Pattern Discovery in Biosequences

SDM 2005 tutorial

Stefano Lonardi

University of California, Riverside

Latest version of the slides at http://www.cs.ucr.edu/~stelo/slides/

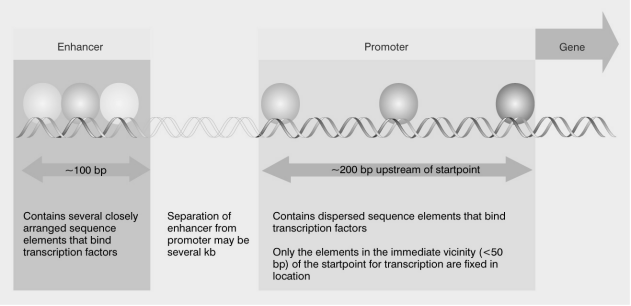
Roadmap

- Intro
- Basic concepts
- Classification of patterns
- Complexity results
- Efficient algorithms for pattern discovery
  - Deterministic patterns: Enumerative
  - Rigid patterns: Enumerative: Teiresias, Weeder, Tiling
    Sampling: Winnower, Projection
- Appendix

Intro

Discovery of regulatory elements

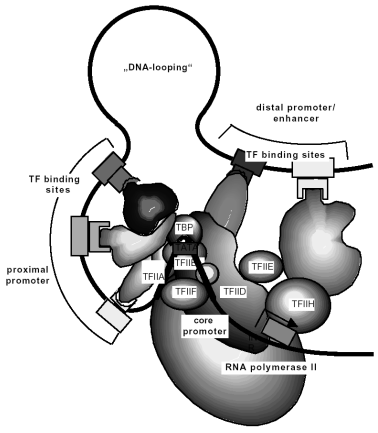
- *Promoter:* a region of DNA involved in binding of RNA polymerase to initiate transcription
- *Enhancer:* a region of DNA that increases the utilization of (some) promoters (it can function in either orientation and any location relative to the promoter)
- *Repressor:* a region of DNA that decreases the utilization of (some) promoters



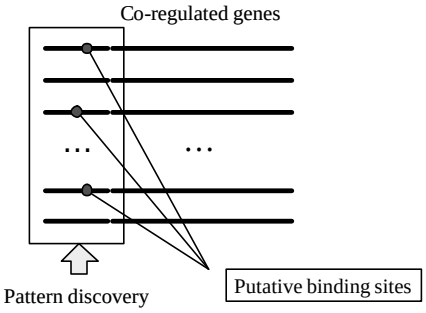
Source: Lewin, genes VII

Transcription

- Different factors are involved in the transcription machinery
  - presence of transcription factors and their binding sites
  - ability of DNA to bend
  - relative location of the binding sites
  - presence of CpG islands (“p” is for phosphate)
  - …



Transcription factors binding sites



Basic concepts

Some notations

Σ: alphabet

$a, b, c, \ldots$: symbols from Σ

x: sequence/string over Σ, $|x| = n$

$\{x_1, x_2, \ldots, x_k\}$: multi-sequence, $\sum_{i=1}^{k} |x_i| = n$

y (or w): substring of x, $|y| = m$

y^i : the substring $\underbrace{yy\cdots y}_{i \text{ times}}$ $(i \geq 0)$

Some notations

$y_{[i]}$:the i-th symbol of y $(1 \leq i \leq m)$

$y_{[i,j]}$:the string $y_{[i]}y_{[i+1]}\cdots y_{[j]} (1 \leq i \leq j \leq m)$

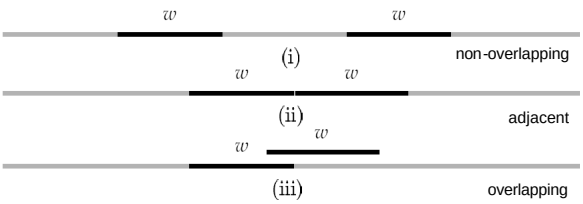
$y_{[1,j]}$:are the prefixes of y $(1 \leq j \leq m)$

$y_{[j,m]}$:are the suffixes of y $(1 \leq j \leq m)$

$f(y)$:number of occurrences of y

sometimes called the support of y

Occurrences: types



For our purposes, any of the above is simply an occurrence
Keep in mind that in some cases you may have to distinguish them

Example (DNA)

x_1 = **CCACCCTTTTGTGGGGCTTCTATTTCAAGG**

x_2 = **TTGTTCTTCCTGCATGTTGCGCGCAGTGCG**

x_3 = **TTCTAAAAGGGGCATTATCAGAAAAAGAAG**

x_4 = **GTGTAAAATTGTGTGCTACCTACCGTATTA**

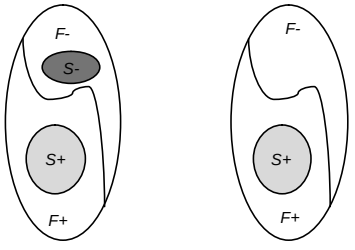
- Σ = {**A,C,G,T**} $|\Sigma| = 4$ $n = 120$
- e.g., y = **AAAA** is a substring of x_3 and x_4
  - $f(y) = 4$ (occurrences can overlap)

"Pattern Discovery": the problem

Pattern discovery: the problem

- Given a set of sequences S^+ and a model of the source for S^+
- Find a set of patterns in S^+ which have a support that is "statistically significant" with respect to the probabilistic model
- If we are also given negative examples S^-, we must ensure that the patterns do not appear in S^-

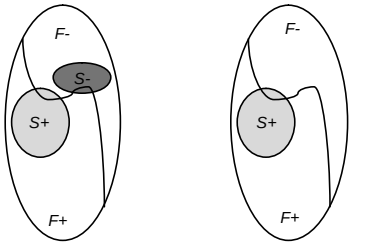
Pattern discovery problem



Pos & Neg examples

Only Pos examples

Noisy data



Pos & Neg examples Only Pos examples

Pattern discovery "dimensions"

- Type of learning
  - from positive examples only (unsupervised)
  - from both positive and negative examples (supervised)
  - noisy data
- Type of patterns
  - deterministic, rigid, flexible, profiles, …
- Measure of statistical significance
- *A priori* knowledge

A classification of patterns

Types of patterns

- Deterministic patterns
- Rigid patterns
  - Hamming distance
- Flexible patterns
  - Edit distance
- Matrix profiles

✓ A *motif* is any of these patterns, as long as it is associated with statistical/biological significance

Deterministic Patterns

- Definition: *Deterministic patterns* are strings over the alphabet Σ
  - e.g., "**TATAAA**" (TATA-box consensus)
- Discovery algorithms are faster on these types of patterns
- Usually not flexible enough for the needs of molecular biology

Rigid patterns

- Definition: Rigid patterns are patterns which allow substitutions/"don't care" symbols
  - e.g., the patterns under IUPAC alphabet {**A,C,G,T,U,M,R,W,S,Y,K,V,H,D,B,X,N**} where for example **R**=[**A**|**G**], **Y**=[**C**|**T**], etc.
  - e.g, "**ARNNTTYGA**" under IUPAC means "**A**[**A**|**G**][**A**|**C**|**G**|**T**][**A**|**C**|**G**|**T**]**TT**[**C**|**T**]**GA**"
- Note that the size of the pattern is not allowed to change

