Abstract

Motivation: Optimal sequence alignment based on the Smith–Waterman algorithm is usually too computationally demanding to be practical for searching large sequence databases. Heuristic programs like FASTA and BLAST have been developed which run much faster, but at the expense of sensitivity.

Results: In an effort to approximate the sensitivity of an optimal alignment algorithm, a new algorithm has been devised for the computation of a gapped alignment of two sequences. After scanning for high-scoring words and extensions of these to form fragments of similarity, the algorithm uses dynamic programming to build an accurate alignment based on the fragments initially identified. The algorithm has been implemented in a program called SALSA and the performance has been evaluated on a set of test sequences. The sensitivity was found to be close to the Smith–Waterman algorithm, while the speed was similar to FASTA (ktup = 2).

Availability: Searches can be performed from the SALSA homepage at http://dna.uio.no/salsa/ using a wide range of databases. Source code and precompiled executables are also available.

Contact: torbjorn.rognes@labmed.uio.no

Introduction

Algorithms for the computation of an optimal local alignment of two sequences based on a substitution matrix and linear gap penalties have been described by Smith and Waterman (1981) and Gotoh (1982). However, for the purpose of searching large sequence databases, where a query sequence must be aligned to thousands or millions of database sequences, the exhaustive algorithms are usually too computationally demanding to be practical. Special hardware solutions for performing optimal sequence alignment faster than general computers exist (Hughey, 1996). However, those are expensive and therefore not generally available.

Sequence database searching tools like FASTA (Wilbur and Lipman, 1983; Lipman and Pearson, 1985; Pearson and Lipman, 1988; Pearson, 1990) and BLAST (Altschul et al., 1990, 1997) have been successful at finding most of the significant database matches much faster than the rigorous algorithms. These tools are based on a process consisting of an initial screening of the database sequences for high-scoring words, followed by an extension of the identified hits, creation of an alignment based on the fragments found, and finally a calculation of the score or statistical significance of each database sequence based on the score of the initially identified high-scoring regions.

FASTA uses a dynamic programming algorithm to calculate the score (InitN) of a constrained alignment of some of the highest scoring initial regions. However, the resulting score is a very rough estimate of the optimal score.

Here, a new algorithm called SALSA (Searching with Assembly of Local Sequence Alignments) is presented for the computation of a good estimate of the optimal alignment of two sequences. After the initial scanning for high-scoring words and extension of hits, this algorithm considers the most important fragments identified to build an accurate alignment using dynamic programming. SALSA produces a good estimate of the optimal score, without a substantial increase in computation time. The algorithm therefore appears very useful for searching sequence databases.

System and methods

The implementation of the algorithm was written in C++ and compiled with the GNU gcc 2.7.2 compiler on a computer with dual Pentium Pro 200MHz CPUs running Linux 2.0. An Apache web server that provides the services of SALSA on the World Wide Web is also running on this computer.

Algorithm

Definition of alignment

Given a sequence $A = a_1, a_2, ..., a_{m-1}, a_m$, of length $m$, and a sequence $B = b_1, b_2, ..., b_{n-1}, b_n$, of length $n$, both consisting of symbols from an alphabet $Z$ of size $c$. 
Aligning A and B can be considered as the process of transforming sequence A into sequence B by replacement, deletion and insertion of symbols.

An alignment of two sequences is illustrated in Figure 1. Between the two sequences in the figure a vertical line ‘|’ indicates identical symbols, while a plus sign ‘+’ indicates similar symbols.

Insertions in sequence B is indicated by the placement of a gap symbol ‘–’ at the corresponding positions in sequence A. Likewise, deletions of symbols from sequence A is indicated by gap symbols in sequence B.

In a global sequence alignment the entire length of both sequences must be aligned. Only a subsequence of each of the sequences will be aligned in a local alignment.

**Scoring rules and optimal alignment**

Substitutions are scored using a substitution score matrix \( H \) of size \( c \times c \). The element \( h_{a,b} \) from \( H \) gives the score for the substitution of a symbol \( a \) from sequence A with symbol \( b \) from sequence B. The BLOSUM62 matrix (Henikoff and Henikoff, 1992) is commonly used as an amino acid substitution score matrix.

