Exploring the Space of Consensus RNA Secondary Structure Motifs Using Suffix Arrays

Truong Nguyen and Marcel Turcotte
School of Information Technology and Engineering

uOttawa

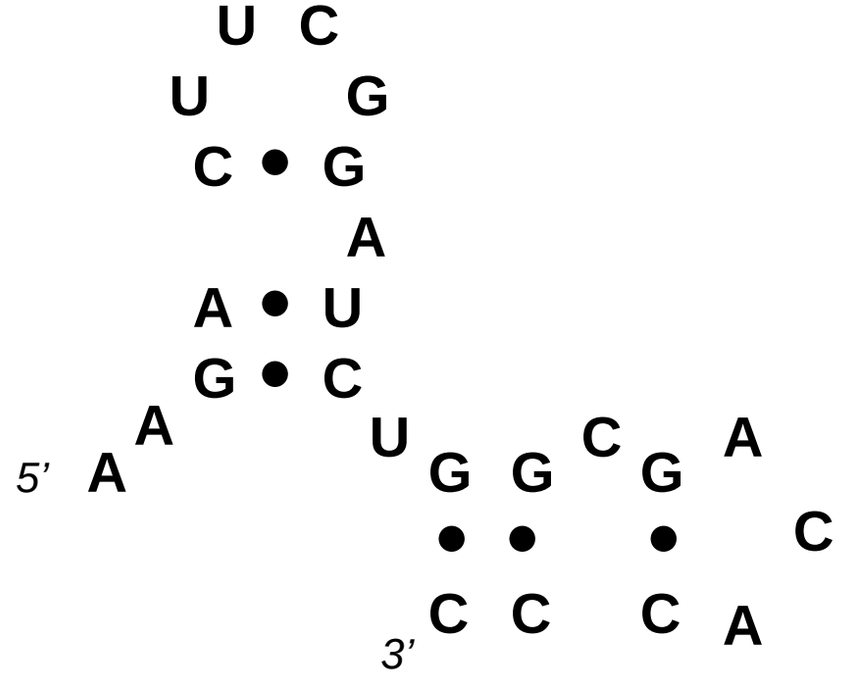
L'Université canadienne
Canada's university

Summary

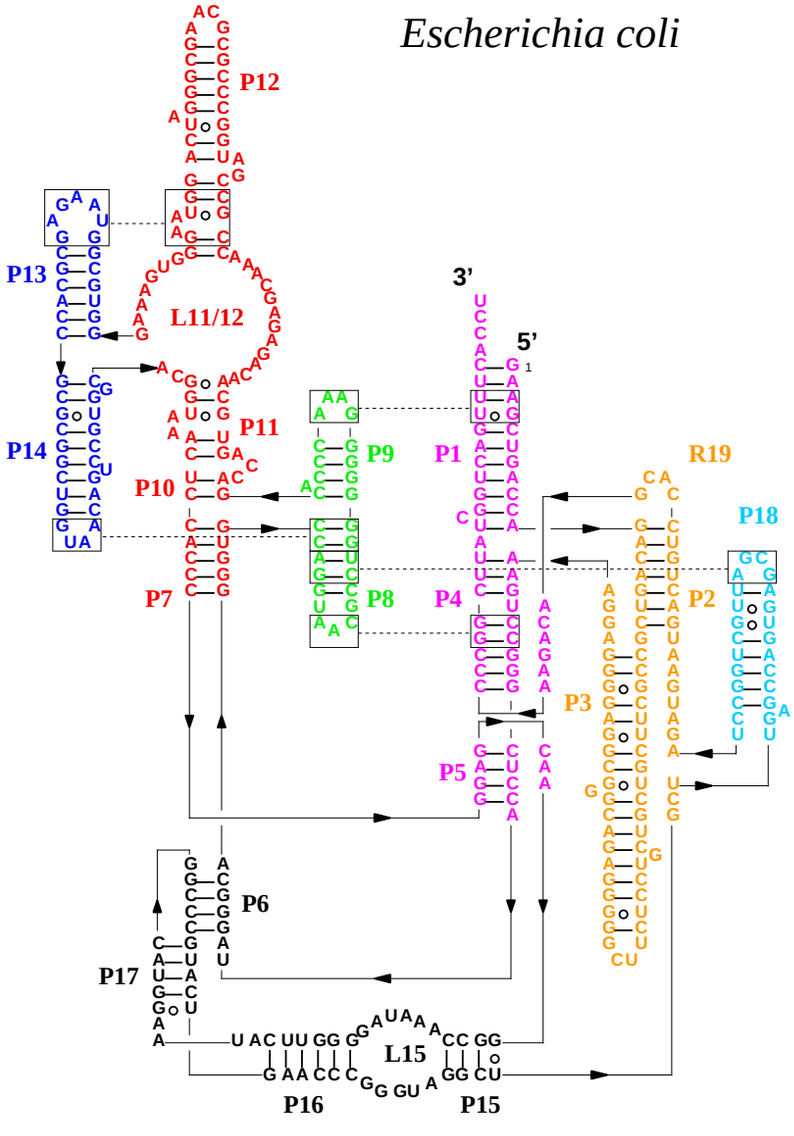
- Novel approach for discovering consensus secondary structure motifs in unaligned RNA sequences;
- Exhaustive exploration of a space induced from a Seed sequence using minimum support contraints;
- Uses suffix arrays for enumerating stems
  (first step of the motif inference algorithm);
- Uses suffix arrays for efficiently matching RNA secondary structure motifs
  (pattern matcher).