Hamming distance

- Definition: Given two strings *y* and *w* such that $|y|=|w|$, the Hamming distance *h(w,y)* is given by the number of mismatches between *y* and *w*
- Example:
  y=**GATTACA**
  w=**TATAATA**
  h(w,y)=h(y,w)=3

Hamming neighborhood

- Definition: Given a string *y*, all strings at Hamming distance at most *d* from *y* are in its *d-neighborhood*
- Fact: The size *N(m,d)* of the *d*-neighborhood of a string *y*, $|y|=m$, is

$$N(m,d)=\sum_{j=0}^{d}\binom{m}{j}\left(|\Sigma|-1\right)^{j}\in O\left(m^{d}\,|\Sigma|^{d}\right)$$

Hamming neighborhood

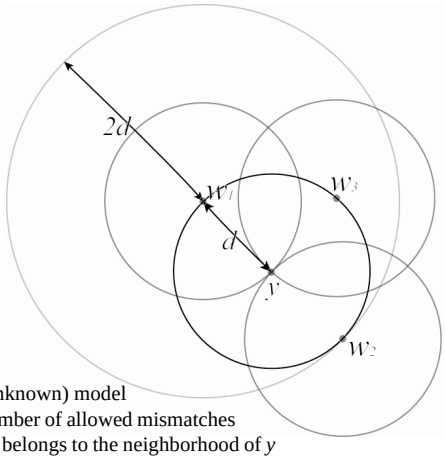
- Example:
  y = **ATA** the 1-neighborhood is
  {CTA,GTA,TTA,
  AAA,ACA,AGA,
  ATC,ATG,ATT,
  ATA}

- This set can be written as a rigid pattern
  {NTA|ANA|ATN}

Models

- We may be able to observe occurrences of the neighbors of y, but we may never observe an occurrence of y

- Definition: The center of the d- neighborhood y is also called the *model*



y is the (unknown) model
d is the number of allowed mismatches
w_1, w_2, w_3 belongs to the neighborhood of y

Hamming neighborhood

- Fact: Given two strings w_1 and w_2 in the d-neighborhood of the model y, then $h(w_1,w_2) \leqslant 2d$
- The problem of finding y given $w_1, w_2, \ldots$ is also called the *Steiner sequence* problem
- Unfortunately, even if we were able to determine exactly all the w_i in the neighborhood, there is no guarantee to find the unknown model y

Example

- Suppose $m=4$, $d=1$ and that we found occurrences of **{AAAA,TATA,CACA}**
- The pairwise Hamming distance is 2 but there is no string at Hamming distance 1 to each of these

Word match filtering

- Fact: Given two strings w_1 and w_2 in the d-neighborhood of the model y, they both contain an occurrence of a word of length at least $\lfloor m/(2d+1) \rfloor$
- Example: y = **GATTACA**
  w_1 = **GATTTCA**
  w_2 = **GGTTACA**
  TT and **CA** are occurring exactly. In fact $\lfloor m/(2d+1) \rfloor = \lfloor 7/3 \rfloor = 2$

Flexible patterns

- Definition: *Flexible patterns* are patterns which allow substitutions/"don't care" symbols and variable-length gaps
  - e.g., Prosite **F-x(5)-G-x(2,4)-G-*-H**
- Note that the length of these pattern is variable
- Very expressive
- Space of all patterns is huge

Edit distance

- Definition: the *edit distance* between two strings y and w is defined as the minimum number of edit operations - insertions, deletions and substitutions - necessary to transform y into w (matches do not count)
- Definition: a sequence of edit operations to transform y into w is called an *edit script*

Edit distance

- The *edit distance problem* is to compute the edit distance between *y* and *w*, along with an optimal edit script that describes the transformation
- An alternative representation of the edit script is the *alignment*

Example

- Given *w* = **GATTACA**
  y = **TATATA**

  GATTACA? ATTACA? TTACA
  ? TATACA? TATATA (1 ins, 2 del, 1 sub)

  GATTACA? TATTACA
  ? TATACA? TATATA (0 ins, 1 del, 2 sub)
- Edit distance is 3

Corresponding global alignment

- Given *w* = **GATTACA**
  y = **TATATA**
- We can produce the following *alignments*

  GAT-TAC-A G-ATTAC-A

  --TATA-TA -TAT-A-TA

  where "**–**" represents a *space* (we cannot have "**–**" aligned with "**–**")

Profiles

- *Position weight matrices*, or *profiles*, are $|\Sigma| \times m$ matrices containing real numbers in the interval *[0,1]*
  – e.g.

A	*0.26*	*0.22*	*0.00*	*1.00*	*0.11*
C	*0.17*	*0.18*	*0.59*	*0.00*	*0.35*
G	*0.09*	*0.15*	*0.00*	*0.00*	*0.00*
T	*0.48*	*0.45*	*0.41*	*0.00*	*0.54*

  – *consensus*

Distance for profiles

The relative entropy $H(p||q)$ between two discrete probability distributions $p = \{p_1, \ldots, p_k\}$ and $q = \{q_1, \ldots, q_k\}$ is defined by

$$H(p||q) = \sum_{i=1}^{k} p_i \log \frac{p_i}{q_i}$$

Also called *cross-entropy* or *Kullback-Liebler* distance. It is easy to verify that $H(p,q) \geq 0$ with equality iff $p = q$.

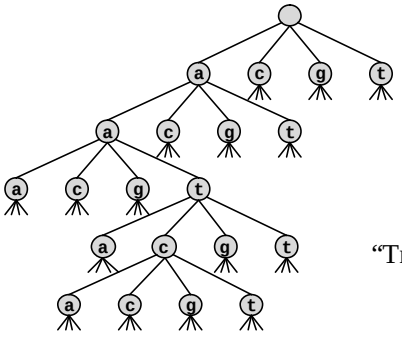
Discovering Deterministic Patterns

The problem

- Input: a string x of length n, a support q
- Output: all substrings y occurring at least q times in x

- There are $O(n^2)$ substrings
- Can we find the frequent substrings faster?

