Multiple String Alignment

Asger Klinkby

June 2020

Contents

1 Introduction

This paper is an attempt to make research in the field of bioinformatics more accessible to students with a background in computer science.

In biology, many problems regarding DNA sequences and chains of amino acids can be cast as problems purely concerning strings of letters. This makes the field easily accessible to computer scientists. [1]

The problem of multiple string alignment is a good example. It is useful for finding patterns in strings of DNA, RNA, and amino acids. All three of which we abstract away, and instead just consider strings over some alphabet. The patterns we want to find are similarities in different strings, more precisely we want to align the strings such that substrings where each letter in one string is the same as in an other string become evident. These patterns can be used in finding what part of a sequence is critical to the function or structure encoded by the sequence.[1] We use multiple strings because they can reveal patterns not seen when considering only two strings. [2]

2 Basic definitions

2.1 Multiple alignment

Given k strings $s_1 \dots s_k$ we define a multiple alignment, or simply an *alignment* $\mathscr{A}$ to be a $k \times m$ matrix, where each row i corresponds to the string s'_i which is one of the original strings with some number of spaces inserted. The spaces are inserted as part of the alignment process. And the number of spaces is chosen so the strings all have equal length m. We will use the notation $s_i[n]$ to mean the n'th character of the i'th string. When writing the strings we denote a space "$-$", and the strings all consist of characters from $\Sigma' = \Sigma \cup \{-\}$, a finite alphabet Σ with the space character added. Columns with all spaces are assigned zero value by $v(\cdot)$ the value function.[2] [3]

$$k\left\{\overbrace{\begin{bmatrix} a & b & - & a & b \\ a & - & a & a & b \\ & & \vdots & & \\ b & b & - & - & b \end{bmatrix}}^{m}\right. \begin{array}{l} \}\, s'_1 \\ \}\, s'_2 \\ \\ \}\, s'_k \end{array}$$

Figure 1: An example of an alignment of a multiple alignemt $\mathscr{A}_1$

[1]Molecular strings can differ significantly throughout much of the string even though they are evolutionarily and functionally related [1].

[2]Alternatively columns with all spaces are not allowed

2.2 Pairwise alignment

A *pairwise alignment* $\mathscr{A}_{i,j}$ is formed by rows i and j from an alignment $\mathscr{A}$. We say that $\mathscr{A}$ *induce* $\mathscr{A}_{i,j}$.

Definition 2.1. *Induce*
Given a multiple alignment $\mathscr{A}$, and two strings s_i and s_j in $\mathscr{A}$. The alignment $\mathscr{A}_{i,j}$ of s_i and s_j *induced* by $\mathscr{A}$ is obtained by removing all other strings from the alignment. That is restricting the matrix to the rows i and j. Any opposing spaces may be removed.

A pairwise alignment can also be considered without any reference to a larger multiple alignment. We will use the notation (s_i, s_j) for such pairwise alignments.

We assign a *value* function $v(s_i[p], s_j[q])$ to the different pairs of opposing characters in a pairwise alignment, such that any particular pair of characters in Σ can have a unique value. We assume $v(\cdot)$ satisfy the *triangle inequality*[3], so for any three characters x, y, and z in Σ' it holds that $v(x,z) \leq v(x,y) + v(y,z)$.

We use this function, $v(\cdot)$, to describe the value of a pairwise alignment, $\mathrm{V}(\mathscr{A}_{i,j})$, which we define as $\sum_{n=1}^{m} v(s_i[n], s_j[n])$. The optimal (minimum) value of $\mathrm{V}(\mathscr{A}_{i,j})$ is denoted $\mathrm{D}(\mathscr{A}_{i,j})$.

We differentiate between, $\mathrm{V}(\mathscr{A}_{i,j})$, the value of pairwise alignment induced by an alignment and $\mathrm{V}(s_i, s_j)$ the alignment of strings s_i and s_j without any reference to a multiple alignment. This is important when considering the optimal alignments. The optimal alignment, $\mathrm{D}(\mathscr{A}_{i,j})$ is different from $\mathrm{D}(s_i, s_j)$ because of the constraints from the other strings in the multiple alignment. In fact $\mathrm{D}(\mathscr{A}_{i,j}) \geq \mathrm{D}(s_i, s_j)$.

