

---

# SimSearch: A new variant of dynamic programming based on distance series for optimal and near-optimal similarity discovery in biological sequences

Sérgio A. D. Deusdado<sup>1</sup> and Paulo M. M. Carvalho<sup>2</sup>

<sup>1</sup>ESA, Polytechnic Institute of Bragança, 5300, Bragança, Portugal

<sup>2</sup>Department of Informatics, Engineering School, University of Minho, 4710, Braga, Portugal  
sergiad@ipb.pt; pmc@di.uminho.pt

**Abstract.** In this paper, we propose SimSearch, an algorithm implementing a new variant of dynamic programming based on distance series for optimal and near-optimal similarity discovery in biological sequences. The initial phase of SimSearch is devoted to fulfil the binary similarity matrices by signalling the distances between occurrences of the same symbol. The scoring scheme is applied further, when analysed the maximal extension of the pattern. Employing bit parallelism to analyse the global similarity matrix's upper triangle, the new methodology searches the sequence(s) for all the exact and approximate patterns in regular or reverse order. The algorithm accepts parameterization to work with greater seeds for near-optimal results. Performance tests show significant efficiency improvement over traditional optimal methods based on dynamic programming. Comparing the new algorithm's efficiency against heuristic based methods, equalizing the required sensitivity, the proposed algorithm remains acceptable.

**Keywords:** Similarity discovery, dynamic programming, distance series

## 1 Introduction

Dynamic programming (DP) is a mathematical technique widely used in multiple research fields providing optimal solutions for complex problems, mostly of combinatorial nature. The final and optimal solution is achieved by analysing the sub-problems recursively and their optimal solutions, combining and integrating the partial solutions. Bellman's seminal work [1] marks the beginning of DP automatization towards informatics, describing a class of algorithms to implement DP. Later on, and naturally, this technique has been adopted by bioinformatics.

In bioinformatics, DP is fundamentally used to discover sequence alignments, considering both local and global alignments [2]. This task involves basically the search for similarities, analysing the involved sequences and pointing out the correct correlated segments. Since biological homologies are commonly approximate, the similarities may contain an acceptable degree of deviation, corresponding to an admissible number of mismatches; thus this attribute increases the problem's complexity. Similarity evaluation is based on the "edit distance" concept. The edit distance corresponds to the minimum number of operations required to convert a sequence into another using three edit operations to insert, delete or substitute symbols. In order to evaluate the correlation degree, a scoring scheme is necessary to assess the similar regions and, on the other hand, penalize deviations (mismatches, substitutions and gaps). The obtained scores are stored in a similarity or scoring matrix providing the basis for further analysis.

DP is called an exhaustive technique since it tests all possible combinations and provides 100% sensitivity. Heuristic based methodologies are evolving as alternative

solutions to attain fast and efficient near-optimal similarity discovery. Searching the sequences with partial but reasonable sensitivity allows to achieve the majority of homologies, it is a trade-off solution since heuristic based algorithms use a fraction of the processing time and resources required by optimal methods.

In this paper we propose SimSearch, an algorithm implementing a new variant of dynamic programming, based on distance series, for optimal and near-optimal similarity discovery in biological sequences. Using distance series enables a simpler similarity matrix construction, as well as, it only requires binary digits to represent similarity regions. The new searching strategy does not use a scoring scheme to store scores in the matrix, instead, it is further used during the matrices' analysis to detect the pattern's presence and extent. Backtrack analysis is used to identify continuities among the sub-results from adjacent matrices, eventually needed to compose wider similarity regions.

The performance tests carried out in our work, show significant efficiency improvement over traditional optimal methods based on dynamic programming.

## **2 Related work**

In the domain of bioinformatics two well-known methodologies, based on dynamic programming, were created and evolved in the last three decades to provide 100% sensibility and complete similarities discovery. The first use of the dynamic programming approach for biological sequences' global alignment was reported by Saul Needleman and Christian Wunsch in 1970 [3] and then, in [4] slightly modified by Sellers. In 1981, Smith and Waterman proposed a new algorithm [5] in order to solve the local alignment problem.

