Effective RNA 3D Structure Prediction via Assembling Compact Tetrahedrons

RNA Informatics Lab@UGA

January 2, 2018

We propose to use *compact tetrahedrons* to assemble 3D structure of query RNA sequences. A compact tetrahedron is 3D tetrahedron with 4 nucleotides as its vertices such that every nucleotide is interacting with (or within the δ-vicinity of) at least two other nucleotides, where δ is a predefined cutoff distance in Angstrom. This note introduces a new algorithm to realize the method and addresses all critical issues related to the new method.

1 Background

2 Addressing the efficiency issue

Compact tetrahedrons is a new notion that can be used to construct a desired backbone 3-tree and, at the same time, to instantiate the 4-cliques on the 3-tree with 3D motifs, thus directly yielding a 3D structure predicted for the query RNA sequence. However, since the fastest algorithms to build optimal k-trees are still of the complexity $\Omega(n^{k+1})$, the inefficiency may hinder the scalability of the method to even moderately long RNA sequences. We propose to address the issue with the notion of *compact 4-cliques*.

A compact 4-clique $\kappa = \{x, y, z, w\}$ is such that $\exists a, b \in \kappa$, $a \neq b$, and either $a = b + 1$ or $\gamma(a, b) = cWW$ where γ defines the type of interaction between a and b. Because γ is predefined as long as the secondary structure of the query RNA is given, there are at most 3 free variables in κ. Consequently, the total number of compact 4-cliques is $O(n^3)$.

We call a backbone 3-tree consisting of only compact 4-cliques a *compact backbone 3-tree* (or simply CB 3-tree). To develop dynamic programming algorithms to compute an optimal CB 3-tree, we first need a strategy that

can efficiently enumerate all but only compact 4-cliques. We propose a method for such a purpose as follows.

Let $\kappa = \{x, y, z, w\}$ be a compact 4-clique such that $x < y < z < w$. We define function $p(x)$, $p(x) > x$, to be the position of nucleotide that forms a cWW interaction with the nucleotide in any given position $x \geq 1$, where $p(x) = 0$ if x is not a part of a cWW interaction. The follow process enumerate all compact 4-cliques given any query RNA sequence of length n and known secondary structure with p as its cWW relation on the nucleotide positions.

ENUMERATE 4-CLIQUE(n, p)

```
1. for x = 1, ..., n − 3
2.       for y = x + 1, ..., n − 2
3.          if (y = x + 1) or (y = p(x))
4.             then
5.                for z = y + 1, ..., n − 1
6.                   for w = z + 1, ..., n
7.                      output {x, y, z, w}
8.             else
9.                for z = y + 1, ..., n − 1
10.                  if (z = y + 1) or (z = p(x)) or (z = p(y))
11.                     then
12.                        for w = z + 1, ..., n
13.                           output {x, y, z, w}
14.                     else
15.                        output {x, y, z, z + 1}
16.                        if p(x) > z + 1
17.                         then
18.                           output {x, y, z, p(x)}
19.                        if p(y) > z + 1
20.                         then
21.                           output {x, y, z, p(y)}
22.                        if p(z) > z + 1
23.                         then
24.                           output {x, y, z, p(z)}
```

* Implementation is in the file enumerate_cb_cliques.hpp:

https://github.com/matthewwicker/CompactTetrahedrons

Though the algorithm may be made more succinct, its run through at most 3 nested **for** loops, enumerating only compact 4-cliques.

3 Dynamic programming algorithm

We now consider a dynamic programming algorithm for the computation of optimal a CB 3-tree. To facilitate the discussion, we assume a fitness function f on every compact 4-clique κ such that the score of a CB 3-tree $G = (V, E)$, where $V = [n]$, is basically the sum of fitness function values over all involved compact 4-cliques:

$$\phi(G) = \sum_{\kappa \in G} f(\kappa)$$

Note that according to the standard definition of k-trees, the base case of a 3-tree is a 3-clique. Therefore, a 3-tree $G = (V, E)$ can be created from (or *pivoted at*, as we call in this note) a 3-clique. Furthermore, a sub-tree G_U of a 3-tree (with some vertices and associated edges removed) is actually a 3-tree created only from a permissible subset of vertices $U \subseteq V$, which is pivoted at some 3-clique. Based on this observation, we are able to define an objective function for creating an optimal CB 3-tree through a recursion pattern.

