Ss. Cyril and Methodius University in Skopje
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

2014

Proceedings of the 11th International Conference for Informatics and Information Technology

Held at Hotel Molika, Bitola, Macedonia
11-13th April, 2014

Editors:
Vangel V. Ajanovski
Gjorgji Madjarov

ISBN 978-608-4699-04-0

A Short Survey of Pair-wise Sequence Alignment Algorithms

Done Stojanov, Aleksandra Mileva
Faculty of Computer Science, UGD
Štip, Macedonia
{done.stojanov, aleksandra.mileva}@ugd.edu.mk

***Abstract* - We give a short survey of several pair-wise local and global sequence alignment algorithms, together with their comparative analysis. The analysis includes type of the algorithm, its time and space complexity, main characteristics, application for local or global alignment, is the algorithm heuristic or optimal, etc.**

Keywords — Needleman-Wunsch, Smith-Waterson, AVID, MUMmer, BWT-SW, Vmatch

I. INTRODUCTION

One of the oldest and most performed computational task in bioinformatics is sequence alignment. Sequence alignment can be used for multiple purposes, like: annotation of newly sequenced genomes, estimation of evolutionary distances, for producing an approximation and simplification of the history of mutations and evolutionary events, etc. The results from alignment show matching bases, positions where base substitutions has occurred and positions of bases' deletions and insertions. The goal is to find the alignment with optimal matching score. For alignment, a scoring function is used, awarding matches and penalizing mismatches and gaps.

Algorithms for sequence alignment can be optimal or heuristic. The number of possible alignments of two sequences grows exponentially with the lengths of the sequences.

Recently there are many algorithms for short-read alignment against a single reference, which are not subject of this paper.

In this paper we give a comparative analysis of several known local and global pair-wise sequence alignment algorithms. Some good surveys can be found in [7, 16, 23].

II. DEFINITIONS

Let Σ be a finite alphabet. Let A and B be the two sequences that have to be aligned, and let $A = a_1 a_2 \dots a_n$ and $B = b_1 b_2 \dots b_m$, where $a_i, b_j \in \Sigma$.

For applications in bioinformatics, three most used alphabets are DNA alphabet $\Sigma = \{A, C, G, T\}$, RNA alphabet $\Sigma = \{A, C, G, U\}$, and amino acid alphabet $\Sigma = \{A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V\}$ with 20 letters. So, sequences are DNA, RNA and proteins.

A *global pairwise alignment* S of two sequences A and B results from inserting gaps in A or B or in the both sequences, and after inserting, both sequences are with same length. This means that all letters and gaps in each sequence have to be aligned. Two gaps can not be aligned, because the semantic meaning of a gap is base deletion in the first sequence and base insertion in the other sequence. Global pairwise alignment, can be generalized to *multiple alignment* of a set $\{A_1, A_2, \dots, A_k\}$ of k sequences, which results from inserting gaps in each sequence A_i, so after inserting, all k sequences are with same length. Multiple alignment algorithms are not subject of this paper.

A *local pairwise alignment* S of two sequences A and B of different lengths, identifies local regions of similarity, or matching substrings between them.

Alignment can be seen as simplified representation of the evolutionary history that separates two sequences A and B. Allowed mutations are: simple point mutations, where single character change in the sequence occurs; insertions, where one or more consecutive characters are inserted in the sequence; and deletions, where one or more consecutive characters are removed from the sequence. Other types of mutations, as duplications and rearrangements are also possible, but are not included in the algorithms for sequence alignment.

In order to find the optimal alignment, a score for match, mismatch and gap insertion is defined. One can use PAM approach [11], BLOSUM approach [17] or by using simple scoring function δ which assigns different values, δ_1 and δ_2 for each character match and mismatch in the sequences, and assigns p as gap penalty for each gap. If the gap penalty for the first position of substring of gaps is different from subsequent positions, we speak about *affine gap penalty*, otherwise, if it is same for all positions of gaps, we speak about *linear gap penalty*. If alignment has gaps, it is *gapped alignment*, opposite to *un-gapped alignment*, which is without gaps.

Algorithms for pairwise sequence alignment can be divided in two distinctive classes: optimal and heuristic algorithms. *Optimal algorithms* always find the optimal alignment/s, and these algorithms are deterministic. All optimal algorithms for sequence alignment use the technique of dynamic programming, which is applicable because every partition of the optimal alignment of A and B is the optimal alignment of the corresponding subsequences from A and B. The main characteristic of these algorithms is the quadratic *O(nm)* time

complexity. They differ mostly by their space complexity, and best optimal algorithms decreases the space complexity from quadratic to linear.

