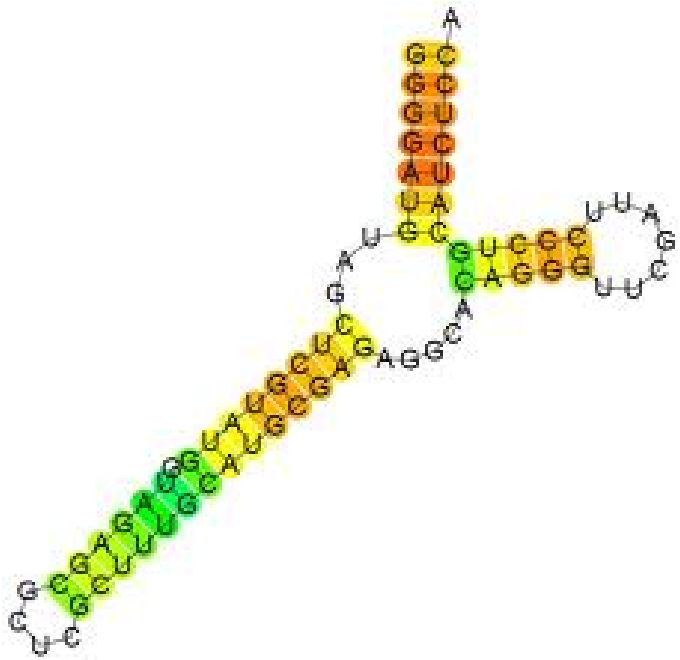
GraphClust: alignment-free structural clustering of local RNA secondary structures.

Yakovlev Pavel

St-Petersburg Academic University

March 16, 2013

Abstract



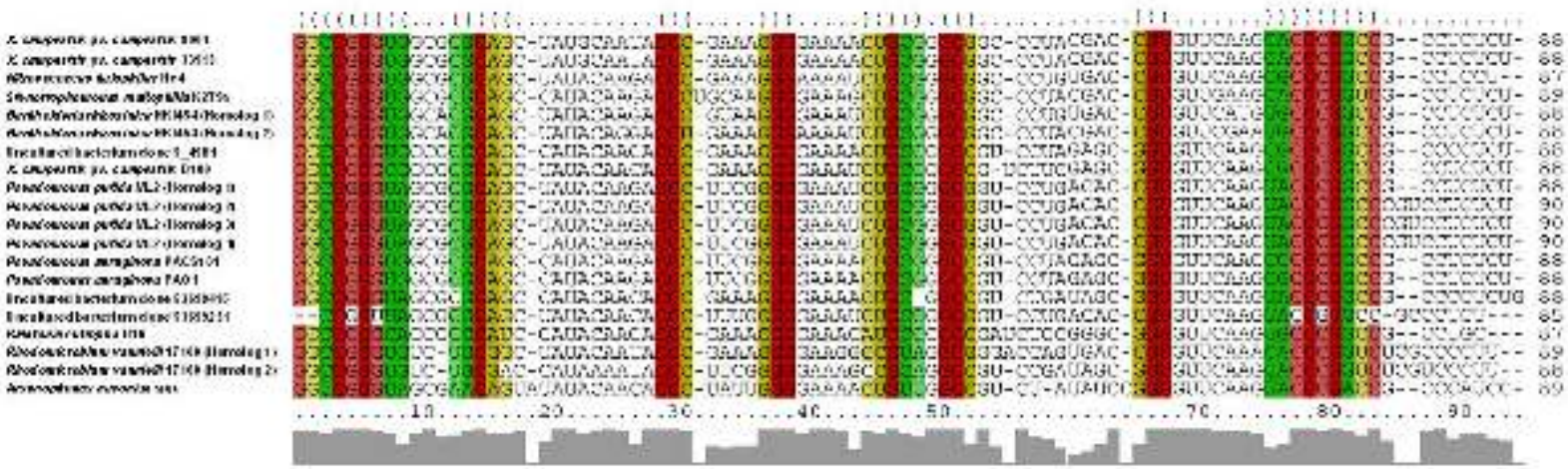
Non-protein coding RNAs (ncRNAs) have moved to a key research in molecular biology, fundamentally challenging established paradigms and rapidly transforming our perception of genome complexity. There is mounting evidence that the eukaryotic genome is pervasively transcribed and that ncRNAs function at various levels regulating gene expression and cell biology.

Motivation

Clustering algorithms are applied to store, compare and predict ncRNAs, because:

- in contrast to protein-coding genes, ncRNAs belong to a diverse array of classes with vastly different structures, functions, and evolutionary patterns
- ncRNAs can be divided into *RNA families* according to inherent functional, structural, or compositional similarities

Question:
So, what is the problem with «old» alignment-based algorithms?



