On pairwise alignment with dynamic programming

Heng Li

Broad Institute, 415 Main Street, Cambridge, MA 02142, USA

ABSTRACT

This *informal* note introduces important literatures on pairwise biological sequence alignment with dynamic programming (DP), disambiguates a few different formulations, investigates the choice of gap cost function and discusses the implementation of DP-based alignment with a focus on practical applications. It targets developers who are interested in the history of DP-based alignment and want to understand it in depth. Importantly, this note is not a general tutorial. Audience should be familiar with the basis of pairwise alignment before reading the note.

1 ALIGNMENT WITH DYNAMIC PROGRAMMING

Let T be the *target* sequence or the *reference* sequence of length $n = |T|$ and Q be the *query* sequence of length $m = |Q|$. Gaps on the target sequence are *deletions*; gaps on the query sequence are *insertions*. Function $s(i, j)$, $0 \le i < n$ and $0 \le j < m$, gives the score between the $T[i]$ and $Q[j]$. We will focus on global alignment algorithms because it is usually more complex than local alignment due to non-trivial initial conditions.

1.1 Levenshtein distance

Levenshtein distance (Levenshtein, 1966) is the minimum sum of substitutions, insertions and deletions among all possible alignments between two sequences. The author proposed the measurement but did not provide an algorithm to compute the distance. Levenshtein distance is the most common measure of edit distance. Other variants of edit distances may include fewer (e.g. no substitutions) or more types of primitive edit operations (e.g. transpositions).

Levenshtein distance can be computed with dynamic programming (DP). Let H_{ij} be the distance between prefixes $T[0..i]$ and $Q[0..j]$. The distance between the full sequences will be $H_{n-1,m-1}$, which can be recursively computed with

$$H_{ij} = \min\{H_{i-1,j-1} + 1 - \delta_{T[i],Q[j]}, H_{i-1,j} + 1, H_{i,j-1} + 1\} \quad (1)$$

where δ_{ab} is the Kronecker delta function, which equals 1 if $a = b$ or 0 otherwise. The initial conditions are: $H_{-1,j} = j + 1$ and $H_{i,-1} = i + 1$. This is an $O(mn)$ algorithm.

Practically faster algorithms exist. Landau et al. (1986) found an $O(kn)$ algorithm that guarantees to find the optimal solution if the edit distance is no larger than k. It is particularly fast if the distance is small (Šošic, 2015).

Following the idea behind the 4-Russian problem (Arlazarov et al., 1970), Wu et al. (1996) made an important observation that the differences between adjacent cells

$$\begin{cases} u_{ij} \triangleq H_{ij} - H_{i-1,j} \\ v_{ij} \triangleq H_{ij} - H_{i,j-1} \end{cases}$$

only take values -1, 0 or 1. They transformed Eq. (1) to

$$\begin{cases} z_{ij} = \min\{1 - \delta_{T[i],Q[j]}, v_{i-1,j} + 1, u_{i,j-1} + 1\} \\ u_{ij} = z_{ij} - v_{i-1,j} \\ v_{ij} = z_{ij} - u_{i,j-1} \end{cases} \quad (2)$$

and used a lookup table to speed up the computation. Myers (1999) furthered this idea with bit-level parallelization, resulting in an $O(mn/w)$ algorithm where w is the size of a machine word in bits. Combined with bounding (Ukkonen, 1985), Myers' algorithm is significantly faster than Eq. (1) for long sequences. Edlib (Šošic and Šikic, 2017) provides an efficient implementation of this algorithm with added functionality.

1.2 Linear gap cost

When Needleman and Wunsch (1970) first proposed to align a pair of biological sequences with DP, they focused on a linear gap cost function $\gamma_0(k; e) = k \cdot e$, $e > 0$. Let the optimal score up to cell (i, j) be H_{ij}. It can be computed with

$$H_{ij} = \max\{H_{i-1,j-1} + s(i, j), H_{i-1,j} - e, H_{i,j-1} - e\} \quad (3)$$

This equation is a generalization of Eq. (1). It gives matches, mismatches and gaps different weights. Typically, we require $2e$ to be larger than the mismatch cost; otherwise the alignment would always prefer two gaps over a mismatch – the final alignment would not contain any mismatches.

The Needleman-Wunsch algorithm has a history of multiple inventions in different fields. Readers are referred to Navarro (2001) for a more complete survey.

