

IMPROVED ALIGNMENT OF HOMOLOGOUS DNA SEQUENCES

Done STOJANOV*, Cveta MARTINOVSKA

Faculty of Computer Science, University Goce Delcev, Krste Misirkov nn – Stip, Republic of Macedonia

*Corresponding author e-mail: done.stojanov@ugd.edu.mk

Received 22 November 2013; accepted 13 December 2013

ABSTRACT

A new aligning approach for homologous DNA sequences is presented, being faster than the standard dynamic programming based implementations. Searching for exact and non-crossing hits as fast as possible, tracking hits' positions in a data vector, being dynamically sorted in ascending order, regarding the starting positions of the hits being identified, the time complexity of the methodology is $O(kn \log_2 m)$, such as: k is the number of exact, non-crossing hits, n is the length of the larger DNA sequence and m is the length of the shorter DNA sequence. The space complexity is linear $O(k)$, what is favorable especially if larger DNA sequences have to be aligned, such as the inequality $k < m$ is satisfied.

KEY WORDS: bioinformatics, time-improved algorithm, linear memory complexity, exact and non-crossing hits.

INTRODUCTION

Since 1970, different concepts for DNA alignment have been proposed. In order to align two sequences $\{a_i\}_{i=1}^n$ and $\{b_j\}_{j=1}^m$ with the Needleman-Wunsch algorithm (Needleman *et al.*, 1970) a score matrix $[s_{i,j}]_{(n+1) \times (m+1)}$ is constructed, such as: $s_{i,j}$ is the score of the best alignment of the subsequences: $a_1 \dots a_i$ and $b_1 \dots b_j$, being recursively calculated as: $s_{i,j} = \max \{s_{i-1,j} + gp, s_{i,j-1} + gp, s_{i-1,j-1} + s(a_i, b_j)\}$, gp is the penalty for aligning a base with a gap and $s(a_i, b_j)$ is the score for aligning bases a_i and b_j , $gp < 0$, $s(a_i, b_j) < 0$ if $a_i \neq b_j$, $s(a_i, b_j) > 0$ if $a_i = b_j$. If $s_{i,j} = s_{i-1,j} + gp$, a_i is aligned with a gap at position j in the sequence b . If $s_{i,j} = s_{i,j-1} + gp$, b_j is aligned with a gap at position i in the sequence a . If $s_{i,j} = s_{i-1,j-1} + gp$, nucleotides a_i and b_j are aligned. Matrix cells $s_{i,0}$, $0 \leq i \leq n$ and $s_{0,j}$, $0 \leq j \leq m$, are set up to $i \times gp$ and $j \times gp$ respectively.

Being computationally expensive as the Needleman-Wunsch algorithm, Gotoh's algorithm (Gotoh, 1982) is affine gap penalty approach. The score for extending opened gap with k gaps is calculated as $s(k) = go + k \times ge$, such as: go is the

gap opening score, ge is gap extending score, $go < ge$, $go < 0$, $ge < 0$. Three matrices are required. One $[m_{i,j}]_{(n+1) \times (m+1)}$ tracking the score of the best alignment for the subsequences: $a_1 \dots a_i$ and $b_1 \dots b_j$, given that bases a_i and b_j are aligned, other $[n_{i,j}]_{(n+1) \times (m+1)}$ tracking the best score of an alignment for the subsequences: $a_1 \dots a_i$ and $b_1 \dots b_j$, given that a_i is aligned with a gap at position j in the sequence b and matrix $[k_{i,j}]_{(n+1) \times (m+1)}$ tracking the best score of an alignment for the subsequences: $a_1 \dots a_i$ and $b_1 \dots b_j$, given that b_j is aligned with a gap at position i in the sequence a . Matrix cells are calculated with the recursive formulas:

$$m_{i,j} = \max \{m_{i-1,j-1} + s(a_i, b_j), n_{i-1,j} + s(a_i, b_j), k_{i,j-1} + s(a_i, b_j)\}, n_{i,j} = \max \{m_{i-1,j-1} + go, n_{i-1,j} + ge\} \text{ and}$$

$$k_{i,j} = \max \{m_{i-1,j-1} + go, k_{i,j-1} + ge\}, \text{ such as: } s(a_i, b_j) \text{ is the score for aligning bases } a_i \text{ and } b_j, go \text{ is opened gap penalty and } ge \text{ is gap extending penalty. Cells } m_{0,0}, n_{0,0} \text{ and } k_{0,0} \text{ are set up to } 0, m_{i,0} = n_{i,0} = go + (i-1) \times ge, k_{i,0} = -\infty, m_{0,j} = k_{0,j} = go + (j-1) \times ge, n_{0,j} = -\infty.$$

