Efficient structural Clustering of Non-Coding RNA

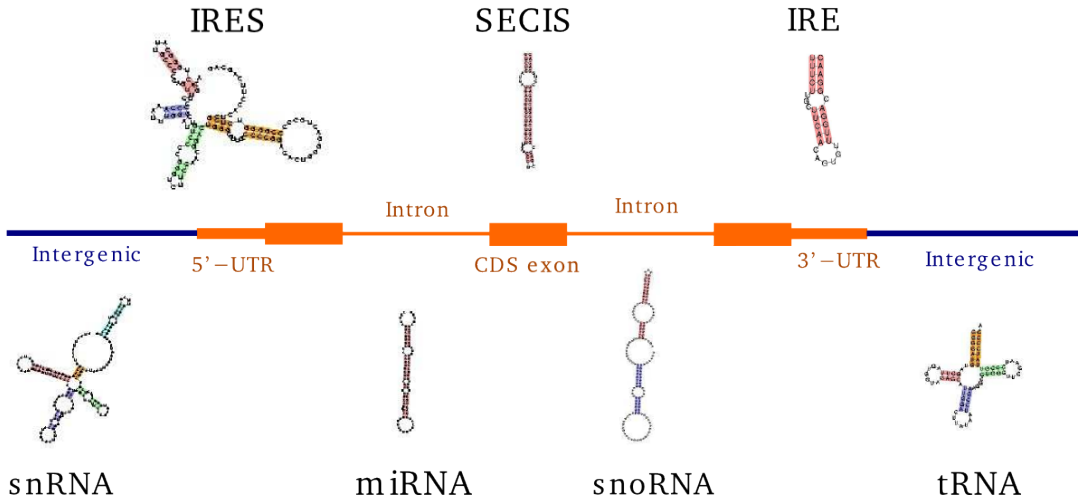
Steffen Heyne

Bioinformatics Group
Albert Ludwigs University Freiburg

Bled, 2012

RNA families

- RNAs can be classified by their structure or function
- RNA structure is more conserved than sequence



Issues

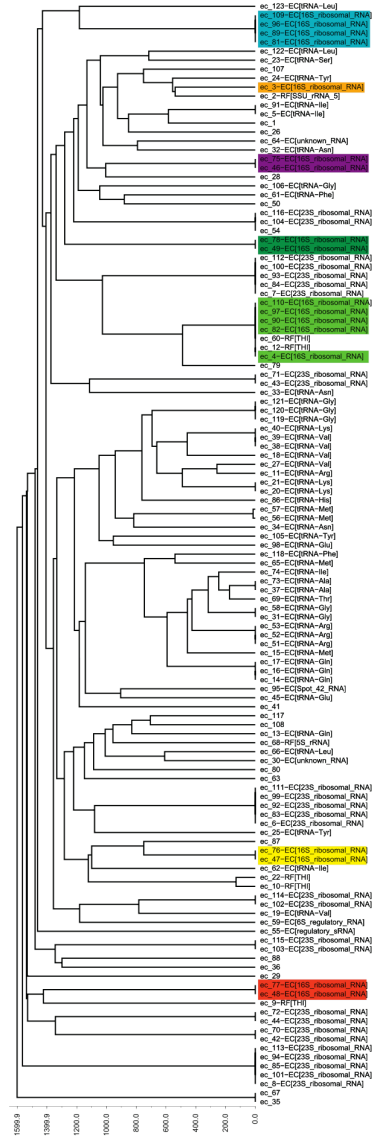
- Prediction methods like RNAz and EvoFold predict thousands of unknown RNAs
- Sequencing data reveals thousends of unknown RNAs

Existing Clustering Approaches

1. All vs. all comparison by sequence structure alignment tools, e.g FoldAlign, LocARNA
2. Stem kernel, BPLA kernel (Sato et al)
3. All vs. all comparison by BLAST, Stral, Mafft

Issues

- BLAST is possible for large instances, but identifies only reliable clusters above 70% mean pairwise identity
- Structural alignment works well for less conserved RNAs, but only some thousands sequences manageable , $O(n^4)$
- Clusters usually taken from "guide tree"



LocARNA Cluster tree (Will et al., 2007)

The proposal in a nutshell

1. Extract candidate ncRNA fragments (10K seq of 100 nt)
   Contribution: use RNAshapes to get representitve structures of RNA candidates
2. Cluster fragments according to sequence **and** structure
   Contribution: use graph kernel and locality sensitive hashing
   $\mapsto$ *linear efficiency*
3. Refine clusters/families C_i (via structural alignment)
   Contribution: use global (future: local) alignment for cluster tree, subtree ranking
4. Make models for C_i, scan genome to collect and remove all members of C_i
5. Iterate and find additional clusters/families

The proposal in a nutshell

1. Extract candidate ncRNA fragments (10K seq of 100 nt)
   Contribution: use RNAshapes to get representitve structures of RNA candidates
2. Cluster fragments according to sequence **and** structure
   Contribution: use graph kernel and locality sensitive hashing
   $\mapsto$ *linear efficiency*
3. Refine clusters/families C_i (via structural alignment)
   Contribution: use global (future: local) alignment for cluster tree, subtree ranking
4. Make models for C_i, scan genome to collect and remove all members of C_i
5. Iterate and find additional clusters/families

NSPDK Graph Kernel (Costa and Grave, 2010)

- Neighbourhood Subgraph Pairwise Distance Kernel
- NSPDK suitable for large datasets of sparse graphs with discrete vertex and edge labels
- Decomposition kernel, parts: pairs of subgraphs
- Efficiency: limit radius and distance of subgraphs
- Efficiency: graph serialization to check for identical strings
- Iterative hashing maps string encoding to integer code
- Linear w.r.t. vertex size of the component on sparse graphs with bounded degree

NSPDK Graph Kernel (Costa and Grave, 2010)

- Graphs originate from RNA sequences (structures)
- New: materialize implicit feature encoding -> integer code is feature key and value is (normalized) count of occurences
- Efficiency: number of features is limited to $O(r^* d^* |V(G)|^2)$, even to $O(|V(G)|)$ as in a sparse graph only small number of vertices is reachable within a fixed d
- Explicit feature encoding: sparse vector in $\mathbb{R}^m$ (with a very high dimension m) for a given graph G with linear number of non-zero features
- Overall encoding comlexity is linear in its vertex set size (with hidden coefficient)

