RESEARCH ARTICLE

Two Simple and Efficient Algorithms to Compute the SP-Score Objective Function of a Multiple Sequence Alignment

Vincent Ranwez*

Montpellier SupAgro, UMR AGAP, 34060, Montpellier, France

* Vincent.ranwez@supagro.fr

Abstract

Background

Multiple sequence alignment (MSA) is a crucial step in many molecular analyses and many MSA tools have been developed. Most of them use a greedy approach to construct a first alignment that is then refined by optimizing the sum of pair score (SP-score). The SP-score estimation is thus a bottleneck for most MSA tools since it is repeatedly required and is time consuming.

Results

Given an alignment of n sequences and L sites, I introduce here optimized solutions reaching $O(nL)$ time complexity for affine gap cost, instead of $O(n^2L)$, which are easy to implement.

Citation: Ranwez V (2016) Two Simple and Efficient Algorithms to Compute the SP-Score Objective Function of a Multiple Sequence Alignment. PLoS ONE 11(8): e0160043. doi:10.1371/journal.pone.0160043

Editor: Yang Zhang, University of Michigan, UNITED STATES

Received: May 30, 2016

Accepted: July 12, 2016

Published: August 9, 2016

Data Availability Statement: All relevant data are within the paper.

Funding: This work has been supported by the French research agency: Agence Nationale de la Recherche (ANR-10- BINF-01-02 "Ancestrome") (http://www.agence-nationale-recherche.fr/).

Competing Interests: The authors have declared that no competing interests exist.

Introduction

A wide range of molecular analyses rely on multiple sequence alignments (MSA), e.g., prediction of tridimensional structures [1], phylogenetic inference [2] or detection of positive selection [3]. In all these studies, the initial MSA can strongly impact conclusions and biological interpretations [4,5]. As a consequence, MSA is a richly developed area of bioinformatics and computational biology.

Most MSA software use a greedy approach to construct a first alignment that is then refined by optimizing the sum of pair score (SP-score) [6,7] using a 2-cut strategy. This strategy consists in partitioning the current solution into two sub-alignments that are subsequently re-aligned; the resulting MSA replaces the previous one if its SP score is improved [7,8,9,10]. The SP-score estimation is thus a bottleneck for most MSA tools since it is repeatedly required and is time consuming as existing solutions have a time complexity of $O(n^2L)$ for an alignment of n sequences and L sites. The practical importance of having an efficient solution to compute SP-score lies within the following 'Muscle software paper' [8] quotation: "Notice that computation of the SP score dominates the time complexity of refinement and of MUSCLE overall[. . .]. It is natural to seek an $O(nL)$ expression for [SP-score estimation. . .], but to the best of our knowledge no solution is known."

The main result of this paper is an optimized algorithmic solution to estimate SP-score for affine gap costs in $O(nL)$. I also introduce a more versatile solution able to handle more general gap cost penalty functions in $O(nL + n^2G_{max})$, with $G_{max} \leq L$ being the maximum number of gap intervals within one aligned sequence.

Materials and Methods

Definitions and notations

A multiple sequence alignment (MSA) for a set of sequences $\{s_1...s_n\}$, defined with alphabet Σ, is a set of n sequences $\{S_1...S_n\}$ which are defined on an enriched alphabet $\Sigma \cup\{\text{'-'}\}$ so that all S_i have the same length L and, $\forall i$, removing '-' from S_i leads to s_i. The aim of MSA tools is to position gaps (stretches of '-') so that characters at the same position (constituting a site) are (most likely) homologous. This is usually achieved through a heuristic optimization of the sum of pair score (SP-score). The SP-score of an MSA is obtained by considering all pairwise alignments it induced. Given two sequences S_i and S_j of MSA $\mathcal{A}$ the corresponding pairwise restriction ($\mathcal{A}\,|\{S_i,S_j\}$) is the alignment made up of two sequences S_i' and S_j' obtained by removing the '-' of S_i (resp. S_j) whenever S_j (resp. S_i) also has a gap at this position/site. The score of this pairwise alignment is the sum of the substitution/match scores induced by sites without gaps, plus the sum of scores associated to the gap intervals (maximal sub-sequences of consecutive gap symbols) observed in S_i' and S_j'.

