The program BLAST

- BLAST is the most popular program to perform sequence database searches.
- The name is an acronym for *Basic local alignment search tool*.
- As in [Altschul et al., 1990], we will describe BLAST for the case where the input sequences are proteins.
- Therefore the corresponding subprogram of BLAST is BLASTP.
- Nevertheless we will use the term BLAST and not BLASTP here.
- Our exposition very closely follows [Cameron et al., 2004].
- The BLAST algorithm is a five-stage process that is efficient and sensitive when searching sequence databases for local alignments.
- The steps progressively reduce the search space, but each is more fine-grain and takes longer to process each sequence.
- Table 1 shows the average time spent performing each stage of the algorithm.

Table 1: Average running time of each stage of the BLAST algorithm according to the evaluation of [Cameron et al., 2004].

Stage	Task	relative overall time
1	find blast hits	37%
2	identify pairs of hits on the same diagonal	18%
3	perform ungapped extension	13%
4	perform gapped extension	30%
5	collect traceback information and display alignments	2%

Stage 1: find blast hits

- In the first stage, each database sequence is compared against the query sequence to compute blast hits.
- To explain what this is, suppose we want to search a protein sequence database, given a score function σ, satisfying $\sigma(\alpha \to \beta) = -\infty$ for any deletion or insertion operation $\alpha \to \beta$.
- That is, the model of blast hits does not allow for insertions and deletions.
- Again we suppose a query sequence w and an arbitrary but fixed database sequence u.

Definition 1

Let $q \in \mathbb{N}$ and $k \geq 0$ be a threshold. A blast-hit is a pair (i, j) of indices such that $score_\sigma(\underbrace{u[i \dots i+q-1]}_{q\text{-gram in } u}, \underbrace{w[j \dots j+q-1]}_{q\text{-gram in } w}) \geq k$. $\square$

Stage 1: find blast hits

We now describe an algorithm to find blast-hits. This algorithm is iterated over all database sequences u.

(1) As in Fasta, the query sequence is preprocessed. While the preprocessing in Fasta only collects information for determining exact matches, the BLAST preprocessing takes into consideration non-exact matches. This is achieved by constructing the k-environment $Env_k(w)$ for the query sequence w. This is defined as follows:

$$\begin{aligned} Env_k(w) = \{(s,j) \mid\ & s \in \mathcal{A}^q, 1 \leq j \leq |w| - q + 1, \\ & score_\sigma(s, w[j \ldots j+q-1]) \geq k\} \end{aligned}$$

$Env_k(w)$ thus represents for each position in the query w the q-grams s achieving a score of k or greater when compared to the q-gram at that position. As s can be any q-gram, the environment does not depend on the database sequences.

Example 1

- Consider the alphabet $\mathcal{A} = \{\mathtt{a}, \mathtt{c}\}$ and the unit score function σ which assigns score 2 to any match and -1 to any mismatch.
- Suppose $k = 3$, $q = 3$ and consider the q-gram $x = aca$ contained in some query sequence at say position j.
- Then one would enumerate all $|\mathcal{A}|^q = 2^3 = 8$ q-words $s \in \{\mathtt{aaa}, \mathtt{aac}, \mathtt{aca}, \mathtt{acc}, \mathtt{caa}, \mathtt{cac}, \mathtt{cca}, \mathtt{ccc}\}$ and verify $score_\sigma(s, x) \geq k$:

	a	c	a	$\sum$
aaa	$+2$	-1	$+2$	**3**
aac	$+2$	-1	-1	0
aca	$+2$	$+2$	$+2$	**6**
acc	$+2$	$+2$	-1	**3**
caa	-1	-1	$+2$	0
cac	-1	-1	-1	-3
cca	-1	$+2$	$+2$	**3**
ccc	-1	$+2$	-1	0

So the pairs $(\mathtt{aaa}, j)$, $(\mathtt{aca}, j)$, $(\mathtt{acc}, j)$, $(\mathtt{cca}, j)$ are elements in $Env_k(w)$.