Efficient Clustering - Features

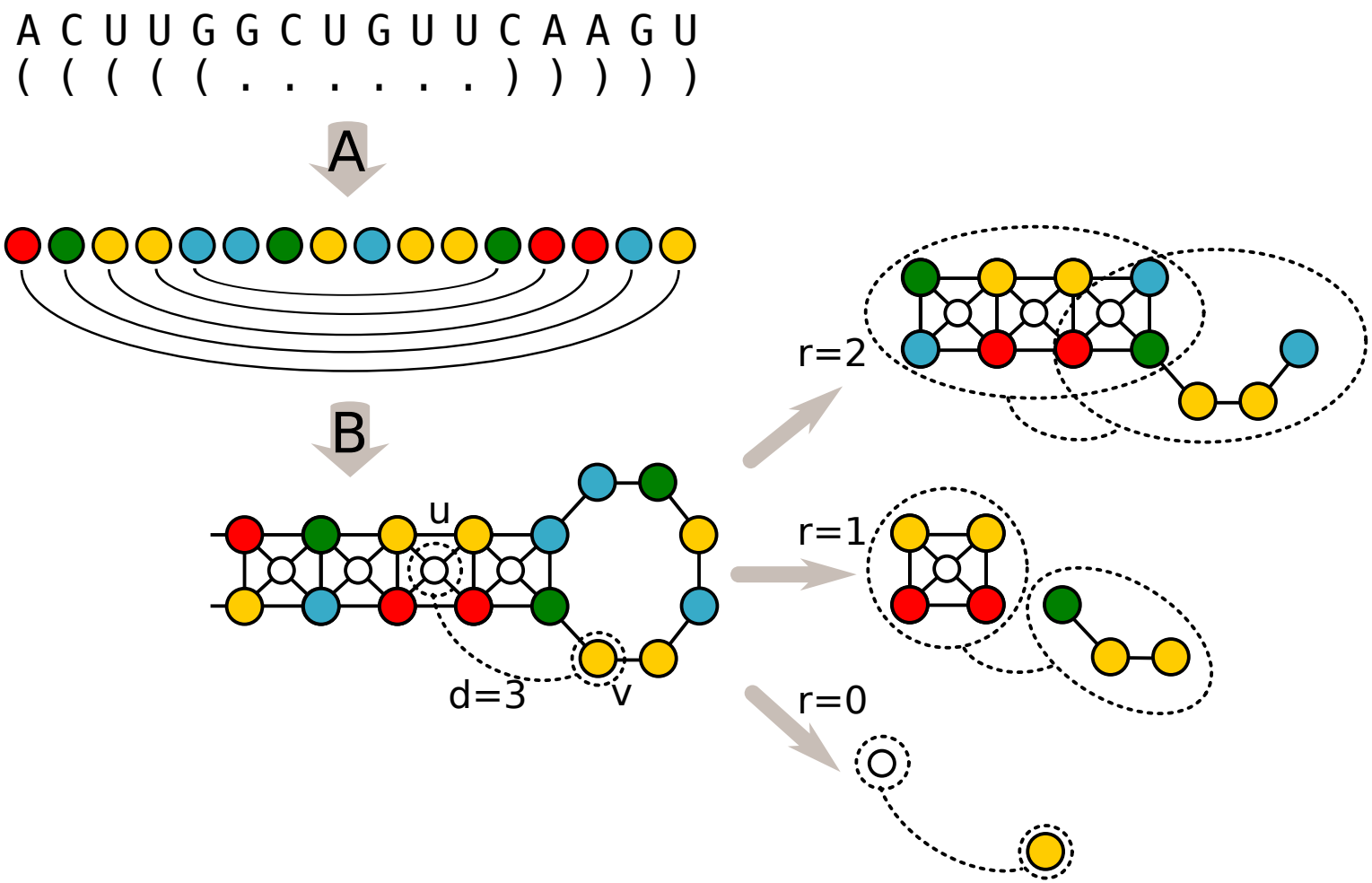


Figure: **RNA secondary structure encoding and Graph Kernel Features.** Top A) The graph encoding is information preserving. B) The importance of features derived from stacking base-pairs quadruplets is emphasized using additional vertices. Bottom: Example of features induced by the graph kernel `NSPDK` for a pair of vertices u, v at distance 3 with radius 0,1,2.

How to get clustering fast?

- Approximate nearest neighbor queries (sublinear), Indyk and Motwani (1998) proposed efficient solution for high dimensional cases
- locality-sensitive hash function: prob. of collision is higher for objects that are close
- Min-Hash function (Broder, 1997): approx. Jaccard index $s(x,z) = \frac{|x \cap z|}{|x \cup z|}$, require sparse binary vectors $\mathbb{R}^m \mapsto \{0,1\}^m$
- set of random hash functions: $h_i : \mathbb{N} \mapsto \mathbb{N}$
- surprise: $P(mh(x,h_i) = mh(z,h_i)) = \frac{|x \cap z|}{|x \cup z|} = s(x,z)$
  -> min-hash collision is estimator of Jaccard similarity
- for N min-hash functions: n/N is estimator for $s(x,z)$ with expected error $O(1/\sqrt{N})$, e.g. $N = 400$: error ≤ 0.05

How to get clustering fast?

- Approximate nearest neighbor queries (sublinear), Indyk and Motwani (1998) proposed efficient solution for high dimensional cases
- locality-sensitive hash function: prob. of collision is higher for objects that are close
- Min-Hash function (Broder, 1997): approx. Jaccard index $s(x,z) = \frac{|x \cap z|}{|x \cup z|}$, require sparse binary vectors $\mathbb{R}^m \mapsto \{0,1\}^m$
- set of random hash functions: $h_i : \mathbb{N} \mapsto \mathbb{N}$
- surprise: $P(mh(x,h_i) = mh(z,h_i)) = \frac{|x \cap z|}{|x \cup z|} = s(x,z)$
  -> min-hash collision is estimator of Jaccard similarity
- for N min-hash functions: n/N is estimator for $s(x,z)$ with expected error $O(1/\sqrt{N})$, e.g. $N = 400$: error ≤ 0.05

How to get clustering fast?

- Approximate nearest neighbor queries (sublinear), Indyk and Motwani (1998) proposed efficient solution for high dimensional cases
- locality-sensitive hash function: prob. of collision is higher for objects that are close
- Min-Hash function (Broder, 1997): approx. Jaccard index $s(x,z) = \frac{|x \cap z|}{|x \cup z|}$, require sparse binary vectors $\mathbb{R}^m \mapsto \{0,1\}^m$
- set of random hash functions: $h_i : \mathbb{N} \mapsto \mathbb{N}$
- surprise: $P(mh(x,h_i) = mh(z,h_i)) = \frac{|x \cap z|}{|x \cup z|} = s(x,z)$
  -> min-hash collision is estimator of Jaccard similarity
- for N min-hash functions: n/N is estimator for $s(x,z)$ with expected error $O(1/\sqrt{N})$, e.g. $N = 400$: error ≤ 0.05

How to get clustering fast?

- Approximate nearest neighbor queries (sublinear), Indyk and Motwani (1998) proposed efficient solution for high dimensional cases
- locality-sensitive hash function: prob. of collision is higher for objects that are close
- Min-Hash function (Broder, 1997): approx. Jaccard index $s(x,z) = \frac{|x \cap z|}{|x \cup z|}$, require sparse binary vectors $\mathbb{R}^m \mapsto \{0,1\}^m$
- set of random hash functions: $h_i : \mathbb{N} \mapsto \mathbb{N}$
- surprise: $P(mh(x,h_i) = mh(z,h_i)) = \frac{|x \cap z|}{|x \cup z|} = s(x,z)$
  -> min-hash collision is estimator of Jaccard similarity
- for N min-hash functions: n/N is estimator for $s(x,z)$ with expected error $O(1/\sqrt{N})$, e.g. $N = 400$: error ≤ 0.05

How to get clustering fast?

