Comparing DNA Applying Dynamic Programming: A Historical Review

Done Stojanov

Faculty of Computer Science University ,,Goce Delčev", Štip, Macedonia

***Abstract* – In this paper we consider the application of dynamic programming in bioinformatics for comparing and aligning DNA sequences. Dynamic programming algorithms are old-fashioned in terms of computational complexity, but in terms of accuracy these algorithms are still irreplaceable.**

***Keywords* – dynamic programing, DNA, alignment, algorithms**

1. Introduction

Dynamic programming [1] is a commonly used optimization approach in almost all applied sciences. The idea behind this approach is the following: "*By joining solutions of divisions of the original problem, the solution of the original problem can be found*". The division is forwarded until is no longer possible and in fact this is the phase when unit solutions are obtained. Now, by returning back and joining by the way these solutions, the solution of the original problem is obtained. As typical implementations we can point out the computation of Fibonacci sequence and the shortest path in graph [2].

DOI: 10.18421/TEM64-32
https://dx.doi.org/10.18421/TEM64-32

Corresponding author: Done Stojanov,
Faculty of Computer Science University ,,Goce Delčev", Štip, Macedonia
Email: done.stojanov@ugd.edu.mk

Received: 27 September 2017
Accepted: 25 October 2017
Published: 27 November 2017

In bioinformatics, dynamic programming has been used for comparison of DNA content. Comparing pieces of coding DNA (genes) is an essential task in genetics, since relations and modifications relative to each other can be easily revealed. The similarity (difference) is a key indicator for likely (unlikely) homology and this is the base of each phylogenetic study. The process of comparing one piece of DNA relative to other is known as alignment. This process can be mathematically solved, but since DNA genome can be up to several giga-bases long (for example: a typical human genome contains 3×10^9 nucleotdes, organized in 23 chromosomes being distributed in each cell) this process can be facilitated and sped up by applying computer technology or computer programs which are implementations of already proposed algorithms.

Alignments performed based on dynamic programming always create exact solution upon predefined metrics. The metrics being exploited, impact the structure of the solution and there is always difficulty to choose the right metrics that reveal the real phylogenetic relation between the samples being compared. In most of the cases, applying dynamic programming imposes quadratic computational and storage requirements that may limit the practical implementation in some cases especially if longer pieces of DNA, such as chromosomes or even complete genomes are compared on machines with moderate hardware performance.

In order to save in computational expenses, heuristic algorithms have been also used for the same purpose. What is common for these algorithms is that they utilize the knowledge of the preprocessed data in order to cut in time of execution and memory required. However, in most of the cases partial and incomplete solutions are derived compared to dynamic programming algorithms and they are commonly used only for analysis of big data in eukaryotic DNA. In this group of algorithms we can enumerate: BLAST [3], FASTA [4], Pattern Hunter [5], BLAT [6], FLASH [7], MUMmer [8], AVID [9], GLASS [10], LAGAN [11] and SPA [12].

The main interest of this paper is to introduce the reader with the concepts of applying dynamic programming for pairwise analysis of DNA. By a historical viewpoint these algorithms are old-fashioned, but in terms of accuracy they are irreplaceable and this underlines its importance and impact in modern science.

2. Dynamic programming algorithms

Dynamic programming results in local or global solution. Local alignments are performed when highly conserved DNA motifs have to be revealed, while global solutions depict end-to-end mutual similarities and differences. Pairwise techniques or methods for comparison of two samples are subject of interest of this paper.

Metrics has to be specified in advance and in the most cases this assumes: $s(a_i, b_j) > 0$ an award or positive score is assigned for pairing equal nucleotides$(a_i = b_j)$, $s(a_i, b_j) \leq 0$ a non-positive score is assigned for a pair of mismatching nucleotides$(a_i \neq b_j)$ and a negative score $g, g < 0$ is assigned if nucleotide is paired with gap. Under these conditions we try to find optimal solution.

An optimal solution is the one for which the sum of all scores is maximal or there is no other alignment which over scores the optimal solution.