Stage 1: find blast hits

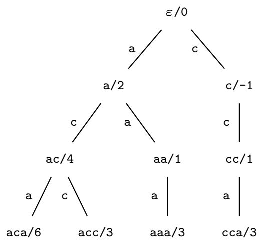
- An environment can be computed by first enumerating all q-grams in w.
- As seen in the previous example, common prefixes of these q-grams lead to the same positional scores.
- For example, for all q-grams beginning with a, we obtain a first score $+2$ when comparing them to aca.
- So it makes sense to only evaluate this score once and group the prefixes with a common first character in a subtree below some edge marked with that character.
- If we continue this further down, up to the last position, we obtain a trie, representing q-words.
- A trie is a tree whose edges are labeled by single characters, such that for each node α and each character a there is at most one edge from α labeled a.
- A trie represents a sequence set, say S, if the sequences in S can be read from the paths of the trie.

Stage 1: find blast hits

- On each trie-node α representing the sequence y of length q' for some $q' \leq q$, one maintains $score_\sigma(y, x[1 \dots q'])$ where x is the current q-gram of w.
- Each node of the trie at depth q with a score $\geq k$ represents a sequence from the k-environment of w, see Example 2 for an illustration.
- To speed up the computation, one can apply methods to prune the trie.
- This means that once a node is reached which cannot be on a path representing a sequence in the environment, then the entire subtree below the node need not be computed.
- Details of this approach are subject to an exercise.

Example 2

Consider the alphabet $\mathcal{A} = \{\mathrm{a}, \mathrm{c}\}$ and the unit score function σ which assigns score 2 to any match and -1 to any mismatch. Suppose $k = 3$, $q = 3$ and consider the q-gram $x = \mathrm{aca}$ contained in some query sequence, say at position j. The k-environment for aca is computed using this trie:



Each node is labeled by the string it represents and the score achieved when comparing the string to the corresponding prefix of aca; the leaves at depth q represent the strings in the environment. The following table shows the vector V and $Pos(s)$ which will be explained later.

$$\begin{array}{lcccccccc} & \mathrm{aaa} & \mathrm{aac} & \mathrm{aca} & \mathrm{acc} & \mathrm{caa} & \mathrm{cac} & \mathrm{cca} & \mathrm{ccc} \\ V = [& 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0] \\ Pos & j & & j & j & & & j & \end{array}$$

Stage 1: find blast hits

(2) Note that all strings $s \in \mathcal{A}^q$ satisfying $score_\sigma(s, w[j \dots j+q-1]) \geq k$ are of the same length q. We can therefore represent each $s \in \mathcal{A}^q$ as a unique integer $\overline{s}$ in the range from 0 to $r^q - 1$ where $r = |\mathcal{A}|$.
(3) $Env_k(w)$ is efficiently represented as follows:

- by a vector V of r^q bits such that $V[\overline{s}]$ is 1 iff $(s, j) \in Env_k(w)$ for some j.
- for each $s \in \mathcal{A}^q$ satisfying $V[\overline{s}] = 1$, we store the set $Pos(s) = \{j \mid (s, j) \in Env_k(w)\}$.

Example 2 shows V and $Pos(s)$ for a q-gram $x = aca$. To represent the sets $Pos(s)$ and to efficiently access them, we can apply techniques similar to those used for representing the sets $h_w(c)$ in the Fasta-algorithm.

Stage 1: find blast hits

(4) Each database sequence u is then processed by shifting a window of length q over it.

- Suppose that $i + q$ characters have been processed and let s be the current substring of length q in the window, i.e. $s = u[i \ldots i + q - 1]$.
- Then we compute $\overline{s}$ in constant amortized time and look up if $V[\overline{s}]$ is 1.
- If this is the case, then by definition of V, there is some j, such that $(s, j) \in Env_k(w)$.
- These positions j can be found in the set $j \in Pos(s)$ which can efficiently be enumerated.
- For all $j \in Pos(s)$, (i, j) is a blast hit.
- This is processed further in the next steps.