- Approximate nearest neighbor queries (sublinear), Indyk and Motwani (1998) proposed efficient solution for high dimensional cases
- locality-sensitive hash function: prob. of collision is higher for objects that are close
- Min-Hash function (Broder, 1997): approx. Jaccard index $s(x,z) = \frac{|x \cap z|}{|x \cup z|}$, require sparse binary vectors $\mathbb{R}^m \mapsto \{0,1\}^m$
- set of random hash functions: $h_i : \mathbb{N} \mapsto \mathbb{N}$
- surprise: $P(mh(x,h_i) = mh(z,h_i)) = \frac{|x \cap z|}{|x \cup z|} = s(x,z)$
  -> min-hash collision is estimator of Jaccard similarity
- for N min-hash functions: n/N is estimator for $s(x,z)$ with expected error $O(1/\sqrt{N})$, e.g. $N = 400$: error ≤ 0.05

How to get clustering fast?

- Approximate nearest neighbor queries (sublinear), Indyk and Motwani (1998) proposed efficient solution for high dimensional cases
- locality-sensitive hash function: prob. of collision is higher for objects that are close
- Min-Hash function (Broder, 1997): approx. Jaccard index $s(x,z) = \frac{|x \cap z|}{|x \cup z|}$, require sparse binary vectors $\mathbb{R}^m \mapsto \{0,1\}^m$
- set of random hash functions: $h_i : \mathbb{N} \mapsto \mathbb{N}$
- surprise: $P(mh(x,h_i) = mh(z,h_i)) = \frac{|x \cap z|}{|x \cup z|} = s(x,z)$
  -> min-hash collision is estimator of Jaccard similarity
- for N min-hash functions: n/N is estimator for $s(x,z)$ with expected error $O(1/\sqrt{N})$, e.g. $N = 400$: error ≤ 0.05

How to get clustering fast?

- instance signature: result of all N min-hash functions
- build reverse index, constant for fixed N -> linear for dataset
- returned neighbors are resorted by true NSPDK similarity to x, take k-closest elements as the $k-$neighborhood $N_k(x)$
- neighbors can be returned in constant time
- candidate clusters: rank each x by density of its k-neighborhood, take top ranking neighborhoods
- even faster: compute neighborhood only for a sample of input instances

How to get clustering fast?

- instance signature: result of all N min-hash functions
- build reverse index, constant for fixed N -> linear for dataset
- returned neighbors are resorted by true NSPDK similarity to x, take k-closest elements as the $k-$neighborhood $N_k(x)$
- neighbors can be returned in constant time
- candidate clusters: rank each x by density of its k-neighborhood, take top ranking neighborhoods
- even faster: compute neighborhood only for a sample of input instances

How to get clustering fast?

- instance signature: result of all N min-hash functions
- build reverse index, constant for fixed N -> linear for dataset
- returned neighbors are resorted by true NSPDK similarity to x, take k-closest elements as the $k-$neighborhood $N_k(x)$
- neighbors can be returned in constant time
- candidate clusters: rank each x by density of its k-neighborhood, take top ranking neighborhoods
- even faster: compute neighborhood only for a sample of input instances

How to get clustering fast?

- instance signature: result of all N min-hash functions
- build reverse index, constant for fixed N -> linear for dataset
- returned neighbors are resorted by true NSPDK similarity to x, take k-closest elements as the $k-$neighborhood $N_k(x)$
- neighbors can be returned in constant time
- candidate clusters: rank each x by density of its k-neighborhood, take top ranking neighborhoods
- even faster: compute neighborhood only for a sample of input instances

How to get clustering fast?

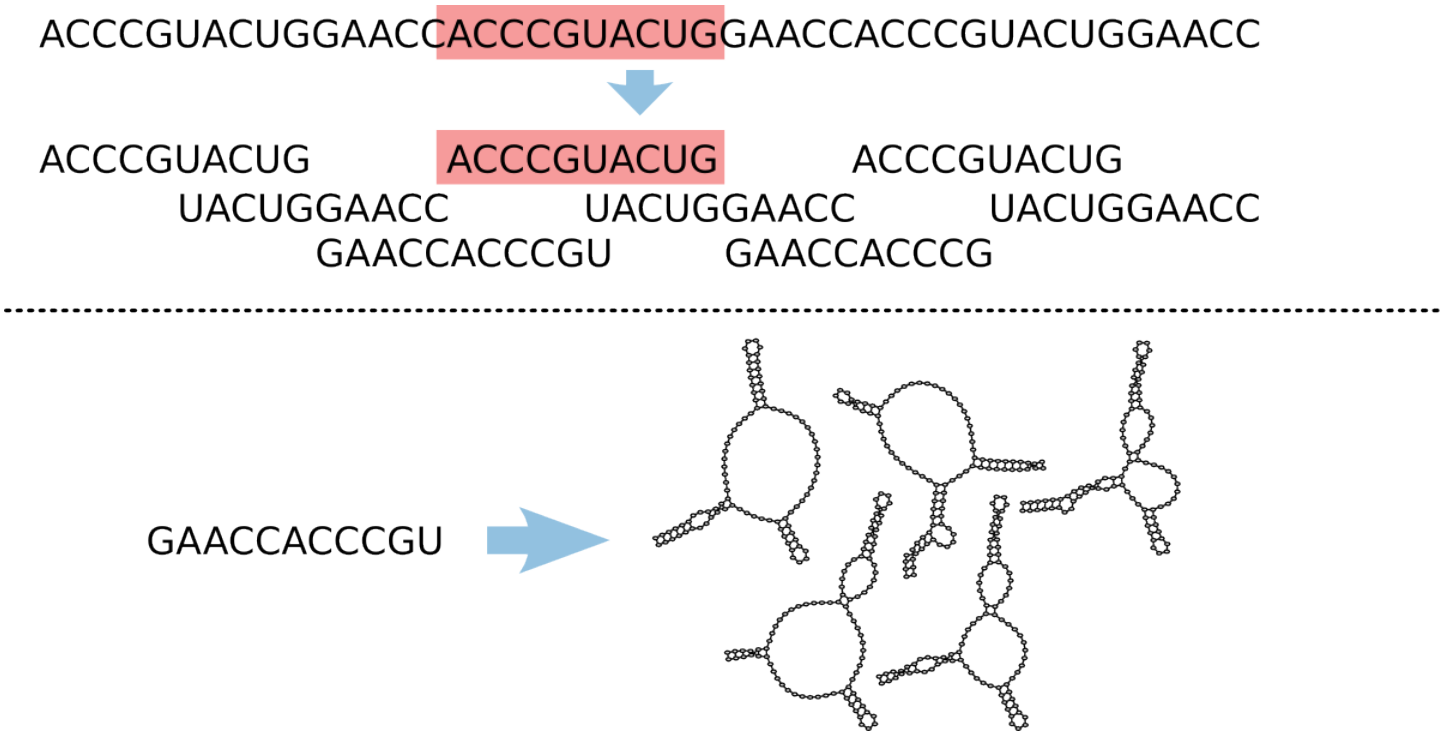
- instance signature: result of all N min-hash functions
- build reverse index, constant for fixed N -> linear for dataset
- returned neighbors are resorted by true NSPDK similarity to x, take k-closest elements as the $k-$neighborhood $N_k(x)$
- neighbors can be returned in constant time
- candidate clusters: rank each x by density of its k-neighborhood, take top ranking neighborhoods
- even faster: compute neighborhood only for a sample of input instances

How to get clustering fast?