2.3 Pairwise alignment problem

The **pairwise alignment problem** is: Find the pairwise alignment with the minimum value of $V(\mathscr{A}_{i,j})$. A dynamic programming solution is:

$$T(i,j) = min\Big(T(i-1,j-1)+v(s_1[i], s_2[j]),\ T(i-1,j)+v(s_1[i], "-"),\ T(i,j-1)+v("-", s_2[j]\Big)$$

If the length of s_1 and s_2 is n and m then the optimal value for the alignment of S_1 and S_2 is $T(n,m)$ and can be computed in $O(nm)$ time. The alignment can be found by using traceback in a dynamic programming table with pointers included. See page 217 in [1] for details.

2.4 Multiple alignment problem

The **multiple alignment problem** is: Find the multiple alignment with the minimum SP-score.

[3]This is somewhat reasonable, but there might be useful cost functions where triangle inequality does not hold

Definition 2.2. *Sum-of-pairs score*
The value of an alignment is simply defined as the sum of pairs of alignments:

$$\mathrm{V}(\mathscr{A}) = \sum_{i<j} \mathrm{V}(\mathscr{A}_{i,j})$$

This is called the *sum-of-pairs* score (SP-score) for an alignment.[4]

2.4.1 An exact solution

To find an exact or optimal alignment of k strings of length m dynamic programming can be used. The pseudo code for such a solution can be found on page 344 in [1]. However, the running time is $O(2^k \cdot m^k)$, which is impractical for most uses. Some improvements have been made to this result, but the exact SP-score alignment problem have been shown by [4] to be NP-complete[5]. [1]

3 Approximate solutions

Because exact solutions are infeasible, we will instead try to find approximate solutions. The rest of this paper will concentrate on finding good approximate solutions for the multiple alignment problem.

Though the solutions are approximate, they will have bounded error. For example, the first solution we will be looking at will give a SP-score at most twice the optimal. In working with these methods a large part of the work is in fiddling around with the spaces to get the alignments to fit together. This stems from the fact that an alignment is simply strings with spaces inserted at specific places.
Central to solutions we will look at is the concept of an alignment being *consistent* with a tree.

Definition 3.1. *Consistent*
For the strings $s_1 \dots s_k$ let each string label a node in a graph G, and let edges between nodes describe pairwise alignments. An alignment $\mathscr{A}(G)$ of the strings labeling G is *consistent* with the graph if:

$$\begin{gathered}\text{The nodes labeled } s_i \text{ and } s_j \text{ are adjacent in } G \\ \Updownarrow \\ \text{The alignemt of } s_i \text{ and } s_j \text{ induced by the alignment } \mathscr{A}(G) \text{ is optimal.}\end{gathered}$$

Using these definitions the following lemma describes the relation between trees and multiple alignments.

[4]In this paper we will only consider SP-score, though there are many other ways to evaluate multiple alignments. SP-score is maybe not the best measure, but it is easy to work with. [1]

[5]In this context it is interesting because it makes it highly unlikely that there exist a polynomial time solution to the exact problem. If we found a polynomial time solution for this problem, we would at the same time have found a solution for some of the deepest open questions in complexity theory. A short explanation of the reasoning behind this can be found in the appendix.

Lemma 3.1. *Given any tree T where each node is labeled with a distinct string, there is a multiple alignment $\mathscr{A}(T)$ of these strings which is consistent with T.*

Proof. We will construct the alignment $\mathscr{A}(T)$ inductively. First, align two adjacent strings optimally (using the solution described earlier).
For the induction step, assume that some number of strings labeling adjacent nodes are already aligned. Now align another string s_{new} that is adjacent to a string s already in the alignment. From the previous alignments there might have been inserted some number of spaces into s. We call the string s with potential spaces inserted s'. When aligning s_{new} and s' we might add additional spaces to s'. If we insert a space between the $n'th$ and $n+1'th$ character of s' we will insert a space between these two characters in all the other strings already aligned. Since opposing spaces have zero cost, $v("-","-")=0$, this will not affect the scores. By induction the alignemt $\mathscr{A}(T)$ can be constructed. [1] $\square$

If all the strings have length n it will take $O(n^2)$ time to compute each pairwise alignment. With k being the number of strings it will take $O(kn^2)$ time to find $\mathscr{A}(T)$

3.1 Center star method

The first method for finding alignments we are going to look at is the *center star method* first described in [2]. We will start by showing how to obtain an alignment consistent with a star graph. Then we will show that this alignment gives a SP-score at most twice the optimal.