Enumerating the $O(n^2)$ patterns



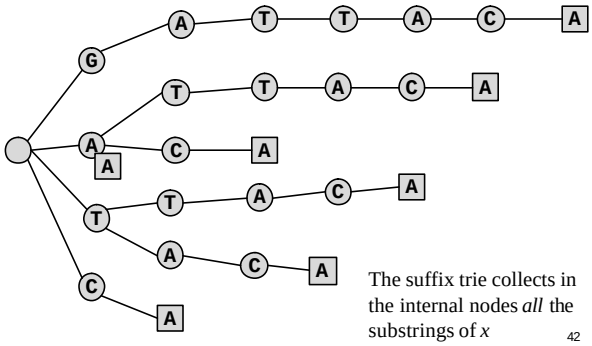
"Trie"

Suffix trie

- We build a trie with all the suffixes of the text *x*
- Example: if *x* = **GATTACA** we use

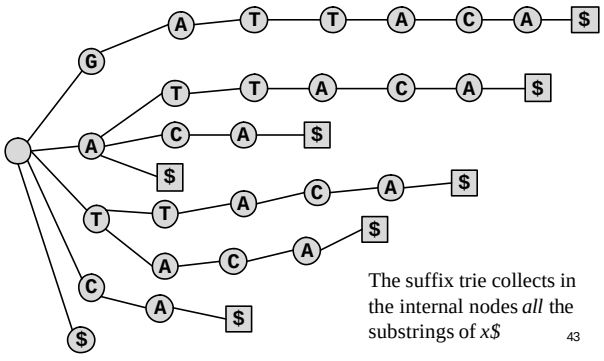
GATTACA
ATTACA
TTACA
TACA
ACA
CA
A

Suffix trie for "**GATTACA**"



The suffix trie collects in the internal nodes *all* the substrings of *x*

Suffix trie for "**GATTACA$**"



The suffix trie collects in the internal nodes *all* the substrings of *x$*

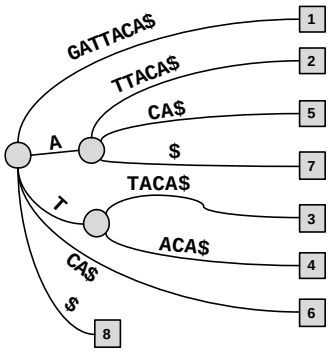
Suffix trie

- Construction $O(n^2)$
- Space $O(n^2)$
- Query $O(m)$

- We can do better by removing unary nodes from the tree, and coalescing the edges
- The result is called *suffix tree*

Suffix tree for "GATTACA$"

1 2 3 4 5 6 7 8



The suffix tree collects in the *implicit* internal nodes *all* the substrings of *x$*

The *locus* of a string is the node in the tree corresponding to it

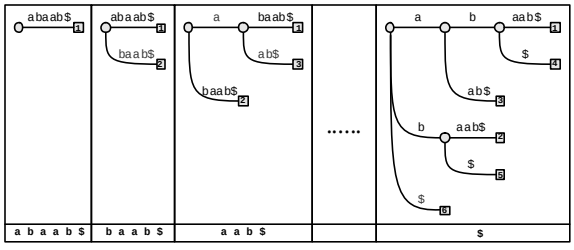
The label in the leaves identifies the suffix position (used to find pos all occs)

The number of leaves in the subtree corresponds to the number of occurrences

Space analysis

- Every node is branching
- The number of leaves is *n*
- Therefore the overall number of nodes is at most *2n-1*
- Use two integers (constant space) to identify labels on the arcs
- Therefore the overall size of the tree is *n*

Brute force construction



Worst case $O(n^2)$

1 2 3 4 5 6
a b a a b $

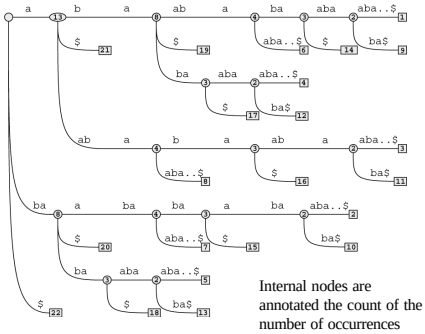
Average case $O(n \log n)$

Computing number of occurrences

ANNOTATE_$f(w)$ (**suffix_tree** T)
  for each leaf u of T **do**
    let $f(L(u)) = 1$
  visit T in depth-first traversal, for each internal node u **do**
    let $f(L(u))$ equal to the sum of $f(\cdot)$ of the children of u

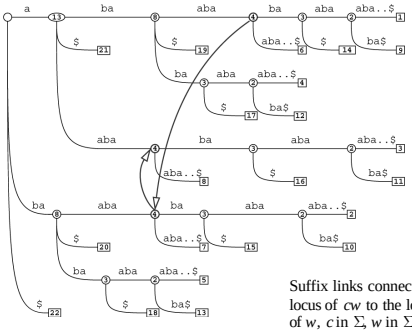
Time complexity is *O(n)*

Suffix tree for "**abaababaabaabababaaba$**"



Internal nodes are annotated the count of the number of occurrences

Suffix links

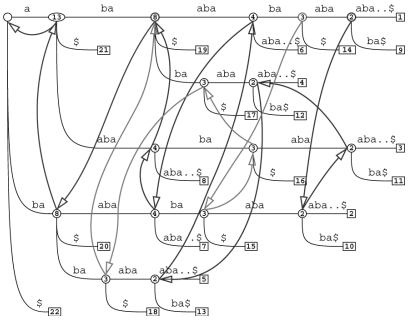


Suffix links connect the locus of *cw* to the locus of *w*, *c* in Σ, *w* in Σ*

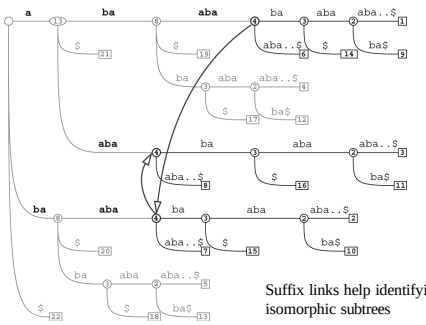
Suffix trees

- Assume constant size finite alphabet
- Suffix trees can be built in *O(n)* time and space [Weiner 1973, McCreight 1976, Ukkonen 1995, Farach 1997]
- Number of occurrences can be computed in *O(n)* time
- Observe that several subtrees of the suffix tree are isomorphic
- Idea: merge isomorphic trees to save space

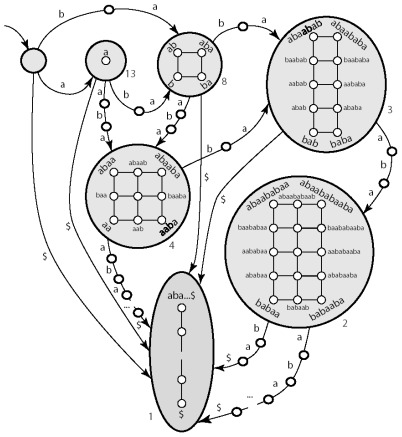
All the suffix links (except leaves)



Suffix links



Suffix links help identifying isomorphic subtrees



Remarks

- Frequent substrings can be found in *O(n)* time and space
- Pros:
  - exhaustive
  - linear time and space
- Cons:
  - limited to deterministic patterns

Discovering Rigid Patterns

Complexity results

- Li *et al.*, [Li 1999] proved several important theoretical facts
- Many of the problems in pattern discovery turn out to be NP-hard
- For some there is a polynomial time approximation scheme (PTAS)

Consensus Patterns

- Consensus patterns problem: Given a multisequence $\{x_1,x_2,\ldots,x_k\}$ each of length n and an integer m, FIND a string y of length m and substring t_i of length m from each x_i such that $\Sigma_i\, h(y,t_i)$ is minimized
- Theorem [Li *et al.*, 1999]: The consensus pattern problem is NP-hard

Closest string

- Closest string problem: given a multisequence $\{x_1,x_2,\ldots,x_k\}$ each of length n, FIND a string y of length n and the minimum d such that $h(y,x_i)\leqslant d$, for all i
- Theorem: The closest string problem is NP-hard

Closest substring

- Closest substring problem: given a multisequence $\{x_1,x_2,\ldots,x_k\}$ each of length n and an integer m, FIND a median string y of length m and the minimum d such that for each i there is a substring t_i of x_i of length m satisfying $h(y,t_i) \leqslant d$
- Theorem: The closest substring problem is NP-hard (it is an harder version of Closest string)

NP-hard: what to do?

