Hard bioinformatics problems: genome assembly and multiple sequence alignment

Solon P. Pissis

CWI, Amsterdam, The Netherlands
Vrije Universiteit, Amsterdam, The Netherlands

`solon.pissis@cwi.nl`

December 2, 2024

Genome assembly

We cannot read very long DNA strands with very high accuracy.

We can read short ones (e.g. 1000 bp) with very high accuracy.

This process cuts the strand at random positions.

The order of the fragments is unknown.

Genome assembly: Reconstruct the genome from the fragments.

Informally: *Given a set S of strings, construct the best string that includes every string in S as a substring.*

A specific formalization: *Given a set S of strings, construct a shortest string T that includes every string in S as a substring.*

Example

Let $S = \{\texttt{tagg}, \texttt{catt}, \texttt{gga}, \texttt{tta}, \texttt{gagtat}\}$. A shortest common superstring (SCS) is

$$T = \texttt{cattaggagtat}.$$

Genome assembly

We cannot read very long DNA strands with very high accuracy.

We can read short ones (e.g. 1000 bp) with very high accuracy.

This process cuts the strand at random positions.

The order of the fragments is unknown.

Genome assembly: Reconstruct the genome from the fragments.

Informally: *Given a set S of strings, construct the best string that includes every string in S as a substring.*
A specific formalization: *Given a set S of strings, construct a shortest string T that includes every string in S as a substring.*

Example

Let $S = \{\texttt{tagg}, \texttt{catt}, \texttt{gga}, \texttt{tta}, \texttt{gagtat}\}$. A shortest common superstring (SCS) is

$$T = \texttt{cattaggagtat}.$$

Genome assembly

We cannot read very long DNA strands with very high accuracy.

We can read short ones (e.g. 1000 bp) with very high accuracy.

This process cuts the strand at random positions.

The order of the fragments is unknown.

Genome assembly: Reconstruct the genome from the fragments.

Informally: *Given a set S of strings, construct the best string that includes every string in S as a substring.*
A specific formalization: *Given a set S of strings, construct a shortest string T that includes every string in S as a substring.*

Example

Let $S = \{\texttt{tagg}, \texttt{catt}, \texttt{gga}, \texttt{tta}, \texttt{gagtat}\}$. A shortest common superstring (SCS) is

$$T = \texttt{cattaggagtat}.$$

Reductions as a tool for proving hardness

Say we suspect that a problem is computationally hard.

As a first step, we can settle for a relative statement:

- Problem A is at least as hard as Problem B

To prove such a statement, we reduce B to A:

- Tranform **any** instance of B to **some** instance of A

In other words:

- If you had a black box $\mathcal{A}$ that solves instances of A, how can you solve any instance of B using a polynomial number of steps, plus a polynomial number of calls to $\mathcal{A}$?

What does polynomial mean?

- If n is the input size, then $n^{O(1)}$ is polynomial in n.
- $O(1)$ means any positive constant c.
- e.g. n^2 and n^5 are polynomial in n; 2^n is not!

Reductions as a tool for proving hardness

If problem B can be reduced to problem A, we denote this by

$$B \leq_P A.$$

This means:

- B is polynomal-time reducible to A.

It also means:

- A is at least as hard as B because if you can solve A in polynomial time, then you can solve B in polynomial time.

Let us conclude.

Theorem
If B cannot be solved in polynomial time and $B \leq_P A$, then A cannot be solved in polynomial time.

Choosing a famously hard problem B

Recall that A is the shortest common superstring (SCS) problem.

To prove that A is hard, we need to prove $B \leq_P A$.

- For some problem B that is famously hard (NP-complete).

Definition

A problem is called *nondeterministic polynomial* (NP) if its solution can be guessed and verified in polynomial time — nondeterministic means that no particular rule is followed to make the guess.

Definition

If a problem is NP and all other NP problems are polynomial-time reducible to it, then the problem is called *NP-complete*[1].

This has an important implication:
Finding an efficient algorithm for any NP-complete problem implies that an efficient algorithm can be found for all such problems.

[1] For finding the first such problem, see the Cook-Levin theorem.

