Algorithms for Bioinformatics

2018/2019

Searching for Similar Sequences in Databases

Pedro G. Ferreira
[d**CC**] @ Faculty of Sciences University of Porto

Outline

- Finding similar sequences based on sequence alignment algorithms.
- The need for more efficient algorithms to search in larger databases of sequences.
- BLAST concepts and programs.
- K-mer indexing scheme.
- Implementing a simple version of BLAST.
- Exercises.

Sequence Similarity

- Sequence alignment, either global or local, provides a measure or score of the edit distance between the optimal alignments of two sequences.

- A typical situation in biological research is to have a sequence for which we do not have yet any information. Bioinformatics can provide an annotation for the sequence, i.e. to find information about the function of this sequence.

- Using information from similar sequences to infer function of our query sequence is one of the most widely used strategies.

- This requires scanning large databases of sequences and compare the query sequence against all the sequences in the database, selecting those with higher degree of similarity.

Exercises

- Write a function that finds the most similar sequence to a query sequence. The function should use the local alignment algorithm developed previously. It receives as input:
  - *Query sequence (query)*
  - *List of sequences (list_of_seqs)*
  - *Substitution matrix (sm)*
  - *Gap penalty (g)*

It should return the alignment with the sequence of best score.

Exercises


```
1. def align_query (query, ls, ms, g):
2.     bestScore = -1
3.     bestSeq = None
4.     bestAl = None
5.     for seq in ls:
6.         al = smith_Waterman(query, seq, ms, g)
7.         if al[2] > bestScore:
8.             bestScore = al[2]
9.             bestSeq = seq
10.            bestAl = al
11.    bestAlin = recover_align_local(bestAl[0], bestAl[1], query, bestSeq)
12.    return bestAlin, bestScore
13.
```

Sequence Similarity

- The pairwise sequence alignment algorithms seem to be the obvious choice to find similar sequences in databases. They are very accurate but are not efficient enough when it is necessary to scan very large sets of sequences.

- Pairwise alignments have quadratic complexity. And they need to fill two matrices, that for very long sequences, can be memory consuming. When databases reach hundreds of thousands or even several millions of sequences this approach is not practical.

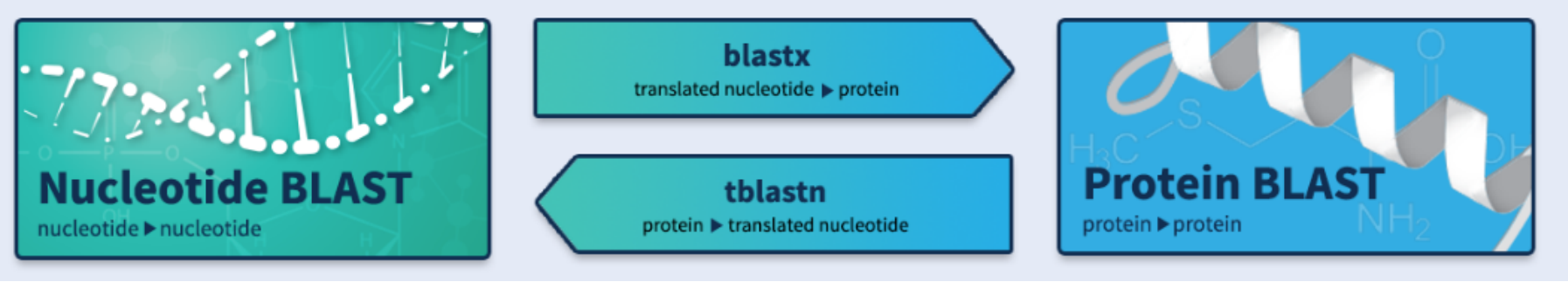
- Moreover they require as input a number of parameters that can widely impact the final results, i.e. the final set of similar sequences.

- We need more suitable algorithms, eventually based on heuristics to speed-up the search and to handle more cases of lowly sequence similarity.

BLAST

- The BLAST (Basic Local Alignment Search Tool) program was developed to find regions of local similarity between sequences. The program compares nucleotide or protein sequences to sequence databases and calculates the statistical significance of matches.