3.1.1 Obtaining the alignment

A star graph is a tree with one central node, c, and $k-1$ leaves. Let all the nodes be labeled by strings $s_1 \dots s_k$. We will now find the string c which minimizes the cost of pairwise alignments with all the other nodes, that is the string c that minimizes $\sum_j \mathrm{D}(s_c, s_j)$. Choose this string as the label for the center node c, and label the other nodes with the remaining strings. We use M to denote this minimum value of $\sum_j \mathrm{D}(s_c, s_j)$ for the center node.

From lemma 3.1 we know how to construct an alignment, let us call it $\mathscr{A}^{star}$, which is consistent with a tree and hence the star.

3.1.2 Time complexity

First we have to find the optimal pairwise alignments of every pair of strings, this takes $O(k^2n^2)$. Then use these to find the string s_c that minimizes $\sum_j \mathrm{D}(s_c, s_j)$, which takes $O(k^2)$ time. The construction of the alignment using lemma 3.1 takes $O(kn^2)$. Thus the overall time to find the alignment $\mathscr{A}^{star}$ is $O(k^2n^2)$. [5]

3.1.3 Error bound

We will now study the relation between the score of the optimal alignment and the alignment obtained by the *center star method.* This can be described by the ratio $\frac{\mathrm{V}(\mathscr{A}^{star})}{\mathrm{V}(\mathscr{A}^{optimal})}$. Our goal is to show that this fraction is at most 2, meaning that the star method gives a SP-score at most twice the optimal.

Remember that $\mathrm{V}(\mathscr{A}) = \sum_{i<j} \mathrm{V}(\mathscr{A}_{i,j})$ and note that $\mathrm{V}(\mathscr{A}_{i,j}) \geq \mathrm{D}(\mathscr{A}_{i,j})$.

Lemma 3.2. *Assume we use a scoring function $v(\cdot)$ that satisfy the triangle inequality. Then the value of the pairwise alignment of s_i and s_j induced by $\mathscr{A}^{star}$ is smaller than the value of the optimal alignment of (s_i, s_c) and (s_c, s_j) combined.*

$$V(\mathscr{A}^{star}_{i,j}) \leq V(\mathscr{A}^{star}_{i,c}) + V(\mathscr{A}^{star}_{c,j}) = D(s_i, s_c) + D(s_c, s_j)$$

Proof. To prove the inequality, let x, y and z be characters from s_i, s_c and s_j respectively. Then the triangle inequality states that $v(x,z) \leq v(x,y) + v(y,z)$. The definition of $\mathrm{V}(\mathscr{A}_{i,j})$ is $\sum_{n=1}^{m} v(s_i[n], s_j[n])$, therefore the inequality holds. To prove the equality, note that the nodes labeled by s_i and s_c are adjacent. Since $\mathscr{A}^{star}$ is consistent with the star, this means that the pairwise alignment of adjacent nodes s_i and s_c induced by $\mathscr{A}^{star}$ is optimal. The same applies for s_j and s_c. Thus the equality holds.