Answer:
They are awfull and do not work.

Why?

- Multiple Sequence Alignment (MSA) is an NP-complete problem.
- Approximation algorithms have very high computational complexity too (dynamic programming):
  - Sankoff Algorithm (1985) - $O(n^4)$
  - Carrillo-Lipman Algorithm (1989)
- Thus, the tools, based on these algorithms, are limited to relatively small datasets.
  - LocARNA (2007)
  - FoldAlign (2007)

Smart quote

Gorodkin: *'Even using substantially more sophisticated techniques, genome-scale ncRNA analyses often consume tens to hundreds of computer years. These high computational costs are one reason why ncRNA gene finding is still in its infancy.'*[1]

[1] Gorodkin,J. et al. (2010) De novo prediction of structured RNAs from genomic sequences. Trends Biotechnol, **28**, 9–19.

Solutions

Postulate:
ncRNAs can be also grouped by *RNA classes* whose members, in contrast with *RNA families*, have no discernible homology at the sequence level, but still share common structural and functional properties.

Two main classes of clustering approaches:

1. Class uses simplifications in the representation of structures.
   - heavily depend on the correctness of the structures
   - computational prediction of secondary structures are known to be error prone
2. Class uses sequence information as prior knowledge to speed up the computation.
   - sequences are first clustered by sequence-alignment
   - family assignments of structured RNAs obtained from sequence alignments at pairwise sequence identities below 60% are often wrong

GraphClust (University of Freiburg, Germany):

- comparing and clustering RNAs according to both sequences and structures
- alignment-free
- linear-time
- scales to datasets of hundreds of thousands of sequences
- extended fast graph kernel technique designed for chemoinformatics
- ready-to-use pipeline, that is available on request *for free*

Approach pipeline:

1. Try a small number of probable, but sufficiently different, structures for each RNA sequence.
2. Encode each structure as a labeled graph preserving all information about the nucleotide type and the bond type - sequence's structure is represented as a graph with several disconnected components.
3. Compute the similarity between the representative graphs using a [chemoinformatics] graph kernel.
4. Evaluate each neighborhood and select as candidate clusters those that contain very similar elements.
5. Dataset scanning: associating missing sequences to clusters and removing them from set.
6. Reiterating the procedure

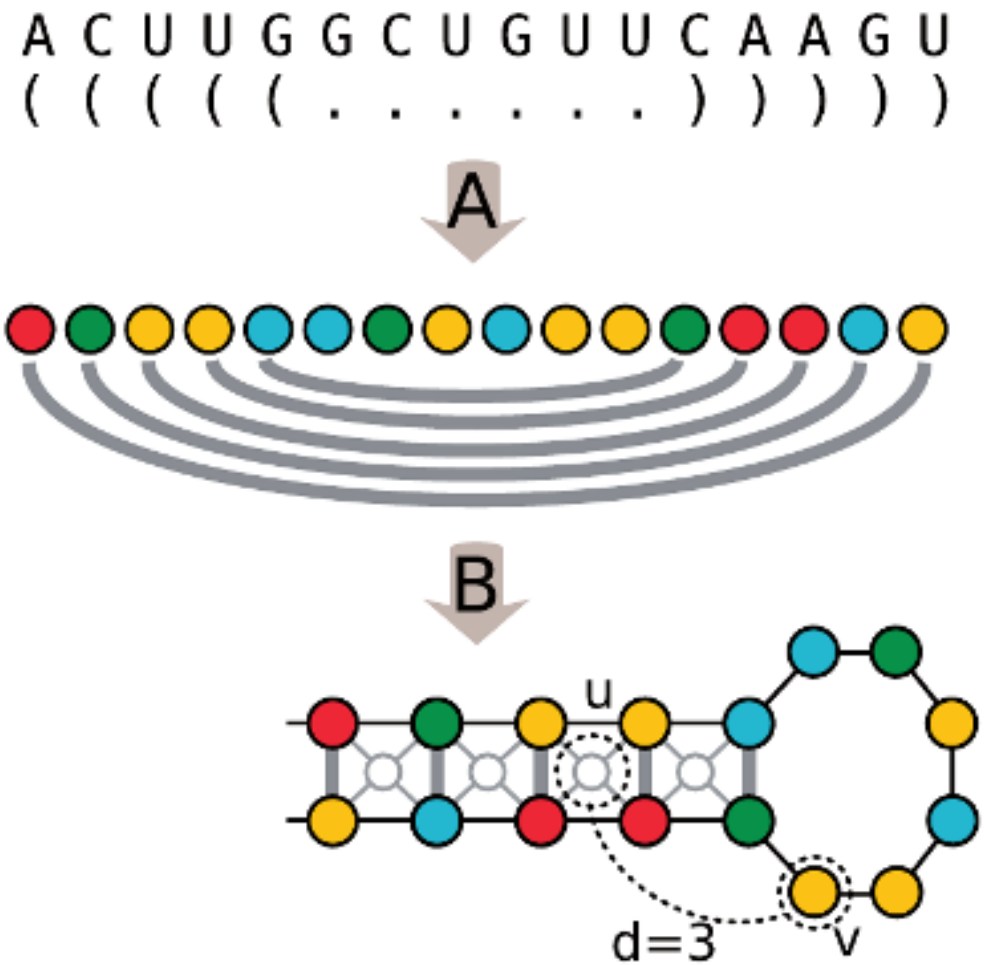
Obscure:

- How structures are encoded as graphs?
- How to compare graphs or graph substructures?
- Why we have not quadratic number of comparisons of graphs?

Minimum free energy-based secondary structure prediction has been shown to be error prone (Gardner et al., 2005). So, we want to use representation of the entrie ensemble of low-energy conformations.

Problem: Resorting to a complete enumeration of near-optimal structures would yield a tremendously large number of structures.
Solution: We opt for abstract shape analysis methods.

Answer 1
Graph encoding for structures



We consider subsequences of our ncRNA sequence, as overlapping windows. Procedure depends on two parameters:

W - window sizes (30 and 150 nt in tests)

O - overlap parameter for window start positions

Than we take the set **I** most representative structures of each subsequence and encode them as disconnected components.

But how to compare these graphs?

Graph similarity is the central problem for all learning tasks such as clustering and classification on graphs.
Subgraphs polymorphism is NP-complete problem.

Graph kernels:

- Compare substructures of graphs that are computable in polynomial time
- Examples: walks, paths, cyclic patterns, trees

Good graph kernel criteria:

- Expressive
- Efficient to compute
- Positive definite
- Applicable for wide range of graphs

NSPDK - Neighborhood Subgraph Pairwise Distance Kernel (Costa and Grave, 2010)

Key fetures:

- very fast kernel
- suitable for large datatsets
- works with sparse graphs with discrete vertex and edge labels
- works with special pair of subgraphs, called «neighborhood» subgraphs
- instance of *decomposition kernel* (Haussler, 1999)

Definitions:

- $G(V, E)$ - graph, $r \geq 0$. Than $N_r^v(G)$ - *neighborhood subgraph* of G.

  Where $N_r^v(G)$ rooted in v and induced by the set of vertices within distance not exceeding r.

- $R_{r,d}$ - *neighborhood-pair relation*.

  Where $R_{r,d}$ - indicates that the distance between the roots of two neighborhood subgraphs of radius r is exactly equal to d.

- $k_{r,d}(G, G') = \sum_{\substack{A,B \in R_{r,d}^{-1}(G) \\ A',B' \in R_{r,d}^{-1}(G')}} 1(A \cong A')1(B \cong B')$ - *decomposition kernel*.

  $R_{r,d}^{-1}$ - indicates all possible pairs of neighborhood subgraphs or radius r, whose roots are at distance d.

Than, *NSPDK* (non-normalized form) is:

$$K(G, G') = \sum_r \sum_d k_{r,d}(G, G')$$

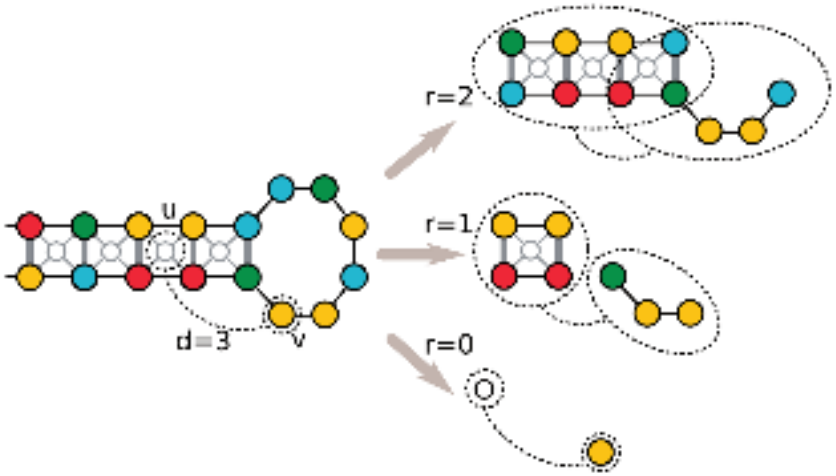
For efficiency reasons, the "upper bound" form:

$$K_{r^*,d^*}(G, G') = \sum_{r=0}^{r^*} \sum_{d=0}^{d^*} k_{r,d}(G, G')$$

Normalization like decomposition kernel:

$$\hat{k}_{r,d}(G, G') = \frac{k_{r,d}(G, G')}{\sqrt{k_{r,d}(G, G) k_{r,d}(G', G')}}$$