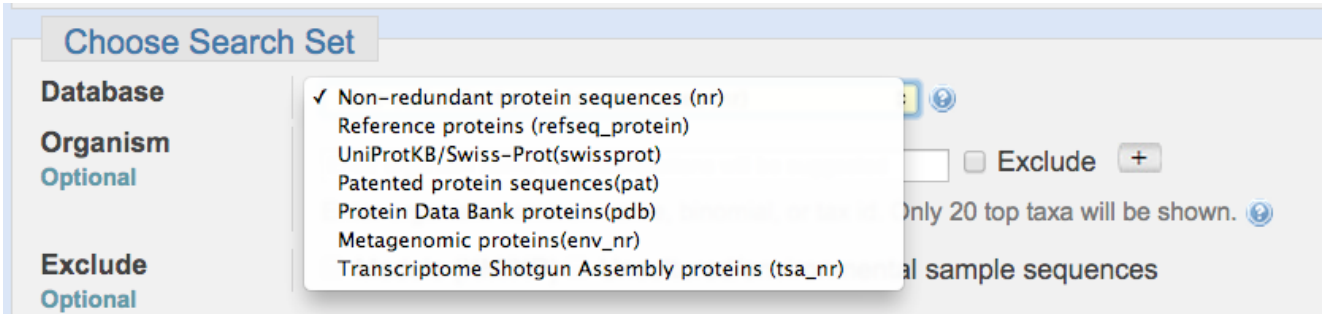
- It is actually a set of programs:
  - BLASTN for nucleotide sequences
  - BLASTSP, BLASTX, TBLASTN, TBLASTX for protein sequences.



https://blast.ncbi.nlm.nih.gov/Blast.cgi

BLAST Programs

Program	Query Sequence	Database
Blastn	Nucleic acid	Nucleic acid
Blastp	Protein	Protein
Blastx	Translated nucleotide	Protein
Blastn	Protein	Translated Nucleic acid
Blastx	Translated nucleotide	Translated nucleotide All frame translation



BLAST

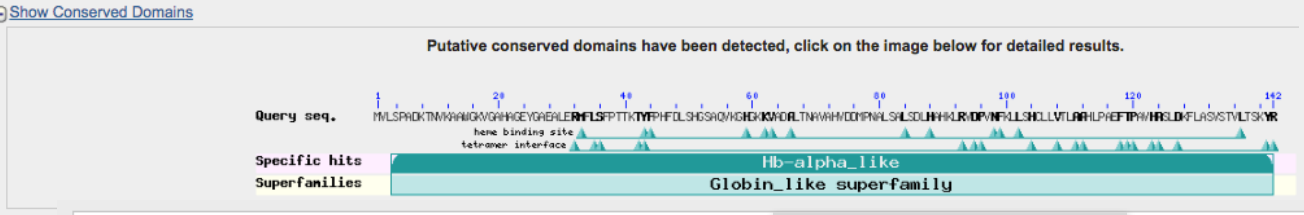
- BLAST can be used to infer functional and evolutionary relationships between sequences as well as help to identify members of gene families.

- Questions that BLAST may help to solve:
  i) From which species does the DNA I have sequenced comes from?
  ii) Which species (e.g. bacterial species) have a protein that shares an evolutionary history with my protein?
  iii) What other genes encode for proteins with similar sequence and structure similar to the one I have just sequenced?

- Statistical score may help to determine which alignments are statistically significant, i.e. do not occur by chance.

BLASTP

Blast the **protein** sequence from HBA_HUMAN.



Graphical output

Detected motifs and domains

Sequences producing significant alignments:

Select: All None Selected:0

Alignments Download GenPept Graphics Distance tree of results Multiple alignment

Description	Max score	Total score	Query cover	E value	Ident	Accession
hypothetical protein [Klebsiella pneumoniae]	288	288				
hemoglobin alpha 2 [synthetic construct]	287	287	100%	5e-98	100%	AAX29522.1
hypothetical protein [Escherichia coli]	287	287	100%	6e-98	100%	WP_108997664.1
MULTISPECIES: hypothetical protein [Enterobacteriaceae]	286	286	100%	7e-98	100%	WP_108961785.1
Chain B, A Cis-Proline In Alpha-Hemoglobin Stabilizing Protein Directs The Structural Reorganization Of Alpha-Hemoglobin	286	286	100%	9e-98	100%	3IA3_B
mutant hemoglobin alpha 2 globin chain [Homo sapiens]	286	286	100%	1e-97	99%	AKZ66543.1
MULTISPECIES: hypothetical protein [Enterobacteriaceae]	286	286	100%	1e-97	100%	WP_109027992.1
mutant hemoglobin subunit alpha 2 [Homo sapiens]	286	286	100%	1e-97	99%	AXY55002.1

