Smith–Waterman algorithm

From Wikipedia, the free encyclopedia

The **Smith–Waterman algorithm** performs local sequence alignment; that is, for determining similar regions between two strings or nucleotide or protein sequences. Instead of looking at the total sequence, the Smith–Waterman algorithm compares segments of all possible lengths and optimizes the similarity measure.

The algorithm was first proposed by Temple F. Smith and Michael S. Waterman in 1981.[1] Like the Needleman–Wunsch algorithm, of which it is a variation, Smith–Waterman is a dynamic programming algorithm. As such, it has the desirable property that it is guaranteed to find the optimal local alignment with respect to the scoring system being used (which includes the substitution matrix and the gap-scoring scheme). The main difference to the Needleman–Wunsch algorithm is that negative scoring matrix cells are set to zero, which renders the (thus positively scoring) local alignments visible. Backtracking starts at the highest scoring matrix cell and proceeds until a cell with score zero is encountered, yielding the highest scoring local alignment. One does not actually implement the algorithm as described because improved alternatives are now available that have better scaling (Gotoh, 1982) [2] and are more accurate (Altschul and Erickson, 1986).[3]

Explanation [edit]

A matrix H is built as follows:

$$H(i,0) = 0,\ 0 \le i \le m$$

$$H(0,j) = 0,\ 0 \le j \le n$$

$$H(i,j) = \max \left\{ \begin{array}{ll} 0 & \\ H(i-1,j-1) + s(a_i,b_j) & \text{Match/Mismatch} \\ \max_{k \ge 1}\{H(i-k,j) + W_k\} & \text{Deletion} \\ \max_{l \ge 1}\{H(i,j-l) + W_l\} & \text{Insertion} \end{array} \right\},\ 1 \le i \le m, 1 \le j \le n$$

Where:

- a, b = Strings over the Alphabet Σ
- $m = \text{length}(a)$
- $n = \text{length}(b)$
- $s(a,b)$ is a similarity function on the alphabet
- $H(i,j)$ – is the maximum Similarity-Score between a suffix of a[1...i] and a suffix of b[1...j]
- W_i is the gap-scoring scheme

Example [edit]

- Sequence 1 = ACACACTA
- Sequence 2 = AGCACACA
- $s(a,b) = +2 \text{ if } a = b \text{ (match)},\ -1 \text{ if } a \ne b \text{ (mismatch)}$
- $W_i = -i$

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Català
Čeština
Deutsch
Español
فارسی
Français
Italiano
Polski
Português
Русский
Српски / srpski
ไทย
Edit links

$$H = \begin{pmatrix} & - & A & C & A & C & A & C & T & A \\ - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & 2 & 1 & 2 & 1 & 2 & 1 & 0 & 2 \\ G & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ C & 0 & 0 & 3 & 2 & 3 & 2 & 3 & 2 & 1 \\ A & 0 & 2 & 2 & 5 & 4 & 5 & 4 & 3 & 4 \\ C & 0 & 1 & 4 & 4 & 7 & 6 & 7 & 6 & 5 \\ A & 0 & 2 & 3 & 6 & 6 & 9 & 8 & 7 & 8 \\ C & 0 & 1 & 4 & 5 & 8 & 8 & 11 & 10 & 9 \\ A & 0 & 2 & 3 & 6 & 7 & 10 & 10 & 10 & 12 \end{pmatrix} \qquad P = \begin{pmatrix} & - & A & C & A & C & A & C & T & A \\ - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\ G & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \nwarrow & \uparrow \\ C & 0 & \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow \\ A & 0 & \nwarrow & \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\ C & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \nwarrow & \leftarrow & \leftarrow \\ A & 0 & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \leftarrow & \nwarrow \\ C & 0 & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \leftarrow & \leftarrow \\ A & 0 & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \uparrow & \nwarrow & \nwarrow \end{pmatrix}$$

To obtain the optimum local alignment, start with the highest value in the matrix (i,j). Then, go backwards to one of positions $(i-1, j)$, $(i, j-1)$, and $(i-1, j-1)$ depending on the direction of movement used to construct the matrix. This methodology is maintained until a matrix cell with zero value is reached.

In the example, the highest value corresponds to the cell in position (8,8). The walk back corresponds to (8,8), (7,7), (7,6), (6,5), (5,4), (4,3), (3,2), (2,1), (1,1), and (0,0),

Once finished, the alignment is reconstructed as follows: Starting with the last value, reach (i,j) using the previously calculated path. A diagonal jump implies there is an alignment (either a match or a mismatch). A top-down jump implies there is a deletion. A left-right jump implies there is an insertion.

For the example, the results are:

Sequence 1 = `A-CACACTA`

Sequence 2 = `AGCACAC-A`

Motivation [edit]

One motivation for local alignment is the difficulty of obtaining correct alignments in regions of low similarity between distantly related biological sequences, because mutations have added too much 'noise' over evolutionary time to allow for a meaningful comparison of those regions. Local alignment avoids such regions altogether and focuses on those with a positive score, i.e. those with an evolutionarily conserved signal of similarity. A prerequisite for local alignment is a negative expectation score. The expectation score is defined as the average score that the scoring system (substitution matrix and gap penalties) would yield for a random sequence.

