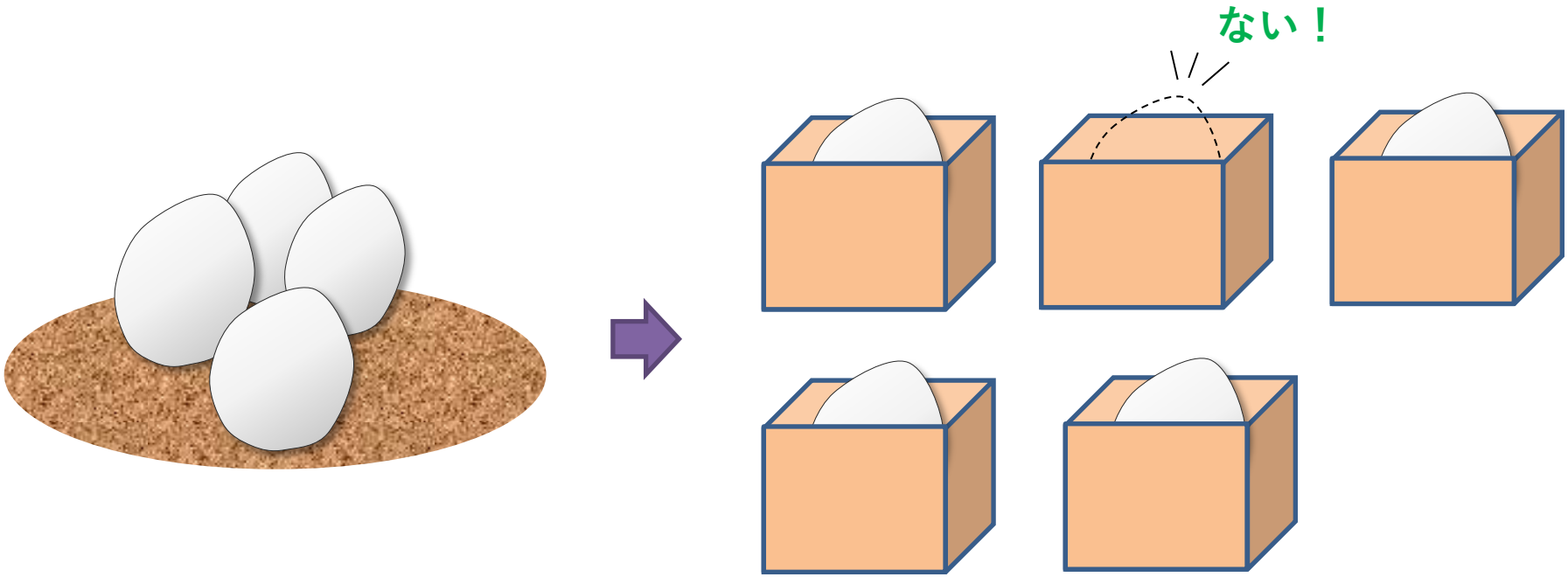
あいまい文字列検索

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

「鳩ノ巣原理　(Pigeonhole Principle)」

- 鳩がn羽、巣がm個　→
  - $[n/m]$羽以上いる巣が必ず1つある
  - $[n/m]$羽以下しかいない巣が必ず1つある
- 例
  - 鳩が6羽、巣が5つ　→　2羽以上いる巣が必ず1つある
  - 鳩が4羽、巣が5つ　→　空の巣が必ず1つある
  - 鳩が115羽、巣が20　→　5羽以下しかいない巣が必ず1つある



鳩ノ巣原理を用いたフィルタリング

- 一番簡単なテクニック [Navarro & Baeza-Yates '99]
  - クエリーを$k+1$分割
  - それらが一つでもexact matchingしている場所を探す
    - ハッシュ・接尾辞木・接尾辞配列等なんでも使ってよい
  - その周辺をチェック

テキスト

パタン

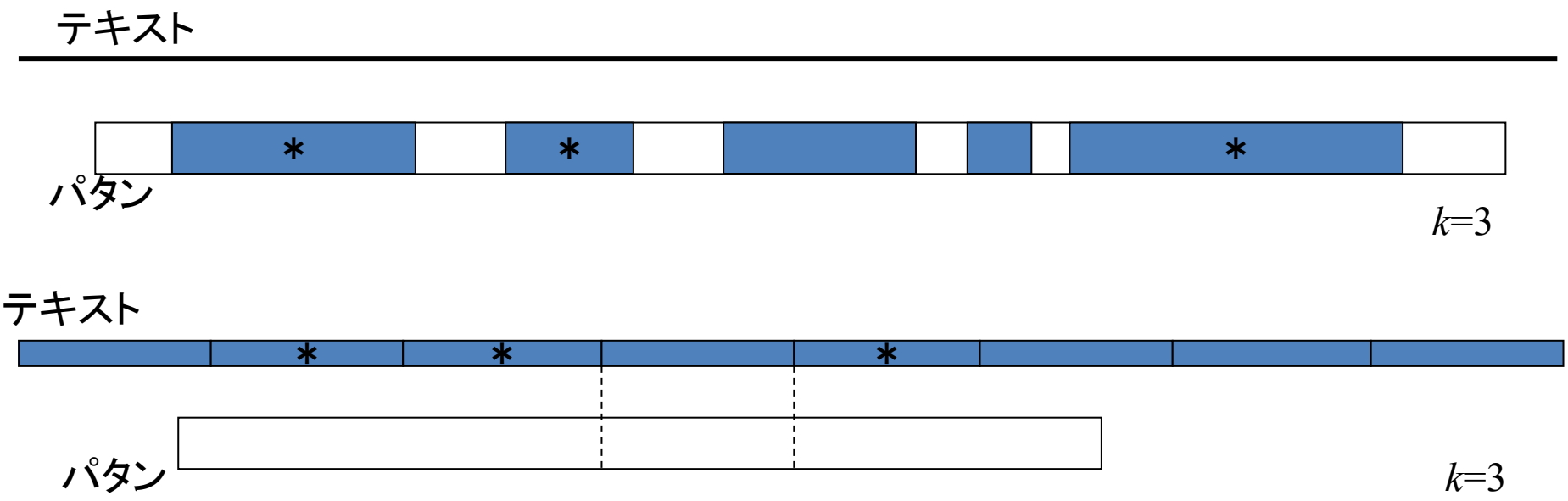
$k=3$であれば4分割

↓

必ず一つは完全マッチングするはず

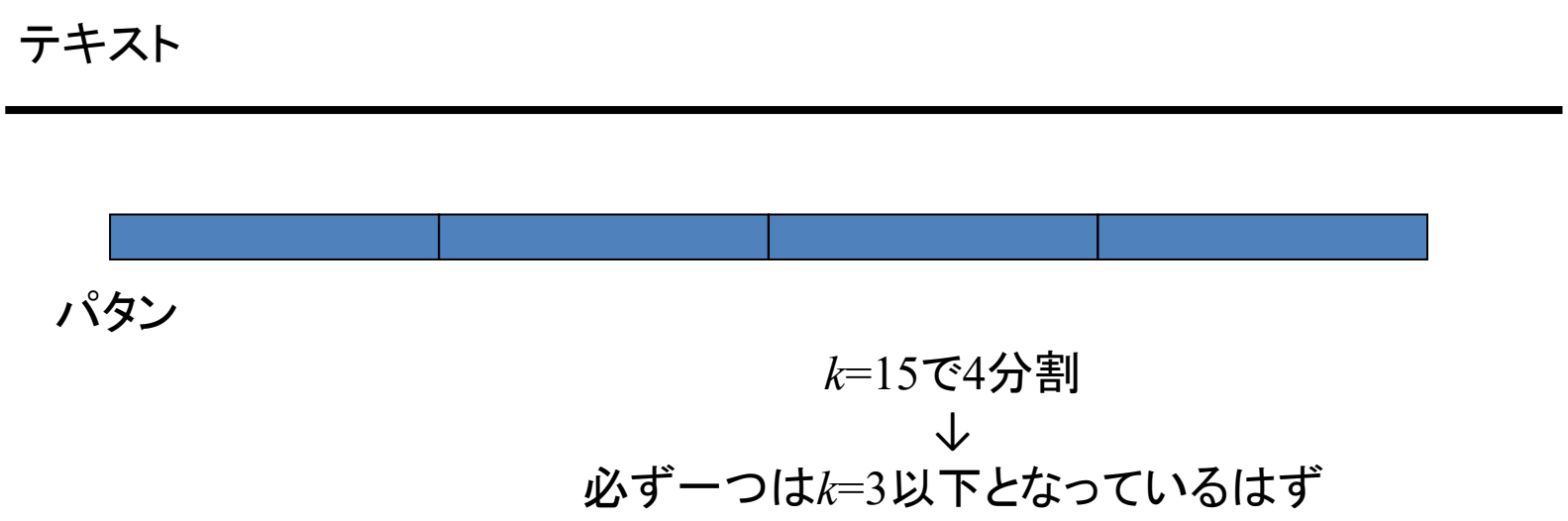
そのバリエーション

- $k+s$箇所の重なりのない部分文字列を検索し、s個のexact matchが存在するかをチェックすることも可能
  - ハッシュ等を用いる場合、分割長がクエリーによって変化するのは困る＝サイズは一定にすることが可能
- 逆にテキストを適当な長さで分割し、パタン側にそれらがあるかを見る、ということも可能