- Change the problem
  - e.g., “relax” the class of patterns
- Accept the fact that the method may fail to find the optimal patterns
  - Heuristics
  - Randomized algorithms
  - Approximation schemes

Discovering Rigid Patterns

- We report on five recent algorithms
- Teiresias [1998]
- Winnower [2000]
- Projection [2001]
- Weeder [2001]
- Tiling motifs [2003]
- *(disclaimer: my selection is biased)*

Planted *(m,d)*-motif problem

- Proposed by Pevzner *et al*.
- Randomly generate $k=20$ sequences of $n=1,000$ symbols over the DNA alphabet
- Randomly generate a pattern y of length m
- Generate an instance of y by changing d symbols at random
- Inject one instance of y at a random position in each sequence

Planted *(m,d)*-motif problem

- The problem is to determine the *unknown* pattern y of length m in a set of $k=20$ nucleotide sequences each of length $n=1,000$, and each one containing exactly one occurrence of a string w such that $h(y,w)=d$

Teiresias

Teiresias algorithm

- By Rigoustos and Floratos [Rigoustos 1998]
- The worst case running time is exponential, but works reasonably fast on average

Teiresias patterns

- Teiresias searches for rigid patterns on the alphabet $\Sigma \cup \{.\}$ where "." is the don't care symbol
- Symbols from Σ are called "solid"
- In Teiresias, there are some constrains on the density of "." that can appear in a pattern

$<L,W>$ patterns

- Definition: Given integers L and W, $L \leqslant W$, y is a $<L,W>$ pattern if
  - y is a string over $\Sigma \cup \{.\}$
  - y starts and ends with a symbol from Σ
  - any substring of y containing exactly L solid symbols has to be shorter (or equal) to W

    [that is, any substring of length L contains at most W-L don't cares]

Example of <3,5> patterns

- **AT..CG..T** is a <3,5> pattern
- **AT..CG.T.** is not a <3,5> pattern, because it ends with "."
- **AT.C.G..T** is not a <3,5> pattern, because the substring **C.G..T** is 6 characters long [contains more than 5-3=2 don't cares]

Teiresias

- Definition: A pattern *w* is more *specific* than a pattern *y*, if *w* can be obtained from *y* by changing one or more "." to symbols from Σ, or by appending any sequence of $\Sigma \cup \{.\}$ to the left or to the right of *y*
- Example: given *y* = **AT.CG.T**, the following patterns are *more specific* then *y* **ATCCG.T, CAT.CGCT, AT.CG.T.A, T.AT.CGTT.A**

Teiresias

- Definition: A pattern *y* is *maximal* with respect to the sequences $\{x_1,x_2,\ldots,x_k\}$ if there exists no pattern *w* which is more specific than *y* and *f(w)=f(y)*
- Given $\{x_1,x_2,\ldots,x_k\}$ and parameters *L,W,K*, Teiresias reports *all* the *maximal* <*L,W*> patterns that have support at least *K*

Teiresias algorithm

- Idea: if *y* is a <*L,W*> pattern with support at least *K*, then its substrings are also <*L,W*> patterns with support at least *K*
- Therefore, Teiresias assembles the maximal patterns from smaller patterns
- Definition: A pattern *y* is *elementary* if is a <*L,W*> pattern containing exactly *L* symbols from Σ

Teiresias algorithm

- Teiresias works in two phases
  - Scanning: find all elementary patterns with support at least *K*; these become the initial set of patterns
  - Convolution: repeatedly extend the patterns by "gluing" them together
- Example: *y* = **AT..CG.T** and *w* = **G.T.A** can be merged to obtain **AT..CG.T.A**

Convolution phase

- For each elementary pattern *y*, try to extend it with all the other elementary patterns
- Any pattern that cannot be extended without losing support can be potentially maximal

Convolution phase

- To speed-up this phase, one wants to avoid the all-against-all comparison
- The authors devise two partial orderings $<_{pf}$ and $<_{sf}$ on the universe of patterns
- Using these orderings to schedule the convolution phase, they guarantee that
  - all patterns are generated
  - a maximal pattern *y* is generated before any non-maximal pattern subsumed by *y*

Partial ordering $<_{pf}$

- Definition: determine whether $y <_{pf} w$ or $w <_{pf} y$ using the following algorithm
  - align *y* and *w* such that the leftmost residues are in the same column
  - examine one column after the other (left to right) and stop whenever one column has a residue and the other has a "."
  - if the residue comes from *y* then $y <_{pf} w$
  - if the residue comes from *w* then $w <_{pf} y$

Example

- y = **ASD...F**
  w = **SE.ERF.DG**
  $y <_{pf} w$

- y = **ASD...F**
  w = **SE.ERF.DG**
  $w <_{sf} y$

Teiresias algorithm

- Initialize the stack with elementary patterns with support at least K
- Order the stack according to $<_{pf}$ and $<_{sf}$
- Repeat
  - Repeat
    - Try to extend the top pattern to the right with all the others in the prefix-wise ordering
    - If a new pattern is formed with have enough support, it becomes the new top
  - Until the top can no longer be extended to the right
  - Do the same for left extension, using the ordering $<_{sf}$
  - Check the top for maximality, if so pop it and report it
- Until stack is empty

Remarks on Teiresias

- It can be proved that Teiresias correctly reports all $<L,W>$ maximal patterns
- Pros:
  - provably correct
  - fast on average input
- Cons:
  - exponential time complexity
  - limited to $<L,W>$ patterns

Winnower

Pevzner and Sze, UCSD

Winnower

- Invented by Pevzner and Sze [Pevzner 2000]
- Initially designed to solve the *(15,4)*-motif challenge

Winnower

- Pevzner and Sze show that the most popular algorithms (Consensus, GibbsDNA, MEME) fail to solve (most of the times) the *(15,4)*-motif problem *[n=600, k=20]*
- (Note: this comparison is not totally *fair)*
- Why the *(15,4)*-motif problem is difficult?
- Because two strings in the class of the *(15,4)* unknown pattern may differ by as many as 8 positions out of 15, a rather large number

Winnower

- Idea: Search for a set of strings of length *m* such that any two in a set differ at most by *2d* positions
- Remember however that this may not be sufficient

Winnower

- How to find groups of patterns such that given any two elements w_1 and w_2 in the group, $h(w_1,w_2) \leqslant 2d$?
- One could generate *(k choose 2)* multiple alignments to find out all pairs of substrings of length *m* that have at most *2d* mismatches (Consensus [Hertz & Stormo 1999])

