A Computational Model for RNA Multiple Structural Alignment

Eugene Davydov and Serafim Batzoglou

Dept. of Computer Science, Stanford University,
Stanford CA 94305, USA,
{edavydov,serafim}@cs.stanford.edu

Abstract. This paper addresses the problem of aligning multiple sequences of non-coding RNA genes. We approach this problem with the biologically motivated paradigm that scoring of ncRNA alignments should be based primarily on secondary structure rather than nucleotide conservation. We introduce a novel graph theoretic model (NLG) for analyzing algorithms based on this approach, prove that the RNA multiple alignment problem is NP-Complete in this model, and present a polynomial time algorithm that approximates the optimal structure of size $S$ within a factor of $O(\log^2 S)$.

1 Introduction

Noncoding RNA (ncRNA) genes are among the biologically active features in genomic DNA. They are polymers of four nucleotides: A (adenine), C (cytosine), G (guanine), and U (uracil). Unlike regular genes, ncRNAs are not translated into protein, but rather fold directly into secondary and tertiary structures, which can have a variety of structural, catalytic, and regulatory functions [6].

The structural stability and function of ncRNA genes are largely determined by the formation of stable secondary structures through complementarity of nucleotides, whereby G-C, A-U, and G-U form hydrogen bonds that are energetically favored. This secondary structure can be predicted from the nucleotide sequence as one minimizing (some approximation of) the free energy [18, 19], which is largely determined by the formation of the hydrogen bonds. In ncRNAs, such bonds almost always occur in a nested fashion, which allows the optimal structure to be computed in time $O(n^3)$ in the length of the input sequence using a dynamic programming approach [13, 11]. Algorithms that do not assume a nested structure are even more computationally expensive [15]. However, the stability of ncRNA secondary structures is not sufficiently different from the predicted stability of random genomic fragments to yield a discernible statistical signal [16], limiting the application of current ncRNA detection methods to the simplest and best understood structures, such as tRNAs [12].

One of the most promising ways of detecting ncRNA genes and predicting reliable secondary structures for them is comparative sequence analysis. During the course of genome evolution, mutations that occur in functional regions of
the genome tend to be deleterious, and therefore unlikely to fix, while mutations that occur in non-functional regions tend to be neutral and accumulate. As a result, functional regions of the genome tend to exhibit significant sequence similarity across related genomes, whereas regions that are not functional are usually much less conserved. This difference in the rate of sequence conservation is used as a powerful signal for detecting protein-coding genes \cite{1, 2} and regulatory sites \cite{14, 10}, and could be applied to ncRNA genes. However, their function is largely determined by their secondary structure, which in turn is determined by nucleotide complementarity: RNA genes across different species are similar in the pattern of nucleotide complementarity rather than in the genomic sequence. As a result, conventional sequence alignment methods are not able to properly align ncRNAs \cite{7}.

One biologically meaningful approach to ncRNA multiple alignment is finding the largest secondary structure common to all the sequences, lining up the nucleotides forming this structure, and then aligning corresponding leftover pieces as one would align genomic sequences which have no evolutionary pressure favoring complementary substitution. However, this approach has never been applied in practice because the task of finding the largest common secondary structure among several sequences is computationally challenging: the straightforward extension of the dynamic programming algorithm using stochastic context-free grammars (SCFGs) has a running time of $O(n^{3k})$, where $k$ is the number of sequences being aligned, which is prohibitive even for two sequences of moderate length \cite{9}.

The problem of aligning multiple RNA sequences has been proven to be NP-Complete for certain scoring schemes and metrics \cite{17, 3}. However, when analyzing the computational complexity of ncRNA multiple alignment, it’s more relevant to focus on the complexity of finding the largest common secondary structure, because for most biologically meaningful ncRNAs the remaining pieces should be relatively short and easy to align.

In this paper we introduce a novel theoretical framework for analyzing the problem of ncRNA multiple structural alignment. We present the Nested Linear Graph (NLG) model and formulate the problem of computing the largest common secondary structure in this model in terms of finding the largest common nested subgraph. We then prove this problem to be NP-Complete, and present a polynomial-time algorithm which approximates the optimal solution within a factor of $O(\log^2 S)$, where $S$ is the size of the optimal solution. We conclude with a discussion of the NLG model in general and our algorithm and results in particular.

2 A Graph Theoretic Formulation

A linear graph is a graph whose vertices, $V$, are points on some line $L$. Genomic sequences naturally yield themselves to linear graph representations, because each of their nucleotides can correspond to a point, and the sequence can correspond to the line. For modeling ncRNA folding and secondary structure, we form
the linear graph with edges connecting pairs of vertices that represent complementary nucleotide pairs (A-U, C-G, and G-U). A typical linear graph induced by an RNA sequence is shown in fig. 1.