The comparison of sequences using dynamic programming is often a slow process due to the intensive computing required. To overcome this constraint new approaches based on heuristics were developed to obtain near-optimal solutions, reducing significantly the time required to perform the homology search and, concomitantly lowering the processing and memory requirements to complete the task. Representative heuristic-based algorithms are BLAST [6-8] and PatternHunter [9, 10].

Using distance series to analyse biological sequences is not a novelty in bioinformatics, statistical analysis using distance series are common [11, 12]. However, this methodology is still underdeveloped as regards sequence comparison and pattern discovery.

Despite the trend of alignment applications development in recent years indicates a preponderance of heuristic-based solutions, optimal homology search is still a necessity in several bioinformatics applications, such as biological data compression, DNA linguistics study and others, therefore the motivation for the development of efficient optimal homology search algorithms remains and SimSearch is a contribution.

### **2.1 Smith-Waterman algorithm**

When considering optimal alignment and dynamic programming, the most used algorithm to compute the optimal local alignment is the Smith-Waterman [5] with Gotoh's [13] improvements for handling multiple sized gap penalties. The two sequences to be compared, the query sequence and the database sequence, are

defined as  $Q=q_1, q_2 \dots q_m$  and  $D=d_1, d_2 \dots d_n$ . The length of the query sequence and database sequence are  $m=|Q|$  and  $n=|D|$ , respectively. A scoring matrix  $W(q_i, d_j)$  is defined for all residue pairs. Usually the weight  $W(q_i, d_j) \leq 0$  when  $q_i \neq d_j$  and  $W(q_i, d_j) > 0$  when  $q_i = d_j$ . The penalty for starting a gap and continuing a gap are defined as  $G_{init}$  and  $G_{ext}$ , respectively.

$$E_{i,j} = \max \left\{ \begin{array}{l} E_{i,j-1} - G_{ext} \\ H_{i,j-1} - G_{init} \end{array} \right\} \quad (1)$$

$$F_{i,j} = \max \left\{ \begin{array}{l} F_{i-1,j} - G_{ext} \\ H_{i-1,j} - G_{init} \end{array} \right\} \quad (2)$$

The alignment scores ending with a gap along  $D$  and  $Q$  are Equation (1) and Equation (2), respectively.

$$H_{i,j} = \max \left\{ \begin{array}{l} 0 \\ E_{i,j} \\ F_{i,j} \\ H_{i-1,j-1} - W(q_i, d_j) \end{array} \right\} \quad (3)$$

The values for  $H_{ij}$ ,  $E_{ij}$  and  $F_{ij}$  are equal to 0 when  $i < 1$  or  $j < 1$ .

By assigning scores for matches or substitutions and insertions/deletions, the comparison of each pair of characters is weighted into a matrix by calculation of every possible path for a given cell. In any matrix cell the value represents the score of the optimal alignment ending at these coordinates and the matrix reports the highest scoring alignment as the optimal alignment.

The Smith-Waterman algorithm runs in quadratic time and requires quadratic space:  $O(nm)$  to compare sequences of lengths  $n$  and  $m$ .

## 2.2 BLAST algorithm

The first stage of BLAST [6-8] - and many other homology search tools - involves identifying hits: short, high-scoring matches between the query sequence and the sequences from the collection being searched. The definition of a hit and how they are identified differs between protein and nucleotide searches, mainly because of the difference in alphabet sizes. This algorithm involves first identifying high scoring sequence pairs (HSPs) by identifying short words of length  $W$  in the query sequence (seeds), which either match or satisfy some positive-valued threshold score  $T$  when aligned with a word of the same length in a database sequence.  $T$  is referred to as the neighbourhood word score threshold [6]. These initial neighbourhood word hits act as seeds for initiating searches to find longer HSPs containing them. The word hits are extended in both directions along each sequence for as far as the cumulative alignment score can be increased. Extension of the word hits in each direction are halted when: the cumulative alignment score falls off by the quantity  $X$  from its maximum achieved value; the cumulative score goes to zero or below, due to the accumulation of one or more negative-scoring residue alignments; or the end of either sequence is reached. The BLAST algorithm parameters  $W$ ,  $T$ , and  $X$  determine the sensitivity and speed of the alignment. The BLAST program uses BLOSUM62 scoring matrix [14] alignments and, by default, a word length ( $W$ ) of 11.