Stage 1: find blast hits

- Note that the search in the database sequence is entirely based on looking up values in tables V and Pos according to integer codes.
- No comparison of symbols according to a score function is necessary, as this has all been done in the preprocessing step.
- This is one of the main reasons why blast is so fast in determining the blast hits (but still takes a large share of the entire runtime of BLAST, see Table 1).
- The first three steps (1)–(3) only depend on the query sequence w.
- Hence they only have to be performed once, before processing the database sequences in step (4).

Example 3

Suppose that σ is the BLOSUM62-score function, shown earlier. Let $q = 4$, $k = 22$ and

$$w = \texttt{MGHLPLAWLSQ}$$

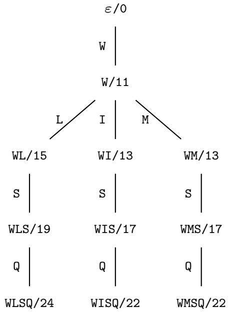
```
    MGHLPLAWLSQ
    12345678901
```

Then we obtain the following k-environment $Env_k(w)$, in which the score of each element is given as a subscript of the pair. Note that LPLA at position 4 is the only q-gram in w not generating an element in $Env_k(w)$.

$(\texttt{MGHL}, 1)_{23}$	$(\texttt{GHLP}, 2)_{25}$	$(\texttt{GHVP}, 2)_{22}$	$(\texttt{GHIP}, 2)_{23}$
$(\texttt{GHMP}, 2)_{23}$	$(\texttt{HLPL}, 3)_{23}$	$(\texttt{PLVW}, 5)_{22}$	$(\texttt{PLAW}, 5)_{26}$
$(\texttt{PLGW}, 5)_{22}$	$(\texttt{PLSW}, 5)_{23}$	$(\texttt{PLTW}, 5)_{22}$	$(\texttt{PLCW}, 5)_{22}$
$(\texttt{PVAW}, 5)_{23}$	$(\texttt{PIAW}, 5)_{24}$	$(\texttt{PFAW}, 5)_{22}$	$(\texttt{PMAW}, 5)_{24}$
$(\texttt{LAWL}, 6)_{23}$	$(\texttt{AWLS}, 7)_{23}$	$(\texttt{WLSQ}, 8)_{24}$	$(\texttt{WISQ}, 8)_{22}$
$(\texttt{WMSQ}, 8)_{22}$			

Fig. 1 shows the implicit trie generated when constructing $Env_k(\texttt{WLSQ})$.

Figure 1: The trie implicitly generated by the algorithm constructing $Env_k(\texttt{WLSQ})$ for $k = 22$ and $q = 4$. Each node of depth $q' \leq q$ is labeled by the q'-gram it represents and the sum of the scores when comparing it with $\texttt{WLSQ}[1 \ldots q']$. The leaves are of depth q and in the environment they appear with the position 8, which is the start position of WLSQ in the query sequence w.



Stage 1: find blast hits

- The size of the vector and the sets $Pos(s)$ grow exponentially with q and $1/k$.
- So these parameters should be selected carefully.
- For protein sequences, $q = 4$ and $k = 22$ (if σ is the BLOSUM62-score function) seem to be a reasonable choice.
- Once w has been preprocessed, u is processed in $O(|u| + z)$ time, where z is the number of blast hits.
- In an older version of BLAST used until about 1998, each blast hit triggered the processing of the next stage, namely the ungapped extensions.
- In the newer version of BLAST (BLAST 2.0 and later), an intermediate step was introduced, which is described next.

Stage 2: Identify pairs BLAST hits on the same diagonal

- In this stage, BLAST looks for pairs of blast hits on the same diagonal within some maximal distance, say δ, see the left side of Figure 2.
- Consider two hits (i, j) and (i', j') on the same diagonal d, i.e. $j - i = d = j' - i'$ and suppose that $i < i'$.
- Hence $i + h = i'$ for some $h > 0$.
- As the blast hits are computed in increasing order of the positions in the database sequence, (i, j) is computed before (i', j').
- From the assumption we can conclude $j' = i' + d = i + h + d = i + d + h = j + h$.
- Thus, with respect to the same diagonal, blast hits are computed in increasing order of the positions in both sequences and with the same distance h between i and i' as well as j and j'.

