生物配列解析アルゴリズム
2

渋谷

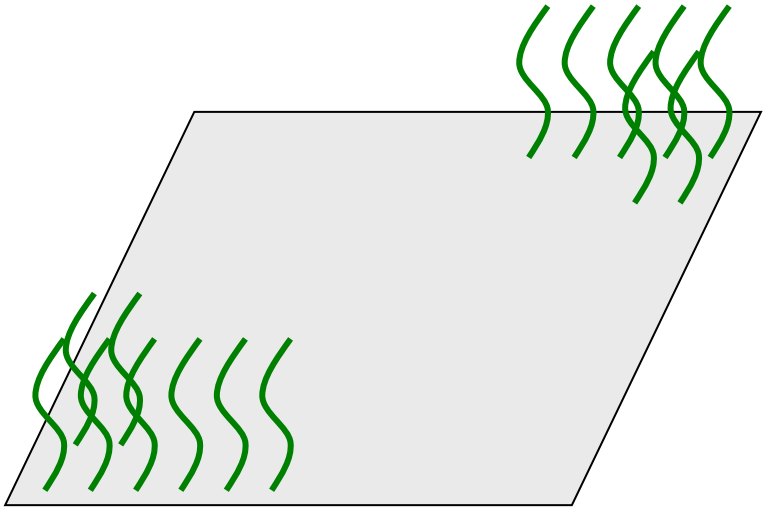
東京大学医科学研究所ヒトゲノム解析センター
（兼）情報理工学系研究科コンピュータ科学専攻
http://www.hgc.jp/~tshibuya

今回の話題

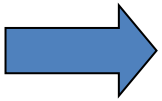
- DNAアセンブリ（続き）
- ペプチド配列決定アルゴリズム
- RNA構造予測アルゴリズム
  - Nussinov
  - Zuker
  - Akutsu
- 準最適解アルゴリズム
  - Eppstein
- 木のedit distanceについて

SBH (1)

- マイクロアレイ
  - 同時にたくさんの配列の有無を同時に判定する
    - 相補塩基の結合を利用する
    - ある程度ならば量も測ることが可能
- SBH (Sequence by Hybridization)
  - たくさん(～1万)の文字列を部分文字列として持つかどうかから元の文字列を推測する問題



マイクロアレイ



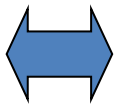
実験結果

配列1	Yes
配列2	No
配列3	No
配列4	Yes
配列5	No
...	

SBH (2)

- 最も簡単な想定
  - k–mer（長さkの文字列）をすべて生成する
  - 完全に一致したかどうかが判定できる

ATG
TGA
GAG
AGT
GTT

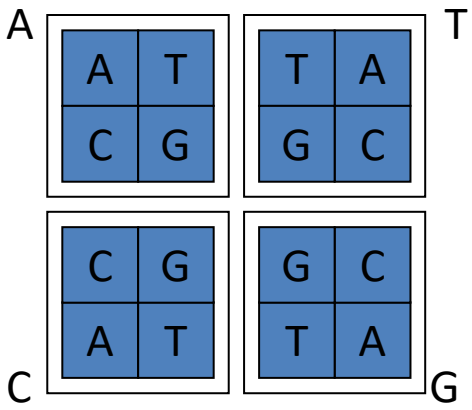


ATGAGTT

SBH (3)

- グレイコードとSBH
  - SBH用マイクロアレイの実際の構成法
    - 一文字ずつ付加していく
    - 付加する領域と付加しない領域は離れているとうれしい
  - Gray code
    - $G_0 = \{0, 1\}$
    - $G_i = \{g_1, g_2, \ldots, g_{2^i}\}$とすると

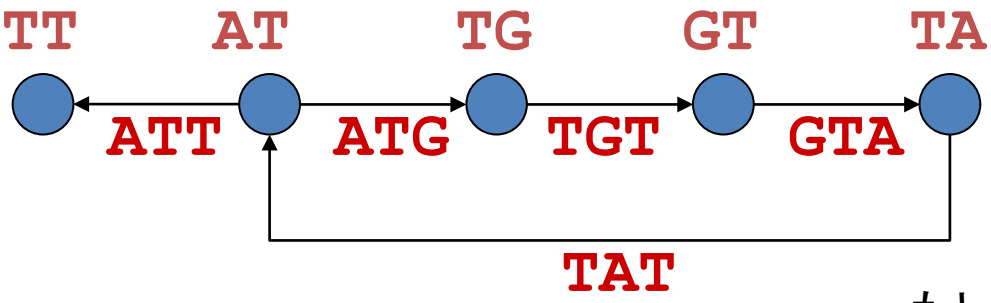
      $$G_{i+1} = \{0g_1, 0g_2, \ldots, 0g_{2^i}, 1g_{2^i}, 1g_{2^i-1}, \ldots, 1g_1,\}$$
  - 2次元4文字に拡張してマイクロアレイを作成
    - エラーが少なくなる



ひとつの文字列が1度しか現れない場合

Eulerian Path問題

オイラーパスが存在しない場合は適当な枝をk重枝にする



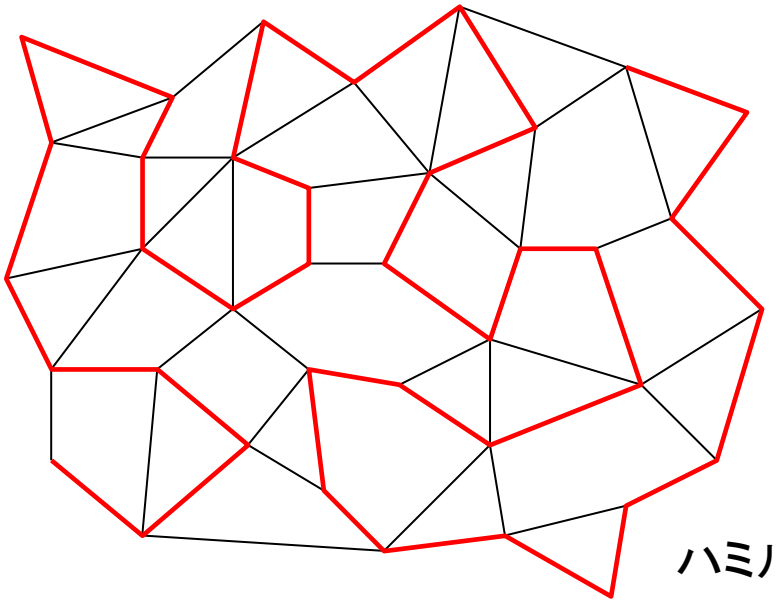
もとの長さのまま計算しようとするとハミルトニアンパス問題になってしまう

ATG
TGT
GTA
TAT
ATT

→ ATGTATT

オイラーパスとハミルトニアンパス

- オイラーパス・閉路
  - グラフ上ですべての枝を一度通るパス・閉路
  - 線形時間アルゴリズムが存在
- cf.ハミルトニアンパス・閉路
  - グラフ上ですべての点を一度通るパス・閉路
  - NP完全

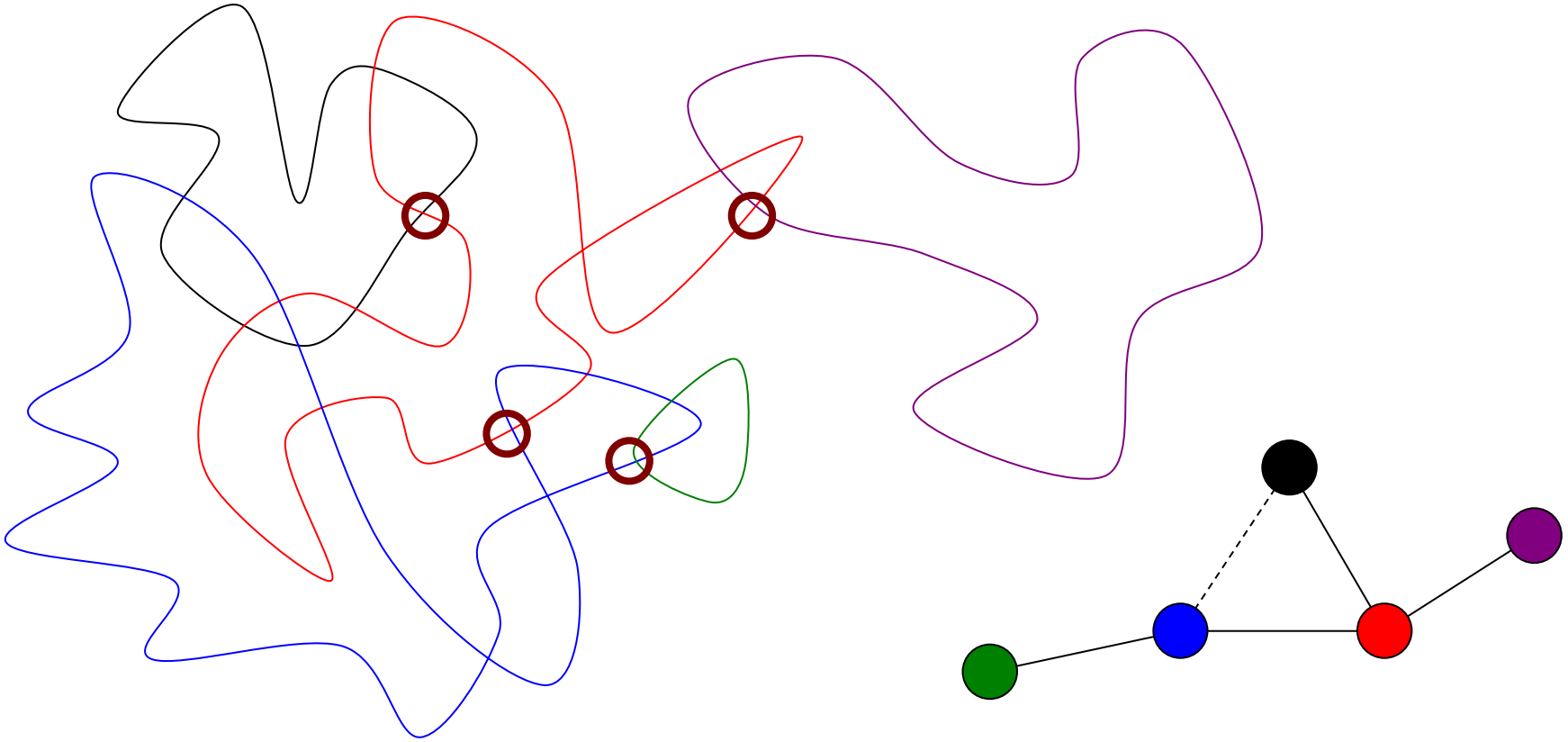


ハミルトニアンパス

奇数次数の点が3つ以上あるため、オイラーパスは存在しない

オイラー閉路のアルゴリズム

- 適当に閉路をたくさん作成する
- 閉路同士のグラフを考え、DFSで全域木を作成する
- 全域木の枝それぞれについて、それに対応する元のグラフの点を選び、そこにおける巡回順を交換してつなげていく



SBHの問題点

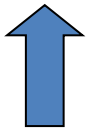
- あまり長い配列を解読することができない
  - オイラーパスはユニークではないことが多い
  - すぐ2回以上同じk-merが現れてしまう
  - 実際の実験ではせいぜい１００～１２５程度
    - 1本の長さは6～7程度

- 解決法
  - 複数回、SBHをかける（前の解を利用してプローブを設計）
  - 他の技術と組み合わせる
  - 既知の配列集合から選ぶのに用いる
  - Universal base の利用！

SBH with Universal Bases

- Universal base
  - どの塩基とも結合できる便利な人工塩基
    - 他にも特定の2つの塩基と結合する塩基などもある
- これを用いて何をするか？
  - PatternHunterに似ている
    - ただしSBHが先

ATATCGTAGATCATGGC



reconstruct!