This particular phase of the project focuses on the exploration of the search space. We are currently investigating objectives functions in a second phase.

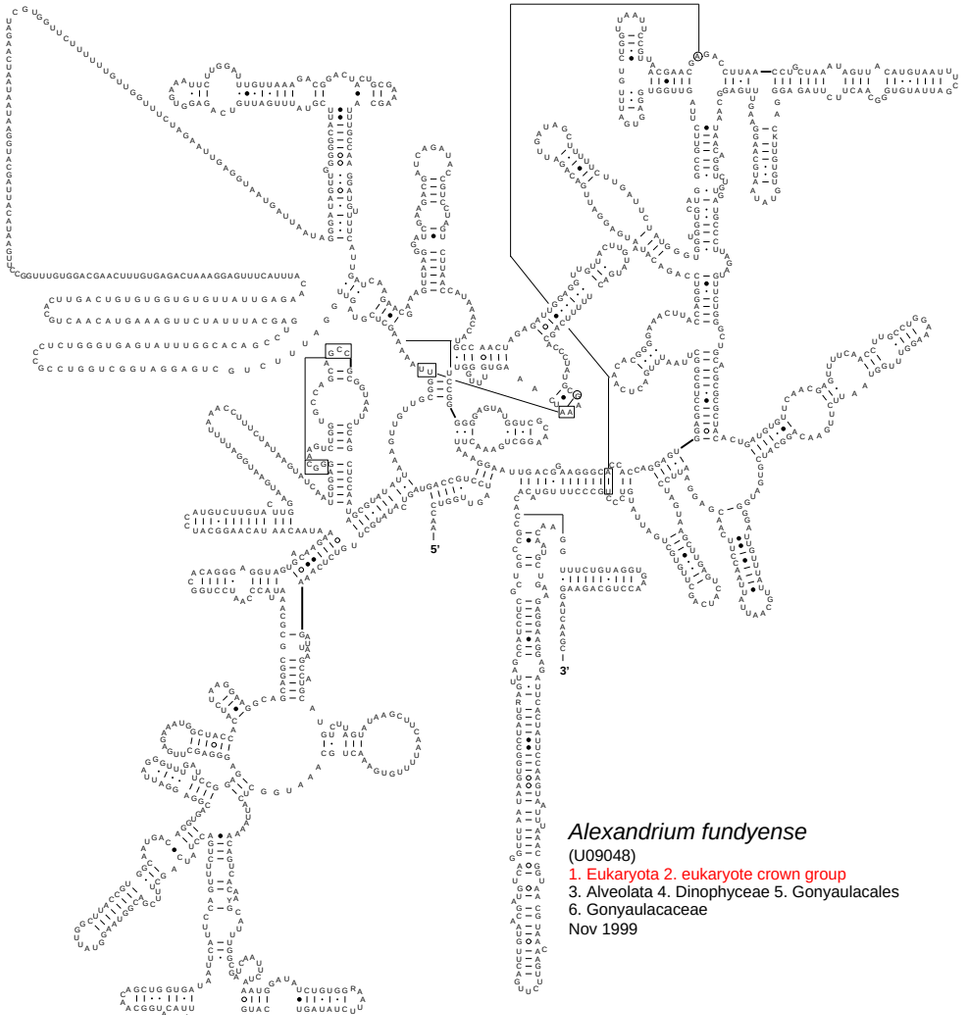
RNA Secondary Structure: Hairpins, Bulges, Loops, MBLs




```
5′ AAGACUUCGGAUCUGGCGACACCC 3′
   ..(((....).)).((.(...)))
```


Escherichia coli
P12
P13
L11/12
P14
P11
P10
P7
P9
P8
3'
5'
P1
P4
P5
R19
P18
P2
P3
P6
P17
L15
P16
P15

Secondary Structure: small subunit ribosomal RNA



Alexandrium fundyense
(U09048)
1. Eukaryota 2. eukaryote crown group
3. Alveolata 4. Dinophyceae 5. Gonyaulacales
6. Gonyaulacaceae
Nov 1999

Motivation

- RNA secondary structure elements are playing an important role in the regulation of gene expression, for example:
  - RNA sensors and riboswitches are regions of untranslated messenger RNAs that can sense the level of metabolites, and modulate the expression of certain genes accordingly;
  - microRNAs regulate the expression of protein genes by targeting a complementary region of their mRNAs;
  - IRES elements enable cap independent translation of protein genes (collaboration Holcik/CHEO).

- Hepatitis *Delta* Virus is a small RNA virus that has limited protein coding capacity, yet unknown RNA secondary structure motifs must be interacting with the host proteome for their replication and proliferation (collaboration Pelchat/BMI).