The SP-score of an alignment is classically obtained in $O(n^2L)$ by summing up the score of its $\binom{n}{2}$ induced pairwise alignments (Algorithm 1 and 2), and can be decomposed into two terms: SPs, the contribution of substitutions/matches and, SPg the contribution of induced gap intervals (denoted IG').

In molecular biology $|\Sigma|$ is a constant so typically small (4 for nucleotides and 20 for amino acids) that $L|\Sigma|^2$ and $n|\Sigma|^2$, compared to nL, can safely be ignored in asymptotic complexity analysis. Under this assumption, all solutions described here have an $O(nL)$ space complexity.

Algorithm. 1. Basic $O(n^2L)$ computation of the SP-score of an alignment $\mathcal{A}$ of n sites and L sequences given subst(.,.) and g_cost(.) functions. The subst(.,.) function provides, in $O(1)$, the elementary score for two non gap characters on the same site, e.g. using BLOSUM matrix [11]. The g_cost(.) function provides, also in $O(1)$, the cost of a gap interval based on its position and size. The compute_gap_intervals(.) subroutine (Algorithm 2) returns in $O(|S|)$ the list of the gap intervals of its input sequence S.

```
Algorithm 1: compute_SP_score
Input: -The $n$ aligned sequences $S_i$ of alignment $\mathcal{A}$
    -a function subst(x,y) returning in $O(1)$ the score for two non gap charac-
ters x and y on the same site of $\mathcal{A}$
    -a function g_cost(IG') returning in $O(1)$ the cost of a gap interval IG'
Output: the SP score of this alignment
$SP_s = 0$; $SP_g = 0$
for $S_i$ in $S_1 \dots S_n$
  for $S_j$ in $S_{i+1} \dots S_n$
    $S_i' = S_j' = ""$
    for $k$ in 1...L
      if(not ($S_i[k]$ == '-' and $S_j[k]$ == '-'))
        $S_i' = S_i' + S_i[k]$
        $S_j' = S_j' + S_j[k]$
      if($S_i[k] \neq$ '-' and $S_j[k] \neq$ '-')
        $SP_s = SP_s$ + subst($S_i'[k], S_j'[k]$)
    for IG' in compute_gap_intervals($S_i'$) $\cup$ compute_gap_intervals($S_j'$)
      $SP_g = SP_g$ + g_cost(IG') // e.g., g_cost(IG') ={ return $gap_o$
+ IG'[length]*$gap_{ext}$}
```

```
Return $SP_s + SP_g$
```

Algorithm. 2. An $O(L)$ algorithm to compute the list of gap intervals, ordered by their gap start position, of a sequence S of length L. Note that, thought IG[length] is not explicitly set, it is assumed it can be access since IG[length] is simply IG[end]-IG[start]+1.

```
Algorithm 2: compute_gap_intervals
Input: An aligned sequences $S_i$ of length $L$
Output: The list of gap intervals of $S_i$, ordered by gap start position
$LG$ = { } ; // the list of gap intervals of $S_i$ found so far
$IG$ = NULL; // the current gap interval
for $k$ in 1...$L$
  if ($S_i[k]$ == '−' and $IG$ = = NULL) // start a new gap interval
    $IG$ = new Interval(start = k)
  if ($S_i[k] \neq$ '−' and $IG \neq$ NULL) // the current gap interval finish at previous
position
    $IG$[ end] = k-1; append a copy of $IG$ to the list $LG$
    $IG$ = NULL
if ($IG \neq$ NULL) // handle terminal gap if any
    $IG$[end] = L; append a copy of $IG$ to the list $LG$
Return $LG$
```