### 2.3 PatternHunter algorithm

The novel introduction of spaced seed idea in the filtration stage of sequence comparison by Ma et al. [9] has greatly increased the sensitivity of homology search without compromising the speed of search. This is the underlying idea of PatternHunter I [9] and II [10], a new generation of general purpose homology search tools, considered prominent solutions in similarity discovery in biological data. PatternHunter is a heuristics-based algorithm, highly efficient and sensitive due to the use of spaced seeds. Spaced seeds are strategically designed gapped *n*-grams aiming to identify similarity regions in biological sequences. In PatternHunter, the residues of a seed are interleaved with “don’t care” filler residues in order to allow matches to sub-strings in the query sequences that do not include all the residues in the seed one after the other. Further analysis is used to extend the seed in both directions in an effort to obtain longer homologous subsequences. Multiseed filtration was a natural evolution of PatternHunter, achieving improved sensitivity.

## 3 The new algorithm

SimSearch is originally an optimal similarity discovery algorithm as it performs an exhaustive search for repeats. However, it includes a filtering strategy which consists of using a minimum length seed to accelerate the discovery of high scoring similarity regions. The SimSearch seeds are, in fact, oriented to locate pairs of patterns already found in the sequence(s) contrarily to the seed concept of other algorithms such as BLAST or PatternHunter. In this way, the proposed algorithm searches for existing patterns already registered in the similarity matrices.

The similarity identification method used by SimSearch is based on symbols’ distance series analysis and its correlation. Basically, regarding genomic sequences, each one of the four bases (*a, c, g, t*) occurs in the sequence frequently but not periodically, so collecting the distance (in bases) between occurrences can be useful to discover similarity. In fact, if *n* consecutive symbols appear in the sequence equidistant to a next occurrence then it is safe to affirm that a similarity of length *n* is present.

Initially, the sequence is analysed to determine and list the distance from the last occurrence of each symbol. Using a linked list for each symbol, the first occurrence of each symbol will constitute the head of the linked list, the node stores a value corresponding to the number of symbols preceding it in the sequence. The next occurrences will form the next nodes, containing information about the distance from the last symbol’s occurrence. The tail of the list will correspond to the last occurrence of the symbol in the sequence. Formally, the distance series are composed by a number of terms equal to the number of identical symbol’s occurrences; each term is a number representing the distance in symbols to reach the next occurrence of an identical symbol, except for the first term whose value corresponds to its ordinal position in the sequence.

SimSearch can be adapted to execute different sequences’ comparisons or similarity discover within a sequence. SimSearch was initially conceived to support a biological data compression application, thus to find out repetitions within the sequence. In this case, we only have the query sequence, the database sequence is used, for instance, when searching palindromes - the patterns present in the reverse complementary sequence. To compare two sequences, the SimSearch’s linked lists

must register the distance to the next occurrence of the base  $b$  in the database sequence to the same base  $b$  in the query sequence.

### 3.1 The similarity matrices

Each similarity matrix is filled out writing the symbol “1” in the matrix lines present in the linked list, corresponding to the symbol in each column. In this manner, for each symbol of the query sequence only the future occurrences are signalled. The inherent simplicity of the process provides great efficiency, allowing the analysis of large sequences using moderate processing time and computing resources.

Going into details, and providing a practical example, let's consider a genomic sequence  $x = \text{tatccgcattatgcgata}$ , with length  $m=18$ . Table 1 contains the resulting similarity matrix, showing the patterns corresponding to successive 1s. A pattern initiated at  $x[0]$ , of length 6 and containing a mismatch was highlighted, the respective repetition occurs 9 (line number) symbols ahead at  $x[9]$ .