![Fig. 1. A linear graph representation of the RNA sequence UACGUG. The nucleotides are represented by points on a line in the same order as in the sequence. Each edge is represented by an arc to emphasize that it does not pass through the nodes between its two endpoints. Edges are drawn between nodes representing complementary nucleotide pairs A-U, C-G, and G-U.](image)

Two edges \( \overline{ab} \) and \( \overline{cd} \) of a linear graph intersect if exactly one of \( c \) and \( d \) lies on the line segment \( \overline{ab} \) (and vice versa). A linear graph is nested if no two edges of the graph intersect each other. For a linear graph derived from an RNA sequence, a nested subgraph represents a plausible fold of that sequence. Thus, in the NLG model, the problem of finding the largest secondary structure of an ncRNA is precisely the problem of finding the largest nested subgraph in the linear graph derived from the sequence. For multiple ncRNA alignment, where we seek the largest common secondary structure, the appropriate NLG formulation is finding the largest common nested linear subgraph (MAX-NLS) among the linear graphs induced by the sequences (see fig. 2). We now formulate this problem precisely and formally analyze its computational complexity.

![Fig. 2. The MAX-NLS of several linear graphs; its edges have been emphasized in bold to distinguish them from the edges of the original linear graphs. Note that the MAX-NLS is not necessarily unique, but its size is.](image)

### 3 Complexity Analysis of MAX-NLS

Let \( G_1, \ldots, G_m \) be the linear graphs derived from ncRNA sequences \( S_1, \ldots, S_m \) respectively. The MAX-NLS of these graphs is the largest nested graph \( G_c \) such that \( G_c \) is a subgraph of \( G_i \) for all \( i = 1, \ldots, m \). For any problem instance \( I \) = \( \{G_1, \ldots, G_m\} \), we write MAX-NLS(I) to indicate this \( G_c \). To represent the size (number of edges) of this graph, we use the notation \( |\text{MAX-NLS}(I)| \).
Note that the MAX-NLS problem represents a slight generalization of the RNA multiple alignment problem, in that we do not constrain the linear graphs to be derived from RNA strings by connecting every pair of nodes corresponding to complementary nucleotides with an edge. We motivate this relaxation in the discussion section of the paper.

The MAX-NLS is an optimization problem, because our objective is to maximize the size of the common nested subgraph of \( \{ G_1, \ldots, G_m \} \). We now formulate the corresponding decision problem, where our objective is to answer a boolean query.

**Definition 1.** The NLS decision problem (D-NLS) is to determine, given an input \( G_1, \ldots, G_m \) and a positive integer \( k \) (where \( 1 < k < \min_i |G_i| \)), whether there exists a common nested linear subgraph of \( G_1, \ldots, G_m \) of size \( \geq k \).

**Theorem 1.** D-NLS is NP-Complete.

*Proof (of Theorem 1).*

We proceed by demonstrating a polynomial reduction from 3-SAT, a well-known NP-Complete problem [4].

**Definition 2.** Let \( x_1, \ldots, x_k \) be boolean variables. Let \( \psi_1, \ldots, \psi_n \) be logical clauses, with each clause \( \psi_i \) being a disjunction of 3 literals, where each literal is either a variable \( x_j \) or its negation, \( \neg x_j \). The 3-SAT problem is to determine, given this as input, whether there exists an assignment for the \( k \) variables which satisfies all \( n \) clauses simultaneously.

To establish the reduction we need to demonstrate that the existence of a polytime algorithm for D-NLS yields a polytime algorithm for 3-SAT. As such, we show that given any input instance \( I_{3-SAT} \) and a polytime algorithm \( A \) for D-NLS, we can construct, in polynomial time and space, an instance \( I_{D-NLS} \) such that computing \( A(I_{D-NLS}) \) will allow us to answer whether the instance \( I_{3-SAT} \) is satisfiable. However, to simplify the description of this construction, we must define the notion of a c-thick edge (see fig. 3).

![Fig. 3. A 4-thick edge intersecting a 5-thick edge. Any edge not shown must intersect either all edges in either stack, or none at all.](image)

**Definition 3.** In a linear graph, an edge \( \hat{ab} \) contains an edge \( \hat{cd} \) if and only if both \( c \) and \( d \) lie strictly between \( a \) and \( b \) on the line. \( \hat{ab} \) directly contains \( \hat{cd} \) whenever \( \hat{ab} \) contains \( \hat{cd} \) and there is no other edge \( e \) that contains \( \hat{cd} \) but is itself contained by \( \hat{ab} \).