It is also possible to apply Myer's bit-parallelis to the linear gap cost if $s(i, j)$ and e are small integers (Loving et al., 2014).

1.3 Classical formulation of affine gap cost

With a linear cost function, a long gap is considered to arise from a series of small gaps independently. In evolution, however, a long gap at times results from one event (e.g. a transposon insertion). A linear gap cost often breaks a long gap into small pieces and complicates the interpretation of alignment. Therefore, it is discouraged to use a linear cost to produce alignment for evolutionarily related sequences.

The limitation of linear cost motivated Waterman et al. (1976) to use a more general cost function. This work was later folded to Smith and Waterman (1981), *the* Smith-Waterman paper. With a general cost, the time complexity to find the optimal alignment is $O(mn \max\{m, n\})$. Gotoh (1982) showed that when the cost function takes a form $\gamma_1(k; q, e) = q + k \cdot e$, which is called *affine gap cost*, it is possible to solve the alignment problem in $O(mn)$ time. Altschul and Erickson (1986) fixed an issue in the original Gotoh's algorithm and introduced the formulation we commonly use today.

1.3.1 Durbin's formulation To give alignment a probablitistic interpretation, Durbin et al. (1998) introduced

$$\begin{cases} M_{ij} = \max\{M_{i-1,j-1}, E_{i-1,j-1}, F_{i-1,j-1}\} + s(i, j) \\ E_{ij} = \max\{M_{i-1,j} - q, E_{i-1,j}\} - e \\ F_{ij} = \max\{M_{i,j-1} - q, F_{i,j-1}\} - e \end{cases} \quad (4)$$

This formulation has a natural connection to pair-HMM with each state having a clear meaning in alignment. It, however, has one problem: it disallows transitions between E and F states and thus forbids insertions immediately followed by deletions (and vice versa). When the gap extension cost e is smaller than the half of a mismatch cost, transitions between E and F may yield a better alignment score. It is possible to add transitions between E and F in Eq. (4), but in practice, AE86's formulation (Altschul and Erickson, 1986) will be simpler and faster to implement.

1.3.2 AE86's formulation In Eq. (4), if we let:

$$H_{ij} \triangleq \max\{M_{ij}, E_{ij}, F_{ij}\}$$

Durbin's formulation allowing E–F transitions becomes

$$\begin{cases} E_{ij} = \max\{H_{i-1,j} - q, E_{i-1,j}\} - e \\ F_{ij} = \max\{H_{i,j-1} - q, F_{i,j-1}\} - e \\ H_{ij} = \max\{H_{i-1,j-1} + s(i, j), E_{ij}, F_{ij}\} \end{cases} \quad (5)$$

Algorithm 1: AE86's formulation with affine gap cost

Input: Target sequence T and query Q; scoring matrix $S(\cdot,\cdot)$ and affine gap cost $\gamma_1(k;q,e) = q + k \cdot e$
Output: Best alignment score between T and Q

Function ALIGNSCORE(T, Q, q, e) **begin**

1 **for** $a \in \Sigma$ **do** ▷ *Generate query profile*
  for $j \leftarrow 0$ **to** $|Q| - 1$ **do**
    $P[a][j] \leftarrow S(Q[j], a)$
for $j \leftarrow 0$ **to** $|Q| - 1$ **do**
  $H[j] \leftarrow -q - j \cdot e$ ▷ $H[j] = H_{-1,j-1}$
  $E[j] \leftarrow -2q - 2e - j \cdot e$ ▷ $E[j] = E_{0j}$
$H[j] \leftarrow 0$ ▷ $H_{-1,-1} = 0$
for $i \leftarrow 0$ **to** $|T| - 1$ **do**
  $f \leftarrow -2q - 2e - i \cdot e$ ▷ $f = F_{i0}$
  $h \leftarrow -q - e - i \cdot e$ ▷ $h = H_{i,-1}$
  $p \leftarrow P[T[i]]$
  for $j \leftarrow 0$ **to** $|Q| - 1$ **do**
2 $s \leftarrow p[j]$ ▷ $s = S(Q[j], T[i])$
    $h' \leftarrow \max\{H[j] + s, E[j], f\}$
    $H[j] \leftarrow h$ ▷ $H[j] = H_{i,j-1}$
    $h \leftarrow h'$ ▷ $h = H_{ij}$
    $E[j] \leftarrow \max\{h - q, E[j]\} - e$ ▷ $E[j] = E_{i+1,j}$
    $f \leftarrow \max\{h - q, f\} - e$ ▷ $f = F_{i,j+1}$
  $H[|Q|] \leftarrow h$
return $H[|Q|]$