Statistical score

Score

Percentage of the query sequence that is aligned

Identity percentage

Significance of results

- As described by the NCBI:

  "The Expect value (E) is a parameter that describes the number of hits one can "expect" to see by chance when searching a database of a particular size. It decreases exponentially as the Score (S) of the match increases."

- An E-value of 1 corresponds to an expected match by chance of one sequence with a similar score in the target database.

- The lower the E-value the more significant the result, i.e. less likely to have occurred just by chance. A typical significant E-value can be > 1e-5.

- Depending on the goal of the analysis the remaining parameters should also be taken into account. For instance, when searching for homologs one may expect > 90% sequence similarity and > 90% query coverage.

BLAST Concepts

- **Query sequence** = sequence that we want to know more about.
- **Target sequence** = sequence in the database that was matched by our query sequence.
- **D** is the database of sequences to search for.

- Aligning sequences against a database:

  - Setup: Given a database D of sequences and a query sequence Q, retrieve sequences in D similar to Q.

  - How to define similar?

obtain identity percentage, e.g. ABCDE ABXDE, id% = 80%

  - Rational:

if we are rejecting sequences in D with id% < X then focus on sequences with common long stretches and reject the others.

Aligning Sequences Against Database

K-mer Indexing

- **D Pre-processing**: Scan every word of length K in S_i in D, keep its location in lookup table H.
- **Q scan**: scan every k-word in Q and get its location in H.
- What is an hash function?

Aligning Sequences Against Database

K-mer Indexing

- **D Pre-processing**: Scan every word of length K in S_i in D, keep its location in lookup table H.
- **Q scan**: scan every k-word in Q and get its location in H.
- What is an hash function?

  Function that maps values to a data structure of fixed size. The function receives a value and returns an hash value.
  Ideally, the lookup function has O(1) complexity (O(n) in the worst scenario).

Aligning Sequences Against Database

- Blast is an heuristic approach based on the idea of K-indexing
- Algorithm:

1. **Seeding**: find common subwords between query Q and database sequences D → seeds
2. **Extension**: starting from seeds, extend alignment in both directions → *high-scoring segment pairs* (HSP)
3. **Evaluation**: assess the statistical significance of each HSP

Aligning Sequences Against Database

- Seeding
  - Window scanning of Q to generate K-words: L1-set
  - Find neighbourhood words for each k-word until threshold T: L2-set
  - Merge L = L1 U L2
  - Look into H table where L words occur: *seeds*

- Seeding Improvements
  - Low complexity regions can cause spurious hits:
  - Filter out low complexity in your query
  - Filter most over-represented items in your database

Aligning Sequences Against Database

- Extension
  - Extend until cumulative score drops below minimum score X
  - Original BLAST. Each hit is extended in both directions until the running alignments score has dropped more than X below the maximum score yet attained.
  - BLAST 2.0. allows alignments with gaps. If two non-overlapping hits are found within distance A of one another on the same diagonal, then merge the hits into an alignment and extend the alignment in both directions until the running alignments score has dropped more than X below the maximum score yet attained. DP can be used for optimal alignment.

Aligning Sequences Against Database

- Evaluation
  - P-value: Probability that the HSP was generated as a chance alignment.
  - Score: -log of the probability
  - E-value: expected number of such alignments given database

From https://blast.ncbi.nlm.nih.gov/

"The Expect value (E) is a parameter that describes the number of hits one can "expect" to see by chance when searching a database of a particular size. It decreases exponentially as the Score (S) of the match increases."

"The lower the E-value, or the closer it is to zero, the more "significant" the match is. However, keep in mind that virtually identical short alignments have relatively high E values. This is because the calculation of the E value takes into account the length of the query sequence."