Answer 2
NSPDK



Full isomorphism test is computationally expensive. Reducing the task:

1. String encoding: uses a technique based on the distance information between pairs of vertices and can be computed in $O(|V|)$.
2. Hashing procedure: encoding string to integer code (Costa and Grave, 2010).
3. Isomorphism test: equality test between thier integer codes.

NSPDK limits the number of features to $O(r^* d^* |V|^2)$ pairs of neighborhood subgraphs (one feature for each pair of vertices times each possible combination of values for the radius and the distance).
$r^* \in [0; 5]$ $d^* \in [0; 10]$
For sparse graphs, the number of vertices that are reachable within fixed small distance is typically small $\Rightarrow$ dependency on the vertex set size can be more tightly approximated by $O(|V|)$.

Conclusion: each graph is mapped into a sparse vector in R^m - high-dimensional feature space. Number of non-zero features in this space is linear in $|V|$.

Some definitions:

- P - instances [graphs] set
- x, z, p - any instance
- x_j - element of instance
- $N_k(x)$ - the set of the k-closest instances.

So, how to cluster graphs using NSPDK? Compare all the possible pairs of graphs is too expensive. That's why let us compare only similar graphs.

$D_k(x) = \frac{1}{k} \sum_{z \in N_k(x)} K_{r^*,d^*}(x, z)$ - k-neighborhood density is good metrics for clustering.

To make it as fast, as possible, we want to count $N_k(x)$ very fast. It is perfect to do it in constant time.

MinHash algorithm will help us with this task.

MinHash algorithm:

1. Binarize all instances $P = \{p_1, \ldots, p_n\}$ from $R^m \mapsto \{0,1\}^r$.
2. Choose some $f_i : N \mapsto N$ - random hash functions with terms: $\forall x_j \neq x_k,\ f_i(x_j) \neq f_i(x_k)$ and $\forall x_j \neq x_k,\ P(f_i(x_j) \leq f_i(x_k)) = 1/2$.
3. Get hash functions $h_i(x) = argmin_{x_j \in x} f_i(x_j)$[1]. So, they return the first feature indicator under a random permutation of the features oreder.
   Useful facts:
   - $P(h_i(x) = h_i(z)) = \frac{|x \cap z|}{|x \cup z|} = s(x,z)$ - Jaccard similarity, the fraction of common x and z features over the total number of none-null features of x and z.
   - Error rate $\gamma = \left[\frac{1}{N^2}\right]$, where N - number of hash functions.
4. Build a tuple $< h_1(x), \ldots, h_N(x) >$
5. Collect the set of instances for $h^* = h_i(x)$ as $Z_i(h^*) = \{z \in P\ :\ h_i(z) = h^*\}$.
6. $Z = \{Z_i\}_{i=1}^N$ - approximate neighborhood, that can be used to find $N_k(x)$.

These steps can be performed in constant time (N is fixed, Z independent of the dataset size $|P|$).

[1] These hash functions are called *min-hash function* (Broder, 1997).