Stage 2: Identify pairs BLAST hits on the same diagonal

- Hence, for each diagonal d it suffices to keep track of the last index $f(d) = i$ such that there is a blast hit (i, j) satisfying $d = j - i$.
- For any new blast hit (i', j') one computes the diagonal index $d = j' - i'$.
- If $f(d)$ is defined, then $f(d) < i'$ and one checks $i' - f(d) \leq \delta$, where δ is the *maximal distance on diagonal*-parameter.
- If $i' - f(d) \leq \delta$ holds, then the distance of the two blast hits

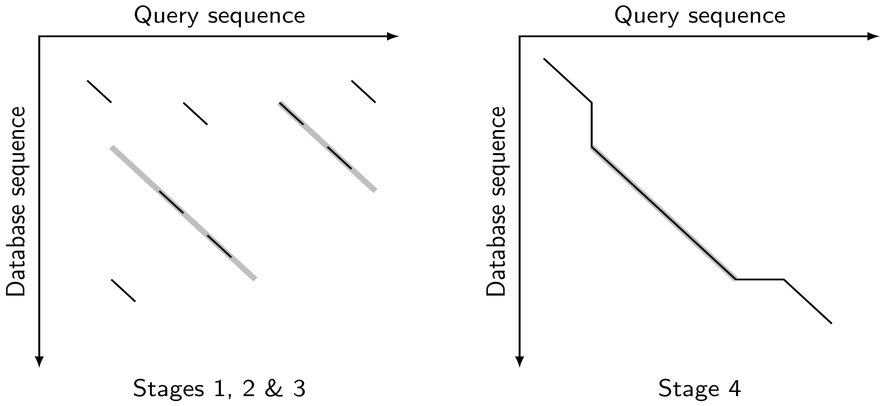
  $$(\underbrace{f(d)}_{i}, \underbrace{d + f(d)}_{j=d+i}) \text{ and } (i', j') \text{ on diagonal } d$$

  is at most δ and one has found a pair of relevant blast hits which is further processed.
- In any case (whether $f(d)$ is defined or not), one updates $f(d) \leftarrow i'$.

Stage 3: Ungapped extensions

- Suppose we have a pair of blast hits on the same diagonal within some maximal distance δ.
- To determine if the pair of blast hits occurs in a region of pairwise similarity, one computes the mid point between the two hits and uses this as the starting point for the ungapped left-extension and the right-extension method described below.
- The sensitivity of the extensions depends on a “drop-off” parameter $X_d > 0$.
- An appropriate choice of parameters q, k, δ, X_d leads to a method which is fast and sensitive.

Figure 2: Illustration of the first four stages of the BLAST algorithm. In stage 1 blast hits are identified, shown as short black lines. In stage 2 one identifies pairs of hits occurring near each other and on the same diagonal. These are subject to ungapped extensions in stage 3, with the result shown as a longer grey line. In this example, the longer of the two ungapped extensions scores above some threshold and is passed on to stage 4, where it is used as a starting point for constructing a high-scoring gapped alignment.



Stage 3: Ungapped extensions

- We here describe the ungapped extension for a mid-point (i, j) between two blast hits.
- Let $m = |u|$ and $n = |w|$.
- For the extension to the left, the sequences $u[1 \ldots i-1]$ and $w[1 \ldots j-1]$ are compared from right to left.
- For the extension to the right, the sequences $u[i \ldots m]$ and $w[j \ldots n]$ are compared from left to right.
- Each pair of characters $(u[i-\ell], w[j-\ell])$, and $(u[i+r], w[j+r])$, for $1 \leq \ell \leq \min\{i-1, j-1\}$ and $0 \leq r \leq \min\{m-i, n-j\}$, delivers a score according to the score function σ.