- instance signature: result of all N min-hash functions
- build reverse index, constant for fixed N -> linear for dataset
- returned neighbors are resorted by true `NSPDK` similarity to x, take k-closest elements as the $k-$neighborhood $N_k(x)$
- neighbors can be returned in constant time
- candidate clusters: rank each x by density of its k-neighborhood, take top ranking neighborhoods
- even faster: compute neighborhood only for a sample of input instances

Representing ncRNA as graphs

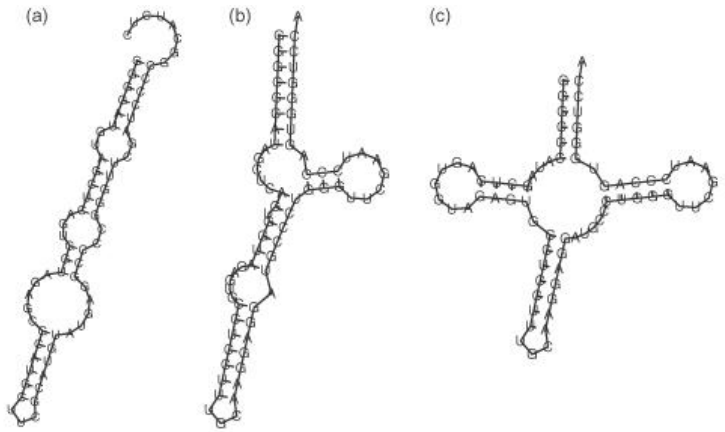
- Given a ncRNA sequence consider all substrings obtained as windows of size $W_1, W_2, \ldots, W_p$ at intervals $I_1, I_2, \ldots, I_m$
- Consider a set of k most representative structures for each subsequence
- $\Rightarrow$ graph with disconnected components



Representative structures

- Sample set of folding structures...
- which exhibit significantly different shapes (*abstraction levels*)
- in a small energy range above the minimum free energy
  $\mapsto$ representative structures
- 150nt seq: $\approx$ 2500 vertices, windows $= \{30, 150\}$, shift 20%,
  $\approx$ 8000 features with $r = 2$ and $d = 4$

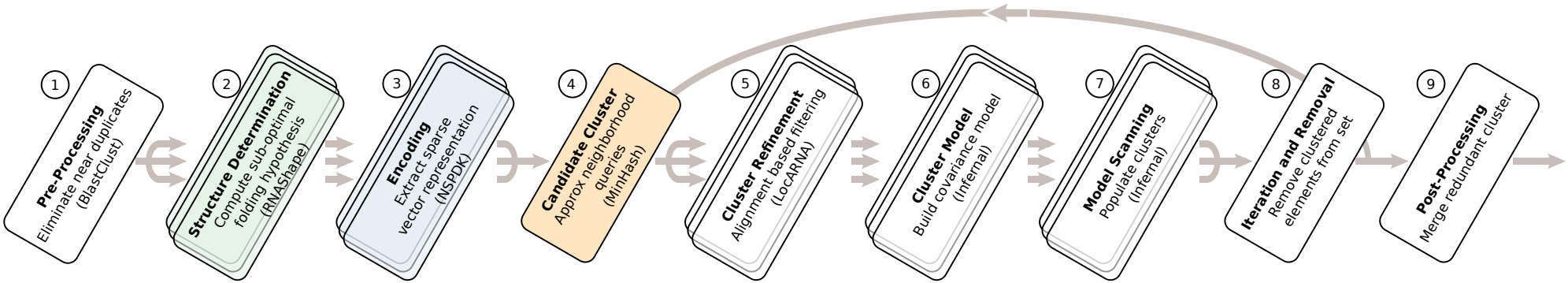
R. Giegerich, B, Voß and M. Rehmsmeier, "Abstract shapes of RNA", NAR (2004)



(d)

```
Shape     GGGCCCAUAGCUCAGUGGUAGAGUGCCUCCUUUGCAAGGAGGAUGCCCUGGGUUCGAAUCCCAGUGGGUCCA
[]        ((((((((((((((.((((.....(((((((...)))))))..)))))))))))).........)))))))).  -35.9 kcal/mol
[[][]]    ((((((((.....(.((((.....(((((((...)))))))..)))))))(((.......))).)))))))).  -32.2 kcal/mol
[[][][]]  ((((((...((((.......)))).(((((((...)))))))).....(((((.......))))).))))))).  -31.7 kcal/mol
```

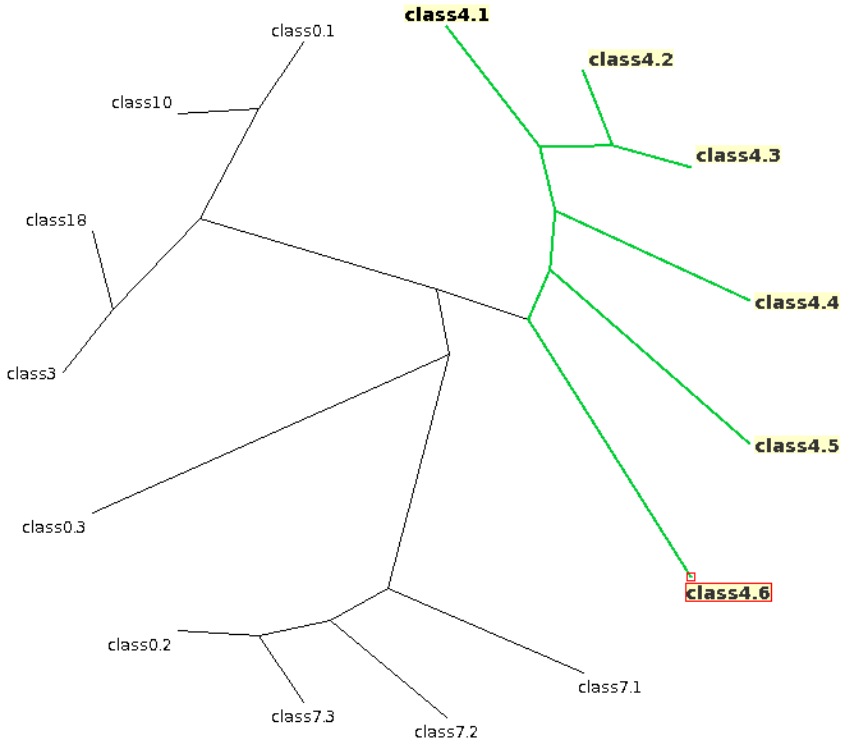
The Pipiline



Complete clustering pipeline diagram. Phases that are executed in parallel are represented in stacked boxes. 1) filter near-duplicates, 2) compute sub-optimal structures, 3) compute sparse vector encoding, 4) compute global feature index and return top dense sets, 5) refine clusters with structural alignment procedure, 6) build covariance model with remaining high quality instances, 7) populate each cluster with retrieved instances, 8) remove clustered instances and iterate from step 4, 9) merge redundant clusters.