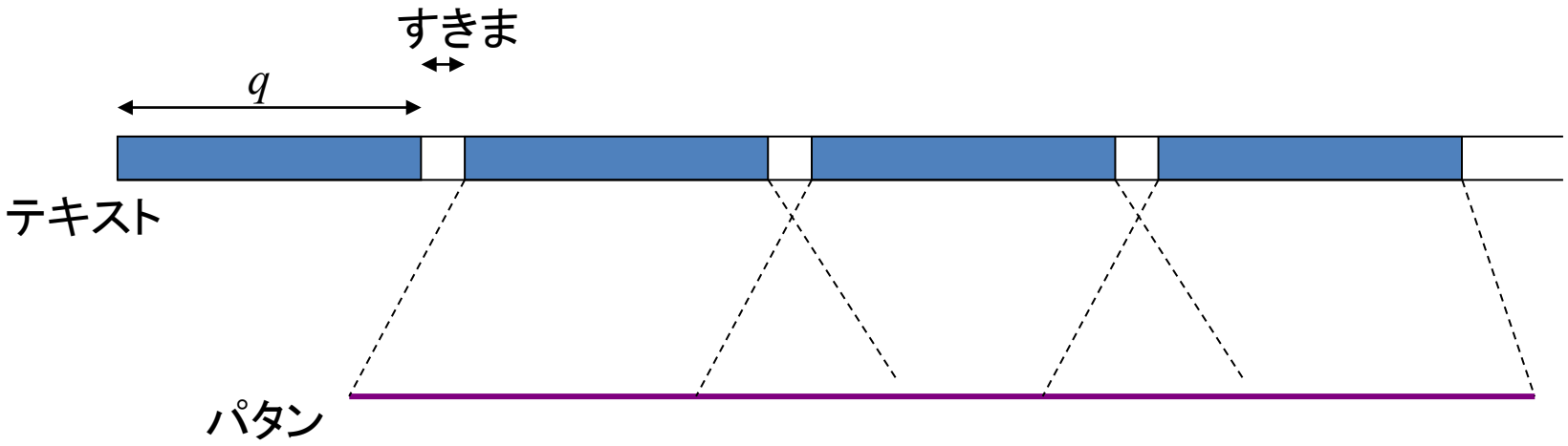
Navarro & Baeza-Yates '01

- 2つの方法の中間の手法
  - j 分割するとedit distanceが k/j 以下のものが一つは存在するはず
  - それを探すのに、suffix tree（array）を用いる
    - k が小さければ、実用に耐える



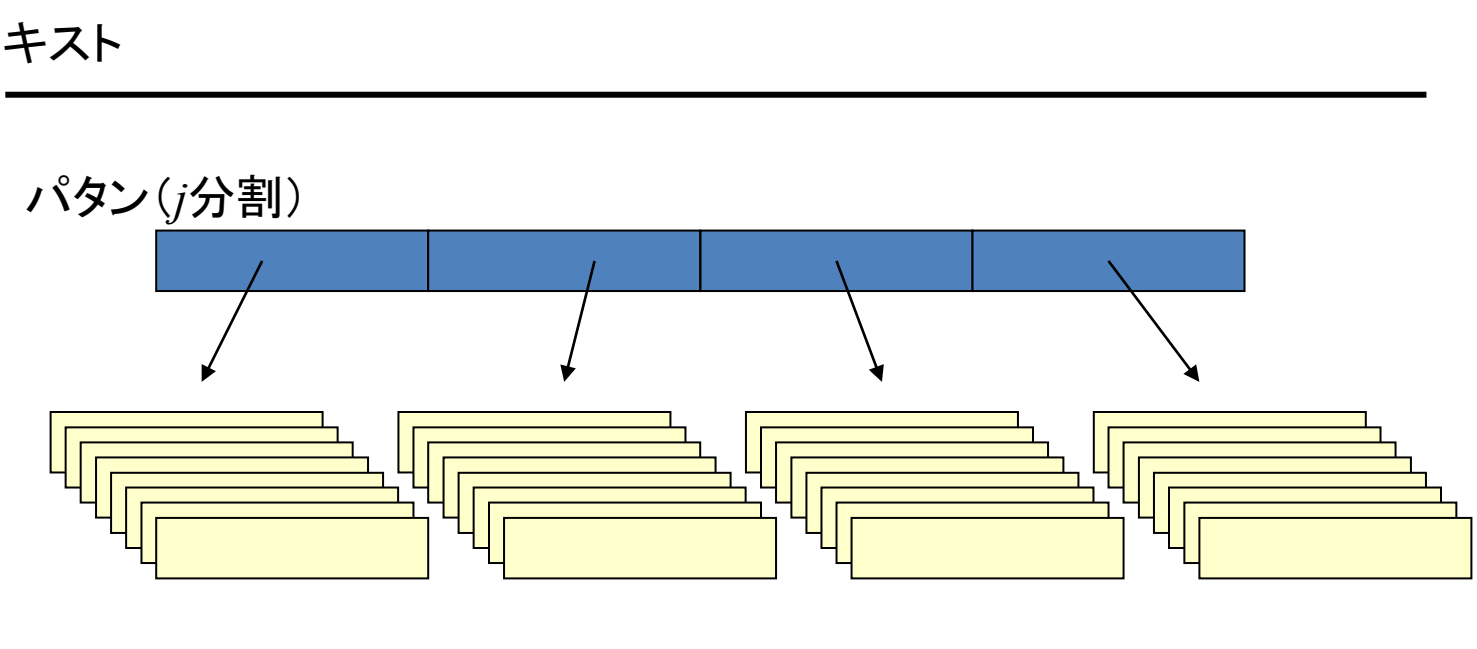
Navarro-Sutinen-Tarhio '98

- 逆にテキスト側を長さqの部分文字列に分解する
  - q−gram: 文字列の長さqの部分文字列全体の集合
  - q−sample: そのサンプル
  - q−sampleのtrieを作成し、その上でsuffix tree上でのDPと同様の計算を行う



Myers '94

- パタンを分割した後、可能性のあるものを全部列挙
  - アルファベットサイズが大きいと難しい

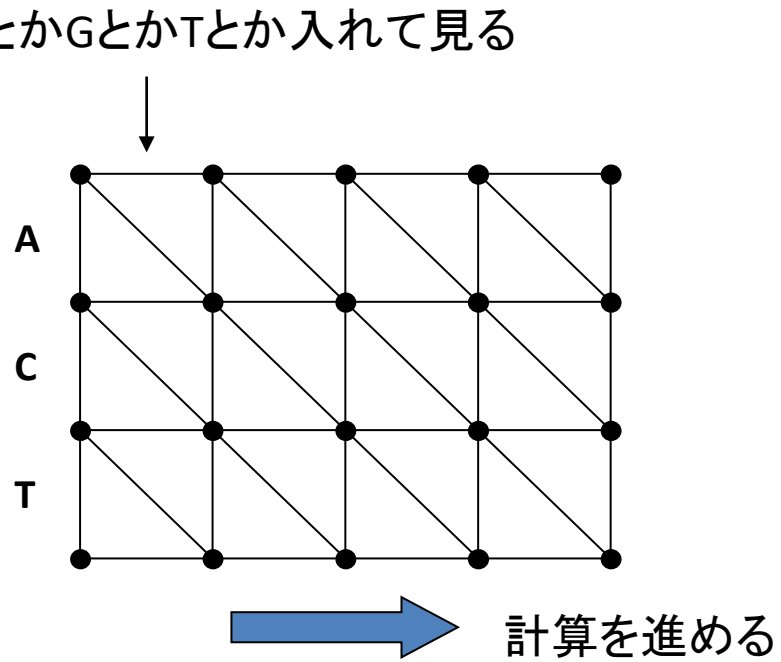


edit distanceがk/j以下の
maximalなものをすべて生成
↓
これを検索する

全近傍列挙の方法 （1）

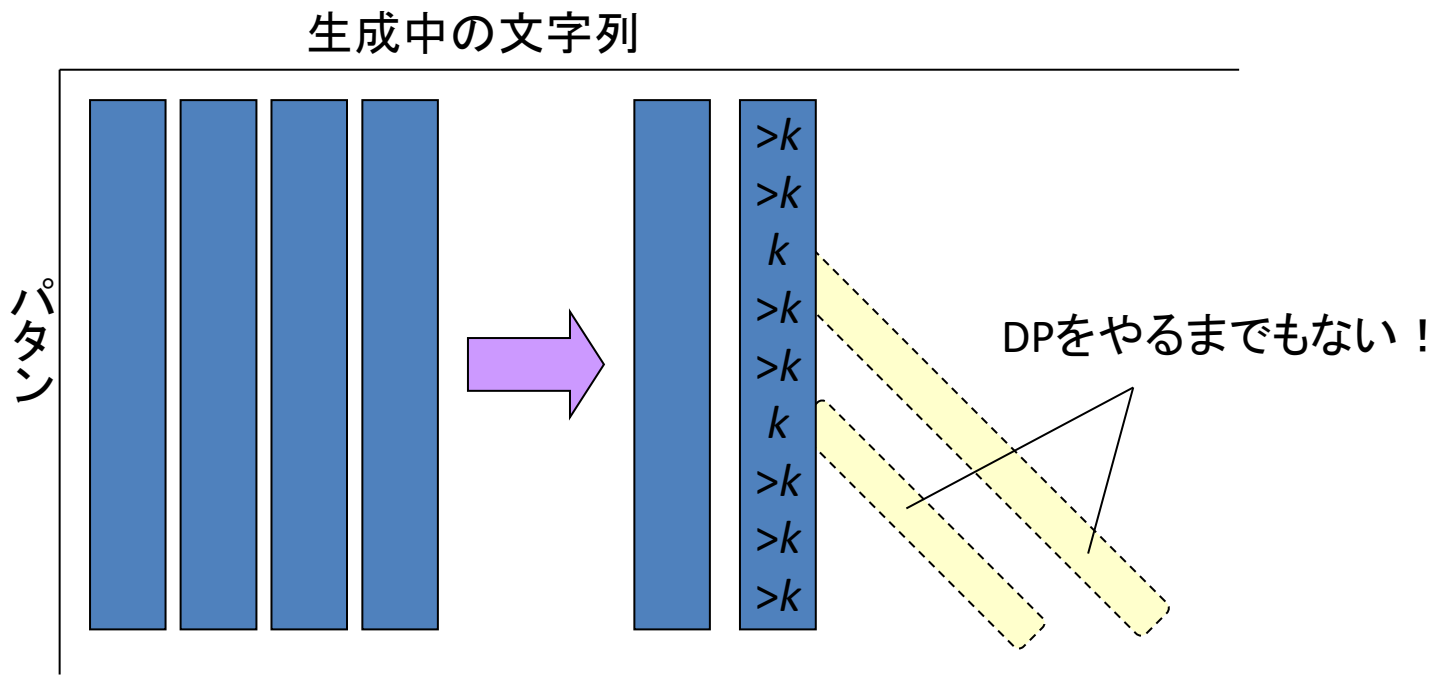
ナイーブな手法

- DPをやりながら文字をいろいろ生成する
  - 接尾辞木を辿るのと同じ感覚で
  - もうだめならばそこで枝を刈る
- $O(s \cdot m^2 \cdot t)$
  - s: アルファベットサイズ、m: パタンの長さ、t: 全近傍のサイズ