A c-thick “edge” in a linear graph is a set of \( c \) edges \( e_1, \ldots, e_c \) with the properties that:
(i) for all $i, j$ such that $i > j$, $e_i$ contains $e_j$
(ii) for any other edge $e'$, either $e'$ intersects all $e_i$, or it intersects none of them

We can now describe the construction of $I_{D-NLS}$ from $I_{3-SAT}$, as depicted in fig. 4. Given the set of variables $x_1, \ldots, x_k$ and the clauses $\psi_1, \ldots, \psi_n$, we construct $k + 1$ linear graphs: one corresponding to each boolean variable in $I_{3-SAT}$, and an extra graph $x'$ whose purpose will be clarified shortly. Each of the $n$ graphs consists of two intersecting $c_1$-thick edges, each of which contains a sequence of $k$ similar groups of edges, where each group corresponds to a particular clause $\psi_i$. Such a group is depicted in detail at the bottom of fig. 4.

**Fig. 4.** Constructing an instance of D-NLS from an instance of 3-SAT. Each variable $x_j$ gives rise to a linear graph $x_j$, which consists of two intersecting $c_1$-thick edges, each of which contains $n$ edge groups corresponding to clauses $\psi_i$. Every clause group consists of 3 $c_2$-thick edges in sequence, as well as up to 3 mutually intersecting selection edges, which are present if $\psi_i$ does not depend on $x_j$, or the truth value induced upon $x_j$ by the label of the $c_1$ edge makes $\psi_i$ TRUE. Finally, $I_{D-NLS}$ contains an extra linear graph $x'$, consisting of only one $c_1$ edge, which contains the standard collection of $3n c_2$ edges, as well as all possible selection edges. The goal of the $x'$ graph is to force an alignment where every other graph $x_j$, $x'$ aligns to either the TRUE or FALSE portion of $x_j$, thus corresponding to a truth assignment to all variables of the 3-SAT problem.

The edge group varies slightly depending on which $x_j$ and $\psi_i$ it corresponds to. The portion common to all such groups consists of three $c_2$-thick edges none of which intersects or contains the other. Beyond these, each group has up to three of the following set of mutually intersecting edges: an edge that contains the first and second $c_2$-thick edges, an edge that contains only the second, and an edge that contains the third, as illustrated in fig. 4. An edge is missing from the group only if the corresponding literal in the clause $\psi_i$ is not in agreement with the truth assignment induced by the $c_1$ edge to $x_j$. To be more precise, let $\psi_i = \eta_a x_a \lor \eta_b x_b \lor \eta_c x_c$, where $a \leq b \leq c$, and the corresponding $\eta$ can either be the identity or negation $\neg$. The edge corresponding to $\eta_a x_a$ is absent if and only if $a = j$ and $\eta_a x_a$ is false under the truth assignment induced by the $c_1$ edge.
of \( x_j \). If \( a = b \) and \( \eta_a = -\eta_b \), the edge is present; since if a clause contains the disjunction \( x_j \lor \neg x_j \), it is automatically satisfied and the edge should exist.

The \( k + 1 \) graph consists of only one \( c_1 \) edge, and \( n \) clause groups each of which contains all 3 selector edges in addition to the \( 3c_2 \) edges. The basic premise of this construction is that if (and only if) there is a satisfying assignment, we will be able to match the \( x' \) graph to the corresponding \( c_1 \) edge in each of the \( k \) graphs, and align the \( n \) clause groups within. Only because the assignment is satisfying will we be able to align one additional selector edge from every clause, giving us the largest possible common subgraph.

**Lemma 1.** Let \( c_2 = n + 1 \) and let \( c_1 = 3n^2 + 4n + 1 \). Under the scheme described above, the \( k + 1 \) linear graphs have a common nested subgraph of size \( 6n^2 + 8n + 1 \) if and only if \( \psi_1, \ldots, \psi_k \) are satisfiable.

**Proof (of Lemma 1).**

Suppose the clauses are satisfiable, that is, there exists some assignment to \( x_1, \ldots, x_k \) which satisfies them all. We align the \( c_1 \) edge of the \( x' \) graph with the \( c_1 \) edge of graph \( j \) that corresponds to the value of \( x_j \) in this truth assignment. We then align the \( c_2 \) edges to each other. Now consider a particular selector edge in some clause \( \psi_i \). Because of the way we aligned the \( c_1 \) edges, if this edge is absent in any of the half-graphs we selected, it is because its corresponding literal is false in that clause given the truth assignment. However, since we assumed our assignment is satisfying, every clause must have a literal that evaluates to \( \text{TRUE} \). The corresponding selector edge must be present in every graph.

We can choose at most one selector edge per clause, since they all intersect each other. Because we can choose one from every clause, we have a total of \( c_1 + 3n c_2 + n = 6n^2 + 8n + 1 \).