This is AE86's formulation. In practice, we sometimes compute the cells in the following order

$$\begin{cases} H_{ij} = \max\{H_{i-1,j-1} + s(i,j), E_{ij}, F_{ij}\} \\ E_{i+1,j} = \max\{H_{ij} - q, E_{ij}\} - e \\ F_{i,j+1} = \max\{H_{ij} - q, F_{ij}\} - e \end{cases} \tag{6}$$

with initial conditions

$$\begin{cases} H_{-1,-1} = 0 \\ H_{-1,j} = -q - e - j \cdot e \quad (0 \le j < m) \\ H_{i,-1} = -q - e - i \cdot e \quad (0 \le i < n) \\ E_{0j} = -2q - 2e - j \cdot e \quad (0 \le j < m) \\ F_{i0} = -2q - 2e - i \cdot e \quad (0 \le i < n) \end{cases} \tag{7}$$

We don't need $E_{-1,\cdot}$ or $F_{\cdot,-1}$ because Eq. (6) does not start with these initial values.

AE86 can be impelmented in different ways depending on the order of computation and how row scores are stored. Algorithm 1 gives one implementation. At the beginning of each iteration at line 2, $f = F_{ij}$, $h = H_{i,j-1}$, $H[j] = H_{i-1,j-1}$ and $E[j] = E_{ij}$. The loop computes H in cell (i,j), E in the next row and F in the next column. This algorithm uses a query profile at line 1. It is a common techique to accelerate the inner loop.

1.3.3 Effect of affine gap cost For simplicity, we use one score $a > 0$ for all types of matches and one cost $b > 0$ for all types of mismatches. Ignore gaps for now. If the identity between T and Q is below $1 - a/(a + b)$, T and Q will get a negative alignment score. The $b : a$ ratio sets the minimum identity. Now suppose we have a long deletion. If $\lceil q/(a + b) \rceil$ or more residues on the query adjacent to the gap are mismatches but have a perfect match to a subsequence in the gap, the perfect match will yield a higher alignment score and split the long gap in two. Gap open cost q determines how easily a long gap to be split into two or more smaller gaps. Gap extension e is directly related to E–F transitions: if $b > 2e$, there may be insertions immediately followed by deletions. q and e together control the total number of gaps. Suppose in a small region there are x mismatches and one gap. We may achieve a better score by opening a new gap and add $2y$ gap extensions if $x \cdot b > q + 2y \cdot e$, approximately. We also note that scoring must satisfy $2(q + e) > b$; otherwise the alignment would not contain any mismatches.

1.4 Affine gap cost: SIMD acceleration

SIMD CPU instructions perform one action on a vector of data at the same time. For example, with SSE2, a type of SIMD, we can compute the sum of two vectors of sixteen 8-bit integers with one CPU instruction. This is much faster than summing with sixteen standard instructions. How many data can be processed with one SIMD instruction depends on the number of bits in the vector and the max value of each element in the vector. For example, SSE instructions operate on 128-bit vectors. We can process four 32-bit integers or eight 16-bit integers at the same time. AVX instructions operate on 256-bit vectors. It doubles the bandwidth of SSE.

SIMD has been used to speed up DP-based pairwise alignment. There are two general classes of SIMD algorithms: inter-sequence and intra-sequence. Inter-sequence algorithms (Rognes, 2011) align multiple pairs of sequences at the same time. It is conceptually easier to implement and faster to run but it is tricky to use with other alignment routines. Intra-sequence algorithms align one sequence at a time. There are several ways to implement intra-sequence pairwise alignment, depending on how to organize multiple data into one vector.