MinHash
Demo Time

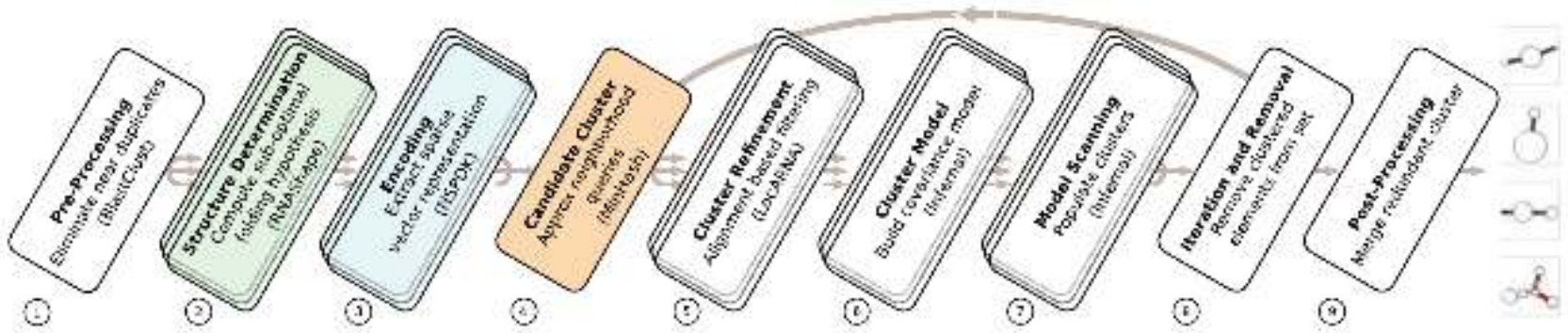
```
def generic_hash( s, seed ):
  result = 1
  for char in s:
    result = int(seed*result + ord(char)) & 0xFFFFFFFF;
  return result

def hashgen( size ):
  seed = math.floor( random.random() * size ) + 32
  return functools.partial(generic_hash, seed = seed)

def signature( set, functions ):
  return [min(map(f, set)) for f in functions]

def similarity( sig_a, sig_b, num_funcs ):
  return sum(x == y for x, y in zip(sig_a, sig_b)) / num_funcs

def minhash( max_error ):
  num_funcs = int(1 / max_error ** 2)
  functions = map(hashgen, xrange(num_funcs))
  return [functools.partial(signature,  functions = functions),
          functools.partial(similarity, num_funcs = num_funcs)]
```

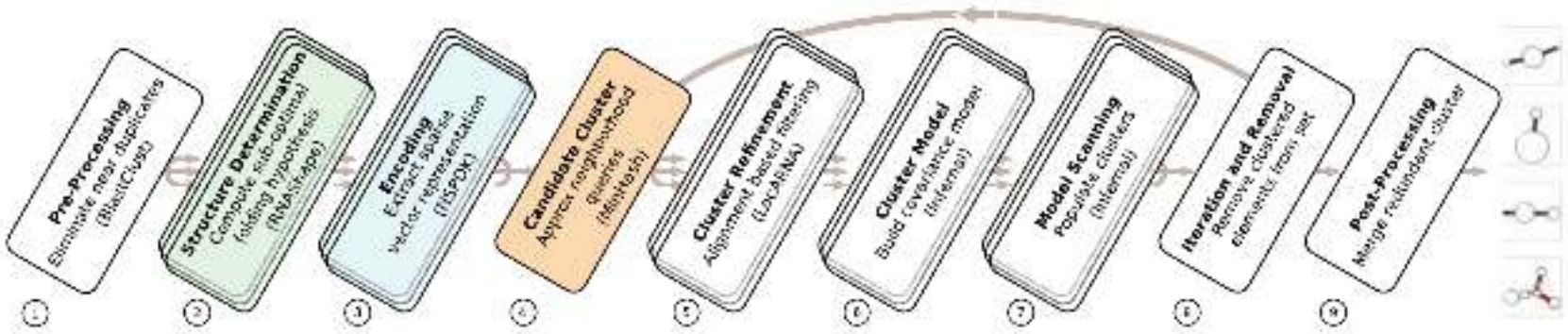


Phase 1. Preprocessing

- Get RNA sequences from RNA-seq or computational methods, mask all repeats to avoid clusters made of genomic repeats.
- Split long sequences into smaller fragments (windows)[1].
- Near identical sequences are removed using *blastclust*[2].

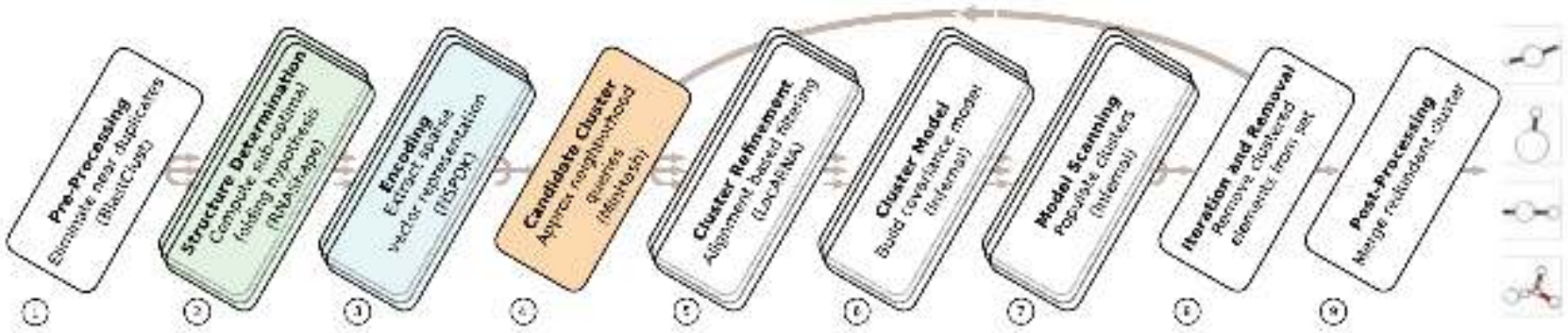
[1] to enable detection of small signals

[2] The program begins with pairwise matches and places a sequence in a cluster if the sequence matches at least one sequence already in the cluster. Megablast algorithm is used.