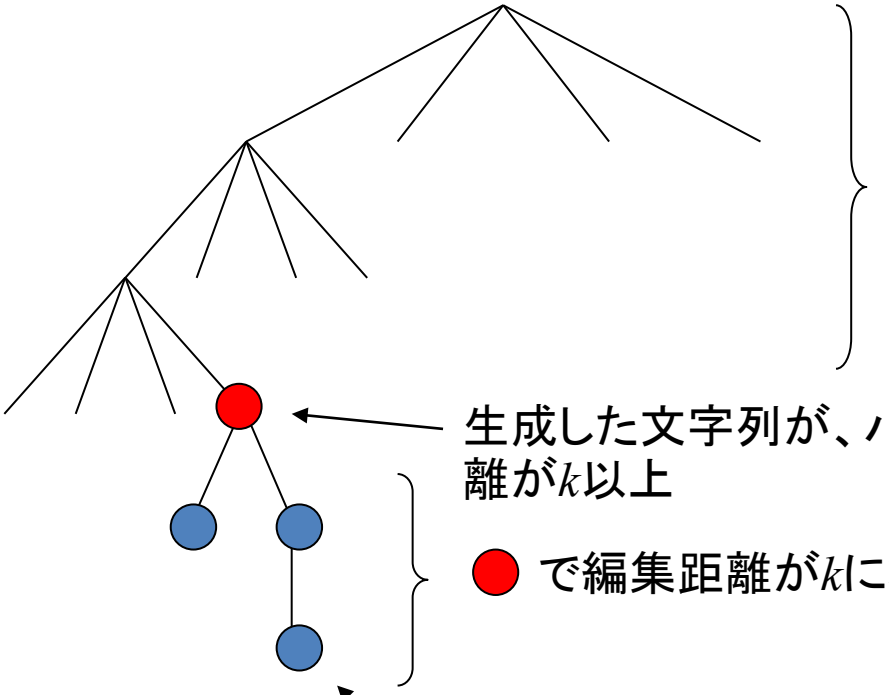
全近傍列挙の方法 （2）

- DPの列がすべて k 以上になったら、その後の計算は簡略化できる



全近傍列挙の方法 （3）

- DPの列がすべてk以上の値になった前と後の生成木の状況



編集距離がk未満の間は必ずすべて文字を生成している（枝分かれがアルファベットサイズとなっている）
＝DPの段数は全体で　O（解の個数）

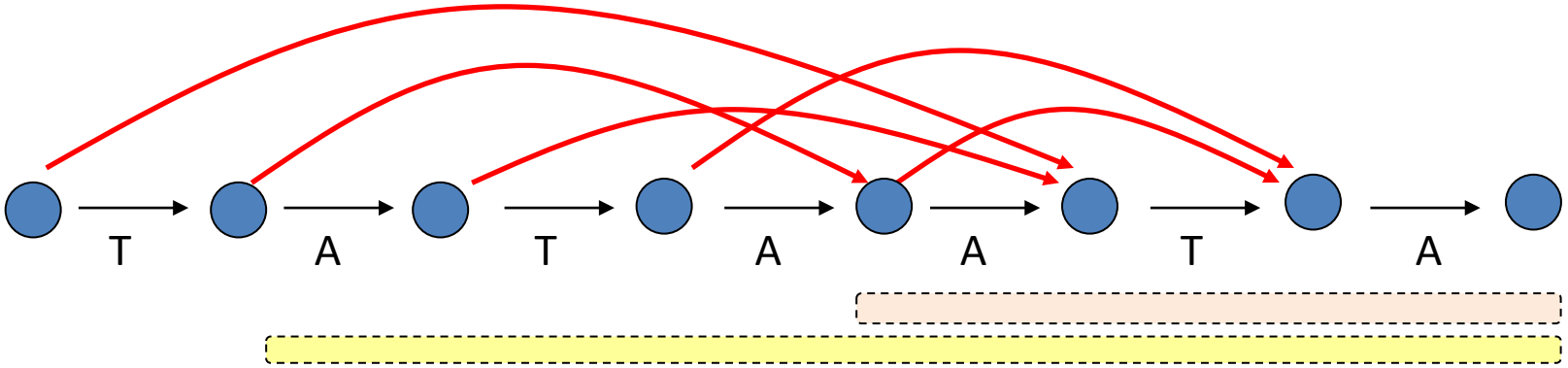
生成した文字列が、パタンのどの接頭辞に対しても、編集距離がk以上

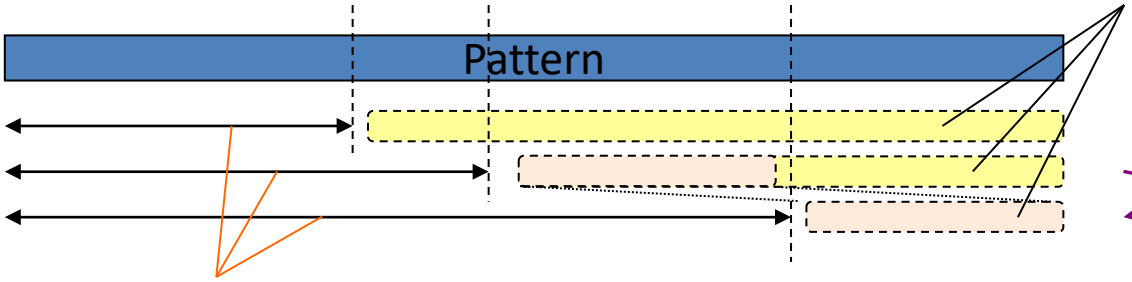
● で編集距離がkになっているところに対応する解が存在

別の解を接頭辞として含む場合は出力する必要はない

全近傍列挙の方法 （4）

- 接頭辞として含まれるかどうかの判定は？
  - 逆順の文字列に対するKnuth-Morris-PrattのFailure Linkを作成すると、それを利用可能





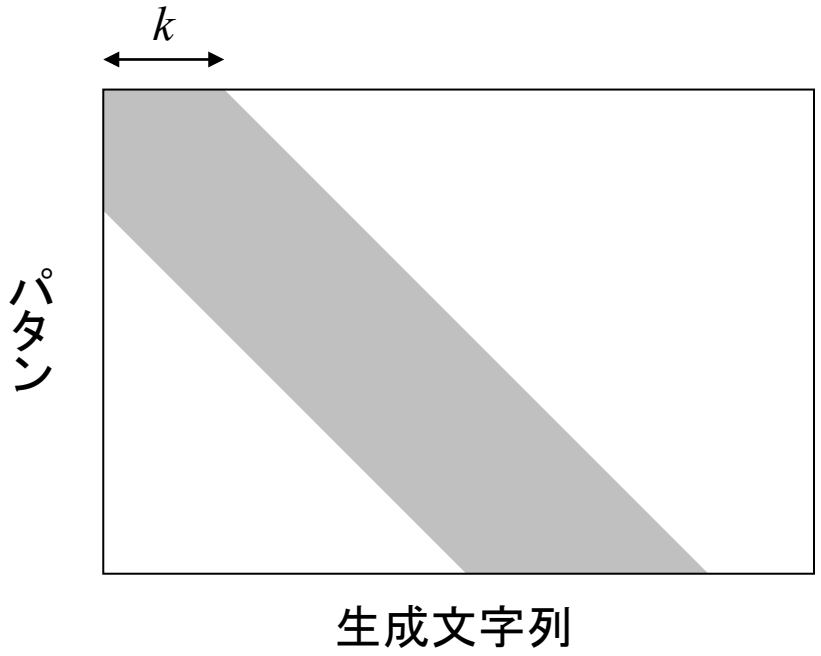
後はパタンの接尾辞を加えるだけなのだが、、、

完全に接頭辞として含むようであれば長い方を無視

パタンのこれらの部分のいずれもが
そこまで作成した文字列と編集距離＝k

全近傍列挙の方法 （5）

- 高速化の手法その2
  - DPは当然 $O(k)$ の幅だけの計算でよい
- よって全体の計算量は最終的に $O(kt+m)$
  - k: 編集距離、t: 近傍の大きさ、m: パタンサイズ
  - ただし、すべて明示的に近傍をすべて出力する場合には $O(kt + mt)$