For homologous DNA sequences, the concept of pairwise alignment within a specific diagonal band is computationally less expensive, compared with the typical dynamic programming-based approaches, that require fixed $O(nm)$ time. The idea of aligning two sequences, within a diagonal band, is discussed by Fickett (Fickett, 1984) and Unkkonen (Unkkonen, 1985). Chao, Person and Miller (Chao *et al*, 1992) presented banded-based algorithm for DNA pairwise alignment, which requires $O(NW)$ time, such as: N is the length of the shorter DNA sequence and W is the width of the band. The k -band algorithm is typical banded-based aligning approach, determining the best alignment of two same-size sequences, using at most k diagonals away from the main diagonal. Initially k is taken as a small number (usually 1), gradually increasing its value, until the score of the optimal alignment found within the k -diagonals wide band around the main diagonal α_k equals the score of the optimal global alignment, what is the case if $\alpha_k \geq m(n - k - 1) - 2(k + 1)gp$, such as: n is the length of the sequences, m is the uniform match cost and gp is gap penalty.

Instead of aligning whole sequences, segment-based sequence alignment approaches have been also proposed. DALIGN (Morgenstern *et al*, 1998) and DIALIGN 2 (Morgenstern *et al*, 1999) are such implementations, based on identification of consistent collections of diagonals (non-overlapping aligned DNA segments). Each diagonal is weighted. The set of consistent diagonals with maximum sum of weights defines the optimal alignment. AVID (Bray *et al*, 2003) searches for a

collection of non-overlapping and non-crossing matches. Once the collection is found, an alignment can be easily generated.

ClustalW (Thompson *et al.*, 1994) and Tcoffe (Cédric *et al.*, 2000) are multiple sequence alignment techniques. Aligning all pairs of sequences, in order to determine highly-related pairs of sequences, then combining them and performing alignments of closely-related dynamic programming aligned pairs of sequences, in order to construct evolutionary tree, is the idea behind ClustalW, employing substitution matrix. Tcoffe is a set of computational tools for multiple sequence alignment, without implicit use of substitution matrix.

SPA (Super Pairwise Alignment) (Shen *et al.*, 2002) is statistically-based method for pairwise alignment of homologous sequences, measuring local percentage similarity within window of size n . Each time the percentage of similarity is below a certain threshold, an insertion has occurred. Minimum gaps are inserted, in order to increase the percentage of similarity above the threshold. SPA result is not always the optimal (best scoring) one, but its computational complexity is linear.

Sequence database searching implementation such as: BLAST (Altschul *et al.*, 1990) and FASTA (Lipman *et al.*, 1985), being as fast as SPA, are mainly used for identification of similar sequences regarding referent DNA sequence.

In this paper, a new methodology for global alignment is proposed, based on fast identification of exact and non-crossing hits. Matching regions' positions are stored in a vector, based on the space-efficient representation (Stojanov *et al.*, 2012), (Stojanov *et al.*, 2013). Once identified, the positions where gaps are inserted are determined, i.e. the alignment is constructed.