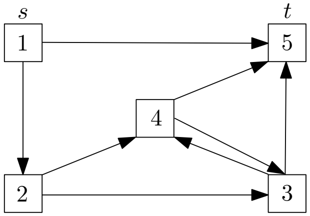
Choosing a famously hard problem B

It is not known whether any polynomial-time algorithm exists for NP-complete problems. This remains one of the most important questions in computer science.

The Hamiltonian Path problem (HP) is such a hard problem: NP-complete.[2]

- For the rest we assume that B is HP
- And want to show that $B \leq_P A$; namely, HP $\leq_P$ SCS

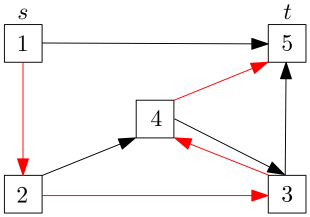
Problem B: Given a directed graph $G(V, E)$, find a path that starts from node designated s, goes through each node in V exactly once, and ends at node designated t.



[2]This was proved by Karp in 1972.

The reduction from B to A

Problem B: Given a directed graph $G(V,E)$, find a path that starts from node designated s, goes through each node in V exactly once, and ends at node designated t (in red is a solution).



Set A_v for every node $v \neq t$

$A_1 = \{\bar{1}2\bar{1}\}$

$A_1 = \{\bar{1}2\bar{1}, \bar{1}5\bar{1}\}$

$A_1 = \{\bar{1}2\bar{1}, \bar{1}5\bar{1}, 2\bar{1}5\}$

$A_1 = \{\bar{1}2\bar{1}, \bar{1}5\bar{1}, 2\bar{1}5, 5\bar{1}2\}$

A connector for every node v

$C_2 = \{2\#\bar{2}\}$

$C_3 = \{3\#\bar{3}\}$

$C_4 = \{4\#\bar{4}\}$

$C_{1,5} = \{c\#\bar{1}, 5\#\$\}$

For every out-neighbor u

$\bar{v}u_{\text{first}}\bar{v}$

$\ldots, \bar{v}u_{\text{next}}\bar{v}, \ldots$

$u_{\text{first}}\bar{v}u_{\text{next}}, \ldots$

$\ldots, u_{\text{last}}\bar{v}u_{\text{first}}$

Explanation

$v\#\bar{v}$

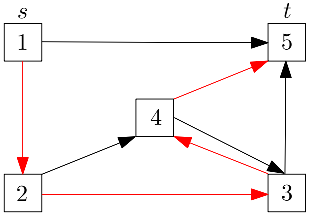
$v\#\bar{v}$

$v\#\bar{v}$

$c\#\bar{u}_{\text{source}},\ u_{\text{target}}\#\$$

The reduction from B to A

Problem B: Given a directed graph $G(V, E)$, find a path that starts from node designated s, goes through each node in V exactly once, and ends at node designated t (in red is a solution).



Gadgets for every node $\neq t$

$A_1 = \{\bar{1}2\bar{1}, \bar{1}5\bar{1}, 2\bar{1}5, 5\bar{1}2\}$

$A_2 = \{\bar{2}3\bar{2}, \bar{2}4\bar{2}, 3\bar{2}4, 4\bar{2}3\}$

$A_3 = \{\bar{3}4\bar{3}, \bar{3}5\bar{3}, 4\bar{3}5, 5\bar{3}4\}$

$A_4 = \{\bar{4}5\bar{4}, \bar{4}3\bar{4}, 5\bar{4}3, 3\bar{4}5\}$

A connector for every node

$C_2 = \{2\#\bar{2}\}$

$C_3 = \{3\#\bar{3}\}$

$C_4 = \{4\#\bar{4}\}$

$C_{1,5} = \{c\#\bar{1}, 5\#\$\}$

$$S = A_1 \cup A_2 \cup A_3 \cup A_4 \cup C_2 \cup C_3 \cup C_4 \cup T$$

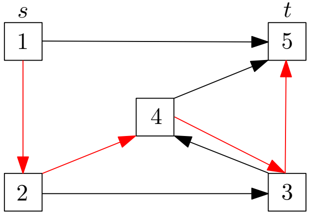
G has a directed HP $\iff$ S has a superstring $T : |T| = 2|E| + 3|V|$.

$$T = c\#\bar{1}2\bar{1}5\bar{1}2\#\bar{2}3\bar{2}4\bar{2}3\#\bar{3}4\bar{3}5\bar{3}4\#\bar{4}5\bar{4}3\bar{4}5\#\$$$

$|T| = 31 = 2 \cdot 8 + 3 \cdot 5 = 2|E| + 3|V|$.