ATA..G..G..C
TAT..T..A..A
ATC..A..T..T
TCG..G..C..G
CGT..A..A..G
GTA..T..T..C

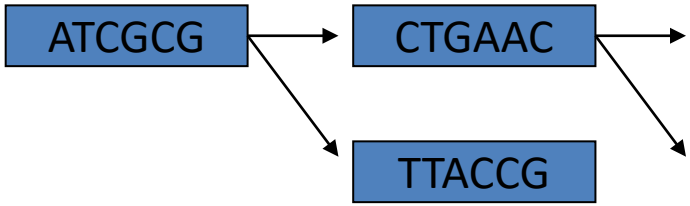
オリジナルのパタン

$(1)^n(0^{n-1}1)^m$

ランダムの方がよいとも言われる

どうやって再構成するか？

- 力任せのBFSでやる
  - n文字ごとに次のn文字でありえるものを列挙
    - 枝分かれする場合はBFS
    - 枝分かれがなくなるか理論値より長くなったら終了
    - 複数解があれば、できるだけ全部がマッチして、しかもできるだけ短いものを出力
  - 計算時間は「高い確率」で線形時間
    - 最悪は指数時間



実験的にも理論的にもなかなかよい

- 理論的な上限
  - Yes/Noが k 個あると、2^k 本の文字列を区別できる
  - すなわち理想的には$k/2$の長さまで判定できてしかるべき
    - $k/2$の長さの配列の総数＝$4^{2/k}(=2^k)$
- そしてこの方法だと、理論的にも期待値$O(k)$を達成できる
- 問題点
  - エラーを考えていない
  - universal base同士の結合を考えてない←技術的問題

ということで、これは実用には至っていない

Short Read Assembler

- Short Read での古典的手法の問題
  - 信頼できるOverlapグラフが得られない
    - 十分な長さのOverlapが得られない
    - エラーレートも従来よりも高い
  - Overlapグラフが大きくなりすぎ、処理できない
    - 高いエラーレートと短いリードの影響をなくすために高いカバレージが必要
      - 50-100x （昔のSanger系では7-10xだった）
    - メモリ・計算時間ともにきわめて困難
    - ただし、最近は、並列度の高いツールも出てきている（Forge）

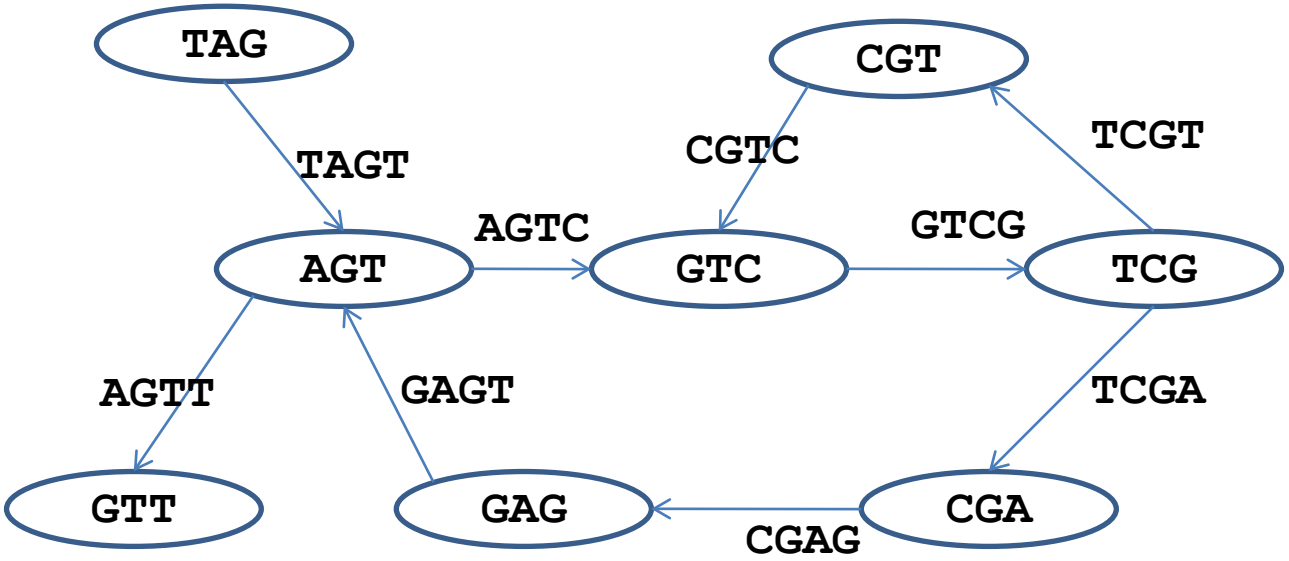
SBH的手法を用いる（de Bruijnグラフ）

Velvet, ABySS, SOAPdenovo, Allpath, Euler (Euler-SR), SSAKE
などほとんどの次世代シーケンサー用アセンブラ

de Bruijn Graph

- すべての長さkの部分文字列(k-mer)を枝、その間の$k-1$の長さのoverlapを点と考えたグラフ
  - Overlap graphと点と枝の役割が逆
  - (理想的には)この上でオイラー・パスを求めればよい
    - 頻度も重要(カバー数から予測)
  - overlap計算も(理想的には)いらない
  - NGSで十分「深く」読めば、(理想的には)「すべて」のk-merを読むことも可能

TAGTCGTCGAGTT



この場合、各枝の頻度が分かれば、元の配列をアセンブリできる

注意

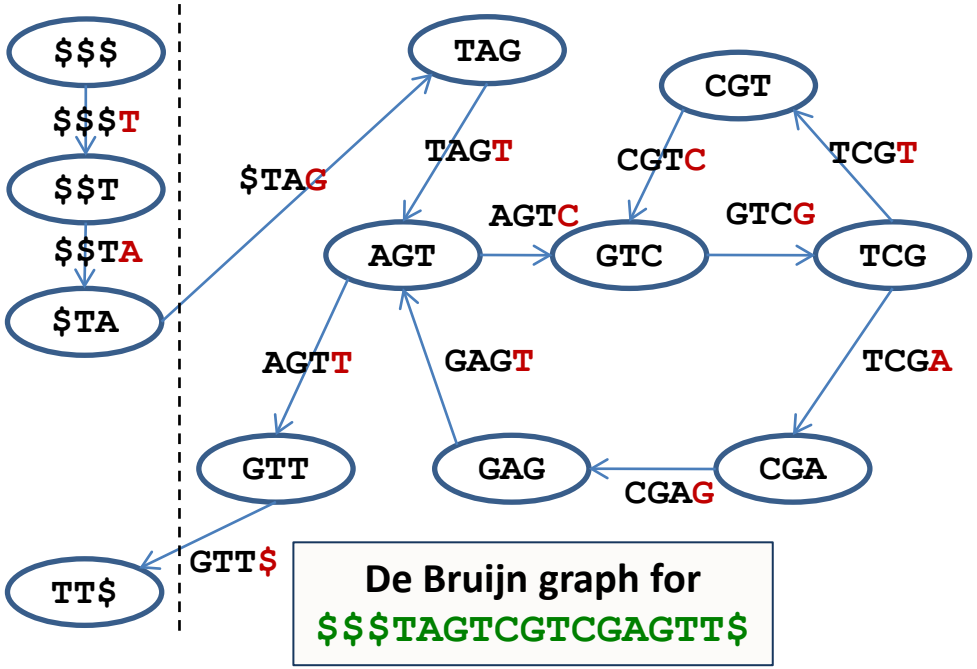
- ノードのラベル同士が $k-2$ のoverlapを持っていても、実際にその間の枝に対応する配列が元のリードの中になければ、枝は作らない
  - k–merのOverlap graphのline graphではない、ということ

部分文字列として**AGTTA**がない限り、
これらのノードがあっても枝は結ばない

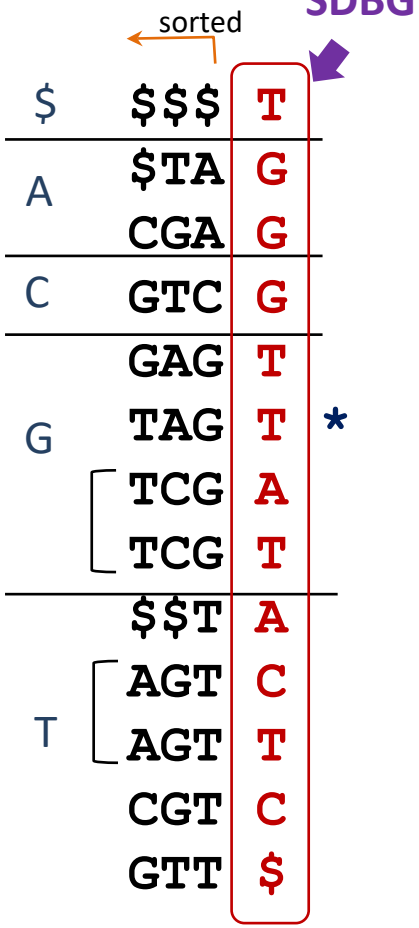
FM-index的表現：SDBG (Succinct de Bruijn Graph)

[Bowe, Onodera, Sadakane, Shibuya 2012]

- Sort edges according to the reversed prefix of length $k-1$
  - *i.e.*, Compute the lexicographical order of GAT for the edge k-mer TAGT
- SDBG: the last characters in the above sorted order
- Additional small auxiliary support data structures
  - Additional 2 bit flags per edge
    - stored in o(n) compressed rank/select data structure