Winnower

- Winnower builds a graph *G* in which
  - each vertex corresponds to a distinct string of length *m*
  - two vertices are connected by an edge if the Hamming distance between the corresponding strings is at most *2d*, and the strings do not come from the same sequence (remember that we are guaranteed that there is only one occurrence of the unknown pattern in each sequence)

Graph for the *(15,4)*-problem

- The authors report that for each "signal"-edge there are about 20,000 spurious-edges
- Finding the signal among the noise is a "daunting task"

Winnower

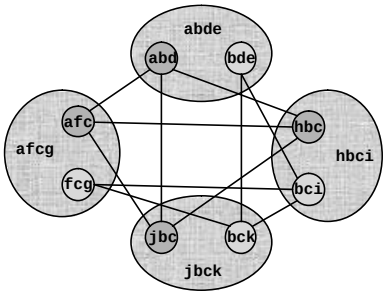
- Winnower searches the graph *G* for *cliques*, which are subsets of vertices totally connected
- But the problem of finding large cliques in graphs is *NP*-complete

Multipartite graphs

- Definition: A graph *G* is *n*-partite if its vertices can be partitioned into *n* sets, such that there is no edge between any two vertices within a set
- Fact: Winnower's graph is *k*-partite

Example

- Given sequences {**abde,afcg,hbci,jbck**} we look for a *(3,1)*-motif



Idea

- Each vertex of the clique has to be in a different partition
- We look for cliques that have exactly one vertex in each partition

Extendable cliques

- Definition: a vertex *u* is a *neighbor* of a clique $\{v_1,\ldots,v_s\}$ if $\{v_1,\ldots,v_s,u\}$ is also a clique for *G*, when $s<k$
- Definition: a clique is called *extendable* if it has at least one neighbor in every part of the *k*-partite graph *G*

Extendable cliques

- Definition: A clique with *k* vertices, each in a different partition is called *maximal*
- Consider a maximal clique and take a subset of *t* of its vertices: this subset is an extendable clique
- Idea: remove edges that do not belong to extendable cliques

Extendable cliques

Fact: For any clique of size k there are $\binom{k}{t}$ extendable cliques with t vertices

Fact: Any edge belonging to a clique with k vertices is member of at least $\binom{k-2}{t-2}$ extendable cliques of size t

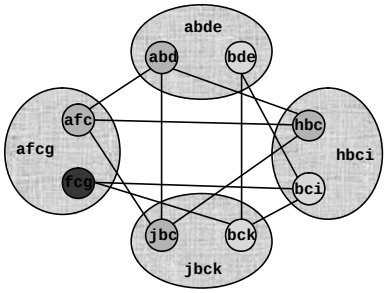
Idea

An edge that is not member of at least $\binom{k-2}{t-2}$ expandable cliques of size t cannot be part of a maximal clique and therefore it can be removed

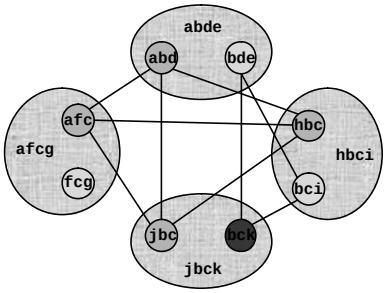
t=1

- For *t=1*, each vertex is a clique
  - it is extendable if it is connected to at least one vertex in each partition
- Delete all edges corresponding to vertices that do not have a neighbor in each partition
- Iterate

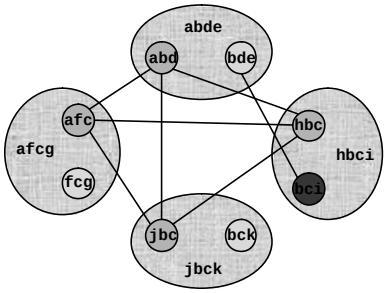
Example



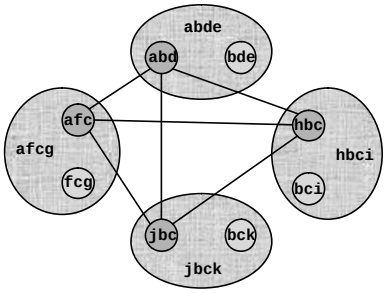
Example



Example



Example



t=2

- For *t=2*, each pair of vertices *u,v* such that there is an edge *(u,v)* is a clique
  - it is extendable if there is vertex *z* in each of the other *k-2* partitions such that *(u,v,z)* is a cycle of length 3
  - each edge should belong to at least *(k-2 choose t-2)=(n-2 choose 0)=1* clique of size 2

t>2

- For *t=3*, Winnower removes edges that belong to less than *k-2* extendable cliques of size *3*
- For *t=4*, Winnower remove edges that belong to less than *(k-2)(k-1)/2* extendable cliques of size *4*
- …

Remarks on Winnower

- Pros:
  - more effective than Meme, Consensus and GibbsDNA for the (15,4) problem
- Cons:
  - randomized
  - time-complexity can be very high (e.g., for *t=3* is $O(n^4)$)
  - need to know *m* and *d* in advance
  - assume exactly one occurrence per sequence

Projection

Random Projection algorithm

- Proposed by Buhler and Tompa [Buhler 2001]
- The algorithm was initially designed to solve the *(m,d)*-motif planted problem

Analysis on *(m,d)*-motif problem

Suppose A,C,T,G have probability 1/4. Then the probability that a pattern of size m occurs at a given position is $p_{(0)} = (1/4)^m$. If we allow up to one mismatch, the probability becomes $p_{(1)} = p_{(0)} + m(3/4)(1/4)^{m-1}$. If we allow at most two, it becomes $p_{(2)} = p_{(1)} + \frac{m(m-1)}{2}(3/4)^2(1/4)^{m-2}$. In general, if we allow up to d mismatches, $p_{(d)} = \sum_{i=0}^{d} \binom{m}{i} \left(\frac{3}{4}\right)^i \left(\frac{1}{4}\right)^{m-i}$.

Analysis on *(m,d)*-motif problem

If Z is the r.v. for the number of occurrences, then $P(Z > 0) = 1 - P(Z = 0) = 1 - (1 - p_{(d)})^{n-m+1}$

If we have k sequences, we get that the probability that a particular y occurs at least once in each sequence is $\left(1 - (1 - p_{(d)})^{n-m+1}\right)^k$.

Therefore, the expected number of patterns is $E(n,m,k,d) = 4^m \left(1 - (1 - p_{(d)})^{n-m+1}\right)^k$.

