Annals of West University of Timişoara, ser. Biology, 2018, vol. 21 (2), pp.185-192

ALGORITHMS FOR GENOMIC SIMILARITY SEARCH: A REVIEW

Done STOJANOV
Faculty of Computer Science, University Goce Delcev, Krste Misirkov nn – Štip, Republic of Macedonia
**Corresponding author e-mail: done.stojanov@ugd.edu.mk*
Received 29 June 2018; accepted 10 December 2018

ABSTRACT
In this paper we consider the properties of the most popular algorithms for genomic similarity search. All these methods use some type of heuristics to improve the speed of search that has negative impact upon the accuracy. Some of these methods split the content into small pieces of data, some of them use hashing, while some of them utilize the computational benefits of suffix tree. In many cases the output depends of the user-proved parameters that define the sensitivity and the range of search.
KEY WORDS: *heuristics, algorithms, genomic, similarity, search.*

INTRODUCTION

Genomic similarity search has been widely explored scientific topic nowadays and a lot of algorithms and software tools (desktop and web applications) have been developed. In spite of the different concepts that were used, two major groups of algorithms evolved. The first group or deterministic algorithms are based on predefined metrics. Deterministic algorithms are based on dynamic programming and they report the best or the highest score solution. However due to the fact that they operate on predefined metrics they require too much memory and time ($O(n \times m)$ time and memory) that may limit the real-time application of these algorithms. In practice they can be applied to short fragments or prokaryotic samples.

All older algorithms for genomic similarity search, such as: Needleman-Wunsch (Needleman et al., 1970), Sellers (Sellers, 1974), Smith-Waterman (Smith et al., 1981), Waterman-Eggert (Waterman et al., 1987) and Ficket (Ficket, 1984) are deterministic.

Needleman-Wunsch (Needleman et al., 1970) is the oldest known algorithm for this purpose and it performs global alignment on two samples applying dynamic programming metrics. The algorithm of Sellers (Sellers, 1974) can be seen as an alternative of Needleman-Wunsch. Both algorithms work for the same purpose, but Sellers instead of maximizing the score of the alignment, minimizes the distance between the samples.

On the other hand Smith-Waterman (Smith et al., 1981) performs local alignment, i.e. it detects the most similar fragments between two samples, performing gapped alignment to these regions. Dynamic programming scoring scheme is also used in Smith-Waterman.

Instead of searching for one best solution, Waterman-Eggert (Waterman et al., 1987) looks for the k best or highest scoring alignments. At first dynamic programming matrix of Smith-Waterman type is computed and the algorithm looks for the highest scoring alignment in the matrix. After printing it, the cells of the highest scoring alignment in the matrix are set up to

0 and the algorithm looks for the next highest scoring alignment. Waterman-Eggert applies this procedure until the k best or highest scoring solutions are reported.

The concept of diagonal alignment reduces the space and time complexity down to $O(k \times n), k < m$. The problem was discussed and solved by Ficket (Ficket, 1984). Ficket proved that the best or the highest scoring solution converges to the main diagonal in dynamic programming matrix and therefore cells at distance $d_{i,j} > D$ do not have to be computed, since it not likely that the path of pointers that defines the best solution will go through any of those cells. However the inquiry regards the value of the parameter D (the size of the diagonal range) remains that limits the application of this algorithm only to very similar samples.

As we mentioned before, due to the unfavorable time complexity, all deterministic algorithms have limited application only to small-size samples. On the other hand algorithms which are based on some type of heuristics, such as: FASTA (Pearson et al., 1988), BLAST (Altschul et al., 1997), Pattern Hunter (Ma et al., 2002), BLAT (Kent, 2002), FLASH (Califano et al., 1993), YASS (Noé et al., 2005), MUMmer (Delcher et al., 2003) and AVID (Bray et al., 2003) can solve this problem more efficiently. The high processing power of these algorithms is due to the data pre-processing or the cuts in the space where they search for solution, what is a common practice in all heuristic algorithms. Heuristic algorithms, the way they operate and their properties are topics of this paper.