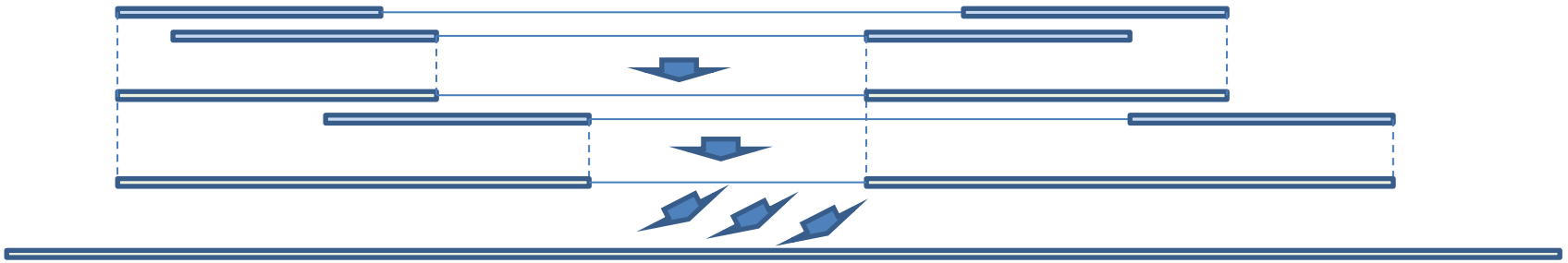


Our succinct representation



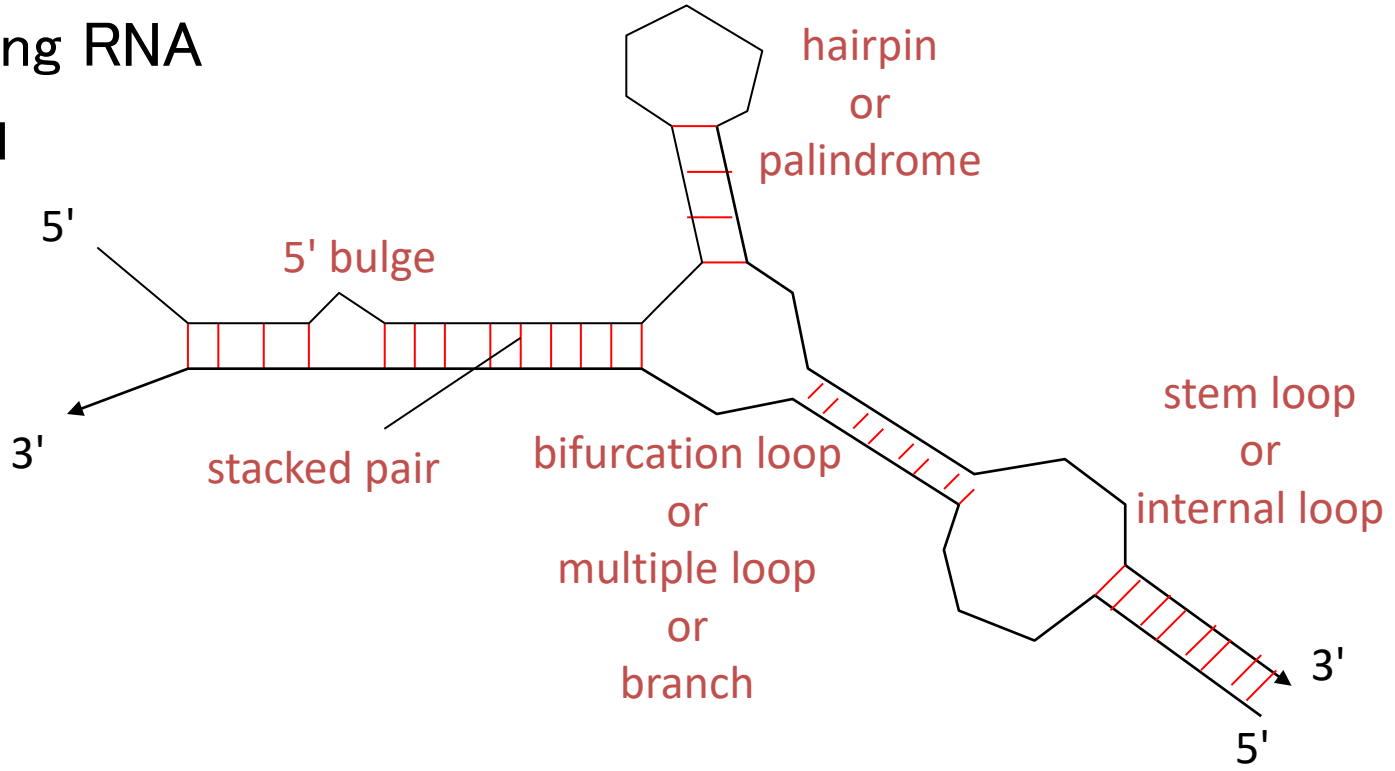
実際はもっと複雑

- （たとえば）ALLPATHアセンブラのアルゴリズム（概略）
  - de Bruijn Graph上で、枝分かれのない部分を求める(unipath)
  - paired readでくっついているunipathをまとめたデータ（全体のごく一部）をたくさん作成し、そのデータごとに（並列に）アセンブリ
    - まず適当なカバレージの（＝リピートに含まれていなさそうな）paired readを得る
    - 両readともにoverlapを持つpaired readを探して、結合
    - これを繰り返して、真ん中のギャップがなくなるまで繰り返す
      - 多くはこれで一つだけの配列が得られる
      - リピートが絡むと、大量（$>10^3$)の候補が得られることもある　(all path)
  - できたものをOverlap-layout的手法でつなぎあわせる（ここは並列化できない）
    - だめな候補を削除



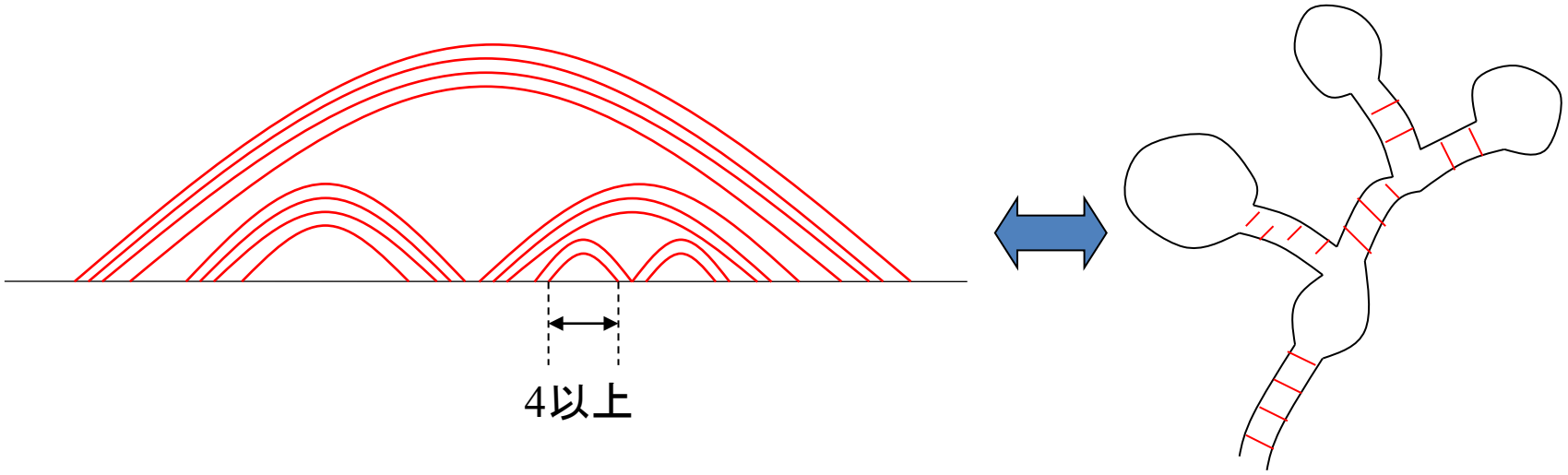
RNAとその構造

- RNA（リボ核酸）
  - A(Adenine), C(Cytosine), G(Guanine), U(Uracil)の4種類の塩基からなる細胞内の塩基鎖
  - C–G, A–U, G–U(弱い)が結合する→立体構造
    - 結合した塩基鎖は逆方向
- ホットトピック
  - non–coding RNA
  - RNA–seq



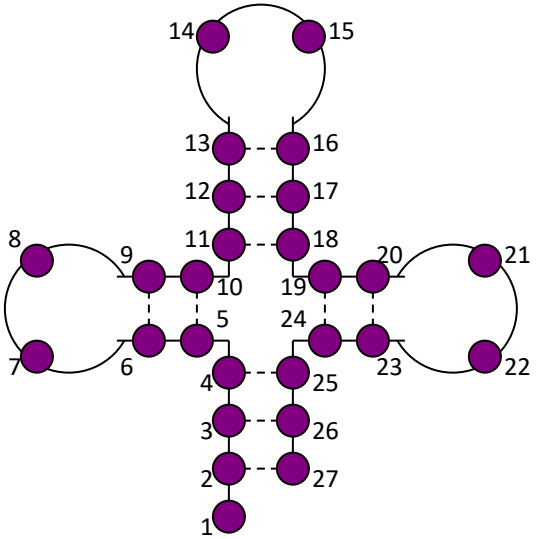
RNA2次構造

- 2次構造とは
  - $(i:j)$ と $(k:m)$ という結合がある場合に、$i<k$ ならば $j>m$ であるような構造
    - $(i:j)$: i 番目の塩基と j 番目の塩基の結合
      - $|i-j| \geq 4$
  - 計算をこれに限定すると探索空間が減るので、よくこれに限定した構造予測がなされる

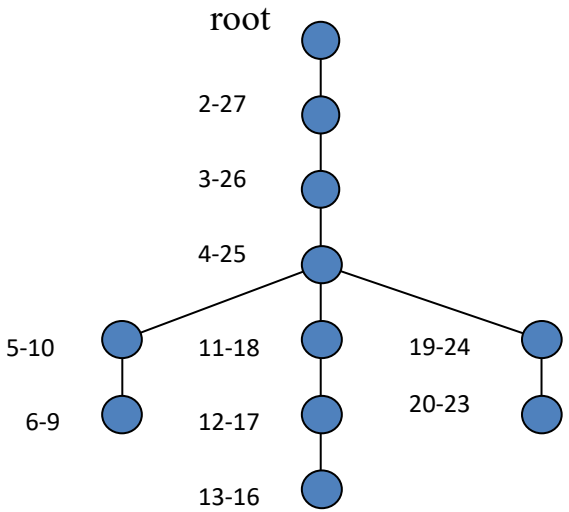


2次構造の表現方法

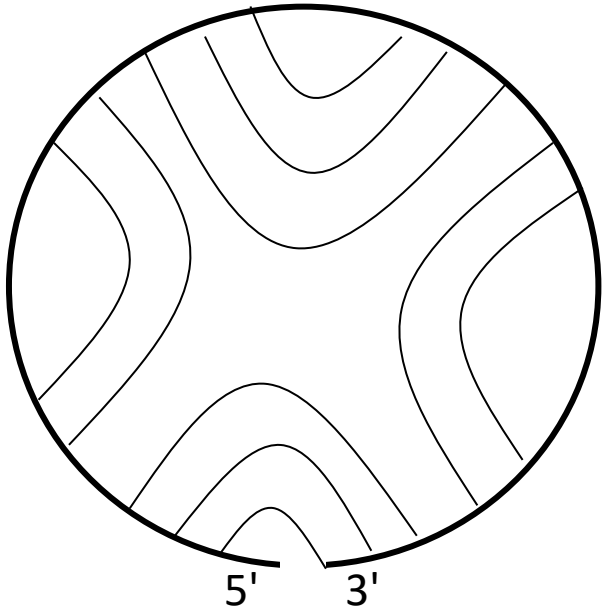
- 様々な表現方法がある



Tree Representation

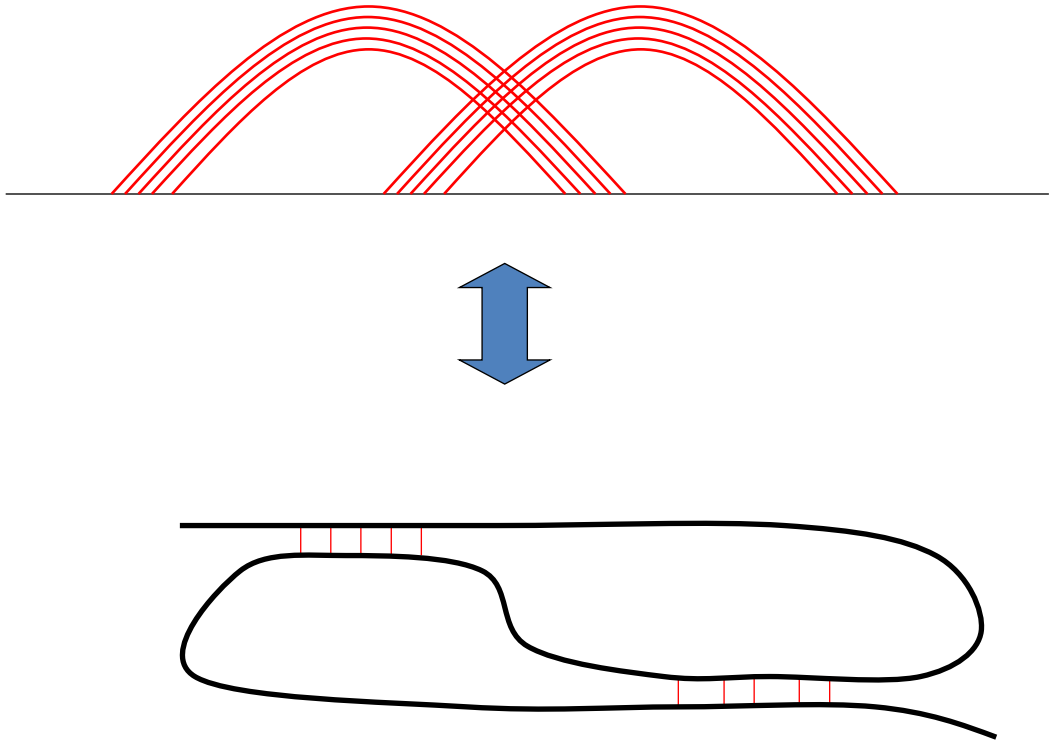


Circular Representation



二次構造ではない構造

- Pseudo-knot
  - 決して珍しいわけではない



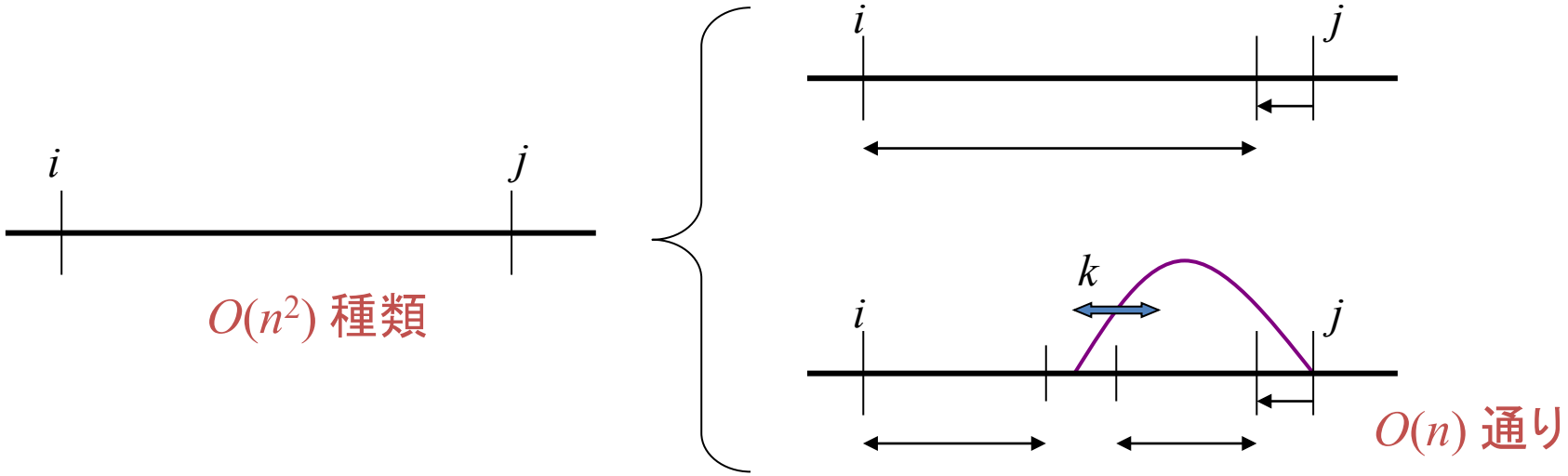
古典的予測手法: Nussinov et al. '78 (1)