Cluster refinement

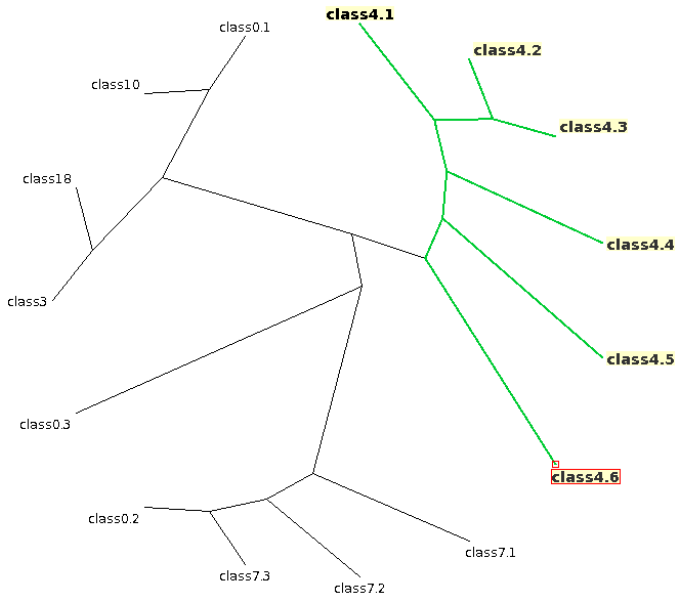
1. Take candidate cluster (dense region with $k = 10..25$)
2. Align cluster all vs. all by structural alignment (LocARNA)
3. build cluster tree and rank subtrees



LocARNA Cluster tree with known class labels

Structural Alignment - issues

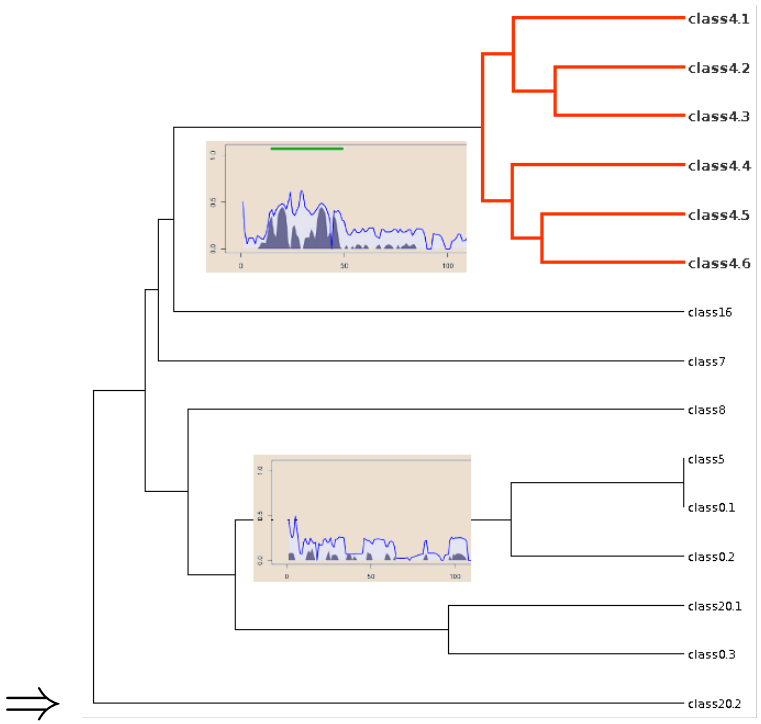
- Ideally independent of input fragments (method)
- Unknown “location" of signal in each input seq
- Use global, semi-global or local alignment for all vs. all?
- Global or semi-global alignment probably fails if signal is local wrt. input sequences
  $\Rightarrow$ use LocARNA local alignment or LocARNA-P reliabilities



LocARNA-P reliabilities

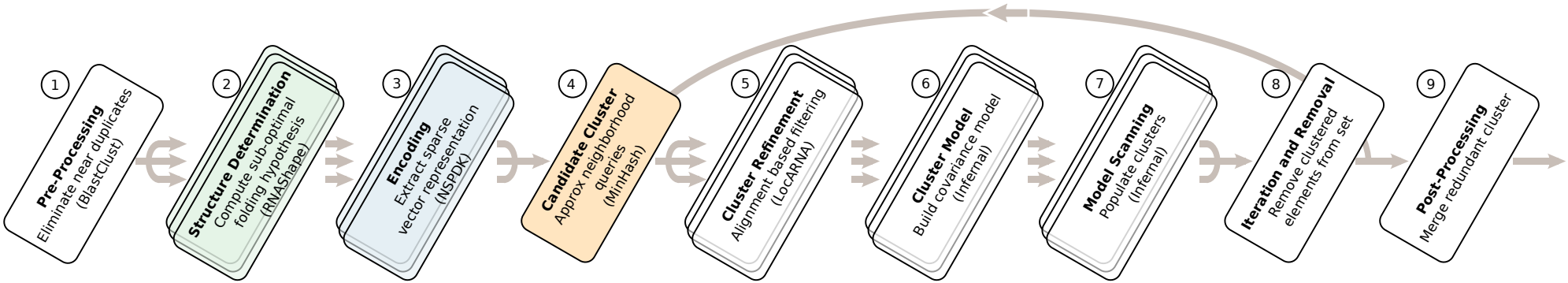
1. Align all vs. all by local/global alignment
2. build WPGMA tree
3. align subtrees with LocARNA-P
4. check for region with high reliability signal in alignment & extract region