□

Theorem 3.3. *The ratio between the score of the optimal alignment $\mathscr{A}^{optimal}$ and the alignment obtained by the center star method is at most 2. Actually:*

$$\frac{V(\mathscr{A}^{star})}{V(\mathscr{A}^{optimal})} \leq 2 - \frac{2}{k} < 2$$

Proof. Instead of working with $\frac{\mathrm{V}(\mathscr{A}^{star})}{\mathrm{V}(\mathscr{A}^{optimal})}$ we will consider an equal ratio $\frac{W(\mathscr{A}^{star})}{W(\mathscr{A}^{optimal})}$. We define $W(\mathscr{A}) \equiv \sum_{(i,j)} \mathrm{V}(\mathscr{A}_{i,j})$ where (i,j) is an ordered pair. Since $W(\mathscr{A}) = 2\mathrm{V}(\mathscr{A})$ we have $\frac{\mathrm{V}(\mathscr{A}^{star})}{\mathrm{V}(\mathscr{A}^{optimal})} = \frac{W(\mathscr{A}^{star})}{W(\mathscr{A}^{optimal})}$.

First we will show that the numerator, $W(\mathscr{A}^{star})$, is smaller than $2(k-1)M$ and then we show that the denominator, $W(\mathscr{A}^{optimal})$, is larger than kM.

$$\begin{aligned}
W(\mathscr{A}^{star}) &= \sum_{(i,j)} \mathrm{V}(\mathscr{A}^{star}_{i,j}) && \text{by definition of } W() \\
&\leq \sum_{(i,j)} \mathrm{D}(s_i, s_c) + \mathrm{D}(s_c, s_j) && \text{by lemma 3.2} \\
&\leq 2(k-1) \sum_{i} \mathrm{D}(s_i, s_c) && \forall i \ \mathrm{D}(s_i, s_c) \text{ or } \mathrm{D}(s_c, s_i) \text{ appear } 2(k-1) \text{ times.} \\
&= 2(k-1)M && M = \sum_{j} \mathrm{V}(s_c, s_j)
\end{aligned}$$

$$\begin{aligned}
W(\mathscr{A}^{optimal}) &= \sum_{(i,j)} \mathrm{V}(\mathscr{A}^{optimal}_{i,j}) && \text{by definition of } W() \\
&\geq \sum_{(i,j)} \mathrm{D}(s_i, s_j) && (1) \\
&= \sum_{i=0}^{k} \sum_{j} \mathrm{D}(s_i, s_j) && \text{since } (i,j) \text{ is ordered} \\
&\geq k \sum_{j} \mathrm{D}(s_c, s_j) && \text{since } \sum_j \mathrm{D}(s_c, s_j) \leq \sum_j \mathrm{D}(s_i, s_j) \\
&= kM && M = \sum_j \mathrm{D}(s_c, s_j)
\end{aligned}$$

(1) since the induced alignment is at best as good as the optimal pairwise alignment.

This gives us:

$$\frac{V(\mathscr{A}^{star})}{V(\mathscr{A}^{optimal})} = \frac{W(\mathscr{A}^{star})}{W(\mathscr{A}^{optimal})} \leq \frac{2(k-1)M}{kM} = \frac{2(k-1)}{k} = 2 - \frac{2}{k} \leq 2$$

□

3.1.4 Conclusion

The center star method thus gives us a way to efficiently find an alignment of a large number of strings. For k strings of length n we can find an alignment in $O(k^2n^2)$ time, and guarantee that the SP-score of the alignment is at most twice the optimal.

3.2 Generalizing the result

In the center star method we assembled optimal pairwise alignments into a multiple alignment. What if we instead assembled triples of optimally aligned strings into a large alignment. Or what about l optimally aligned strings assembled into a multiple alignment. Assembling optimal alignments of 2 strings we obtained an error bound on the SP-score of $2 - \frac{2}{k}$ times the optimal. With 3 strings we can obtain an error bound of $2 - \frac{3}{k}$. And generally with l strings we get an error bound of $2 - \frac{l}{k}$, as long as $l < k$.

To show this result in depth is beyond the scope of this report. So the next section only covers some parts of the argumentation. For a full proof see Bafna et. al. [3] and Pevzner [6].

3.3 The l-stars method

To describe the general solution we will consider a new structure called an l-star. It is a star where the leaves are replaced by cliques of l nodes. A *clique* is a subset of a graph where all nodes are adjacent. For each of these cliques one of the nodes are c the center node of the star. The number of cliques is $\frac{k-1}{l-1}$, the number of non-center nodes divided by the number of non-center nodes in each clique. An ordinary star can be thought of as a 2-star.