Stage 3: Ungapped extensions

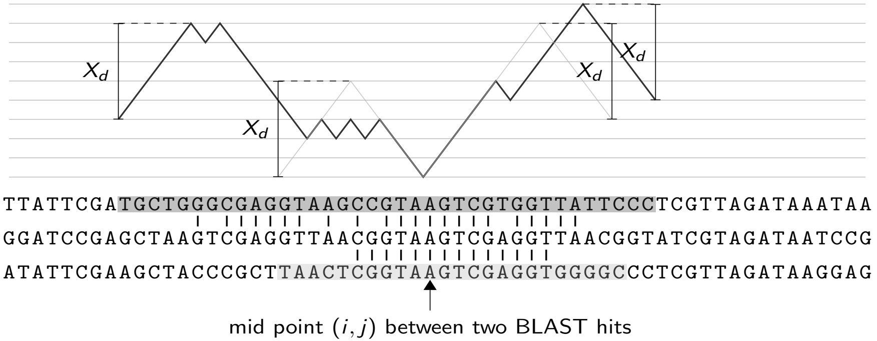
- For both extensions, the scores are accumulated and the maximum value $X_{\max}$ reached during the extension is kept track of.
- As soon as a score smaller than $X_{\max} - X_d$ is reached, the extension is stopped, see Algorithm 1 show the pseudocode of a function computing the length of the extension to the right.
- The pair of sequences delivered by the extensions to the left and to the right is called maximum segment pair (MSP, see Fig. 3).
- For any such MSP, a significance score is computed.
- If this is better than some predefined significance threshold, then the MSP is processed by the next stage.

Algorithm 1 (BLAST hit extension)

A right-extension function starting at coordinates (i, j) on the middle between 2 blast hits on the same diagonal for the database seq. u and the query seq. w. σ is the score function and $X_d > 0$ is the drop-off param.

```
function $extendhitright(u, w, \sigma, X_d, i, j)$
$(X_{\max}, accum\_score, r) \leftarrow (0, 0, 0)$
while $i + r \leq |u|$ and $j + r \leq |w|$ do
    $accum\_score \leftarrow accum\_score + \sigma(u[i + r], w[j + r])$
    if $accum\_score < X_{\max} - X_d$ then
        break
    end if
    if $accum\_score > X_{\max}$ then
        $X_{\max} \leftarrow accum\_score$
    end if
    $r \leftarrow r + 1$
end while
return $r - 1$
```

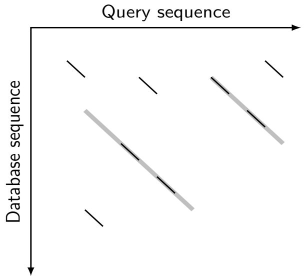
Figure 3: Illustration of the blast hit extension strategy for a midpoint marked by the arrow. Match score is 1 and mismatch score is -1. At the bottom, two queries are shown above and below the database sequence in the middle, respectively. At the top, the accumulated score obtained when comparing the dark grey and light grey part of the query sequence against the corresponding part of the database sequence is shown as a thick and a thin line, respectively. The extension stops when the accumulated score drops below $X_{\max} - X_d$, which happens when the boundaries of the query substring in the grey boxes is reached.



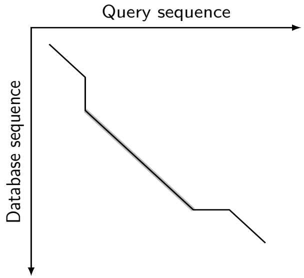
X_d
X_d
X_d
X_d
TTATTCGATGCTGGGCGAGGTAAGCCGTAAGTCGTGGTTATTCCCTCGTTAGATAAATAA
GGATCCGAGCTAAGTCGAGGTTAACGGTAAGTCGAGGTTAACGGTATCGTAGATAATCCG
ATATTCGAAGCTACCCGCTTAACTCGGTAAGTCGAGGTGGGGCCCTCGTTAGATAAGGAG
mid point (i, j) between two BLAST hits

Stage 4: Gapped extension

- In the fourth stage, a gapped alignment is performed to determine if the high scoring ungapped region forms part of a larger, higher scoring alignment.
- The following illustration (right side) illustrates an example of a gapped extension.
- The single, high-scoring ungapped extension identified in Stage 3 is considered as the basis of a gapped alignment, and the black line shows the alignment identified through this process.