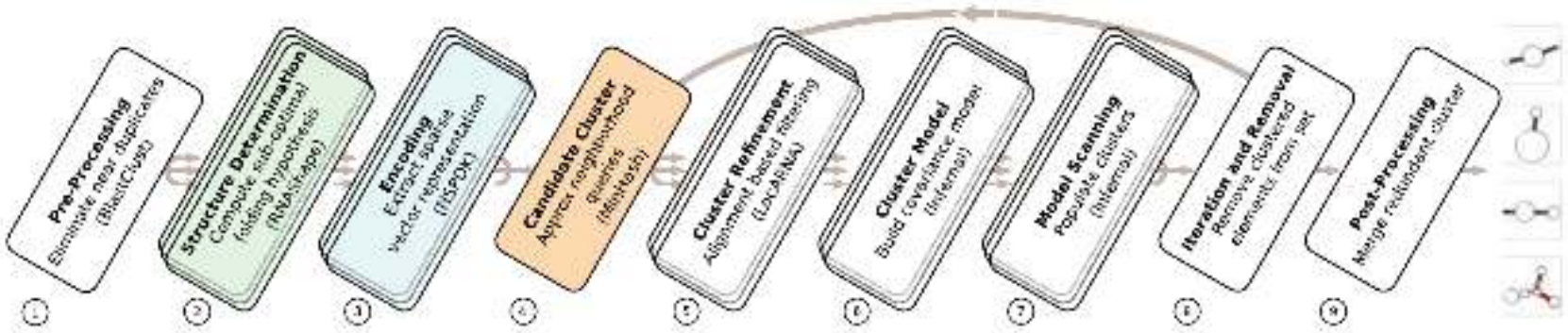
Phase 2. Structure determination
Try set of structures for each window with *RNAShape* tool.

150 nt sequence processing into set of disconnected graphs of ≈ 2500 total vertices.
$O = 20\%;\ W = 30, 150;\ I = 3$



Phase 3. Parallel encoding
Encoding set of structures into sparse vectors, by counting K_{r^*,d^*}.

150 nt sequence processing into a sparse vector with ≈ 8000 features.
$r^* = 2;\ d^* = 4$