The reduction from B to A

Problem B: Given a directed graph $G(V, E)$, find a path that starts from node designated s, goes through each node in V exactly once, and ends at node designated t (in red is a solution).



Gadgets for every node $\neq t$

$A_1 = \{\bar{1}2\bar{1}, \bar{1}5\bar{1}, 2\bar{1}5, 5\bar{1}2\}$
$A_2 = \{\bar{2}3\bar{2}, \bar{2}4\bar{2}, 3\bar{2}4, 4\bar{2}3\}$
$A_3 = \{\bar{3}4\bar{3}, \bar{3}5\bar{3}, 4\bar{3}5, 5\bar{3}4\}$
$A_4 = \{\bar{4}5\bar{4}, \bar{4}3\bar{4}, 5\bar{4}3, 3\bar{4}5\}$

A connector for every node

$C_2 = \{2\#\bar{2}\}$
$C_3 = \{3\#\bar{3}\}$
$C_4 = \{4\#\bar{4}\}$
$C_{1,5} = \{c\#\bar{1}, 5\#\$\}$

$$S = A_1 \cup A_2 \cup A_3 \cup A_4 \cup C_2 \cup C_3 \cup C_4 \cup T$$

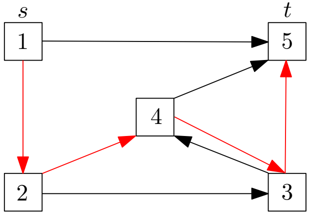
G has a directed HP $\iff$ S has a superstring $T : |T| = 2|E| + 3|V|$.

$$T' = c\#\bar{1}2\bar{1}5\bar{1}2\#\bar{2}4\bar{2}3\bar{2}4\#\bar{4}3\bar{4}5\bar{4}3\#\bar{3}5\bar{3}4\bar{3}5\#\$$$

$|T'| = 31 = 2 \cdot 8 + 3 \cdot 5 = 2|E| + 3|V|$.

The reduction from B to A

Problem B: Given a directed graph $G(V, E)$, find a path that starts from node designated s, goes through each node in V exactly once, and ends at node designated t (in red is a solution).



Gadgets for every node $\neq t$

$A_1 = \{\bar{1}2\bar{1}, \bar{1}5\bar{1}, 2\bar{1}5, 5\bar{1}2\}$
$A_2 = \{\bar{2}3\bar{2}, \bar{2}4\bar{2}, 3\bar{2}4, 4\bar{2}3\}$
$A_3 = \{\bar{3}4\bar{3}, \bar{3}5\bar{3}, 4\bar{3}5, 5\bar{3}4\}$
$A_4 = \{\bar{4}5\bar{4}, \bar{4}3\bar{4}, 5\bar{4}3, 3\bar{4}5\}$

A connector for every node

$C_2 = \{2\#\bar{2}\}$
$C_3 = \{3\#\bar{3}\}$
$C_4 = \{4\#\bar{4}\}$
$C_{1,5} = \{c\#\bar{1}, 5\#\$\}$

$$S = A_1 \cup A_2 \cup A_3 \cup A_4 \cup C_2 \cup C_3 \cup C_4 \cup T$$

G has a directed HP $\iff$ S has a superstring $T : |T| = 2|E| + 3|V|$.

Theorem (Gallant et al., JCSS 1980)

The SCS problem is NP-complete.

Multiple sequence alignment (MSA)

MSA is a computationally hard problem. Let's be more precise...

Definition

Given a fixed MSA M of sequences $S_1, \ldots, S_k$ and a cost function, let $d_M(S_i, S_j)$ denote the cost of the alignment between S_i and S_j **as implied by** M. Then the sum-of-pairs (SP) score is defined as:

$$\text{SP-score}(M) = \sum_{1 \leq i < j \leq k} d_M(S_i, S_j).$$

MSA implies a pairwise alignment between every pair of sequences. The implied pairwise alignment may not be optimal!

