A General Framework for Exact Partially Local Alignments

Falco Kirchner\(^1\), Nancy Retzlaff\(^2,1\) and Peter F. Stadler\(^{1,2,3,4,5}\)

\(^1\)Bioinformatics Group, Department of Computer Science, Interdisciplinary Center for Bioinformatics, and Competence Center for Scalable Data Services and Solutions Dresden/Leipzig, Universität Leipzig, Härtestraße 16-18, D-04107 Leipzig, Germany

\(^2\)Max Planck Institute for Mathematics in the Sciences, Inselstraße 22, D-04103 Leipzig, Germany

\(^3\)Institute for Theoretical Chemistry, University of Vienna, Währingerstraße 17, A-1090 Wien, Austria

\(^4\)Santa Fe Institute, 1399 Hyde Park Road, Santa Fe NM 87501, U.S.A.

Keywords: Multiple Sequence Alignments, Dynamics Programming.

Abstract: Multiple sequence alignments are a crucial intermediate step in a plethora of data analysis workflows in computational biology. While multiple sequence alignments are usually constructed with the help of heuristic approximations, exact pairwise alignments are readily computed by dynamic programming algorithms. In the pairwise case, local, global, and semi-global alignments are distinguished, with key applications in pattern discovery, gene comparison, and homology search, respectively. With increasing computing power, exact alignments of triples and even quadruples of sequences have become feasible and recent applications e.g. in the context of breakpoint discovery have shown that mixed local/global multiple alignments can be of practical interest. vaPLA is the first implementation of partially local multiple alignments of a few sequences and provides convenient access to this family of specialized alignment algorithms.

1 INTRODUCTION

Global multiple alignments are typically constructed as intermediate data structure to support a comparative or evolutionary analysis homologous sequences. Alignment problems are naturally treated as optimization problems: a scoring function evaluates the similarities in an alignment column and/or the pattern of gaps. Multiple alignments are almost exclusively treated globally, that is, all parts of the input sequence is scored. The notion of “local multiple alignments” appears mostly in the context of phylogenetic footprinting (Lukashin and Rosa, 1999; Blanchette et al., 2002) and related pattern discovery problems (Tabei and Asai, 2009), where substrings are considered that appear with a limited number of mismatches in some or all input sequences.

Local variants of sequence alignment, on the one hand, play an important role in pairwise alignments. Local alignments, i.e., maximally similar substrings within pairs of longer sequences, are a natural way to identify conserved domains. The semi-global variant of pairwise alignment, in which one sequence, usually called “query”, is expected to appear as approximate substring of a larger “subject”, again is a natural formalization of homology search, implemented e.g. in gotohsan (Hertel et al., 2009). Overlap alignments (Jones and Pevzner, 2004) allowing free end gaps on all sequences have applications e.g. in sequence assembly (Rausch et al., 2009). Until recently, the generalization of these variants to more than two sequences has received very little attention.

Pairwise alignment problems can be solved exactly for a wide range of cost models by means of dynamic programming. In fact, the algorithms of Needleman and Wunsch (1970) for global alignments, Smith and Waterman (1981) for local alignments, and the extension to affine gap costs by Gotoh (1982) are among the early, paradigmatic example of dynamic programming. The basic recursive structure is readily extended to more than two input sequences (Carillo and Lipman, 1988; Lipman et al., 1989); the time and space complexity, however, grows exponentially with the number of sequences. Exact dynamic programming solutions thus have been used in practice only for 3-way (Gotoh, 1986; Dewey, 2001; Konagurthu et al., 2004; Kruspe and Stadler, 2007) or 4-way (Steiner et al., 2011) alignments. Since multiple sequence alignment problems (for arbitrary numbers of input sequences \( X \) are typically NP-hard...
(Kececioglu, 1993; Wang and Jiang, 1994; Bonizzoni and Della Vedova, 2001; Just, 2001; Manthey, 2003; Elias, 2006), they are solved by heuristic approximation algorithms, see Chatzou et al. (2016), Baichoo and Ouuzounis (2017) or Nute et al. (2018) for a recent reviews.