**Table 1.** The binary similarity matrix based on distance series

	T	A	T	C	C	G	C	A	T	T	A	T	G	C	G	A	T	A
1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	
2	1	0	0	0	1	0	0	0	0	1	0	0	1	0	0	1		
3	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0			
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
5	0	0	0	0	0	0	0	0	0	0	1	1	0					
6	0	1	1	0	0	0	0	0	0	0	0	0						
7	0	0	1	0	0	1	1	0	0	1	1	0						
8	1	0	0	0	0	0	0	1	1	0	0							
9	1	1	1	0	1	1	0	0	0	0								
10	0	0	0	1	0	0	0	1	0									
11	1	0	0	0	0	0	0	0										
12	0	0	0	0	0	0	0											
13	0	0	0	0	0	0												
14	0	1	1	0	0													
15	0	0	0	0														
16	1	1	0															
17	0	0																
18	0																	

The number of 1s in the similarity matrix can be determined using the following theorem (Theorem 1).

#### Theorem 1

Being  $S(n) = 1 + 2 + 3 + 4 + 5 + \dots + n$ , with  $n \in \mathbb{N}$ , described as the “sumtorial” of  $n$ , considering the expression  $S(n) = \frac{n(n+1)}{2}$  as a valid method to calculate

$S(n)$  and having  $n_1$  as the sum of symbol “A” occurrences,  $n_2$  as the sum of symbol “C” occurrences,  $n_3$  as the sum of symbol “G” occurrences and  $n_4$  as the sum of symbol “T” occurrences, the expression to calculate the number of 1s to be written in the similarity matrix will be:

$$\text{Number of 1s} = S(n_1-1) + S(n_2-1) + S(n_3-1) + S(n_4-1)$$

Applying the theorem to the segment  $x = \text{"tatccgattatgcgata"}$ , with  $n_1=5$ ,  $n_2=4$ ,  $n_3=3$  e  $n_4=6$ , the total number of cells signalled with 1 will be:  $S(4) + S(3) + S(2) + S(5) = 10 + 6 + 3 + 15 = 34$ .

Analysing the time complexity involved in the matrix's filling out process, the worst case occurs when the sequence is uniform (all the bases are equal), and, in this case, the complexity can be expressed as  $O(S(n-1))$ , being  $n$  the length of the sequence. The average case, also the most likely to occur, corresponds to a balanced distribution of all the four genomic symbols, thus the time complexity is  $O(4S((n/4)-1))$ .

SimSearch operates by dividing the sequence to examine in manageable subsequences, specifically segments of dimension  $d=10.000$  symbols by default. Facing the impossibility of representing here the full SimSearch subtable, Table 1 shows a division of the whole matrix in  $3 \times 3$  sub-matrices. The number of matrices to process is also calculated by the expression in Theorem 1. Being  $n$  the length of the sequence and  $d$  the dimension of the matrices, the number of matrices to process will be  $S(n/d)$ .

The next step is devoted to filter the similarity occurrences, avoiding irrelevant patterns. The "irrelevant" definition depends on the assumed scoring scheme and the resulting distance edition cost.

### 3.2 Similarity discovery

The similarity matrices include all the present similarities, i.e., 100% sensitivity at this stage. To uncover the similar segments requires the analysis of the similarity matrices in order to identify different kinds of successions of 1s in the matrix. The shape of the successions indicates the quality of the patterns. In fact, the proposed methodologies allow the identification of exact, approximate, reverse, or palindrome patterns.

The patterns' discovery begins with the detection of seeds. Considering a seed of length  $W=9$ , the algorithm analyses the matrices in several ways suspending the search when a seed is located (in the example, nine successive 1s). A local analysis is carried out to examine the maximum extent of the pattern. The proposed algorithm uses bitwise operations to locate the seeds quickly, an adaptation of the algorithm SBNDM [15] was also included for fast seed matching. SimSearch seeds are limited to 32 characters due to bit-parallelism operations restrictions.