Now suppose we indeed have a common nested subgraph of size \( 6n^2 + 8n + 1 \). As there are a total of \( 6nc_2 c_2 \) edges and up to \( 2n \) selector edges that may be chosen simultaneously, we could only have \( 6n^2 + 8n \) edges without choosing a \( c_1 \) edge. Thus, we must align a \( c_1 \) edge, in which case we might as well align the whole stack of them. That leaves \( 3n^2 + 4n \) edges to be included. Note that each \( c_2 \) stack contributes more than the selector edges could simultaneously, so we must choose all \( 3n \) \( c_2 \) stacks for a total of \( 3n^2 + 3n \) edges. This leaves \( n \) edges to be accounted for, all of which must be selector edges, one from each clause.

Note that the \( c_1 \) alignment we choose induces a truth assignment to our variables. As we just showed, the size of our alignment implies not only that the \( c_1 \) and \( c_2 \) edges are aligned, but also that under this truth assignment, every clause has a selector edge that is present in every graph’s chosen \( c_1 \) half. In particular, that edge is present in the graph corresponding to its literal, meaning that under this induced truth assignment, the clause is satisfied because the literal is \( \text{TRUE} \). Since this applies to all the clauses, \( \psi_1, \ldots, \psi_k \) are all satisfied.

\( \Box \)

The time required for this construction is \( O(kn^2) \); thus, we have demonstrated a polynomial reduction to D-NLS from 3-SAT, and D-NLS is NP-Complete.

\( \Box \)
4 Approximating MAX-NLS with MAX-FLS

In view of Theorem 1 there is little hope for a tractable exact algorithm for MAX-NLS. Therefore, we present a polynomial time approximation algorithm that guarantees optimality within a factor of $O(\log^2 S)$, where $S$ is the size of the optimal solution. The polynomial time is achieved by restricting attention to a subclass of nested linear graphs and finding the optimal member of this restricted subclass. The main tradeoff here is the choice of the restriction: if the subclass is too narrow, our approximation will be poor; otherwise, finding the optimal member of the subclass may still be NP-Complete.

The restriction that yields our algorithm is best presented as a composition of two restrictions. First, we consider the subclass of NLGs that are flat.

**Definition 4.** A branching edge in a nested linear graph is an edge $e$ that contains two edges $e_1$ and $e_2$, neither of which contains the other. A nested linear graph is flat if it contains no branching edges. The flat order of a nested linear graph is the size of its largest flat subgraph.

The optimization problem corresponding to this restriction is that of finding the largest common flat nested linear graph (MAX-FLS). We now show that this restriction yields a solution that is suboptimal by a factor of at most $O(\log S)$.

**Theorem 2.** Every nested linear graph $G$ with flat order $F_G$ satisfies $|G| \leq F_G \log(F_G)$.

**Proof (of Theorem 2).**

We begin by introducing the tree representation of nested linear graphs in order to relate the main notions of our argument to familiar data structures. The basic transformation is mapping each NLG edge to a node in the tree, as shown in fig. 5. We first add an edge containing the entire NLG, for the sake of uniformity. We then construct the tree by mapping each edge $e_i$ to a tree node $n_i$. A node $n_i$ is a parent of another node $n_j$ whenever its corresponding edge $e_i$ directly contains $e_j$ (see Definition 3 for the notion of direct containment). While this transformation is rather elementary, it affords us insights into the notion of flat order. Noting that the notion of a branching edge in an NLG corresponds precisely to a branching node in the tree, we observe the following:

(i) When viewed as a subtree, the path from the root to any leaf contains no branching nodes and is therefore flat. Thus, the flat order $F_T$ satisfies $F_T \geq h(T)$, where $h(T)$ is the node height of $T$ (number of nodes in the longest root-leaf path),

(ii) Consider any disjoint subtrees of $T$ satisfying the property that nodes in different subtrees cannot be descendants or ancestors of each other in $T$. The union of their flat subtrees will also be flat, as no branching nodes can be introduced by taking the union of flat constituents that have no ancestor relationships amongst one another. Consequently, for any split node in the tree, the sum of the flat orders of its subtrees is $\leq F_T$. 
We now examine an arbitrary tree $T_n$ with flat order $n$. We show that $|T_n| \leq n \log(n) + 1$. We establish the general result by strong induction: assuming the formula holds for every $n' < n$, we show that it holds for $n$. We enumerate the required base cases in figure 7.

Each tree can be represented as an initial trunk of length $\ell \geq 0$, followed by a split into some number of subtrees. Among these we then consider the subtree with the largest flat order. If its flat order is $> n/2$, we recursively divide that subtree into a trunk, a splitting node, and the subtrees at that node. We continue this process until no subtree has flat order $> n/2$, as shown in fig. 6. Note that there can only be one subtree with flat order $> n/2$, so we will never have to subdivide more than one subtree at each level.