For simplicity, we assume each vector consists of four elements. Wozniak (1997) put $(H_{ij}, H_{i+1,j-1}, H_{i+2,j-2}, H_{i+3,j-3})$ into a vector and fills the DP matrix along its diagonal. Rognes and Seeberg (2000) took a block of column cells into a vector $(H_{ij}, H_{i,j+1}, H_{i,j+2}, H_{i,j+3})$. Farrar (2007) interleaved rows into $(H_{ij}, H_{i,t+j}, H_{i,2t+j}, H_{i,3t+j})$ in a striped manner, where $t = \lfloor (m + 3)/4 \rfloor$ – the algorithm collates cells distant apart. In practice, Farrar's striped algorithm is the fastest and most often used (Szalkowski et al., 2008; Zhao et al., 2013). Daily (2016) developed a programming library that implements all three intra-sequence algorithms.

1.5 Affine gap cost: Suzuki's formulation

When working with long sequences that yield large alignment scores, we may need to use 32-bit integers to hold the score arrays. With SSE, we can only process four cells at a time. Inspired by Myers (1999) and Loving et al. (2014), Hajime Suzuki proposed to rewrite Eq. (6) with differences between cells:

$$\begin{cases} u_{ij} \triangleq H_{ij} - H_{i-1,j} \\ v_{ij} \triangleq H_{ij} - H_{i,j-1} \\ x_{ij} \triangleq E_{i+1,j} - H_{ij} \\ y_{ij} \triangleq F_{i,j+1} - H_{ij} \end{cases} \tag{8}$$

as

$$\begin{cases} z_{ij} = \max\{s(i,j), x_{i-1,j} + v_{i-1,j}, y_{i,j-1} + u_{i,j-1}\} \\ u_{ij} = z_{ij} - v_{i-1,j} \\ v_{ij} = z_{ij} - u_{i,j-1} \\ x_{ij} = \max\{0, x_{i-1,j} + v_{i-1,j} - z_{ij} + q\} - q - e \\ y_{ij} = \max\{0, y_{i,j-1} + u_{i,j-1} - z_{ij} + q\} - q - e \end{cases} \tag{9}$$

where z_{ij} is a temporary variable that does not need to be stored. We can prove that all variables in these equations are bounded by gap costs and the extreme match and mismatch scores, but not by the sequence lengths or the peak alignment score. For small scores, we can encode 16 cells in one SSE vector.

In practice, Suzuki's formulation is about twice as slow as Algorithm 1 without SSE vectorization, because it involves more computation and cannot use a query profile. For long sequences when the peak score does not fit 16-bit integers, 16-way vectorized Suzuki's formulation is twice as fast as striped 4-way vectorization (Farrar, 2007). Another advantage of Suzuki's algorithm is that it can be adapted for banded alignment.

1.6 Piece-wise affine gap cost

Affine gap cost still has issues with long gaps. Recall that an exact match of length $\lceil q/(a + b) \rceil$ in the middle of a long gap splits the gap into two

if moving this exact match to either edge of the gap leads to mismatches (Section 1.3.3). For noisy reads, it is not infrequent for a long gap to be split by incidental sequencing errors. We would prefer to increase the gap open cost q to avoid such a split, but this would contradict the high INDEL error rate of some sequencing data.

The root cause of this dilemma is that gaps are caused by two different mechanisms: evolution which may create a long gap with one event, and sequencing errors which generate gaps as relatively independent events. A better gap cost should be concave, such that $\gamma(k+1)-\gamma(k)$ is smaller with larger k.

Miller and Myers (1988) found an $O(mn\log\max\{m,n\})$ algorithm for a concave gap cost function $\gamma(k)$. When $\gamma(k)$ is a piece-wise affine cost composed of p affine cost functions, their algorithm finds the optimal alignment in $O(mn\log p)$ time. Gotoh (1990) proposed an $O(mn\cdot p)$ algorithm, which is simpler and probably faster for small p in practice.

When $p=2$, the two-piece affine cost takes the form

$$\gamma_2(k;q,e,\tilde{q},\tilde{e})=\min\{q+k\cdot e,\tilde{q}+k\cdot\tilde{e}\}$$

