(Rapid) Local Sequence Alignment
BLAST

BLAST: Basic Local Alignment Search Tool

BLAST is a family of **rapid approximate** local alignment algorithms[2]. BLAST is usually used to match a single DNA sequence S to a database $D = \{D_1, \dots D_N\}$ of DNA sequences.

Different variants of BLAST produce alignments for S and D represented in either an alphabet of nucleotides or an alphabet of amino acids (in fact, S and D may be in different alphabets!).

BLAST outputs two things:

- Good alignments between the query string S and strings from D;
- A p-value: the estimate of probability that the reported alignment can occur by chance.

The output of BLAST is usually sorted in ascending order by p-value (i.e., the lower the probability of a chance match, the better the local alignment is).

BLAST in a Nutshell

BLAST consists of three key procedures:

1. **Rapid search for *seed matches*.** On this stage, for each string $D_i \in D$, any locations that can have a "good" local match are rapidly identified. (*the specifics of this part is one of two things what makes BLAST different from other methods*).
2. **Completion of local matches.** *Seed matches* are extended to form local alignments. (*Different variations of BLAST have used different strategies for extending seed matches.*).
3. **Estimation of p-values.** For each local alignment, the probability that it may occur by chance is estimated. The produced matches are sorted in ascending order by the p-value. (*This is the second "specialty" of BLAST*).

Rapid Search for Seed Matches

Inputs. The problem involves the following inputs:

- alphabet $\Sigma = \{a_1 \dots, a_M\}$;
- string $S = s_1 \dots s_n$ called *query string*;
- string $T = t_1 \dots t_m$ called *database string*;
- substitution matrix $Score : \Sigma \times \Sigma \longrightarrow \mathcal{R}$;
- value τ, a *similarity threshold*, for *seed alignments*;
- an integer $k << \min(m, n)$, the length of the *seed alignments*.

Seed alignments. A pair of substrings $S_i = s_i \dots s_{i+k-1}$ and $T_j = t_j \dots t_{j+k-1}$ of length k is called a *seed alignment* iff

$$Score(S_i, T_j) = \sum_{l=0}^{k-1} Score[s_{i+l}, t_{j+l}] \geq \tau.$$

Problem: given the inputs above, find all pairs (i, j), such that (S_i, T_j) is a *seed alignment. Do it fast!*

Idea. Σ is a constant-length alphabet. In BLAST, k - the length of a seed alignment **is a constant**.

- For $\Sigma = \{A, T, C, G\}$ (alphabet of nucleotides), k is usually set to be in the range between 9 and 12 (a common value is 11).
- For $\Sigma =$ the amino acid alphabet, $k = 3$.

BLAST uses the following key observations:

Observation 1: The number of all possible strings of size k in alphabet Σ, $|\Sigma|^k = M^k$ is a constant!

Observation 2: For each k-tuple $V = v_1 \dots v_k$, the set of all k-tuples $W_1, \dots, W_s$, such that

$$Score[V, W_i] \geq \tau$$

can be precomputed in advance in constant time!

Note: In fact, given k, BLAST precomputes the matrix $Score_k$ of similarity scores between all pairs of k-tuples from $|\Sigma|$:

$$Score_k : \Sigma^k \times \Sigma^k \longrightarrow \mathcal{R},$$

such that

$$Score_k[v_1 \dots v_k, w_1 \dots w_k] = \sum_{i=1}^{k} Score[v_i, w_i].$$

Then, based on the input value of τ, for each k-tuple V, BLAST computes the list $Neighbors(V)$ of all k-tuples $W_1, \dots W_s$, such that $Score_k[V, W_i] \geq \tau$.

Rapid Search Procedure. String $S = s_1 \dots s_n$ has $n-k+1$ k-tuples:
$S_1 = s_1 \dots s_{k-1}$
$S_2 = s_2 \dots s_k$
...
$S_{n-k+1} = s_{n-k+1} \dots s_n$

The rapid *seed match* search proceeds as follows:

1. For each S_i, retrieve $Neighbors(S_i)$.
2. Construct set $Neighbors(S) = \cup_{i=1}^{n-k+1} Neighbors(S_i)$.
3. For each string $V \in Neighbors(S)$, search for all occurrences of V in T.
4. Report a *seed match* between each S_i, such that $V \in Neighbors(S_i)$ and each T_j, such that $T_j = V$.

Why this works. **Step 3** in the procedure above is a repeated search for **exact matches** between a k-tuple V and substrings of T. This can be done in time linear in m (length of T). Because the size of $Neighbors(S)$ is constant (it is less than $|\Sigma|^k$) [1], **Step 3** takes $O(m)$ time.

In practice, the rapid matches can be done in a number of ways:

- **Suffix trees.** A suffix tree for T can be precomputed. If k is known in advance, a traversal of the suffix tree can be used to label all internal nodes with node-paths of size k (or the next closest size) with the list of leaf nodes in the subtree.

  This suffix tree can be used to efficiently produce the list of seed matches, when searches for k-tuples from $Neighbors(S)$.

- **Indexing.** An index of k-tuple occurrences in T can be precomputed in advance and stored in an easy-to-access structure - e.g., in a *hashmap*. Given a k-tuple V from $Neighbors(S)$, the list of all its occurrences is retrieved in $O(1)$ time from the index structure.

- **Aho-Corasick algorithm.** Aho-Corasick algorithm[1] solves the problem of searching for a set $V = \{V_1, \dots, V_s\}$ of exact matches in a string T in time $O(n+m+z)$ where, n is the length of all strings from V, m is the length of T and z is the total number of matches found.

  The algorithm works by efficiently representing the collection of strings V as a *keyword tree* with *backlinks* the provide for efficient navigation when mismatches are found.

