プログラム設計とアルゴリズム
第14回 (1/17)

早稲田大学高等研究所　講師
福永津嵩

生命情報科学における
アルゴリズムとデータ構造

生命情報科学とは？

- 現在の科学は測定装置の進歩などによって得られる実験データが
  大規模化し、またそれらがデータベースとして蓄積されている

- これらのデータからどのような知を得るかが重要な課題(データサイエンス)

- 生命情報科学は、生物学におけるデータサイエンスであり、
  データサイエンスの中でも古い歴史を持つ

- 「得られた大規模データをどう処理するか」という情報工学的課題と、
  「データに基づいてどう生命現象をモデル化するか」という生命科学的課題

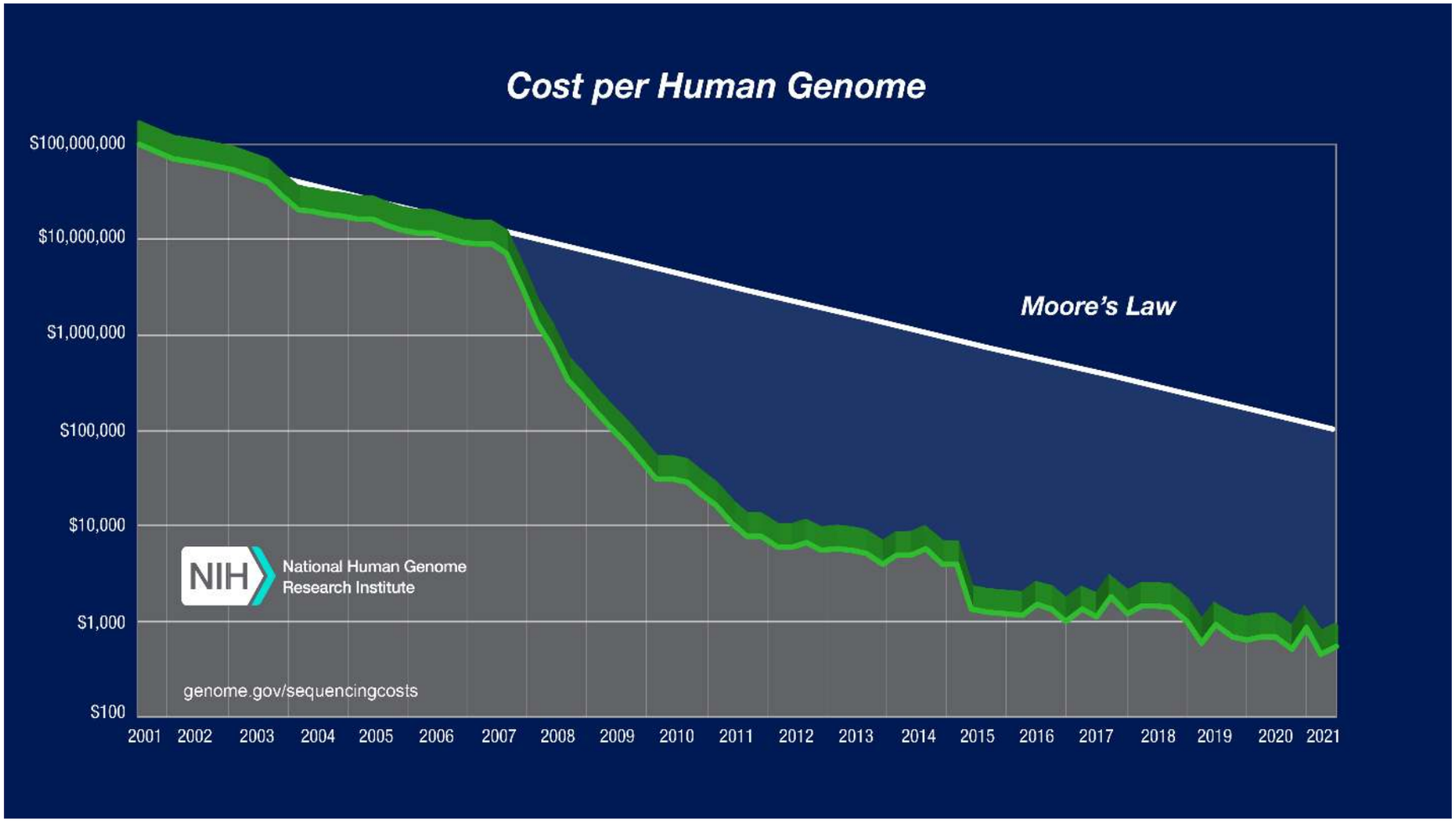
情報科学的な課題

- 大規模データの整理・検索 (データベース)
- 大規模データの解析 (アルゴリズム・機械学習)
- 高速化 (並列計算)

- 大規模データの種類
  - ゲノム配列 ・アミノ酸配列 (文字列データ)
  - 遺伝子発現量 (数値データ)
  - テキスト(論文)データ
  - 生体分子の相互作用ネットワーク　(グラフデータ)