We can now write the formula for the number of nodes in $T_n$. From the diagram,

$$|T_n| = \sum_{i=1}^{k} |T_{a_i}| + |T_{b_i}| + \sum_{j=1}^{\ell} \ell_j.$$
By the inductive assumption $|T_{a_i}| \leq a_i \log(a_i) + 1$ and $|T_{b_r}| \leq b_r \log(b_r) + 1$, so

$$|T_n| \leq \sum_{i=1}^{k_r} (a_i \log(a_i) + 1) + (b_r \log(b_r) + 1) + \sum_{j=1}^{r} \ell_j.$$

By construction, all $a_i$ and $b_r$ are $\leq n/2$. Furthermore, since $n \geq \sum_{i=1}^{k_r} a_i + b_r$, at most 3 of $\{a_1, \ldots, a_{k_r}, b_r\}$ may be $> n/4$. When $a_i \leq n/4$, $a_i \log(a_i) + 1 \leq a_i \log(2a_i) \leq a_i \log(n/2)$, similarly for $b_r$. Thus,

$$|T_n| \leq \sum_{i=1}^{k_r} a_i \log(n/2) + b_r \log(n/2) + 3 + \sum_{j=1}^{r} \ell_j.$$

To prove that this implies $|T_n| \leq n \log(n) + 1$, we now consider 3 cases:

1. $h(b_r) \geq 2$

Then, according to observation (i), $n \geq \sum_{j=1}^{r} \ell_j + h(b_r) \geq \sum_{j=1}^{r} \ell_j + 2$, therefore,

$$|T_n| \leq \log(n/2) b_r + \sum_{i=1}^{k_r} a_i + 1 + n \leq n \log(n/2) + n + 1 = n \log(n) + 1.$$

2. $h(b_r) = 0$

Then $T_{b_r}$ has no nodes, and since by construction it is the largest subtree in its level, it must be that the splitting node at the bottom of trunk $\ell_r$ has no children. This means that either the entire tree is a single trunk, in which case $|T_n| = n \leq n \log(n) + 1$, or that $\ell_r > n/2$, since we had to subdivide $T_{b_{r-1}}$. In this case, we have $|T_n| \leq \sum_{i=1}^{k_r-1} (a_i \log(a_i) + 1) + \sum_{j=1}^{r} \ell_j$. Since $a_i \leq n/2$ by construction, we have $a_i \log(a_i) + 1 \leq a_i \log(2a_i) \leq a_i \log(n)$, and therefore $|T_n| \leq \log(n) \sum_{i=1}^{k_r-1} a_i + \sum_{j=1}^{r} \ell_j$, which transforms to $|T_n| < (n/2) \log(n) + n$ since observation (ii) implies $\sum_{i=1}^{k_r-1} a_i \leq n - \ell_r < n/2$. Finally, since $n \leq (n/2) \log(n)$ for $n \geq 4$, we have

$$|T_n| < n \log(n).$$

3. $h(b_r) = 1$

In this case $T_{b_r}$ consists of a single node, so $b_r = 1$. We may now write $|T_n| \leq \log(n/2) \sum_{i=1}^{k_r} a_i + 3 + 1 \log(1) + \sum_{j=1}^{r} \ell_j$, since at most 3 elements of $\{a_i\}$ may be $> n/4$. Noting that 1 = $b_r \log(2) \leq b_r \log(n/2)$ as long as $n \geq 4$, we have $|T_n| \leq \log(n/2) (b_r + \sum_{i=1}^{k_r} a_i) + 1 + \sum_{j=1}^{r} \ell_j$. Applying the results of observations (i) and (ii), we have the familiar inequalities $b_r + \sum_{i=1}^{k_r} a_i \leq n$ and $1 + \sum_{j=1}^{r} \ell_j \leq n$, yielding

$$|T_n| \leq n \log(n/2) + n + 1 = n \log(n) + 1.$$
The assumption $n \geq 4$ can be eliminated by noting that the largest trees with flat order $< 4$ still obey the equation. These trees are shown in fig. 7.

Thus, for an arbitrary tree $T$ with flat order $F_T$, $|T| \leq F_T \log(F_T) + 1 = O(F_T \log(F_T))$, which is precisely the statement of the theorem for nested flat graphs.

It is noteworthy to observe that this bound is asymptotically tight. Consider the family of trees $T_i$ defined recursively as:

- $T_0$ = a single node.
- $T_{i+1}$ = a trunk of length $2^i$ nodes, which splits into two subtrees $T_i$, as shown in fig. 8.

