String matching problems from Bioinformatics which still need better solutions

Gaston H. Gonnet
ETH Informatik
Zürich, Switzerland

SPIRE 2002, Lisboa, September 13th, 2002

The problems

- Coming from (molecular) Bioinformatics
- Real problems
- Returning a score rather than binary answer (and we search for the highest scoring matches)
- Not solved well (or efficiently) at present
- Common denominator: non-trivial function to score, typically a likelihood

The problems II

- Exact matches are never interesting

- The result is seldom used directly, most of the time it becomes part of the information going into a higher-level process
  - classification problems
  - structure prediction
  - multiple sequence alignments
  - phylogenetic trees

The data

Molecular information, for the purpose of this paper, consists of sequences of DNA/RNA (4 symbols) or sequences of amino acids (20 symbols) normally called proteins.

A DNA sequence:

`ATGATCGTAAATAACACACACGTGCTTACCCTACCACTTTATACCACCAC...`

An RNA sequence and the protein it encodes:

```
AUGAUCGUAAAUAACACACACGUGCUUACCCUACCACUUUAUACCACCAC...
 M  I  V  N  N  T  H  V  L  T  L  P  L  Y  T  T  T...
```

A protein or amino acid sequence:

`MIVNNTHVLTLPLYTTTTCHTHPHLYTDFTYAHGCYSIYHLKLTLLSDSTSLHGPS...`

The scoring function

A scoring function typically has the form

$$S_i = F_i(s[i], s[i+1], ..., s[i+L-1])$$

where $s[i]$ is the i^{th} symbol in the sequence and L is the length of the pattern to be scored.

E.g. for $L = 6$

$$S_1 = F_1(M, I, V, N, N, T)$$

This is too generic, quite often we have scores which are of the form:

$$S_i = R(F_i(s[i]), F_{i+1}(s[i+1]), ..., F_{i+L-1}(s[i+L-1]))$$

and R is a relatively simple reduction function.

E.g.

$$S_1 = \frac{\text{max}(F_1(M), F_2(I), F_3(V), F_4(N), F_5(N), F_6(T))}{1/6(F_1(M) + F_2(I) + F_3(V) + F_4(N) + F_5(N) + F_6(T))}$$

Scores and probabilities

An important class of scoring functions are derived from probabilities. If $Pr\{\text{event}_i\}$ denotes the probability of a certain event, for example the probability that the sequence is homologous to some pattern, then

$$S_i = -\log Pr\{\text{event}_i\}$$

is a very practical and reasonable scoring function.

In this context it makes perfect sense to search for the highest scoring sequences, and not just a single one or a single target score.

Identifying α-helices

The first problem is related to the identification of α-helices in protein sequences. A similar, but simpler, problem arises with the identification of β-sheets (or β-strands). We will concentrate on the first problem.

An α-helix is a secondary structure of proteins, found in almost every protein. About 10% of the total amino acids in any genome are part of α-helices. Recognition of α-helices from the primary structure, that is the sequence of amino acids, is usually called "secondary structure prediction", and is a very important and active area of research.

α-helices

An α-helix is a helical structure which rotates exactly 100° with every amino acid. Its existence and stability is normally due to its surroundings, a helix cannot stand by itself. In particular, most helices find themselves in the interface between the inner and outer part of the protein. This information is very useful to find (or confirm) helices, as one side of the helix is hydrophobic (the inner side) and the other is hydrophilic (the outer side, typically exposed to water).

Hydrophobicity (hydrophilicity)

Hydrophobicity is the property of "disliking" water, like oily substances. Each amino acid has an index of hydrophobicity. Hydrophobicity indices typically range from -1 to 1. An index of 1 means that the amino acid is highly hydrophobic, and index of -1 means it is highly hydrophilic (likes water). A natural score for a sequence of L amino acids starting at position j is the the internal product:

$$R_j = \sum_{i=j}^{j+L-1} H_i \sin\left(\frac{2\pi i \times 100}{360} + \theta\right)$$

where θ is the phase angle of the helix with respect to the inner/outer limit and H_i is the hydrophobicity index of amino acid i.