  などなど

DNAシーケンサーの技術革命

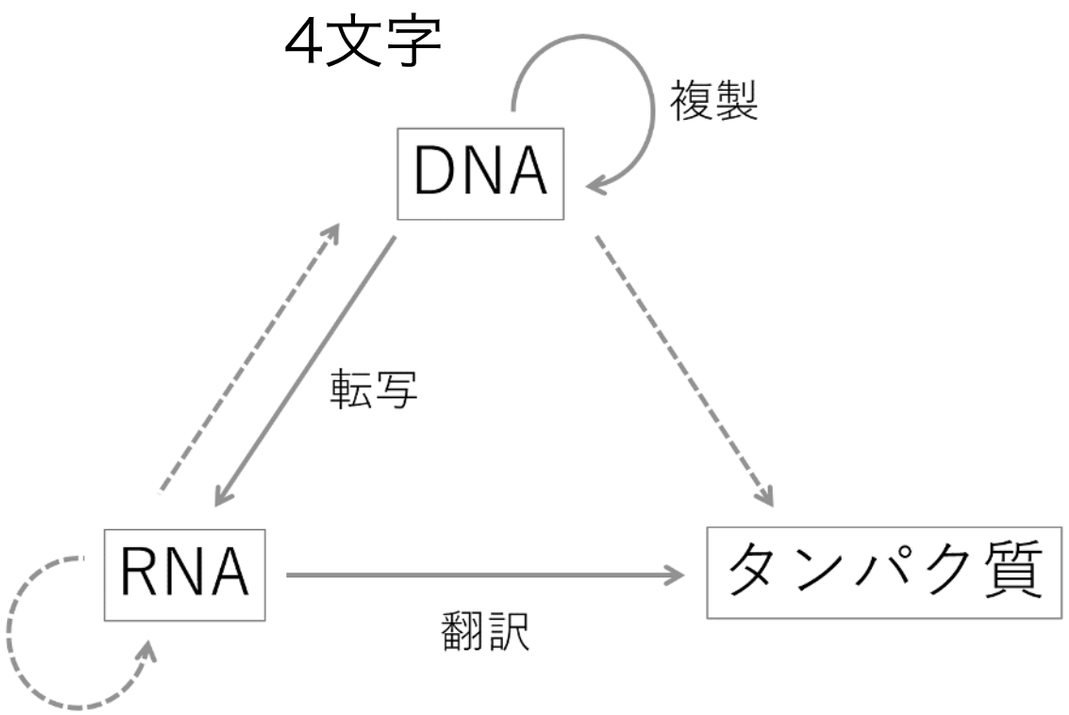


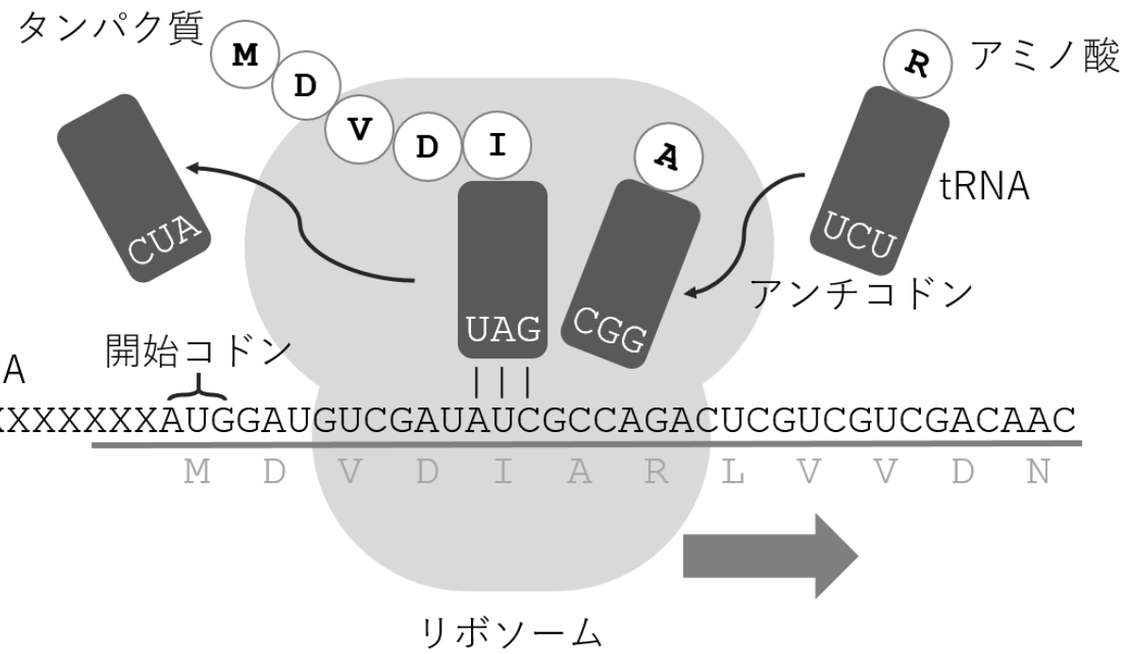
ヒトゲノム(全DNA)は
およそ30億文字の
ATCGからなる文字列

ゲノム解読のコストは
劇的に下がり、
スピードは劇的に
上がっている

現在では、ヒト1人の
ゲノムを10分1000$で
読む事が可能

分子生物学のセントラルドグマ





配列アライメント

- 配列解析において最も基本となる解析は、配列アライメントである
  これは、二本の配列の類似性を計算し、配列を整列する処理である

- 配列アライメントの基本は、編集処理の計算である

編集距離

2 つの文字列 S, T が与えられます．S に以下の 3 通りの操作を繰り返し施すことで T に変換したいものとします．そのような一連の操作のうち，操作回数の最小値を求めてください．なお，この最小値を S と T との編集距離とよびます．

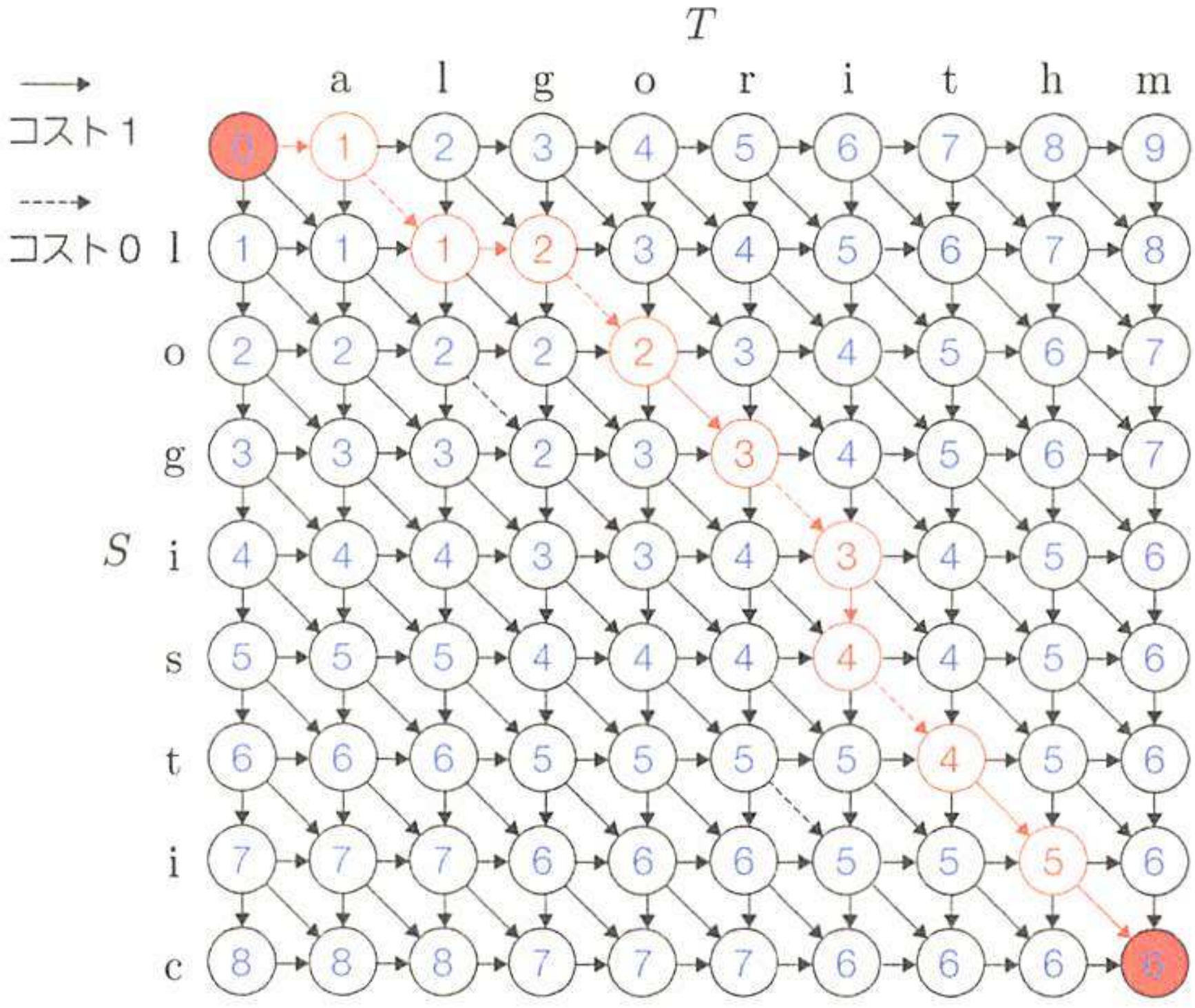
- 変更：S 中の文字を 1 つ選んで任意の文字に変更する
- 削除：S 中の文字を 1 つ選んで削除する
- 挿入：S の好きな箇所に好きな文字を 1 文字挿入する