Efficient algorithm to compute SP-score using general gap cost penalties

The SPs part of the SP-score can be computed in O(nL) by using a table of size $L|\Sigma|$ containing for each site the number of each (non gap) symbol (e.g. [8]). This strategy does not work for SPg except for the basic, but unrealistic, constant gap cost where g_cost(IG') = IG'[length].gap-cost. I introduce here a more efficient solution to the SP-score computation problem accounting for most gap function penalties (including affine, log, log-affine penalties). The main idea is to pre-compute the list of gap intervals of each sequence S_i, ordered by gap start position, this can easily be done in $O(L)$ using Algorithm 2. The compute_gap_intervals(S'_i) and compute_gap_intervals(S'_j) of Algorithm 1, observed in $\mathcal{A}|\{S_i,S_j\}$, can then be efficiently deduced by processing the gaps of compute_gap_intervals(S_i) ∪ compute_gap_intervals(S_j) according to order of opening (as done to merge two sorted lists in linear time, e.g. during a merge step of the 'merge sort' algorithm) while maintaining the number of gaps facing gaps encountered so far (i.e. the shift between S and S' site coordinate systems for current position). The resulting SP-score algorithm (Algorithm 3) has a complexity of $O(nL + n^2G_{max})$, with $G_{max} \leq \lceil L/2 \rceil$ the maximum number of gap intervals within one aligned sequence, instead of $O(n^2L)$. Note that the difference with the naïve algorithm is especially important when sequences contain few long gap stretches but that in the worst case, where most sequences have a number of gap intervals close to L, this algorithm has the same $O(n^2L)$ complexity as the naïve solution.

Optimal algorithm to compute SP-score using affine gap cost penalties

Affine gap penalties (where g_cost(IG') = gap_O + IG'[length].gap_{ext}) are frequently used. For such gap penalties, the total of gap extension penalties (SP_{ge}) can also be efficiently computed in $O(nL)$, by counting the number of gaps per site. However, gap opening cannot be counted exactly based on local site information (e.g [8]) only approximated. Though pessimistic gap count approximation [9] is often used during the dynamic programming steps producing new candidate alignments, the exact SP score is generally preferred to decide which alignments are better than the current one. Algorithm 4 provides a simple and exact solution to compute the SP-score under affine gap penalties in $O(nL)$, which is also the time complexity for just reading an alignment of n sequences of L sites.

The key idea is to note that a gap IG_i will add a number of gap opening penalties equal to n minus the number of interval IG_j so that $IG_i \subseteq IG_j$. In order to find out how many gaps encompass IG_i, sites are processed from left to right while maintaining an array indicating, for each left site, the number of gap stretches already opened at this position and not yet closed. For all gaps IG_i closing at the current position i, the value stored at index $IG_i[leb]$ of this array provides, in $O(1)$, the number of gap stretches encompassing IG_i; before considering site i+1, this array is maintained updated by decreasing by 1 all values stored at indices between $IG_i[leb]$ and i for all IG_i ending at i—hence updates for all sites overall require $O(nL)$. External and internal gaps are often penalized with different affine functions. The proposed $O(nL)$ solution can handle this refinement by: firstly, using different characters (e.g. '-' and '_') to represent the two different gap types while computing SP_{ge}; and secondly, testing each gap interval type in Algorithm 3 (using gap interval start/end positions) to select the adequate gap opening cost.