Deletion and insertions are penalized by a gap penalty \( G \) that is dependent upon the length \( g \) of the gap. Only linear gap penalty functions of the form \( G(g) = Q + Rg \) will be considered here, where \( Q \) is the gap open penalty and \( R \) is the gap extension penalty.

The total alignment score is the sum of the individual substitution scores for all aligned positions minus all the gap penalties.

The goal of optimal local sequence alignment is to find the highest scoring local alignment. In general there might be several optimal alignments. The optimal local sequence alignment can be found using a dynamic programming algorithm (Smith and Waterman, 1981). A specialized version of this algorithm with linear gap penalties has an \( O(n^2) \) time complexity (Gotoh, 1982).

**Heuristic algorithm for sequence alignment**

The heuristic method for sequence alignment used in SALSA has many similarities to FASTA and BLAST, but includes a significant post-processing stage that increases the sensitivity.

The heuristic alignment algorithm can be divided into three stages: (1) scanning of the database sequence for word matches; (2) extension of these hits to form fragments, also known as high scoring segment pairs (HSPs) in BLAST (Altschul et al., 1990, 1997); and (3) calculation of a final sequence similarity score based on the fragments found.

**Database scanning:** In the first stage of the searching process SALSA, as well as FASTA and BLAST, look for small continuous regions (words) of high similarity between the query sequence and each database sequence.

Using a word length \( k \) of 2–4 amino acids this step can be done very quickly at the expense of memory by creating a lookup table. This lookup table contains \( c^k \) entries, one for each possible \( k \)-tuple or word, and is computed just once for each query sequence. This can be done by scanning the query sequence for matches to each of the words and storing the matching position(s) in the table. This table allows rapid retrieval of the positions of the query sequence that a given word from the database sequence will match.

SALSA is similar to BLAST in that it uses score-based matches with words of a few amino acids. Only those words that have a word score \( S \) above or equal to the threshold \( T \) will match. The word score \( S \) is the sum of the \( k \) entries in the substitution matrix that corresponds to the \( k \) pairs of symbols from the matching words. SALSA requires only one match, as opposed to BLAST 2.0, which requires two nearby hits on the same diagonal.

Given a database sequence of length \( n \), the database sequence can be considered as \( n - k + 1 \) overlapping \( k \)-tuples. As the database sequence is scanned, each of these tuples is looked up in the table. The current position in the database sequence and the matching position(s) in the query sequence define points that are part of potentially high scoring regions, and should be good candidates for further examination.

**Hit extension:** All the hits found during database scanning are now extended to form fragments. A fragment is actually an ungapped alignment of a continuous subsequence from each of the two sequences. Initially, a fragment consists of the matching word, but is subsequently extended in both
directions along the diagonal until a maximal fragment score $S$ is obtained. Diagonal $d$ consists of all pairs $(a_i, b_i)$ of symbols where $j - i = d$. The fragment score $S$ is calculated by summing scores from the substitution matrix $H$. The extension process is stopped when the potential fragment score drops below zero or a predefined level below the maximal score currently obtained for the fragment. The use of sentinel values at both ends of the sequences and in the score matrix $H$ can simplify termination of the extension process.

**Final score calculation:** Version 1.4 of BLAST reported the score of the single highest scoring fragment for each database sequence as well as the calculated statistical significance of each match by taking into account the score and positions of some other compatible high-scoring fragments.

The new version 2.0 of BLAST creates a very accurate gapped alignment starting on one of these fragments, but only for those having a very high score. A fragment score as high as 40 is usually required to trigger such a gapped alignment, which means that a gapped alignment is computed only for a small fraction of the database sequences.

FASTA takes a different approach, and calculates an alignment score (InitN) that is based on the scores of several compatible fragments and penalizes for intervening gaps by a joining penalty. However, this results in a rather crude estimate of the optimal alignment score.