MATERIALS AND METHODS

FASTP is one of the oldest methods for massive genomic database search, which was afterwards upgraded to FASTA (Pearson et al., 1988). These methods (FASTP and FASTA) differ in the type of data they operate on. In spite of FASTP which works only with protein data, FASTA is applicable for protein and nucleotide data. Before explaining how FASTA works, we should emphasize that we are taking about algorithm that runs in several phases. In the first phase, the query is split into short words of fixed size k.For nucleotide data $k = 6$(for protein $k = 2$). Later, during the search, hits of these words in the database are identified. Note that each hit is tracked by tuple (i, j) such as i stands for the position of the short word in the query, while j stands for the position of the hit in the database. Consecutive hits form visually diagonals or extended hits and they are best known as diagonal extensions. Each of these extensions is properly scored and only the best scoring diagonal extensions are considered as a part of the alignment. For instance, PAM250 matrix is used for this purpose when the algorithm operates on protein data. Significant and neighboring diagonal extensions are connected together by applying the Smith-Waterman gapped algorithm and that is done by inserting gaps in the space that separates the neighboring diagonal extensions.

This procedure is graphically illustrated on Fig. 1. Each line section regardless of its thinness represents diagonal extension and as we said before each one of these diagonal extensions is made of several short consecutive hits that are joined together, Fig. 1. Not all diagonal extensions are part of the final solution. The less significant diagonal extensions (shorter and thinner line sections on Fig. 1) are rejected, while the high-scoring (longer and bold line sections on Fig. 1) are connected together what results into an alignment. The connection is made by inserting gaps in the space between the neighboring diagonal extensions (dashed segments on Fig. 1) applying the Smith-Waterman algorithm.

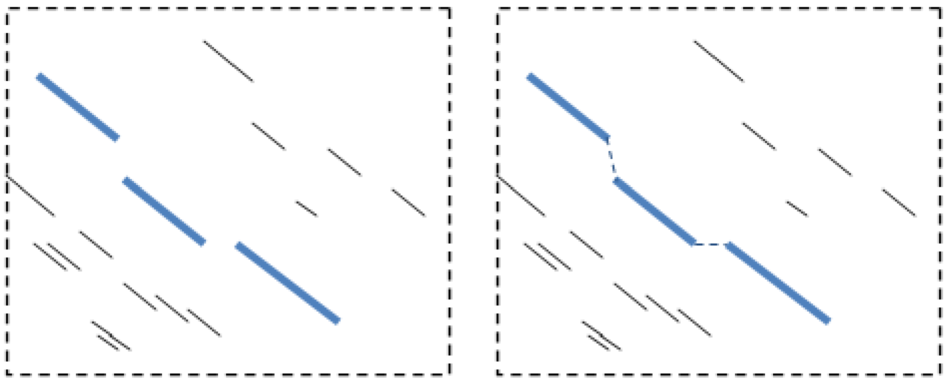

FIG. 1. Graphic illustration of FASTA

BLAST (Basic Alignment Search Tool) (Altschul et al., 1997) is another useful approach for genomic database search. The first version was able to perform only un-gapped alignments, while the later upgrade (PSI-BLAST) performed gapped alignments. Theoretically there is no conceptual distinction between FASTA and BLAST, because both search for hits of short words in massive data, which is a widely explored concept for searching genomic database upon query. However we should note that in terms of the speed, BLAST runs faster than FASTA and it can be applied to protein and nucleotide data.

They way BLAST operates must be divided in several phases. When it comes to protein data, a list of query-extracted words of fixed size w is construced at the beginning. This list is afterwards extended with words that are not part of the query but for which the score of the un-gapped alignment with query-extracted words is greater than T. PAM or BLOSUM substitution matrix is used for this purpose. In the next stage, perfect hits of these words in the database are identified, such as the program tracks the position of every single hit. These hits are referred as seeds and they are extended in the both directions (to the left and to the right). The extension goes on until the score of the alignment increases. Same as FASTA, BLAST considers only the most significant extensions for which the score of the alignment is greater than S. Note that the parameters: w, T and S are provided by the user and therefore different solutions are expected upon different combinations of these parameters.