MATERIALS AND METHODS

Searching for exact and non-crossing common hits

In order to find the longest common substring for the DNA sequences $a = \{a_i\}_{i=1}^{|a|}$ and $b = \{b_j\}_{j=1}^{|b|}$, $a_i, b_j \in \{A, C, T, G\}$, $|a| \geq |b|$, words of size $m-k$ from the both sequences are compared, until exact match of maximum size is identified, gradually increasing k for 1, starting from $k=0$. The comparison of the same size words $x: x_1x_2 \dots x_{m-k-1}x_{m-k}$, $x \in a$ and $y: y_1y_2 \dots y_{m-k-1}y_{m-k}$, $y \in b$ is performed recursively as follows:

- If $x_1 = y_1$ and $|x| > 1$, the comparison procedure is called for: $x: x_2 \dots x_{m-k-1}x_{m-k}$ and $y: y_2 \dots y_{m-k-1}y_{m-k}$.
- If $x_1 = y_1$ and $|x| = 1$, the starting and final positions of the words $x: x_1x_2 \dots x_{m-k-1}x_{m-k}$ and $y: y_1y_2 \dots y_{m-k-1}y_{m-k}$ in a and $b: p_{a,s}, p_{a,f}, p_{b,s}$ and $p_{b,f}$ are returned.
- If $x_1 \neq y_1$, the next overlapping word in a , $x: x_2x_3 \dots x_{m-k}x_{m-k+1}$ is compared to $y: y_1y_2 \dots y_{m-k-1}y_{m-k}$.

```

compare (x : x1x2...xm-k-1xm-k , y : y1y2...ym-k-1ym-k)
if (x1 == x2)
if (|x| > 1)
compare (x : x2...xm-k-1xm-k , y : y2...ym-k-1ym-k)
else return pa,s , pa,f , pb,s and pb,f
else compare(x : x2x3...xm-kxm-k+1 , y : y1y2...ym-k-1ym-k)

```

Comparing procedure is discussed for the short DNA samples: $a = \text{ACTG}$ and $b = \text{ACG}$. The search for the longest hit, starts with check for match of size $|b| - k = 3 - 0 = 3$, $k = 0$. Therefore sequence a overlapping words of size 3: $\{\text{ACT}, \text{CTG}\}$ have to be compared with sequence $b = \text{ACG}$.

In order to compare words ACT and ACG, the starting bases are compared, at first. Since they match, the comparison procedure is called for the substrings: CT and CG. The match between the starting nucleotides requires substrings: T and G to be compared. The mismatch between T and G, implies comparison of the second overlapping word in a , CTG with the sequence $b = \text{ACG}$. The mismatch between the starting nucleotides C and A, is enough to draw a conclusion that exact match of size 3 does not exist. The number of performed base-to-base comparisons until now equals 4.

In the following step has to be checked whether exist exact match of size $|b| - k = 3 - 1 = 2$, $k = 1$. For that purpose sequence a overlapping words of size 2: $\{\text{AC}, \text{CT}, \text{TG}\}$ have to be compared with sequence b overlapping words: $\{\text{AC}, \text{CG}\}$ of the same size, until exact match of size 2 is identified, certainly if exists. Two base-to-base comparisons are performed in order to identify the AC hit, which is the longest exact match for the DNA samples. The total number of performed base-to-base comparisons equals 6.

The starting and final positions of the hit in a and b , are stored in a data vector, $v = [(p_{a,s}, p_{a,f}, p_{b,s}, p_{b,f}), \dots]$, such as: $p_{a,s}$ is the starting position of the hit in a , $p_{a,f}$ is the final position of the hit in a , $p_{b,s}$ is the starting position of the hit in b and $p_{b,f}$ is the final position of the hit in b . After identifying the hit AC, the structure of the vector is $v = [(0, 2, 0, 2)]$.

Applying the same procedure, out of the hit (hits) being identified, all non-crossing and exact hits can be identified (fig. 1). Searching for hit between TG and G, the G (Guanine) hit is reported, what requires two additional base-to-base comparisons. The hit is also tracked in the data vector, being sorted in ascending order, regarding the starting positions of the hits being identified, each time its structure changes. For the sample sequences, the final structure of the data vector is $v = [(0, 2, 0, 2), (3, 3, 2, 2)]$. The DNA alignment is build, based on the data stored in v .