2.1. Needleman-Wunsch algorithm

This algorithm [13] finds end-to-end solutions. In order to compare sequence $\{a_i\}_{i=1}^{n}$ and $\{b_j\}_{j=1}^{m}$ $[s_{i,j}]_{(n+1)\times(m+1)}$ dynamic programming matrix has to be computed. After initializing cells $s_{i,0}, 0 \leq i \leq n$ and $s_{0,j}, 0 \leq j \leq m$ to $i \times g$ and $j \times g$ respectively, remaining cells are computed by applying equation (1). Cell $s_{i,j}$ tracks pointer to: $s_{i,j-1}$, $s_{i-1,j}$ or $s_{i-1,j-1}$ depending of which of the previous cells its value was derived.

$$s_{i,j} = \max\{s_{i,j-1} + g, s_{i-1,j} + g, s_{i-1,j-1} + s(a_i, b_j)\} \quad (1)$$

Once the matrix has been computed, the path of pointers from the rightmost cell at the bottom to the leftmost cell in the top row defines the structure of the optimal solution. Diagonal pointer indicates that nucleotides have to be paired, while vertical or horizontal pointer indicates gap insertion in the sequence being referred, Figure 1.

		A	T	...	G
	0	g	2×g		m×g
A	g	max(g + g , g + g ,0 + s(A, A))			
G	2×g	.....			
...	...				
G	n×g				

Figure 1. Needleman-Wunsch dynamic programming matrix

Since dynamic programming matrix of n rows and m columns is computed and stored in the memory, this algorithm requires $O(n \times m)$ time and memory.

2.2. Algorithm of Sellers

The algorithm of Sellers [14] works opposite to Needleman-Wunsch, but the purpose is the same. The end-to-end solution here is found with minimization of the total difference (distance) between the samples, while Needleman-Wunsch maximizes the total similarity.

According to Ulam [15] the difference between two sequences is the minimum number of steps required to transform one sample into the other. During this transformation, one nucleotide may substitute other or nucleotide insertion/deletion may occur. Applying equation (2) for computing the minimum distance between two DNA samples $\{a_i\}_{i=1}^{n}$and$\{b_j\}_{j=1}^{m}$, $O(n \times m)$ time and memeory are also required, Figure 2.

$$d_{i,j} = min \begin{cases} d_{i-1,j} + 1 \\ d_{i,j-1} + 1 \\ d_{i,j} + \delta, \delta = 0 \text{ ако } a_i = b_j;\ \delta = 1 \text{ ако } a_i \neq b_j \end{cases} \quad (2)$$

		A	T	...	G
	0	1	2		m
A	1	min(1 + 1,1 + 1,0 + 0)			
G	2	.....			
...	...				
G	n				

Figure 2. Distance matrix of Sellers

2.3. Smith-Waterman algorithm

Unlike Needleman-Wunsch and Sellers, Smith-Waterman [16] creates best local solution. This solution usually reveals the most conserved DNA motifs. In the initial phase, all cells in the first row and the first column of dynamic programming matrix are set up to 0, i.e. $s_{i,0} = 0, 0 \leq i \leq n$ and$s_{0,j} = 0, 0 \leq j \leq m$. The rest of the dynamic programming matrix is computed by applying equation (3). Pointers from each cell $s_{i,j}, 1 \leq i \leq n, 1 \leq j \leq m$ to

at least one of the neighboring cells in the same or the previous row wherefrom the values were obtained, are also tracked.

$$s_{i,j} = max \begin{cases} 0 \\ s_{i-1,j-1} + s(a_i, b_j) \\ s_{i-1,j} + g \\ s_{i,j-1+g} \end{cases} \quad (3)$$

The solution in this case is determined by the path of pointers starting from the cell with maximum value to the first 0-cell along that path, Figure 3. As well as in the previous cases, programming matrix of n rows and m columns is also computed, which means that $O(n \times m)$ time and memory is also required.