Heuristic algorithms promise to find reasonable good (suboptimal) alignments, or to find the optimal alignment reasonably often. For the sake of this reduction, they obtain linear time and/or space complexity. These algorithms work in two phases for local alignment. In the first, so called preprocessing phase, matching positions of the highly similar regions *X* and *Y* are identified as seeds, and in the second phase, the seeds are extended to local alignment and the scoring function is calculated. Usually, not every initial seed is extended to full alignment, instead, many of them are discarded by filtering, which results in the lower runtime. Also, for heuristic algorithms exist a fraction of good alignments for which no seed is found, and they form false negative rate. Bigger false negative rate, means lower sensitivity for the algorithm. Li et al [24] showed that problem of choosing the optimal seed is NP-hard. Brejova et al [6] defined so called vector seed as generalization of many different seeds.

Definition 1 [6] *Vector seed* is an ordered pair $Q = (v, t)$, where $v = (v_1, v_2, \dots, v_l)$ is the seed vector of real numbers and t is the seed threshold value. $X = (x_1, x_2, \dots, x_n)$ hits the seed Q at position p if $\sum_{i=1}^{l}(v_i \cdot x_{p+i-1}) \geq t$. The number of nonzero positions of in the vector v is the *support* of the seed.

Similarly as seeded local alignment, exists so called anchor-based global alignment, based on heuristic approach that works in three phases. In the first phase, similar regions called anchors are identified, in the second, some anchors are chosen, and in the third phase, other regions between the anchors are aligned. The quality of the final alignment/s depends on the selection of anchors, and anchors are much more easily selected when the sequences are homologous, i.e. similar sequences with a common evolutionary origin. Two popular seed/anchor-based alignment techniques use hash table and suffix/prefix trie, such as suffix tree [35], enhanced suffix array [1] and FM-index [14]. When hash table is used, in the preprocessing phase, for each seed with length k in the sequence A its position in a given array is calculated by hashing function. Afterwards, one can go through the other sequence B and find the positions of seeds in linear time. The primary advantage of suffix/prefix tries is that alignment of multiple copies of the same substring is done once because they collapse on unique path in the trie, compared to hash tables, where the alignment is performed for each copy. Suffix trees can be built and searched in linear time and linear space. All three techniques of suffix/prefix tries has the same linear time complexity, but memory requirements differ - suffix tree requires 12-17B per nucleotide, suffix array requires 6.25B and FM-index requires 0.5-2B per nucleotide [23].

Some of the presented algorithms have parallel versions, like MPI-LAGAN [36], which is parallel version of LAGAN [8]. Also, some of the algorithms have their version for multiple sequence alignment, such as LAGAN again.

III. Algoritms and Their Analysis

First algorithms for local and global sequence alignment use dynamic programming for obtaining optimal alignment.

A. Optimal sequence alignment algorithms

Needleman-Wunsch pioneering algorithm [27] appeared in 1970 and it is an optimal algorithm for global sequence alignment, based on dynamic programming. It has unfavorable *O(nm)* time and space complexity. Scores are specified by the $(n+1) \times (m+1)$ matrix $S = [S_{i,j}]$ of similarity. The first row is obtained by the formula $S_{0,j} = p \cdot j$, and the first column is obtained by the formula $S_{i,0} = p \cdot i$. The values of the remaining cells are computed by the following formula:

$$S_{i,j} = max \begin{cases} S_{i-1,j} + p \\ S_{i,j-1} + p \\ S_{i-1,j-1} + \delta(a_i, b_j) \end{cases}$$

After completion of the matrix, the highest score is the optimal score, and it can be found in the last cell $S_{n,m}$. The optimal alignment is constructed by tracing back from the last cell to the cell $S_{0,0}$.

Other global sequence alignment algorithms try to reduce the space complexity to be less then quadratic. Hirschberg [18] algorithm is in fact, divide and conquer version of the previous algorithm, and it has linear space complexity. It stores only current and previous row of the Needleman-Wunsch matrix. In each step, the algorithm finds partitioning point (x, y), which divide the two sequences A and B into subsequences $A = A_l A_r$ and $B = B_l B_r$. This point provide obtaining an optimal global sequence alignment of A and B, by concatenating the optimal global alignments of subsequences A_l and B_l, and subsequences A_r and B_r.