By induction, it is clear that both the height and the flat order of $T_i$ are equal to $2^i$. The number of nodes is defined by the recurrence $|T_{i+1}| = 2|T_i| + 2^i$, the solution to which is $|T_i| = 2^{i-1}(i + 2)$. Thus, for any tree $T$ of this family,

$$|T| = (1/2)F_T(2 + \log(F_T)) = \Theta(F_T \log(F_T)).$$

\section{Approximating MAX-FLS with MAX-LLS}

We now further restrict the subclass of NLGs to examine by introducing the notion of \textit{level} flat graphs, and the corresponding optimization problem \textbf{MAX-LLS}. First, however, we prove a useful property of flat linear graphs.

\textbf{Theorem 3.} Any flat nested linear graph $G$ can be written as a union of $k \geq 0$ disjoint subsets, $G = \bigcup_{i=1}^{k} C_i$, where each $C_i$ is a column of edges, i.e. a $|C_i|$-thick edge.
Proof (of Theorem 3).
Consider any edge \( e \in G \), and let \( E \) be the set of edges that either contain or are contained by \( e \). Because \( G \) is flat, \( E \) must form a column: if two distinct edges in \( E \) both contain \( e \), they must contain each other or intersect; if they are both contained by \( e \), they must contain each other, otherwise \( e \) is a branching edge. Now note that by exactly the same reasoning, there can be no edge \( e' \in E \) that contains or is contained by an edge \( g \in G - E \), since \( g \) and \( e \) cannot contain or be contained by one another: if \( e' \) contains them both it must be a branching edge, if \( e' \) is contained by both then they must intersect, and if \( e' \) contains one and is contained by the other, then one must contain the other.

Thus, \( E \) is completely disjoint with respect to containment from \( G - E \). Thus, we can let \( C_1 = E \), and continue subdividing \( G - E \) in this manner to obtain \( C_2, \ldots, C_k \). In the end, each \( C_i \) is a column separate from one another, and \( G = \bigcup_{i=1}^k C_i \). \( \square \)

Definition 5. Consider any flat nested linear graph \( G = \bigcup_{i=1}^k C_i \), where each \( C_i \) is a column. \( G \) is level if \( |C_1| = \ldots = |C_i| \).

The MAX-LLS optimization problem is therefore to find the largest level flat subgraph in a set of linear graphs. We now show that this further restriction yields an approximation within a factor of \( O(\log |G_F|) \) of the optimal solution \( G_F \) to MAX-FLS.

Theorem 4. For any flat nested linear graph \( G_F \), its largest level subgraph \( G_L \) with size \( L = |G_L| \) satisfies \(|G_F| = O(L \log L)\).

Proof (of Theorem 4).
We first define two properties of linear graphs that are particularly important for level graphs.

Definition 6. The length \( \ell_G \) of a linear graph \( G \) is the size of the largest subgraph of \( G \) that consists solely of edges that do not intersect or contain one another, i.e. a flat graph where \( |C_i| = 1 \) for all \( i \). The height \( h_G \) of \( G \) is the size of the largest subgraph of \( G \) that consists solely of one column, i.e. a flat graph consisting of one \( h_G \)-thick edge.

These definitions are applicable to any linear graphs, but for level graphs they induce a compact representation since each level graph corresponds to an ordered pair \((h, \ell)\), as shown in fig. 9.

![Fig. 9. Level graphs \( a \) \((h, \ell) = (2, 7)\) and \( b \) \((h, \ell) = (4, 2)\). These particular graphs represent points on the level signature of the flat graph shown in fig. 10.](image)

We now consider an arbitrary flat graph \( G_F \) with height \( h_G \) and length \( \ell_G \). For each \( h = 1, \ldots, h_G \), we let \( F(h) \) be the largest value such that the level graph
\((h, F(h))\) is a subgraph, noting that \(1 \leq F(h) \leq \ell_G\) (see fig. 10). The discrete function \(F\) is thus uniquely defined for any flat graph \(G_F\). We call this function the level signature of a flat graph. Note that the level signature is unique for any flat graph, although two distinct flat graphs may produce the same level signature simply because of different order of the columns.

![Fig. 10. Representing the possible level subgraphs of a flat graph \(G\) with a discrete nonincreasing function \(F\), its level signature. Each point \((h, F(h))\) corresponds to a level graph with \(F(h)\) columns of height \(h\) that is the largest level subgraph of \(G\) of height \(h\). The shaded area represents \(L\), the size of the largest level subgraph of \(G\). The hyperbola \(\hat{F}\) has the equation \(h\ell = L\) and lies above all other points of \(F\).]

Each point \((h, F(h))\) corresponds to a level subgraph of \(G_F\), as depicted in fig. 9. The size of this subgraph is \(hF(h)\), therefore, the largest level subgraph of \(G_F\) corresponds to the point with the largest \(hF(h)\), say \((h^*, \ell^*)\). Thus, \(L = |G_L| = h^*\ell^*\).