- 最も結合するペアの数が多い二次構造を求めるアルゴリズム
  - 時間計算量 $O(n^3)$
    - 空間計算量 $O(n^2)$
  - Pseudo-knotは考えない
    - branchも考えなければ、$O(n^2)$にできる
- 考え方
  - 長さ 1 の部分文字列は二次構造を持たない
  - すべての長さ k の部分文字列に対して、結合ペア数が最大の二次構造を求める
    - 長さ k-1 以下の部分文字列に対する解を利用して一つの解を得るのに $O(n)$
    - 対応する部分文字列の数は$n-k+1$
  - $k=n$ まで順番に計算していく

実際の漸化式

- $MMS(i,j)$: i 番目から j 番目までの部分文字列に対する解

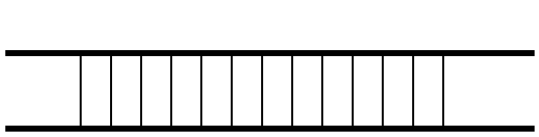
$$MMS(i,j) = \max(MMS(i,j-1),$$
$$MMS(i,k-1) + MMS(k+1,j-1) + 1$$
$$\text{for all } k(i \le k < j, match(k,j) = 1))$$

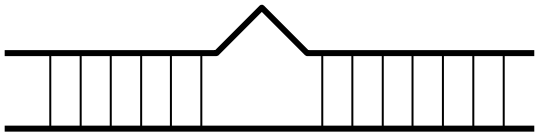


ブランチがなければ、$O(n^2)$

ギャップ・ペナルティ的なものを考えるには？

- 途中に変なものが混じるようなものにはペナルティーを課したい

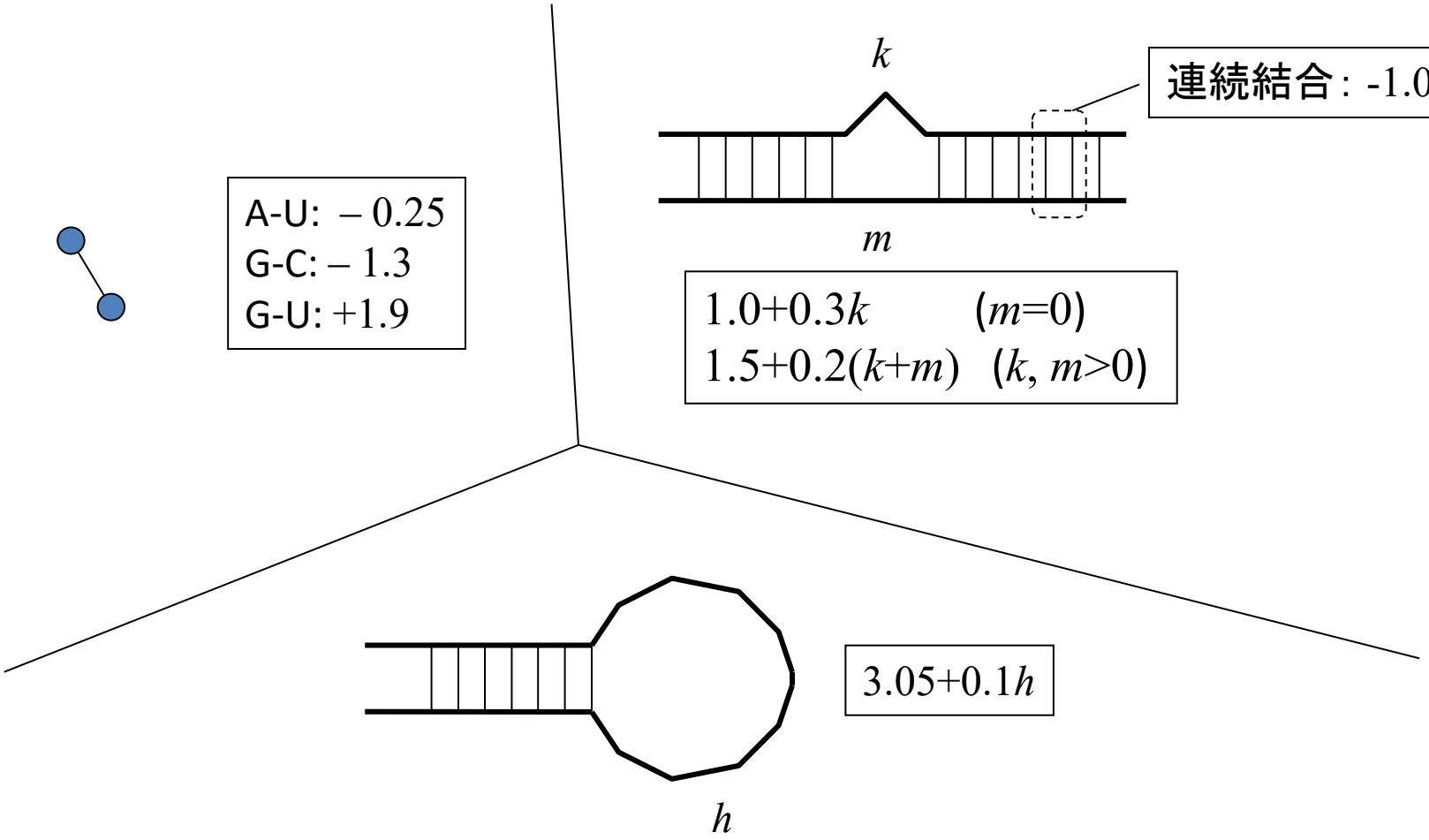




左の方が良さそうだが、結合数は同じ

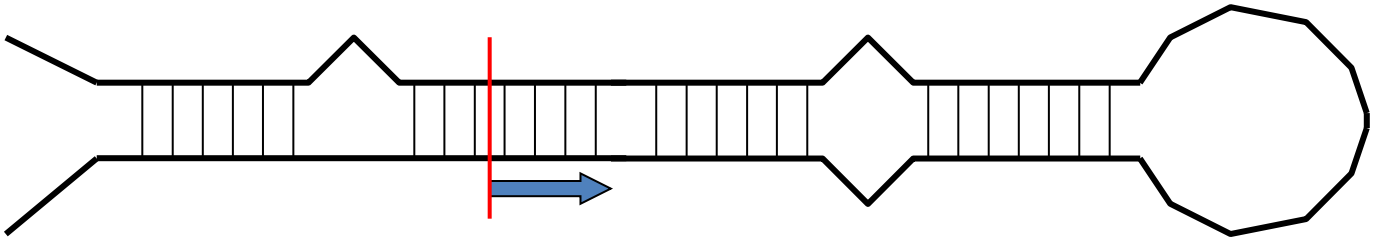
Waterman and Smith '78 (1)

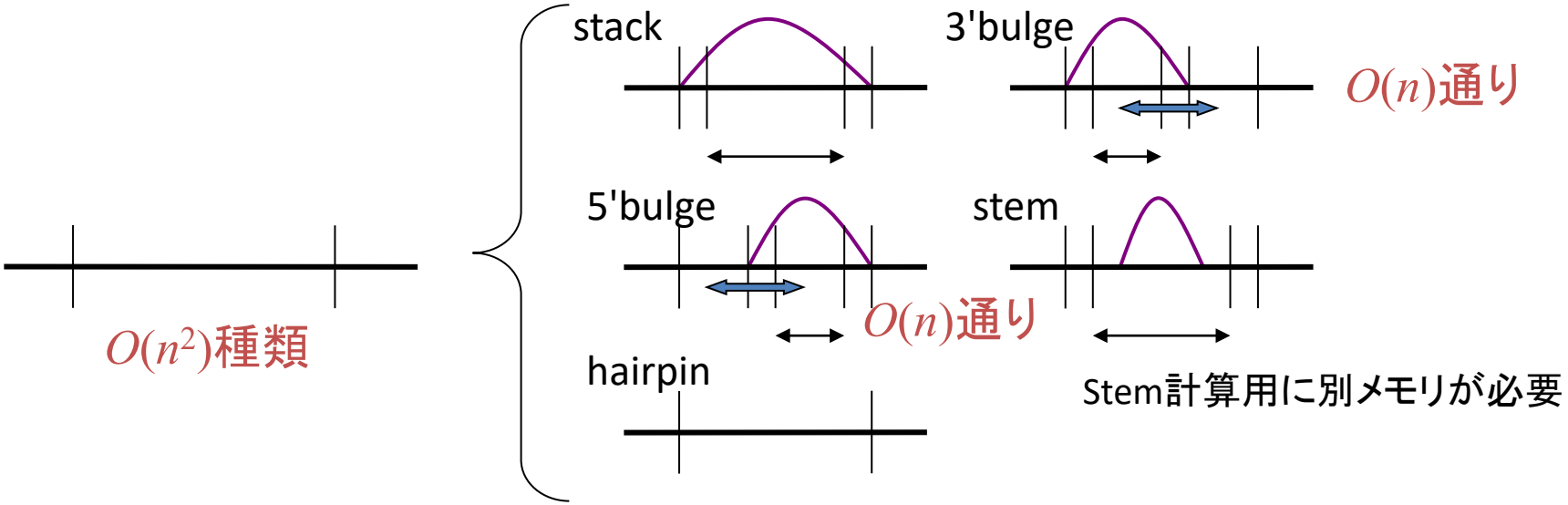
- ペナルティのつけ方＝エネルギーを定義する
- エネルギー最小化の問題



Waterman and Smith '78 (2)

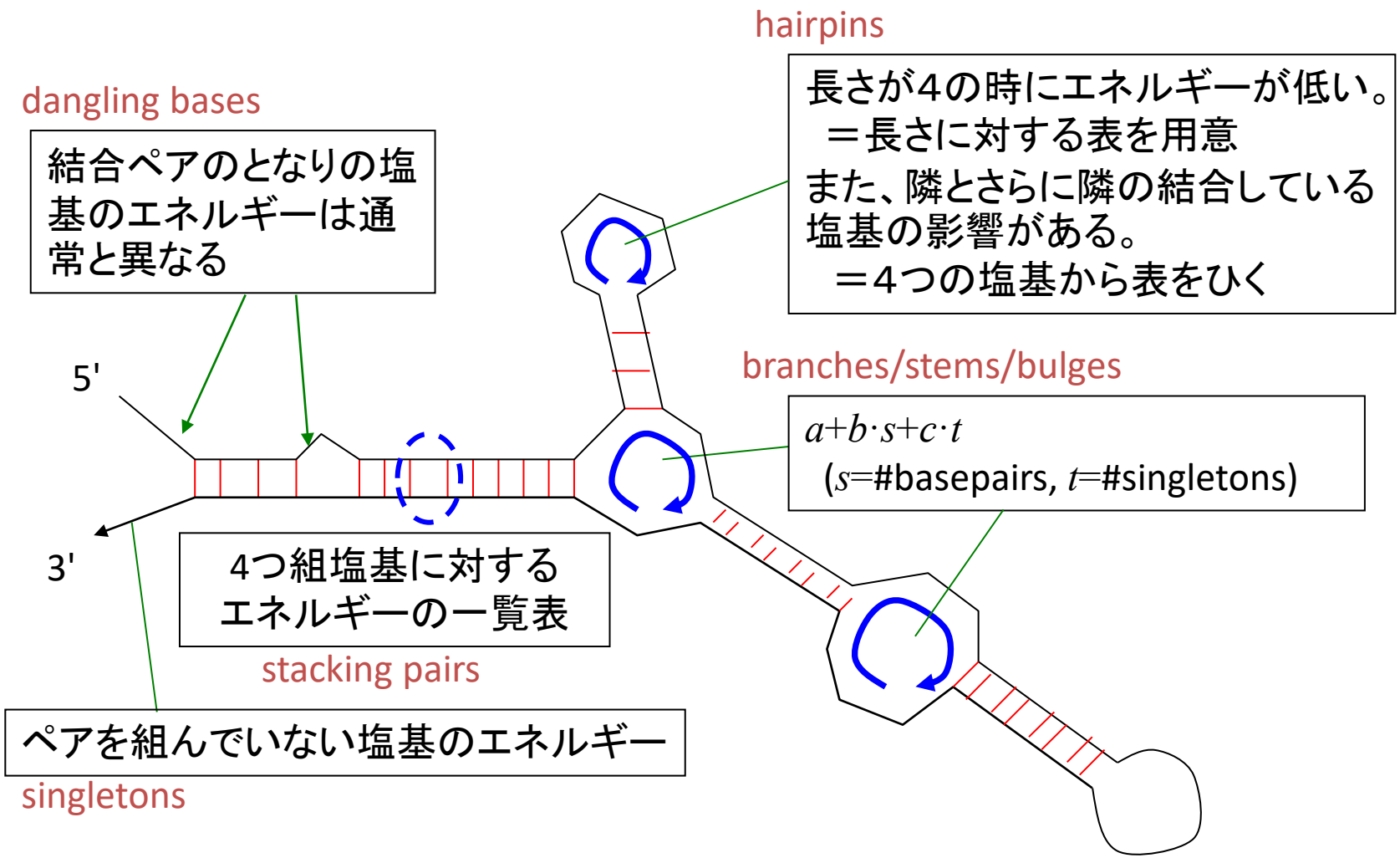
- Branchは無視して最適な構造を求めるアルゴリズム
- 同様のDPで$O(n^3)$で計算できる