In an attempt to calculate a more accurate estimate of the optimal alignment score, SALSA attempts to assemble the fragments found into a gapped alignment. This has been found to produce a good estimate of the optimal alignment score.

For the highest scoring sequences ($E < 100$) in the initial alignment, SALSA will finally calculate the score of an optimal alignment constrained to a band that covers all diagonals of the initial alignment plus 20 diagonals on each side.

**Alignments based on assembly of fragments**

An optimal sequence alignment of biological sequences, as produced by the Smith–Waterman algorithm, usually consists of a few longer fragments of similarity intervened by small gaps, like the example in Figure 2.

If gap penalties were infinitely high, gaps could be ignored, and the optimal alignment would simply be the highest scoring fragment. As gap penalties are lowered, gaps must be taken into account when looking for the optimal alignment. However, with reasonably high gap penalties, as used in practical applications, optimal alignments would still usually consist of a few high-scoring fragments chained together.

If all the fragments involved in the optimal alignment could be found, it might be possible to construct an optimal alignment by an appropriate connection of the fragments. If a few of the lowest scoring fragments were not found, a slightly suboptimal alignment could be created, still giving a good estimate of the optimal score. The aim is to build an alignment from all the fragments found in the initial stages of the searching process.

**Connecting fragments using dynamic programming**

SALSA uses a dynamic programming approach to find the optimal arrangement of the fragments found. Fragments that score below a threshold (17) are not considered, as they may increase the computation time substantially without much increased sensitivity.

The fragments are stored in a table $T$ that allows fast sequential access to all fragments on a given diagonal $d$. This table contains $(m + n - 1)$ sorted double-linked lists, each containing the fragments on a specific diagonal. A separate linked list $L$ of the fragments enables access to the fragments sequentially by their position along the database sequence, independently of the diagonal.

For each fragment, the position, length and score $S$ is stored. In addition, all fragments are assigned a new alignment score $X$ which will represent the best score of an alignment ending at that fragment. It is initially set equal to the score of the fragment, so that $X = S$. A pointer $E$ to the preceding fragment in an alignment is also needed, and is initially set to zero.

All fragments on the list $L$ are processed in order as follows: For each fragment $F_i$, a number of candidate fragments $F_j$, are selected from $L$. Connection is then attempted between $F_i$ and each of the candidate fragments $F_j$, to see if the alignment score $X_j$ of $F_j$ can be increased. A new alignment score $Y$ is computed by adding together the alignment score $X_j$ of $F_j$ and the fragment score $S_j$ of $F_j$ and subtracting the gap penalty. The ends of the fragments must also be extended.
Fig. 3. The six different ways to arrange two fragments $F_i$ (light gray) and $F_j$ (dark gray) relative to each other on different diagonals (ignoring mirrors along main diagonal), and a possible connection of them (thin black line).

or trimmed and the score must be adjusted accordingly. If the resulting score $Y$ is greater than the previous score $X_j$ for the candidate fragment $F_j$, $X_j$ is replaced by $Y$ and the pointer $E$ updated to point to $F_i$.

The fragment with the highest alignment score $X$ represents the overall best local alignment. Following the pointers $E$ backwards from that fragment gives the alignment.

**Optimal placement of gaps between connected fragments**

Figure 3 illustrates 6 different ways two fragments can be arranged relative to each other on different diagonals.

In order to connect two fragments by a gap, their ends need to be trimmed or extended. If arranged as in case 1 in Figure 3, then usually both fragments will have to be extended. In cases 2, 3, 4 and 5, the fragments will usually have to be trimmed. Possible positioning of the gap is illustrated in Figure 4.

Correct positioning of the gap is important for obtaining the optimal score. The optimal gap placement is found by computing the accumulated scores $U$ and $V$ along the two extension paths indicated by dotted lines in Figure 4, starting in the upper left end of both. The gap should be placed where their difference $U-V$ is highest.

Placement of the gap in one of the light gray areas is considered next, starting at the border of the dark gray area, and then further away from the dark area until the potential score falls a certain level below the overall best score currently obtained.