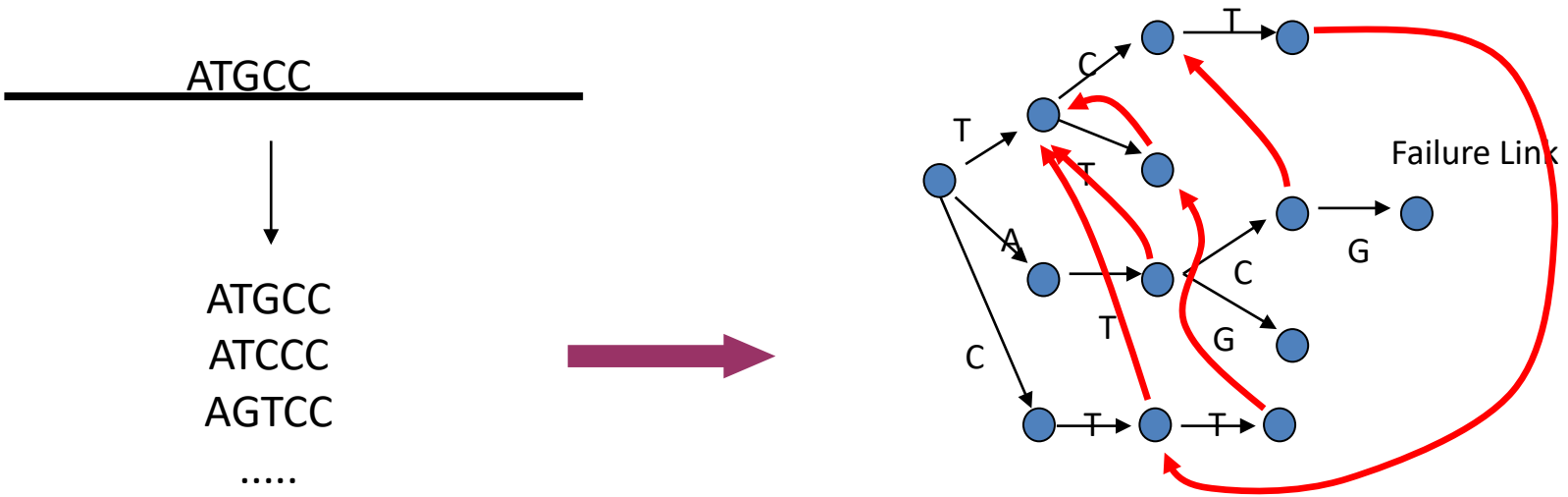
アラインメント・スコアを元に検索するには

- Inexact matchingの技法を使うのは困難
  - フィルタリングや接尾辞木等の技術を用いても、あるスコア以上のものをすべて列挙、といったことはかなり非効率
  - ヒューリスティックのアルゴリズムが活躍
    - 考え方自体は類似するところが多い
- BLAST, FASTA
  - 分子生物学者にとってのバイオインフォマティクスの代名詞
  - 類似配列をデータベースから検索するヒューリスティック
    - アラインメントのスコアが統計的に有意に高いものを出力する
    - BLASTは（基本的には）ギャップは考えない
  - 統計的評価を工夫
    - *P*–value, *E*–valueの表示

BLAST (1)

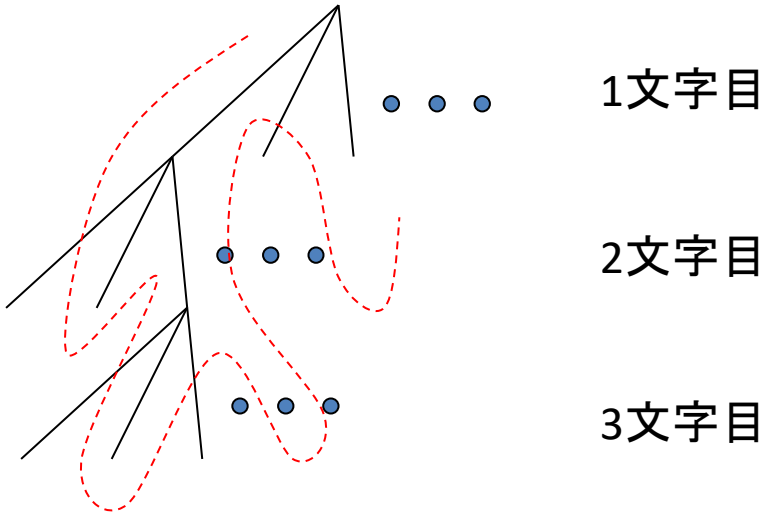
[Altschul et al., '90]

- BLAST=Basic Local Alignment Tool
- アルゴリズム
  - クエリーのすべての長さwの部分文字についてその近傍の文字列を列挙
  - それらの文字列を認識するAho-Corasickオートマトンを作成
  - データベースをそのオートマトンでチェックする
  - マッチした部分について前後をギャップは考慮せずにチェックし、高得点の部分を出力する



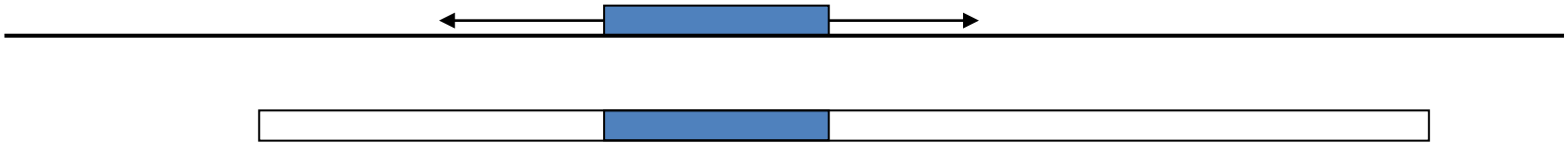
BLAST (2)

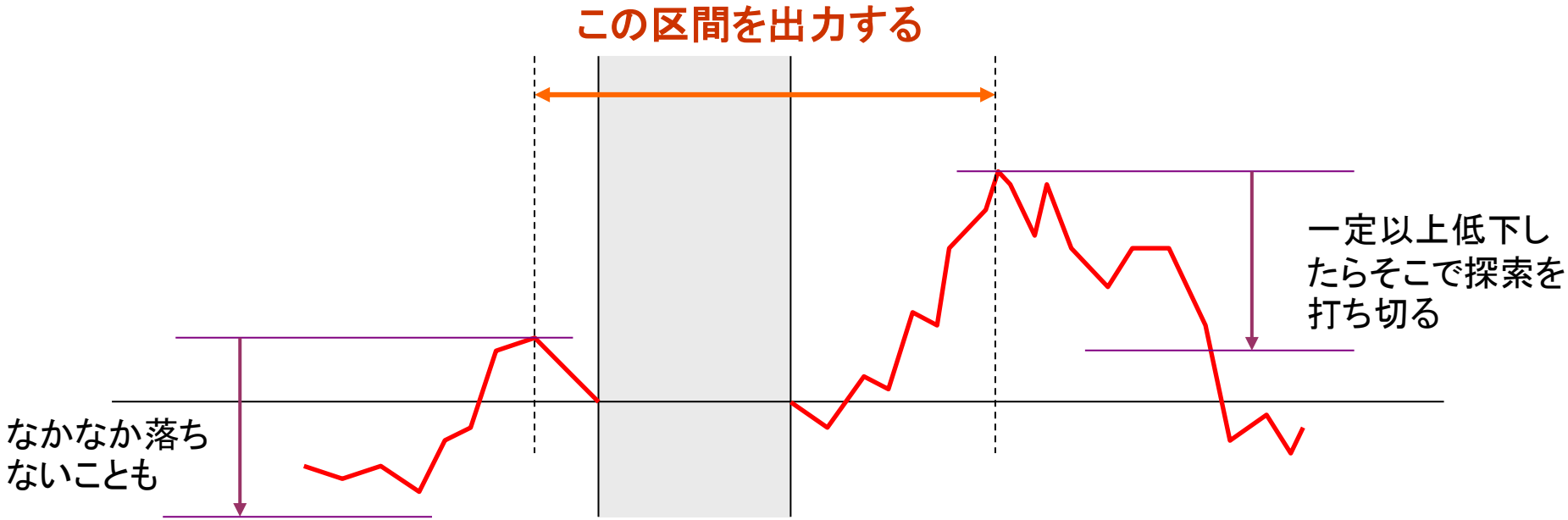
- 部分文字列の選び方
  - DNAではw=11前後、アミノ酸ではw=3前後が標準
- 列挙はギャップを考慮しないので、非常に簡単
  - 近傍サイズに対して線形時間で列挙可能
  - 文字列を出力するならば　$O(wt)$（t: 近傍サイズ）
    - 深さ優先探索（または幅優先探索）
  - スコアの小さい順に出力するならば$O(wt \log t)$
    - ヒープを用いる



BLAST (3)

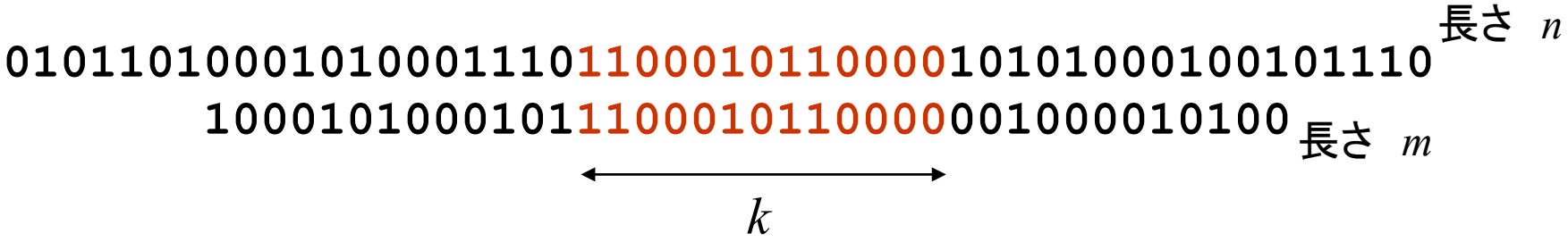
- マッチした部分を前後に延ばすには
  - ギャップをいれずにスコアが一番大きくなる点を探す
    - 両端まで探さず、それまでに見つかった最高点よりある一定値以上スコアが下がると打ち切るヒューリスティックを用いる





BLASTのE-valueの解析 （1）

- 非常に簡単化した問題
  - ランダムな2本の 01 列（長さ n, m）を考える
  - ある特定位置から長さ k の一致がある確率
    - $1/2^k$
  - 長さ k の共通部分 01 列の存在の数の期待値
    - $(n-k+1)(m-k+1)/2^k$
    - E-valueという
  - 二つの間に長さ k 以上の共通部分01列が存在する確率
    - $1-(1-1/2^k)^{(n-k+1)(m-k+1)}$
      - となり同士が独立だと仮定
    - P-valueという