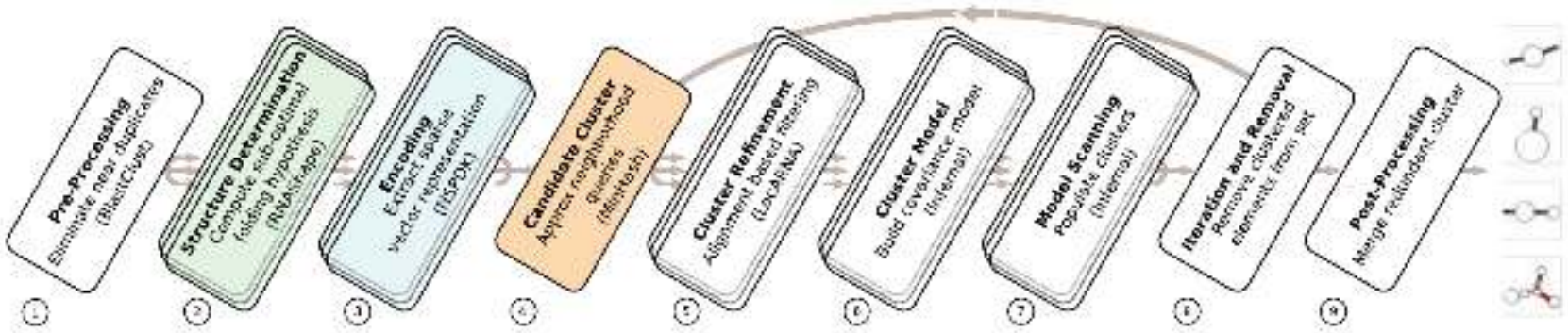
Phase 4. Candidate cluster

c_i - candidate clusters

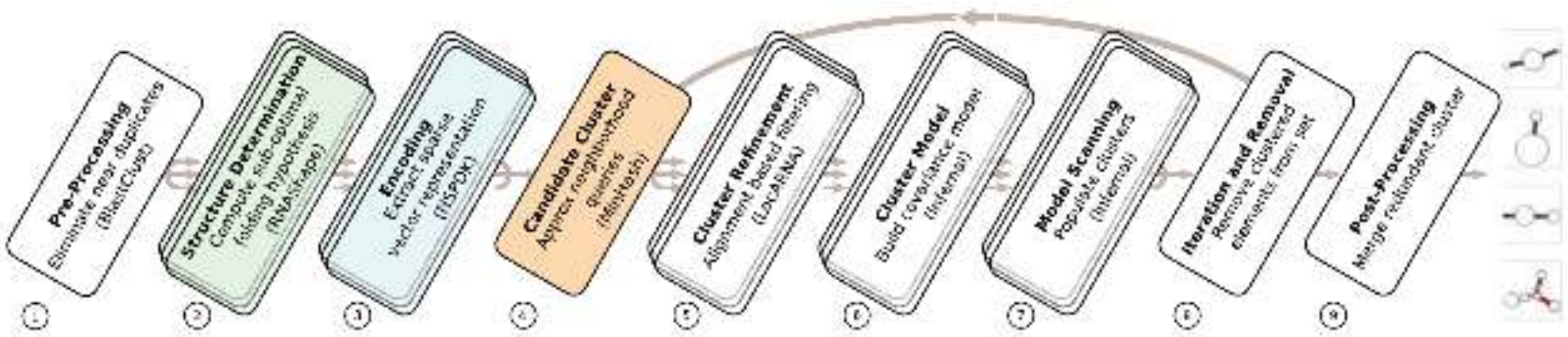
c_i sorted as $\forall i < j,\ D(c_i) > D(c_j)$

Building the union of candidate clusters:

$C_j = \bigcup_{i=1}^{j} c_i$ with greedily discarded c_k if $|C_j \cap c_k| > \mathit{threshold}$

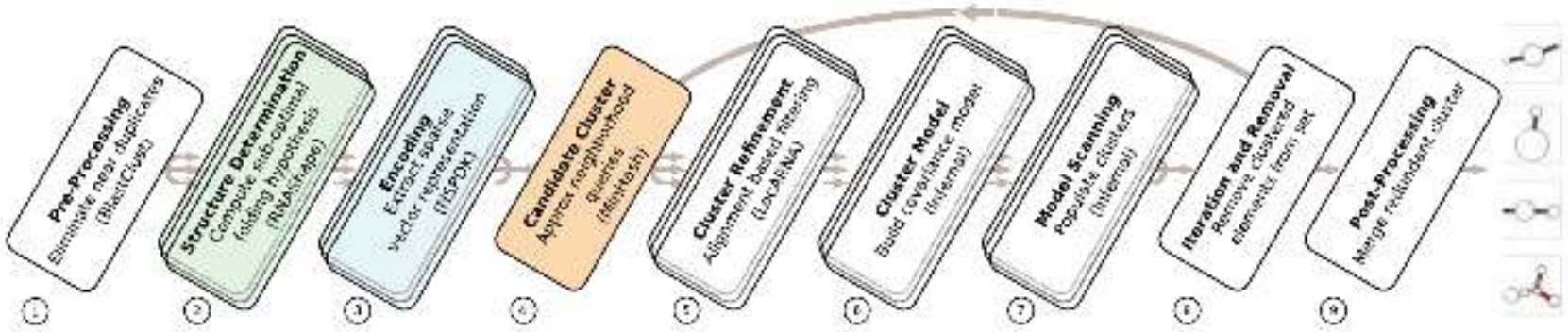


Phase 5. Parallel cluster refinement
Candidate clusters contains sequences, similar under NSPDK measure.
Each cluster adjusts using sequence-structure alignment tool *LocARNA* (Will et al., 2007). We choose only top ranked RNAs in each cluster.