RNA Secondary Structure Prediction Problem

- **Problem:** find a secondary structure (list of base pairs) with minimum free energy (MFE), for example, according to the nearest neighbor model (NN);

- The average accuracy of secondary structures derived using the NN is maximum when the maximum distance between the elements of pair is less than 150 nt (e.g. tRNAs and 5S, 69% and 71%);

- Even in that range of distances there are several sequences for which the MFE and the native structure are quite different (i.e. low accuracy).

Simultaneous alignment and structure prediction of RNAs

- Beeta Masoumi and Marcel Turcotte. Simultaneous alignment and structure prediction of RNAs: Are three input sequences better than two? In S. V. Sunderam et al., editor, 2005 International Conference on Computational Science (ICCS 2005), Lecture Notes in Computer Science **3515**, pages 936-943, Atlanta, USA, May 22-25 2005;

- Beeta Masoumi and Marcel Turcotte. Simultaneous alignment and structure prediction of three RNA sequences. International Journal of Bioinformatics Research and Applications, *In Press*;

- Pros: higher mean PPV, better worse case scenarios, reproduces well subtle details, such as the variable loop of some tRNAs;

- Cons: $\mathcal{O}(|S_1|^2 M^4)$ space, $\mathcal{O}(|S_1|^3 M^6)$ time.

Why proposing a new method?

We think that existing methods are not appropriate for studying regulatory motifs.

- MFE methods do not scale well;
- Less structured;
- Modular;
- Unaligned;
- Independent (not evolutionary related).

Seed: Research objectives (1/2)

Developing a tool taking as input an ensemble of (unaligned) sequences and producing as output a list of conserved structural **motifs**.

With the following additional constraints:

- No (or little) sequence similarity;
- More than one family present in the input sequences.

Seed: Research objectives (2/2)

For this particular phase of the project, we wanted answers to the following questions.

- Are support and exclusion constraints sufficiently powerful to make an exhaustive search of the secondary structure space feasible?
- Does the search space contain biologically interesting motifs?

Overview (1/6)

- Input: k unaligned sequences;
- Select a Seed sequence;
- Within this search space induced from the Seed sequence report all the motifs that are matching enough of the input sequences (support).

Phase I focused on building an efficient framework for exploring the space of RNA secondary structure motifs.

Phase II (just started) will focus on building effective objective functions.

Overview (2/6)

```
>RD0260 (*)
GCGACCGGGGCUGGCUUGGUAAUGGUACUCCCCUGUCACGGGAGAGAAUGUGGGUUCAAAUCCCAUCGGTCGCGCCA
>RD0500
GCCCGGGUGGUGUAGUGGCCCAUCAUACGACCCUGUCACGGUCGUGACGCGGGUUCAAAUCCCGCCUCGGGCGCCA
>RD1140
GGCCCCAUAGCGAAGUUGGUUAUCGCGCCUCCCUGUCACGGAGGAGAUCACGGGUUCGAGUCCCGUUGGGGUCGCCA
>RD2640
GGGAUUGUAGUUCAAUUGGUCAGAGCACCGCCCUGUCAAGGCGGAAGAUGCGGGUUCGAGCCCCGUCAGUCCCGCCA
>RE2140
GCCCCCAUCGUCUAGAGGCCUAGGACACCUCCCUUUCACGGAGGCGACAGGGAUUCGAAUUCCCUUGGGGGUACCA
>RE6781
UCCGUCGUAGUCUAGGUGGUUAGGAUACUCGGCUCUCACCCGAGAGACCCGGGUUCGAGUCCCGGCGACGGAACCA
>RF6320
GUCGCAAUGGUGUAGUUGGGAGCAUGACAGACUGAAGAUCUGUUGGUCAUCGGUUCGAUCCCGGUUUGUGACACCA
```

In this example, there are 7 input sequences and RD0260 has been selected to be the Seed sequence.

Overview (3/6)

```
[ find_all_stems ]

GCGACCGGGGCTGGCTTGGTAATGGTACTCCCCTGTCACGGGAGAGAATGTGGGTTCAAATCCCATCGGTCGCGCCA

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
(((.............................................................)))

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
(((((((((.....................................................)))))))))

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
(((.....................................................)))
...
                                                                NNNNNNNNNNNNNNNNN
                                                                (((...........)))

                                                                      NNNNNNNNNNN
                                                                      (((.....)))
```

Overview (4/6)

```
[ fix_all ]

GCGACCGGGGCTGGCTTGGTAATGGTACTCCCCTGTCACGGGAGAGAATGTGGGTTCAAATCCCATCGGTCGCGCCA

NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
(((.............................)))

NNGNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNTNN
(((.............................)))

NCNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNGN
(((.............................)))

GNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNT
(((.............................)))

GCNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNGT
(((.............................)))
...
```

Overview (5/6)

```
[ combine_all ]

GCGACCGGGGCTGGCTTGGTAATGGTACTCCCCTGTCACGGGAGAGAATGTGGGTTCAAATCCCATCGGTCGCGCCA

                    GNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNC
                    (((.....................................)))

                                        +

                         NNNTNNNNNNNNGNNN
                         (((((......)))))

                                        =

                    GNNNNNNNTNNNNNNNNGNNNNNNNNNNNNNNNNNNNNNNNNNNNNC
                    (((..(((((......)))))......................)))
```

The motifs with insufficient support are rejected.