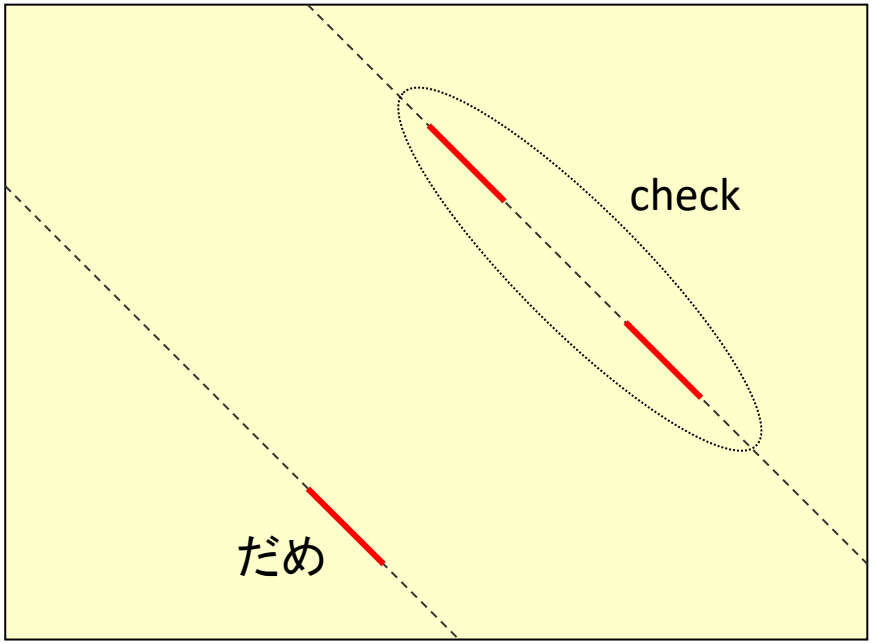


BLASTのE-valueの解析（2）　[Karlin, Altschul, '90]

- BLASTのE−value
  - （ギャップなしで）スコアがS以上の部分文字列のペアの数の期待値
  - $E = k \cdot m \cdot n \cdot e^{-\lambda S}$
    - m: クエリー長、n: データベースサイズ
    - k:　定数
      - 解析的な上限・下限もある程度与えられるが、ランダム文字列に対する実験で計算するのがよい
    - λ は次の式を満たす（スコアテーブルから数値計算で得る）
      - $\Sigma_{i,j} p_i \cdot p_j \cdot e^{-\lambda s_{ij}} = 1$
      - p_i ： i 番目の文字の出現確率
      - $S_{ij} = c \cdot \log (q_{ij} / p_i \cdot p_j)$:　スコアテーブル
        - q_{ij} :　i, j 番目の文字の同時出現確率　/ c: 適当な定数
- BlastのP−value
  - 偶然にそのスコア以上によいものが得られる確率
  - $P = 1 - e^{-E}$

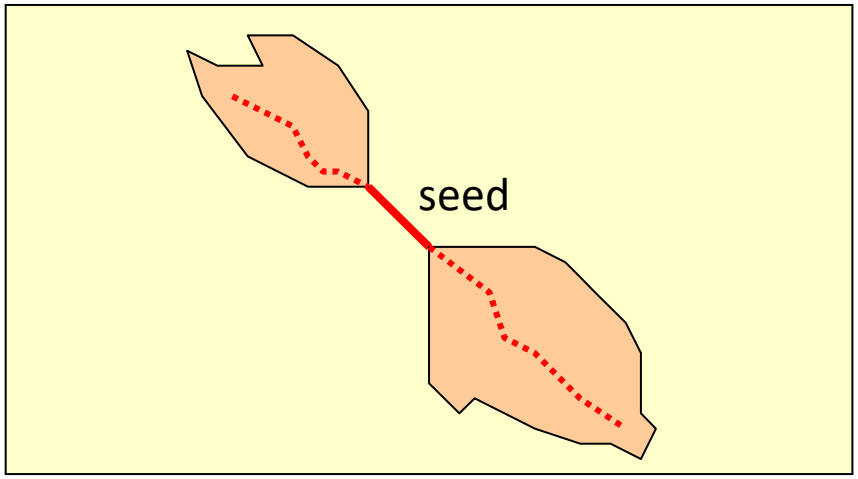
Two-hit method

- 位置関係が同じマッチが2つ以上あるもののみをチェックする
  - BLASTの計算ではextensionが9割の計算時間を占める



Gapped-BLAST

- アルゴリズム
  - 2-hit法で見つかった領域の中である長さ(適当な定数)の最も高い得点の領域を見つける
  - その周辺を通常のDPで計算する
- E-valueの解析は困難だが、BLASTと同様の挙動を示す
  - ランダム配列上でシミュレーション



ランダム配列上でのシミュレーションとは

- 入力配列and/orデータベース中の配列を適当にランダムにパーミュテーションさせる
- それに対して検索をかけて、いくつヒットするか数える
- 計算時間を減らしたい場合には、
  - データベース中からランダムにいくつかサンプリングしたものに対して同様の計算をする。
- ヒットが少ない時は、
  - クエリーを小さなものに分割してヒットの多いクエリーを作成し、その結果から予測する

（豆知識） ランダム・パーミュテーションの作り方

線形時間アルゴリズム

for ($i = 1 \sim n-1$)

{ i 番目の要素と ***i* 番目**から n 番目までの中からランダムに選んだ要素を交換する。

（$O(1)$時間）}

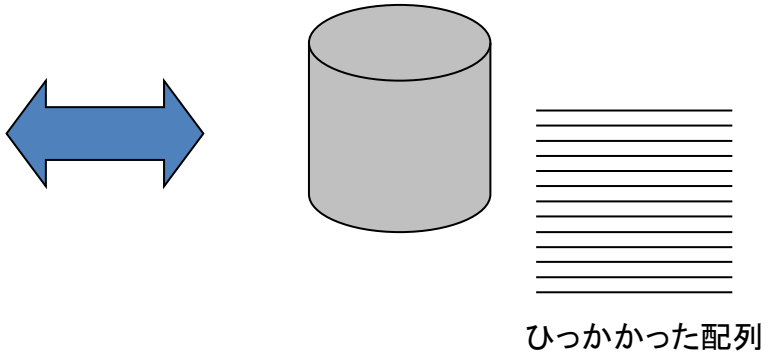
cf. 乱数＋ソート

- 乱数列をソートしてランクを計算
  - 生成した乱数列には同じ値は含まれないものと仮定
- しかし、この方法だと $O(n \log n)$時間かかってしまう
  - 用途によってはこちらの方法が有用なこともある
    - 複雑なランダムモデルやセキュリティ用途など

PSI-BLAST

- プロファイル（Weight matrix）を考慮した検索アルゴリズム
- PSI = Position Specific Iterated (= weight matrix) BLASTアルゴリズム
  - まずは普通にBLASTする
    - たくさんひっかかる
  - ひっかかった配列からプロファイルを作る
  - プロファイルでデータベースをBLASTする
  - ひっかかる配列が収束するまで繰り返し
- E−value等はシミュレーションによる

	1	2	3	4
A	0.3	0.1	0.05	0.25
T	0.1	0.7	0.2	0.25
C	0.2	0.1	0.7	0.4
G	0.4	0.1	0.05	0.1



何故こんなことをするか？

- 配列中には、機能をつかさどる部分が、特徴的な配列として表れている場合がある
- 単に検索しただけでは、機能のある領域もそうでない領域も、配列全体をメリハリなく検索することになってしまう
- そこで、類似配列をたくさんひっかけることによって、機能をつかさどる特徴的配列をいぶりだして、その集団をプロファイルで表現し、その特徴的配列を検索するようにすれば、重要な類似機能を持つような配列を検索することにつながる、と考えられている。

BLAT (Blast-Like Alignment Tool)

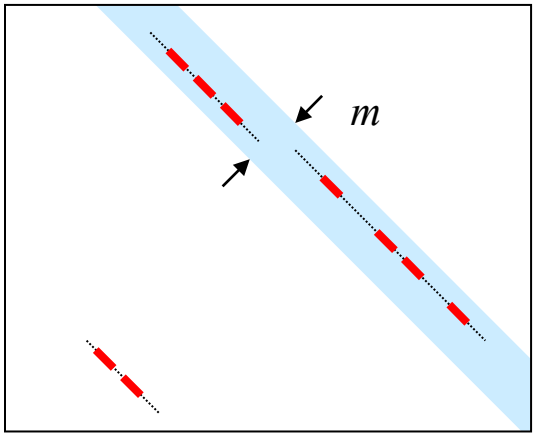
[Kent '02]

- BLASTはデータベースをなめていた＝非効率
  - データベース側でハッシュによるindexを持っておく
- アルゴリズム
  - データベース上でk−mer (k−gram)に対するハッシュを作成
  - それらがp個（任意に指定）出現する配列（あるいは領域）を選択
  - その周辺をアラインメント
    - BLASTより頑張って遠いマッチ同士をくっつける、といったこともできる

FASTA

[Pearson, Lipman, '98]

- アルゴリズム
  - 長さkの一致する部分配列を持つ配列をすべて探す
    - ハッシュを用いる（長さはタンパク質で1－3、DNAで4－6程度）
  - それらが「同じ位置関係」にあるものをグループ化し、よいものをn個（10程度）選ぶ（BLASTに類似）
  - つなげられそうなものはギャップペナルティを考慮しつなげることを考え、その周辺で幅m（32程度）でDPでよりきちんとしたアラインメントを計算する
- E−valueはシミュレーションで計算
  - 入力をシャッフルしてから、データベースのサブセットに検索をかける



BWA

- 近年のNGS（次世代シークエンサー）の特徴
  - 大量データ
  - １本１本は短い（100-200bp）
  - Illuminaの場合は、ギャップは少ない・エラーレートは１％以下
    - そうでないシーケンサーもある
- アルゴリズム
  - FM-indexで候補を絞る
    - matching statisticsを活用して、#errorの下限が十分小さい箇所のみを検査する
  - その周りをきっちり探索

FASTA/BLASTの改善（PatternHunter）

- 実は無駄な計算をしている！
  - それを避けることで、より少ない計算でより感度の高いアルゴリズムはできないか？　→　PatternHunter

[Ma, M., et al. 2002]