Algorithm. 3. Given the gap interval lists LG_i, LG_j of sequences $S_i \in \mathcal{A}$ and $S_j \in \mathcal{A}$; this algorithm returns in $O(|LG_i| + |LG_j|)$ the restricted gap interval lists LG'_i, LG'_j that would be observed in $\mathcal{A}|\{S_i,S_j\}$ without actually building this restricted alignment.

```
Algorithm 3: compute_pairwise_restricted_gap_intervals
Input:—$LG_i$, $LG_j$ the ordered lists of gap intervals for $S_i \in \mathcal{A}$ and $S_j \in \mathcal{A}$
Output:—$LG'_i$, $LG'_j$ the lists of gap intervals in $\mathcal{A}$|{ $S_i$, $S_j$}
$IG_i$ = first($LG_i$); $IG_j$ = first($LG_j$)
shift = 0;
$LG'_i = LG'_j$ = { }
$IG'_i$ = new Interval(start = -1)
$IG'_j$ = new Interval(start = -1) // using -1 allows to check if interval start
has already been set or not
while($IG_i \neq$NULL and $IG_j \neq$NULL)
  if($IG_i$[ start] == $IG_j$[ start])
      if($IG_i$ == $IG_j$) // both intervals disappear when $\mathcal{A}$ is restricted to
$\mathcal{A}$|{ $S_i$, $S_j$}
        $IG_i$ = next($LG_i$); $IG'_i$ = new Interval(start = -1)
        $IG_j$ = next($LG_j$); $IG'_j$ = new Interval(start = -1)
      elif($IG_j \subset IG_i$) // $IG_j$ disappear during restriction
        $IG_j$ = next($LG_j$); $IG'_j$ = new Interval(start = -1)
        if($IG'_i$[start] == -1) //$IG'_i$[start] is now known
          $IG'_i$[start] = $IG_i$[start] -shift
      else // ($IG_i \subset IG_j$) // $IG_i$ disappear during restriction
        ...... // similar to previous case swapping i and j
      shift = shift+|$IG_i \cap IG_j$|
  elif($IG_i$[start] < $IG_j$[start])
    if($IG_j \subset IG_i$) // $IG_j$ disappear during restriction, shift increase
      if($IG'_i$[start] == -1) // set $IG'_i$[start], if not already done, before
increasing shift
        $IG'_i$[start] = $IG_i$[start] -shift
    shift = shift+|$IG_i \cap IG_j$|
    $IG_j$ = next($LG_j$); $IG'_j$ = new Interval(start = -1)
  else // $IG_j$ start after $IG_i$ and is not included in $IG_i$
    if($IG'_i$[start] == -1)
      $IG'_i$[start] = $IG_i$[start] -shift
    if($IG_i \cap IG_j \neq \oslash$) //$IG'_j$[start] is now known and shift increase
      $IG'_j$[start] = $IG_j$[start] -shift
      shift = shift + |$IG_i \cap IG_j$|
    $IG'_i$[end] = $IG_i$[end] -shift
    append $IG'_i$ to $LG'_i$
    $IG_i$ = next($LG_i$); $IG'_i$ = new Interval(start = -1)
  else // ($IG_j$[start] < $IG_i$[start] )
    ...... // similar to previous case swapping $i$ and $j$
if ($IG_i \neq$ NULL) // handle last gaps in $LG_i$
  if ($IG'_i$[start] == -1){$IG'_i$[start] = $IG_i$[start] -shift}
  $IG'_i$[end] = $IG_i$[end] -shift
  append $IG'_i$ to $LG'_i$
  while (($IG_i$ = next ($LG_i$) $\neq$ NULL)
    append new Interval (start = $IG_i[start]$ -shift; end = $IG_i[end]$ -shift) to $LG'_i$
if ($IG_j \neq$ NULL)
    ...... //similar to previous block replacing $i$ with $j$
return $LG'_i$, $LG'_j$
```

Conclusion

This paper introduces an optimized algorithmic solution to estimate SP-score for affine gap costs in $O(nL)$ and a more versatile solution able to handle more gap cost penalty functions in $O(nL + n^2G_{max})$, with $G_{max} \leq \lceil L/2 \rceil$ being the maximum number of gap intervals per sequence. These optimizations will obviously be part of the next release of MACSE [10], the MSA software we developed to align nucleic sequences with respect to their amino acid translation while allowing them to contain frameshifts and/or stop codons (http://bioweb.supagro.inra.fr/macse/). Moreover, once stated those two algorithms are quite straightforward and can easily be included in the numerous existing MSA software relying on SP-score.