A similar procedure is used in cases 2–5, where both fragments must be trimmed.

**Selection of fragments for making connections**

In order to make the algorithm fast, it is important to limit the number of candidate fragments that need to be tested by the optimal gap placement procedure.

An important observation is that for all gaps in an optimal alignment, the score of the partial alignment on either side of the gap must be higher than the gap penalty. Otherwise, the lowest scoring partial alignment and the gap can be removed, resulting in a higher score. For instance, the score of the first fragment must always be larger than the penalty of the gap between the first and second fragment in an optimal alignment. This principle can be used to limit substantially the number of candidate fragments that must be considered.

The maximal gap, $g_m$, between the current fragment $F_i$ and a candidate fragment $F_j$, can be computed from the alignment score $X_i$ of $F_i$:

$$g_m = \left[ \frac{X_i - Q}{R} \right] - 1$$

(1)
For example, when connecting the first fragment \( F_1 \) with score 20 to the next fragment \( F_2 \), the maximum allowed gap penalty is \(-19\). With gap open and extension penalties of \( Q = 10 \) and \( R = 2 \) the maximal size of the gap is 4.

In turn, this principle can also be applied when connecting the partial alignment of fragments \( F_1 \) and \( F_2 \) to the next fragment \( F_3 \), using the combined alignment score \( X_2 \) of \( F_1 \) and \( F_2 \).

Connecting fragments \( F_i \) and \( F_j \) on diagonals \( d_i \) and \( d_j \), respectively, results in a gap of size \( g = |d_j - d_i| \). This gap is subject to the gap penalty \( G = Q + Rg \). This limits the diagonal of a candidate fragments \( F_j \) to the range \((d_i - g_m, d_i + g_m)\).

Within those diagonals, additional restrictions can be imposed. A candidate fragment must be located so that at least a part of the candidate fragment \( F_j \) is below or to the right of some part of the current fragment \( F_i \). This eliminates candidate fragments arranged as in case 6.

When fragments \( F_i \) and \( F_j \) are arranged as in case 1, candidate fragments \( F_j \) that are too far away from the source fragment \( F_i \) are not tested as it is quite costly to calculate. Based on the average score of \(-0.5\) for the alignment of two random amino acids using the BLOSUM62 score matrix, an estimate of the alignment score is calculated. If the estimated score is below a certain level, a connection will not be attempted.

If several candidate fragments \( F_j \) on the same diagonal are arranged relatively to \( F_i \) as in case 1, it is only necessary to test the first of these fragments.

For example, before connecting two fragments \( F_i \) and \( F_j \), a potential maximal alignment score can be determined. If this results in a score that is below the current best alignment score \( X_j \) for \( F_j \), the connection need not be tested.

### Statistical significance of alignments

The statistical significance of the gapped alignments was calculated as described by Karlin and Altschul (1990) using the \( K, \lambda \) and \( H \) parameters as estimated by Altschul and Gish (1996). The expected number \( E \) of database sequences with an alignment score at least \( S \) is given by the following equation, where \( M' \) and \( N' \) are the effective lengths of the query sequence and the database, respectively.

\[
E = M' N' K e^{-\lambda S}
\]

The effective lengths \( M' \) and \( N' \) are calculated from the real lengths \( M \) and \( N \) by subtracting an edge correction term \( L \):

\[
L = \frac{\ln (KMN)}{H}
\]

### Implementation

The new algorithm has been implemented in a program called SALSA. The program reads sequence databases in the NCBI BLAST 1.4 file format, where the database is split in three separate files, the sequences, the descrip-
tions, and index tables. SALSA reads query sequence files formatted as plain text, or in FASTA, EMBL, GenBank or PIR flat file formats. At present, SALSA only accepts amino acid query sequences.

The database may contain either nucleotide or amino acid sequences. If the database contains nucleotide sequences, the sequences will be translated using the standard genetic code in all 6 frames, which will be compared individually with the amino acid query sequence.

An example of the output produced by the program is shown in Figure 5.