Another improvement of Needleman-Wunsch is given by Fickett [15]. It tries to reduce the time complexity by reordering the calculation of matrix values, in order to avoid wasted computation.

Smith-Waterman [32] algorithm is based on Needleman-Wunch approach, without negative values in the cells, generating optimal gapped local alignment. It has the same quadratic time and space complexity. The first row and column of the $(n+1) \times (m+1)$ matrix $S = [S_{i,j}]$ are initialized to zero, and the other values are computed by:

$$S_{i,j} = max \begin{cases} S_{i-1,j} + p \\ S_{i,j-1} + p \\ S_{i-1,j-1} + \delta(a_i, b_j) \\ 0 \end{cases}$$

The optimal local alignment is obtained by tracing back from the cell with the highest score to the first zero cell.

B. Heuristic pair-wise local sequence alignment algorithms

Instead of finding one optimal alignment, Waterman and Eggert [34] came up with an idea of identifying *k* suboptimal local alignments, in *O(knm)* time and quadratic space. Huang and Miller [19] presented linear-space variant of the Waterman-Eggert algorithm.

FASTA [25, 29] is the first seed-based alignment algorithm based on hashing. In the first stage a look up for matching

substrings of length k (called hot-spots) is conducted. For indexing hot-spots between query and database sequence, a hash table is employed. Consecutive hot-spots form continuous diagonals in dynamic programming matrix. By choosing 10 best diagonal runs of consecutive hits from the matrix and performing gapped alignment in order to align perfectly matching regions, using variant of the Smith-Waterman algorithm, local gapped alignment is produced. With FASTA, seeds with length less then k could be missed. Since the algorithm employs matrix, the space complexity is *O(nm)*. The average time complexity of the algorithm is $O\left(\frac{nm}{|\Sigma|^k}\right)$.

BLAST (Basic Local Alignment Search Tool) [2, 3] is heuristic approach for searching a database of protein or DNA sequences against target query sequence. A list of similar sequences regarding the query is reported as an output. By breaking the query sequence in overlapping words of size k (the default value for k is 3), for each word at position p in the query a list of words of the same size, scoring at least T with the p-word is generated. During the second phase, the database is scanned for list words' hits. By extending hits in the both directions, until the score of the alignment significantly drops, a local pairwise alignment (ungapped or gapped, depending on whether gaps' insertion is permitted or not during the extension phase) is generated. BLAST has the same time complexity as dynamic programming algorithms *O(nm)*, but since non-significant local alignments are discarded, its running time is about 500 time faster than standard dynamic programming algorithms. Since lookup table of size $|\Sigma|^k$ is stored in the memory, BLAST overall space complexity is $O(|\Sigma|^k + nm)$.

BLAT (BLAST-Like Alignment Tool) [20] is pairwise alignment algorithm that runs about 500 times faster when aligning mRNA/DNA sequences and about 50 times faster when aligning protein sequences, compared to the pre-existing tools. Searching for exact or almost exact hits, BLAT is less sensitive than BLAST. Due to the reduced sensitivity, BLAT is not recommended tool for searching more distantly related sequences, but when it comes to closely related sequences, better time performance results are obtained in comparison to BLAST. During the search stage, three different strategies are used in order to find homologous regions: searching for perfect hits, allowing at least one mismatch between two hits and searching for multiple perfect matches, which are in close proximity to each other.

PatternHunter [26] introduced spaced seeds, building at first an index of A for this model of seeds. Spaced seeds require exact match on k positions, which do not have to be consecutive. It is more sensitive than BLAST and BLAT, with the same time and memory complexity.

BLASTZ [31] is the fastest of all algorithms in BLAST family. Main speedup is obtained by removing all substring repeats in the sequences. BLASTZ uses so called transition seeds of length k, with at most one transition from one to other character, which are extended in both directions without gaps, until score drops below some threshold value t_1. Afterwards, it performs gapped alignments called zones, with score above other threshold value t_2. For the regions between zones, the previous procedure is repeated with smaller value of k and smaller thresholds.

YASS [28] is a variant with tradeoff between FASTA and BLAST. It uses small exact repeats obtained by hashing as seeds and multiple seed criterion that allows an arbitrary number of possibly overlapping seeds. Afterwards, an extension is performed, using new criterion called group criterion, based on the total nucleotide size of the group.