### 3.3 The scoring scheme

SimSearch uses a scoring scheme to analyse the similarity matrices, although it is not registered in the matrices. Whenever a seed is detected, the scoring scheme is used attempting to extend the pattern in both sides up to a pre-defined threshold. This means that the attempt to discover a larger pattern allows a certain number of mismatches. SimSearch's default scoring scheme is: *Match=1 point; Mismatch=-1 point; Open a gap=-5 points*. However, the user may redefine these values.

The accumulation of successive negative score, exceeding a defined threshold value, will imply the end of the pattern analysis. On the other hand, only patterns with a minimum score are considered.

For certain applications, a minimum pattern length (or score, in a different perspective) is required, so SimSearch includes a parameter to set this value.

3.4 Exact repetitions and overlapped patterns

As referred above, the exact repetitions are identified by consecutive successions of 1s in the lines of the similarity matrices. An important disruption that affects pattern discovery algorithms corresponds to the existence of overlapped patterns within a larger pattern. This normally implies redundant processing as overlapped patterns are not relevant to functional genomics as they are to stringology.

The proposed algorithm, through its similarity matrices, keeps record of all the patterns, including the overlapped ones. For instance, the pattern “aaaaa” contains three occurrences of the pattern “aaa”. Similarly, the pattern “ctctctc” contains three occurrences of the pattern “ctc”. In the analysis stage, SimSearch includes mechanisms to avoid redundant processing regarding overlapped sub-patterns.

Table 2 contains the resulting similarity matrix considering the sequence “atcatcatc”. For simplicity reasons, the matrix is only filled with the relevant 1s.

- Analysing the similarity matrix (see Table 2), two patterns are visible:
- the first corresponds to “atcatc”, present both at 1<sup>st</sup> and 4<sup>th</sup> bases. However, there are overlapped sub-patterns within it.
  - the second pattern corresponds to a sub-pattern of the first pattern, where the segment “atc” is repeated at 1<sup>st</sup> and 7<sup>th</sup> bases, thus there is no significant discovery as the first pattern covers the same region.

Tables 2 and 3. Redundant patterns examples

	A	T	C	A	T	C	A	T	C
1									
2									
3	1	1	1	1	1	1			
4									
5									
6	1	1	1						
7									
8									

	A	A	A	A	T	A	A	A	A
1	1	1	1	1		1	1	1	1
2	1	1	1	1	1	1	1	1	
3	1	1	1	1	1	1	1		
4	1	1	1	1	1	1			
5	1	1	1	1	1				
6	1	1	1	1					
7	1	1	1						
8	1								

Multiple overlapped patterns cause a dense region of 1s in the similarity matrix as shown in Table 3, considering the sequence “aaaataaaa”. The larger exact repeat, “aaaa”, occurs at 1<sup>st</sup> and 5<sup>th</sup> bases. Several minor overlapped patterns occur within the larger ones, for instance, the pattern “aa” occurs six times.

In order to overcome the redundancy inherent to overlapped patterns, SimSearch protects itself from over-searching. This is accomplished calculating an exclusion region based on the upper triangle limited by the diagonal that encloses the larger pattern (see the triangle in Table 3). In this example, several minor patterns were discovered before the larger pattern. The discovery of the larger pattern will dismiss the minor ones and avoid the subsequent search for more irrelevant pattern within the larger one.

3.5 Reverse repetitions

Basically a reverse pattern is a symmetric pattern. Considering the sequence “attgcgtta”, the pattern “attg” occurs in the reverse form at the end of the sequence. Table 4, representing the similarity matrix originated by the sequence “attgcgtta”, shows that the reverse pattern is also registered in the matrix, but not in a horizontal

line as in the exact pattern occurrences. The reverse patterns are registered in the diagonal lines, not the regular diagonal (45°) but in the 67,5° diagonal.