A World Wide Web service that can be found at http://dna.uio.no/salsa/ has been established, and allows anyone with Internet access to run searches using SALSA. Users may choose from a wide range of sequence databases, input a query sequence in any format, and select appropriate search options. The results are presented with alignments and links to full sequence entries. Source code and precompiled executables are also available.

SALSA is implemented as a multi-threaded application using POSIX threads, which allows it to fully take advantage of multiple CPUs in symmetric multiprocessing (SMP) computers.

Results

The sensitivity of SALSA was evaluated using a test set of 11 different amino acid query sequences (Table 1). The 11 sequences represent a range of well characterized protein families. The same test set has previously been used for the evaluation of BLAST 2.0 (Altschul et al., 1997).

To see how SALSA compared to other published algorithms for database searching, the SWISS-PROT release 34 protein database (59 021 sequences with a total of 21 210 388 amino acid residues) was searched using the 11 query sequences. The number of sequences found with a score better than a minimal score corresponding to an E-value of 1.0 was counted. An E-value as high as of 1.0 was chosen in order to examine the ability of the programs to also find the more distantly related sequences in addition to the more obvious homologs.

SALSA (version 1.8.0) was compared to the SSEARCH (version 3.1t10) (Pearson, 1991), WU-BLAST (version 2.0a19) (Warren Gish, unpublished, http://blast.wustl.edu/), BLAST (version 2.0.5) (Altschul et al., 1997) and FASTA (version 3.1t10) (Pearson and Lipman, 1988) programs. SSEARCH is a Smith–Waterman implementation (Smith and Waterman, 1981) that uses Phil Green’s (University of Washington) SWAT optimizations (unpublished) for increased speed. The non-threaded (single CPU) version was used for all programs.

All searches were done using the BLOSUM62 matrix (Henikoff and Henikoff, 1992) and gap open and extension penalties of 10 and 1 respectively. WU-BLAST, SALSA and BLAST were run using a word size of 3 and a word score threshold of 11. FASTA was run with ksub = 2. WU-BLAST was run with the recommended ‘postsw’ option. E-value calculations were based on Karlin–Altschul statistics (Karlin and Altschul, 1990) with the estimated parameters λ = 0.255, K = 0.035 and H = 0.19 (Altschul and Gish, 1996).

Table 1. Test set of query sequences and search results. The columns indicate the protein family name, the accession number and id of the query sequence, the minimal score corresponding to E = 1.0, and the number of sequences found by SSEARCH, WU-BLAST, SALSA, BLAST and FASTA in the SWISS-PROT release 34 database with scores greater than or equal to the minimal score. The bottom line shows the total CPU time used by each program. See results for further details

<table>
<thead>
<tr>
<th>Protein family</th>
<th>Accession</th>
<th>Score</th>
<th>SSEARCH</th>
<th>WU-BLAST</th>
<th>SALSA</th>
<th>BLAST</th>
<th>FASTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serine protease</td>
<td>P00762</td>
<td>73</td>
<td>280</td>
<td>280</td>
<td>280</td>
<td>279</td>
<td>276</td>
</tr>
<tr>
<td>Serine protease inhibitor</td>
<td>P01008</td>
<td>77</td>
<td>110</td>
<td>110</td>
<td>109</td>
<td>110</td>
<td>106</td>
</tr>
<tr>
<td>Ras</td>
<td>P01111</td>
<td>72</td>
<td>304</td>
<td>301</td>
<td>299</td>
<td>281</td>
<td>293</td>
</tr>
<tr>
<td>Globin</td>
<td>P02232</td>
<td>71</td>
<td>145</td>
<td>145</td>
<td>145</td>
<td>133</td>
<td>134</td>
</tr>
<tr>
<td>Hemagglutinin</td>
<td>P03435</td>
<td>77</td>
<td>163</td>
<td>159</td>
<td>159</td>
<td>152</td>
<td>128</td>
</tr>
<tr>
<td>Interferon α</td>
<td>P05013</td>
<td>72</td>
<td>53</td>
<td>53</td>
<td>53</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>Alcohol dehydrogenase</td>
<td>P07327</td>
<td>76</td>
<td>142</td>
<td>140</td>
<td>139</td>
<td>141</td>
<td>136</td>
</tr>
<tr>
<td>Histocompatibility antigen</td>
<td>P10318</td>
<td>75</td>
<td>319</td>
<td>319</td>
<td>317</td>
<td>318</td>
<td>291</td>
</tr>
<tr>
<td>Cytochrome P450</td>
<td>P10635</td>
<td>77</td>
<td>219</td>
<td>219</td>
<td>218</td>
<td>219</td>
<td>198</td>
</tr>
<tr>
<td>Glutathione S-transferase</td>
<td>P14942</td>
<td>73</td>
<td>107</td>
<td>106</td>
<td>104</td>
<td>98</td>
<td>92</td>
</tr>
<tr>
<td>H+ -transporting ATP synthase</td>
<td>P25705</td>
<td>77</td>
<td>209</td>
<td>207</td>
<td>206</td>
<td>206</td>
<td>197</td>
</tr>
</tbody>
</table>