Class	c0	c0	c0	c20	c20	c16	**c4**	**c4**	c5	**c4**	c7	c8	**c4**	**c4**	**c4**
c0	0	0	0	0	0	0	0	20	0	20	0	0	0	0	19
c0	0	0	18	18	0	0	20	0	0	19	0	0	0	0	0
c0	18	18	0	18	0	0	0	0	18	21	0	0	19	0	0
c20	0	0	0	0	0	22	0	22	0	0	0	0	0	22	23
c20	18	0	0	20	0	0	0	18	0	0	0	18	0	18	0
c16	0	0	0	22	0	0	18	22	0	0	0	0	0	0	0
c4	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**35**	**0**	**30**	**0**	**0**	**33**	**31**	**35**
c4	**0**	**0**	**0**	**0**	**0**	**0**	**35**	**0**	**0**	**29**	**0**	**0**	**28**	**31**	**42**
c5	0	0	18	19	0	0	18	0	0	18	0	0	0	0	21
c4	**0**	**0**	**0**	**0**	**0**	**0**	**30**	**29**	**0**	**0**	**0**	**0**	**37**	**32**	**33**
c7	0	0	0	0	0	0	21	0	0	0	0	19	22	0	0
c8	0	0	0	0	0	0	20	0	0	21	19	0	23	0	0
c4	**0**	**0**	**0**	**0**	**0**	**0**	**33**	**28**	**0**	**37**	**0**	**0**	**0**	**30**	**32**
c4	**0**	**0**	**0**	**0**	**0**	**0**	**31**	**31**	**0**	**32**	**0**	**0**	**30**	**0**	**33**
c4	**0**	**0**	**0**	**0**	**0**	**0**	**35**	**42**	**0**	**33**	**0**	**0**	**32**	**33**	**0**



LocARNA masked score matrix and WPGMA cluster tree with rel. profile

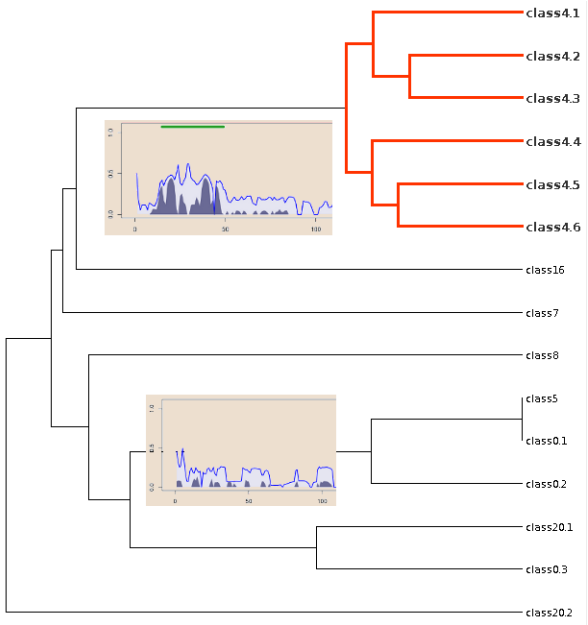
The Pipiline

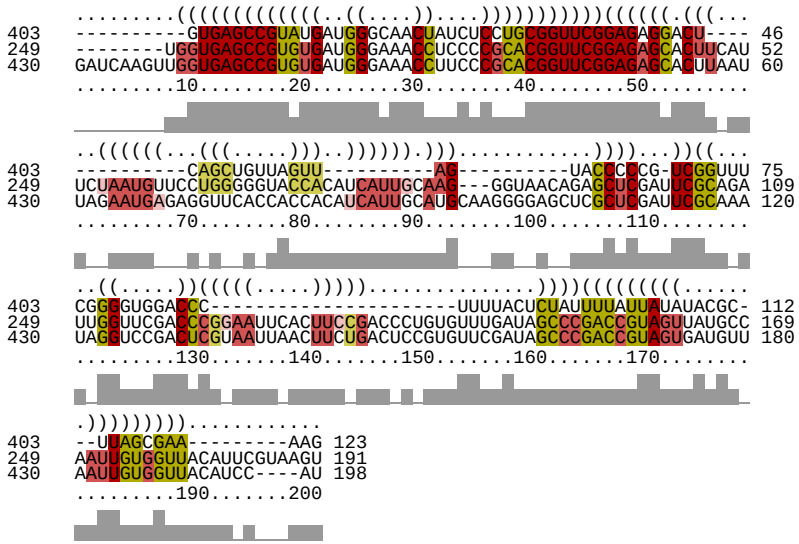


Complete clustering pipeline diagram. Phases that are executed in parallel are represented in stacked boxes. 1) filter near-duplicates, 2) compute sub-optimal structures, 3) compute sparse vector encoding, 4) compute global feature index and return top dense sets, 5) refine clusters with structural alignment procedure, 6) build covariance model with remaining high quality instances, 7) populate each cluster with retrieved instances, 8) remove clustered instances and iterate from step 4, 9) merge redundant clusters.

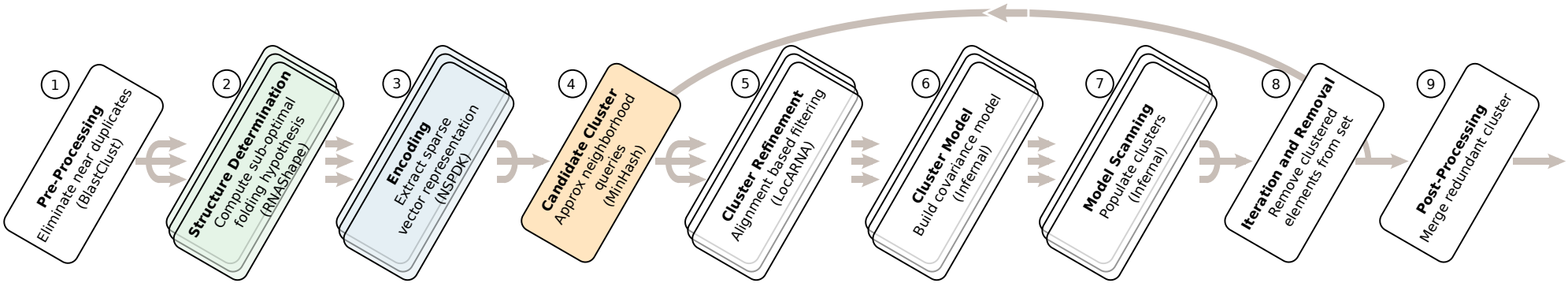
CM-Models and Scanning

1. Subtree alignment (full or high reliability region) is used as input alignment for CMbuild to get covariance model
2. (CM-model is calibrated)
3. Scan with CM-model against database (CMsearch) of input fragments





The Pipiline



Complete clustering pipeline diagram. Phases that are executed in parallel are represented in stacked boxes. 1) filter near-duplicates, 2) compute sub-optimal structures, 3) compute sparse vector encoding, 4) compute global feature index and return top dense sets, 5) refine clusters with structural alignment procedure, 6) build covariance model with remaining high quality instances, 7) populate each cluster with retrieved instances, 8) remove clustered instances and iterate from step 4, 9) merge redundant clusters.