Another motivation for using local alignments is that there is a reliable statistical model (developed by Karlin and Altschul) for optimal local alignments. The alignment of unrelated sequences tends to produce optimal local alignment scores which follow an extreme value distribution. This property allows programs to produce an expectation value for the optimal local alignment of two sequences, which is a measure of how often two unrelated sequences would produce an optimal local alignment whose score is greater than or equal to the observed score. Very low expectation values indicate that the two sequences in question might be homologous, meaning they might share a common ancestor.

The Smith–Waterman algorithm is fairly demanding of time: To align two sequences of lengths *m* and *n*, *O(mn)* time is required. Smith–Waterman local similarity scores can be calculated in O(m) (linear) space if only the optimal alignment needs to be found, but naive algorithms to produce the alignment require O(mn) space. A linear space strategy to find the best local alignment has been described.[4] BLAST and FASTA reduce the amount of time required by identifying conserved regions using rapid lookup strategies, at the cost of exactness.

An implementation of the Smith–Waterman Algorithm, SSEARCH, is available in the FASTA sequence analysis package from [4]. This implementation includes Altivec accelerated code for PowerPC G4 and G5 processors that speeds up comparisons 10–20-fold, using a modification of the Wozniak, 1997 approach,[5] and an SSE2 vectorization developed by Farrar[6] making optimal protein sequence database searches quite practical. A library, SSW, extends Farrar's implementation to return alignment information in addition to the optimal Smith-Waterman score.[7]

Accelerated versions [edit]

FPGA [edit]

Cray demonstrated acceleration of the Smith–Waterman algorithm using a reconfigurable computing platform based on FPGA chips, with results showing up to 28x speed-up over standard microprocessor-based solutions. Another FPGA-based version of the Smith–Waterman algorithm shows FPGA (Virtex-4) speedups up to 100x[8] over a 2.2 GHz Opteron processor.[9] The TimeLogic DeCypher and CodeQuest systems also accelerate Smith–Waterman and Framesearch using PCIe FPGA cards.

A 2011 Master's thesis [10] includes an analysis of FPGA-based Smith–Waterman acceleration.

GPU [edit]

Lawrence Livermore National Laboratory and the US Department of Energy's Joint Genome Institute implemented an accelerated version of Smith–Waterman local sequence alignment searches using graphics processing units (GPUs) with preliminary results showing a 2x speed-up over software implementations.[11] A similar method has already been implemented in the Biofacet software since 1997, with the same speed-up factor.[12]

Several GPU implementations of the algorithm in NVIDIA's CUDA C platform are also available.[13] When compared to the best known CPU implementation (using SIMD instructions on the x86 architecture), by Farrar, the performance tests of this solution using a single NVidia GeForce 8800 GTX card show a slight increase in performance for smaller sequences, but a slight decrease in performance for larger ones. However the same tests running on dual NVidia GeForce 8800 GTX cards are almost twice as fast as the Farrar implementation for all sequence sizes tested.

A newer GPU CUDA implementation of SW is now available that is faster than previous versions and also removes limitations on query lengths. See CUDASW++.

Eleven different SW implementations on CUDA have been reported, three of which report speedups of 30X.[14]

SIMD [edit]

In 2000, a fast implementation of the Smith–Waterman algorithm using the SIMD technology available in Intel Pentium MMX processors and similar technology was described in a publication by Rognes and Seeberg.[15] In contrast to the Wozniak (1997) approach, the new implementation was based on vectors parallel with the query sequence, not diagonal vectors. The company Sencel Bioinformatics has applied for a patent covering this approach. Sencel is developing the software further and provides executables for academic use free of charge.

A SSE2 vectorization of the algorithm (Farrar, 2007) is now available providing an 8-16-fold speedup on Intel/AMD processors with SSE2 extensions.[6] When running on Intel processor using the Core microarchitecture the SSE2 implementation achieves a 20-fold increase. Farrar's SSE2 implementation is available as the SSEARCH program in the FASTA sequence comparison package. The SSEARCH is included in the European Bioinformatics Institute's suite of similarity searching programs.

Danish bioinformatics company CLC bio has achieved speed-ups of close to 200 over standard software implementations with SSE2 on an Intel 2.17 GHz Core 2 Duo CPU, according to a publicly available white paper.

Accelerated version of the Smith–Waterman algorithm, on Intel and AMD based Linux servers, is supported by the GenCore 6 package, offered by Biocceleration. Performance benchmarks of this software package show up to 10 fold speed acceleration relative to standard software implementation on the same processor.

Currently the only company in bioinformatics to offer both SSE and FPGA solutions accelerating Smith–Waterman, CLC bio has achieved speed-ups of more than 110 over standard software implementations with CLC Bioinformatics Cube [*citation needed*]

The fastest implementation of the algorithm on CPUs with SSSE3 can be found the SWIPE software (Rognes, 2011),[16] which is available under the GNU Affero General Public License. In parallel, this software compares residues from sixteen different database sequences to one query residue. Using a 375 residue query sequence a speed of 106 billion cell updates per second (GCUPS) was achieved on a dual Intel Xeon X5650 six-core processor system, which is over six times more rapid than software based on Farrar's 'striped' approach. It is faster than BLAST when using the BLOSUM50 matrix.