Stages 1, 2 & 3



Stage 4

Stage 4: Gapped extension

- The gapped alignment algorithm used by BLAST differs from Smith-Waterman local alignment.
- Rather than exhaustively computing all possible paths between the sequences, the gapped scheme explores only insertions and deletions that augment the high-scoring ungapped alignment.
- Therefore, this step begins by identifying a seed point that lies within a high-scoring portion of the ungapped region.
- After this, a gapped alignment is attempted, using affine gap scores.
- The method is very similar to the computation of optimal global alignment for affine gap costs.

Stage 4: Gapped extension

- A gapped alignment stops when the score falls below a value determined by another dropoff parameter, Y_d.
- This parameter controls the sensitivity and speed trade-off: the higher the value of Y_d, the greater the alignment sensitivity but the slower the search process.
- If the resulting gapped alignment scores more than a minimum value (which is determined from an external E-value cutoff parameter) it is passed on to the fifth and final stage of BLAST.
- On average, according to [Cameron et al., 2004], less than 0.01 percent of the gapped alignments performed during the third stage score above the default E-value cutoff of 10.

Stage 5: Collect traceback information and display alignments

- BLAST uses Karlin-Altschul statistics [Altschul and Gish, 1996, Karlin and Altschul, 1990] to report the statistical significance of each gapped local alignment as an E-value.
- The E-value is the number of local alignments with at least the same score that one expects when comparing a random query sequence with typical amino-acid composition against a random database sequence.
- Of course, the length of the query sequence and the size of the database sequence are taken into consideration.
- There are three steps to determine the statistical significance of a gapped local alignment:

Stage 5: Collect traceback information and display alignments

1. A nominal score S is determined for each local alignment based on a scoring matrix (BLOSUM [Henikoff and Henikoff, 1993] or PAM [Dayhoff et al., 1978]) and gap scores.
2. The nominal score is converted to a normalized score
$$S' = \frac{\lambda S - \ln K}{\ln 2}$$
where λ and K are precomputed by random simulation for each scoring matrix and gap penalty combination. Scores in this normalized form are expressed in bits and are comparable across different scoring schemes.
3. The normalized score is converted into an E-value E as follows:
$$E = \frac{Q}{2^{S'}}$$
where $Q \approx mn$ is the size of the search space, where m and n are the lengths of the query and the database sequence, respectively.

Stage 5: Collect traceback information and display alignments

- The resulting value of E is reported to the user.
- As the equations above are invertible, one can determine the minimum nominal score required to achieve a specific E-value.
- This approach is used by BLAST to determine the cutoff parameter from the user-specified E-value.

Altschul, S. and Gish, W. (1996).
Local alignment statistics.
Methods Enzymol, 266:460–80.

Altschul, S., Gish, W., Miller, W., Myers, E., and Lipman, D. (1990).
A Basic Local Alignment Search Tool.
JMB, 215:403–410.

Cameron, M., Williams, H. E., and Cannane, A. (2004).
Improved gapped alignment in blast.
IEEE/ACM Trans. Comput. Biol. Bioinformatics, 1(3):116–129.

Dayhoff, M., Schwartz, R., and Orcutt, B. (1978).
A Model of Evolutionary Change in Proteins. Matrices for Detecting Distant Relationships.
Atlas of Protein Sequence and Structure, 5:345–358.

Henikoff, S. and Henikoff, J. (1993).
Performance Evaluation of Amino Acid Substitution Matrices.
Proteins: Structure, Function, and Genetics, 17:49–61.

Karlin, S. and Altschul, S. (1990).
Methods for Assessing the Statistical Significance of Molecular Sequence Features by using General Scoring Schemes.
PNAS, 87:2264–2268.