Benchmarking data

- (1) ncRNAs from Rfam, max. 400nt, 3901 seqs, 502 families, 124 have more than 5 members (Will et al., 2007)
- (2) small ncRNAs from bacteria, max 400nt, 941 seqs, 49 families, 41 have more than 3 members, 50nt random context (NCBI)
- 15 iterations, 10 candidate clusters, 15 knn
- Comparision: LocARNA + RNAsoup, F-measure, Rand Index

Benchmark Results

			Quality (MERGED)		Time in secs		
i	#Seq	#C	F	Rand	Phase 4	Time_i	Time_{ALL}
Rfam benchmark							
0						8314	8314
1	271	5	0.882	0.888	458	14995	23309
2	629	14	0.834	0.932	416	19962	43272
3	1076	23	0.868	0.956	334	15108	58380
7	2181	58	0.877	0.985	154	11964	104940
15	2821	130	**0.834**	**0.984**	77	2491	**129626**
Small ncRNA benchmark							
0						720	720
1	140	10	0.942	0.945	42	2434	3154
2	232	20	0.926	0.939	27	3395	6549
3	270	26	0.936	0.935	17	7681	14230
7	329	35	0.890	0.897	5	250	23186
15	360	43	**0.858**	**0.866**	1	92	**24301**

TABLE: **Results for Rfam and small ncRNA benchmark set.** Results for each iteration i on the MERGED partition. Clustering quality is given as F measure and Rand index. The total number of clustered sequences is indicated with #Seq. The total number of clusters after merging is given by #C. Time_i denotes the total time for iteration i, Time_{ALL} is the total serial time up to iteration i.

Comparison Locarna+RNAsoup:

(1) Rfam: F=0.588, R= 0.586, $k_{RNAsoup} = 0.8$; time = 370d

(2) small ncRNAs: F=0.729, R=0.88, $k_{RNAsoup} = 0.4$, time = 7d

Large scale datasets

- Drosophila genome: 16k RNAz hits (Rose et al, 2007)
- Takifugu: 11k RNAz hits (Rose et al, 2008)
- human EvoFold: 37k (Pedersen et al, 2006)
- human lincRNAs: 8k (Cabili et al, 2011)
- Zebra fish lincRNAs: 1k (Pauli et al, 2011)
- human 3'UTR: 36k (Refseq)
- split into fragments of 150-200nt, 10 iterations, 10 candidate clusters

Application Results

Species Reference	Type	Method	Input	Size [Mb]	Time*	Cluster	MPI_{avg}	$SCI_{>0.5}$
benchmark								
Bacteria	small ncRNAs	misc	363	0.06	6.8h	39	0.75	29
Human	predicted	EvoFam	699	0.03	0.3h	37	0.52	36
Misc	small ncRNAs	Rfam	3,900	0.51	36h	130	0.64	98
de-novo discovery								
Fugu	lincRNAs	RNA-seq	5,877	0.09	10.3h	99	0.39	16
Fugu	predicted	RNAz	11,287	1.36	13.3h	97	0.39	22
Fruit fly	predicted	RNAz	17,765	2.15	20.4h	95	0.34	23
Human	lincRNAs	RNA-seq	31,418	5.40	3.6d	95	0.34	3
Human	predicted	EvoFold	37,258	1.37	5.7d	117	0.75	109
Human	3'UTRs	RefSeq	118,514	21.91	12.8d	106	0.34	13
$\sum$			227,081	32.88	25.7d	815	-	349

Results

- recheck found clusters against Rfam: tRNAs, miRNAs, snRNAs for RNAz, UTR elements in EvoFam, snoRNA in 3'UTR
- human lincRNAs: 55% of found clusters expressed in same tissue (testes, kidney, brain)
- human lincRNAs: 50% of found clusters show enriched GO terms, many with "neuron..."
- Fugu lincRNAs: 5 times more structural cluster with SCI>0.5 (genome duplication!)

Issues

- splitting long sequences into fragments can disturb strutural motifs
- many tools, many parameters, many things to tune
- pipeline adapted to (our) SGE
- CMcalibrate currently not used (needs hours per model), just bitscore cutoff (15-20)
- Can LocARNA-scan replace the Infernal part!?
- Merging of clusters via cmcompare?
- ...

Good news

...but it is the first time that we can handle genome wide RNA datasets in a reasonable time!!!

Thanks to

Fabrizio
Dominic
Sita
Rolf

and the whole Bioinformatics Group in Freiburg

This work is submitted to ISMB 2012

		BEST			Partititon MERGED						ORACLE			SOFT		
Iteration	#Seq	#C	F	Rand	#C	F	Rand	Accuracy	Precision	Recall	#C	F	Rand	#C	F	Rand
1	271	10	0.545	0.376	5	0.882	0.888	0.998	0.912	0.869	5	0.882	0.888	10	0.938	0.215
2	629	20	0.649	0.699	14	0.834	0.932	0.997	0.891	0.814	13	0.877	0.932	20	0.888	0.372
3	1076	30	0.743	0.864	23	0.868	0.956	0.997	0.928	0.841	22	0.894	0.957	30	0.879	0.429
4	1737	40	0.778	0.956	33	0.872	0.984	0.996	0.950	0.834	31	0.908	0.985	40	0.878	0.612
5	1844	50	0.774	0.943	42	0.860	0.984	0.997	0.924	0.839	39	0.906	0.985	50	0.862	0.609
6	2019	60	0.783	0.780	50	0.879	0.985	0.998	0.927	0.867	47	0.918	0.986	60	0.867	0.549
7	2181	69	0.786	0.783	58	0.877	0.985	0.998	0.922	0.875	55	0.910	0.986	69	0.865	0.554
8	2321	79	0.787	0.781	67	0.873	0.985	0.998	0.911	0.883	62	0.912	0.986	79	0.853	0.558
9	2391	89	0.782	0.782	77	0.856	0.985	0.998	0.892	0.872	70	0.908	0.986	89	0.838	0.559
10	2440	99	0.773	0.781	86	0.845	0.985	0.998	0.881	0.868	77	0.904	0.983	99	0.824	0.555
11	2503	109	0.767	0.779	94	0.846	0.984	0.998	0.878	0.872	84	0.907	0.983	109	0.821	0.555
12	2587	118	0.765	0.659	102	0.843	0.985	0.999	0.876	0.869	91	0.907	0.984	118	0.815	0.481
13	2671	128	0.768	0.661	112	0.839	0.985	0.999	0.868	0.869	100	0.902	0.984	128	0.807	0.482
14	2742	138	0.771	0.661	122	0.837	0.984	0.999	0.868	0.867	109	0.900	0.984	138	0.801	0.482
15	2821	148	0.766	0.649	130	0.834	0.984	0.999	0.856	0.876	117	0.892	0.983	148	0.796	0.483