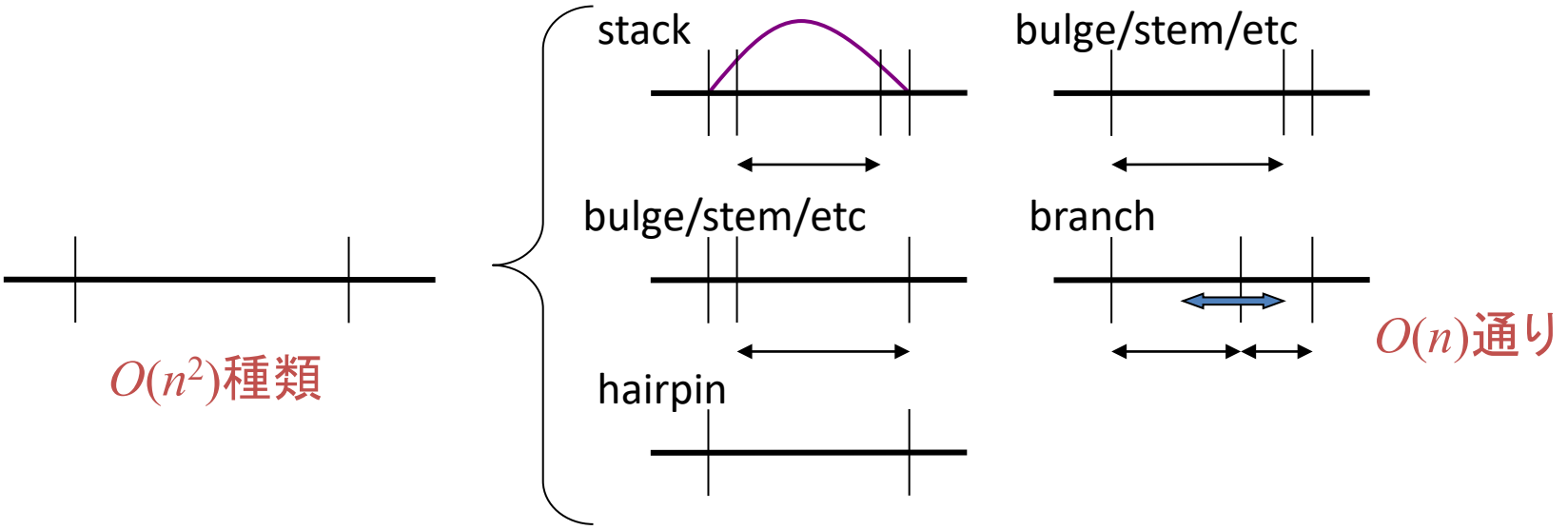
Zuker and Stiegler '81 (1)

- いろいろ条件を追加しても $O(n^3)$

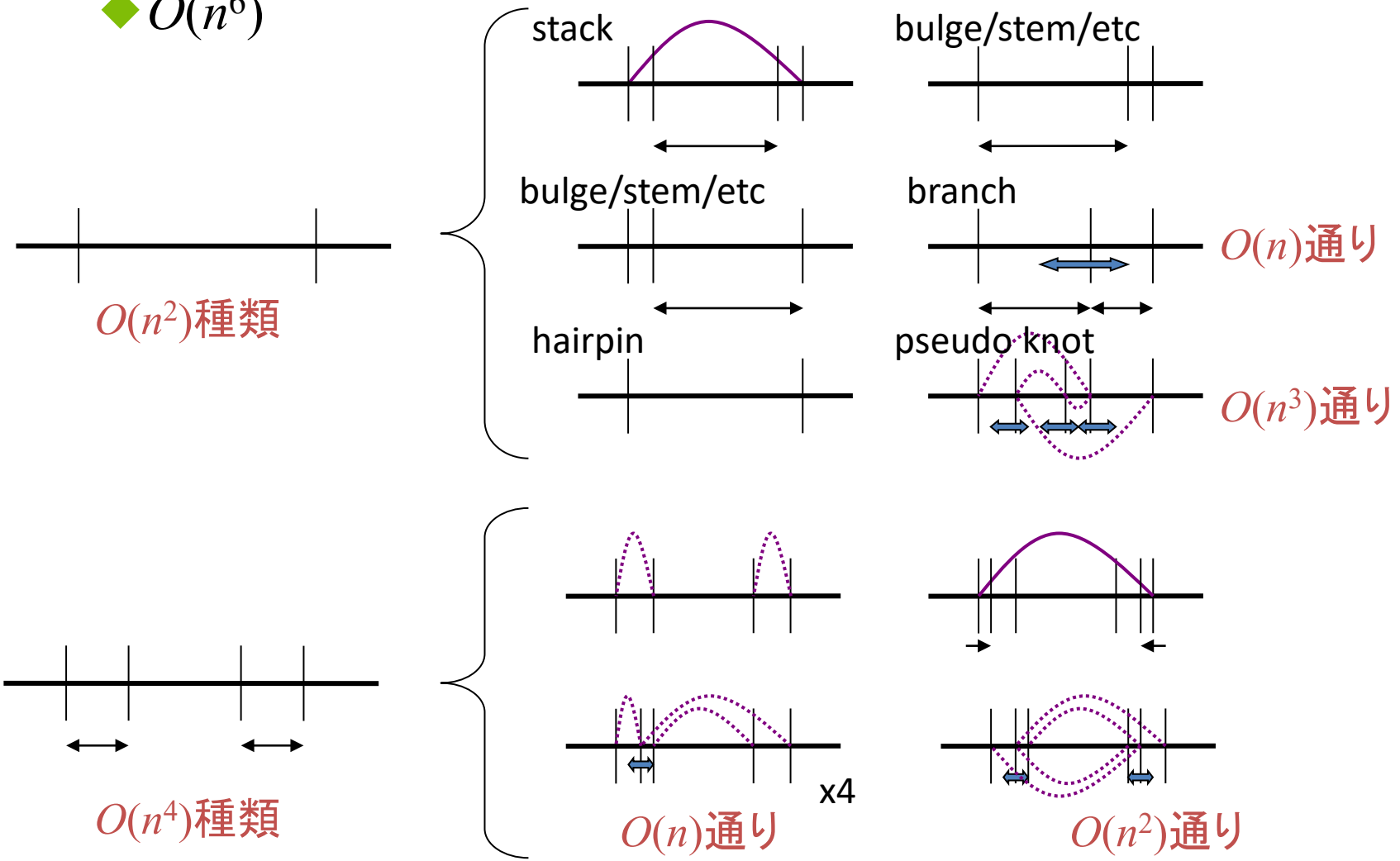


Zuker and Stiegler '81 (2)

- 考え方は全く同じ！



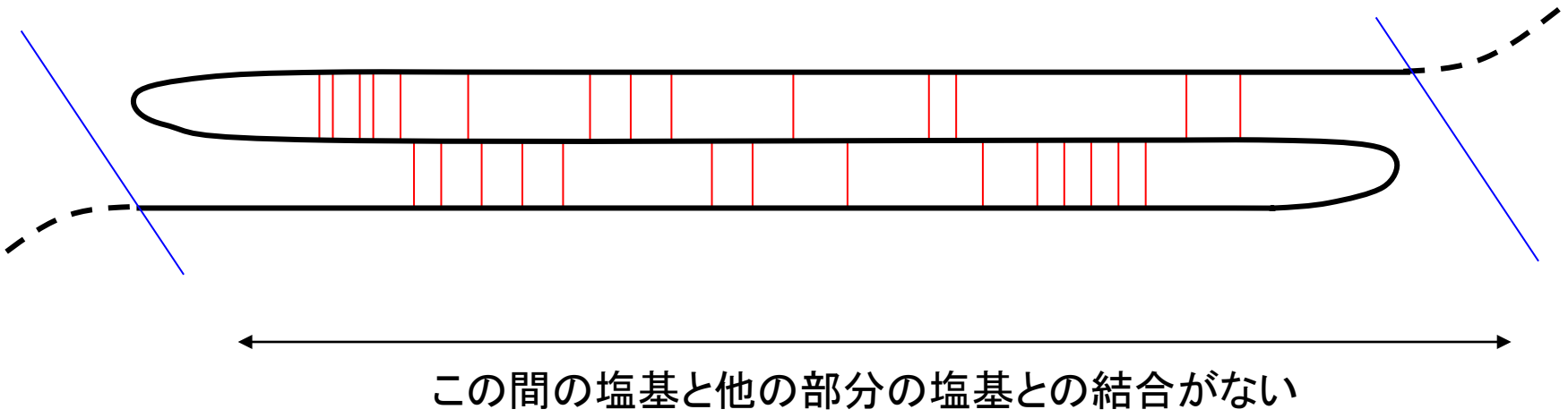
- さらにpseudo-knotを追加すると、、、
  - $O(n^6)$



Simple Pseudo-Knot (1)