Let \(\hat{F}\) be the hyperbola passing through \((h^*, \ell^*)\) with the equation \(h\ell = L\). By definition of \((h^*, \ell^*)\), all points on \(F\) must lie below this hyperbola. Note that the area under \(F\) given by \(\sum_{h=1}^{h_G} F(h)\) gives the size of the original flat graph \(G_F\), because \(F(h)\) counts the number of columns containing an edge at height \(h\). We now rewrite the sum as \(|G_F| = \ell_G + \sum_{h=2}^{h_G} F(h)\), noticing that the area represented by the sum is a subset of the area under \(\hat{F}\) from \(h = 1\) to \(h = h_G\). Thus,

\[
|G_F| \leq \ell_G + \int_{1}^{h_G} \hat{F}(h) \, dh.
\]

Since \((1, \ell_G)\) and \((h_G, 1)\) are both points of the level signature, \(\ell_G \leq L\) and \(h_G \leq L\). Evaluating the integral, we have \(\int_{1}^{h_G} \hat{F}(h) \, dh = \int_{1}^{\ell_G} L/h \, dh = L \log h_G\). Thus, \(|G_F| \leq L + L \log L = O(L \log L)\). \(\square\)
6 A Polytime Algorithm for MAX-LLS

To briefly summarize the results of theorems 2 and 4, a nested linear graph $G_N$ of size $S$ has a flat subgraph $G_F$ of size $F$ satisfying $S = O(F \log F)$. Rewriting, we have $F = \Omega(S/\log F) = \Omega(S/\log S)$ since $F \leq S$. The flat graph $G_F$ in turn has a level subgraph $G_L$ of size $L$ satisfying $F = O(L \log L)$, which can be similarly rewritten as $L = \Omega(F/\log S)$ (since $L \leq S$). Combining these equations yields $L = \Omega(S/\log^2 S)$.

Since the largest level flat subgraph of $G_N$ has size $L = \Omega(S/\log^2 S)$, and the optimal common level subgraph MAX-LLS has by definition size $\geq L$, we have thus shown that MAX-LLS approximates MAX-NLS within a factor of at most $O(\log^2 S)$. We now present an algorithm to compute MAX-LLS in polynomial time.

The main idea of the algorithm is to efficiently search the space of level subgraphs for the one with the largest size. For an input instance $I$ consisting of $k$ linear graphs $G_1, \ldots, G_k$, let $\ell_I = \min_{i=1}^{k} \text{length}(G_i)$, and $h_I = \min_{i=1}^{k} \text{height}(G_i)$; these will be computed in the course of the algorithm. We now demonstrate how to find, for any $h \leq h_{G_i}$, the largest level $(h, \ell)$ which is a subgraph of $G_i$.

For any edge $e = x_i \rightarrow x_j$ where $x_i < x_j$, we compute a subset $S(e)$ of the edges containing $e$. Each edge in $S(e)$ is indexed by its left coordinate. Iterating through all edges of $G_i$, we only add an edge $e' = x_{i'} \rightarrow x_{j'}$, if $i' < i$ and $j' > j$. If $S(e)$ already contains an edge $e_c$ with left coordinate $x_{i'}$, we will only keep whichever of $e'$ and $e_c$ has a smaller right coordinate. This ensures that we only keep the smallest edge containing $e$ for each left coordinate. Thus, $S(e)$ will have size $O(n)$ for every edge.

Using $S(e)$ we can compute the height of every edge in the graph (the height of an edge $e$ is the height of the tallest column where that $e$ is the top edge). We can think of the edges of our linear graph $G_i$ as nodes in a directed acyclic graph (DAG) $G^*_i$, where the edge $e \rightarrow e'$ is present in $G^*_i$ if and only if $e' \in S(e)$. Furthermore, we add an auxiliary source node $s$ that has edges to every other node of $G^*_i$, and assign weight of $-1$ to all edges of $G^*_i$. Clearly, an edge of height $h$ in $G_i$ will have shortest path distance $-h - 1$ from $s$ in $G^*_i$. Thus, computing edge heights in $G_i$ is equivalent to computing shortest path distances from $s$ in $G^*_i$.

Thus, to label the edges of $G_i$ with their height we construct the DAG $G^*_i$, and use the Bellman-Ford algorithm for DAGs [5] to compute the shortest path distances from $s$ to every node of $G^*_i$. This computation will be linear in the size of $G^*_i$. We call this procedure vertical labeling.