Example

Let the cost of match be -1, of mismatch 1, and of gap 2. Let M be:

```
AT
A-
-T
AT
AT
```

Cost of column 1: $\text{cost}(\mathtt{A}, \mathtt{A}) + \text{cost}(\mathtt{A}, \mathtt{-}) + \cdots + \text{cost}(\mathtt{A}, \mathtt{A}) = 2$

Cost of column 2: $\text{cost}(\mathtt{T}, \mathtt{-}) + \text{cost}(\mathtt{T}, \mathtt{T}) + \cdots + \text{cost}(\mathtt{T}, \mathtt{T}) = 2$

$\text{SP-score}(M) = 2 + 2 = 4$

$d_M(\mathtt{A}, \mathtt{T}) = 4$ but the optimal alignment of A and T has cost 1!

Multiple sequence alignment (MSA)

MSA is a computationally hard problem. Let's be more precise...

Definition

Given a fixed MSA M of sequences $S_1, \ldots, S_k$ and a cost function, let $d_M(S_i, S_j)$ denote the cost of the alignment between S_i and S_j **as implied by** M. Then the sum-of-pairs (SP) score is defined as:

$$\text{SP-score}(M) = \sum_{1 \le i < j \le k} d_M(S_i, S_j).$$

MSA implies a pairwise alignment between every pair of sequences. The implied pairwise alignment may not be optimal!

Example

Let the cost of match be -1, of mismatch 1, and of gap 2. Let M be:

```
AT
A-
-T
AT
AT
```

Cost of column 1: $\text{cost}(\texttt{A}, \texttt{A}) + \text{cost}(\texttt{A}, \texttt{-}) + \cdots + \text{cost}(\texttt{A}, \texttt{A}) = 2$

Cost of column 2: $\text{cost}(\texttt{T}, \texttt{-}) + \text{cost}(\texttt{T}, \texttt{T}) + \cdots + \text{cost}(\texttt{T}, \texttt{A}) = 2$

$\text{SP-score}(M) = 2 + 2 = 4$

$d_M(\texttt{A}, \texttt{T}) = 4$ but the optimal alignment of A and T has cost 1!

Multiple sequence alignment (MSA)

MSA is a computationally hard problem. Let's be more precise...

Definition

Given a fixed MSA M of sequences $S_1, \ldots, S_k$ and a cost function, let $d_M(S_i, S_j)$ denote the cost of the alignment between S_i and S_j **as implied by** M. Then the sum-of-pairs (SP) score is defined as:

$$\text{SP-score}(M) = \sum_{1 \leq i < j \leq k} d_M(S_i, S_j).$$

MSA implies a pairwise alignment between every pair of sequences. The implied pairwise alignment may not be optimal!

Example

Let the cost of match be -1, of mismatch 1, and of gap 2. Let M be:

```
AT
A-
-T
AT
AT
```

Cost of column 1: $\text{cost}(\mathtt{A}, \mathtt{A}) + \text{cost}(\mathtt{A}, \mathtt{-}) + \cdots + \text{cost}(\mathtt{A}, \mathtt{A}) = 2$

Cost of column 2: $\text{cost}(\mathtt{T}, \mathtt{-}) + \text{cost}(\mathtt{T}, \mathtt{T}) + \cdots + \text{cost}(\mathtt{T}, \mathtt{A}) = 2$

$\text{SP-score}(M) = 2 + 2 = 4$

$d_M(\mathtt{A}, \mathtt{T}) = 4$ but the optimal alignment of `A` and `T` has cost 1!

Multiple sequence alignment (MSA)

MSA is a computationally hard problem. Let's be more precise...

Definition

Given a fixed MSA M of sequences $S_1, \ldots, S_k$ and a cost function, let $d_M(S_i, S_j)$ denote the cost of the alignment between S_i and S_j **as implied by** M. Then the sum-of-pairs (SP) score is defined as:

$$\text{SP-score}(M) = \sum_{1 \leq i < j \leq k} d_M(S_i, S_j).$$

MSA implies a pairwise alignment between every pair of sequences. The implied pairwise alignment may not be optimal!