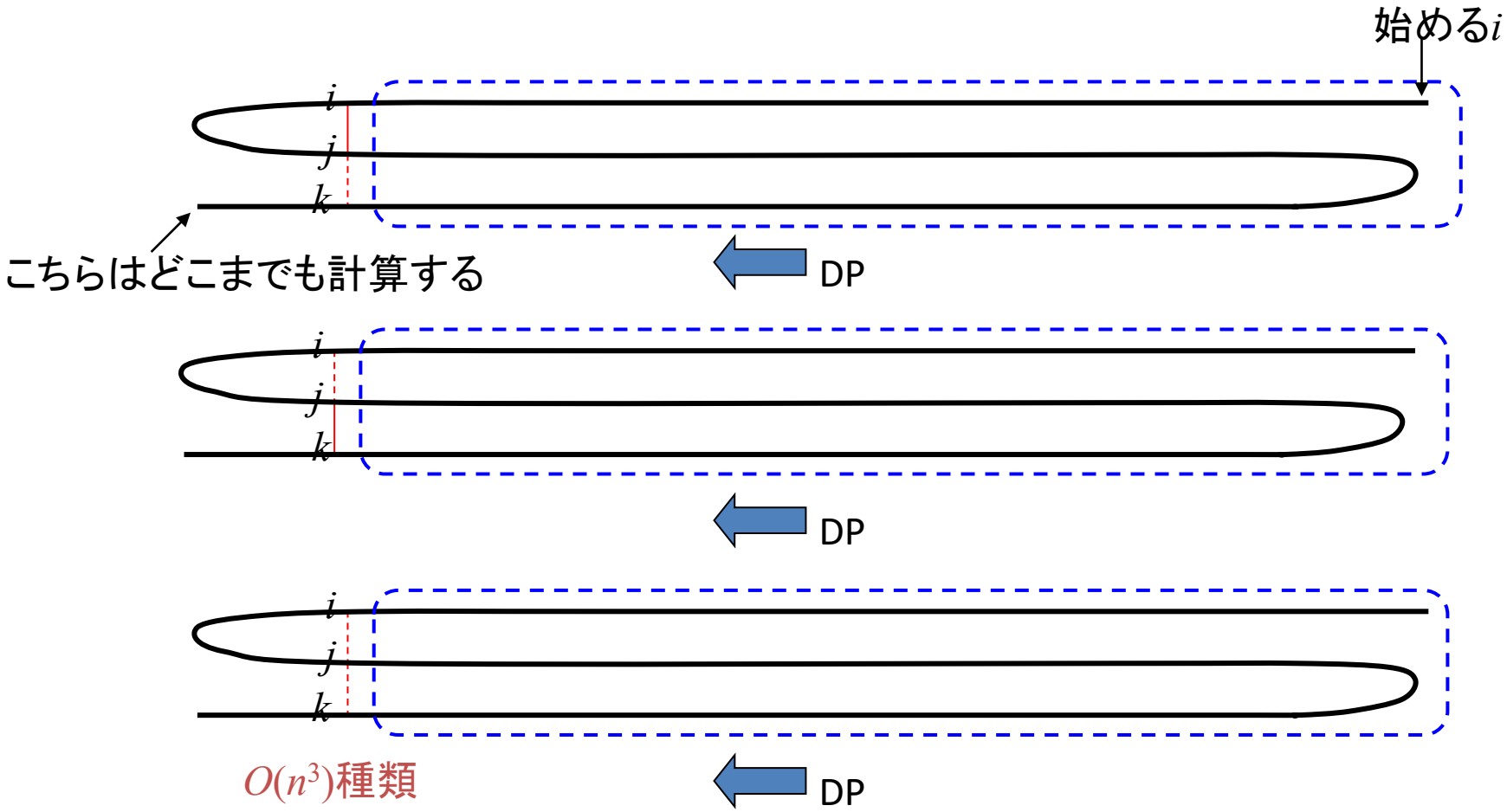
[Akutsu '00]

- 制限をつけて、計算量を少し落とすことが可能
  - $O(n^4)$



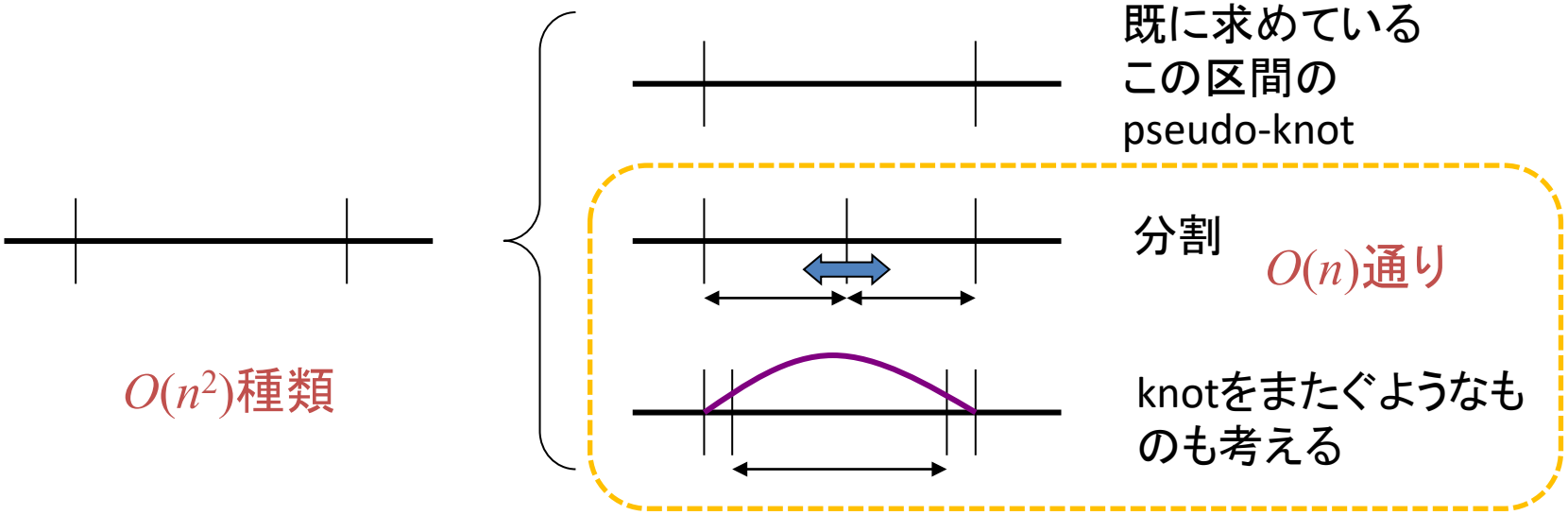
Simple Pseudo-Knot (2)

- pseudo-knot区間内のDP計算
  - $O(n^3)$
  - これを始点となる i すべてに関して計算→$O(n^4)$



Simple Pseudo-Knot (3)

- Multiple simple pseudo-knotにしても問題なく $O(n^4)$



Context Free Grammarとの類似点

- やっていることはかなり文脈自由文法チックなこと
  - Nussinov's algorithm
    - S→aSu | uSa | gSc | cSg | gSu | uSg |
      SS | aS | cS | uS | gS | ε
  - それぞれの生成規則にスコアがついていると思えばよい
- この考えを用いてスコアの学習が可能(確率文法)
  - 遷移確率を定義
    - 構造がわかっていれば、遷移確率がわかる
      - 十分な数の学習データが必要
    - あるいはEMアルゴリズム等で文章の期待値を最大化する
      - ただしえられるものは局所解
  - 確率がわかれば構造を予測することができる
    - Cocke-Younger-Kasami algorithm

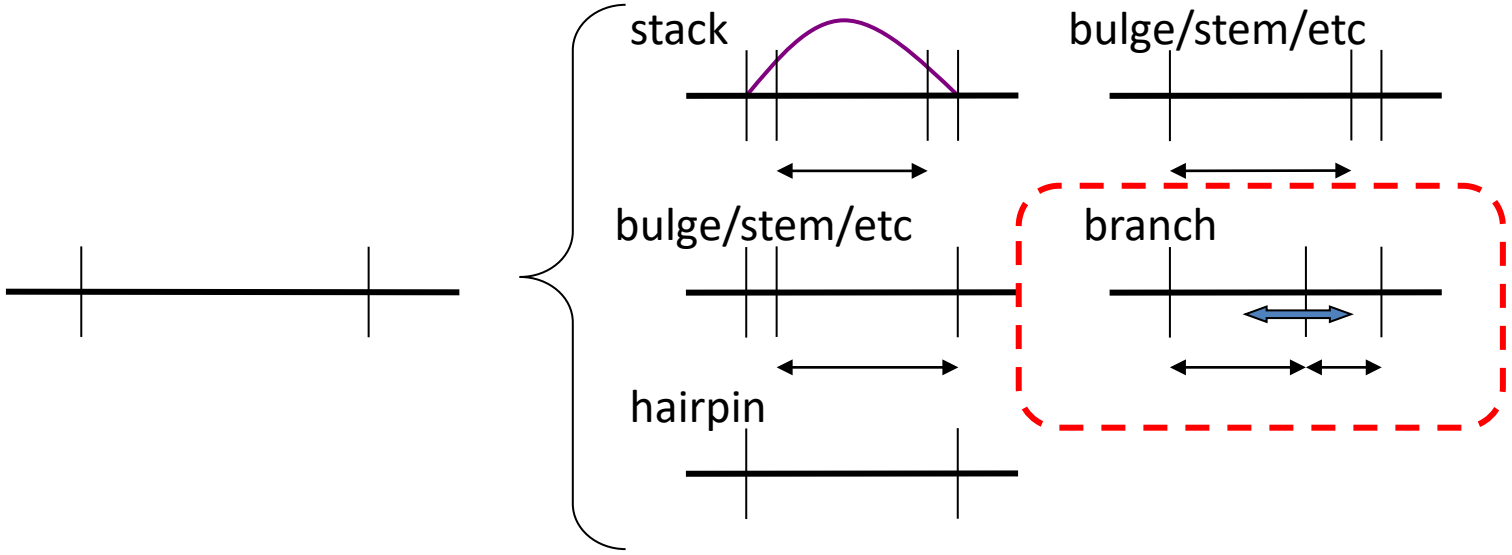
Suboptimal Structures [Zuker '89]

- こうやってエネルギー最小の構造を求めても、正しい構造を予測できる確率は実際にはかなり低い
  - エネルギーが低い候補構造は、最適解以外にもたくさんあり、どれが正しいか不明

    → **エネルギーが低い構造を列挙したい**
- 方法
  - Zukerのアルゴリズムではすべての区間において最適解を求めているが、それらのうちの上位のいくつかを出力する
  - すべての区間 $[i..j]$ において k 番目までの解を求める
    - $O(kn^3)$
  - すべての区間 $[i..j]$ において、最適解よりΔ以上悪くない解をすべて求める
    - 計算時間不明
- 最短路問題に帰着される問題ならば、もっと効率のよい解法あり
  - DPやA*アルゴリズムで求めたアラインメントの準最適解
  - HMMにおけるViterbiアルゴリズムの準最適解

$O(kn^3)$ アルゴリズム

- Branchの計算において、k 番目まで同士の和の上位 k番目までの計算をしないといけない
  - ヒープを用いて$O(k \log k)$
  - がんばれば $O(k)$ にもできるが難しい
- それ以外の場合も $O(k)$
  - ソートする方が楽でその場合 $O(k \log k)$

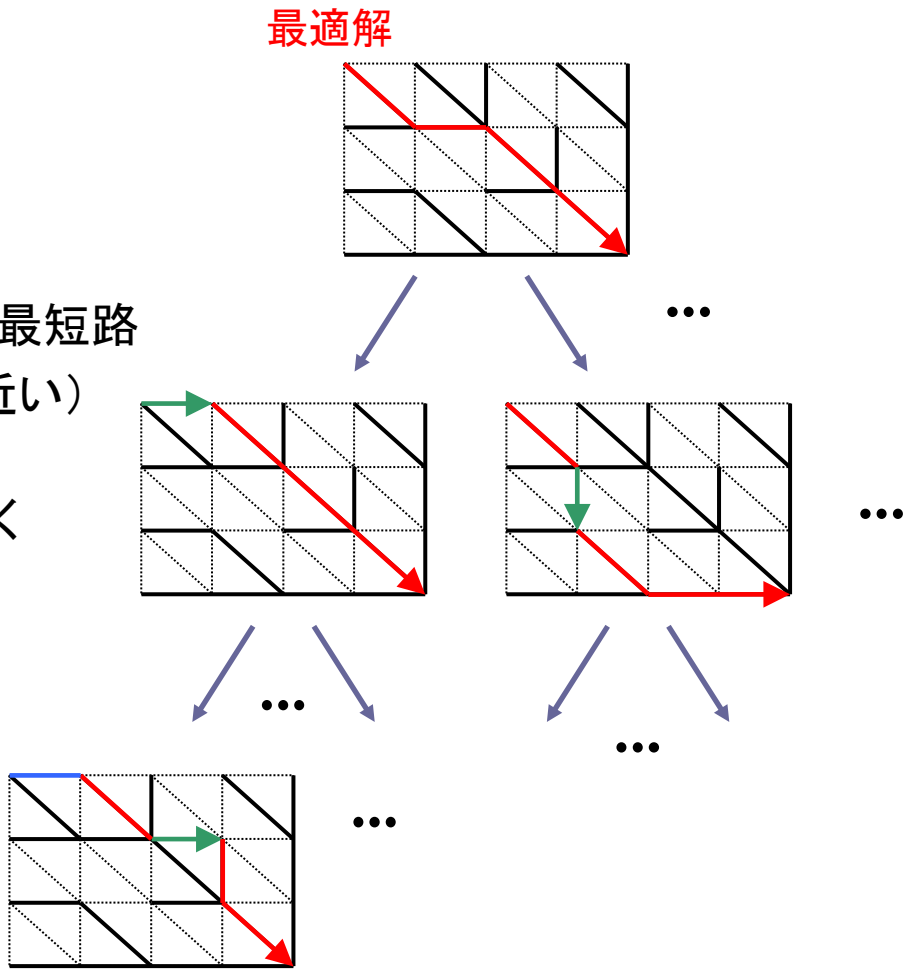


k最短路問題とその関連問題について

- k 最短路問題
  - 2点間のパスを短いものから k 本求める
    - 順番に求めるか、セットを求めるか？
- 準最適解問題
  - 2点間のパスで最短路+Δの長さ以下のものをすべて求める
- GΔ問題
  - 最短路+Δ以下の長さの準最適解が通過することのある点と枝からなる部分グラフを求める