In particular, given a triplet of nucleotide indexes $\{v_1, v_2, v_3\}$, adding a choice of a permissible index x forms a 4-clique $\{v_1, v_2, v_3, x\}$. Adding another index y to $\{v_1, v_2, v_3\}$ forms a different 4-clique $\{v_1, v_2, v_3, y\}$. Note that these two 4-cliques will belong to two different "branches" of the 3-tree if they both exist in the 3-tree. Therefore, the choices of x and y cannot be arbitrary but exclusive and depend on given subsets of permissible indexes associated with the "branches" intended to be created. Along this line of consideration, four possible triplets of indexes may exist after x is added to $\{v_1, v_2, v_3\}$:

Summary: we are only able to select an addition based on the enumerated cliques

$$\{v_1, v_2, x\},\ \{v_1, v_3, x\},\ \{v_2, v_3, x\},\ \{v_1, v_2, v_3\}$$

which in turn can be used to result more triplets with permissible indexes in exclusive ways across these four triplets. This leads to the following recursive solution to create an optimal CB 3-tree.

Let $\{v_1, v_2, v_3\}$ be a triplet of indexes, where $v_1 < v_2 < v_3$. We define binary string $I = i_1 i_2 i_3 i_4$ to be *interval indicator* for the four index intervals V_1, V_2, V_3 and V_4, in the order of increasing indexes, partitioned by the indexes $\{v_1, v_2, v_3\}$, exclusive of v_1, v_2, v_3, such that for each $j = 1, 2, 3, 4$,

$$V_j = \{a : i_{j-1} < a < i_j \text{ and } i_j = 1\}$$

where by default, $i_0 = 0$ and $i_4 = n + 1$.

More awkwardly, but intuitively (for me): binary string 1111 indicates that the following intervals are open: [0,i1],[i1,i2], [i2,i3],[i4,n]

Here, open means that we are able to add a new vertex within that interval

For example, if the triplet is $\{5, 10, 20\}$ and $n = 100$. The interval indicator $I = 1010$ gives the following four index intervals: $V_1 = [1..4]$, $V_2 = \emptyset$, $V_3 = [11..19]$, $V_4 = \emptyset$.

Definition. Let $\{v_1, v_2, v_3\}$ be a triplet of indexes and $I = i_1 i_2 i_3 i_4$ be an interval indicator of intervals V_1, V_2, V_3 and V_4. Define $M(v_1, v_2, v_3, I)$ to be the optimal value of a CB 3-tree *pivoted at* 3-clique $\{v_1, v_2, v_3\}$ created only for indexes within the index intervals defined by the indicator I.

Summary: given a triple and an interval, we have four cases

When $\bigcup_{j=1}^{4} V_j \neq \emptyset$, $M(v_1, v_2, v_3, I)$ is computed recursively as the maximum over the following *four possible cases*:

When the interval is not closed for all 1,..,4

Case 1 Phi Constraints yield:

A = x y 2 0
B = x y 0 4
C = x 0 3 4
D = 0 2 3 4

Pick one or all x = 1
Pick one y = 1

if i_2 is 1 then we pick one; else we pick neither of both.

if i_3 is 1 then we pick either one or all; else we pick either none or both.

if i_4 is 1 then we pick one; else we pick either none or a pair.