- MAQRDAとMQRDAIでは、

```
MAQRDA– 挿入
 |X||||X
M–QRDAI
 削除
```

のため、編集距離は2となる

(復習)動的計画法:編集距離

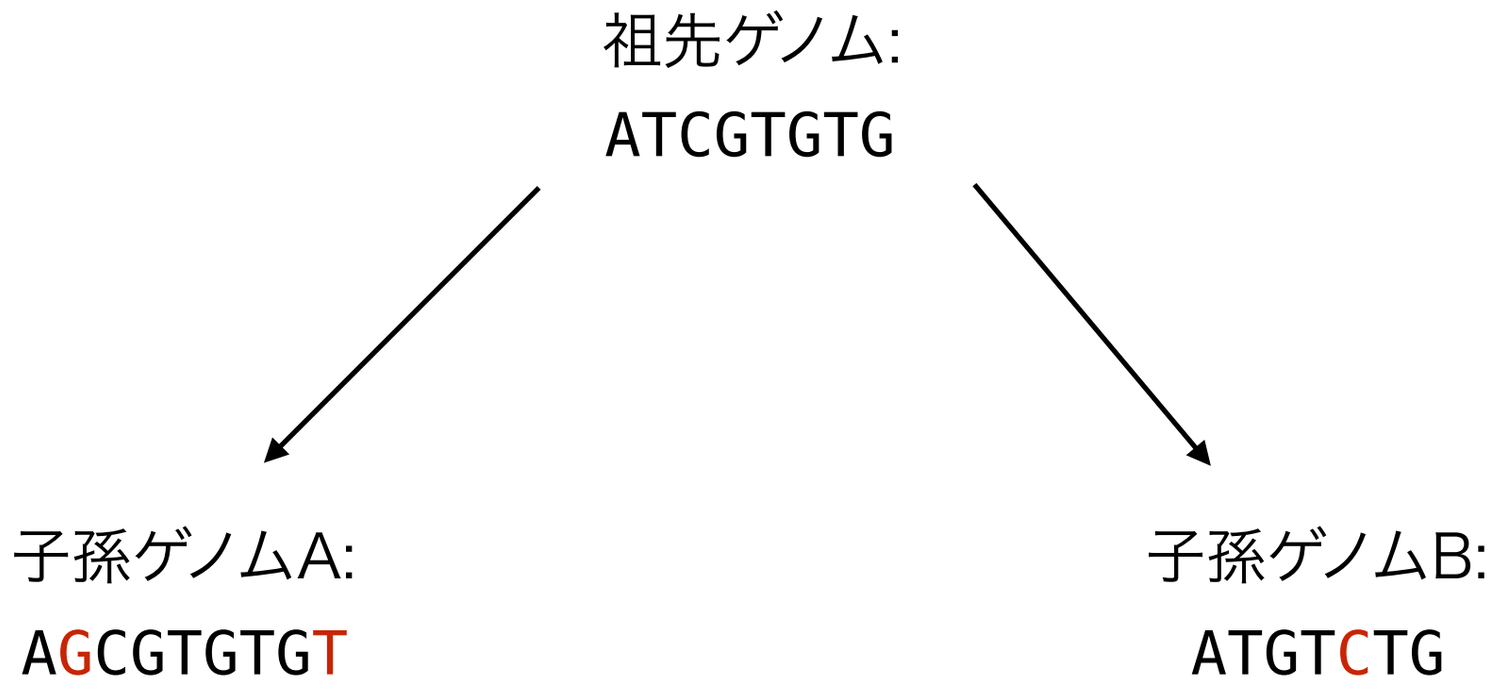


なぜ配列アライメントが重要なのか？

- 一つの理由としては、複数種間のゲノムを比較解析するため

- 複数の生物は進化的な関係にあり、祖先を辿れば同一の種であった。

- 種間におけるゲノムの違いは、同一であったゲノムが種分化の後に
  変異を積み重ねる事で違いが生じたものである。



なぜ配列アライメントが重要なのか？

- 複数種のゲノムを比較することで、

1. 変異しやすいまたは変異しにくい、DNA(アミノ酸)配列領域を推定できる
   変異しにくい箇所は、機能的に重要な領域である可能性がある

2. 特定の生物が特有に獲得/変異した遺伝子(ゲノム領域)を調査出来る
   その領域は、生物固有の機能に関わっている可能性がある

3. ある生物における機能不明遺伝子が、別の生物の機能既知遺伝子に
   アライメントされた場合、その機能不明遺伝子の機能を推定することが出来る

(などなど)

配列アライメントの課題

- 配列アライメントのプログラムを設計する時は、まず問題(何を解きたいのか？)
  を明確にすることが重要、その上で重要な視点は次の2つ

1. どのようなモデルでアライメントを行うのか？

2. 実用可能なほど十分に高速か？

配列アライメントの課題

- 配列アライメントのプログラムを設計する時は、まず問題(何を解きたいのか？)を明確にすることが重要、その上で重要な視点は次の2つ

1. どのようなモデルでアライメントを行うのか？

2. 実用可能なほど十分に高速か？

編集距離のモデル

編集距離

2 つの文字列 S, T が与えられます．S に以下の 3 通りの操作を繰り返し施すことで T に変換したいものとします．そのような一連の操作のうち，操作回数の最小値を求めてください．なお，この最小値を S と T との編集距離とよびます．

- 変更：S 中の文字を 1 つ選んで任意の文字に変更する
- 削除：S 中の文字を 1 つ選んで削除する
- 挿入：S の好きな箇所に好きな文字を 1 文字挿入する

・ このモデルは、変更・削除・挿入を同様のスコアとして扱っている

・ MAQRDAとMGQRIAでは、

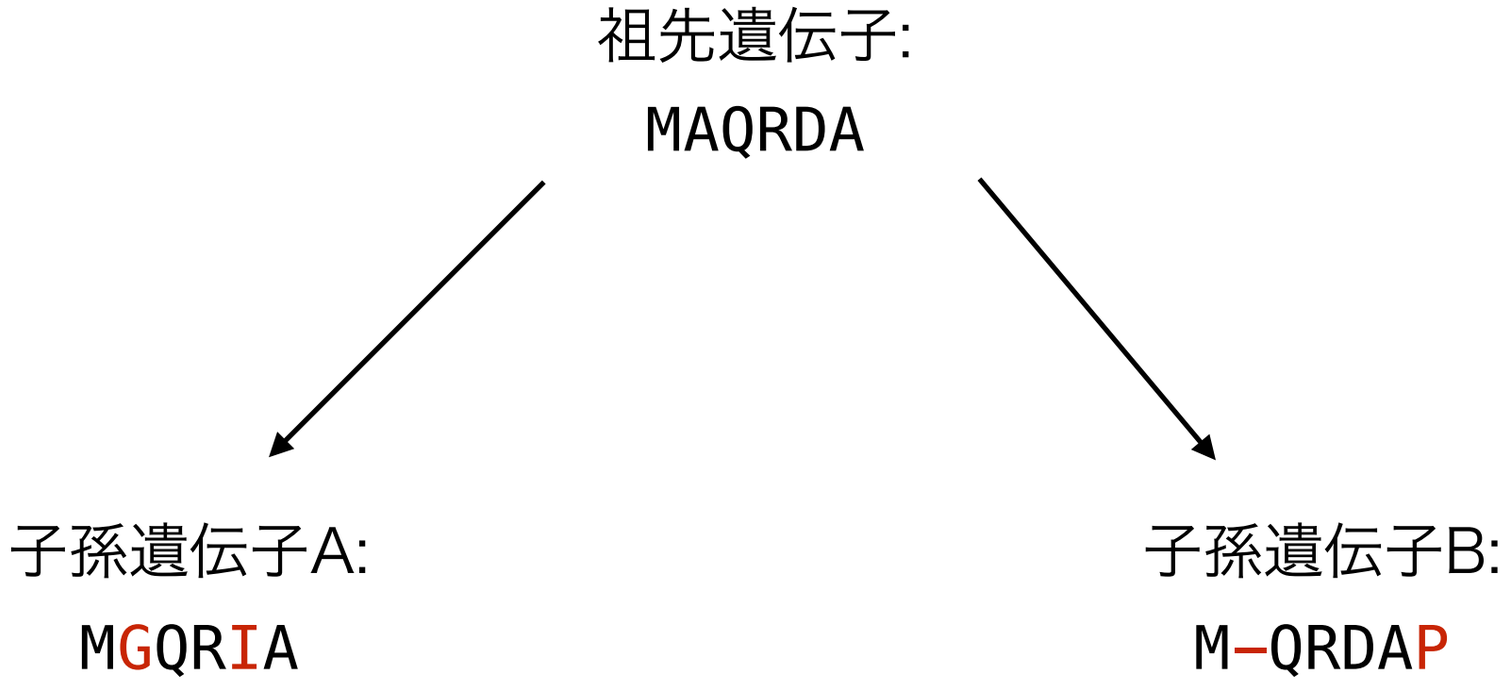
MAQRDA
|X||X|
MGQRIA
のため、編集距離は2となる

・ 同じく、
MAQRDAとMQRDAPでは、

MAQRDA-
|X||||X
M-QRDAP
のため、編集距離は2となる

編集距離のモデル

・すなわちこのモデルは、比較ゲノムの観点から見ると、進化の過程において、
　1. アミノ酸間の変異は全て等確率で起こりうる
　2. 変異と挿入削除が同程度の確率で入りうる
ことを仮定したモデルである