As the exact 3-way and 4-way alignments have increased in usage, variants of the problem that combine local and global alignments have been proposed for specialized application scenarios. Al Arab et al. (2017) considered the fate of sequences in the wake of mitochondrial genome rearrangements by simultaneously comparing the rearranged region to both of its ancestors. This approach made it possible to distinguish tandem duplication random loss (TDRL) from reversal or transposition events. This specialized 3-way alignment problem suggested the need to develop a general theoretical framework for alignments that consider part of their input local and part global. As shown by Retzlaff and Stadler (2018) it is possible – and convenient – to allow the user determine separately for each input sequence and each of its ends, whether it is to be treated as global, i.e., deletions of a prefix or a suffix are penalized, or as local, allowing the omission of prefixes or suffixes at no cost. We will briefly outline the theoretical results in the following section. While the presentation by Retzlaff and Stadler (2018) is purely theoretical and did not supply a reference implementation, the present contribution closes this gap.

2 THEORY

The basic idea behind the framework of Retzlaff and Stadler (2018) boils down to two ingredients: (1) Each input sequence is either local or global on the left and either local or global on the right. This is entirely the user’s choice and provided with the input. (2) In a particular alignment column, a sequence may be inactive (if up to this column its prefix is considered unaligned), active (if it contributes to the column either with one of its characters or with a gap this is scored), or it is dead (if its suffix is considered unaligned). Consequently a left-local sequence starts out inactive, while a left-global sequence starts out active. Correspondingly, a right-global input is still active at the end of the alignment, while a right-local sequence must be dead at the end of the alignment. The partially local alignment problems can be solved by dynamic programming just as the classic pairwise problems mentioned in the introduction. As usual, a scoring (memoization) table $S$ holds the optimal alignments of prefixes. The only difference is that $S$ now depends not only on the length of prefixes but also on the state (inactive, active, or dead) of each sequence in a given column of the alignment. It is sufficient to record the set $A$ of active and $D$ of dead sequences, since the inactive sequences are given by $X \setminus \{A \cup D\}$. As the alignment progresses, an inactive sequence may become active only once, and an active may transit at most once to the dead state. Consecutive alignment columns thus have state pairs $(A', D')$ and $(A, D)$ that are comparable w.r.t. the partial order

\[
(A', D') \preceq (A, D) \iff \begin{cases} A' \cup D' \subseteq A \cup D \\ D' \subseteq D \end{cases}
\]

The state changes can be performed stepwisely. As shown by Retzlaff and Stadler (2018), $(A', D')$ is an immediate predecessor of $(A, D)$ if either exactly on one inactive sequence become active or exactly one active sequence transitions to the dead state. We denote this relation by $\prec$. The initial condition is $A = A_0$, where $A_0$ is the set of left-global sequences, and $D = \emptyset$.

In a quite general form (which uses the heuristic version of the affine gap cost model for more than two sequences), the recursion for the optimal alignment score are of the form

\[
S_f^{(A, D)} = \max_{\pi} \left\{ \max_{(A', D') \prec (A, D)} \left[ S_f^{(A', D')} + s(\pi) \right] \right\}
\]

The variable $\pi$ (a non-null binary vector) denotes gap pattern in the last alignment column, the lower multi-
index \( I \) describes the lengths of the prefixes included in the alignment including this column. Thus \( I - \pi \) is the vector of prefix lengths in the previous column. The scoring function \( s(.) \) in the most general form depends on both gap patterns as well as the actual sequence entries. The second alternative does not move in the alignment but changes the state, a step that may also be associated with a cost \( s \), which in the most general case may depend on \( I, \pi, A, D, A', D' \). Equation (2) describes the recursion for additive scores. The special case that a sequence \( k \) is that both left- and right-local remains completely unaligned is handled by a directed transition from inactive to dead restricted to \( I_k = 0 \), see Retzlaff and Stadler (2018) for details.