Building a scoring function

The score for each j and L depends on θ, so we first have to find the best θ for each position j.

Expanding and collecting we obtain:

$$R_j = S_j \cos\theta + C_j \sin\theta$$

with

$$S_j = \sum_{i=j}^{j+L-1} H_i \sin 5/9\pi i \qquad C_j = \sum_{i=j}^{j+L-1} H_i \cos 5/9\pi i$$

The phase angle θ which gives the maximum score for each j is easily obtained by solving the derivative of R equated to 0. The θ giving the maximum is not interesting, the maximum score is:

$$R_j = \sqrt{C_j^2 + S_j^2}$$

A linear time searching algorithm

A linear time algorithm is easy to derive (for a fixed L). The values of C_j and S_j can be updated in constant time by adding the $(j+L)^{th}$ term and subtracting the j^{th} term.

It is clear that such an algorithm will work for any score which is a function of sums (or other functions with an inverse) over individual values.

Not quite there yet

Although this algorithm is linear, it is still not good enough, as the parameter L depends on nature, and it has values from 5 to 30.

This imposes a new constraint on the problem, we would like to solve this searching problem for a score which is

$$R_j = \max_{L=5..30} \sqrt{C_j^2 + S_j^2} = \sqrt{\max_{L=5..30} C_j^2 + S_j^2}$$

Problem1: To do this quickly remains an open problem. (Even if it is more efficient in the constant term, not necessarily asymptotically.)