・このモデルは「正しい」モデルなのか？

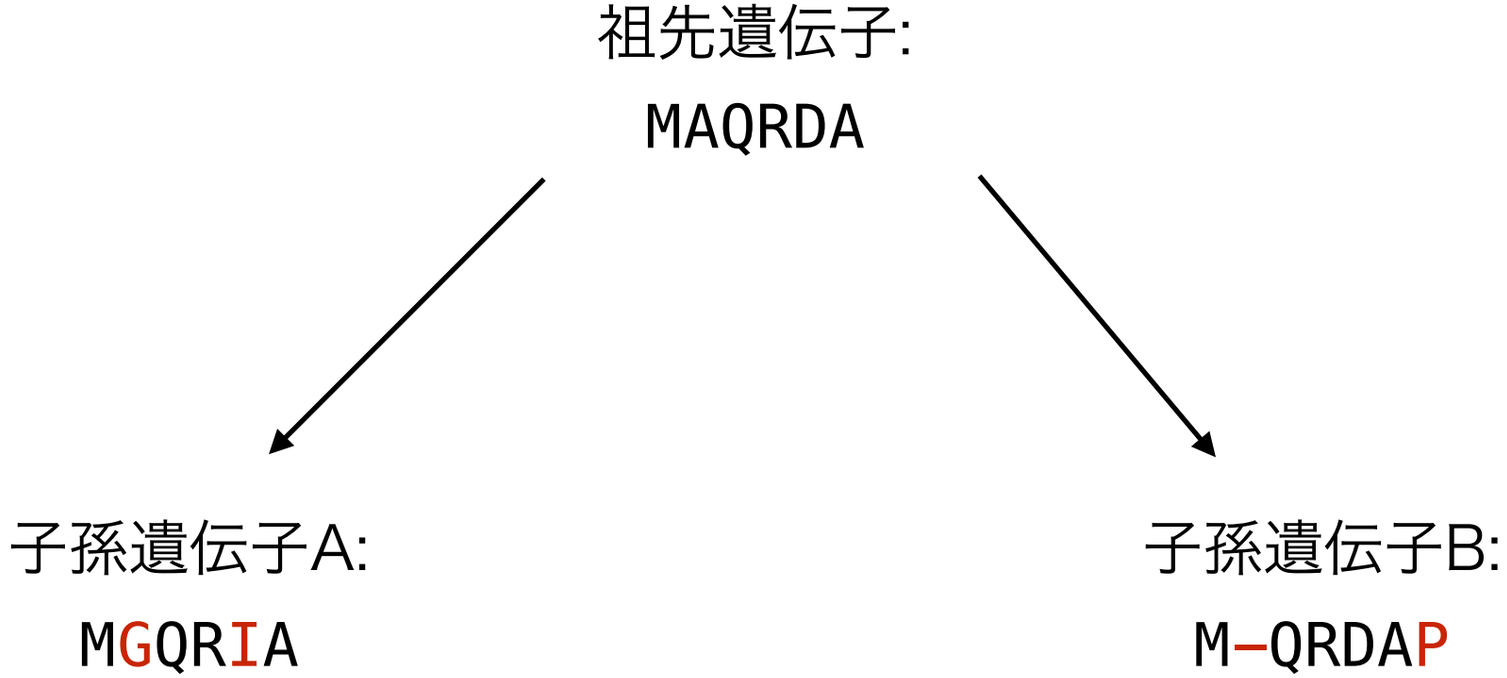
All models are wrong

but some are useful

George E.P. Box

編集距離のモデル

・すなわちこのモデルは、比較ゲノムの観点から見ると、進化の過程において、
　1. アミノ酸間の変異は全て等確率で起こりうる
　2. 変異と挿入削除が同程度の確率で入りうる
ことを仮定したモデルである

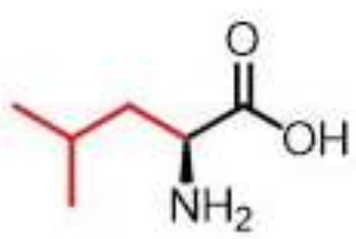


・このモデルは「役に立つ」モデルなのか？
→それなりに役に立つがもっと役に立つモデルがある

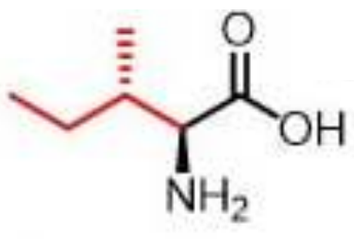
アミノ酸間の変異は全て等確率で起こりうる？

- 20種類のアミノ酸には様々な化学的性質があり、似たアミノ酸と
  似ていないアミノ酸がある

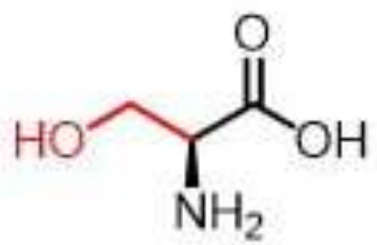
- 例)
  ロイシンとイソロイシンは共に疎水性アミノ酸であり、良く似ている
  セリンは親水性アミノ酸で、これらには似ていない



ロイシン
Leu L



イソロイシン
Ile I



セリン
Ser S

- ロイシンからイソロイシンへの変化は起きやすいが、
  ロイシンからセリンへの変化は起きにくい

- このような傾向をスコアに反映させた方が精度が上がることが知られている

変異と挿入削除が同程度の確率で入りうる？

- 挿入削除の方が稀な変異であるため、これも起こりにくいものとして
  モデル化する。

- さらに挿入削除の場合は、一気に複数個の要素が挿入／削除する事がある。
  これをモデル化したものがアフィンギャップと呼ばれる

- 例)　MAQRDA
  　　|XXX||
  　　M---DA

  リニアギャップ(gap penalty = 2)であれば、このギャップのスコアは
  2*3 = 6 (各ギャップは独立に入ったと仮定)

  アフィンギャップ(gap open penalty = 3, gap extention penalty = 1)
  であれば、このギャップのスコアは3+1*2=5

アフィンギャップのモデルの正しさ

- アフィンギャップは進化における挿入／削除を近似的にモデル化しているが、
  実は正しくモデル化出来ているわけではない

- (アフィンギャップコストで仮定したモデルによって得られるギャップ長の
  分布が、現実に見られるギャップ長の分布と異なっている)

- このため、より現実を反映したギャップスコアに基づいて
  アライメントを行おうという研究が(情報学的には)ある

- しかし、これらのモデルは生物学的な研究ではほぼ使われていない
  なぜなら、モデルが複雑になり計算が大変になる一方で、
  (今の所)精度が向上しているわけではないため

  (より正しそうなモデルであっても必ずしも役に立つとは限らない)

まだモデル化していないこと

- 現在のモデルでは、各アミノ酸／塩基は独自に変異する
  ことを仮定しているが、実際には共変異することがある

- ゲノムレベルでは、逆位や転座といった進化も起きる

- 一般的なアライメントツールではこれらの現象は
  モデル化されていない
  (複雑な事象をモデル化するほど計算時間が増大する)

- これらを研究したい場合は、それに特化した
  ツールを使う(あるいは作る)必要がある。

種1　MPILG・・TV

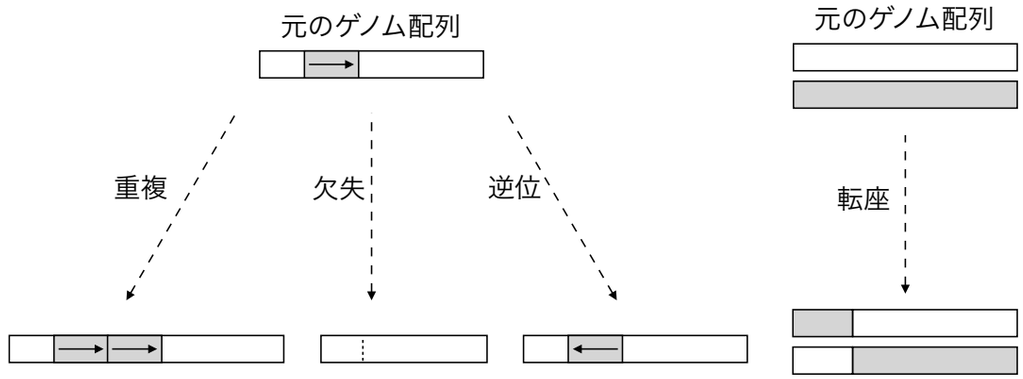
種2　MPISM・・TQ

種3　MPILA・・TV

種4　MEFSK・・PQ

種5　MPFLV・・PV

種6　MPFSH・・PQ



配列アライメントの課題

- 配列アライメントのプログラムを設計する時は、まず問題(何を解きたいのか？)を明確にすることが重要、その上で重要な視点は次の2つ

1. どのようなモデルでアライメントを行うのか？

2. 実用可能なほど十分に高速か？

配列アライメントの時間計算量

- 編集距離の計算量は$O(N^2)$であった(N:配列長)。
  アミノ酸間の置換で異なるスコアを使ったり、アフィンギャップを用いても
  $O(N^2)$で計算する事ができる

- これは多項式時間アルゴリズムであるが、大規模データを処理する上では
  非常に遅い、よって更なる高速化が必要

- どのように高速化するか？
  1. アルゴリズムを改良する
  2. 並列計算を行う
  3. 実装を頑張る
  4. ヒューリスティクスを使う

  (現実には1~4が様々に組み合わさっている)