BWT-SW [22] uses FM-index [14] and Burrows-Wheeler Transform (BWT) for emulating suffix trie and obtaining better storage complexity. Authors reported $O(n^{0.628}m)$ time complexity for ungapped alignment. They also stated "how to modify the dynamic programming to allow pruning but without jeopardizing the completeness".

FLAG [33] generates local alignment in linear time and space, when two homologous sequences are aligned. For each overlapping window, the longest matching region is found as a seed, being afterwards extended to local alignment. The algorithm uses a clever way, without have to perform too many unnecessary matching checks for obtaining the "best" local alignment.

C. Heuristic pair-wise global sequence alignment algorithms

MUMmer [12] is the first widely used and efficient anchor-based global alignment algorithm for two genome sequences. It uses suffix tree data structure to find maximal unique matches (MUMs) between A and B, providing $O(n+m)$ time complexity and linear space complexity. MUMs are unique if they occur exactly once in each of the sequences. After finding all MUMs, the algorithm sort them according to their position in the sequences, then picks the longest set of non-conflicting anchors, and aligns the regions between the chosen anchors with Smith-Waterman algorithm. It is open-source now and it has better computational time for homologous sequences, because in that case, Smith-Waterman works less. MUMmer additionally locates all single nucleotide polymorphisms, large inserts, significant repeats, tandem repeats and reversals. Its subsequent two versions are described in [13] and [21]. An additional option of MUMmer 3.0 is the identification of all maximal matches, including non-unique ones, what increases the storage and computational time for creating the output file.

GLASS (GLobal Alignment SyStem) [4] is slower, but more sensitive algorithm than MUMmer. It starts by finding all common K-mers in two sequences. For building alignments, only non-overlapping and non-crossing K-mers with score above some threshold value T are considered. For the regions between the K-mers, the same algorithm is performed recursively for smaller values of K (K takes one of the values 20, 15, 12, 9, 8, 7, 6 and 5). At the end, remaining regions are aligned using standard dynamic programming algorithm.

Another anchor-based global alignment algorithm that uses suffix tree, is AVID [5]. The suffix tree is built for the concatenation of two sequences with a special character N between them. The suffix tree is searched for maximal repeated substrings in linear time. A maximal repeated substring that crosses the boundary between the two sequences represents a maximal match between the two sequences. An anchor set is a collection of non-overlapping, non-crossing matches with

length at least half the length of the longest match. The matches are sorted at first, and for selecting good anchors with score above some threshold, a variant of Smith-Waterman algorithm is used, similarly to GLASS. For alignment of regions between the anchors, AVID makes a recursive calls to the same previous process, and previously discarded shorter matches are reconsidered for anchoring in the later rounds. The recursion ends when there are either no remaining bases to be aligned, or there are no significant matches in the remaining sequences. If these remaining sequences are short, they are aligned using the Needleman-Wunsch algorithm, but if they are long, the lack of anchor indicates no significant alignment between them. AVID has good performances only for homologous sequences.

LAGAN (Limited Area Global Alignment of Nucleotides) [8] heuristics use anchors build by chaining an ordered subset of local alignments, which are obtained by seeding strategy with allowed mismatches in the seed. Every local alignment is built from more than one short seeds using CHAOS algorithm [9]. Global alignment is obtained by applying CHAOS recursively in the areas with sparse anchors, so that each consecutive pair of anchors is separated by a distance smaller than a given maximum, which leads to one rough global map. Afterwards, the Needleman-Wunsch algorithm is performed on the limited area around the rough global map [8]. Sensitive anchoring scheme of LAGAN, makes it suitable for close and distantly related sequence pairs.

SPA (Super Pair-wise alignment) [30] is fast global alignment algorithm for homologous sequences, with reduced time and space complexity to $O(m)$, without sacrificing too much accuracy. By measuring the percent of local similarity within shifting window of size k, all positions where nucleotides' deletions occurred can be identified. If by addition of gap/s at concrete position/s, the percent of local similarity within the shifting window is increased over certain threshold, that the insertion of gap/s is permitted.

Vmatch [1] is a variant of MUMmer where suffix tree data structure is replaced with enhanced suffix array. The name enhanced suffix array stands for data structures built of suffix array and additional tables. The time complexity remains the same, but the memory requirements are drastically reduced. Sequence alignment is only one feature of the software Vmatch.

IV. CONCLUSIONS

Our analysis is summarized in the Table 1. There are several described algorithms that are not included, due to the fact that their time and storage complexity is difficult to be computed.