Stats of spurious *(m,d)*-motifs in simulated data *(k=20,n=600)*

m	d	$E(600,m,20,d)$	$E(600,m+1,20,d)$	apc	Correct	Spurious	19/20	iter
9	2	1.6	6.1×10^{-8}	0.28	11	5	4	1483
11	3	4.7	3.2×10^{-7}	0.026	1	13	6	2443
13	4	5.2	4.2×10^{-7}	0.062	2	15	3	4178
15	5	2.8	2.3×10^{-7}	0.018	0	7	13	6495
17	6	0.88	7.1×10^{-8}	0.022	0	8	12	9272

Bottom-line: the *(9,2)-*, *(11,3)-*, *(13,4)-*, *(15,5)-* and *(17,6)*-motif problems are probably impossible to solve

Random Projections

- Idea: select t random positions and for each substring of length m of the text hash its selected positions into a table
- Hopefully, the cell corresponding to the planted motif will be the one with the highest count

Random Projection algorithm

- Parameters *(m,d), n, k, s,* possibly *i*
- Set $t < m-d$ and $4^t > k(n-m+1)$
- Build a table with all substrings of length *m*
- Repeat *i* times
  - Select randomly *t* positions
  - Repeat for all substrings in the table
    - Increase the count of the cell indexed by the *t* positions
- Select all cells with count $\geqslant s$

Random Projection algorithm

- We want $t < m-d$ because we want to sample from the "non-varying" positions
- The number of iterations *i* can be estimated from *m, d* and *t*

Random Projection algorithm

- Since we are hashing $k(n-m+1)$ substrings of size *m* into 4^t buckets, if $4^t > k(n-m+1)$ each bucket will contain on average less than one substring (set $s=1$)
- The constrain is designed to filter out the noise
- The bucket corresponding to the planted motif is expected to contain *more* motif instances than those produced by a random sequence

Random Projection algorithm

- If the constrain $4^t > k(n-m+1)$ cannot be enforced, the authors suggest to set $t = m-d-1$ and the threshold $s = 2\,[k(n-m+1)/4^t]$ (twice the average bucket size)

Motif refinement

- The algorithm will try to recover the unknown motif from each cell having at least *s* elements
- The primary tool for motif refinement is expectation maximization (EM)

Experiments

- Projection can handle the *(15,4)- (14,4)- (16,5)-* and *(18,6)*-motif problem *(k=20, n=600)*
- Winnower fails the *(14,4)- (16,5)-* and *(18,6)*-motif problem

Results

m	*d*	Gibbs	WINNOWER	SP-STAR	PROJECTION	Correct	iter
10	2	0.20	0.78	0.56	0.82	20	72
11	2	0.68	0.90	0.84	0.91	20	16
12	3	0.03	0.75	0.33	0.81	20	259
13	3	0.60	0.92	0.92	0.92	20	62
14	4	0.02	0.02	0.20	0.77	19	647
15	4	0.19	0.92	0.73	0.93	20	172
16	5	0.02	0.03	0.04	0.70	16	1292
17	5	0.28	0.03	0.69	0.93	19	378
18	6	0.03	0.03	0.03	0.74	16	2217
19	6	0.05	0.03	0.40	0.96	20	711

k=20, n=600, winnower *(t=2),* projection *(t=7,s=4, 20* random instances*)*

Remarks about Projection

- Pros:
  - fast and effective
- Cons:
  - need to know *m* and *d* in advance
  - randomized

Weeder

Weeder

- Proposed by Pavesi, Mauri and Pesole [Pavesi 2001]
- Draw ideas from PRATT by [Jonassen 1995, Jonassen 1997] and [Sagot 1998]
- It is an exhaustive approach for a particular class of rigid patterns

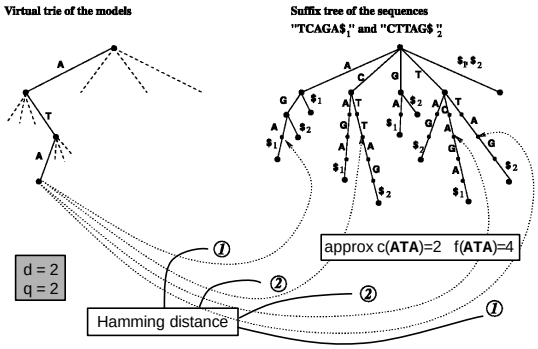
Exhaustive approach

- Suppose that you want to spell out all possible *(m,d)* rigid patterns that has at support least *q*
- One way to do it, is to use a (generalized) suffix tree [Sagot 1998]

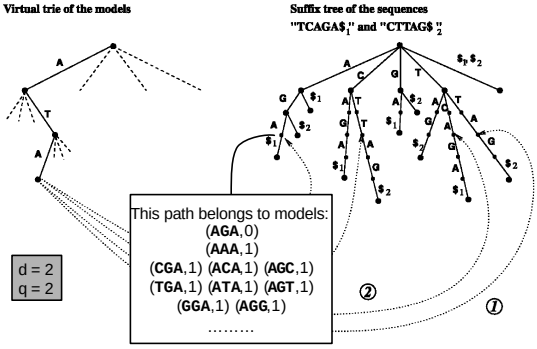
Idea [Sagot 1998]

- Any deterministic pattern (substring) *w* corresponds to a path in the tree ending in a node *u*, called the *locus* of *w* – the number of leaves in the subtree rooted at *u* gives the support
- Any model (rigid pattern) corresponds to a set of paths in the tree ending in nodes $\{u_1,u_2,\ldots,u_l\}$ – the total number of leaves in the subtrees rooted at $\{u_1,u_2,\ldots,u_l\}$ gives the support

Example



Example



Exhaustive approach [Sagot 1998]

- Start with all paths of length *d* with enough support (they represent valid models)
- At each path-extension keep track of the mismatches and the support
  - if the number of mismatches has not been reached the model will be extended by the symbols in Σ (therefore the number of models will be scaled up by a factor $|\Sigma|$)
  - otherwise we are allowed just to follow the arcs

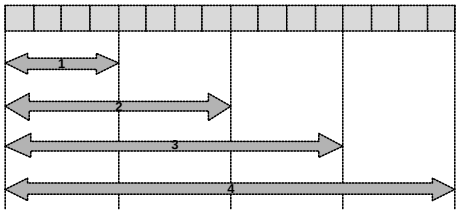
Time complexity [Sagot 1998]

- Finding all the models with support=occurrences in a single sequence takes $O(n\ N(m,d)) = O(n\ m^d\ |\Sigma|^d)$
- Note that the complexity is exponential (with *d)*

Weeder

- Pavesi *et al.*, implemented the algorithm by Sagot but it was running too slow, and they decided to change the class of patterns
- Weeder is designed to find rigid patterns which have an amount of mismatches proportional to their length (the same constrain applies also to all their prefixes)

Example ϵ =0.25



Time complexity

- By restricting the number of mismatches to εm, the time complexity becomes $O(n\, k \lceil 1/\varepsilon \rceil^{\varepsilon m} |\Sigma|^{\varepsilon m})$

The *(15,4)*-motif challenge … again

- Since the restriction on the density of the mismatches, the authors report that Weeder has probability *0.6* to catch the motif in ONE sequence
- Then, the probability of Weeded to get the motif in all the 20 sequence is almost zero
- On the other hand, running the Sagot's version is too time-consuming

Idea

- Split the set of sequence into two halves
- Run Weeder on each of the two sets requiring support *k/4* (instead of *k/2)*
- The probability that the *(15,4)*-motif will be in either subset is 0.98
- The pool of model candidates is then processed with Sagot's algorithm

Remarks about Weeder

- Pros:
  - Possibly exhaustive (if using Sagot's algorithm)
  - The relative error rate ε may be more meaningful than *d* and allows one not to specify in advance *m*
- Cons:
  - Very slow if run exhaustively - it cannot be considered exhaustive in practice