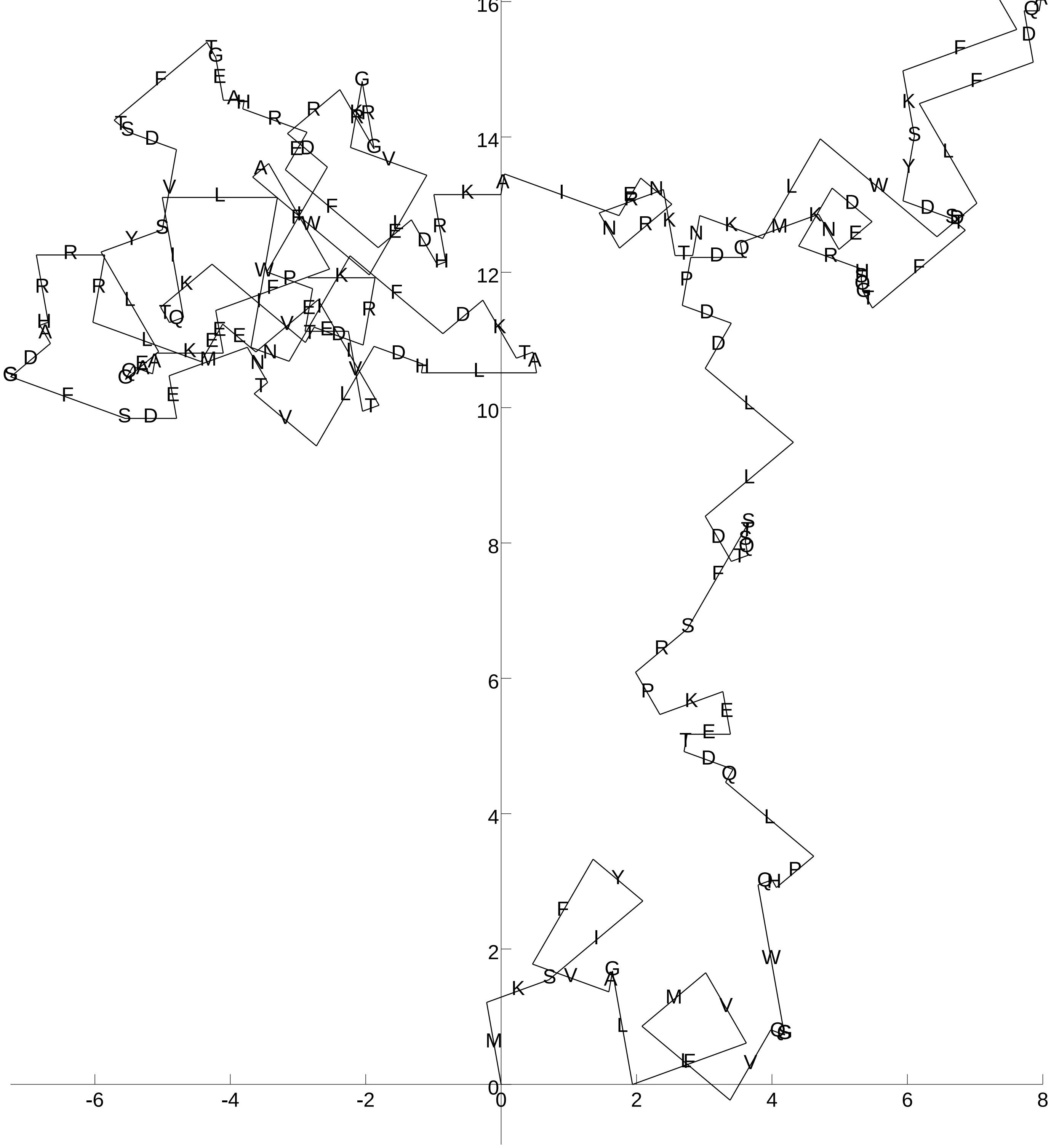
Geometrical interpretation

This optimal score has a surprisingly simple geometrical interpretation. If every H_i is viewed as a vector from $(0,0)$ to $(H_i \cos 5/9\pi i, H_i \sin 5/9\pi i)$, then we are looking for the longest path of L consecutive vectors.

So what we are doing is searching for long straight stretches in a consecutive graph of the H_is.

We can solve this problem for all possible values of L by finding the convex hull of this set of points and then finding its diameter. This would require $O(n \log n)$ time.

The graph of H_i



16
14
12
10
8
6
4
2
0
-6
-4
-2
0
2
4
6
8

Other scoring functions

Other scoring functions for $\alpha-$helices may also be interesting, for example

$$R_j = \sqrt{\max_{L=5..30} \frac{C_j^2 + S_j^2}{\sum_{i=j}^{j+L-1} H_i^2}}$$

Problem 2: Are there any mathematical properties of the score R which make this search more efficient?

Reduction functions

We call reduction functions any function which maps a set of values into a single value. E.g. sum, max, ave, etc. Earlier we stated that a reduction function which accepts an inverse is easy to compute. For example for sum, we can compute:

```
run_sum := sum( F(s[i]), i=1..L );
for i from 1 to n do
    .... evaluate scoring function ....
    run_sum := run_sum - F(s[i]) + F(s[i+L]);
    od;
```

Is it possible to do this efficiently for functions which are associative but do not necessarily have an inverse (a monoid)? For example, max? (or the inverse is extremely expensive, like matrix multiplication).

Updating max in $O(1)$ time

A little known fact is that monoid operations can also be updated in constant time using in total $O(L)$ space. It goes as follows:

Let k be the largest integer such that $2^k \leq L/2$

We compute k vectors ${}^{i}F_j$ which contain 2^i summarized results each. More precisely

$${}^{i+1}F_j = \max({}^{i}F_j, {}^{i}F_{j+2^i}) \qquad {}^{0}F_j = F_j(s[j])$$

The only ${}^{i}F_j$ which are kept are the ones for which j is a multiple 2^i.

Maintaining these values requires on the average $\frac{2^k-1}{2^k}$ operations per position.

The ${}^{i}F_j$ arrays

j	F_j	${}^1F_j$	${}^2F_j$	${}^3F_j$
3	8.1467			
4	6.1208	7.2938	7.7943	
5	7.2938			
6	2.4366	7.7943		
7	7.7943			
8	0.1776	6.5518	9.9351	9.9351
9	6.5518			
10	9.9351	9.9351		
11	4.4696			
12	6.8383	6.8383	6.8383	
13	1.9662			
14	5.1864	6.3316		
15	6.3316			

Max in $O(1)$ time II

The values of ${}^{i}F_{j}$ can be kept in circular arrays of length $\left\lceil \frac{L}{2^i} \right\rceil$.