Overview (6/6)

```
[ combine_all ]

GCGACCGGGGCTGGCTTGGTAATGGTACTCCCCTGTCACGGGAGAGAATGTGGGTTCAAATCCCATCGGTCGCGCCA

                                CNTNNNNNGNG
                                (((.....)))

                                      +

                                               GNNTNNNNGNNC
                                               ((((....))))

                                      =

                                CNTNNNNNGNGNNNGNNTNNNNGNNC
                                (((.....))).((((....))))
```

Subsequently, the 3 helices motifs, 4 helices motifs . . . will be produced.

Observations

- Huge search space;
- Support and exclusion should be powerful constraints;
- Motifs will be matched against a fix set of sequences (over and over again).

Motif discovery framework

A motif discovery approach can be characterised by,

1. its search space;

2. the algorithm that is used to search the space;

3. its objective function.

A. Brazma, I. Jonassen, I. Eidhammer et D. Gilbert (1998) *Journal of Computational Biology* 5:279-305.

Search space (1/2)

Let $\Sigma = \{A, C, G, T\} \cup \{N, N'\}$, the nucleotides alphabet augment with the joker symbols N and N', where $N \in \{A, C, G, T\}$ and N' is its complement.

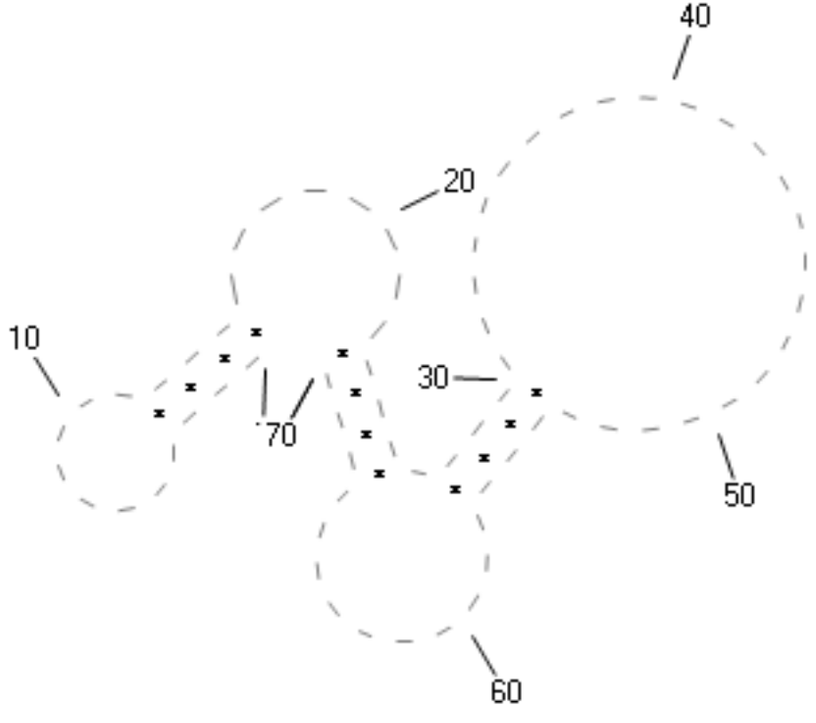
The notation $5 : E$ represents the 5' end of a paired region, and E is a word on Σ.

The notation $3 : E$ represents the 3' end of a paired region, and E is a word on Σ.

The notation $D : n$ represents a distance constraint.

```
5:CNGA D:7 3:TCN'G D:7 5:NNGG D:0 5:NAAG D:23 3:CTTN' D:9 3:CCN'N'
```

Search space (2/2)



5:CNGA D:7 3:TCN’G D:7 5:NNGG D:0 5:NAAG D:23 3:CTTN’ D:9 3:CCN’N’

Seed: Search algorithm

Sequential covering

1. **while** there are more examples

   (a) select an example randomly (Seed sequence);
   (b) build the most specific motif;
   (c) general-to-specific search;
   (d) remove all the examples containing an instance of the “best” motif

The number of families of motifs present in the input sequences is not know *a priori*, this kind of algorithm “may” help uncover this number.

Suffix trees and suffix arrays (1/2)

Give a text T (database) and a pattern P (query), generally s.t. $P << T$

- Tree structure containing all the suffixes of the input text;
- Can be built in linear time w.r.t. the size of the text (T);
- Requires a linear amount of storage w.r.t. the size of T;
- Can answer the decision question, "does P occur in T", in linear time w.r.t. the size of the pattern (P);
- (Exposes all the repeated substrings of T);
- With additional linear time/space processing, can find in constant time the longest common extension for any two pairs of positions.

Suffix trees and suffix arrays (2/2)

- Generalized suffix tree contains all the suffixes for $k > 1$ sequences;
- Can be used for finding the longest common substrings;
- A suffix array is an array that contains the starting positions of all the suffixes in lexicographic order;
- Can be build in linear time, occupies a linear amount of space with a smaller constant than suffix trees, and better locality properties;
- Using a linear amount of storage/time, additional tables can be built (such lcp, rank, childtable) s.t. all the suffix tree algorithms (in particular bottom up/top down traversal) can be simulated on a suffix array;
- Resulting algorithms are simpler to implement.