Eppstein '94 (1)

- Path Heap
  - 全パス集合を表すヒープ
  - ルート：最適解（のひとつ）
  - Sidetrack
    - 終点への最短路木上にない枝
  - 子パス
    - 親パスPの途中まで(P')＋sidetrack＋最短路
      - P'はPの最後の（＝終点に一番近い）sidetrackを必ず含むようにする
        - 親がルートである場合を除く
    - 必ず、子パス長≧親パス長、となる
- k 最短路
  - ヒープを作成して上から辿る
- 準最適解列挙
  - ヒープをDFSで探索



Eppstein '94 (2)

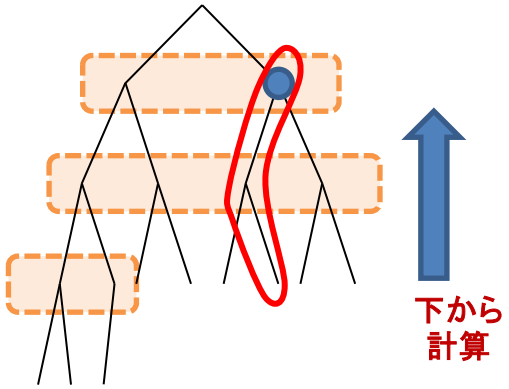
- Path Heapの問題点
  - パスの子供の数が$O(n)$になってしまう
    - そのまま上から辿るときに、効率が悪い
    - ひとつのパスの子供についてはヒープで管理
      - 次数を制限→ 4
  - 無限の大きさになってしまう
    - $O(|E|+|V|)$の大きさのグラフで表現可能
    - その計算時間は$O(|E|+|V|)$
      - 簡易アルゴリズムでは$O(|E|+|V| \log |V|)$
      - 超ナイーブアルゴリズムだと$O(|E|+|V|^2)$

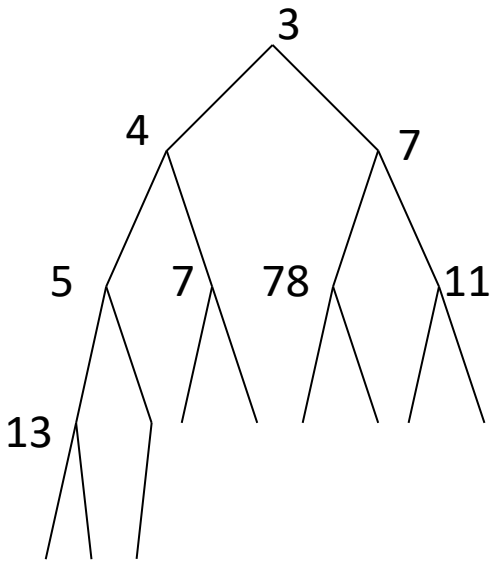
Eppstein '94 (3)

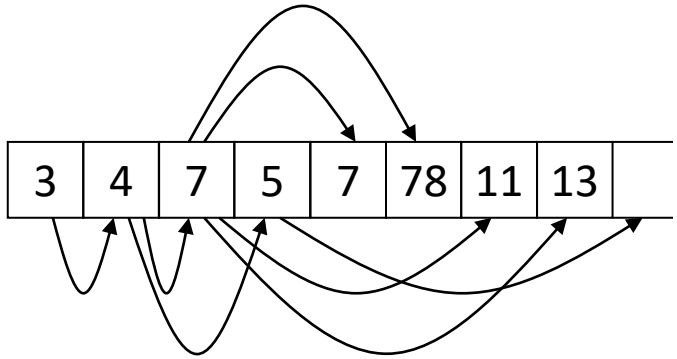
- Path Heap Graphの作成方法
  - Dijkstra法（アラインメントの場合DPも可）で終点への最短路木を作成し、各sidetrack e に対し e を用いた時に損する量を $loss(e)$ として計算
    - ある点 v に関してそれを始点とする枝のうち最小の $loss(e)$ を持つ枝を $sidetrack(v)$ とおく
  - 各点から出て行くsidetracksに対して $loss(e)$ の値に基づきヒープを作成する
    - 出次数 d の点における計算量は $O(d)$ でOK！→全体で $O(m)$
  - すべての点に関してその点から終点への最短路上の点 v の $sidetrack(v)$ を $loss(sidetrack(v))$ に基づいてヒープを作成する
    - 線形時間で可能（ただし難しい）

Eppstein '94 (4)

- ヒープの線形時間の古典的構成法
  - 適当な順番で配列に押し込む
  - 後ろから順番に自分の子孫を全て修正していく
    - $n/2 + 2n/4+3n/8+4n/16+... \rightarrow O(n)$





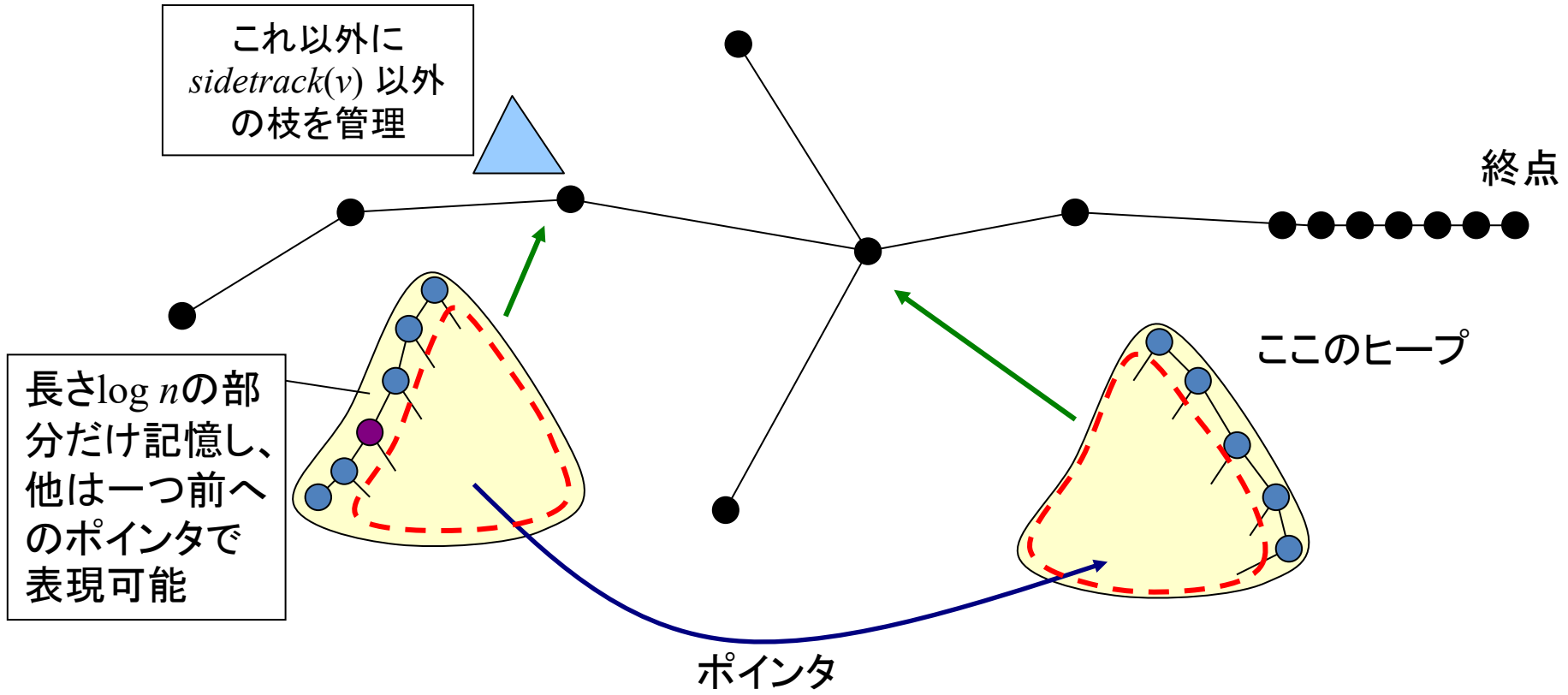


必ず子より親の方が小さい値を持つ

i番目の親は$i/2$番目
（ルートを1番目とした場合）

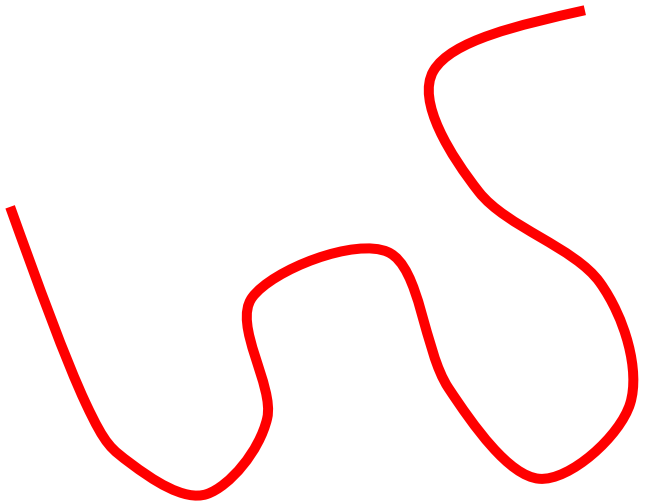
Eppstein '94 (5)

- PathHeapの作成方法
  - うまくバランスさせながら、必要なところしか記憶しない
    - AVL木を使う（バランスできるが、余計なメモリが必要）
    - もっと賢い方法→一番右側のパスに挿入し一番左側に持ってくる
  - より高度なアルゴリズムを用いれば$O(|V|+|E|)$も可能だが、かなり難しい



Eppstein '94 (6)

- あとは上から辿るだけ
  - $O(k \log k)$
    - ヒープを用いる
    - ソートされている
  - $O(k)$
    - [Frederickson '93]
    - ソートされていない
  - 最短路＋Δ以内
    - DFSで可能
  - 解のパス（や対応するアラインメント）を出力する場合
    - $O(nk+k \log k)$や$O(nk)$
    - その出力サイズが$O(n)$であるため



バリエーション

[Shibuya '97]

- 少しヒープに細工をするだけでより有用な準最適解だけの列挙を行うことも可能

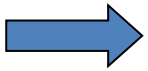
最適解

```
REAFSQAIWRATFAQVPESRSLFKR==
ADFLV-ALF-EKFPDSANFFADFKGKS
KNG-S-LLFGLLFKTYPDTKKHFKHFD
LAAVF-TAYPDIQARFPQFAGK-DVAS
GSGVE-ILY-FFLNKFPGNFPMFKKLG
```

準最適解

```
REAFSQAIWRATFAQVPESRSLFKR==     REAFSQAIWRATFAQVPESRSLFKR==
ADFLV-ALF-EKFPDSANFFADFKGKS     ADFLV-ALF-EKFPDSANFFADFKGKS
KNG-S-LLFGLLFKTYPDTKKHFKHFD     KNG-S-LLFGLLFKTYPDTKKHFKHFD
LAA-VFTAYPDIQARFPQFAGK-DVAS     LAAVF-TAYPDIQARFPQFAG-KDVAS
GSG-VEILY-FFLNKFPGNFPMFKKLG     GSGVE-ILY-FFLNKFPGNFPMFKKLG
```

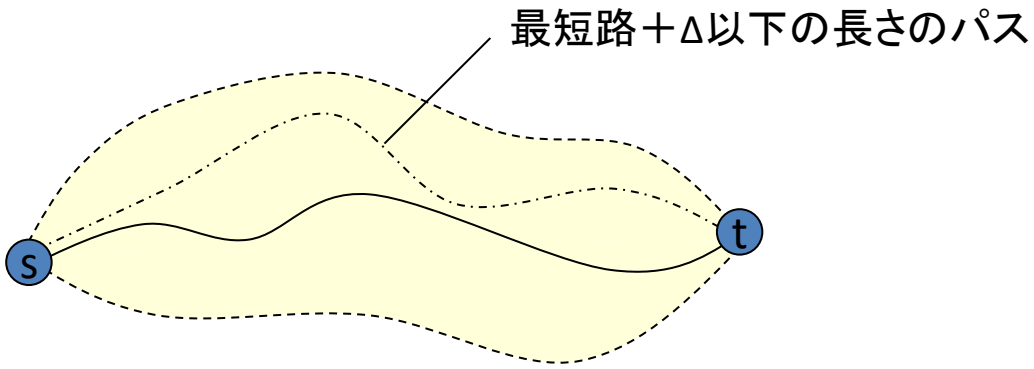
組み合わせによる準最適解（列挙不要）

無視！ 

```
REAFSQAIWRATFAQVPESRSLFKR==
ADFLV-ALF-EKFPDSANFFADFKGKS
KNG-S-LLFGLLFKTYPDTKKHFKHFD
LAA-VFTAYPDIQARFPQFAG-KDVAS
GSG-VEILY-FFLNKFPGNFPMFKKLG
```