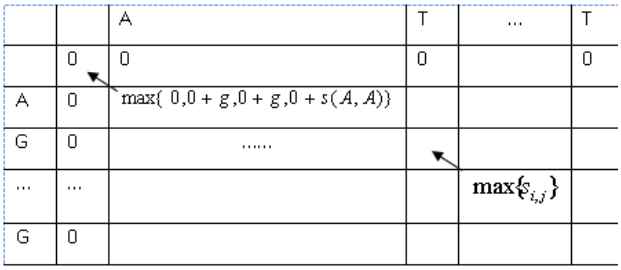

Figure 3. Smith-Waterman dynamic programming matrix

2.4. Waterman–Eggert algorithm

Waterman-Eggert [17] can be seen as an extension of Smith-Waterman or instead of searching for one best local solution, in this case ***k***-best or ***k***-highest scoring, local alignments are outputted.

After computing Smith-Waterman dynamic programming matrix, the highest scoring local alignment is outputted, which is defined by a path of pointers from the cell with maximum value to the first 0-cell, Figure 4. Once the best solution has been outputted, all cells along that path are set up to 0.

Now we search the modified matrix in order to find the new highest scoring local alignment, which in fact is going to be the second of the k-best local alignments we search for, Figure 5. After printing and this alignment, all cells through the path of the alignment are set up also to 0.

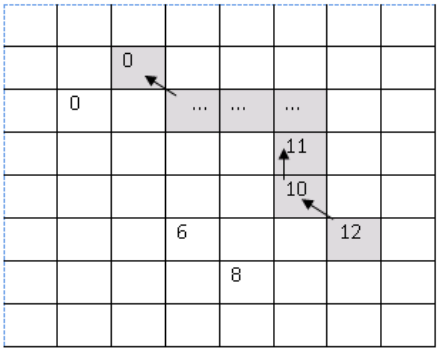

Figure 4. First best solution

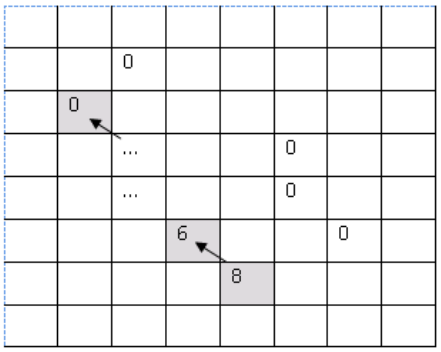

Figure 5. Second best solution

The same procedure is repeated until k-highest scoring alignments are identified. Please note that the paths of the alignments should not over cross, or in other words each of the identified solutions is independent.

2.5. Diagonal alignments: attempts to improve computational complexity

In order to improve the computational performances, algorithms that perform diagonal alignments were proposed.

The idea that stands behind this group of algorithms is the fact that in most of the cases the optimal alignment is described by a path of pointers that converges to the main diagonal of dynamic programming matrix. Since it is not likely the path of the pointers of the optimal solution to goes through the farthest cells relative to the main diagonal, these cells do not have to be computed what is going to improve the computational performance.

Given DNA sequences $\{a_i\}_{i=1}^{n}$ and $\{b_j\}_{j=1}^{m}$ that have to be aligned, only cells $s_{i,j}$ for which $|i - j| \leq \frac{k}{2}$ are computed, where k is the length of the diagonal bend over the main diagonal in dynamic programming matrix, equation (4), Figure 6.

Cells out of the diagonal bend and they are $s_{i,j}$ for which $|i-j| > \frac{k}{2}$ are not computed and in order not to affect the accuracy of the computation of the cells in the diagonal bend they are set up to large negative values, usually $-\infty$, Figure 6.

$$s_{i,j} = max \begin{cases} s_{i-1,j-1} + s(a_i, b_j) \\ s_{i-1,j} + g, \text{ако } |i-1-j| \leq \frac{k}{2} \\ s_{i,j-1} + g, \text{ ако } |i-j+1| \leq \frac{k}{2} \end{cases} \quad (4)$$

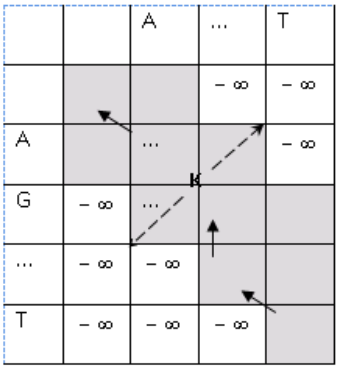

Figure 6. Alignment within diagonal band

However, this approach may be applied only for comparison of highly similar DNA samples that guarantee that the cut in the computational steps will not affect the accuracy of the solution.