3.3.1 Combining clique alignments

When working with l-stars we want a way of getting an alignment, $\mathscr{A}^{l\text{-}star}$, from the alignments $\mathscr{A}_1, \cdots, \mathscr{A}_{\frac{k-1}{l-1}}$ of the cliques, such that $\mathscr{A}^{l\text{-}star}$ is compatible with all $\mathscr{A}_i$.

Definition 3.2. *Compatible*
Given an alignment $\mathscr{A}$ of $s_1 \dots s_k$ and an alignment $\mathscr{A}'$ on some subset of the strings. $\mathscr{A}$ is *compatible* with $\mathscr{A}'$ if they align the subset of strings the same way.

To do this we insert spaces in every $\mathscr{A}_i$ and then union them all. The union is straight forward since no clique contains the same strings, other than the string s_c. We only need to have the spaces in place in such a way that the line containing s_c match, and all the matrices specifying the alignments are of equal size.

Like in lemma 3.1 this is done by considering the number of spaces between any two characters $s_c[i]$ and $s_c[i+1]$ of the center string s_c, finding the maximum number of spaces between $s_c[j]$ and $s_c[j+1]$ in any of the alignments $\mathscr{A}_i$, and including that number of spaces between $s_c[i]$ and $s_c[i+1]$ in every $\mathscr{A}_i$. [3]

$$l\left\{\overbrace{\begin{bmatrix} a & b & - & a & b \\ a & - & a & a & b \\ & & \vdots & & \\ b & b & - & - & b \end{bmatrix}}^{m_i}\right.\begin{matrix} \}\, s_1 \\ \}\, s_2 \\ \\ \}\, s_5 \end{matrix}$$

Figure 2: An example of an alignment of a clique $\mathscr{A}_1$

$$l\left\{\overbrace{\begin{bmatrix} - & a & b & a & b \\ b & a & b & - & b \\ & & \vdots & & \\ b & a & - & a & - \end{bmatrix}}^{m_i}\right.\begin{matrix} \}\, s_1 \\ \}\, s_6 \\ \\ \}\, s_9 \end{matrix}$$

Figure 3: An example of an alignment of a clique $\mathscr{A}_2$

$$\begin{bmatrix} - & a & b & - & a & b \\ - & b & a & b & - & b \\ - & & \vdots & & & \\ - & b & a & - & a & - \\ b & a & b & - & - & b \\ & & \vdots & - & & \\ b & a & - & - & a & - \end{bmatrix}\begin{matrix} \}\, s_1 \\ \}\, s_2 \\ \\ \}\, s_5 \\ \}\, s_6 \\ \\ \}\, s_9 \end{matrix}$$

Figure 4: An example of alignment $\mathscr{A}$ the union of $\mathscr{A}_1$ and $\mathscr{A}_2$ with spaces added

3.3.2 Weighted edges

To prove the error bound of $2 - \frac{k}{l}$ we will consider graphs with weighted edges. Since the edges describe pairwise alignments, this amounts to multiple alignments where the induced pairwise alignments count differently towards the SP-score.

We will need to describe the values of all the pairwise alignments induced by a multiple alignment as a $k \times k$ matrix. For this we will use the notation $S(\mathscr{A})$, and define $S(\mathscr{A}) \equiv [\mathrm{V}(\mathscr{A}_{i,j})]$. Now we can describe the SP-score as $E \cdot S(\mathscr{A})$, where E is the unit matrix, a $k \times k$ matrix with all $1's$ except $0's$ at the diagonal.

Now consider C(G) a $k \times k$ matrix describing the weight of each edge in the graph G. Its entries are described by the function c_{ij} wich we define as:

$$c_{i,j} = \begin{cases} k - (l-1) & \text{if } i = c \vee j = c \\ 1 & \text{if } i, j \neq c \wedge i \text{ is in the same clique as } j \\ 0 & \text{otherwise (non-edges)} \end{cases}$$

This function attributes $k - (l - 1)$ weight to all edges connected to the center node and 1 weight to all other edges in the star.