From the above it is now trivial to see that we could compute the max in $O(\log L)$ steps (which is not a bad practical variant).

However, if we want to save the $\log L$ factor or the monoid operation is very expensive, then we can use the following algorithm.

Max in $O(1)$ time III

We will compute the values of the maximum for increasing values of j in $O(1)$ operations by doing the following grouping. This grouping is best illustrated by an example. Suppose that $L = 20$ hence $k = 3$, and $j = 77$ then

$$\max(\overbrace{\max(F_{77}, \max({}^1F_{78}, {}^3F_{80}))}^{\text{tail computation}}, \quad \overbrace{\max({}^3F_{88}, F_{96})}^{\text{head computation}} \quad)$$

Notice the grouping, which identifies the 2^k blocks in the middle and groups on the right for the left part and on the left for the right part.

Max in $O(1)$ time IV

It is easy to see, that as the computation proceeds, with this grouping, $O(1)$ computation is needed at each step.

$$\max(\overbrace{\max({}^1F_{78}, {}^3F_{80})}^{\text{tail}}, \overbrace{\max({}^3F_{88}, {}^1F_{96})}^{\text{head}})$$

$$\max(\overbrace{\max(F_{79}, {}^3F_{80})}^{\text{tail}}, \overbrace{\max(\max({}^3F_{88}, {}^1F_{96}), F_{98})}^{\text{head}})$$

$$\max(\overbrace{{}^3F_{80}}^{\text{tail}}, \overbrace{\max({}^3F_{88}, {}^2F_{96})}^{\text{head}})$$

Probabilistic sequences

A probabilistic sequence (also called profile) is a sequence which is defined by probability vectors for each position. It is interpreted as each position having several choices (all the values of the probability vector which are non zero). This is a natural generalization of matching a regular expression, where we additionally want to find the most likely match (highest probability) of all the possible ones.

The following is an example of a profile or probabilistic sequence of length 3 for DNA:

pos	A	C	G	T
1	0	0.50	0.25	0.25
2	1	0	0	0
3	0.25	0.25	0.25	0.25

Uses of probabilistic sequences

There are three main uses of probabilistic sequences:

- Probabilistic ancestral sequences

- Patterns

- Motifs

The scores that we compute in this section can be used for direct string matching or can be used in the dynamic programming alignment of sequences with indels.

Scoring probabilistic sequences

Scoring an amino acid against a probabilistic position is not completely trivial if it is to be done according to the standard model of evolution. The score is given by the quotient between the probability of being homologous, i.e. coming from a common ancestor divided the probability of being matched by chance (null hypothesis).

$$S = 10 \log_{10} \frac{Pr\{\text{common ancestor}\}}{Pr\{\text{random occurrence}\}}$$

Scoring probseqs II

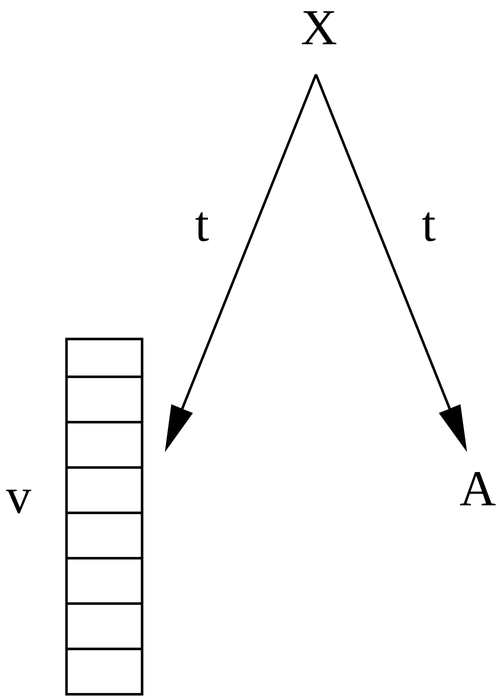


Figure 1.