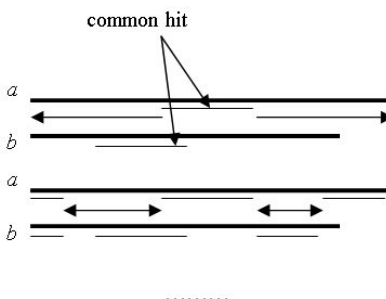


Fig. 1. Searching for hits

Comparison with the Dynamic Programming and Generalized Suffix Tree

The longest common substring for two strings can be also found by dynamic programming or generalized suffix tree (Gusfield, 1999). Applying dynamic programming requires $O(|a| \times |b|)$ time and space. To find the longest common substring for the DNA samples: $a = \text{ACTG}$ and $b = \text{ACG}$, dynamic programming matrix $[m_{i,j}]_{0 \leq i \leq |a|, 0 \leq j \leq |b|}$ has to be filled out, applying equation (1). The longest diagonal of increasing numbers reveals the longest common substring (fig. 2).

$$\begin{cases} m_{0,j} = 0, m_{i,0} = 0 \\ m_{i,j} = m_{i-1,j-1} + 1, \text{ if } a_i = b_j \\ 0, \text{ if } a_i \neq b_j \end{cases} \quad (1)$$

| | b | A | C | G |
|-----|-----|---|---|---|
| a | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 0 | 0 |
| C | 0 | 0 | 2 | 0 |
| T | 0 | 0 | 0 | 0 |
| G | 0 | 0 | 0 | 1 |

Fig. 2. The longest common substring for the DNA samples: $a = \text{ACTG}$ and $b = \text{ACG}$

In this case $|a| \times |b| = 4 \times 3 = 12$ base-to-base comparisons have to be performed and memory map of size 4×3 has to be reserved. Following the proposed search method, 6 base-to-base comparisons are required to find the longest common substring and the size of the data vector equals 8. Obviously for DNA sequences with high percentage of base identity, applying the proposed approach, the number of base-to-base comparisons and memory required can be reduced up to near linear. The total

number of base-to-base comparisons performed to identify all exact and non-crossing hits equals 8, what is less than the number of base-to-base comparisons performed by the Needleman-Wunsch or Smith-Waterman algorithm, that require $|a| \times |b| = 4 \times 3 = 12$ base-to-base comparisons.

Employing generalized suffix tree, the longest common substring can be found in $O(|a| + |b|)$ time, by merging the both strings into one and searching for the longest repeated substring. By merging the sample sequences $a = \text{ACTG}$ and $b = \text{ACG}$ in one, being separated within by the separator 'N', the string ACTGNACG is obtained. The longest repeating substring within the previous string is the substring AC . Holding a generalized suffix tree in the memory might cause memory overload, especially if large DNA sequences have to be aligned. The proposed approach eliminates the possibility of memory overload, when aligning DNA sequences with high percentage of base identity.

DNA alignment

Alignment's formation is discussed for the DNA samples: $a = \text{AGCCAAGCTACTT}$ and $b = \text{AGGAATGCTATT}$. The longest hit, found according to the proposed methodology, is GCTA . The positions of the hit in the sequences are stored in the vector v , $v = [(6,9,6,9)]$.

DNA left and right positioned regions, out of the hit are searched Fig. 3, in order to find hits within. The hit found within the left positioned regions: AGCCAA and AGGAAT is AG , whose positions are added in the vector v , $v = [(6,9,6,9), (0,1,0,1)]$. Since vector v structure has been changed, the vector is sorted in ascending order, regarding hits' starting positions, $v = [(0,1,0,1), (6,9,6,9)]$. The hit found within the right positioned regions: CTT and TT , is TT , whose positions are also added in the vector v , $v = [(0,1,0,1), (6,9,6,9), (11,12,10,11)]$.

Finally, according to the proposed methodology, a search for hit within regions CCAA and GAAT is performed Fig. 3. The hit AA is reported. Its positions in the sequences are also added in the vector v , $v = [(0,1,0,1), (6,9,6,9), (11,12,10,11), (4,5,3,4)]$. Once again the structure of the vector has been changed, causing vector v rearrangement in ascending order, regarding the starting positions of the matching regions, $v = [(0,1,0,1), (4,5,3,4), (6,9,6,9), (11,12,10,11)]$.