考え方

- 互いに重なりが少ないようなseedを選べば、より効率的になるのではないか？
  - BLAST同様ギャップは考えない

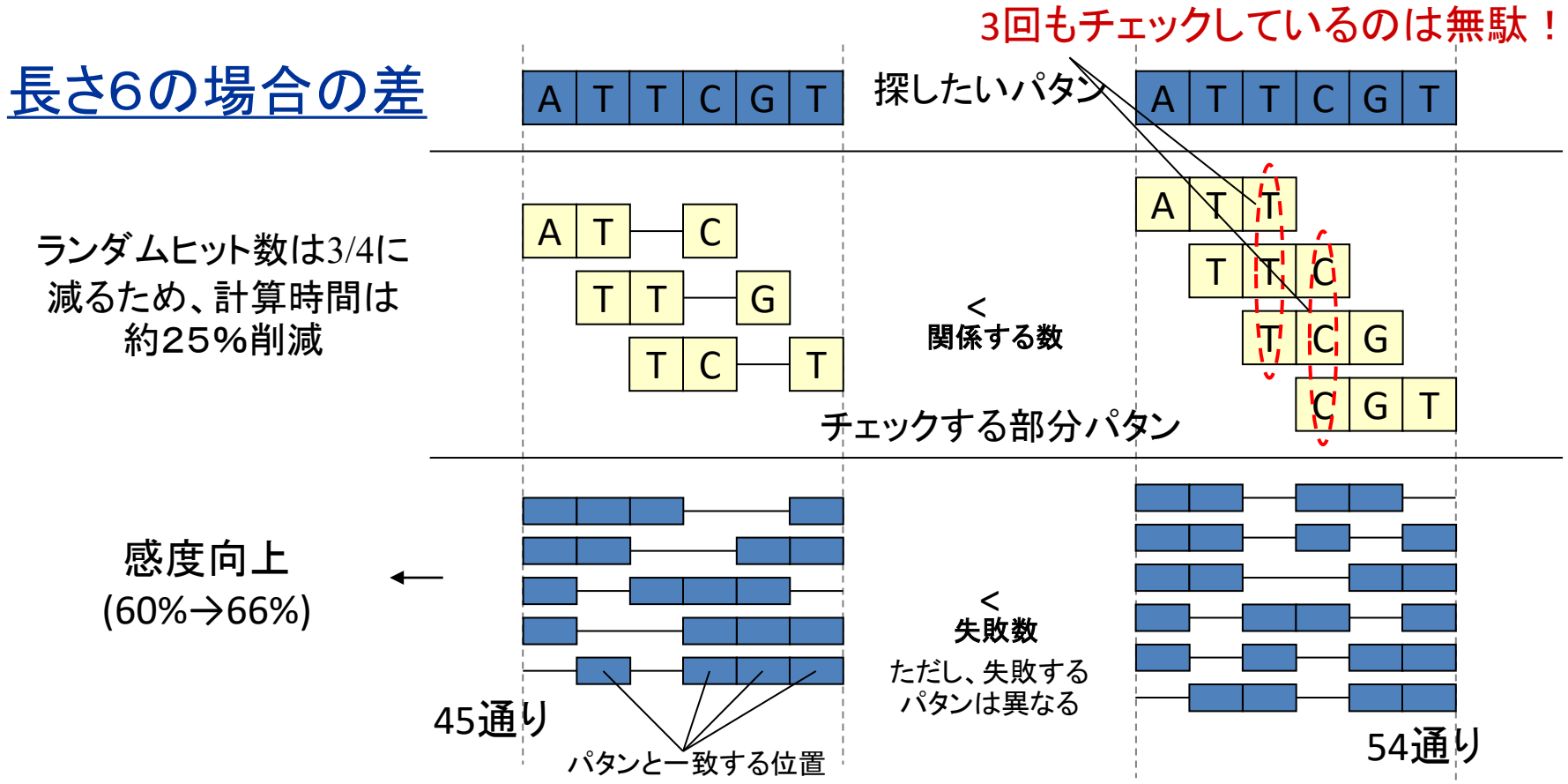
```
   ATGCCTGAAATATTCCTTAGGAT
---------------------------------------------------
   ATG.C..A.A..TT.CTT
    TGC.T..A.T..TC.TTA
     GCC.G..A.A..CC.TAG
      CCT.A..T.T..CT.AGG
       CTG.A..A.T..TT.GGA
        TGA.A..T.C..TA.GAT
```

重なりが少ない

重み11のパタン

例

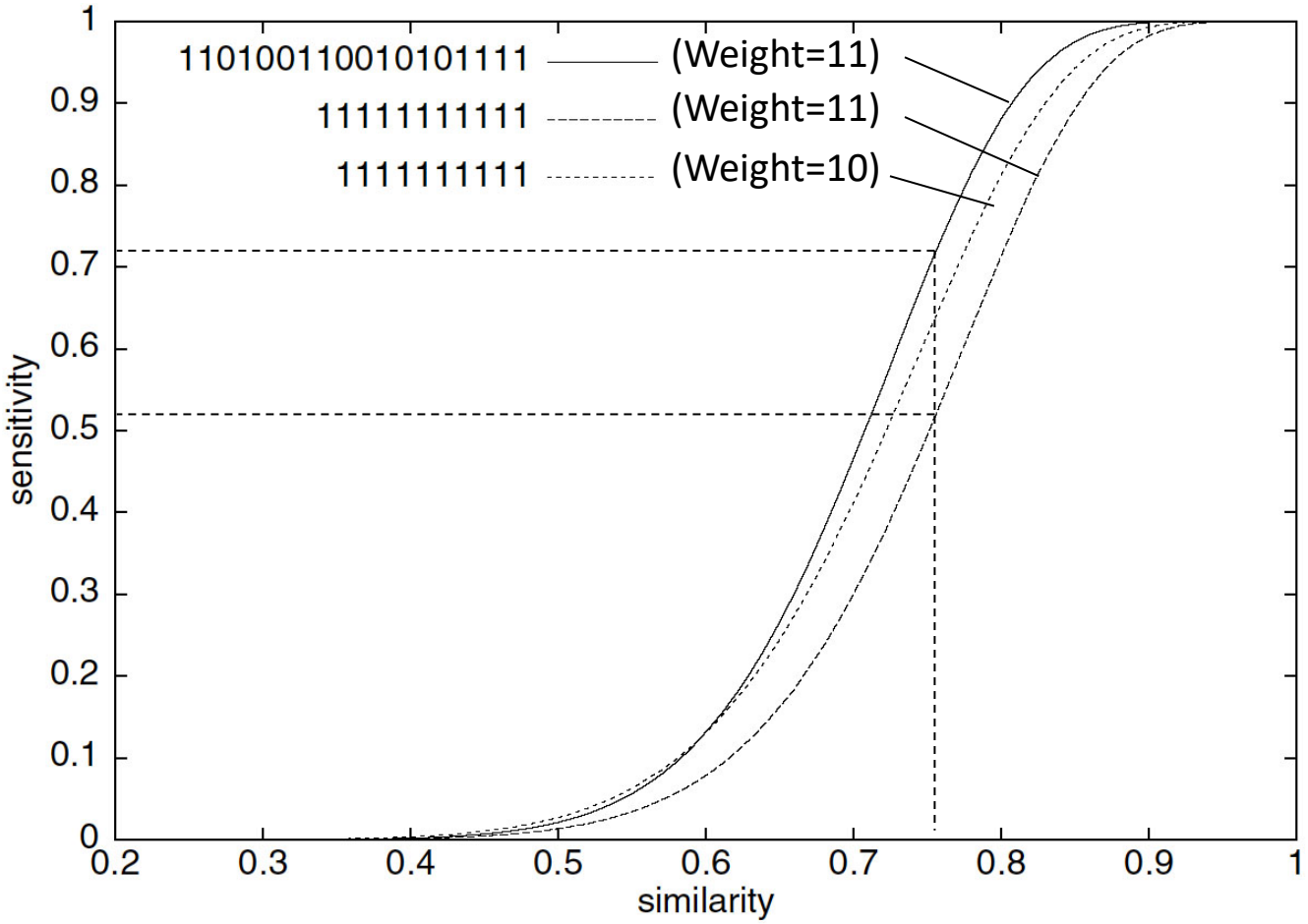
- ○○×○というパタン（重み3）でフィルタリングを行い、長さ6で（ギャップなしで）編集距離2以下のものを探したい
  - 編集距離が2のものは $3^2 \cdot {}_6C_2 = 135$ 通り



注）　この例では、編集距離0、1のもの（19通り）はどちらの方法でもすべて見つかる

シミュレーション結果

- もっと大きくしてもやはり感度が高い



Taken from [Ma, et al, '02] Bioinformatics, Vol. 18(3), pp. 440-445.