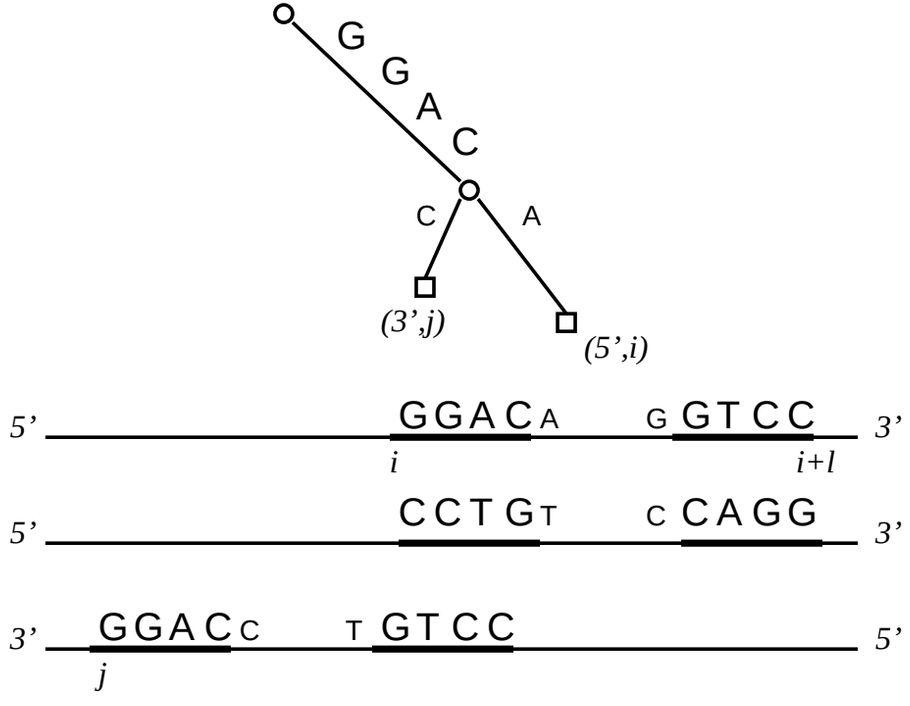
Building the most specific motif (1/4)

Let a be the Seed sequence picked up randomly at the previous step.

All the complementary regions of some minimum length, possibly containing GU base pairs and mismatches are enumerated.

Conceptually done using suffix trees and LCA (Lowest Common Ancestor) — in practice, suffix arrays have been used.

Building the most specific motif (2/4)



$\Rightarrow$ where $j = |S| - i - l + 1$.

Building the most specific motif (3/4)

```
GCGGCGGTGGCTGGCTTGGTAATGAGCAACCGTCGCCACGGGAGAGAATGTGGGTTCAAATCCCATCGGTCGCGCCA
(((X............................X)))
i                                 j
     ((((X..................X))))      -> GU base pair
     i                        j
          (((X..........X)))           -> Mismatch
          i                j

Output

((((((((((((.............))))))))))))
```

Building the most specific motif (4/4)

1. Build a generalised suffix tree for S and $\overline{S}$ — where $\overline{S}$ is the reverse complement of S;

2. Annotate the tree for LCA queries;

3. **For** $i = 1 \ldots |S|$

   (a) **For** $l = c_1 \ldots c_2$

      i. $j = |S| - i - l + 1$

      ii. **If** LCA$((5', i), (3', j)) \geq c_1$ a complementary region has been found.

$\Rightarrow$ The actual algorithm has an additional inner loop allowing for GU base pairs and up to k mismatches.

Exploring the space of sec struc motifs (1/3)

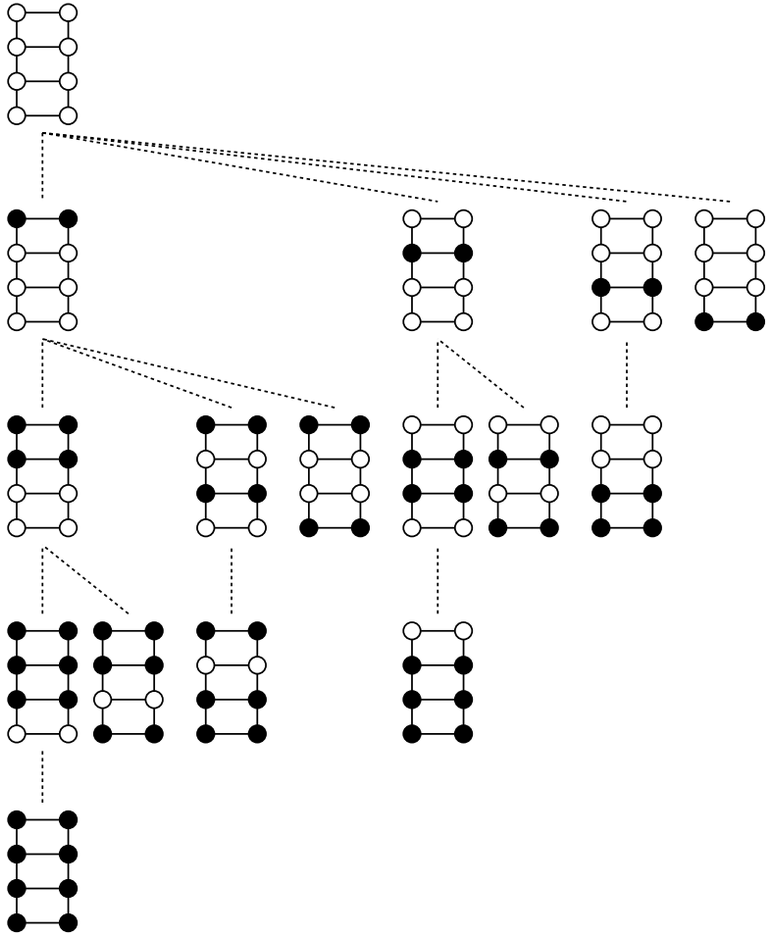
The result of the previous step is a list of biological palindromes for the sequence S — this list possibly contains many conflicting stems.