Vector $v = [v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, \dots, v_{4i-4}, v_{4i-3}, v_{4i-2}, v_{4i-1}]$ is obtained, being further processed. If $v_{k-8}v_{k-7}v_{k-6}v_{k-5}$ and $v_{k-4}v_{k-3}v_{k-2}v_{k-1}$ are two consecutive hits, the differences: $v_{k-4} - v_{k-7} - 1$ and $v_{k-2} - v_{k-5} - 1$ are calculated. If $v_{k-4} - v_{k-7} - 1 - (v_{k-2} - v_{k-5} - 1) > 0$, then $|v_{k-4} - v_{k-7} - 1 - (v_{k-2} - v_{k-5} - 1)|$ gaps are inserted in the sequence b , following the hit $v_{k-6}v_{k-5}$. If

$v_{k-4} - v_{k-7} - 1 - (v_{k-2} - v_{k-5} - 1) < 0$, then $|v_{k-4} - v_{k-7} - 1 - (v_{k-2} - v_{k-5} - 1)|$ gaps are inserted in the sequence a , following the hit $v_{k-8}v_{k-7}$. Inserting $|v_{k-4} - v_{k-7} - 1 - (v_{k-2} - v_{k-5} - 1)|$ gaps in one of the sequences, implies shift of the matching regions for $|v_{k-4} - v_{k-7} - 1 - (v_{k-2} - v_{k-5} - 1)|$ positions, in the sequence where the gaps have been inserted. Therefore, if $|v_{k-4} - v_{k-7} - 1 - (v_{k-2} - v_{k-5} - 1)|$ gaps are inserted in the sequence a , the positions of the matching regions: $v_{k-4}v_{k-3}, v_kv_{k+1}, \dots, v_{4i-4}v_{4i-3}$ are increased for $|v_{k-4} - v_{k-7} - 1 - (v_{k-2} - v_{k-5} - 1)|$. If $|v_{k-4} - v_{k-7} - 1 - (v_{k-2} - v_{k-5} - 1)|$ gaps are inserted in the sequence b , the positions of the matching regions: $v_{k-2}v_{k-1}, v_{k+2}v_{k+3}, \dots, v_{4i-2}v_{4i-1}$ are increased for $|v_{k-4} - v_{k-7} - 1 - (v_{k-2} - v_{k-5} - 1)|$.

According to the previous, two bases 4-1-1=2, separate hits: AG and AA in the sequence a and one base 3-1-1=1 separates the same hits in the sequence b , what implies that one gap, 2-1=1 should be inserted in the sequence b , in order to align regions: AG and AA. Since one gap is inserted in the sequence b , all following hits in the same sequence are shifted for one place, causing vector v change, $v = [(0,1,0,1), (4,5,4,5), (6,9,7,10), (11,12,11,12)]$.

The next pair of neighboring hits is processed: AA and GCTA. Zero bases, 6-5-1=0, separate hits: AA and GCTA in the sequence a and one base 7-5-1=1, separates the same regions in the other sequence, what implies that one gap, 0-1=-1 should be now inserted in the sequence a , after the hit AA. Since one gap is inserted in a , all following hits are shifted for one place, causing vector v update, $v = [(0,1,0,1), (4,5,4,5), (7,10,7,10), (12,13,11,12)]$.

Applying the same logic, one gap is inserted in the sequence b , 12-10-1-(11-10-1)=1, following the hit GCTA, in order to align matching regions GCTA and TT. The final structure of the data vector is $v = [(0,1,0,1), (4,5,4,5), (7,10,7,10), (12,13,12,13)]$.