We can now put forth an equation describing the total weight of the l-star:

$$C(G)E = \underbrace{(k - (l-1))(k-1)}_{\text{center edges}} + \underbrace{1 \cdot \frac{k-1}{l-1}\binom{l-1}{2}}_{\text{clique edges}}$$

It is simply the weight of the edges connected to the center times the number of such edges, and the weight remaining edges times the number of remaining edges. The number of remaining edges is calculated as the number of cliques times the number of edges in a clique (excluding edges to the center). [3] In the appendix we show that this equation can be rewritten as $\binom{k}{2}(2 - \frac{l}{k})$.

Now for this matrix $C(G)$ the following lemma holds, and is proven in [6].

Lemma 3.4. *Given an l-star, G, and any alignment $\mathscr{A}$ of $s_1 \dots s_k$:*

$$ES(\mathscr{A}) \leq C(G)S(\mathscr{A})$$

This can be used to show that there exist an alignment compatible with the cliques of G that is optimal when weighted by $C(G)$, and that this alignment can be obtained by combining optimal alignments of the cliques.

3.3.3 Balanced sets of l-stars

We define a balanced set of l-stars $\mathscr{G}$ to be a collection of l-stars with the property $\sum_{G \in \mathscr{G}} C(G) = pE$ that the sum of all weight matrices should be equal to the unit matrix E multiplied by some scalar p.

The following lemma is proven in [3].

Lemma 3.5. *If $\mathscr{G}$ is a balanced set of l-stars then:*

$$\min_{G\in\mathscr{G}}(C(G)\cdot S(\mathscr{A}_G)) \leq \frac{p}{|\mathscr{G}|}\min_{\mathscr{A}}(E\cdot S(\mathscr{A}))$$

3.3.4 Obtaining the alignment

We are now ready to describe a procedure for constructing multiple alignments using a balanced set of l-stars:

Algorithm 1: Align [3]

```
Result: An alignment
Construct a balanced set of $l$-stars, $\mathscr{G}$
for each $l$-star $G \in \mathscr{G}$ do
    Assemble an alignment $\mathscr{A}_G$ that is optimal with respect to C(G)
      from alignments that are optimal for each of its cliques
Choose $G$ with the corresponding alignment $\mathscr{A}_G$ such that
  $C(G) \cdot S(\mathscr{A}_G)$ is the minimum over all $l$-stars in $\mathscr{G}$.
return $\mathscr{A}_G$
```

This algorithm is polynomial in l and k [3] and will return an alignment with an error bound of $2 - \frac{l}{k}$ of the optimal.

Theorem 3.6. *Given a balanced set of l-stars $\mathscr{G}$, Align will return an alignment with an error bound of $2 - \frac{l}{k}$ of the optimal.*

Proof. Align returns an alignment $\mathscr{A}_G$ that minimizes $C(G)\cdot S(\mathscr{A}_G)$, and is optimal for l-star $G \in \mathscr{G}$.

From lemma 3.4 and 3.5 and the fact that $\frac{p}{|\mathscr{G}|} = \frac{C(G)\cdot E}{E\cdot E} = 2 - \frac{l}{k}$ we get the inequality:

$$E\cdot S(\mathscr{A}_G) \leq C(G)\cdot S(\mathscr{A}_G) \leq (2-\frac{l}{k})\cdot ES(\mathscr{A}^{optimal})$$

□

3.3.5 Conclusion

We have now generalized the result from the center star method. Through a sketch of the l-star method we have seen that an error bound of $2 - \frac{l}{k}$ is attainable in polynomial time.

4 Critique

In order to make the problem of finding biologically interesting patterns in sequences of DNA and chains of amino acids attainable, we had to make some assumptions. A central assumption was that we can design a cost function where the alignment with the optimal score is also the alignment best describing the biological relation between the sequences being aligned. The definition of SP-score is mathematically motivated and does not necessarily make good biological sense. When using approximation algorithms we also rely on what we could call *smoothness*: That solutions that are close to optimal also have a close biological relation to the optimal solution. We would, therefore, need to argue that the SP-score cost function has this property.