**Table 4.** A reverse repetition detected in the sequence “attgcgtta”

	A	T	T	G	C	G	T	T	A
1			1					1	
2									
3									
4				1					
5									
6			1						
7									
8	1								

### 3.6 Approximate repetitions

In genomics, approximate repetitions are naturally more abundant than exact repetitions, the explanation lies on the results of genomic maintenance and evolutionary mechanisms [16]. The study of approximate repeats is important in functional genomics and several bioinformatics tools are available exclusively focused on this subject [17, 18]. The SimSearch’s similarity matrices also record the pattern interruptions.

**Tables 5 and 6.** Approximate patterns represented in the similarity matrices

	A	G	A	C	T	A	A	A	C
1						1	1		
2	1					1			
3				1					
4				1					
5	1	0	1	1					
6	1								
7	1								
8									

	A	A	G	G	A	A	T	G	G	A
1	1		1		1			1		
2										
3			1							
4	1	1		1						
5	1			1	1	1				
6				1						
7										
8			1							
9	1									

#### 3.6.1 Substitutions

The proposed algorithm identifies substitutions in a pattern if a succession of 1s, in the same line, contains interspersed 0s. Analysing the sequence “agactaac” in Table 5, an approximate pattern containing one substitution is present, specifically, the segments “agac” and “aaac” only differ on the second base.

#### 3.6.2 Insertions/Deletions (Indels)

Approximate repeats may also include rearrangements such as insertions or deletions. In fact, the symmetry property implies an insertion for each deletion and vice-versa, i.e., if a certain string needs an insertion to equalize a second string then the equalization may also be operated with a deletion in the second string. The word indel is used in these cases.

For instance, when analysing the sequence “aaggaatgga”, the pattern “aagga” occurs twice depending on a deletion of the seventh base. From other perspective, the pattern “aatgga” occurs twice depending on an insertion (“t”) in the third base.



The SimSearch similarity matrices reflect these situations as a succession of 1s whose continuity may be extended in adjacent lines (see Table 6). If consecutive indels occur (gap) then the distance between lines corresponds to the extent of the gap.

## 4 Results and Discussion

The performance analysis of sequence alignment is necessary to assess the efficiency of the different methodologies [19]. In order to evaluate and compare the performance of the new algorithm three different competitors were chosen considering their importance, methodology and efficiency. The first choice was SSearch [20] (version 35), a classical dynamic programming algorithm used to achieve optimal local alignments. SSearch uses Pearson's implementation of the method of Smith and Waterman. The second choice was the BLAST tool (version 2.2.18), the most famous and widely used application in bioinformatics to execute local and global alignments. The third choice was a state-of-the-art and top performing tool, PatternHunter (version 2).

SimSearch's default settings are, mainly, similarity matrices of 10.000\*10.000 cells, seed length  $W=11$  and local alignments with scores no less than 16. The scoring scheme is the one defined in section 3.3. Other contenders use, when applicable, the same settings to provide equal conditions. Simsearch was tested for optimal homology search using  $W=3$ . PatternHunter was tested using 2 and 8 seeds for different sensitivity comparisons.

A prototype of SimSearch was developed, implemented in C language, and compiled with best optimizations using *gcc*. Performance tests were executed using a system based on an Intel Pentium IV - 3,4 GHz - 512KB cache - 1GB DDR-RAM.

Two genomic sequences were tested separately for similarity regions discovery: a human gene (*humghcsa*) with ~65 Kbases and an entire genome (*e. coli*) with ~4,6 Mbases. The assessed processing times are exposed in Table 7.

**Table 7.** Performance vs. sensitivity comparison

	Execution times					
	SimSearch (optimal)	SimSearch (near-optimal)	SSearch	BLASTn	PHunter (2 seeds)	PHunter (8 seeds)
<i>humghcsa</i>	24 seconds	14 seconds	7,5 hours	4 seconds	2 seconds	6 seconds
<i>e. coli</i>	1,22 days	17,67 hours	5 years*	52 seconds	14 seconds	45 seconds

\*estimation

SimSearch detects assertively all the patterns present in a genomic sequence, PatternHunter and especially BLAST miss some patterns due to heuristics limitations. SimSearch overperforms clearly traditional DP algorithms being comparable with heuristics-based methodologies when the required sensitivity is near maximal.

