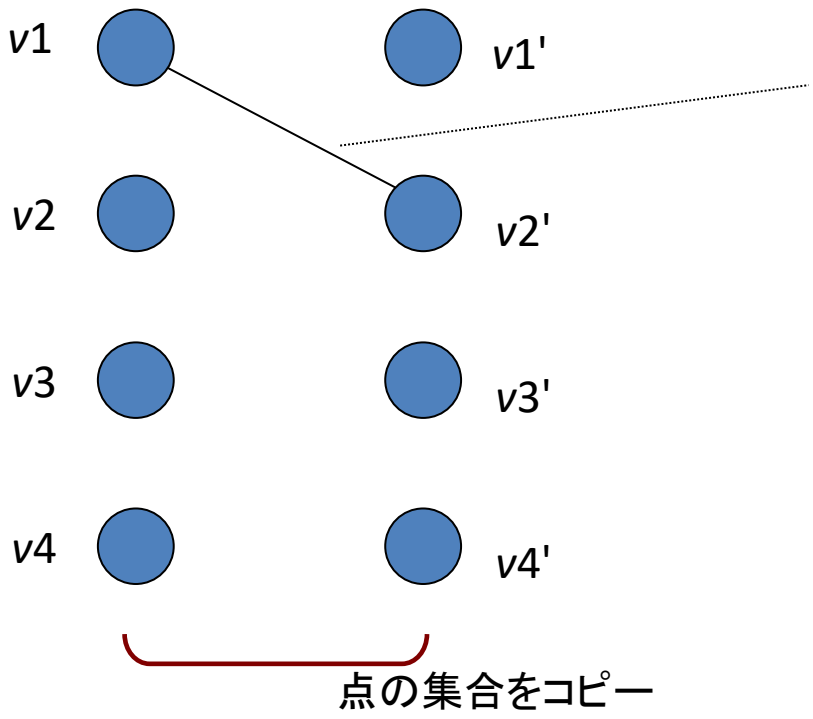




このグラフ上での最小閉路カバーに相当

最小閉路カバー(Minimum cost cycle cover)

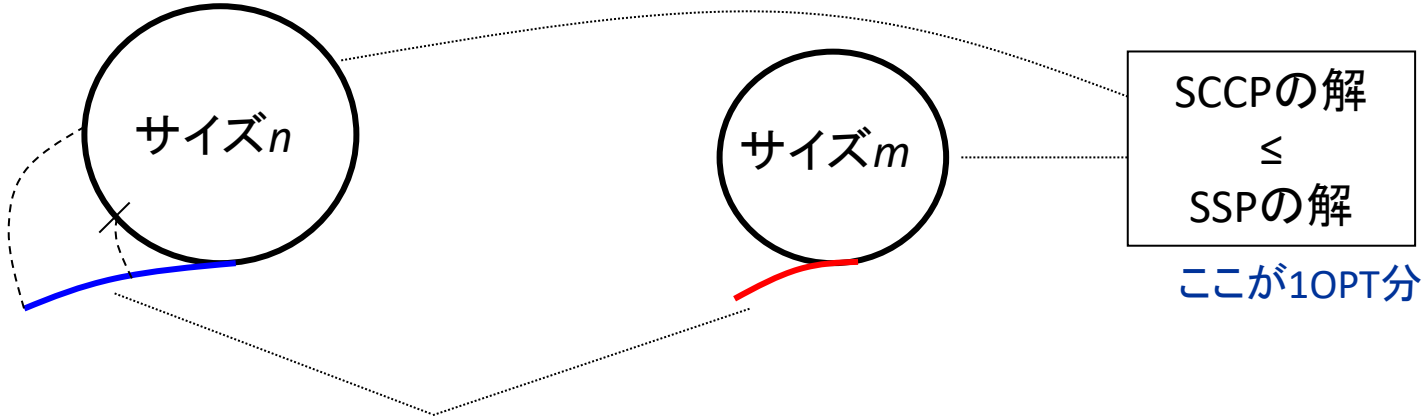
- 二部グラフ上での最小重みマッチングに帰着できる
  - $O(n^3)$



SSPのグラフにおいて有向枝(1,2)に枝があった場合に同じ重みの枝を作成

SCCP→SSP

- SCCPの解からナイーブにSSPの解を作成する
  - ひとつだけ選んで結び目を解く
  - できたものを単につなげるだけ
- するとこれは4OPTの解になっている！



この2つの文字列のprefix-suffixマッチングのサイズがn+m（≤ OPT）①
より大きくなることはない
cf. ユークリッドの互助法（もし違ったら最小公約数の長さのサイクル1つでOK、ということになる）

≤ OPT

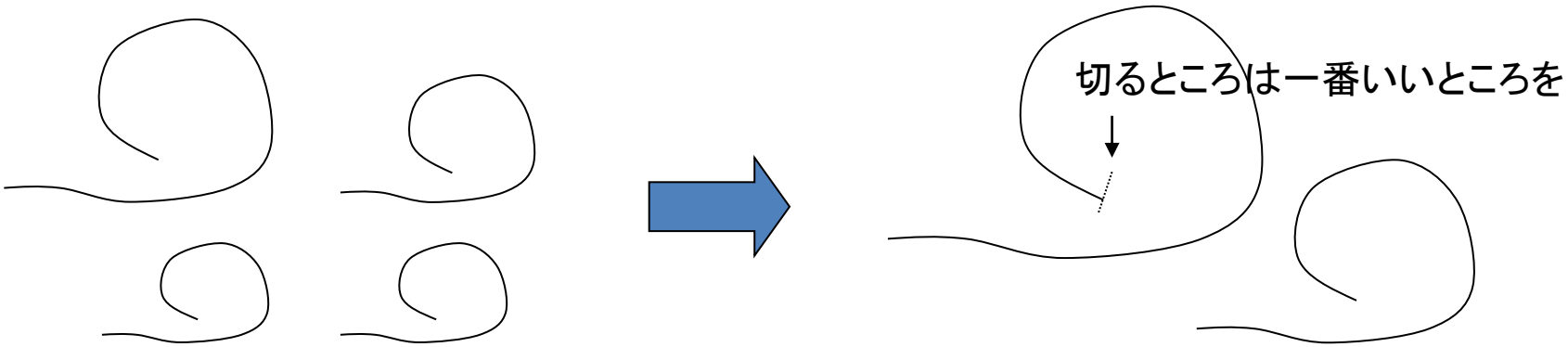
①のところをつなげなかったので倍の長さになった可能性があるため

ここが3OPT分

この飛び出た文字列だけのSSPの解＋2×OPT＞飛び出た文字列をつなげたもの

もう少し頑張れば3OPTにできる

- 2SCCP
  - ひとつのサイクルが2つ以上の文字列から構成されるようなSCCP（*i.e.*, 文字列1つからなるサイクルを許さない）
- 3OPTのアルゴリズム
  - SCCPを計算し、各サイクルから文字列を作る。
  - それらに対し2SCCPを計算
  - できたサイクルから「一番短い」文字列を作成しつなげる
- さらに頑張ることも可能



詳しい解析はGusfield本参照