It remains an open problem in the field to find a fully automatic way to compute multiple string alignments that properly represent the different biological relations between sequences. There are many challenges in finding a way to describe the problem mathematically in a way that captures the biology we are trying to represent. [7]

5 Conclusion

This paper was an attempt to make research in the field of bioinformatics more accessible to students with a background in computer science. We saw how a biological problem regarding DNA sequences and chains of amino acids, could be cast as problems purely concerning strings of letters. But also how some of the assumptions we made to make the problem attainable, weakened the biological relevance of our results.

We described the general framework for multiple string alignment and reasoned about alignments, the multiple alignment problem, and approximation methods. We used a bounded error approximation method to give an efficient approximate solution to a NP-complete problem. We looked in detail at the center star method with an error bound of $2 - \frac{2}{k}$ and running time $O(k^2n^2)$. And at a sketch of the generalization to l-stars giving an error bound of $2 - \frac{l}{k}$, still in polynomial time.

It is an open problem to find more efficient methods or methods with a smaller error bound. But as mentioned previously another direction would be to find a more biologically relevant measures of alignment to optimize.

6 Appendix

6.1 Short description of NP-completeness

If you are interested in, but not yet familiar with complexity theory, a short explanation of NP-completeness is as follows.

NP means non-deterministic polynomial time and refers to the set of problems that can be solved i polynomial time by a non-deterministic Turing machine. That this problem is NP-complete means that this problem is in NP and that every problem in NP can be reduced to this problem. P is the set of problems that can be solved in (deterministic) polynomial time. One of the deep open questions in complexity theory is if $P = NP$. If a NP-complete problem is also in P, then because every problem in NP can be reduced to this problem, we would have $NP \subseteq P$. And by definition $P \subseteq NP$. Thus $P = NP$.

6.2 The equation describing total edge weight

We show how the equation describing combined weight of all edges in an l-star can be rewritten.

$$\begin{aligned}
C(G)E &= (k-(l-1))(k-1) + 1 \cdot \frac{k-1}{l-1}\binom{l-1}{2} \\
&= (k-(l-1))(k-1) + \frac{k-1}{l-1}\frac{(l-1)(l-2)}{2} \\
&= (k-(l-1))(k-1) + \frac{(k-1)(l-2)}{2} \\
&= \frac{2(k-(l-1))(k-1) + (k-1)(l-2)}{2} \\
&= \frac{(k-1)(2(k-(l-1)) + (l-2))}{2} \\
&= \frac{(k-1)(2k-2l+2+l-2)}{2} \\
&= \frac{(k-1)(2k-l)}{2} \\
&= \frac{2k(k-1) - l(k-1)}{2} \\
&= 2\frac{k(k-1)}{2} - \frac{l(k-1)}{2} \\
&= 2\frac{k(k-1)}{2} - \frac{k(k-1)}{2}\frac{l}{k} \\
&= 2\binom{k}{2} - \binom{k}{2}\frac{l}{k} \\
&= \binom{k}{2}(2 - \frac{l}{k})
\end{aligned}$$

References

[1] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, USA, 1997. Primarily Chapter 14.

[2] Dan Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bulletin of Mathematical Biology*, 55(1):141–154, 1993.

[3] Vineet Bafna, Eugene L. Lawler, and Pavel A. Pevzner. Approximation algorithms for multiple sequence alignment. *Theoretical Computer Science*, 182(1):233 – 244, 1997.

[4] Lusheng Wang and Tao Jiang. On the complexity of multiple sequence alignment. *Journal of computational biology*, 1(4):337–348, 1994.

[5] Wing-Kin Sung. Multiple sequence alignment. `https://www.comp.nus.edu.sg/~ksung/algo_in_bioinfo/slides/Ch6_MSA.pdf`, 2009.

[6] Pavel A. Pevzner. Multiple alignment, communication cost, and graph matching. *SIAM Journal on Applied Mathematics*, 52(6):1763–1779, 1992.

[7] David Morrison. Multiple sequence alignment is not a solved problem. *Department of Organismal Biology, Uppsala University*, 08 2018.