Example

Let the cost of match be -1, of mismatch 1, and of gap 2. Let M be:

```
AT
A-
-T
AT
AT
```

Cost of column 1: $\text{cost}(\mathtt{A}, \mathtt{A}) + \text{cost}(\mathtt{A}, \mathtt{-}) + \cdots + \text{cost}(\mathtt{A}, \mathtt{A}) = 2$

Cost of column 2: $\text{cost}(\mathtt{T}, \mathtt{-}) + \text{cost}(\mathtt{T}, \mathtt{T}) + \cdots + \text{cost}(\mathtt{T}, \mathtt{A}) = 2$

$\text{SP-score}(M) = 2 + 2 = 4$

$d_M(\mathtt{A}, \mathtt{T}) = 4$ but the optimal alignment of A and T has cost 1!

Multiple sequence alignment (MSA)

MSA is a computationally hard problem. Let's be more precise...

Definition

Given a fixed MSA M of sequences $S_1, \ldots, S_k$ and a cost function, let $d_M(S_i, S_j)$ denote the cost of the alignment between S_i and S_j **as implied by** M. Then the sum-of-pairs (SP) score is defined as:

$$\text{SP-score}(M) = \sum_{1 \le i < j \le k} d_M(S_i, S_j).$$

MSA implies a pairwise alignment between every pair of sequences. The implied pairwise alignment may not be optimal!

We know the following theorem.

Theorem (Just, J. Comput. Biol. 2001)

MSA under SP-score is NP-complete.

Approximation algorithms

Approximation algorithms are polynomial-time algorithms that find approximate solutions to optimization problems:

- in particular, to computationally hard problems;
- with provable guarantees wrt to the optimal solution.

Definition

An algorithm $\mathcal{A}$ for a minimization problem A has an approximation ratio c, if for any instance $\mathcal{I}$ of A, $\frac{\mathcal{A}(\mathcal{I})}{\mathcal{OPT}(\mathcal{I})} \leq c$.

Definition

An algorithm $\mathcal{A}$ for a maximization problem A has an approximation ratio c, if for any instance $\mathcal{I}$ of A, $\frac{\mathcal{OPT}(\mathcal{I})}{\mathcal{A}(\mathcal{I})} \leq c$.

Instead of heuristics...

...we can try to design an approximation algorithm for MSA!

The Star Alignment (SA) algorithm

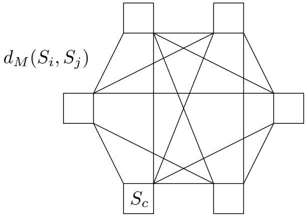
Given k input sequences $S_1, \dots, S_k$, SA works as follows:

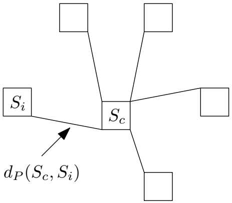
1. Construct all $O(k^2)$ pairwise alignments
2. Let S_c be the sequence in $\{S_1, \dots, S_k\}$ closest to the others; namely, choose S_c that minimizes:

$$\sum_{i \neq c} d_P(S_c, S_i),$$

   where d_P is the optimal cost of pairwise alignment.

3. Progressively align all other $k-1$ sequences onto S_c.





The Star Alignment (SA) algorithm

We make the following standard assumptions:

- The cost function satisfies the triangle inequality:
  $\text{cost}(a, b) \leq \text{cost}(a, c) + \text{cost}(c, b)$,
  a, b, c are column elements.

Example

$\text{cost}(\texttt{A}, \texttt{C}) \leq \text{cost}(\texttt{A}, \texttt{G}) + \text{cost}(\texttt{G}, \texttt{C})$.

- We use the SP-score.

We will prove the following theorem.

Theorem (Jiang et al., STOC 2004)

For any instance $\mathcal{I}$ of MSA, $\frac{SA(\mathcal{I})}{\mathcal{OPT}(\mathcal{I})} \leq 2$.

Example

If for some instance $\mathcal{I}$ of MSA, $\mathcal{OPT}(\mathcal{I}) = 25$, then $SA(\mathcal{I}) \leq 50$.

Note: SA runs in time polynomial in $n = |S_1| + \cdots + |S_k|$.