Let v be the probability vector to be scored against amino acid A and t the distance between x and A and between x and v.

Scoring probseqs III

The quotient of probabilities is:

$$\frac{Pr\{\text{common ancestor}\}}{Pr\{\text{random occurrence}\}} = \frac{\sum_x f_x \left(\sum_i Pr\{x \to i\} v_i\right) Pr\{x \to A\}}{\left(\sum_i v_i f_i\right) f_A}$$

That is, the probability is simply summed over all possible values of the ancestor (x) and all possible values of the probabilistic position (i). Using Dayhoff's mutation matrices, doing some simplifications and taking logarithms to have an additive score, we simplify the score to:

$$S = 10 \log_{10} \frac{u_A}{fv}$$

where $f = (f_1, f_2, ...)$ is the vector of natural amino acid frequencies, $u = v^T M^{2t}$ and M is a 1-PAM mutation matrix.

Scoring probseqs end

By precomputing the vectors u and fv for the pattern, we can compute the score for each position in time $O(1)$.

Probseqs vs. probseqs

Scoring a probabilistic sequence against another is done following the same lines.

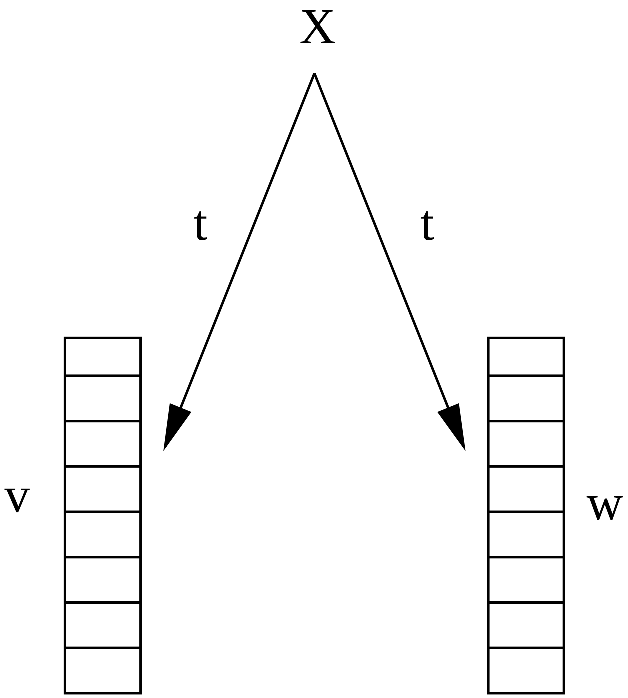


Figure 2.

Probseqs vs. probseqs II

Figure 2 describes the situation of scoring one particular position against another. This time we have a triple sum, over the ancestor and over each of the two probabilistic descendants, i.e.:

$$\frac{Pr\{\text{common ancestor}\}}{Pr\{\text{random occurrence}\}} = \frac{\sum_x \sum_i \sum_j f_x Pr\{x \rightarrow i\} v_i Pr\{x \rightarrow j\} w_j}{fv\; fw}$$

We use the same technique as before to simplify the computation and obtain:

$$S = 10 \log_{10}(u^* w^*)$$

where $v^* = \frac{v}{fv}$ can be precomputed for the first sequence, $w^* = \frac{w^T M^{2t} F}{fw}$ can be precomputed for the second sequence and F is a diagonal matrix containing the natural frequencies, i.e. $F_{ii} = f_i$.

Probseqs vs. probseqs end

The computation of the score, besides precomputation, requires an internal product of two vectors of size 4 or 20 (number of symbols). This is quite efficient since the precomputation is dwarfed by the matching phase.

Problem 3: Is is possible to compute the scores more efficiently for this case?

Final question/problem

Problem 4: All these string matching algorithms assume a brute force underlying string searching strategy. Is it possible to improve the efficiency by using the ideas of better string searching algorithms to these scores?