## 5 Conclusions

In this paper we have proposed SimSearch, a new genomic-oriented homology search algorithm. Similarity discovery within a sequence or among sequences is an intensive and very time-consuming computational task. Optimal alignment solutions are, even today, computationally prohibitive as large genomic sequences analysis

can take years using a conventional DP approach. The proposed algorithm is an exhaustive search algorithm, based on the analysis of similarity matrices obtained registering properly each symbol's distance series. The new algorithm follows a dynamic programming logic and achieves an optimal solution with 100% sensitivity. To improve processing time, SimSearch includes a filtering methodology centering attention in high score segments searched using a fast exact pattern-matching module. SimSearch is incomparably faster than the Smith-Waterman algorithm, the most used optimal local alignment solution based on DP. Compared with heuristics-based solutions SimSearch is, obviously, slower, but its competitiveness can be increased if parameterized to near-optimal search.

**Acknowledgements.** This work has been partially supported by PRODEP.

## References

1. Bellman, R.E.: *Dynamic Programming*. Princeton University Press. (1957)
2. Kruskal, J.B.: *An overview of sequence comparison*. Addison Wesley. (1983)
3. Needleman, S.B., and Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* 48, 443-453. (1970)
4. Sellers, P.H.: On the theory and computation of evolutionary distances. *SIAM J. Appl. Math.* 26, 787-793. (1974)
5. Smith, T.F., and Waterman, M.S.: Identification of common molecular subsequences. *Journal of Molecular Biology* 147, 195-197. (1981)
6. Altschul, S.F., *et al.*: Basic local alignment search tool. *J. Mol. Biol.* 215, 403-410
7. Huang, X., and Miller, W.: (1991) A time-efficient, linear-space local similarity algorithm. *Adv. Appl. Math.* 12, 337-357. (1990)
8. Altschul, S.F., *et al.*: Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Res.* 25, 3389-3402. (1997)
9. Ma, B., *et al.*: Pattern Hunter: fast and more sensitive homology search. *Bioinformatics* 18, 440-445. (2002)
10. Li, M., *et al.*: PatternHunter II: Highly Sensitive and Fast Homology Search. *J. Bioinform. Comput. Biol.* 2, 417-439. (2004)
11. José, M.V., *et al.*: Statistical properties of DNA sequences revisited: the role of inverse bilateral symmetry in bacterial chromosomes. *Physica A: Statistical Mechanics and its Applications* 351, 477-498. (2005)
12. Teodorescu, H.-N., and Fira, L.-I.: Analysis of the predictability of time series obtained from genomic sequences by using several predictors. *Journal of Intelligent and Fuzzy Systems* 19, 51-63. (2008)
13. Gotoh, O.: An improved algorithm for matching biological sequences. *J. Mol. Biol.* 162, 705-708. (1982)
14. Henikoff, and Henikoff: Amino Acid Substitution Matrices from Protein Blocks. In *Natl. Acad. Sci. USA* 89:10915. (1989)
15. Peltola, H., and Tarhio, J.: Alternative Algorithms for Bit-Parallel String Matching. *Lecture Notes in Computer Science* 2857, 80-93. (2003)
16. Schmidt, T., and Heslop-Harrison, J.S.: Genomes genes and junk: the large-scale organization of plant chromosomes. *Trends Plant Sci.* 3, 195-199. (1998)
17. Kolpakov, R., *et al.*: mreps: efficient and flexible detection of tandem repeats in DNA. *Nucleic Acids Res.* 31, 3672-3678. (2003)
18. Lefebvre, A., *et al.*: FORRepeats: detects repeats on entire chromosomes and between genomes. *Bioinformatics* 19, 319-326. (2002)
19. Sanchez, F., *et al.*: Performance Analysis of Sequence Alignment Applications. In *IEEE International Symposium on Workload Characterization*, 51 - 60. (2006)
20. Pearson, W.R.: Searching protein sequence libraries: comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms. *Genomics* 11, 635-650. (1991)