The notation is illustrated in Fig. 1 for the algorithm introduced by Al Arab et al. (2017). In principle, the recursions are easily extended to affine gap costs. However, this incurs another factor \( (2^N - 1) \) in memory for \( N \) sequences since the scoring tables \( S \) becomes explicitly dependent on the gap pattern of the last column. The full recursions for the breakpoint alignment model with affine gap costs, are given in the appendix of Retzlaff and Stadler (2018).

The backtracing recursion is a rather straightforward generalization of the backtracing scheme for the Smith-Waterman algorithm. The first step is to find the optimal score of the partially local alignment. Denote by \( (A^\ast, D^\ast) \) that unique maximal state w.r.t. \( \preceq \) i.e., \( A^\ast \) is the set of all right-global sequences and \( D^\ast \) is the set of all right-local sequences. Hence \( A^\ast \cup D^\ast \) contains all input sequences. The optimal score is the maximum over all multi-indices \( I \) with constraint that \( I_k = n_k \), the length of the \( k \)-th sequence, for all \( k \in A^\ast \) with the maximum taken over all indices \( I_k \) with \( k \in D^\ast \). The maximum value of \( I_k \) determines the right boundary of the right-local sequence. The recursion then proceeds, as usual, to find the index or state transition in Eq. (2). The backtracing recursion terminates as soon as all left-global variables \( k \in A_0 \) have reached the left end oft the sequences, i.e., \( I_k = 0 \) for all \( k \in A_0 \). The left boundary of a left-local sequences \( l \not\in A_0 \) equals the index \( I_l \) at this point.

Equation (2) is the simplest way to explain the recursive structure of the partially local alignment problem. It has the disadvantage that it provides more than one way to obtain a particular partial alignment (characterized by \( I, A, D \)) since state transition can be performed in arbitrary order. As a consequence, Eq. (2) cannot be used to compute partition functions over alignments, and hence to obtain a probabilistic version. As described in some detail by Retzlaff and Stadler (2018), unambiguous recursions can be constructed by allowing state transitions from inactive to active and from active to dead for a sequence \( k \) only in conjunction with appending of a alignment column for which \( \pi_k = 1 \). At the same time, one needs to consider also all possible state transitions with \( (A', D') \preceq (A, D) \). That is,

\[
S^{(A,D)}_I = \max_\pi \max_{(A',D')} \left[ S^{(A',D')}_{I-\pi} + s(\pi) + s' \right]
\]

where \( \max_{(A',D')} \) runs over all \( (A', D') \preceq (A, D) \) such that \( k \in A \setminus A' \) or \( k \in D \setminus D' \) implies \( \pi_k = 1 \).

### 3 IMPLEMENTATION

We have implemented the two variants of the alignment algorithm for partially local alignments with additive gap costs based on Eq. (2) and Eq. (3), respectively. In the practical implementation of Eq. (3) we omit the formal initial state \( (A_0, \emptyset) \) and separately initialize all left-local sequences \( k \) both in the inactive and the active state for \( I_k = 0 \). Similarly, we catch the final states at the right end of right-local sequences without explicitly considering a final transition to the dead state after the last letter has been included into the alignment. Eq. (3) in general requires the consideration of more state changes in each step than Eq. (2).

While Eq. (2) only considers the Hasse diagram of the partial order \( \prec \), its transitive closure is required for Eq. (3). On the other hand, Eq. (3) provides a very convenient starting point for later extensions e.g. to a probabilistic version.

vaPLA is written in Java and uses only standard libraries. The source code is available on GitHub. In principle, vaPLA is capable of accepting an arbitrary number of input sequences (in FASTA format) together with information on whether each of their ends is to be treated globally or locally. However, the resource requirements quickly become prohibitive with the number of input sequences and in particular the number of local ends. vaPLA first explicitly constructs the Hasse diagram of the partial order \( \prec \) for Eq. (3) and uses this information to allocated the memoization tables for the recursions. The partial order can be exported in .dot format and visualized using a standard graph drawing tools such as graphviz. For Eq. (2) only the relation \( \preceq \), i.e., the edges of the Hasse diagram, is used, while Eq. (3) makes use of the entire partial order \( \prec \).