In the list of algorithms that apply alignments within limited diagonal band we can enumerate: Ficket [18], Ukkonen [19] and Chao [20].

2.6. Attempts to linearize memory complexity

Dynamic programming algorithms require quadratic $O(n \times m)$ space during the execution, i.e. the memory requirement is proportional to the product of the length of the sequences being aligned. This may limit the application of dynamic programming algorithms only to SSRs (*Short Sequence Reads*), i.e. they may be inapplicable to longer DNA samples such as chromosomes or even complete genomes.

Hirschberg [21] first considered this problem. Based on dynamic programming, in 1990 Huang [22] proposed the first memory linear algorithm, whose complexity is proportional to the sum of the lengths of the sequences being aligned. This algorithm may be also applied for analysis and identification of SSRs (*Short Sequence Repeats*) in longer DNA samples.

The score of the optimal end-to-end alignment is the value of the rightmost cell at the bottom of dynamic programming matrix. The memory expense for computing this cell can be linearized, if instead of holding the entire dynamic programming matrix in the memory, two by two rows are dynamically computed, swapped and tracked in the memory. This is possible since $s_{i,j}$ is affected only by cells $s_{i,j-1}, s_{i-1,j}, s_{i-1,j-1}$ which are values that are placed in the same or the previous row relative to the row that contains $s_{i,j}$.

This means that we first compute the first and the second row of dynamic programming matrix. Next the third row is computed by using the values from the second row. Since the values in the first row do not affect any of the values in the third row, this row is removed from the memory and so on, such as at the end only the last two rows of dynamic programming matrix are computed and kept in the memory. The rightmost cell in the last row is the sore of the optimal solution and this is the way of how it can be computed with linear memory expense.

Since applying this approach we kept in the memory only two rows and not the entire dynamic programming matrix, the path of the optimal alignment has been lost and we can't print the structure of the optimal solution. In order to overcome this problem we can apply the *divide and conquer* strategy.

The idea is to find an intersection (v, u), such as by merging the optimal alignments of the subsequences: $(a_1 \dots a_u;\ b_1 \dots b_v)$ and $(a_{u+1} \dots a_n;\ b_{v+1} \dots b_m)$ the optimal alignment of the sequences $\{a_i\}_{i=1}^{n}$ and $\{b_j\}_{j=1}^{m}$ can be obtained. The intersection (v, u) divides the dynamic programming matrix in four quadrants, Figure 7. The north-east and the south-west quadrant are rejected, since the path of the optimal solution does not pass these regions. This procedure is recursively evoked for sub north-east and sub south-west quadrants, until nucleotide-to-nucleotide alignment or nucleotide-to-gap alignments. Now by returning back and merging sub-problems' solutions, the end-to-end alignment is reconstructed.

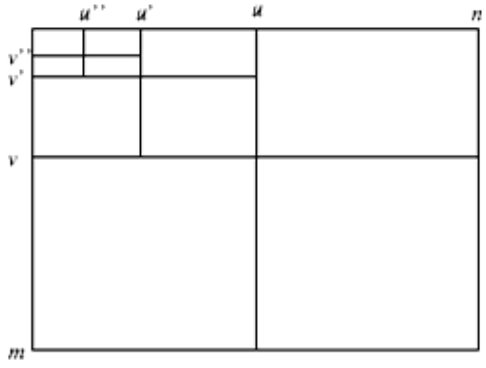

Figure 7. Applying the divide and conquer strategy

3. Discussion

Table 1. summarizes the key features of the algorithms being discussed in the previous section. The computational complexities are also bookmarked.

