Biosequence Algorithms, Spring 2007

Lecture 11: Sequence Database Searching

Pekka Kilpeläinen
University of Kuopio
Department of Computer Science
Bio-sequence databases are the dominant and most successful application of string algorithms in computational biology.

These “databases” are rather archives or collections of sequences than databases in the strict sense of the word.

Overview of this section:

1. Some motivation
2. Sequence DB’s and their use
3. FASTA and BLAST heuristics
Biological discoveries based on sequence similarity are now routine

First such discovery was the connection btw oncogenes and growth-regulating proteins

*Simian sarcoma* is a *retrovirus* that causes cancer in monkeys

It’s *oncogene* called *v-sis* was sequenced in 1983 (Doolittle et. al)

Comparing the amino acid sequence encoded by *v-sis* against published protein sequences revealed significant similarity with a *growth factor* called PDGF
The genome of bacterium *Haemophilus influenzae* was reported in 1995 (Fleischmann et al.)

- 1,743 assumed coding regions were translated into amino acid sequences, and searched for similarity in the Swiss-Prot database
- 1,007 of them matched s.t. the biochemical function could be deduced for each of them
This section probably somewhat outdated?

Major sequence archives of mid-90’s contained

- DNA seqs from 300,000 genes, totaling > 500 Mbp
- fragments of ~ 100,000 proteins, totaling ~ 25 × 10^6

**GenBank** (USA) stores and facilitates retrieval of all published DNA sequences

- ~ 60 × 10^6 sequences, totaling ~ 65 × 10^9 bp (Aug’06)

**EMBL data library** (Europe) and **DNA DataBase of Japan** (DDBJ) are “essentially equivalent”
Major **protein sequence archives** include **Protein Information Resource (PIR, USA)** and **Swiss-Prot (Europe)**

In addition, there are hundreds(?) of different specialized databases (e.g., for specific organisms, cells, or biological functions)

**Databases support sequence retrieval**

- by a sequence title or access number
- by similarity with a query sequence
- over the Web
- on a local machine, after downloading the DB and software for searching, viewing and manipulating it
Heuristic Database Searching

Exact similarity computation btw a query string and database sequences happens with $\Theta(nm)$ time dynamic programming.

With current technology aligning a query against the entire database is not feasible, even with special purpose hardware.

- There are specialized chips for sequence alignment, and services that search databases on a 4,000-processor computer.

Increase in raw computing power may change the situation in future.
Sequences similar to a new one are normally searched with fast **heuristic methods**, before (or instead of) exact similarity computation.

Dominant search applications: **FASTA** and **BLAST**

- are suites of programs tuned for different problem domains (DNA, protein, DNA translated to protein etc.)
- *exclude* large parts of the DB from more careful and time-consuming examination
- do not permit precise analyzes of speed or accuracy
Search is performed by finding good **local alignments** btw the query sequence and the DB sequences

Base of **filtering**:

- good alignments usually include short identical or highly similar fragments

~~First, exact or highly similar occurrences of query subwords are located~~

Then these are extended to longer alignments of sufficiently high similarity
**FASTA**

**FASTA** (short for “fast-all”) is one of the first widely used programs for searching protein and DNA sequence databases (Lipman & Pearson, 1985, 1988)

Current version is 34t26 (2006)

**Very high level description:**

Let $P$ be a query sequence, and $T$ any database sequence (each considered in turn)

A user-specified parameter $ktup$ is used for locating exact occurrences of $ktup$-length substrings ($k$-tuples) of $P$ in $T$

Defaults of $ktup$ are 6 for DNA and 2 for protein sequences
Locating $k$-tuples

1. FASTA computes **hot-spots**, which are pairs $(i, j)$ s.t. $k$-tuple $P[i \ldots i + ktup - 1]$ occurs at $T[j]$

Hot-spots are found by **hashing**

- A hash table of all database $k$-tuples could be precomputed, if there is sufficient space; Then any $k$-tuples of the query string $P$ are looked up from there

- Otherwise all $k$-tuples of $P$ are hashed to the table, and each $k$-tuple of $T$ is looked up from there
Diagonal Runs

Consecutive hot-spots along a common diagonal of the dynamic programming table of $P$ and $T$ form one or more **diagonal runs**

- the **diagonal** $d$ consists of cells $(i, j)$ with $j - i = d$

- FASTA evaluates runs by scoring each hot-spot positively and the space (non-matches) between consecutive hot-spots negatively, and keeps the 10 highest scoring runs

By processing the $k$-tuples of either $P$ or $T$ left-to-right, the diagonal runs can be collected and evaluated in linear time wrt the number of hot-spots found.
Diagonal run ~ a local alignment btw $P$ and $T$ without gaps

2. *(Rescoring)*
FASTA computes an optimal *subalignment* (pair of aligned substrings) from the query-target substrings corresponding to each of the 10 best diagonal runs

- using an amino-acid scoring matrix
  (e.g., PAM250, if comparing protein sequences)

The best subalignment is reported as *init1*
3. FASTA tries to combine “good” subalignments (with score above specified cutoff) of Step 2 into a single high-scoring alignment with gaps.