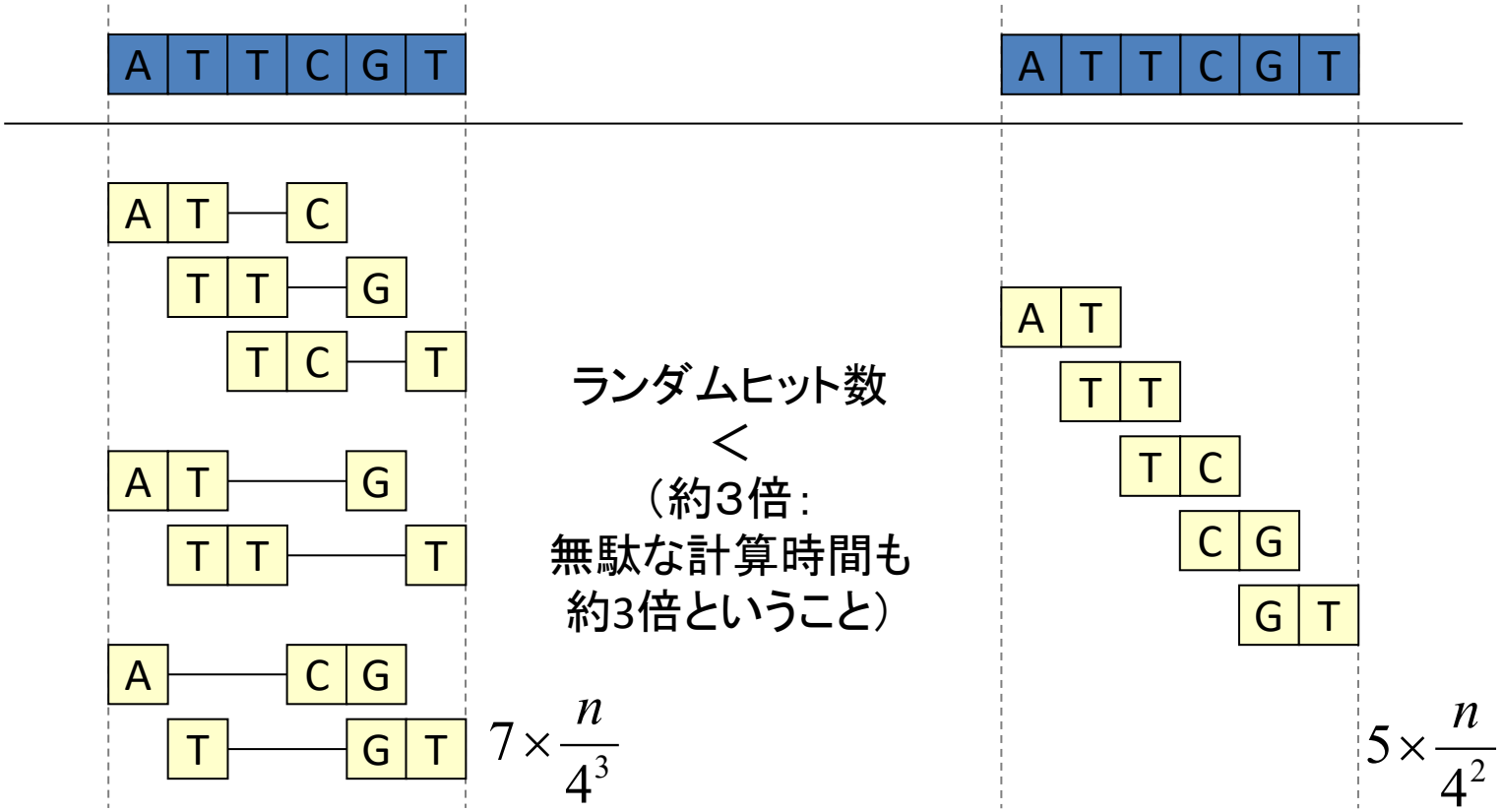
- 異なる形のseedを複数作成して感度を上げる
  - 感度を上げるにはseedを小さくすればよいが、一文字減らすと、ヒット数がアルファベットサイズ倍になることが予想されるため、それよりも異なる形のseedでチェックする方がよい

```
111010010100110111
1111001100101000010111
1101000011000101011111
1110111010001111
```

できるだけ異なる
＝類似度の低いパタン

複数モデルの例

- 長さ6、編集距離2を100%網羅するseedの比較



3種類の重み3のseedで網羅できる

連続seedで網羅するには長さ
2以下にする必要がある

解きたい問題

- 本当に解きたい問題
  - ランダムに発生させた配列 Q とスコアがある一定以上な配列をヒットする（理論的）確率が最も高い重み w、長さがm 以下の k 個のseedのセットを作成する
    - ランダム、といっても、理想的には生物学的にふさわしいモデルに基づいてランダム、がよい。アミノ酸では20種類の文字の出現確率は明らかに異なる。
- 少し簡単な問題
  - 特定の配列に対して同じ問題を解く
- もう少し簡単そうに見える問題
  - いくつかseedを準備しておき、その中から特定の配列に対してよさそうな k 個のseedを選択する

cf. MAXIMUM COVERAGE

- 問題
  - 入力：集合 S とその部分集合 $S_1,S_2,...,S_n$、$k<n$ なる k
  - 出力：できるだけ多くのitemを k 個の部分集合でカバーする
    - 各部分集合はあるseedによってカバーされる配列の集合に相当する
- 難しさ
  - NP-hard
  - グリーディーで作成すると $1-e^{-1}$ (0.632) 近似
    - これはタイト
- 要するに
  - そこそこのものは作ることもできる（かもしれない）が、最適なものを作るのは難しい
  - というわけでヒューリスティックやシミュレーションを用いてデザインすることになる。

実はそれどころか、、、

- 与えられたseedのhit率を計算する問題
  - NP-hard
    - ただし、seedが小さければ、計算できることも多い
    - 0の数が $O(\log n)$ であれば多項式時間
    - PTASは存在
- というわけで一番よい単一のseedを求める問題
  - NP-hard
- 要するに非常に難しい
  - 様々なヒューリスティックも存在
    - TSPのヒューリスティック的な解法
      - 少しずつ変えていくHill climbing、あるいはそのバリエーション

どう生成するか？

- 単一の最適なseed
  - PatternHunterではしらみつぶしに見つけている →　特許
    - DPなどで最低限の効率化はしている
    - ある程度の大きさで一度計算すれば、後は一生それを使いまわすだけでよいので、特に問題はない。
- 複数のseedの場合は
  - さすがにしらみつぶしは無理
  - Greedy　（Maximum Coverage問題と同様）
    - 良いものを一つ作る
    - それに足して計算するための良いものを一つ作る
    - どんどん作っていく

Bio-Dictionary

[Rigoutsos, Floratos 1999]

- <*L*,*W*>−パタン
  - *W*のサイズの部分文字列に*L*個以上の塩基
  - 固定長
- 蛋白質データベースに頻出する<*L*,*W*>−パタン
  - 完全自動生成
  - SwissProt/TrEMBLに複数回出てくる<6,15>パタン
    - エントリー数：　約56,000,000パタン

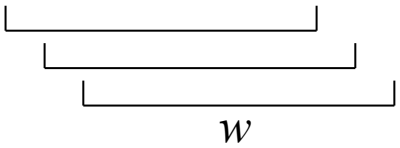
Proteins

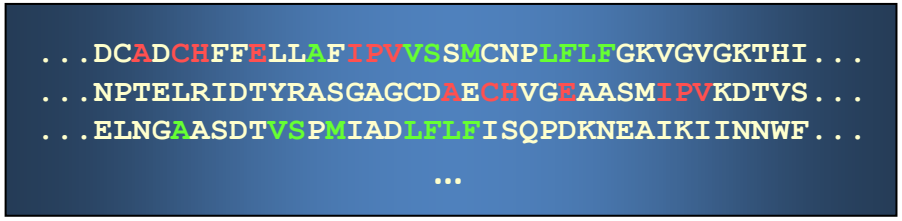
...DCADCHFFELLAFIPVV..

..KDAECHVGEAASMIPVK...

Pattern (Motif)

A.CH..E....IPV





Protein Database

Bio-Dictionary

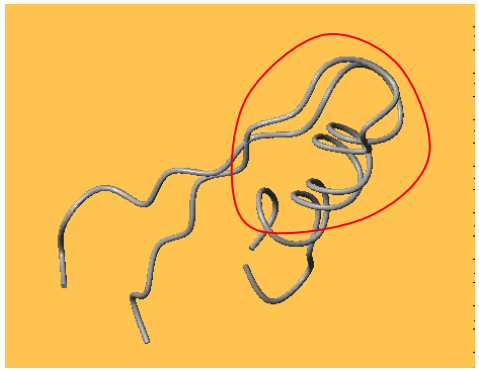
Bio-Dictionaryのパタンの例

#	***Pattern***	"Functional Meaning"
1	G..G.GK[STG]TL	ATP/GTP binding P-loop
2	H.....HRD.K..N	Ser/Thr-protein kinases
3	SGG[QEMRY]..R[VLIA].[IGLMV]R.L	ABC transporters
4	V.I.G.G..G...A	NAD/FAD-binding, Flavoproteins
6	G.GLGL.I	Sensory transd. His-prot. kinases
...		
10	GA.DY[LIV].KP	2-component sensory transduction
11	HR.GR..R....G	DEAD-box helicases
12	GDG[IVAMTD]ND[AILV][PEAS][AMV][LMIF]..A	Cation-transporting ATPases
13	D.FK.[IYVFL]N[DE].[YLFWR]GH..GD.[CLVF]L	Bacterial-type regulators*
14	DKT[GV]TLT	Cation-transporting ATPases
...		
16	KMSKS[LKDIR][GNDFQ]N	Amino-acyl-tRNA synthetases I
17	PTREL..Q	DEAD-box helicases
18	Q..GRAGR	DEAD-box helicases
19	F.[ASDN].[MIVTLA][SAT]HE[LIF]RTP	Sensory transd. His-prot. kinases
...		
27	DL[IVL][LIMVF]LD[ILVW].[ML]P..[DNST]G	2-component sensory transduction
...		
30	LD.GCG.G	Various methyltransferases
...		
34	T.[IVL][FLYMI]VTHD[QLIVP].[ELV]A	ABC transporters
...		
..		

立体構造との関連の例

V[IFLVC].G..G.G[KGC]T.L

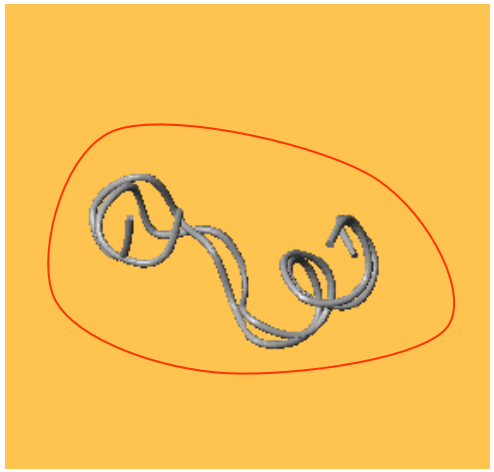
>1ayl
VFFGLSGTGKTTL
>1pox
VCFGSAGPGGTHL



error=2.192 Angstroms

A.AAG

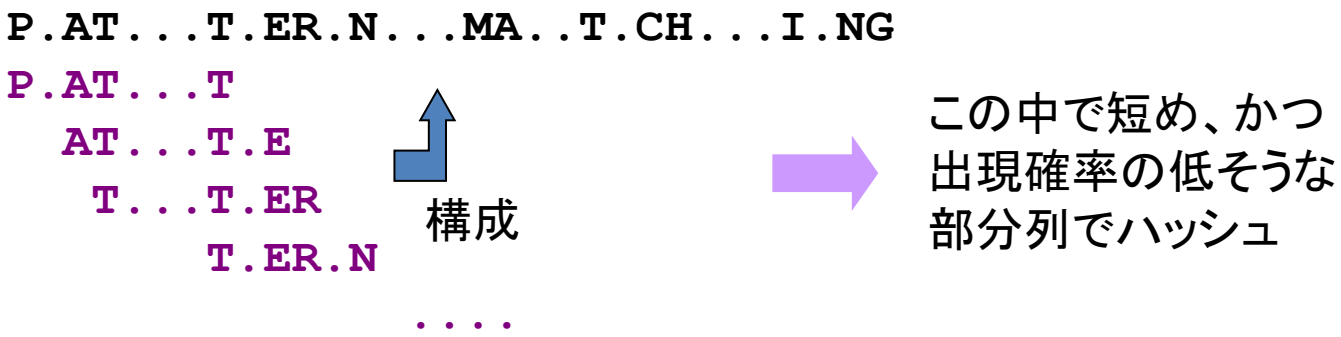
>1uag
ADAAGLPRASSLKAL
>1ttp
AFAAGVTPAQCFEML