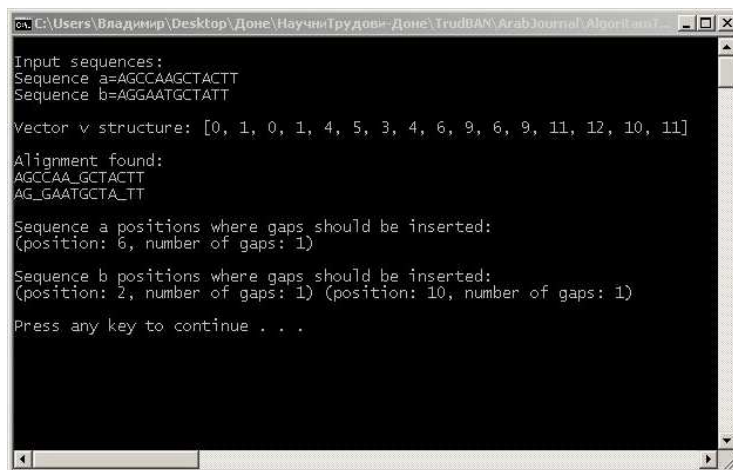
AGCCAAGCTACTT
AGGAATGCTATT

AGCCAAGCTACTT
AGGAATGCTATT

AGCCAAGCTACTT
AGGAATGCTATT

AGCCAAGCTACTT
AGGAATGCTATT

Fig. 3. Searching for exact and non-crossing hits



```

C:\Users\Владимир\Desktop\Дона\НаучниТрудови\Дона\TrudBAN\ArabJournel\Algoritmi
Input sequences:
Sequence a=AGCCAAAGCTACTT
Sequence b=AGGAATGCTATT

Vector v structure: [0, 1, 0, 1, 4, 5, 3, 4, 6, 9, 6, 9, 11, 12, 10, 11]

Alignment found:
AGCCAA_GCTACTT
AG_GAATGCTA TT

Sequence a positions where gaps should be inserted:
(position: 6, number of gaps: 1)

Sequence b positions where gaps should be inserted:
(position: 2, number of gaps: 1) (position: 10, number of gaps: 1)

Press any key to continue . . .
  
```

Fig. 4. DNA alignment

The output of the implementation of the proposed methodology in C++, taking the previous DNA segments as input sequences is shown on Fig. 4. Despite the alignment, vector v initial structure and the positions where gaps are inserted, are also identified.

RESULTS AND DISCUSSIONS

Different size homologous *Lactococcus Phage ASCC 473* and *Lactococcus Phage ASCC 476* fragments were aligned, according to the proposed methodology, on Acer Aspire 5570Z computer, with T2080 processor at 1.73 GHz and 1526 MB RAM. The primary structure of the segments was obtained from EMBL-EBI database. Alignments' time complexity was analyzed, Table 1 third column, having parallelized matching regions search and vector v sorting. The function: $kn \log_2 m$, such as: k is the number of hits, n is the length of the larger DNA fragment and m is the length of the shorter DNA fragment, has been identified as a best fitting function for the running times samples, Table 1 third column, Fig. 5. For homologous DNA sequences, an improved time complexity compared to the dynamic programming based implementations, such as Needleman-Wunsch algorithm is obtained: $O(kn \log_2 m)$, Fig. 5. The previous is justified by the fact, that for homologous DNA sequences, the number of matching regions is less than the number of bases of the shorter DNA sequence, $k \ll m, k \log_2 m < m$, wherefrom the inequality: $O(kn \log_2 m) < O(nm)$ is obtained.

The space complexity was also analyzed, on the previous samples, Table 1 fourth column. Vector v size - k , which represents the required memory, is always less

than the number of bases of the shorter DNA sequence - m , Table 1 fourth column, Fig. 6. Accordingly, the space complexity is linear: $O(k)$, $O(k) < O(m)$, Fig. 6.

Table 1. Time and space complexity analysis

| Sequence a | Sequence b | Running time (s) | Vector v size - k |
|---|--|------------------|-----------------------|
| Lactococcus Phage ASCC 473, base range:[1,45] | Lactococcus Phage ASCC 476, base range: [1,40] | 0,039 | 12 |
| Lactococcus Phage ASCC 473, base range:[1,50] | Lactococcus Phage ASCC 476, base range: [1,47] | 0,072 | 12 |
| Lactococcus Phage ASCC 473, base range:[1,60] | Lactococcus Phage ASCC 476, base range: [1,55] | 0,09 | 16 |
| Lactococcus Phage ASCC 473, base range:[1,65] | Lactococcus Phage ASCC 476, base range: [1,62] | 0,143 | 20 |
| Lactococcus Phage ASCC 473, base range:[1,70] | Lactococcus Phage ASCC 476, base range: [1,68] | 0,135 | 20 |
| Lactococcus Phage ASCC 473, base range:[1,75] | Lactococcus Phage ASCC 476, base range: [1,71] | 0,159 | 24 |
| Lactococcus Phage ASCC 473, base range:[1,80] | Lactococcus Phage ASCC 476, base range: [1,76] | 0,2 | 24 |
| Lactococcus Phage ASCC 473, base range:[1,90] | Lactococcus Phage ASCC 476, base range: [1,86] | 0,218 | 24 |