REFERENCES

[1] M. I. Abouelhoda, S. Kurtz, and E. Ohlebusch, "Replacing suffix trees with enhanced suffix arrays", Journal of Discrete Algorithms 2, 2004, pp. 53–86.

[2] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic local alignment search tool," Journal of Molecular Biology 215 (3), 1990, pp. 403–410.

[3] S. Altschul, T. L. Madden, A. A. Schoffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman, "Gapped BLAST and PSI-BLAST: A new generation of protein database search programs", Nucleic Acids Researsch 25**,** 1997, pp. 3389-3402.

[4] L. Batzoglou, J. Pachter, B. Mesirov, B. Berger, and E. S. Lander, "Human and mouse gene structure: comparative analysis and application to exon prediction," in RECOMB '00: Proc of the 4th Int'l Conference on Computational Molecular Biology, 2000, pp. 46-53.

[5] N. Bray, I. Dubchak, and L. Pachter, "AVID: A global alignment program", Genome research, 13(1), 2003, pp. 97-102.

[6] B. Brejova, D. G. Brown, and T. Vinar, "Vector seeds: an extension to spaced seeds", Journal of Computer and System Science 70(3), 2005, pp. 364-380.

[7] D. G. Brown, "A survey of sequence alignment", Computational Genomics: Current methods. Horizon Press, 2007; N. Stojanovic, ed., pp. 95-120.

[8] M. Brudno, C. B. Do, G. M. Cooper, M. F. Kim, E. Davydov, NISC Comparative Sequence Program, E. D. Green, A. Sidow, and S. Batzoglou, "LAGAN and Multi-LAGAN: Efficient tools for large-scale multiple alignment of genomic DNA", Genome Research 13, 2003, pp. 721-731.

[9] M. Brudno and B. Morgenstern, "Fast and sensitive alignment of large genomic sequences", in Proc of IEEE Computer Science Bioinformatics Conference, 2002, pp. 138-147.

[10] M. Burrows, and D. J. Wheeler, "A block-sorting lossless data compression algorithm", Technical Report 124, Digital Equipment Corporation. CA: Palo Alto, 1994.

[11] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt, "A model of evolutionary change in proteins", Atlas of Prot. Seq. and Struct. 5, 1978, pp. 345–352.

[12] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg, "Alignment of whole genomes", Nucleic Acids Research 27, 1999, pp. 2369-2376.

[13] A. L. Delcher, A. Phillippy, J. Carlton, and S. L. Salzberg, "Fast algorithms for large-scale genome alignment and comparison", Nucleic Acids Research 30, 2002, pp. 2478-2483.

[14] P. Ferragina, and G. Manzini, "Opportunistic data structures with applications", in Proceedings of the 41st Symposium on Foundations of Computer Science (FOCS 2000), Redondo Beach, CA, USA, 2000, pp. 390–398.

[15] J. W. Fickett, "Fast optimal alignment", Nucleic Acids Res. 12 (1), 1984, pp. 175-179.

[16] W. Haque, A. Aravind, and B. Reddy, "Pairwise sequence alignment algorithms – a survey", In Proceedings of the 2009 conference on Information Science, Technology and Applications (ISTA'09), 2009 pp.96-103.

[17] S. Henikoff, and J. G. Henikoff, "Automated assembly of protein blocks for database searching", Nucl. Acids Res. 19, 1991, pp. 6565–6572.

[18] D. Hirschberg, "A linear space algorithm for computing maximal common subsequences", Communications of the ACM 18(6), 1975, pp. 341–343.

[19] X. Huang, and W. Miller, "A time-efficient, linear-space local similarity algorithm", Advances in Applied Mathematics 12, 1991, pp. 337-357.

[20] W. J. Kent, "BLAT- the BLAST-like alignment tool", Genome Research 12 (4), (2002), pp. 656-664.

[21] S. Kurtz, A. Philippy, A. L. Delcher, M. Smoot, M. Shumway, C. Antonescu, and S. L. Salzberg, , "Versatile and open software for comparing large genomes," Genome Biology 5:R12, 2004.

[22] T. W. Lam, W. K. Sung, and S. L. Tam, "Compressed indexing and local alignment of DNA", Bioinformatics 24, 2008, 791–797.

[23] H. Li, N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing", Brief Bioinform. 11(5), 2010, pp. 473-483.

[24] M. Li, B. Ma, D. Kisman, and J. Tromp, "PatternHunter II: Highly Sensitive and Fast Homology Search", Journal of Bioinformatics and Com-putational Biology 2 (3), 2004, pp. 417–439.