When it comes to nucleotide data, the query has to be also split into list of words of 12 bases which are afterwards searched in the database, using two-bit encoding scheme (00, 01, 10, and 11) per base. Nowadays there are many available BLAST implementations, such as: BLASTN (works with nucleotide data), BLASTP (works with protein data) and BLASTX (compares nucleotide query to protein database).

The search for perfect hits has been the main drawback of FASTA and BLAST. Furthermore, BLAST search sensitivity depends of the parameter w or the length of the words that are searched in the database. For high w many hits remain undetected and therefore distant homologies cannot be found. On the other hand, if w is too small, the search slows down.

Pattern Hunter (Ma et al., 2002) is step above FASTA and BLAST, because it searches for imperfect hits, based on model for hits. This means that words with at least $k, k < w$ equal nucleotides are reported as hits, regardless of the order of appearance (consecutive or non-consecutive) what will improve the sensitivity of search. According (Ma et al., 2002) for region of similarity of 70% and $k = 11$, the sensitivity of search is maximal for the model: 111010010100110111, such as 1 stands for equal elements, while 0 stands for unequal elements. Database words that satisfy this or other adopted model are regarded as hits and they are extended in both directions in the final stage. Different solutions are found for different models.

Unlike BLAST which is applicable for distant and closely-related samples, BLAT (Kent, 2002) works only with closely-related samples. BLAT runs faster than BLAST, but the sensitivity of search is reduced. BLAT searches for perfect or near-perfect hits without computing statistical significance.

BLAT runs in two phases: search and alignment phase. In the first phase it searches for perfect, near-perfect or multiple-perfect hits between query and genomic database, upon which an alignment is constructed in the second phase.

Unlike BLAST and FASTA, BLAT indexes the database. Non-overlapping database words of size k are indexed, such as k ranges between 8 and 16 for nucleotide data and between 3 and 7 for protein data. Database words are compared to query words in order to detect: perfect, near-perfect or multiple-perfect hits. According Kent (Kent, 2002) near-perfect hits include one mismatch, while multiple-perfect hits include N perfect hits at mutual distance that does not exceed W.

The expected number of perfect hits equals $F = (q - k + 1) \times \left(\frac{D}{k}\right) + (\frac{1}{A})^k$, the expected number of near-perfect hits equals $F = (q - k + 1) \times \left(\frac{D}{k}\right) \times (k \times \left(\frac{1}{A}\right)^{k-1} \times \left(1 - \frac{1}{A}\right) + \left(\frac{1}{A}\right)^k)$ and the expected number of multiple-perfect hits equals $F_N = F_1 \times S$, such as $F_1 = (q - k + 1) \times \left(\frac{D}{k}\right) \times (\frac{1}{A})^k$, $S = 1 - (1 - \left(\frac{1}{A}\right)^k)^{W/k}$. In previous equations q stands for the length of the query, D is the size of the database, W is the maximum allowed distance between perfect hits in multiple-perfect hits and A stands for the size of the alphabet (for nucleotide data $A = 4$ and for protein data $A = 20$).

FLASH (Califano et al., 1993) detects database motifs that resemble to the query. At the beginning, the program compiles two separate lists that contain information about all overlapping words w of k elements in the target sequence and the query. This information is presented in form of pairs (i, w_i) such as i is the position of the word w_i in the target/query, while w_i is the word which is tracked. Motifs from the target sequence are compared to the query. Common words are identified at first and the differences in positions in the motif and the query are computed afterwards. The motif for which the number of equal differences is the highest is the motif which is most similar (homologous) to the query.

We will illustrate how FLASH works to the target t=ACTGATTG and the query q=GAGTG. For $k = 2$ the program compiles two lists Lt and Lq, such as Lt contains all overlapping words of $k = 2$ elements from the target, while Lq contains all overlapping words of the same size from the query. So we get that Lt={(1,AC),(2,CT),(3,TG),(4,GA),(5,AT),(6,TT),(7,TG)} and Lq={(1,GA),(2,AG),(3,GT),(4,TG)}. When the program comes to the motif m=GATTG which is compared to the query q=GAGTG, two common words are identified: ((4,GA) in Lt; (1,GA) in Lq) and ((7,TG) in Lt; (4,TG) in Lq). The difference between the positions of these words in the motif and the query is equal and it equals 3 (4-1=3 and 7-4=3) and this is true for 2 hits. Since this number is the highest compared to any other motif, we get that the motif m=GATTG is the motif that is homologous to the query q=GAGTG.