配列アライメントの時間計算量

- 配列アライメントは$O(n^2)$から計算量を下げる事が出来るか？

Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false)

Arturs Backurs
MIT
backurs@mit.edu

Piotr Indyk
MIT
indyk@mit.edu

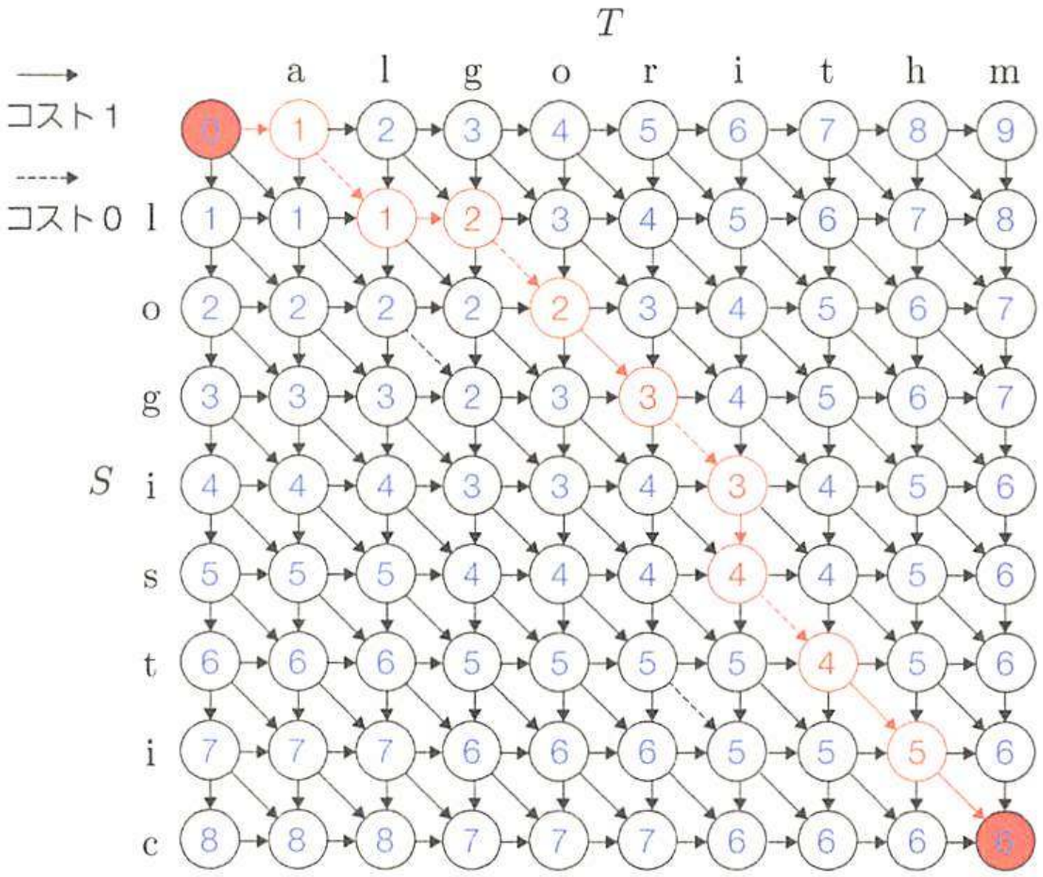
(STOC 2015)

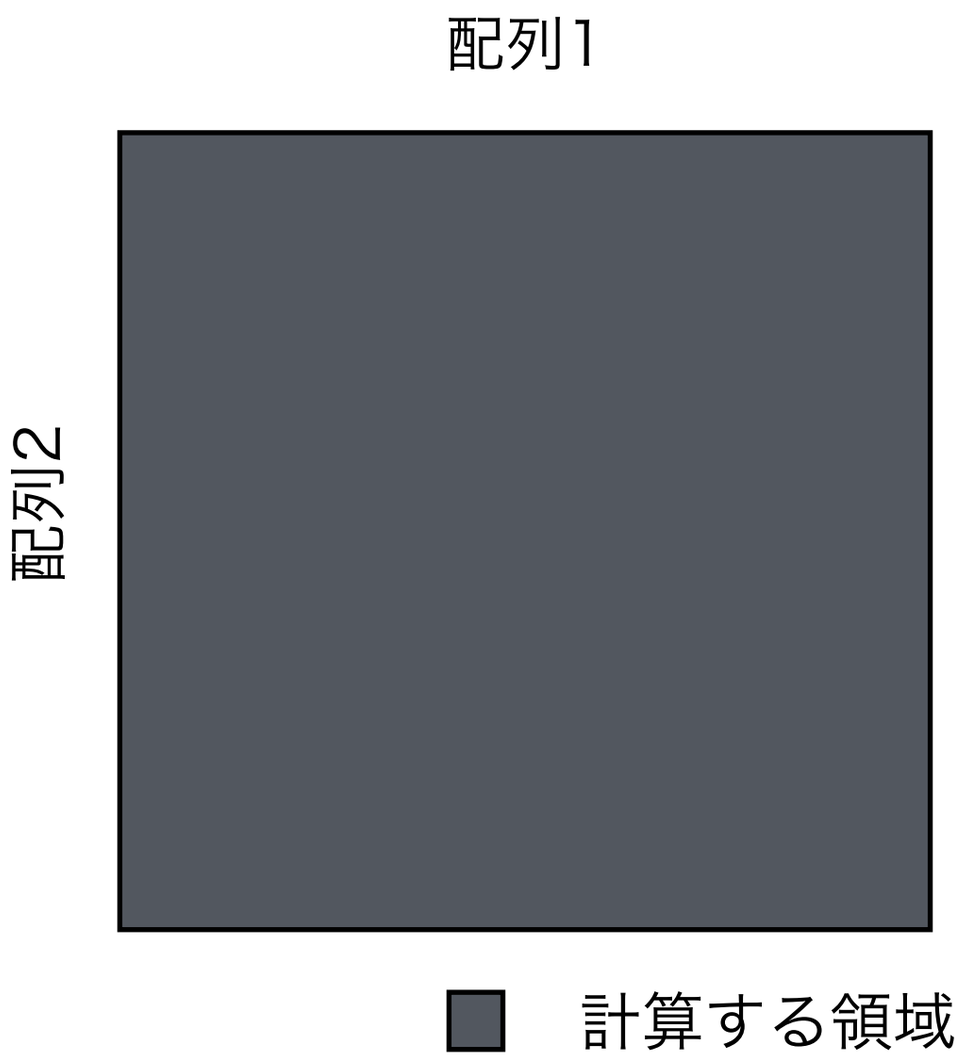
- SETH:
  簡潔には、SATと呼ばれる問題は$O((2-\varepsilon)^n)$では解けないとする仮説

- SETHを真だとすると(多くの研究者が真だと思っているが)、
  配列アライメントは$O(n^{2-\varepsilon})$では解けない
  ($O(n^2/\log n)$で解くアルゴリズムは存在する)