Case 1. only if $i_1 = 1$, when V_0 not empty

$$\min_{x<v_1,\Phi_1} \left\{ M(x, v_1, v_2, A) + M(x, v_1, v_3, B) + M(x, v_2, v_3, C) + M(v_1, v_2, v_3, D) + f(x, v_1, v_2, v_3) \right\}$$

where interval indicators $A = a_1 a_2 a_3 0$, $B = b_1 b_2 0 b_4$, $C = c_1 0 c_3 c_4$ and $D = 0 d_2 d_3 d_4$ are subject to constraints Φ_1:

$$\Phi_1 \equiv \begin{cases} a_1 + b_1 + c_1 = 1 \\ a_2 + b_2 = 1 \\ a_3 + d_2 = i_2 \\ b_4 + c_4 + d_4 = i_4 \\ c_3 + d_3 = i_3 \end{cases}$$

Case 2 Phi Constraints yield:

A = 1 x y 0
B = 1 x 0 4
C = 0 y 3 4
D = 1 0 3 4

Pick one x to = 1
Pick one y to = 1

If i_1 is 1 then pick either one or all of the 1's;
else pick none or a pair of ones.

If i_3 is 1 then pick either 3;
else pick neither or both.

If i_4 is 1 then pick one or all fours;
else pick two or none.

Case 2. only if $i_2 = 1$,

$$\min_{v_1<x<v_2,\Phi_2} \left\{ M(v_1, x, v_2, A) + M(v_1, x, v_3, B) + M(x, v_2, v_3, C) + M(v_1, v_2, v_3, D) + f(x, v_1, v_2, v_3) \right\}$$

where interval indicators $A = a_1 a_2 a_3 0$, $B = b_1 b_2 0 b_4$, $C = 0 c_2 c_3 c_4$ and $D = d_1 0 d_3 d_4$ are subject to constraints Φ_2:

$$\Phi_2 \equiv \begin{cases} a_1 + b_1 + d_1 = i_1 \\ a_2 + b_2 = 1 \\ a_3 + c_2 = 1 \\ b_4 + c_4 + d_4 = i_4 \\ c_3 + d_3 = i_3 \end{cases}$$

Case 3 Phi Constraints:

A = 1 2 x 0
B = 1 0 y 4
C = 0 x y 4
D = 1 2 0 4

Pick one x to = 1
Pick one y to = 1

If i_1 is 1 then pick either one or all 1s; else pick a pair or none.

If i_2 is 1 then pick either 2; else pick none or both.

If i_4 is 1 then pick either all or one 4s; else pick a pair or none.

Case 3. only if $i_3 = 1$,

$$\min_{v_2<x<v_3,\Phi_3} \left\{ M(v_1,v_2,x,A)+M(v_1,x,v_3,B)+M(v_2,x,v_3,C)+M(v_1,v_2,v_3,D)+f(x,v_1,v_2,v_3) \right\}$$

where interval indicators $A = a_1a_2a_30$, $B = b_10b_3b_4$, $C = 0c_2c_3c_4$ and $D = d_1d_2d_30$ are subject to constraints Φ_3:

$$\Phi_3 \equiv \begin{cases} a_1 + b_1 + d_1 = i_1 \\ a_2 + d_2 = i_2 \\ a_3 + c_2 = 1 \\ b_3 + c_3 = 1 \\ b_4 + c_4 + d_4 = i_4 \end{cases}$$

Case 4 Phi Constraints:

A = 1 2 0 y
B = 1 0 x y
C = 0 3 x y
D = 1 2 3 0

Pick one or all y = 1
Pick one x

If i_1 is 1 then pick all or one 1; else pick a pair or none

If i_2 is 1 then pick a 2; else pick both or none

If i_3 is 1 then pick a 3; else pick both or none

Case 4. only if $i_4 = 1$,

$$\min_{x>v_3,\Phi_4} \left\{ M(v_1,v_2,x,A)+M(v_1,v_3,x,B)+M(v_2,v_3,x,C)+M(v_1,v_2,v_3,D)+f(x,v_1,v_2,v_3) \right\}$$

where interval indicators $A = a_1a_20a_4$, $B = b_10b_3b_4$, $C = 0c_2c_3c_4$ and $D = d_1d_2d_30$ are subject to constraints Φ_4:

$$\Phi_4 \equiv \begin{cases} a_1 + b_1 + d_1 = i_1 \\ a_2 + d_2 = i_2 \\ a_4 + b_4 + c_4 = 1 \\ b_3 + c_3 = 1 \\ c_2 + d_3 = i_3 \end{cases}$$