Similarly, we compute $R(e)$, a subset of edges that lie to the right of $e$. We only add an edge if its left coordinate $x_{i'} > x_{j'}$, and we only keep one such edge per left coordinate, the one with the smaller right coordinate, ensuring $|R(e)| = O(n)$ for any edge $e$. Using the same approach as with vertical labeling, but with the edges of $G^*_i$ given by $R(e)$ instead of $S(e)$, we obtain a labeling of edges according to the length of the largest flat sequence of non-intersecting edges ending at the given edge. The largest label in the graph will have value $\ell_{G_i}$, the length of the graph.
We generalize this approach to produce the largest level subgraph of height $h$. Starting with the labeling of the edges by height obtained during the vertical labeling phase, we compute $R_h(e)$, which is the same as $R(e)$ in the subset of $G_i$ that has height $\geq h$. In other words, we disregard all edges of height $< h$, and calculate the length of each edge in the remaining subgraph (see fig. 11). The largest level subgraph of $G_i$ with height $h$ will be $(h,F(h))$, where $F(h)$ is the largest label in the graph obtained in this manner. We call this horizontal labeling.

![Diagram](image)

**Fig. 11.** The algorithm for finding the MAX-LLS of a linear graph. First, all edges in the graph are marked with their height (see [a]), using the vertical labeling procedure. Next, for each $h$, all edges of height $< h$ are ignored, and the remaining edges are marked with their length using horizontal labeling. For $h = 1$ and $h = 2$ the results are shown in [b] and [c] respectively.

Using this procedure, we can now find MAX-LLS for an instance $I$ as follows:

1. Label the edges of each graph $G \in I$ according to height using the iteration in the vertical direction.
2. Let $h_I = \min_{G \in I} h_G$.
3. For $h = 1, \ldots, h_I$ and each $G \in I$, compute the length $F_G(h)$ of the largest level subgraph of $G$ with height $h$, using horizontal iteration. For each $h$, let $\ell_h = \min_{G \in I} F_G(h)$. The level graph $(h,\ell_h)$ is the largest common level subgraph for the instance $I$ of height $h$.
4. While iterating from $h = 1$ to $h = h_I$, keep track of the largest level subgraph $(h,\ell_h)$ produced in the previous step. Return this subgraph.

Suppose the $k$ linear graphs in the input $I$ each have $\leq n$ nucleotides and $\leq e$ edges. Each horizontal or vertical labeling procedure takes $O(ne)$, as the DAG constructed for the Bellman-Ford computation will have $O(e)$ nodes and $e \cdot O(n)$ edges. Horizontal labeling must be performed for every $h$ and both types of labeling must be done for each of the $k$ linear graphs. Thus, the overall running time, dominated by horizontal iteration, is $O(khne) = O(kn^2e)$. 


7 Conclusions

We have introduced a novel computational model for RNA multiple structural alignment, by representing each RNA sequence as a linear graph and the multiple alignment as a common nested subgraph. We noted earlier that the MAX-NLS problem represents a relaxation of RNA multiple structural alignment, because a linear graph derived from an RNA sequence by connecting all complementary nucleotide pairs has certain constraints dictating which edges must exist.

There are sound biological and computational reasons to adopt the more general NLG model. At times the complementarity of nucleotides is not sufficient for the formation of a stable hydrogen bond. For instance, adjacent complementary nucleotides are rarely paired in real structures, because geometric constraints prevent them from achieving an orientation that allows participation in hydrogen bonding. It is therefore common to explicitly prevent the structure from pairing such nucleotides (or more generally, nucleotides that are less than some fixed number of bases apart) by modifying the algorithm used to compute it. In the NLG model, this can be accomplished simply by not adding such edges to the linear graphs constructed from each sequence. In general, the NLG model is flexible enough to allow easy incorporation of biological insights that modify the space of permissible pairings. Insights that reduce this space are particularly valuable because by decreasing the number of edges in the resulting linear graphs, the running time of our approximation algorithm improves accordingly. In addition, heuristic approaches to prune certain edges, which are deemed unlikely to be included in the final structure, could be combined with our algorithm in order to reduce running time further. Such enhancements are likely to be incorporated into any practical algorithm that finds biologically meaningful structures.

The approximation quality, while bounded by \( O(\log^2 S) \) in the worst case, will vary depending on the class of ncRNAs being aligned. When mapped back to the RNA sequence, a level graph consists of \( t \) groups of stems, each consisting of \( h \) complementary pairs. Thus, for ncRNA families whose secondary structure fits this pattern well, such as tRNAs, our algorithm will perform more accurately.

Compared to the elaborate free energy functions used by several structure-prediction programs [18,19], the NLG model uses a fairly rough approximation. The main advantage of the NLG model is the ability to incorporate multiple sequence information without having a fixed alignment. The approximation algorithm we presented could be used to obtain a rough alignment and structure, which could then be refined using heuristic methods with more elaborate scoring models. Such a hybrid would combine theoretical bounds on approximation quality derived in the NLG framework with the benefits of heuristic approaches.

8 Acknowledgements

The authors were supported in part by NSF grant EF-0312459. ED was also supported by NIH-NIGMS training grant GM063495. Special thanks to Marina Sirota for assistance with manuscript preparation and editing.
References