For this, each subalignment is presented as a node of a directed graph, and its score as a positive weight of the node.

A node \((P[i \ldots i + l], T[j \ldots j + l])\) has an edge to each subalignment starting at positions \((i', j')\) with \(i' > i + l\) (and \(j' > j + l\), too?)

The edge is given a negative gap weight (based on the distance btw \((i + l, j + l)\) and \((i', j')\))
FASTA then finds a maximum-weight path in this graph.

The result specifies a local alignment which is reported as \textit{initn}.

\textit{initn} may be a sub-optimal local alignment, but it is often close to an optimal one.

The scores of \textit{init1} and \textit{initn} are used to rank the database sequences (How?)
For the highest scoring DB sequences, 4. FASTA computes a third local alignment by extending the diagonal of subalignment init1 into a band of 16–32 diagonals, and then applying Smith-Waterman algorithm restricted to this band.

The local alignment found by this step is reported as opt.

Database sequences are ranked according to their opt scores.

The highest scoring ones are finally examined by the full Smith-Waterman local alignment algorithm.
Estimating Significance

Scores of \textit{init1}, \textit{initn} and \textit{opt} of all database sequences are used to estimate the statistical significance of the highest similarities found btw the query string and database sequences.

\begin{itemize}
  \item \textit{Is the best opt} score significantly higher than the similarities found with many other database sequences?
\end{itemize}
BLAST (Altschul, Gish, Miller, Myers & Lipman, 1990) is the dominant search program for bio-sequence databases

- short for “Basic Local Alignment Search Tool(s)"

Confluence of three lines of research:

- Lipman et al. to improve hot-pot selectivity
- sub-linear expected-time approximate matching of Myers
- probability estimates for the statistical significance of reported matches by Karlin, Altschul & Dembo
BLAST Functionality and Concepts

BLAST searches from the query sequence and each database sequence local regions of high similarity without gaps.

Intuition: similar regions of equal length between proteins suggest functional similarity; Insertions/deletions tend to change the shape and thus the function of a protein.

Given strings $P$ (query) and $T$ (DB sequence), a segment pair is a pair of substrings $P'$ of $P$ and $T'$ of $T$ aligned without spaces

$( \sim \rightarrow |P'| = |T'| )$
A segment pair is **locally maximal** if it cannot be extended or shortened at either end without decreasing its score.

A segment pair of maximal score is a **maximal segment pair** (MSP).

BLAST tries to find and report all DB sequences that have an MSP with $P$ above a cutoff score $C$.  

- selected (by BLAST automatically?) to make it unlikely for random sequences to have MSP’s with $P$ of score above $C$.  

---

BSA Lecture 11: Sequence DB Searching – p.20/24
BLAST Hit Finding Strategy

BLAST considers substrings of a given length $w$ called words ($\sim k$-tuples of FASTA)

BLAST locates words of the DB and words of $P$ that have a non-space alignment with score above a fixed threshold $t$

These hot-spots (called hits) are then extended into segment-pairs with score above $C$, if possible

Hits are located applying exact set matching
Locating and Examining Hits

For searching protein sequences, BLAST(P) constructs for each length-$w$ substring $\alpha$ of $P$ a $t$-neighborhood:

1. all length-$w$ strings of similarity $\geq t$ with $\alpha$

BLAST builds an AC-like automaton of the $t$-neighborhood words; Hits are then located by a linear scan of $T$

2. For searching DNA, the (unmodified) length-$w$ substrings of $P$ are searched, only ($w \sim 12$)

BLAST tries to extend each hit into a locally maximal segment pair with score above $C$; A heuristic restricts the expected time of checking a hit to $O(1)$ (but may miss some MSPs)
BLAST avoids quadratic-time dynamic programming

- Typically 50× faster than Smith-Waterman
- Latest versions also apply S-W to high-scoring DB seqs

Word length $w$ and neighborhood similarity bound $t$ need to be selected to minimize

1. the probability of missing an MSP of score above $C$
2. the size of $t$-neighborhoods, and
3. the frequency of hits

For protein, values like $w = 3–5$ and $t = 17$ (with a version of PAM matrices) are reported as good compromises
The methods are in the below order of relative efficiency:

1. BLAST
2. FASTA
3. Dynamic programming (Smith-Waterman)

BLAST and FASTA are relatively close to each other.

Current opinion seems to be that the order of relative accuracy is the same.

BLAST is competitive with FASTA, but less accurate especially if biologically significant alignments include spaces.