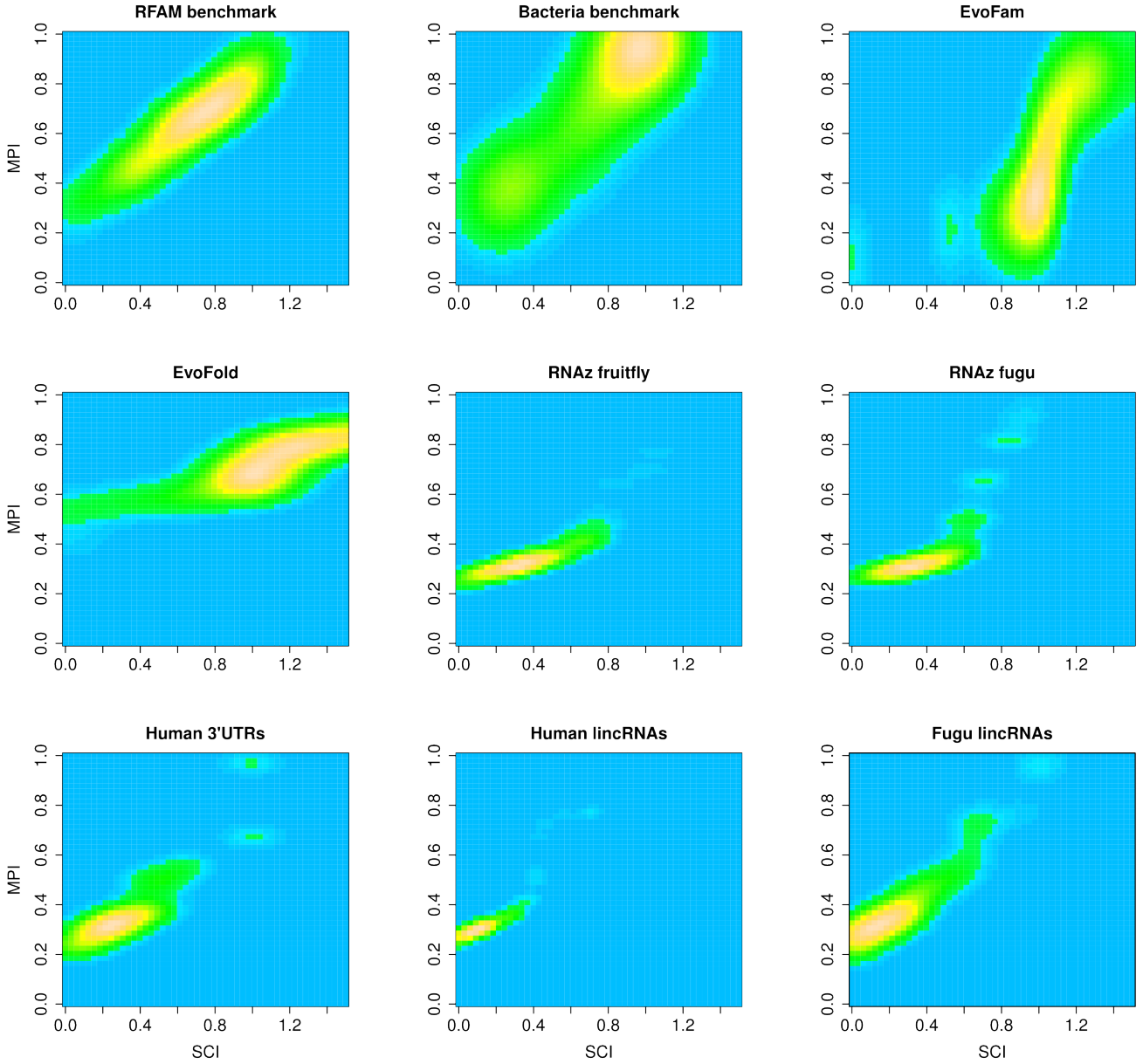
Rfam benchmark set

		Partition														
		BEST			MERGED						ORACLE			SOFT		
Iteration	#Seq	#C	F	Rand	#C	F	Rand	Accuracy	Precision	Recall	#C	F	Rand	#C	F	Rand
1	140	10	0.942	0.945	10	0.942	0.945	0.996	0.924	0.974	10	0.942	0.945	10	0.942	0.897
2	232	20	0.926	0.939	20	0.926	0.939	0.996	0.903	0.971	20	0.926	0.939	20	0.916	0.892
3	270	26	0.936	0.935	26	0.936	0.935	0.996	0.920	0.970	26	0.936	0.935	26	0.928	0.894
4	298	30	0.925	0.922	30	0.925	0.922	0.996	0.902	0.971	29	0.929	0.906	30	0.907	0.874
5	305	32	0.909	0.918	32	0.909	0.918	0.996	0.880	0.973	30	0.925	0.900	32	0.892	0.872
6	319	34	0.897	0.904	34	0.897	0.904	0.996	0.861	0.974	32	0.911	0.887	34	0.881	0.862
7	329	35	0.890	0.897	35	0.890	0.897	0.995	0.851	0.975	33	0.903	0.881	35	0.875	0.857
8	332	36	0.883	0.898	36	0.883	0.898	0.995	0.844	0.966	34	0.895	0.882	36	0.866	0.856
9	335	37	0.882	0.901	37	0.882	0.901	0.995	0.840	0.967	35	0.894	0.884	37	0.863	0.854
10	339	38	0.867	0.891	38	0.867	0.891	0.995	0.831	0.948	36	0.878	0.875	38	0.852	0.846
11	345	39	0.871	0.899	39	0.871	0.899	0.995	0.839	0.946	37	0.881	0.888	39	0.848	0.828
12	349	40	0.868	0.900	39	0.873	0.894	0.995	0.848	0.940	38	0.876	0.889	40	0.839	0.815
13	353	41	0.868	0.902	39	0.871	0.891	0.994	0.848	0.934	38	0.892	0.893	41	0.837	0.794
14	357	42	0.854	0.809	39	0.872	0.886	0.995	0.848	0.936	38	0.893	0.888	42	0.836	0.646
15	360	43	0.843	0.794	39	0.858	0.866	0.993	0.841	0.916	38	0.893	0.874	43	0.827	0.636

small ncRNA benchmark set

i	#C	Phase 2	Phase 3		Time$_i$	Time$_{ALL}$	Time$_C$
0		4169	4145		8314	8314	
		Phase 4	Phase 5-6	Phase 7			
1	10	458	10633	3904	14995	23309	2331
2	20	416	17980	1564	19962	43272	2163
3	30	334	13239	1533	15108	58380	1946
4	40	260	11694	752	12708	71088	1777
5	50	186	11351	783	12321	83409	1668
6	60	171	8667	726	9566	92975	1549
7	70	154	11069	739	11964	104940	1499
8	80	133	3671	597	4402	109342	1366
9	90	120	3841	620	4581	113924	1265
10	100	114	2768	707	3590	117515	1175
11	110	108	2544	492	3145	120660	1096
12	120	99	1917	712	2728	123388	1028
13	130	90	1197	640	1929	125318	964
14	140	83	1220	512	1817	127135	908
15	150	77	1776	636	2491	**129626**	864

Rfam benchmark set times



SCI/MPI for application data