The Star Alignment (SA) algorithm

$SA(\mathcal{I}) = \sum_{1 \le i < j \le k} d_{SA(\mathcal{I})}(S_i, S_j)$	
$= 1/2 \cdot \sum_{i=1}^{k} \sum_{j=1}^{k} d_{SA(\mathcal{I})}(S_i, S_j)$	by SP-score
$\le 1/2 \cdot \sum_{i=1}^{k} \sum_{j=1}^{k} [d_{SA(\mathcal{I})}(S_i, S_c) + d_{SA(\mathcal{I})}(S_c, S_j)]$	simplification
$= 1/2 \cdot \sum_{i=1}^{k} \sum_{j=1}^{k} [d_P(S_i, S_c) + d_P(S_c, S_j)]$	by triangle inequality
$= k/2 \cdot \sum_{i=1}^{k} d_P(S_i, S_c) + k/2 \cdot \sum_{j=1}^{k} d_P(S_c, S_j)$	S_c alignments are opt
$= k \cdot \sum_{i=1}^{k} d_P(S_i, S_c)$	split sum
	first result

$\mathcal{OPT}(\mathcal{I}) = \sum_{1 \le i < j \le k} d_{\mathcal{OPT}(\mathcal{I})}(S_i, S_j)$	
$\ge \sum_{1 \le i < j \le k} d_P(S_i, S_j)$	by SP-score
$= 1/2 \cdot \sum_{i=1}^{k} \sum_{j=1}^{k} d_P(S_i, S_j)$	$d_P \le d_{\mathcal{OPT}(\mathcal{I})}$
$\ge 1/2 \cdot \sum_{i=1}^{k} \sum_{j=1}^{k} d_P(S_c, S_j)$	simplification
$= k/2 \cdot \sum_{j=1}^{k} d_P(S_c, S_j)$	S_c has lowest possible score
	second result

We now have to combine the two results.

This will give us the final result.

The Star Alignment (SA) algorithm

$SA(\mathcal{I}) = \sum_{1\le i<j\le k} d_{SA(\mathcal{I})}(S_i,S_j)$	by SP-score
$= 1/2 \cdot \sum_{i=1}^{k}\sum_{j=1}^{k} d_{SA(\mathcal{I})}(S_i,S_j)$	simplification
$\le 1/2 \cdot \sum_{i=1}^{k}\sum_{j=1}^{k}[d_{SA(\mathcal{I})}(S_i,S_c) + d_{SA(\mathcal{I})}(S_c,S_j)]$	by triangle inequality
$= 1/2 \cdot \sum_{i=1}^{k}\sum_{j=1}^{k}[d_P(S_i,S_c) + d_P(S_c,S_j)]$	S_c alignments are opt
$= k/2 \cdot \sum_{i=1}^{k} d_P(S_i,S_c) + k/2 \cdot \sum_{j=1}^{k} d_P(S_c,S_j)$	split sum
$= k \cdot \sum_{i=1}^{k} d_P(S_i,S_c)$	first result

$\mathcal{OPT}(\mathcal{I}) = \sum_{1\le i<j\le k} d_{\mathcal{OPT}(\mathcal{I})}(S_i,S_j)$	by SP-score
$\ge \sum_{1\le i<j\le k} d_P(S_i,S_j)$	$d_P \le d_{\mathcal{OPT}(\mathcal{I})}$
$= 1/2 \cdot \sum_{i=1}^{k}\sum_{j=1}^{k} d_P(S_i,S_j)$	simplification
$\ge 1/2 \cdot \sum_{i=1}^{k}\sum_{j=1}^{k} d_P(S_c,S_j)$	S_c has lowest possible score
$= k/2 \cdot \sum_{j=1}^{k} d_P(S_c,S_j)$	second result

$$\begin{array}{l} SA(\mathcal{I}) \le k \cdot \sum_{i=1}^{k} d_P(S_i,S_c) \\ \mathcal{OPT}(\mathcal{I}) \ge k/2 \cdot \sum_{i=1}^{k} d_P(S_i,S_c) \end{array} \Longrightarrow \frac{SA(\mathcal{I})}{\mathcal{OPT}(\mathcal{I})} \le \frac{k\cdot\sum_{i=1}^{k} d_P(S_i,S_c)}{k/2\cdot\sum_{i=1}^{k} d_P(S_i,S_c)} = 2$$

This concludes the proof.