[25] D. J. Lipman, and W. R. Pearson, "Rapid and sensitive protein similarity searches", Science 227 (4693), 1985, pp. 1435–1441.

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

[27] S. Needleman, and C. Wunsch, "A general method applicable to the search for similarities in the amino acid sequences of two proteins," Journal of Molecular Biology 48 (3), 1970, pp. 443-453.

[28] L. Noe and G. Kucherov, "YASS: enhancing the sensitivity of DNA similarity search", Nucleic Acids Research, 33, 2005, pp. 540-543.

[29] W. R. Pearson, and D. J. Lipman, "Improved tools for biological sequence comparison", Proceedings of the National Academy of Sciences of the United States of America 85 (8), 1988, pp. 2444–2448.

[30] S. Y. Shen, J. Yang, A. Yao, and P. I. Hwanq, "Super pairwise alignment (SPA): an efficient approach to global alignment for homologous sequences", Journal of Computational Biology 9(3), 2003, pp. 477-486.

[31] S. Schwartz, et al., "Human–Mouse Alignments with BLASTZ", Genome Research 13, 2003, pp. 103-107.

[32] T. Smith, and M. Waterman, "Identification of common molecular subsequences," Journal of Molecular Biology 147(1), 1981, pp. 195–197.

[33] D. Stojanov, S. Koceski, and A. Mileva, "FLAG: Fast Local Alignment Genereting methodology", Romanian Biotechnological Letters 18(1), 2013, pp. 7881-7888.

[34] M. Waterman, and M. Eggert, "A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons", Journal of Molecular Biology 197, 1987, pp. 723 - 728.

[35] P. Weiner, "Linear Pattern Matching Algorithms", In Proceedings of the 14th IEEE Symposium on Switching and Automata Theory, 1973, pp. 1–11.

[36] R. Zhang, H. Rangwala and G. Karypis, "Whole genome alignments using MPI-LAGAN", In Proceeding of IEEE International Conference on Bioinformatics and Biomedicine, 2008

.

Table 1. Analysis of local and global sequence alignment algorithms

Algorithm	**Year**	**Characteristics**	**Time complexity**	**Space complexity**	**Local vs global alignment**	**Optimal vs heuristic algorithms**
Needleman-Wunsch	1970	Dynamic programming	$O(nm)$	$O(nm)$	global	optimal
Hirschberg	1975	Dynamic programming with divide and conquer	$O(nm)$	$O(m)$	global	optimal
Smith-Waterman	1981	Dynamic programming	$O(nm)$	$O(nm)$	local	optimal
Waterman-Eggert	1987	Dynamic programming	$O(knm)$ k-number of suboptimal alignments	$O(nm)$	local	heuristic
Huang-Miller	1991	Dynamic programming	$O(nm)$	$O(m)$	local	heuristic
FASTA	1985	Consecutive seeds of length k and hash table	$O\left(\frac{nm}{\lvert\Sigma\rvert^k}\right)$	$O(nm)$	local	heuristic
BLAST	1990	Consecutive seeds of length k and hash table	$O(nm)$	$O(\lvert\Sigma\rvert^k + nm)$	local	heuristic
BLAT	2002	Seeds with allowed 0, 1, or 2 mismatches	$O(nm)$	$O(\lvert\Sigma\rvert^k + nm)$	local	heuristic
PatternHunter	2002	Spaced seeds and hash table	$O(nm)$	$O(\lvert\Sigma\rvert^k + nm)$	local	heuristic
BLASTZ	2003	Removed repeats, transition seeds	$O(nm)$	$O(\lvert\Sigma\rvert^k + nm)$	local	heuristic
BWT-SW	2008	FM-index	$O(n^{0.628}m)$ ungapped alignment		local	heuristic*
FLAG	2013	Extending of the longest consecutive seeds	$O(m)$* homologous sequences	$O(m)$* homologous sequences	local	heuristic
MUMmer	1999	Anchors and suffix tree	$O(n+m)$	$O(n+m)$	global	heuristic
AVID	2003	Anchors and suffix tree	$O(n+m)$	$O(n+m)$	global	heuristic
SPA	2003	Measure of local similarity	$O(m)$	$O(m)$	global	heuristic
Vmatch	2004	Anchors and enhanced suffix array	$O(n+m)$	$O(n+m)$	global	heuristic