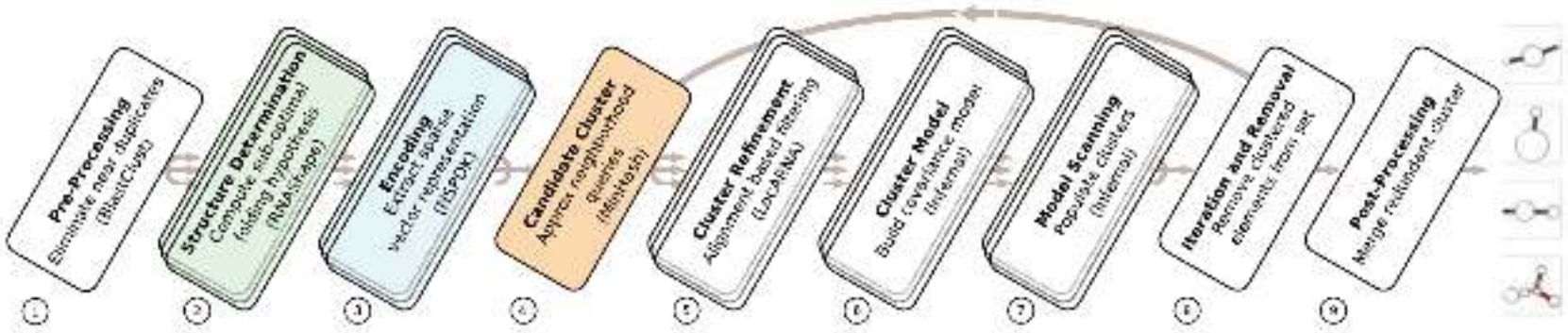
Phase 6. Parallel cluster model
Selected top ranked RNA candidates are realigned with tool *LocARNA-P* (Will et al., 2012). so, we identfy high trusted alignment region and estimate the borders of the common local motif.
At last, we create a covariance model (CM) by applying *INFERNAL* (Nawrocki et al., 2009) tool on the identfied subsequence. Every cluster member gets its CM bitscore[1].

[1] A log-scaled version of score.



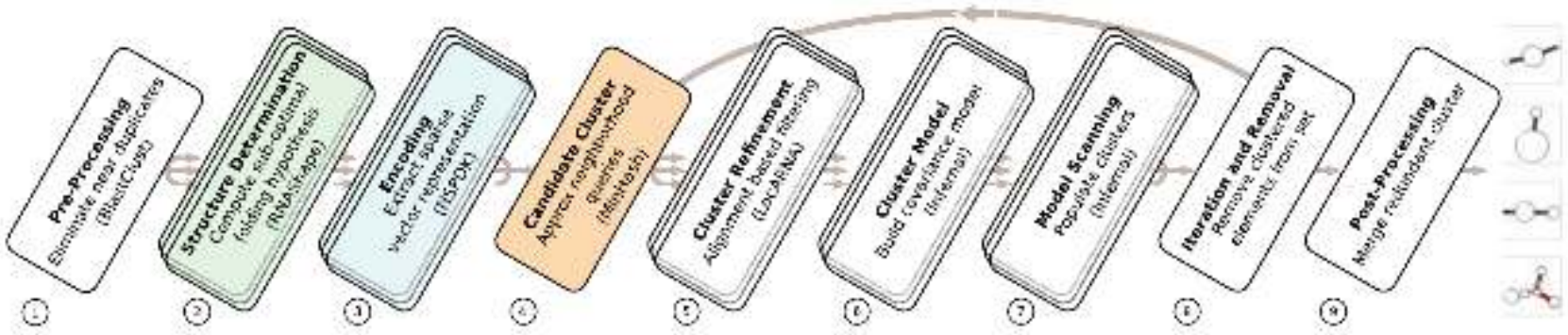
Phase 7. Model scanning
Covariance model of each cluster is used to find additional cluster members in full residual dataset of sequences by CM bitscore or E-value[1].
Some sequences can be assigned to multiplie clusters.

[1] In terms of the authors, the number of instances with a score equivalent or better, than in current instance.



Phase 8. Iteration and removal
All cluster members found in the previous phase are removed from the dataset. A new iteration starts from Phase 4.
The termination conditions are:

- predetermined max number of iterations
- time limit
- empty dataset



Phase 9. Postprocessing

- Merge clusters
  - For every pair of clusters, we compute the relative overlap and merge them if it is more, than 50%.
- Make in-cluster postprocessing
  - Sequences in a cluster finally ranked by their CM bitscore.

Results

Species	Type	Method	Input	Time	Cluster
Benchmark					
Bacteria	Small ncRNAs	Misc	363	6.8h	39
Human	Predicted RNA elemets	EvoFam	699	0.3h	37
Misc	Small ncRNAs	Rfam	3 900	36h	130
De-novo discovery					
Fugu	LincRNAs	RNA-seq	5 877	10.3h	99
Fugu	Predicted RNA elements	RNAz	11 287	13.3h	97
Fruit fly	Predicted RNA elements	RNAz	17 765	20.4h	95
Human	LincRNAs	RNA-seq	31 418	3.6d	95
Human	Predicted RNA elements	EvoFold	37 258	5.7d	117
Human	3'UTRs	RefSeq	118 514	12.8d	106
$\sum$			227 081	25.7d	815

Clustering 3901 sequences using LocARNA takes $\sim$ 370 days.

Q&A

Thank you for your attention!
Q&A