  The algorithm itself uses T to traverse the keyword tree for V. Each time a match is found, the keyword tree is navigated following a tree path. Each time, there is a mismatch, a sequence of *fail* jumps occurs.

  Aho-Corasick algorithm can be used to match any string T and the keyword tree constructed from the set $Neighbors(S)$.

By the numbers. What does it take to precompute the necessary structures?

[1] Although it can be a rather large number.

Amino Acid alphabet. For the alphabet of amino acids we have:

- $|\Sigma| = 21$ (20 amino acids plus the stop codon).
- $k = 3$. This is the usual length used in BLAST.
- Substitution matrices used. BLOSUM62 is typically used. Other matrices in the BLOSUM family can be used. PAM family is used but not as commonly.
- Total number of combinations: $21^3 = 9261$.
- Size of the $Score_k$ matrix: $9261 \cdot 9261 = 85,766,121$ (85+ million).

Nucleotide alphabet. Things are a bit more "interesting":

- $|\Sigma| = 4$.
- $k \in \{9, 10, 11, 12\}$. Usualy value is $k = 11$.
- Substitution matrices used. Usually, $Score[X,X] = 5$, $Score = [X,Y] = -4$ for $X \neq Y$ is used.
- Total number of combinations: $4^11 = 2^22 = 4,194,304$ (over four million).
- Size of $Score_k$ matrix: $4^11 \cdot 4^11 = 2^44 = 17,592,186,044,416$ (over 17.5 trillion).

Completion of Local Matches

Step 2 of BLAST. Once *seed matches* are found, each of them needs to be extended to the best/longest possible match.

Different BLAST versions differ on how this step is handled. In the original BLAST algorithm[2], the process of extension was as follows:

1. For each *seed match* (i, j) reported on **Step 1** of BLAST:
   - extend it in both directions for as long as the score of the new match is above the threshold τ.
   - stop, when the match cannot be extended on either side without its score falling below τ.
   - Report the computed match.

Variants. The origial version did not create local alignments with gaps. Subsequent versions of BLAST improved on this process in a number of ways:

- Gapped alignments. When extending the *seed matches* use a substitution matrix **and** and *indel score* δ to score the alignments.
- Filter out seed matches. Only extend *seed matches* which show up in pairs on the same diagonal within a given number A of positions. This variant filtered out a large number of *seed matches* and improved the performance of BLAST.

Computing p-values of Alignments

Poisson distribution. A discrete random variable X has Poisson distribution with the mean (expected) value λ if the probability $P(X = k)$ is

$$P(X = k) = \frac{\lambda^k \cdot e^{-\lambda}}{k!}$$

Intuition. Consider a certain, low-probability event α that can occur with probability p at each moment of time. Consider a sequence of n independent trials for α. Let X be a discrete random variable that counts, how many times α occurs. When p is reasonable and n is relatively small, the probability that α occurs exactly $k \leq n$ times can be described exactly using the *binomial distibution*:

$$P(X = k) = p^k(1-p)^{n-k}.$$

However, when p is *very small*, but n is *very large*, binomial distibution is not convenient to use. Poisson distribution is an approximation of the binomial distibution in such a situation.

Given n trials, the expected number of times α occurs is np. If n is very large and p is very small, np may be a mid-range number. Variable X will have Poisson distribution with the expected value $\lambda = np$.

Match by chance. Consider a query string S and a database string T, for which BLAST returns a local alignment with a score $\tau' \geq \tau$. In general, two DNA sequences have a good local alignment *if they are related and/or serve similar purposes*. So, *what is the probability that this local alignment of S and T* ***occurred by chance****?*

Simple example. Let (i, j) be a pair of random positions in S and T. Let p be the probability that two letters occurring at random positions of two strings match[2].

An *exact match* of length k starting at position i in S and j in T, then has the following probability of happening by random chance:

$$p' = (1-p)p^k.$$

(Here, $1-p$ is responsible for the match **starting** at s_i and t_j. This means that $s_{i-1} \neq t_{j-1}$, which has the probability of $1-p$).

There are nm possible alignments of S and T. Therefore, the expected number of random alignments of two strings of length k in S and T is

$$\lambda = nm(1-p)p^k.$$

The number of random alignments is a random variable with Poisson distribution with the expected value of λ.

[2]For the nucleotide alphabet this probability, under the assumption of uniform distribution of nucleotides in the DNA strings is $p = \frac{1}{4 \cdot 4} = \frac{1}{16}$. For the alphabet of amino acids, this probability is $p = \frac{1}{21 \cdot 21} = \frac{1}{441}$.

Altschul-Dembo-Karlin statistics. In BLAST, the match between two substrings does not have to be exact, in order to qualify for a good local alignment, so the math is a bit more complex. The Altschul-Dembo-Karlin statistic estimates the expected number of such matches in a pair of strings $S = s_1 \dots s_n$ and $T = t_1 \dots t_m$ as

$$E(\tau) = Kmne^{-\lambda \cdot \tau},$$

where:

- K is constant.
- τ is the similarity threshold.
- λ is the positive root of the following equation:

$$\sum_{s \in \Sigma} \sum_{t \in \Sigma} p_s \cdot p_t \cdot e^{\lambda \cdot Score(s,t)} = 1.$$

  Here p_s and p_t are the frequencies of characters s and t.

The probability that there is a match of a score greater than τ between two "random" subsequences of S and T is

$$P = 1 - e^{E(\tau)}.$$

From these statistics, the p-values for each alignment are computed.

References

[1] Alfred Aho, Margaret Corasick (1975) Efficient String Matching: an Aid to Bibliographic Search, *Communications of the ACM*, Vol. 18, No. 6, pp. 333-340.

[2] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, David J. Lipman (1990), *Journal of Molecular Biology*, Vol. 215, pp. 403–410.