Irredundant and Tiling Motifs

Slides 132-153 by N. Pisanti, U. of Pisa

The problem

- Input: a string *s* of length *n*, a support *q*
- Output: all patterns *p* approximatively repeated at least *q* times in *s*
- The problem is inherently difficult due to the possible exponential output size
- [Parida 2000] Rather than finding all repeated patterns, only find a subset of them that
  - has polynomial size and can be computed efficiently
  - can generate all the others

Rigid motif = motif with don't cares

- Given $x = \Sigma^n$ and a support q, a pattern $w = (\Sigma \cup \{ . \})^*$ is a *motif* $\Leftrightarrow$
  - w starts and ends with a (solid) letter in Σ
  - w has at least q occurrences in x

Example

x = FABCXFADCYZEADCEADC $q = 2$,

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8

w_1 = A • C with L_1 = {1,6,12,16}

w_2 = FA • C with L_2 = {0,5}

w_3 = DC with L_3 = {7,13,17}

the symbol • matches all letters

Specificity of a motif

- For all $s \in \Sigma$ we have
  - $\bullet = s$
  - $s = s$

  … is less specific or equal to …
- recall that $w[i]$ is the i^{th} letter of w define $w[i] = \bullet$ for $i < 1$ and $i > |w|$
- w_1 is *less specific* than $w_2 \Leftrightarrow w_1[i] = w_2[i]$ for all i
- w_1 *occurs* in w_2 at $d \Leftrightarrow w_1[i] = w_2[i+d]$

Maximal motifs

- A motif w is *maximal* $\Leftrightarrow$ for all $y \neq w$ such that w occurs in y, we have $|L_w| > |L_y|$
- A motif is maximal $\Leftrightarrow$ however you specify further (i.e., extend and/or replace a • by a letter), you loose at least one occurrence

Examples

- x = FABCXFADCYZEADCEADC $q = 2$,
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8
- w_1 = A • C with L_1 = {1,6,12,16}, maximal
- w_2 = FA • C with L_2 = {0,5}, maximal
- w_3 = DC with L_3 = {7,13,17}, not maximal because DC occurs in ADC that occurs three times too

Redundant motifs

- w maximal is *redundant* $\Leftrightarrow \exists$ maximal motifs $y_1, y_2, \ldots, y_k \neq w$ such that $L_w = L_{y1} \cup L_{y2} \cup \ldots\ldots \cup L_{yk}$
- that is, the occurrences list of w can be recovered from those of $y_1, y_2, \ldots, y_k$

The basis of irredundant motifs

- In [Parida 2000] the set of all non redundant motifs has been suggested as a *basis*, that
  - has size at most $3n$ for all q
  - can be found in $O(n^3 \log n)$ for all q, and
  - can generate all maximal motifs

A n^2 lower bound

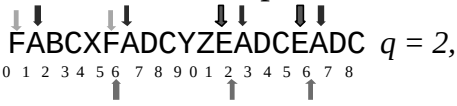
- In the word $x_k = \mathbf{A}^k \mathbf{X} \mathbf{A}^k$ there is an exponential number of maximal motifs
- By suitably prefixing x_k (increasing its size by a constant factor only), at least n^2 of them are also irredundant motifs

Tiling motifs

- Introduced by [Pisanti 2003]
- w maximal is *tiling* ⇔ there are no maximal motifs $y_1, y_2, \ldots, y_k \neq w$ and no integers $d_1, d_2, \ldots, d_k$ such that

$$L_w = (L_{y_1} + d_1) \cup (L_{y_2} + d_2) \cup \ldots \cup (L_{y_k} + d_k)$$

Example



w_3 = ADC with L_3 = {6,12,16} tiling

w_2 = FA.C with L_2 = {0,5} tiling

w_1 = A.C with L_1 = {1,6,12,16} tiled

by w_3 and w_4 because $L_1 = L_3 \cup (L_2 + 1)$
but it is irredundant

w_4 = EADC with L_4 = {11,15} tiling

The basis of tiling motifs

The basis of tiling motifs

- has size at most *n-1* for *q=2*
- can be found in $O(n^2 \log n)$ for $q = 2$
- can generate all maximal motifs for all supports
- it is a subset of the basis of irredundant motifs
- it is symmetric

The ⊕ operator

- Given two symbols $s_1, s_2 \in \Sigma$ with $s_1 \neq s_2$, we have $s_1 \oplus s_2 = \bullet$ and $s_1 \oplus s_1 = s_1$
- Given two strings $x_1, x_2 \in \Sigma^*$, we have $x_1 \oplus x_2 = t$, where $t[i] = x_1[i] \oplus x_2[i]$

The merges

$x = x[0]x[1]x[2]$ ..... $x[k]\,x[k+1]$ ..... $x[n-1]$

$x_1 =$ $x[0]x[1]$ ..... $x[k-1]\,x[k]$ ..... $x[n-2]$

.....

$x_k =$ $x[0]\;x[1]$ ..... $x[n-k-1]$

.....

$x_{n-1} =$ $x[0]$

$$merge_k = x \oplus x_k \quad for\ all\ 1 \le k \le n-1$$

Examples

x = FABCXFADCYZEADCEADC $q = 2$

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8

5 4

$merge_4$ = EADC

$merge_5$ = FA • C

$merge_6 = merge_{10}$ = ADC

$merge_{11} = merge_{15}$ = A • C

all other merges are empty

Tiling motifs and merges (*q=2)*

- All non empty *merges* of *x* are maximal motifs for *x*
- Each tiling motif is a *merge*
- There are at most *n*-1 tiling motifs
- The set of tiling motifs can be found by means of a suitable selection among the *merges*

The non tiling merges

- Among the *merges*, the non tiling motifs are those that are tiled by other *merges;* this check would cost $O(n^3)$ because $\Sigma_{merges\ w} |L_w| = O(n^2)$
- $merge_k$ has two obvious occurrences $occ_k = \{i, i+k\}$, where i is the first non • in $x \oplus x_k$
- For all distinct *merges w* we can collect L_w and
  $T_w = \bigcup_{1 \le k \le n-1} \{occ_k \mid merge_k = w\} \subseteq L_w$
  $[\ \Sigma_{merges\ w} |T_w| < 2n\ ]$
- For tiling *merges w*, we have $T_w = L_w$, *if* $T_w \neq L_w$ then *w* is not tiling

The algorithm

- Find all *merges* and their *occ* in $O(n^2)$
- Find all distinct *merges* in $O(n^2)$
- Find also theirs lists L in $O(n^2 \log n \log \Sigma)$
- Detect and discard tiled motifs in $O(n^2)$

1. Find all *merges* and their *occ*: *the $O(n^2)$ worst case complexity in not reached in practice. This step takes $O(n \cdot 1/p)$ where $p=1/|\Sigma|$ is the probability that two characters match (and 1/p the expected number of comparison to be done before finding a match).* $O(n|\Sigma|)$
2. Find all distinct *merges* with the lists T: *again, the time in practice is linear using hashing techniques.* $O(n)$
3. Find also theirs lists L (and discard those for which T ¹ L): *unfortunately meets the worst case.* *bottleneck*
4. Detect and discard tiled ones : *negligible.* $O(n)$

ABOUT 15 MINUTES ON THE WHOLE C.elegans GENOME (21 millions bases)

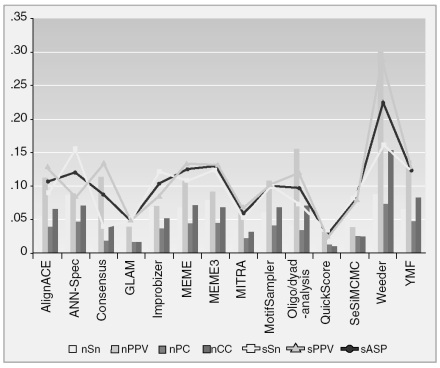
What if $q>2$?