Running time (s)

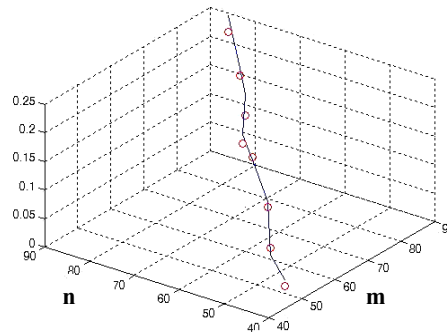


Fig. 5. Running time samples

Vector v size - k

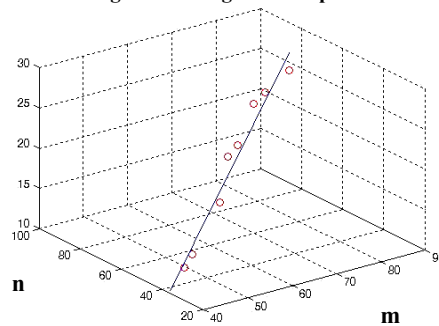


Fig. 6. Memory samples

CONCLUSION

A new DNA alignment methodology is presented, based on fast search and identification of all exact and non-crossing hits. Its time complexity is $O(kn \log_2 m)$, being faster than the standard dynamic programming based implementations, which require $O(nm)$ time. Also the space requirement is linear, what is welcomed, especially if large sequences have to be aligned. The methodology is applicable for DNA sequences of approximately same size, with high percentage of base identity.

REFERENCES

- Altschul S., Gish W., Miller W., Myers E., Lipman D.1990. Basic local alignment search tool. *Journal of Molecular Biology*. 215: 403–410.
- Bray N., Dubchak I., Pachter L.2003. AVID: A global alignment program. *Genome research*. 13: 97-102.
- Cédric N., Higgins D., Heringa J. 2000. T-Coffee: A novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*. 302: 205-218.
- Chao K., Pearson W., Miller W. 1992. Aligning two sequences within a specified diagonal band. *Bioinformatics/computer Applications in The Biosciences – BIOINFORMATICS*. 8: 481-487.
- Fickett W.1984. Fast optimal alignment. *Nucleic Acids Res*.12: 175-179.
- Gotoh O.1982. An improved algorithm for matching biological sequences. *Journal of Molecular Biology*. 162: 705–708.
- Gusfield D. 1999. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, Cambridge University Press, USA.
- Lipman D., Pearson W. 1985. Rapid and sensitive protein similarity searches. *Science*. 227: 1435–1441.
- Morgenstern B., Atchley W., Hahn K., Dress A.1998. Segment-based scores for pairwise and multiple sequence alignments, pp. 115-121. In: *Proceedings of the Sixth International Conference on Intelligent Systems for Molecular Biology*. Montreal, Canada.
- Morgenstern B.1999. DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*. 15: 211-218.
- Needleman S., Wunsch C.1970. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology*. 48: 443-453.
- Shen S., Yang J., Yao A., Hwang P.2002. Super pairwise alignment (SPA): an efficient approach to global alignment for homologous sequences. *Journal of Computational Biology*. 9: 477-486.
- Stojanov D., Koceski S., Mileva A.2013. FLAG: Fast Local Alignment Generating Methodology. *Romanian Biotechnological Letters*. 18: 7881-7888.
- Stojanov D., Mileva A., Koceski S.2012. A New, Space-Efficient Local Pairwise Alignment Methodology. *Advanced Studies in Biology*. 4: 85-93.
- Thompson J., Higgins D., Gibson T.1994. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic acids research*. 22: 4673-4680.
- Ukkonen E.1985. Algorithms for approximate string matching. *Information and Control*. 64: 100-118.