It is concave on the condition that $q+e<\tilde{q}+\tilde{e}$ and $e>\tilde{e}$. Effectively, this cost function applies $\gamma_1(k;q,e)$ to gaps shorter than $\lceil(\tilde{q}-q)/(e-\tilde{e})\rceil$ and applies $\gamma_1(k;\tilde{q},\tilde{e})$ to longer gaps. We can compute the maximal alignment score under $\gamma_2(k)$ with

$$\begin{cases} H_{ij}=\max\{H_{i-1,j-1}+s(i,j),E_{ij},F_{ij},\tilde{E}_{ij},\tilde{F}_{ij}\} \\ E_{i+1,j}=\max\{H_{ij}-q,E_{ij}\}-e \\ F_{i,j+1}=\max\{H_{ij}-q,F_{ij}\}-e \\ \tilde{E}_{i+1,j}=\max\{H_{ij}-\tilde{q},\tilde{E}_{ij}\}-\tilde{e} \\ \tilde{F}_{i,j+1}=\max\{H_{ij}-\tilde{q},\tilde{F}_{ij}\}-\tilde{e} \end{cases} \tag{10}$$

Eq. (9) can be extended to work with piece-wise affine cost in a similar manner.

REFERENCES

Altschul, S. F. and Erickson, B. W. (1986). Optimal sequence alignment using affine gap costs. *Bull Math Biol*, 48:603–16.

Arlazarov, V. L., Dinic, E. A., Kronrod, M. A., and Faradzev, I. A. (1970). On economic construction of the transitive closure of a directed graph. *Dokl. Akad. Nauk SSSR*, 194:487–8.

Daily, J. (2016). Parasail: Simd c library for global, semi-global, and local pairwise sequence alignments. *BMC Bioinformatics*, 17:81.

Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G. (1998). *Biological sequence analysis*. Cambridge University Press.

Farrar, M. (2007). Striped smith-waterman speeds database searches six times over other simd implementations. *Bioinformatics*, 23:156–61.

Gotoh, O. (1982). An improved algorithm for matching biological sequences. *J Mol Biol*, 162:705–8.

Gotoh, O. (1990). Optimal sequence alignment allowing for long gaps. *Bull Math Biol*, 52:359–73.

Landau, G. M., Vishkin, U., and Nussinov, R. (1986). An efficient string matching algorithm with k differences for nucleotide and amino acid sequences. *Nucleic Acids Res*, 14:31–46.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10:707–710.

Loving, J., Hernandez, Y., and Benson, G. (2014). Bitpal: a bit-parallel, general integer-scoring sequence alignment algorithm. *Bioinformatics*, 30:3166–73.

Miller, W. and Myers, E. W. (1988). Sequence comparison with concave weighting functions. *Bull Math Biol*, 50:97–120.

Myers, G. (1999). A fast bit-vector algorithm for approximate string matching based on dynamic programming. *J. ACM*, 46:395–415.

Navarro, G. (2001). A guided tour to approximate string matching. *ACM Comput. Surv.*, 33:31–88.

Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48:443–53.

Rognes, T. (2011). Faster smith-waterman database searches with inter-sequence simd parallelisation. *BMC Bioinformatics*, 12:221.

Rognes, T. and Seeberg, E. (2000). Six-fold speed-up of smith-waterman sequence database searches using parallel processing on common microprocessors. *Bioinformatics*, 16:699–706.

Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. *J Mol Biol*, 147:195–7.

Szalkowski, A., Ledergerber, C., Krähenbühl, P., and Dessimoz, C. (2008). Swps3 - fast multi-threaded vectorized smith-waterman for ibm cell/b.e. and x86/sse2. *BMC Res Notes*, 1:107.

Ukkonen, E. (1985). Algorithms for approximate string matching. *Inform. and Control*, 64:100–118.

Šošic, M. (2015). *An SIMD dynamic programming C/C++ Library*. PhD thesis, University of Zagreb.

Šošic, M. and Šikic, M. (2017). Edlib: a C/C++ library for fast, exact sequence alignment using edit distance. *Bioinformatics*, 33:1394–1395.

Waterman, M. S., Smith, T. F., and Beyer, W. A. (1976). Some biological sequence metrics. *Advan. Math.*, 20:367–87.

Wozniak, A. (1997). Using video-oriented instructions to speed up sequence comparison. *Comput Appl Biosci*, 13:145–50.

Wu, S., Manber, U., and Myers, G. (1996). A subquadratic algorithm for approximate limited expression matching. *Algorithmica*, 15:50–67.

Zhao, M., Lee, W.-P., Garrison, E. P., and Marth, G. T. (2013). Ssw library: an simd smith-waterman c/c++ library for use in genomic applications. *PLoS One*, 8:e82138.