Finally, the base cases for $M(v_1, v_2, v_3, I)$ is when $I = 0000$ that yields all empty index intervals. Under this situation, $M(v_1, v_2, v_3, I) = 0$, regardless the indexes v_1, v_2, v_3.

4 Implementation

Theoretically, one can compute function M with a dynamic programming table of size $n^3 \times 2^4$ and each entry is computed with one variable x to consider, which has the multiplicative factor of n, yielding $O(n^4)$. But based on the discussion in section 2, since we only consider CB 3-trees, the compact 4-cliques only have $O(n^3)$ in number, the total computation time should be

of $O(n^3)$. However, to achieve a practical time within $O(n^3)$, cautions need to be taken in the a few places of an implementation.

First, the enumeration of triplets $\{v_1, v_2, v_3\}$ should consider choosing index x to be included to form a compact 4-clique only. There are two scenarios. One is that v_1, v_2, v_3 are *unrelated*, independent variables and the index x to be included has to be *either* $v_j \pm 1$, or (v_j, x) or (x, v_j) is a cWW interaction, for some $j = 1, 2, 3$. Another is that v_1, v_2, v_3 are already *related*, where x can be any index that is not the same as v_1, v_2, or v_3.

Second, the index interval indicators A, B, C, D in the recursive computation of M are under constraints Φ_1 - Φ_4, respectively. Simple Exhaustive enumeration of all possible binary string combinations for the 4 indicators may result in a large constant $(2^4)^4 = 2^{16} \approx 64,000$. Note that given $I = i_1 i_2 i_3 i_4$, at least one bit in each of A, B, C, D is 0. So the full combination number can be reduced to $(2^3)^4 = 2^{12} \approx 4,000$. A more close examination on the constraints Φ_1 - Φ_4 tells us that it suffices to enumerate binary strings for two indicators, thus further reduces the constant to just $(2^3)^2 = 2^6 = 64$.

Third, the bottom-up fashion for computing M may be a little intriguing. But the data dependency always abides by the following rule: Let T_1 and T_2 be two triplets of indexes and I and J are two index indicators based on T_1 and T_2 respectively, which induce two subsets of indexes V_I and V_J. Then $M(T_1, I)$ needs to be computed before $M(T_2, J)$ **only if** $V_I \subset V_J$.

Therefore, by default, $V_{0000} = \emptyset$, so all base cases can be initialized before the other entries are computed. To simplify the process, one can take a simple recursive top-down approach aided with table look-up.

What are our base cases here?

Should we enumerate all the CB triples and then use the enumerated triples as the base cases:
I = {1,0,0,0}
I = {0,1,0,0}
I = {0,0,1,0}
I = {0,0,0,1}

5 Compact tetrahedrons

The physical 3D instantiation of a compact 3-tree is to use *compact tetrahedrons* to fit compact 4-cliques. We define a tetrahedrons to be such a 3D motif found in known RNA 3D structures that

- (at least) two nucleotides are neighbors on the backbone, or they share a cWW interaction, and
- every nucleotide is interacting with (or within the δ-vicinity of) at least two other nucleotides, where δ is a predefined cutoff distance in Angstrom.

Search through the database of known RNA 3D structures will allow us to retrieve all compact tetrahedrons. Much like in the case of enumerating

compact 4-cliques, caution needs to be taken to avoid excessive use of time (see Section 2) during the search. Specifically, the search should only enumerate those tetrahedrons in which only 3 of the quadruplet of indexes are variables in every RNA that is searched.

A compact 4-clique can be instantiated with one or more 3D compact tetrahedrons. We call the set of compact tetrahedrons for a compact 4-clique *tetrahedron candidates*.