YASS (Noé et al., 2005) is another homology search tool which is based on sensitive and efficient data filtering algorithm. It looks for close hits that form groups of close hits. The distance between hits in the groups must not exceed d. The parameter d is computed dynamically as a function of the frequency of base substitutions and insertions (deletions). However there must be at least n hits in the group in order to consider the cluster as a group of close hits.

Some software tools such as MUMmer (Delcher et al., 2003) and AVID (Bray et al., 2003) exploit the benefits of suffix trees. As far as known, suffix trees can be constructed and searched in $O(n)$ time, where n is the length of the sequence. Suffix trees are also space efficient because $O(n)$ memory is required. The moderate memory requirement makes suffix trees suitable tool for big genomic comparisons.

MUMmer (Delcher et al., 2003) is heuristic approach for fast comparison of big genomic sequences, such as entire genomes. It is based on three concepts: suffix tree, identification of longest common substrings and Smith-Waterman algorithm. MUMmer exploits a suffix tree to identify unique longest common substrings between two samples, while the Smith-Waterman algorithm is used in the final phase to construct the alignment. The unique longest common substrings are referred as MUM-mers and by definition they are hits which are not part of any longer hit.

We will demonstrate the way MUMmer operates to the samples: a:ACTGCACCACACCC and b:ACTCCACCACAACTCC. Hits AC and CT can not be considered for MUM-mers because they are part of longer hit such as ACT, Fig. 2. We cannot also consider CCA hit for MUM-mer because this hit occurs twice times in the sample b, Fig. 2.

The program exploits suffix tree in order to identify the set of all MUM-mers that are identified in $O(n + m)$ time and space, such as n is the length of the sample a and m is the length of the sample b. The set of all MUM-mers is sorted next, wherefrom the longest subset of MUM-mers that occur in the same order in the samples a and b is extracted.

For the samples that we use, the subset of MUM-mers includes hits (1) and (2), Fig. 2. Hit (3) on Fig. 2 cannot be included in the subset because it intersects with hit (2), Fig. 2.

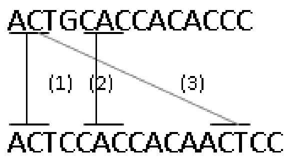

FIG. 2. Identification of MUM-mers

Before the final alignment, a local identification of: long insertions, repetitions, short mutations, tandem repeats and SNPs has to be performed.

The program for global alignment: AVID (Bray et al., 2003) also exploits suffix tree to detect the longest common hits between two samples. The two samples are merged into one with delimiter 'N' and for the joint sequence suffix tree is constructed, Fig. 3. The program traverses the suffix tree to find the longest repeating strings that in fact are the longest common hits between the samples, Fig. 3. The set of all common hits is filtered such as common hits that are shorter than half of the length of the longest hit must be rejected. A set of parallel common hits is afterwards selected. Hits are selected by a modified Smith-Waterman program that computes the score for each selection, based on the length of the hit and the score of the alignment of nearby hits.

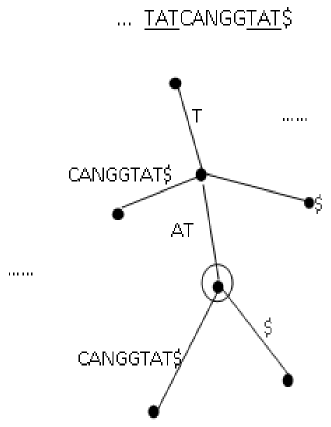

FIG. 3. Searching for repeated substrings

RESULTS AND DISCUSSIONS

Properties of the algorithms discussed in the previous section are listed into Table 1.