error= 1.908 Angstroms

<L,W>パタン発見アルゴリズム

- Teiresias Algorithm
  - k個以上存在する$<l,w>$−pattern
    - すべて列挙
    - maximal pattern のみ
      - もっともspecific なもののみを出力
  - scanning phase
    - 短いパタンの生成
      - 少なくともK回現れるL固定文字のみを持つ$<L,W>$パタンの作成
  - convolution phase
    - パタンでつなげられるパタンを連結
      - *c.f.* アソシエーションルールの作成アルゴリズム



cf. アソシエーション・ルールの発見

- データベースにおいて相関関係を表すパタン
  - X→Y
    - サポート　　#(X,Y)
    - 確信度　　#(X,Y)/#(X)
  - 小さなルールから作っていく

↓**ビール→おむつ**　という関係があるらしい、等↓

客	買い物
Pさん	ビール、パン、おむつ、チーズ、桜餅
Qさん	小麦粉、ビール、おむつ、はさみ、のり、縄跳び
Rさん	おむつ、味噌、牛乳、パン、ペットフード
Sさん	セロリ、にんじん、おむつ、きゅうり、ビール、アボガド、パクチー
Tさん	哺乳瓶、乳母車、独楽、折り紙、かるた、三輪車

Bio-Dictionaryの検索

- クエリー
  - たんぱく質の配列
- 出力
  - マッチするパタン
    - 完全一致のみ
- 高速検索が必要
  - 巨大なDBサイズ
    - ～56,000,000 patterns

Bio-Dictionary

```
I.QVM..L....LR.A
E...MADI..PL
FA.EGK..LR.S.S
AER.AEI...EK
....
```

Query

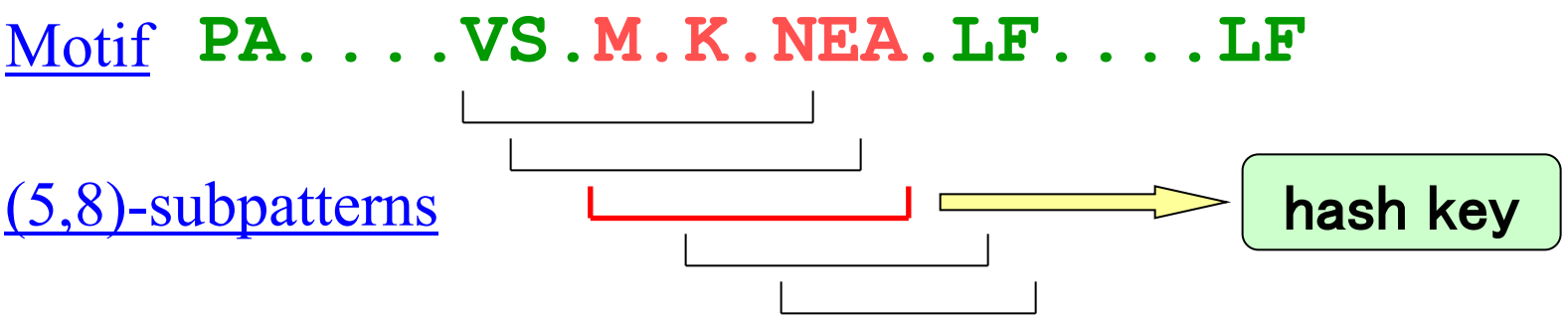
```
>○△◇_protein
ASIRTIQNWQEQGMPVLRGGGKGNEVL
YDSAAVIKWYAERDAEIENEK...
```

Output

```
A..RT...WQE.G
DS..V.KWY
AER.AEI..EK
...
```

Bio-Dictionary検索アルゴリズム

- (l, w)−subpattern を用いたハッシュ
  - 長さ w 以下で、高々 l 個の固定文字を含む部分パタン
  - ハッシュ・キー
    - クエリー配列中での出現頻度が最小の(l, w)−subpattern
    - l, wの最適化が必要　→　最適化することで数十倍高速に
  - 検索時間
    - 長さ1000のたんぱく質の解析：　0.1秒以下 (2GHz x 1CPU)
      - ナイーブな先頭数文字によるハッシュでは数分かかる
    - 他のモチーフDBと比べ遥かに高速



実際の検索例

(3,4)−subpatternによるハッシュの例

protein sequence

...FQITMIIVQCVTLYKLVL...

hash keys to check

I
IV
I.Q
IVQ
I..C
I.QC
IV.C

motifs to check

Motifs hashed by I

Q....I...V.....V
V...I....T...Y

Motifs hashed by I.Q

I.Q..LL
I...I.Q..T.Y

Motifs hashed by I.QC

AI.QC..LV
T..I.QCV
M.I.QC...YK

Check Results

○
×
×
○
×
○
○
・・・
…

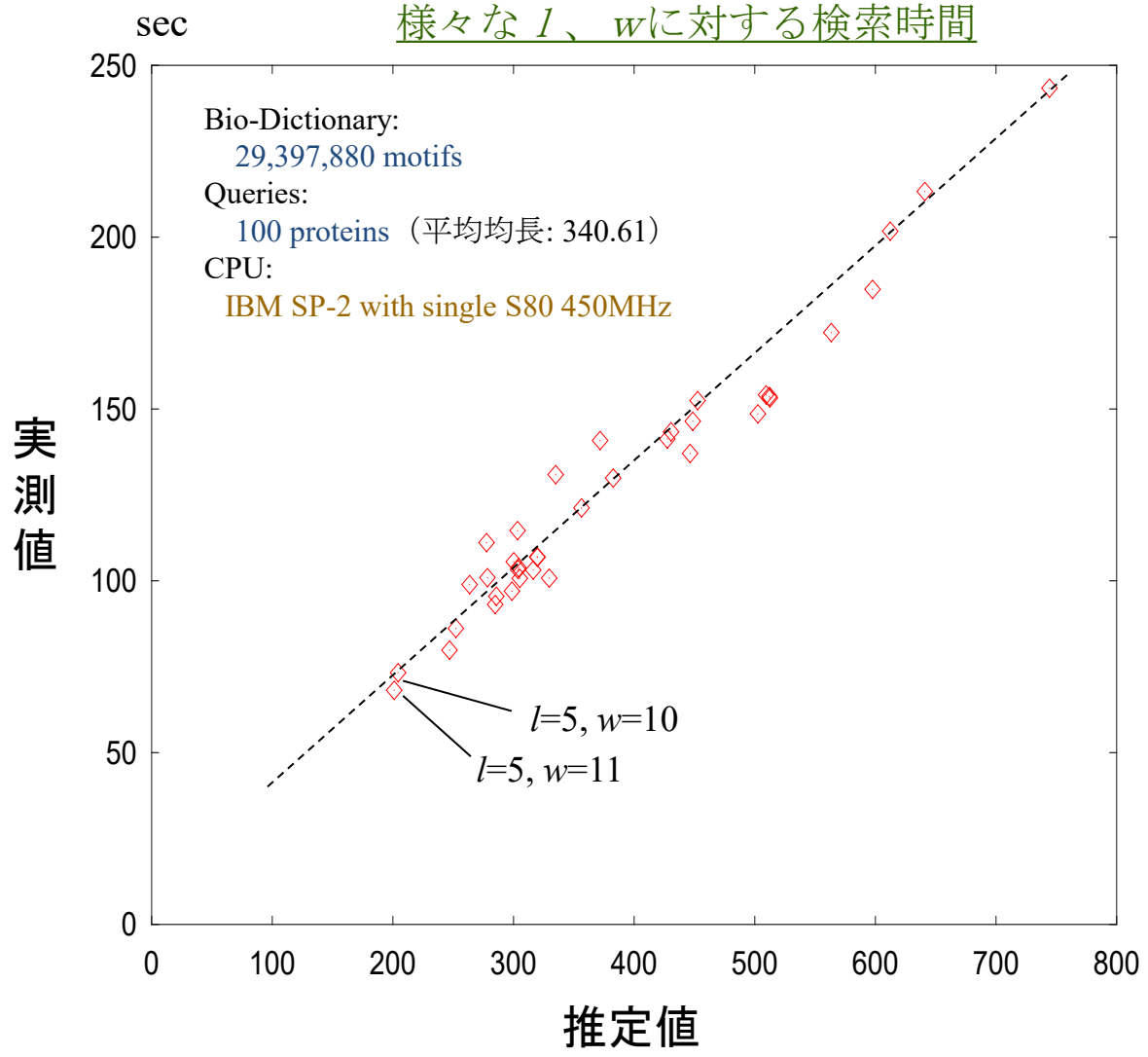
(3,4)パタンでハッシュできないパタンはより小さいパタンでハッシュしておく

前処理でデータベースに対して計算しておく

ハッシュの最適化

- *l*,*w*の最適化
  - 検索時間
    - 塩基頻度から推定可能
- この手法による高速化
  - *l*,*w*の最適化
    - 約10～30倍
  - 単純なPrefixによるハッシュと比較
    - 約4倍

様々な *l* 、 *w*に対する検索時間



[Shibuya, Rigoutsos, 2002]

まとめ

- 配列検索アルゴリズムのいろいろ
  - Navarro & Baeza-Yates algorithm
  - Myers algorithm
  - BLAST・Gapped BLAST・PSI BLAST・BLAT
  - FASTA
  - PatternHunter
  - BioDictionary
- 次回
  - 圧縮