最適解と2箇所以上異なっている

- 最短路＋Δ以下の長さの準最適解が通過することのある点と枝からなる部分グラフを求める
- アルゴリズム
  - 終点への最短路木を求め、$loss(e)$をすべての枝について計算する
  - 枝の長さを$loss(e)$だと思って始点からDijkstra法等を用いてΔ以内で到達できる点と枝の集合を求める＝$G\Delta$

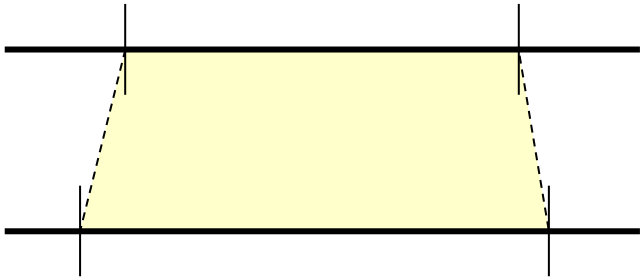


類似構造を持つと思われる配列集合の予測

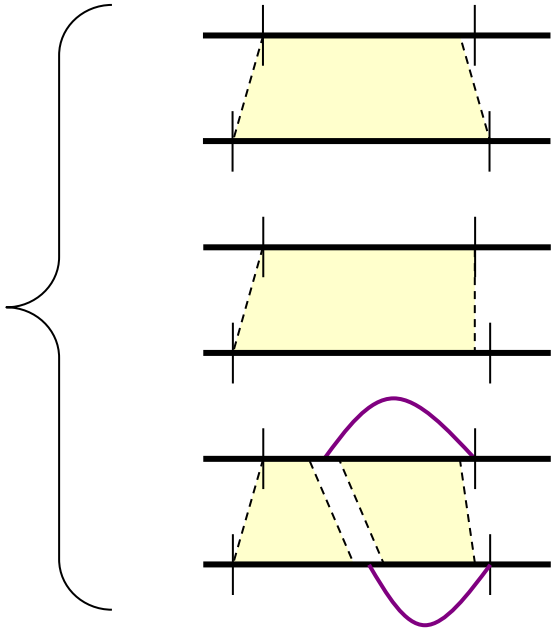
- 簡単な方法
  - マルチプルアラインメントを行う
  - i 列と j 列が結合するかどうか？
    - 何も考えずにエネルギー計算してエネルギーの和を求める
    - 相互情報量を求める
      - 結合することが「多い」のであれば、片方がAであれば、もう片方はTとなることが「多い」 → 結合するペアは互いに独立ではない
      - どの配列においても結合していれば、相互情報量は非常に高くなる
  - あとはその情報を用いて普通にZukerアルゴリズムを走らせる
- 問題点
  - アラインメントする時に文字列が似ていないかもしれない。

2本以上の配列の同時構造予測 [Sankoff '85]

- 最適解はかなりの力技
  - k本に対して$O(n^{3k})$
- Nussikov版簡略版アルゴリズム
  - Zukerに拡張しても基本は同じ



$O(n^4)$種類



$O(n^2)$通り

予測したRNA構造の比較はどうする？

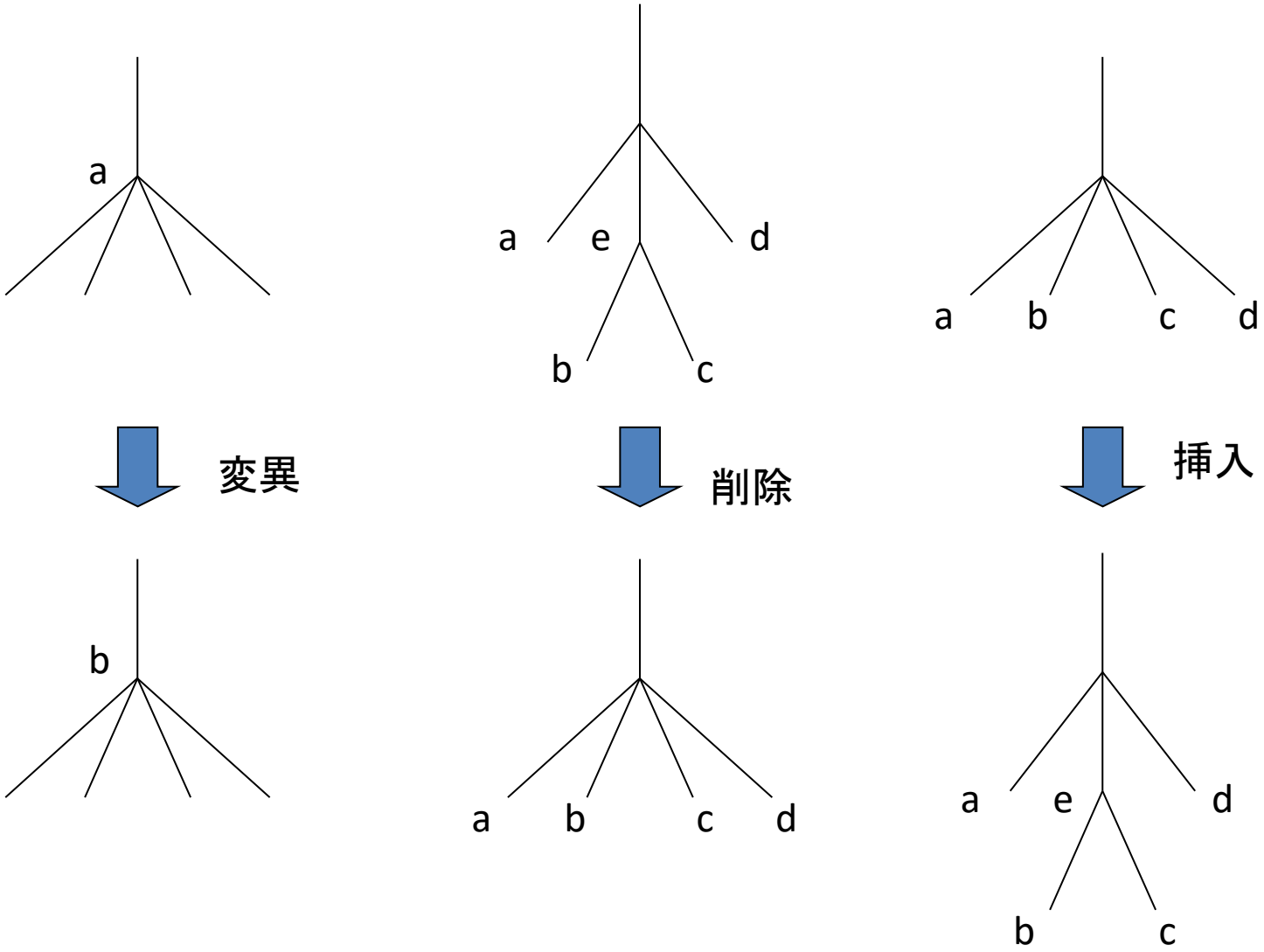
- 木構造の比較をする
  - RNAは木で表現できるため
    - 木はordered tree
      - unordered だとさらにNP-hardな問題になることが多い
- やりたいこと
  - String pattern matching の種々の問題の木への拡張
    - 木構造がどれほど似ているか？
    - 保存されている構造はどこか？
    - 部分構造の検索・類似検索等
      - 接尾辞木の木への拡張の話は既出
    - ただし生物学的な根拠に基づく比較とは言い難い

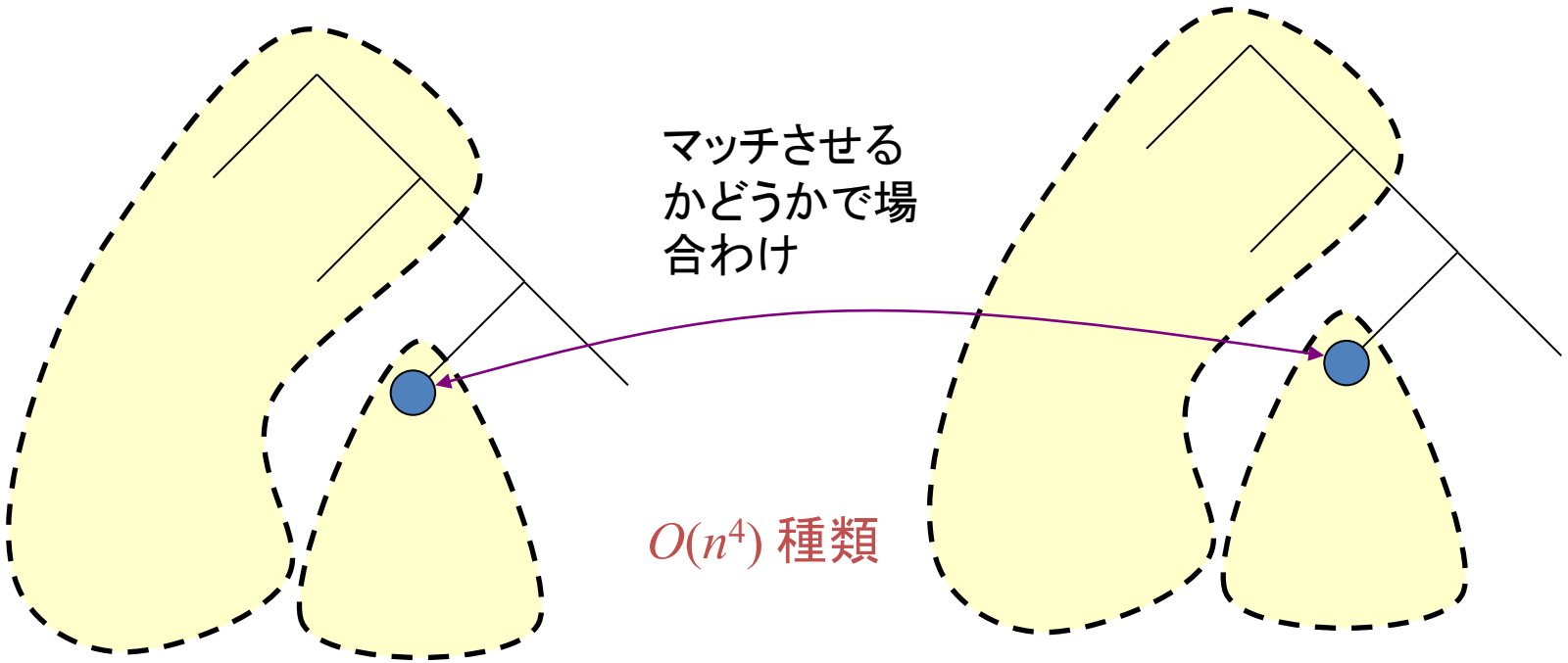
Tree Edit Distance vs Tree Alignment

- 配列の場合、両者はほとんど同じ概念だったが、、、
- 木では異なる概念
  - Edit Distance
    - できるだけ少ない回数の挿入、削除、変異によって片方の木を別の木に変換する
    - 当然ながらmetric distanceである（三角不等式を満たす）
  - Alignment
    - それぞれに適当にノードを挿入してtopologicalに同じ木を作る
      - edit distanceにおいて挿入した後に削除することに相当
      - すなわちedit distance以上の値をとる
      - metric distanceではない（三角不等式を満たさない）
    - ラベルは必ずしも同じでなくてもよいが、ラベル同士のスコアを比較する

Tree Edits

3種類の操作



DP for Tree Edit Distance　[Zhang et al '89]

- 簡単なバージョン
  - ポストオーダーで番号をつける
  - $T[i..j]$ をその間の番号だけからなる森とする
  - $left(i)$ を i の子孫で一番左の孫とする
  - Tにおいてjをiの子孫、T'においてlをkの子孫として
    - $edit_distance(T[left(i)..j], T'[left(k)..l])$ をDPで計算する