- One can show an exponential lower bound on both bases for higher supports
- Again the word $x_k = \mathbf{A}^k \mathbf{X} \mathbf{A}^k$ can be prefixed in order to have $O(n^q)$ tiling (hence irredundant) motifs
- The efficient computation of any basis for unbounded supports is an open problem

Experimental comparison

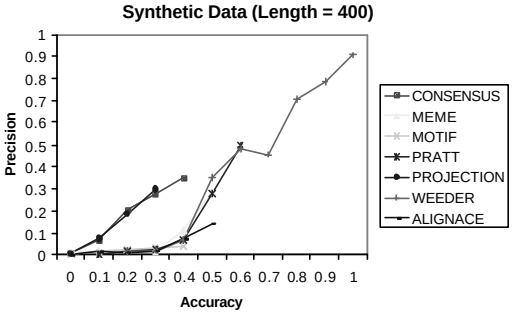
Experimental evaluation

- Recently [Tompa 2005] compared experimentally 13 pattern discovery tools
  - AlignACE, ANN-Spec, Consensus, GLAM, The Improbizer, MEME, MITRA, MotifSampler, Oligo/Dyad-analysis, QuickScore, SeSiMCMC, Weeder, YMF
- 52 datasets containing real binding sites from TRANSFAC (6 fly, 26 human, 12 mouse, 8 yeast)
- Computed several performance measures
- Main conclusions
  - Sensitivity is very low
  - The winner is Weeder



nSn=nucleotide sensitivity
nPPV=nucleotide positive predicted value
nPC=nucleotide performance coefficient
nCC=nucleotide correlation coefficient

sSn=site sensitivity
sPPV=site positive predicted value
sASP=site average performance coefficient
[Figure from Tompa et al. 2005]



References

- [Apostolico 2003] A.Apostolico, M.E.Bock, S.Lonardi, "Monotony of surprise and large-scale quest for unusual words", *Journal of Computational Biology*, vol.10, no.2/3, pp.283—311, 2003.
- [Weiner 1973] P. Weiner, "Linear pattern matching algorithm", *IEEE Symposium on Switching and Automata Theory*, pp. 1—11, 1973.
- [McCreight 1976] E. M. McCreight, "A Space-Economical Suffix Tree Construction Algorithm", *Journal of the ACM*, vol.23, no.2, pp.262—272 , 1976.
- [Ukkonen 1995] E. Ukkonen, "On-line construction of suffix trees", *Algorithmica*, vol.14, no.3, pp. 249—260, 1995.
- [Farach 1997] M. Farach, "Optimal Suffix Tree Construction with Large Alphabets", *Proc. Symposium on Foundations of Computer Science*, pp. 137—143, 1997.
- [Hui 1992] L. C. K. Hui, "Color set size problem with applications to string matching", *Proc. of Symposium Combinatorial Pattern Matching*, LNCS 644, pp. 230—243, 1992.
- [Muthu 2002] S. Muthukrishnan, "Efficient algorithms for document retrieval problems", *Symposium on Discrete Algorithm*, 2002.
- [Li 1999] Ming Li and Bin Ma and Lusheng Wang, "Finding similar regions in many strings", *Proc. of the ACM symposium on Theory of computing*, pp.473—482, 1999.
- [Pisanti 2003] N. Pisanti and M. Crochemore and R. Grossi and M.-F. Sagot, "A Basis of Tiling Motifs for Generating Repeated Patterns and its Complexity for Higher Quorum", *Proc. of Mathematical Foundations of Computer Science*, LNCS 2747, pp. 622—632, 2003.
- [Rigoustos 1998] I. Rigoutsos and A. Floratos, "Combinatorial pattern discovery in biological sequences: The Teiresias algorithm", *Bioinformatics*, vol.14, no.1, pp.55—67, 1998.

References

- [Sagot 1998] L. Marsan and M.-F. Sagot, Algorithms for extracting structured motifs using a suffix tree with application to promoter and regulatory site consensus identification", *Journal of Computational Biology*, Vol.7, no.3/4, pp.345—360, 2000.
- [Pevzner 2000] P. A. Pevzner and S.-H. Sze, "Combinatorial Approaches to Finding Subtle Signals in DNA Sequences", *Proc. Conference on Intelligent Systems for Molecular Biology*, pp. 269—278, 2000.
- [Buhler 2001] M. Tompa and J. Buhler, Finding Motifs Using Random Projections, *Proc. of Conference on Computational Molecular Biology* (RECOMB), pp. 67—74, 2001.
- [Pavesi 2001] G. Pavesi and G. Mauri and G. Pesole, An algorithm for finding signals of unknown length in DNA sequences", *Proc. of Conference on Intelligent Systems for Molecular Biology*, pp. S207—S214, 2001.
- [Parida 2000] L. Parida and Y. Gao and D. Platt and A. Floratos and I. Rigoutsos, "Pattern Discovery on Character Sets and Real-valued Data: Linear Bound on Irredundant Motifs and an Efficient Polynomial Time Algorithm", *Proc. Symposium on Discrete Algorithm*, 297—308, 2000.
- [Jonassen 1995] I. Jonassen I, J.F. Collins, and D. Higgins, Finding flexible patterns in unaligned protein sequences, *Protein Science*, vol.4, no.8, pp.1587—1595, 1995.
- [Jonassen 1997] I. Jonassen, Efficient discovery of conserved patterns using a pattern graph, *Comput. Appl. Biosci.*, vol.13, no.5, pp. 509—522, 1997.
- [Tompa 2005] M. Tompa, N. Li, T. L. Bailey, G. M. Church, B. De Moor, E. Eskin, A. V. Favorov, M. C. Frith, Y. Fu, W. J. Kent, V. J. Makeev, A. A. Mironov, W. S. Noble, G. Pavesi, G. Pesole, M. Regnier, N. Simonis, S. Sinha, G. Thijs, J. van Helden, M. Vandenbogaert, Z. Weng, C. Workman, C. Ye, and Z. Zhu, Assessing Computational Tools for the Discovery of Transcription Factor Binding Sites. Nature Biotechnology, vol. 23, no. 1, January 2005, 137-144.

THE END

Latest version of the slides at http://www.cs.ucr.edu/~stelo/slides/