6 Scoring function M

According to Section 3, the score of function $M(v_1, v_2, v_3, I)$ is dependent on the function $f(v_1, v_2, v_3, x)$ value of compact 4-clique $\{v_1, v_2, v_3, x\}$, where x is chosen over all possibilities allowable by the index indicator I. Here we discuss in detail how to score $f(v_1, v_2, v_3, x)$ and $M(v_1, v_2, v_3, I)$ when compact 4-cliques are instantiated with compact tetrahedrons.

We first define for a triplet of indexes $\{v_1, v_2, v_3\}$ a set of 3D motifs, called *triplet motifs*, each formed by the 3 nucleotides in the 3 index positions of the query RNA sequences. Such motifs can be found in the known RNA structure database, much like the compact tetrahedrons.

Then function

$$M(v_1, v_2, v_3, I) = \langle t, s \rangle$$

where t is a triplet motif for $\{v_1, v_2, v_3\}$ and s is the best score of a 3D structure pivoting at t for the set of nucleotides whose indexes allowed by I. Technically, we associate with $M(v_1, v_2, v_3, I)$ a set of triplet motifs for $\{v_1, v_2, v_3\}$, instead of just t. These triplet motifs for $\{v_1, v_2, v_3\}$ are the pivots of 3D structures of the set of nucleotides whose indexes allowed by I and these 3D structures have the top scores, with s being the top one score. We assume the cardinality of the set of triplet motifs for $\{v_1, v_2, v_3\}$ is $\leq m$.

Initially when $I = 0000$, for $M(v_1, v_2, v_3, I) = \langle t, 0 \rangle$, where t is an arbitrary triplet motif in the associated set of triplet motifs for $\{v_1, v_2, v_3\}$.

Inductively, when $M(v_1, v_2, v_3, I)$ is being computed with vertex x chosen over all possible indexes allowed by I, there are results already from four function sub-cases that have already been computed: $M(v_1, v_2, x, A)$, $M(v_1, v_3, x, B)$, $M(v_2, v_3, x, C)$, and $M(v_1, v_2, v_3, D)$. Note that each of these sub-cases is associated with a set of triplet motifs as well as a top triplet motif with the highest score, respectively,

Let d be a compact tetrahedron for the compact 4-clique $\{v_1, v_2, v_3, x\}$. Let t_a, t_b, t_c, t_d be a top triplet motif computed from the four sub-cases, respectively. Also let $s_a, s_b, s_,s_d$ be the highest score computed for the 3D

structure pivoting at the corresponding triplet motifs, respectively. Then

$$\sigma(x) = \min_{d,t_a,t_b,t_c,t_d} \left\{ RMSD(d, t_a, t_b, t_c, t_d) + s_a + s_b + s_c + s_d \right\}$$

and $s = \min_x \sigma(x)$, aligning with the theoretical framework of computing M for an optimal CB 3-tree as described in Section 3.

Without doubt, maintaining a set of triplet motifs for every triplet $\{v_1, v_2, v_3\}$, instead of single motif, increases the time complexity by $\Omega(m^4)$, where m is the number of triplet motifs maintained for each triplet. Given the constraints applied in compact tetrahedrons, we expect m to be very small, however. We also note that, another multiplicative factor $O(h)$ is needed if h is the number of compact tetrahedrons for every compact 4-clique.

7 Predicted 3D structure retrieval

The scoring process for computing objective function M contains the sufficient information for the 3D structure predicted for the input sequence. A traceback process can be designed to recover all the information of the 3D structure with the final score s that corresponds to $M(v_1, v_2, v_3, I)$, where $I = 1111$ for some triplet $\{v_1, v_2, v_3\}$. This requires that the choices of x and rank of triplet motifs for triplets at every step are kept for the traceback purpose. Actually, if we maintain m 3D motifs for every triplet, we should produce m 3D structures predicted for the query input sequence.