| Total CPU time (s)             | 7463      | 625   | 613     | 189      | 676   |
Compared to SSEARCH, WU-BLAST missed 0.6% of the hits and SALSA missed 1%, while BLAST missed 3% and FASTA over 7%. Especially in the Globin protein family, WU-BLAST and SALSA found many more hits than BLAST and FASTA. The sensitivity of SALSA relative to BLAST and FASTA probably reflects its ability to build accurate gapped alignments from several low-scoring fragments. Appropriately positioned, a few low-scoring fragments may represent a higher alignment score than a single high-scoring fragment. The sensitivity of WU-BLAST is comparable to SALSA or better, however, we cannot evaluate the performance of WU-BLAST since the details of the algorithm are unpublished and unavailable.

Total CPU time for the programs was measured as the fastest of 3 runs. Compared to SALSA, WU-BLAST and FASTA was approximately equal in speed, while BLAST was about 3 times faster, and SSEARCH was about 12 times slower. The speed of BLAST is probably mainly due to its two-hit strategy, requiring two initial word matches within a distance of 40 before an extension is attempted.

Discussion

Algorithms for building local alignments from fragments has previously been described and implemented in tools for aligning very long nucleotide sequences (Chao and Miller, 1995; Chao et al., 1995), but these programs were based on alignment fragments of fixed size k-tuples of matching nucleotides.

By allowing gaps and combining the score of several fragments, SALSA is able to identify significant sequence alignments even when the score of any single fragment is not by itself significant. This is important for identification of distant related sequences with scores in the ‘twilight zone’.

BLAST 2 also computes gapped alignments, however, only on the most promising fraction of the sequences initially identified. BLAST 2 requires the existence of at least one high scoring fragment before a gapped alignment is performed. The gapped alignments produced by BLAST 2 are not restricted to regions on the diagonals of the identified fragments. This method is therefore quite different from the one implemented in SALSA. The gapped alignments created in this way may be more accurate than the alignments produced by SALSA, but also requires more computation time.

The evaluation showed that BLAST was the fastest of the programs for database searching, but SALSA and WU-BLAST were more sensitive and able to approximate the sensitivity of the Smith–Waterman algorithm in much less computation time.

There are a few situations that cause the SALSA alignments to be suboptimal. Some of the necessary fragments might not be found during the initial database scanning. Some fragments with a very evenly distributed score might not have a single word with score above the threshold, and would not cause a hit. This is a problem with all the heuristic programs. An alternative scanning method that is able to examine longer words is required to overcome this problem.

The score of some of the necessary fragments in an optimal alignment might be very low. Such fragments may appear between higher scoring fragments in order to connect them. A more accurate connection algorithm that also examines matrix elements that are not on the same diagonals as the two fragments might avoid these problems.

Acknowledgements

This work was supported by grants from the Research Council of Norway and the Norwegian Cancer Society.

References