ヒューリスティクスを使う

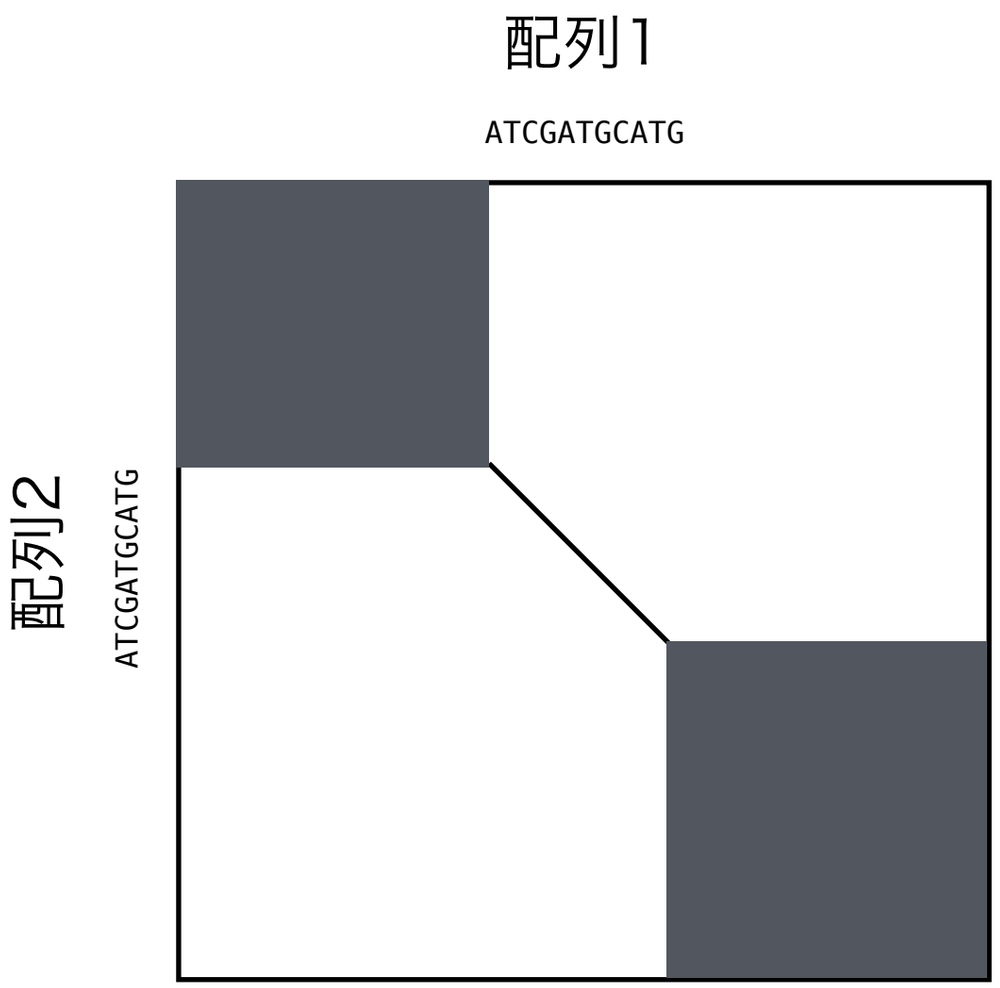
- 配列アライメントがO(n²)になるのは、動的計画法で表の領域を埋める必要があるため
- ヒューリスティクスを使って、表の一部だけ計算するようにする





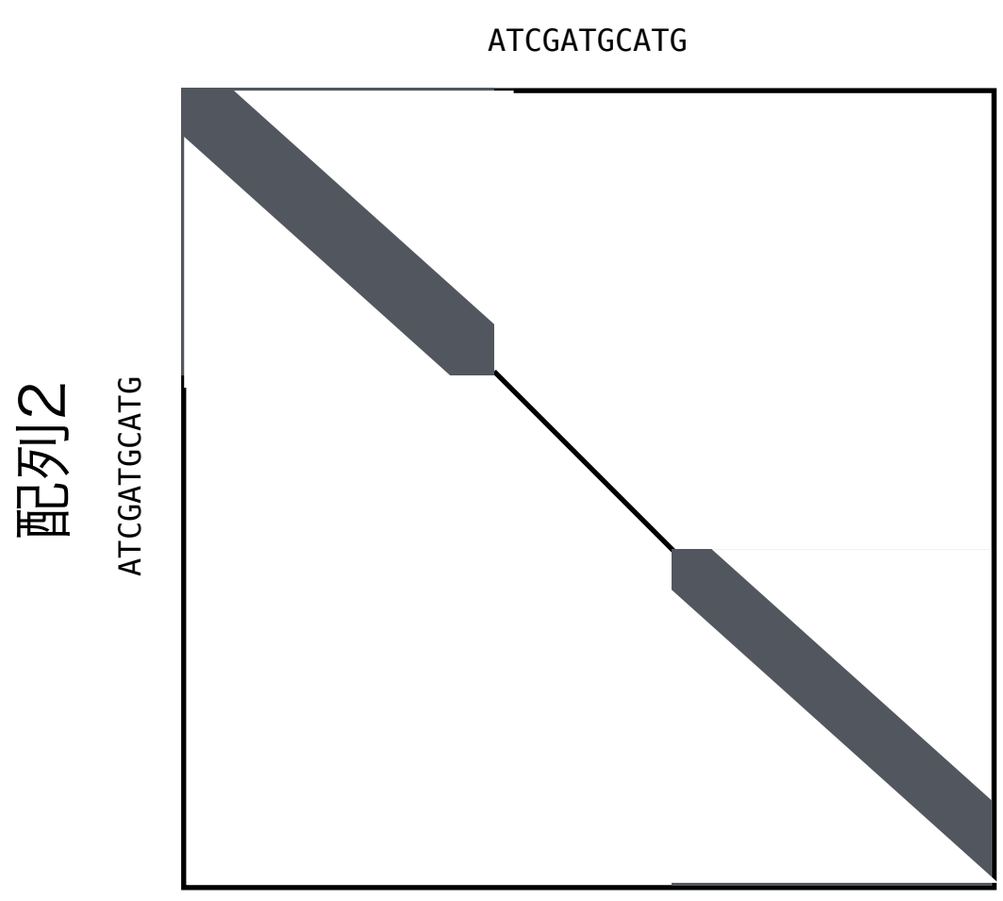
Seed-and-extension

- 配列1と配列2で一定長以上連続して一致している領域(seed)を探す
  seedの領域は固定して、seed以外の部分をアライメントする(extension)



Banded DP

- 配列アライメントは、発見したseed領域のほぼ延長上にあろうと考えられるのでそこから離れている部分は計算しない



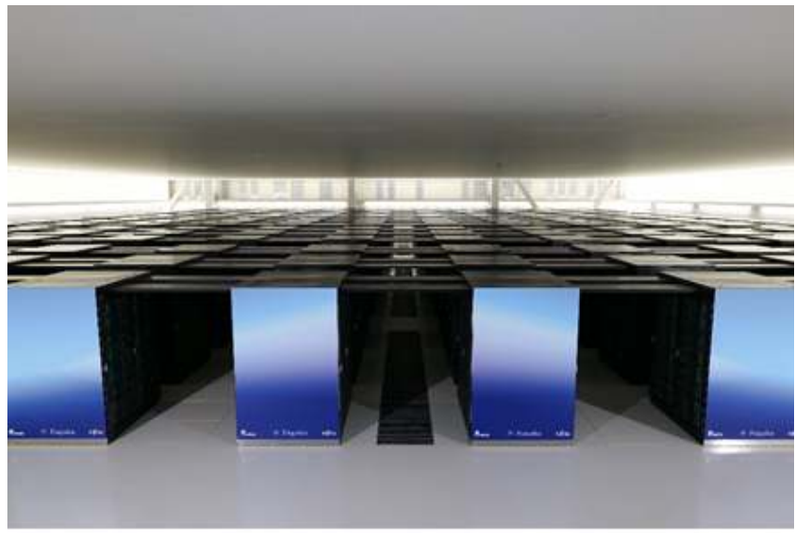
- これらはあくまでヒューリスティクスであるが、わずかな精度低下と引き換えに大きな高速化をもたらすので、現在では一般的に使われるテクニックである

並列計算を行う

- 一般的な計算機では計算が終わらない場合、スパコンを使って
  並列計算を行うというのは現在の科学では基本

スパコン京

スパコン富嶽



東大医科研スパコン

遺伝研スパコン

並列計算を行う

- スパコンそのものは、生命情報科学において広く使われている

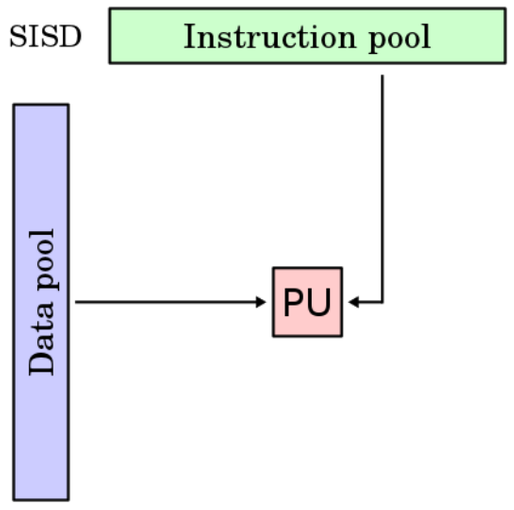
- 配列アライメント解析を行うときは、2本のペアに対してアライメントを行うのではなく、M個の配列が与えられ、その中で(ほぼ)網羅的にアライメントを行う事が多い(M(M-1)/2個のペア)