Aligning Sequences Against Database

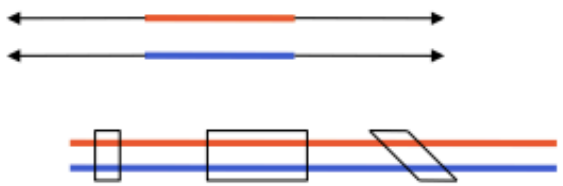
BLAST – no magic involved

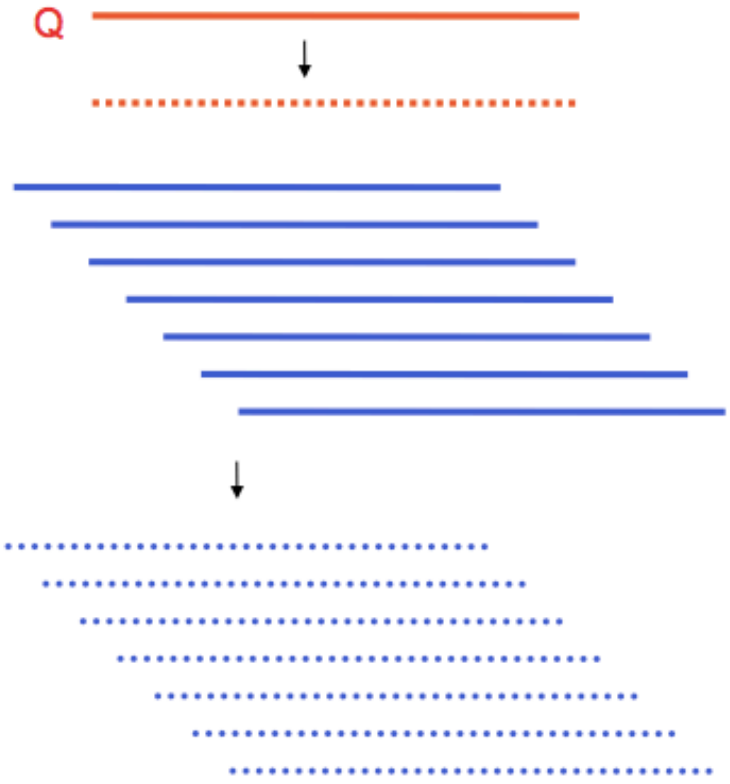
Concept:

1. Break query sequence (Q) and database into short 'words' of length W and create hash table

2. Seek matches between fragments – exceeding score T. Results in collection of high-scoring pairs (HSP).

3. Extend these matches as far as possible, until you get minus scores





Åsa Björklund credit

Adapted from http://www.scienceplug.com/

Aligning Sequences Against Database

Main steps of Blast can be summarised as:

1. Remove regions of low complexity (e.g. sequence repeats) from query sequence.

2. Obtain all possible “words” of size **w** (a parameter of the algorithm), i.e. sub-sequences of length w occurring in the query sequence;

3. For each word from the previous step, compile the list of all possible words of size w that can be defined in the allowable alphabet, whose alignment score (with no gaps) is higher than a threshold **T** (parameter of the algorithm);

4. Search in all sequences from the database, all occurrences of the words collected in the last step, which represent matches (**hits or seeds**) of size **w** between the query and one of the database sequences;

5. Extend all hits from the last step, in both directions, while the score follows a given criterion (typically, the criterion is dependent on the size of the extension);

6. Select the alignments in the previous step with highest scores, normalized for its size (these are named the **high-scoring pairs-HSPs**).

Implementing Simple BLAST

Perform the hashing of the query sequence, i.e. given a parameter *w* for the size of the word, pre-process the query sequence to identify all words of size w keeping the positions where they occur.

Use a dictionary to keep this information and create a function *build_map* for this effect (input: query and w).

Implementing Simple BLAST

Hashing the query sequence.

```
1. def build_map (query, w):
2.     res = {}
3.     for i in range(len(query)-w+1):
4.         subseq = query[i:i+w]
5.         if subseq in res:
6.             res[subseq].append(i)
7.         else:
8.             res[subseq] = [i]
9.     return res
10.
```

Implementing Simple BLAST

- In this class we will develop a simple BLAST implementation. We will not use substitution matrices.
- We will consider a score of 1 for a match and 0 for mismatch without gaps.
- Only perfect hits are considered, i.e. the threshold *T* is equal to *w*.