Each palindrome is first transformed into a generic (structural) motif — all the base pairs are replaced by $N \cdot N'$.

Conceptually, the root of the search tree is ϵ, the empty motif.

Two operations allow to create new motifs (nodes in the search tree): **instantiate** and **combine**.

Instantiate consists of replacing an $N \cdot N'$ base pair by the specific base pair that occurs at that position in the Seed sequence.



Combine creates a new motif by merging two existing motifs. There are two ways to combine motifs, nested or adjacent.

Combining nested motifs

```
  1. 5:NNGG D:40 3:CCN'N'
+
  2. 5:NAAG D:23 3:CTTN'
```

Produces

```
5:NNGG D:0 5:NAAG D:23 3:CTTN' D:9 3:CCN'N'
[    1     [        2       ]      1     ]
```

Combine creates a new motif by merging two existing motifs. There are two ways to combine motifs, nested or adjacent.

Combining adjacent motifs

```
  1. 5:CNGA D:7 3:TCN'G
+
  2. 5:NNGG D:40 3:CCN'N'
```

Produces

```
5:CNGA D:7 3:TCN'G D:7 5:NNGG D:40 3:CCN'N'
[       1         ]     [       2          ]
```

Exploring the space of sec struc motifs (2/3)

By construction, each newly created motif has at least one occurrence.

For each motif, let's define the **support** as the fraction of the k input sequences containing at least one occurrence of this motif.

Any node (motif), and its subtree, is eliminated from the search space if its support is below a user specified threshold, typically 70%.

The current strategy to explore the tree is a breath-first search — this allows the user to exhaustively explore the space of all the motifs containing 1,2,3, etc. stems.

Calculating the support: matching motifs (1/5)

We have developed a non-deterministic algorithm inspired by Baeza-Yates & Gonnet's algorithm for regular expression matching.

Given an input sequence b, $b \neq a$.

The algorithm simultaneously traverses the suffix tree of b and the motif.

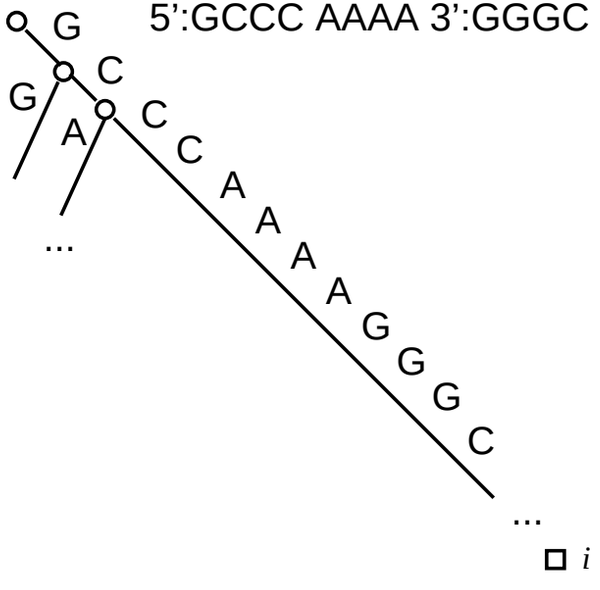
It uses two stacks: the system stack is used to store backtracking points, while an explicit stack is used to validate matching elements of a base pair.

Baeza-Yates RA et Gonnet GH (1996) *J of the ACM* **43**(6):915–936.