Table 1. Features of dynamic programming algorithms

Algorithm	Key features
Needleman-Wunsch	$O(n \times m)$ time and memory complexity, performs end-to-end alignments
Sellers	$O(n \times m)$ time and memory complexity, performs global alignments based on minimizing the distance between the samples
Smith-Waterman	$O(n \times m)$ time and memory complexity, performs local alignments
Diagonal alignments	$O(n \times k)$ time and memory complexity, where k is the length of the diagonal band, usually $k < m$
Memory linear algorithms: Huang	$O(n + m)$ memory complexity, suitable also for detection of short sequence repeats

4. Conclusion

Several algorithms for pairwise alignment of DNA sequences were discussed in this paper. These algorithms are straightforward application of dynamic programming. Despite the fact that most of them have quadratic computational complexities, there were also tries to linearize the computational performance in order to become applicable for analysis of chromosomes and complete genomes on regular computers. However, the computational performance comes to the second place when exact mutations have to be revealed, what makes this group of algorithms still very important as they used to be.

Acknowledgements

This study was supported by the research project Development of Secure and Reliable Techniques for Data Communication, funded by ,,Goce Delčev" University in Štip.

References

[1]. Bellman, R. (1952). On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, *38*(8), 716-719.

[2]. Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, *1*(1), 269-271.

[3]. Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of molecular biology*, *215*(3), 403-410.

[4]. Lipman, D. J., & Pearson, W. R. (1985). Rapid and sensitive protein similarity searches. *Science*, *227*(4693), 1435-1441.

[5]. Ma, B., Tromp, J., & Li, M. (2002). PatternHunter: faster and more sensitive homology search. *Bioinformatics*, *18*(3), 440-445.

[6]. Kent, W. J. (2002). BLAT—the BLAST-like alignment tool. *Genome research*, *12*(4), 656-664.

[7]. Califano, A., & Rigoutsos, I. (1993, June). FLASH: A fast look-up algorithm for string homology. In *Computer Vision and Pattern Recognition, 1993. Proceedings CVPR'93., 1993 IEEE Computer Society Conference on* (pp. 353-359). IEEE.

[8]. Delcher, A. L., Kasif, S., Fleischmann, R. D., Peterson, J., White, O., & Salzberg, S. L. (1999). Alignment of whole genomes. *Nucleic acids research*, *27*(11), 2369-2376.

[9]. Bray, N., Dubchak, I., & Pachter, L. (2003). AVID: A global alignment program. *Genome research*, *13*(1), 97-102.

[10]. Batzoglou, S., Pachter, L., Mesirov, J. P., Berger, B., & Lander, E. S. (2000). Human and mouse gene structure: comparative analysis and application to exon prediction. *Genome research*, *10*(7), 950-958.

[11]. Brudno, M., Do, C. B., Cooper, G. M., Kim, M. F., Davydov, E., Green, E. D., ... & NISC Comparative Sequencing Program. (2003). LAGAN and Multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA. *Genome research*, *13*(4), 721-731.

[12]. Shen, S. Y., Yang, J., Yao, A., & Hwang, P. I. (2002). Super pairwise alignment (SPA): an efficient approach to global alignment for homologous sequences. *Journal of Computational Biology*, *9*(3), 477-486.

[13]. 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.

[14]. Sellers, P. H. (1974). An algorithm for the distance between two finite sequences. *Journal of Combinatorial Theory, Series A*, *16*(2), 253-258.

[15]. Ulam, S. M. (1972). Some combinatorial problems studied experimentally on computing machines. *Zaremba SK, Applications of Number Theory to Numerical Analysis*, 1-3.

[16]. Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of molecular biology*, *147*(1), 195-197.

[17]. 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.

[18]. Fickett, J. W. (1984). Fast optimal alignment. *Nucleic acids research*, *12*(1), 175-179.

[19]. Ukkonen, E. (1985). Algorithms for approximate string matching. *Information and control*, *64*(1-3), 100-118.

[20]. Chao, K. M., Pearson, W. R., & Miller, W. (1992). Aligning two sequences within a specified diagonal band. *Bioinformatics*, *8*(5), 481-487.

[21]. Hirschberg, D. S. (1975). A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, *18*(6), 341-343.

[22]. Huang, X., Hardison, R. C., & Miller, W. (1990). A space-efficient algorithm for local similarities. *Bioinformatics*, *6*(4), 373-381.