Develop a function that given a sequence (target sequence from DB), the map of locations of words of length w in the query sequence (map) and a threshold parameter (t = w) find all the hits between the words in the query and the target sequence.
The result is a list of hits = <match index in query, match index in target>

Implementing Simple BLAST

Develop a function that given a sequence (target), the map of locations of words of length w in the query sequence (map) and a threshold parameter (t = w) find all the hits between the words in the query and the target sequence. The result is a list of hits = <match index in query, match index in target>

```
1. def get_hits (seq, m, w):
2.     res = []  # list of tuples
3.     for i in range(len(seq)-w+1):
4.         subseq = seq[i:i+w]
5.         if subseq in m:
6.             l = m[subseq]
7.             for ind in l:
8.                 res.append( (ind,i) )
9.     return res
10.
```

Implementing Simple BLAST

Extend the hits found in the previous function. Extend in both directions while:
- contribution to the increase in the score is >= 1/2 of the positions in the extension.

The result is a tuple = <index of align. on query, index of align. on sequence, size of align., score> where score is the number of matching characters.

```
1. def extends_hit (seq, hit, query, w):
2.     stq, sts = hit[0], hit[1]
3.     ## move forward
4.     matfw = 0
5.     k=0
6.     bestk = 0
7.     while 2*matfw >= k and stq+w+k < len(query) and sts+w+k < len(seq):
8.         if query[stq+w+k] == seq[sts+w+k]:
9.             matfw+=1
10.            bestk = k+1
11.        k += 1
12.    size = w + bestk
13.    ## move backwards
14.    k = 0
15.    matbw = 0
16.    bestk = 0
17.    while 2*matbw >= k and stq > k and sts > k:
18.        if query[stq-k-1] == seq[sts-k-1]:
19.            matbw+=1
20.            bestk = k+1
21.        k+=1
22.    size += bestk
23.
24.    return (stq-bestk, sts-bestk, size, w+matfw+matbw)
25.
```

Implementing Simple BLAST

Identify the best alignment between the query sequence and a given sequence:

- identify all hits of size w and extend all those hits.
- Select the hit with the best overall score (highest number of matches).
- The result is a tuple with the same format as the one returned by the `extends_hit` function.

```
1. def hit_best_score(seq, query, m, w):
2.     hits = get_hits(seq, m, w)
3.     bestScore = -1.0
4.     best = ()
5.     for h in hits:
6.         ext = extends_hit(seq, h, query, w)
7.         score = ext[3]
8.         if score > bestScore or (score== bestScore and ext[2] < best[2]):
9.             bestScore = score
10.            best = ext
11.    return best
12.
```

Implementing Simple BLAST

Apply the previous function to compare the query sequences with all the sequences in the database db.

- Find the best overall alignment of the query with the sequence in db.
- The result is a tuple as in the same function adding in the last position the index of the sequence in db with the best alignment.

```
1. def best_alignment (db, query, w):
2.     m = build_map(query, w)
3.     bestScore = -1.0
4.     res = (0,0,0,0,0)
5.     for k in range(0,len(db)):
6.         bestSeq = hit_best_score(db[k], query, m, w)
7.         if bestSeq != ():
8.             score = bestSeq[3]
9.             if score > bestScore or (score== bestScore and bestSeq[2] < res[2]):
10.                bestScore = score
11.                res = bestSeq[0], bestSeq[1], bestSeq[2], bestSeq[3], k
12.    if bestScore < 0: return ()
13.    else: return res
14.
```

Conclusions

- Quantifying the similarity between two biological sequences is a task of central importance on bioinformatics.

- Searching in databases for sequences of high similarity of our query sequence is a very common task. It helps to identify the homology of the sequence.

- The suite of BLAST programs it is probably the most widely used program in bioinformatics. It is optimized for different types of query and target sequences.

Exercises

- Complete the code in the MyBlast class.

Take home exercise:

- Develop a variation of the `getHits` function that allows for mismatches. It should allow at most 1 mismatch and return all hits that have w or w-1 matches.