Calculating the support: matching motifs (2/5)

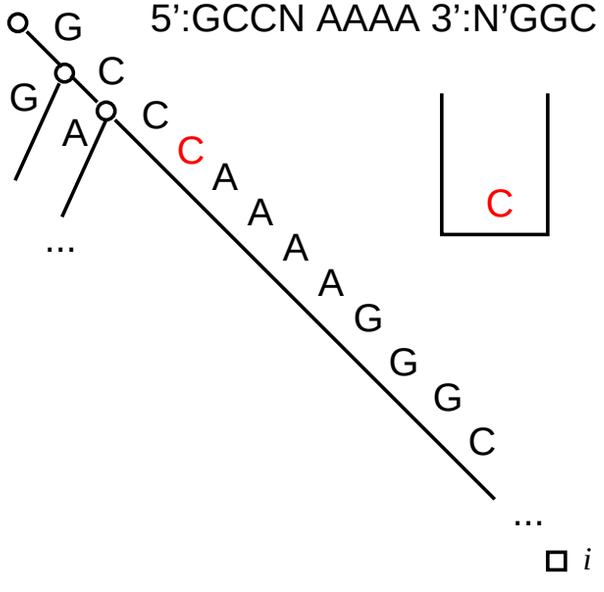
A symbol $\{A, C, G, T\}$ of the motif corresponds to itself in the suffix tree.

Matching a fully instantiated motif can be done in linear time w.r.t. the size of the motif.



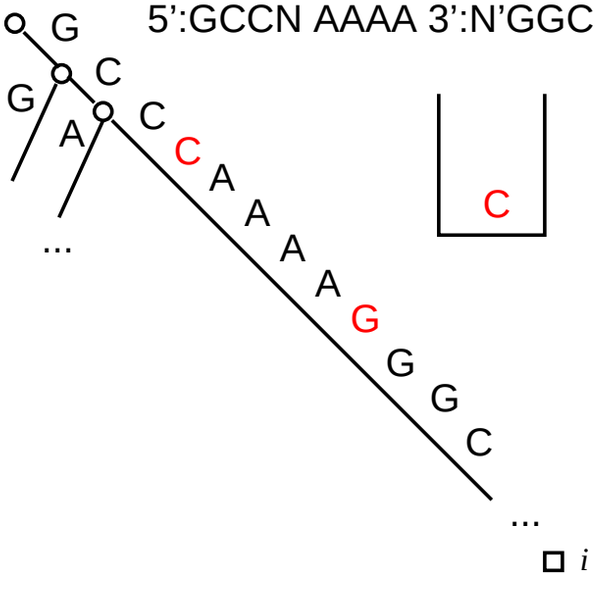
Calculating the support: matching motifs (3/5)

When a joker, N, is found in the 5' end of a stem region, and the last match occurred inside a label, the next character on that branch is pushed onto the base pair stack.



Calculating the support: matching motifs (4/5)

When a joker, N', is found in the 3' end of a stem region, the algorithm succeeds only if the next matching character can form a base pair with the element that is found on the top of the stack, then the top element is removed, and the algorithm continues.



Calculating the support: matching motifs (5/5)

Finally, when a joker is found, N, and the algorithm had stop at an interior node, all the branches are explored recursively (source of non-determinism).

Expressions such as this one, $D : n$, are processed similarly.

Exploring the space of sec struc motifs (3/3)

The algorithm stops if there are no more valid open nodes, or a user defined stopping criteria stops the algorithm.

The motifs are ranked and returned to the user.

Objective function: information content

We currently use a simple objective function simply defined as the information content.

Implementation (1/3)

Suffix arrays are used rather than suffix trees.

Given an input sequence S of length $|S| = n$.

Each suffix is represented by its starting position (an integer), a suffix array lists all the suffixes in lexicographic order.

Uses $\mathcal{O}(n)$ space; with small constant.

$\log_2 n$ bits suffice to represent a position, hence, 32 bits, $4 \times n$ bits, are enough to represent a 4 Gbytes string.

Implementation (2/3)

Manber U et Myers G (1990) *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*: 319 – 327.

Manber U and Myers G (1993) *SIAM J on Computing* **22**(5):935–948.

Until very recently constructing a suffix array was costly, $\mathcal{O}(n \log n)$.

Building in $\mathcal{O}(n)$ time.

Kärkkäinen J et Sanders P (2003) In *Proc. 30th International Colloquium on Automata, Languages and Programming (ICALP '03), LNCS 2719*, 943-955. (Skew algorithm)

Implementation (3/3)

Bottom up traversal,

Abouelhoda M et al. (2003) WABI 2002, *LNCS 2452* :449-463.

Top down traversal,

Abouelhoda M et al. (2002) SPIRE 2002, *LNCS 2476* :31-43.

See Abouelhoda et al for an excellent review.

Mohamed Ibrahim Abouelhoda and Stefan Kurtz Enno Ohlebusch Replacing suffix trees with enhanced suffix arrays (2004) J. of Discrete Algorithms **(2):1**, 53–86.

16,000+ lines of C (now shrinked to some 8,000 lines).