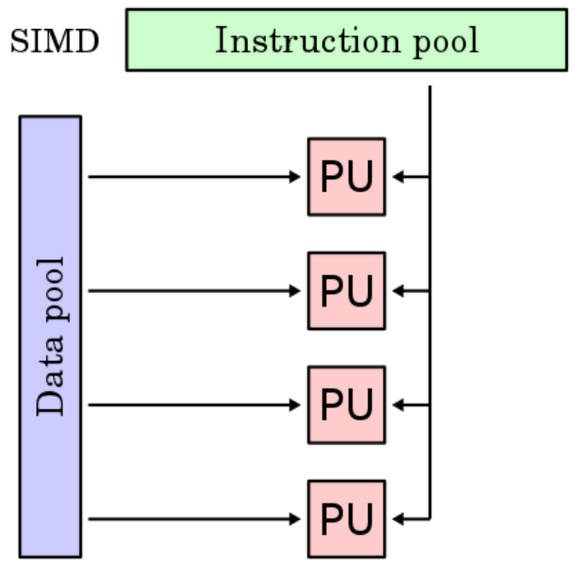
- 並列計算では、アルゴリズムを並列化するより、異なるペアを同時に計算するように並列化する
  (L回独立の計算が出来るとすると、単純にL倍高速化するので)

- アルゴリズムを並列化するには、並列アルゴリズムを構築する必要があるがL倍高速化するアルゴリズムを書くことは難しい

  (大変単純化した話であることに注意)

SISDとSIMD

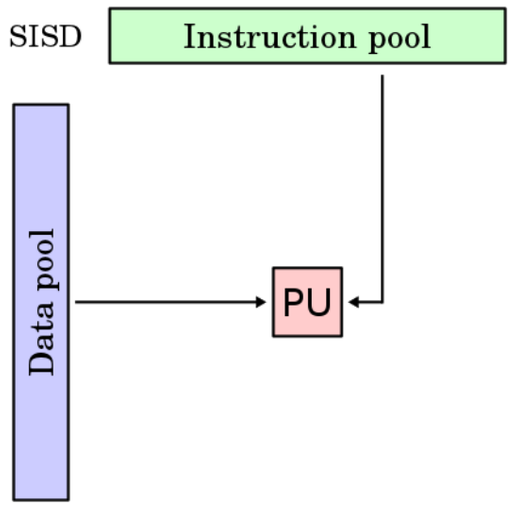


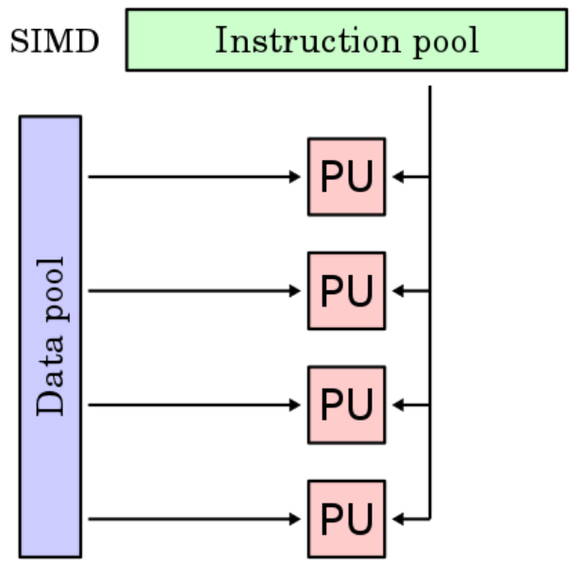


- SISD: single instruction, single data
  単一の命令が一つのデータに対してのみ実行される

- SIMD: single instruction, multiple data
  単一の命令が複数のデータに対して同時に実行される (ベクトル演算)
  たとえば、
  for(int i = 0; i <n; i++){
    a[i] = b[i] + c[i]
  }
  のような演算はSIMDで並列化しやすい

SISDとSIMD





- SIMD命令は現在の標準的なアーキテクチャには備わっている

- C++などでは、コンパイラによって自動的にSIMDを使うよう
  並列化してくれる部分もある(自動ベクトル化)
  プログラム上で明示的に指定することも可能である

- SIMDによる並列化が効きやすいようにアルゴリズムを設計したり、
  あるいはコードを書き換えたりすることも行われる

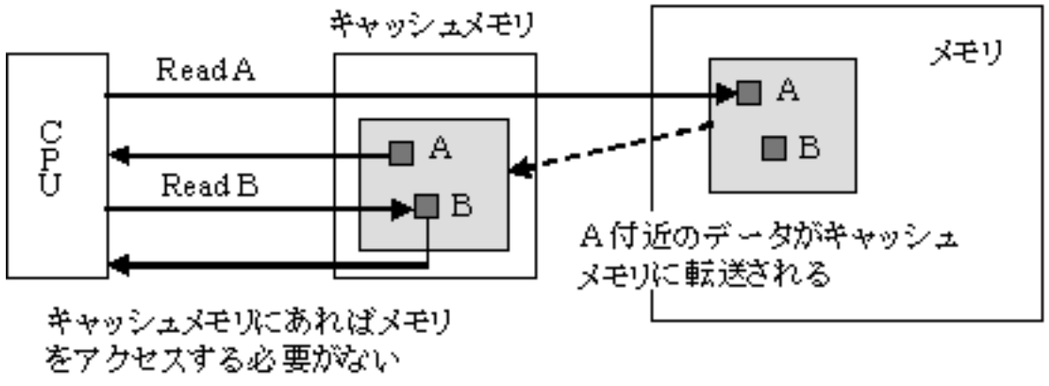
実装を頑張る

- 同一のアルゴリズムを実装しても、実装の言語や方法によって
  実行速度は大きく異なる

- 言語で言えば、一般に、Ruby/Python/Rは遅く、C/C++/Rustは速い
  (Pythonライブラリなどで事実上Cで書かれている場合は除く)

- 実装方法においては、特にメモリアクセスが計算時間に大きな影響を与える

記憶階層



平均取り出し時間
＝　ヒット率×キャッシュメモリからの取り出し時間
＋（１－ヒット率）×メモリからの取り出し時間

- メモリへのアクセスは時間がかかるが、キャッシュメモリへのアクセスは時間がかからない

- メモリへのアクセスを避け、キャッシュメモリへのアクセスが多くなるように実装すると実行時間が減る

- キャッシュメモリはさらにL1, L2, L3に分けられ、L1の方がアクセスが早い(そしてメモリ容量が少ない)

実装を頑張る(マニアック)

- プロセッサが一つのアセンブリ命令を行うには
  1. 命令のフェッチ
  2. 命令デコード
  3. 実行
  4. メモリアクセス
  5. レジスタへのライトバック

  という5つの工程を踏む必要がある

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

- 複数の命令を1つずつおこなっていくのは効率が悪いので、
  この工程は並列化されている。すなわち、
  ある命令Aが実行されているとき、命令Bはデコードされていて、命令Cは
  フェッチされている(およそ5倍の高速化になる)

- これをパイプライン処理という

実装を頑張る(マニアック)

- パイプラインハザードとは、何らかの理由によりパイプライン処理が
  行えない事を指す。その一つが制御ハザードであり、if文の処理の際など
  に出てくる。if文の結果によって次行うべき命令が変わってくるので、
  先取りで命令のフェッチなどを行うことができない。

- アーキテクチャとして良く使われる技術が、分岐予測である。
  if文の結果としてどちらに分岐しやすいかを予測し、そちらが起こるだろう
  と仮定して命令のフェッチなどを行う。外れた場合にはロスとなる。

- 実装において、分岐予測が成功しやすいようにプログラミングをすることで
  プログラムを高速化することが可能である。

- 「コンパイラとCPUの気持ちを考えて実装する」らしい
  (残念ながら私はこの領域に達していない)

アライメントフリー法

- それでも配列アライメントは計算に時間がかかる。より高速な方法として、
  アライメントフリーで配列比較を行う事があり、たとえば、k-merを比較する。

- 例)

  S: ATCGTCTGA
  T: ATGTCTCAT

  という2つの配列を2-merで比較する。

この時、各配列において各2-merの存在を調べ、配列を
その有無を表す特徴量ベクトルに変換する。

例) Sの変換

AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
0	0	0	1	0	0	1	1	1	0	0	1	0	1	1	0

アライメントフリー法

	AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
S	0	0	0	1	0	0	1	1	1	0	0	1	0	1	1	0
T	0	0	0	1	1	0	0	1	0	0	0	1	0	1	1	0