Algorithm. 4. Efficient $O(nL)$ computation of the contribution of gap opening cost (SP_{go}) for an alignment $\mathcal{A}$ of n sites and L sequences.

```
Algorithm 4: compute_SP$_{go}$_using_gap_intervals
Input: -the $n$ aligned sequences $S_1...S_n$ of $\mathcal{A}$
       - the costs of a gap opening within a sequence (gap$_O$) or at its extremities (gap$_{O\_ext}$)
Output: SP$_{go}$: the part of the SP-score of $\mathcal{A}$ due to gap opening costs
nbOpenGap = new Array of $L$ integers initialized to 0;
gapClosing = new Array of $L$ empty lists of Intervals;
for $S_i$ in $S_1...S_n$
  //construct $LG_i$ in $O(L)$ and update nbOpenGap and gapClosing arrays
  $LG_i$ = compute_gap_intervals ($S_i$)
  foreach $IG_i$ of $LG_i$
    for $k$ in $IG_i$[start] ...$IG_i$[end]
      nbOpenGap[k] ++
  append $IG_i$ to gapClosing[$IG_i$[end] ]
SP$_{go}$ = 0; // part of the SP score related to gap opening costs
for $i$ in 1 ... $L$
  foreach $IG_i$ in gapClosing[$i$]
    if ($i$ == $L$ OR $IG_i$[start] == 1)
      SP$_{go}$ = SP$_{go}$+ (n-nbOpenGap[$IG_i$[start] ] )* gap$_{O\_ext}$
    else
      SP$_{go}$ = SP$_{go}$+ (n-nbOpenGap[$IG_i$[start] ] )* gap$_O$
  foreach $IG_i$ in gapClosing[$i$]
    for $k$ in $IG_i$[start] ... $IG_i$[end]
      nbOpenGap[$k$] = nbOpenGap[$k$] -1;
return SP$_{go}$
```

Acknowledgments

This work has been supported by the French research agency: Agence Nationale de la Recherche (ANR-10- BINF-01-02 “Ancestrome”).

Author Contributions

Wrote the paper: VR.

Conceived and wrote the algorithms: VR.

References

1. Przybylski D, Rost B. Alignments grow, secondary structure prediction improves. Proteins. 2002; 46 (2):197–205. PMID: 11807948.
2. Loytynoja A, Goldman N. Phylogeny-aware gap placement prevents errors in sequence alignment and evolutionary analysis. Science. 2008; 320(5883):1632–5. doi: 10.1126/science.1158395 PMID: 18566285
3. Meredith RW, Gatesy J, Murphy WJ, Ryder OA, Springer MS. Molecular decay of the tooth gene Enamelin (ENAM) mirrors the loss of enamel in the fossil record of placental mammals. PLoS Genet. 2009; 5 (9):e1000634. doi: 10.1371/journal.pgen.1000634 PMID: 19730686
4. Wong KM, Suchard MA, Huelsenbeck JP. Alignment uncertainty and genomic analysis. Science. 2008; 319(5862):473–6. doi: 10.1126/science.1151532 PMID: 18218900
5. Blackburne BP, Whelan S. Class of multiple sequence alignment algorithm affects genomic analysis. Mol Biol Evol. 2013; 30(3):642–53. doi: 10.1093/molbev/mss256 PMID: 23144040
6. Altschul SF. Gap costs for multiple sequence alignment. J Theor Biol. 1989; 138(3):297–309. PMID: 2593679
7. Wang X-D, Liu J-X, Xu Y, Zhang J. A Survey of Multiple Sequence Alignment Techniques. In: Huang D-S, Bevilacqua V, Premaratne P, editors. Intelligent Computing Theories and Methodologies: 11th International Conference, ICIC 2015, Fuzhou, China, August 20–23, 2015, Proceedings, Part I. Cham: Springer International Publishing; 2015. p. 529–38.
8. Edgar RC. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. BMC Bioinformatics. 2004; 5:113.Kececioglu JD, Zhang W. Aligning Alignments. CPM '98: Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching. Lecture Notes In Computer Science. 1448: Springer-Verlag; 1998. p. 189–208.
9. Kececioglu JD, Zhang W. Aligning Alignments. CPM '98: Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching. Lecture Notes In Computer Science. 1448: Springer-Verlag; 1998. p. 189–208.
10. Ranwez V, Harispe S, Delsuc F, Douzery EJ. MACSE: Multiple Alignment of Coding SEquences accounting for frameshifts and stop codons. PLoS One. 2011; 6(9):e22594. doi: 10.1371/journal.pone.0022594 PMID: 21949676
11. Henikoff S, Henikoff JG. Amino acid substitution matrices from protein blocks. Proc Natl Acad Sci U S A. 1992; 89(22):10915–9. PMID: 1438297