Timing Results: Matcher (1/2)

Hordeum vulgare $tRNA^{Glu}$ (RE6781)

```
> Full
UCCGUCGUAGUCUAGGUGGUUAGGAUACUCGGCUCUCACCCGAGAGACCCGGGUUCGAGUCCCGGCGACGGAACCA
(((((((..((((.........)))).((((.......)))).....((((.......))))))))))).....

> N_Loop
UCCGUCGNNGUCUNNNNNNNNNNGGAUNCUCGGNNNNNNNNCCGAGNNNNCCGGGNNNNNNNNCCCGGCGACGGANNNN
(((((((..((((.........)))).((((.......)))).....((((.......))))))))))).....

> N_Stem
NNNNNNNUANNNNAGGUGGUUANNNNANNNNNCUCUCACNNNNNAGACNNNNNUUCGAGUNNNNNNNNNNNNNACCA
(((((((..((((.........)))).((((.......)))).....((((.......))))))))))).....

> Generic
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
(((((((..((((.........)))).((((.......)))).....((((.......))))))))))).....
```

Find all matches allowing 1 mismatch in *Bacillus anthracis* 5,227,293 bp

Timing Results: Matcher (2/2)

# Nuc	Vtree	Full	N_Loop	N_Stem	Generic
512,000	0.7	0.000196	0.000270	0.37	0.80
1,024,000	2.0	0.000198	0.000379	0.68	1.67
2,048,000	4.6	0.000239	0.000618	1.32	3.56
4,096,000	10.3	0.000275	0.001201	2.49	7.75

Times in seconds on Sun Fire V20z 2 $\times$ AMD Opteron 248 (2.2 GHz), 8 Gb, Solaris 9 (a single processor was used).

Friday's Results (1/5)

Used a difficult test set for mfold.

tRNA	Coverage	Accuracy
RD0500	58.8	43.5
RD1140	100.0	100.0
RD2640	66.7	63.6
RE2140	95.2	87.0
RE6781	33.3	28.0
RF6320	0.0	0.0
	59.0	53.7

Friday's Results (2/5)

```
$ seed --min_num_stem 3 --max_num_stem 100 --range 2 \
       --save_all_matches --save_as_ct examples/tRNAs-2.fas

Seed 1.0 [Jul 22 2005] - RNA secondary structure motif inference

Copyright (C) 2003-05 University of Ottawa
All Rights Reserved

This program is distributed under the terms of the
GNU General Public License. See the source code
for details.

[ find_all_stems ]
[ size of the motif list is 164 ]
[ filter_by_support ]
[ size of the motif list is 147 ]
```

Friday's Results (3/5)

```
[ filter_keep_longest_stems ]
[ size of the motif list is 89 ]
[ fix_all ]
[ size of the motif list is 445 ]
[ combine_all ]
[ generating all 2 stems motifs ]
[ size of the motif list is 445 ]
[ generating all 3 stems motifs ]
[ size of the motif list is 1939 ]
[ generating all 4 stems motifs ]
[ size of the motif list is 1991 ]
[ done ]
[ size of the motif list is 1991 ]
[ postprocess ]
[ size of the motif list is 52 ]
[ elapsed time 1 minutes, 54 seconds ]
[ total number of match operations is 520835 ]
[ save_matches_as_ct ]
```

Friday's Results (4/5)

Motif 000012: coverage (ave = 38.0, min = 28.6, max = 47.1), accuracy (average = 76.0, min = 60.0, max = 90.0).

```
Seed
  NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
  (((....................(((..........)))..((((..........))))..)))

Actual
(((((((..((((.........)))).(((((.......))))).....(((((.......))))))))))))....
         ^^^^         ^^^^
```

Friday's Results (5/5)

A sightly different run, using `--skip_keep_longest_stems`, increases the cpu time some 11 minutes 52 seconds, however the search space contain motifs with higher accuracy and coverage, e.g. motif = 1073: average maximum coverage = 43.5%, average maximum accuracy = 98.4, motif = 93, average maximum coverage = 60%, average maximum accuracy = 96.9 %.

Conclusions: Seed

- A suffix tree/array based approach allows us to enumerate a substantial fraction of the search space, using a reasonable amount of resources;

- The search space contains biologically interesting candidates.

Future work

- Developing better objective functions (MDL, NN);
- Adding sequence patterns in the loop regions.

eXtended Dynalign

- Sankoff 1985 proposed a set of recurrence equations for simultanesouly solving the alignment and secondary structure determination problems;

  David Sankoff (1985) Simultaneous solution of RNA folding, alignment and protosequence problems. SIAM J. Appl. Math. **45**(5):810–825.

- Objective function is a linear combination of the free energy of each sequence given the common secondary structure;

- Mathews and Turner 2002 created an implementation, called Dynalign, for two sequences;

  D.H. Mathews et D.H. Turner (2002) Dynalign: An Algorithm for Finding the Secondary Structure Common to Two RNA Sequences. J. Mol. Biol. **317**:191–203.

Collaborators

University of Ottawa

Truong Nguyen (M.Sc. student)
Beeta Masoumi (M.Sc. student)

Children's Hospital of Eastern Ontario (CHEO) — Molecular Genetics

Stephen Baird (Ph.D. student)
Martin Holcik (Group leader)
Robert Korneluk (Director)

University of Ottawa — Biochemistry, Microbiology and Immunology

Martin Pelchat (Group leader)

Information

bio.site.uottawa.ca (home page)

bio.site.uottawa.ca/wiki/space/start (news)

bio.site.uottawa.ca/software/seed (downloads and reprints)

bio.site.uottawa.ca/software/x-dynalign (downloads and reprints)

turcotte@site.uottawa.ca (E-mail)