- 特徴量ベクトルを得た後に、Jaccard係数を計算する

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

- 上記の例で言えば、$|S \cap T|=5$、$|S \cup T|=8$なので、$J(S, T) = 0.625$

アライメントフリー法

- 特徴量ベクトルの計算はO(N)で行うことが出来る。
  また、Jaccard係数の計算もO(N)で行うことが出来るため、
  計算量はO(N)である(非常に大雑把な近似であることに注意)。

- しかしこれでもまだ遅い！もっと早くしたい！

- Jaccard距離の計算を近似的に高速に行う手法が
  MinHashである(Broder, 1997)。

MinHash

- あるk-merを数字に変換するハッシュ関数hを考える。

  例)
  h(‘ACTGTATC’) = 10
  h(‘TGTCCGGA’) = 30

  hを使って、2つのゲノムS, Tに含まれるk-merの集合S_kとT_kを、
  数字の集合S_nとT_nに変換する。

- ここで、ハッシュ関数がランダムに振る舞うならば、

$$J(S,T) = P(min\{h(a) \mid a \in S_k\} = min\{h(b) \mid b \in T_k\})$$

  が成立することに着目する。
  つまり、S_nに含まれる数字の最小値と、T_nに含まれる数字の最小値が
  一致する確率が、$J(S_k, T_k)$に等しい。

Hash値の衝突

- hは単射でなくて良い。hが単射である場合を完全ハッシュ関数と呼び、
  望ましい性質を持つが、そのようなhを設計することは難しいことが多い
  (あるいは計算に時間がかかり実用的でなかったりする)

- hが単射ではない場合には、hash値の衝突が起きることがある。すなわち、

  h(‘ACTGTATC’) = 10 そしてh(‘GTACGGAT’) = 10

  のように、異なる文字列から同じハッシュ値が生成されることがある。
  よって、本来は同一ではないk-merを同一とみなして計算が進むことがある。

- 衝突を避けたい場合は、値域を大きくすれば良い。

MinHash

- 例) S = ACTAA、T = AGCTAT
  S_k= {AA, AC, CT, TA}
  T_k= {AG, AT, CT, GC, TA}とする。

- $S_k \cap T_k$ = {CT, TA}である。ここで、CTで一致するとはどういう事かというと、

  {AA, AC, CT, TA}の中でCTが最小、かつ
  {AG, AT, CT, GC, TA}の中でCTが最小

  つまり、
  $S_k \cup T_k$ = {AA, AC, AG, AT, CT, GC, TA}においてCTが最小になるという事
  よってその確率は、$1/|S_k \cup T_k|$

- TAで一致する場合も同様なので、結局$|S_k \cap T_k|/|S_k \cup T_k|$ に等しい。

k-min MinHash

- よって、hash関数をk個用意した後(h_1〜h_k)、
  各ハッシュ関数において最小値が一致するかどうかを調べる。

- 最小値が一致した回数をnとすると、J(S, T) = n/kと推定できる

- hash関数をk個用意するのではなく、一つのハッシュ関数において
  小さい方からk個取得する方法もあり、bottom-k sketchと呼ばれる。

- MinHashのように確率的に振る舞うデータ構造は乱拓データ構造と呼ばれる。

- MinHashは生物種のDNA配列同士を比較する上では良く使われる一方、
  アミノ酸配列の比較の上では精度が不十分であり利用が限られている。

深層学習による特徴量の構成

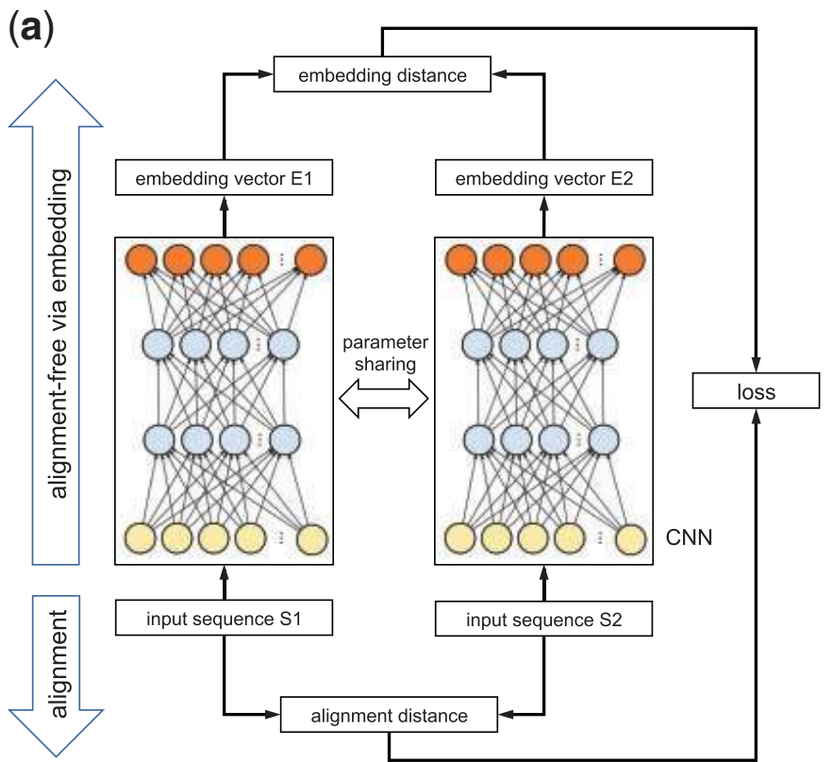
- 先の方法では、k-merの有無によって特徴量ベクトルを構成したが、
  もっと良い特徴量ベクトルの構成方法があるかもしれない。

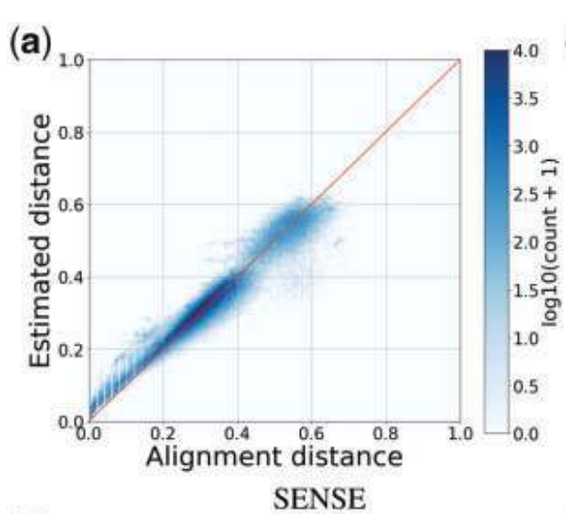
- 深層学習によって、特徴量ベクトルの構成方法そのものを学習するという
  方法も近年提案されている。

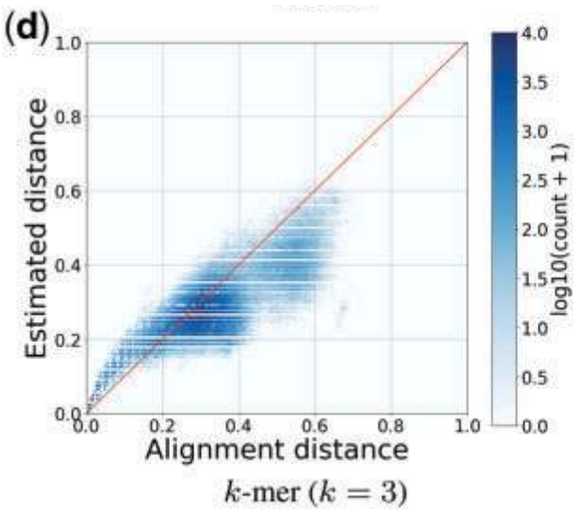
SENSE: Siamese neural network for sequence embedding and alignment-free comparison

Wei Zheng[1,†], Le Yang[1,†], Robert J. Genco[2,3], Jean Wactawski-Wende[4], Michael Buck[5] and Yijun Sun[1,3,*]

(Bioinformatics 2019)







まとめ

- 大規模な生命データを情報解析し、そこから新たな生物学的知見を
  得ることを目指す学問が、生命情報科学である。

- 生命情報科学では、DNA配列またはアミノ酸配列である
  文字列データの解析が、特に大きな比重を占めている。

- 配列解析では、２つの配列を比較する配列アライメントが最も
  基本的な処理である。配列アライメントを行う上では、
  その「モデル」と「計算時間」を考慮する事が重要である。

- 厳密にアルゴリズムを高速化することは難しいので、ヒューリスティックな
  高速化が利用される。その際、並列化や実装上の工夫など様々な
  テクニックが利用される。