There also exists diagonalsw, a C and C++ implementation of the Smith-Waterman algorithm with the SIMD instruction sets (SSE4.1 for the x86 platform and AltiVec for the PowerPC platform). It is licensed under the open-source MIT license.

Cell Broadband Engine [edit]

In 2008, Farrar[17] described a port of the Striped Smith–Waterman[6] to the Cell Broadband Engine and reported speeds of 32 and 12 GCUPS on an IBM QS20 blade and a Sony PlayStation 3, respectively.

See also [edit]

- BLAST
- Sequence mining
- FASTA
- Levenshtein distance
- Needleman–Wunsch algorithm

References [edit]

1. ^ Smith, Temple F.; and Waterman, Michael S. (1981). "Identification of Common Molecular Subsequences" (PDF). *Journal of Molecular Biology* **147**: 195–197. doi:10.1016/0022-2836(81)90087-5. PMID 7265238.
2. ^ Osamu Gotoh (1982). "An improved algorithm for matching biological sequences". *Journal of molecular biology* **162**: 705. doi:10.1016/0022-2836(82)90398-9.
3. ^ Stephen F. Altschul; and Bruce W. Erickson (1986). "Optimal sequence alignment using affine gap costs". *Bulletin of Mathematical Biology* **48**: 603–616. doi:10.1007/BF02462326.
4. ^ Miller, Webb and Myers, Eugene (1988). "Optimal alignments in linear space". *Computer Applications in Biosciences (CABIOS)* **4**: 11–17.
5. ^ Wozniak, Andrzej (1997). "Using video-oriented instructions to speed up sequence comparison" (PDF). *Computer Applications in Biosciences (CABIOS)* **13** (2): 145–50.
6. ^ [a] [b] [c] Farrar, Michael S. (2007). "Striped Smith–Waterman speeds database searches six times over other SIMD implementations" (PDF). *Bioinformatics* **23** (2): 156–161. doi:10.1093/bioinformatics/btl582. PMID 17110365.
7. ^ Zhao, Mengyao; Lee, Wan-Ping; Garrison, Erik P; Marth, Gabor T (4 December 2013). "SSW Library: An SIMD Smith-Waterman C/C++ Library for Use in Genomic Applications". *PLoS ONE* (PLoS) **8** (12). doi:10.1371/journal.pone.0082138. Retrieved 28 April 2014.
8. ^ FPGA 100x Papers: [1], [2], and [3]
9. ^ Progeniq Pte. Ltd., "White Paper - Accelerating Intensive Applications at 10×–50× Speedup to Remove Bottlenecks in Computational Workflows".
10. ^ Vermij, Erik (2011). *Genetic sequence alignment on a supercomputing platform* (PDF) (M.Sc. thesis). Delft University of Technology.
11. ^ "GPU Accelerated Smith–Waterman". SpringerLink.
12. ^ "Bioinformatics High Throughput Sequence Search and Analysis (white paper)". GenomeQuest. Retrieved 2008-05-09.[*dead link*]
13. ^ Manavski, Svetlin A.; and Valle, Giorgio (2008). "CUDA compatible GPU cards as efficient hardware accelerators for Smith–Waterman sequence alignment". *BMC Bioinformatics* **9** (Suppl 2:S10): S10. doi:10.1186/1471-2105-9-S2-S10. PMC 2323659. PMID 18387198.
14. ^ "CUDA Zone". Nvidia. Retrieved 2010-02-25.
15. ^ Rognes, Torbjørn; and Seeberg, Erling (2000). "Six-fold speed-up of Smith–Waterman sequence database searches using parallel processing on common microprocessors" (PDF). *Bioinformatics* **16** (8): 699–706.

16. ^ Rognes, Torbjørn (2011). "Faster Smith–Waterman database searches with inter-sequence SIMD parallelisation". *BMC Bioinformatics* **12**: 221. doi:10.1186/1471-2105-12-221. PMC 3120707. PMID 21631914.
17. ^ Farrar, Michael S. (2008). "Optimizing Smith–Waterman for the Cell Broadband Engine".

External links [edit]

- JAligner — an open source Java implementation of the Smith–Waterman algorithm
- B.A.B.A. — an applet (with source) which visually explains the algorithm.
- FASTA/SSEARCH — services page at the EBI
- UGENE Smith–Waterman plugin — an open source SSEARCH compatible implementation of the algorithm with graphical interface written in C++
- OPAL — a JavaScript implementation of algorithms such as Needleman–Wunsch, Needleman–Wunsch–Sellers and Smith–Waterman
- diagonalsw — an open-source C/C++ implementation with SIMD instruction sets (notably SSE4.1) under the MIT license
- SSW — an open-source C++ library providing an API to an SIMD implemention of the Smith-Waterman algorithm under the MIT license

Categories: Bioinformatics algorithms | Computational phylogenetics | Sequence alignment algorithms | Dynamic programming