TABLE 1. Properties of heuristic algorithms

Algorithm	Properties
FASTA	Works with protein and nucleotide data; Searches for exact and short hits which are afterwards extended and connected into an alignment applying Smith-Waterman algorithm; Runs slowly
BLAST	Applicable to protein and nucleotide data; Looks for query and query-similar words into the database; Generally runs faster than FASTA; The sensitivity and the speed of search depends of the parameter w (the size of words)
Pattern Hunter	Unlike FASTA and BLAST that search for perfect this, Pattern Hunter looks also for imperfect hits based on matching model; Improved sensitivity of search compared to FASTA and BLAST
BLAT	Unlike previous methods that index the query, BLAT indexes the database; Runs very fast and searches for three types of hits: perfect, near-perfect and multiple-perfect hits; Main drawback: works only with closely-related samples
FLASH	Detects motifs in the database that resemble to query
YASS	Looks for clusters of near or close hits which are considered as groups
MUMmer	Suffix tree based approach; Looks for MUM-mers or unique and longest common substrings that are identified in $O(n+m)$ time and space
AVID	Exploits suffix tree; Merges two samples into one with delimiter 'N' for which the suffix tree is constructed; Traverses the suffix tree to find the longest repeating substrings that match the longest common hits

CONCLUSIONS

In this paper we discuss the most popular algorithms and software tools for genomic similarity search. Due to the large scale data, all these methods apply some type of heuristics to speed up the search process. Some of the algorithms, such as FASTA and BLAST split the query into short words which are afterwards searched into the database upon what local similarities are detected. The precision of search can be improved if instead of exact hits, imperfect hits are also considered. The concept of imperfect hits was introduced in Pattern Hunter. Imperfect hits include one or more mismatches and they can be the key for detecting distant homologies. For faster search, some methods such as BLAT, index the database instead of the query. The speed of search can be also improved if groups of near hits are only considered; YASS.

However we must mention that there are also suffix based approaches for this purpose. Those methods such as: MUMmer and AVID exploit the computational benefits of suffix trees which are constructed and searched in $O(n)$ time. MUMmer searches for unique and longest common substrings, while AVID joins two samples into one, for which the suffix tree is constructed. Traversing the suffix tree and looking for the longest and repeating substrings, the longest common hits are identified.

REFERENCES

- Altschul S.F., Madden T.L., Schäffer A.A., Zhang J., Zhang Z., Miller W., Lipman D.J. 1997. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic acids research 25(17):3389-3402.
- Bray N., Dubchak I., Pachter L. 2003. AVID: A global alignment program. Genome research 13(1):97-102.
- Califano A., Rigoutsos I. 1993. FLASH: A fast look-up algorithm for string homology, pp.353-359. In: Computer Vision and Pattern Recognition, IEEE.
- Delcher A.L., Salzberg S.L., Phillippy A.M. 2003. .Using MUMmer to identify similar regions in large sequence sets. Current protocols in bioinformatics (1):10-13.
- Fickett J. W. 1984. Fast optimal alignment. Nucleic acids research 12(1Part1):175-179.

- Kent W.J. 2002. BLAT—the BLAST-like alignment tool. Genome research 12(4):656-664.
- Ma B., Tromp J., Li M. 2002. PatternHunter: faster and more sensitive homology search. Bioinformatics 18(3):440-445.
- Needleman S.B., Wunsch C.D. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of molecular biology 48(3):443-453.
- Noé L., Kucherov G. 2005. YASS: enhancing the sensitivity of DNA similarity search. Nucleic acids research 33(suppl 2):W540-W543.
- Pearson W.R., Lipman, D.J. 1988. Improved tools for biological sequence comparison, pp.2444-2448. In: Proceedings of the National Academy of Sciences.
- Sellers P.H. 1974. An algorithm for the distance between two finite sequences. Journal of Combinatorial Theory, Series A, 16(2):253-258.
- Smith T.F., Waterman M. S. 1981. Identification of common molecular subsequences. Journal of molecular biology 147(1):195-197.
- Waterman M.S., Eggert M. 1987. A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons. Journal of molecular biology 197(4):723-728.