The partial order determines the required resources: one \( \prod_{k\in A} O(n_k) \)-size table is required for each state \( (A, D) \). Writing \( g = |A_0 \cap A'| \) for the number of global sequences, \( s = |A_0 \cup A'\setminus A| \) for the number of sequences that are local and one end and
global at the other, and $\ell$ for the number of local sequences, there are

$$h = 1^g2^s3^\ell$$  \hspace{1cm} (4)

distinct states, because global sequences are always active, semi-local sequences change either state from inactive to to active or from active to dead, while local sequences can pass through all three states. All combinations of these states must be considered, since the relative order (between sequences) of the state transitions is not constrained. With one index variable iterating over each of the $N$ sequences of length $O(n)$, the memory requirements are $O(n^3)$ for each state. The evaluation of the recursion requires $O(2^3)$ score computations for each transition between columns and states, resulting in an upper bound of $O(2^3n^3h^2)$ effort. The effort is reduced to $O(2^{3n}hN)$ by implementing Eq. (2) instead of Eq. (3) as shown in Fig. (3). Both variants are available in the current implementation. Backtracing is implemented in the usual manner: starting from the position $I^*$ and state $(A^*,D^*)$ of the optimal score, $vaPLA$ computes the transition that resulted in the optimal score. At present, co-optimal solutions are not investigated. The first solution encountered is used. The procedure then continues iteratively until a valid start state is reached.

4 BENCHMARK

We use two well known benchmark protein databases to test and evaluate $vaPLA$. OXBench (Raghava et al., 2003) is a completely automatically generated database whereas BALIBASE (Thompson et al., 2005) has a manual step for cleaning initial alignments before the release. Both benchmark sets are intended for global multiple alignments. We therefore inspected a subset of the reference alignments and manually inspected overhanging ends, which we tagged for local instead of global alignment. Figure 2 summarizes the distribution of local ends in benchmark set used here. These sequences are subsequently aligned with $vaPLA$ as well as three of the most common alignment tools: $T$-Coffee (Notredame et al., 2000; Chang et al., 2014), MAFFT (Katoh et al., 2005; Nakamura et al., 2018), and ClustalW (Larkin et al., 2007; Steivers and Higgins, 2018). All three tools are based on initial pairwise alignments and use essentially progressive schemes, hence presenting efficient heuristics rather than exact solutions.

Comparing the performance for few and many local ends, respectively, we can see in Fig. 3 that the running time of $vaPLA$, as expected, strongly depends on how many local ends needs to be handled in one alignment. Without local ends, i.e., for global alignments, the execution time of $vaPLA$ is comparable with $T$-Coffee. For partially local alignments we observe the expected exponential increase with the number of local ends. Since $vaPLA$ is designed as a reference implementation of a much more expensive, exact algorithm, it is clear that it cannot be competitive in terms resource consumption. Figure 3 also com-
The position shift error $PSE$ as defined by Raghava et al. (2003) serves as an alternative measure of alignment accuracy. Consider a pair of (mis)matched positions $i$ in sequence $x$ and $j$ in sequence $y$ in the reference alignment. In the test alignment we consider the same position $i$ in $x$ and its (mis)matched position $j'$ in $y$ and measure the distance $\delta = |j - j'|$. A similar rule is used to compute $\delta$ if there is an in/del between $x$ and $y$ at position $i$.

For the details we refer to (Raghava et al., 2003). The $PSE$ is the average of the contributions $\delta$ of all mismatched pairs in the reference alignment. Omitted prefixes and suffixes at local ends do not enter the $PSE$ score. $vaPLA$ exhibits significantly smaller position shift errors than the three heuristics, Fig. 5.

### 5 DISCUSSION

$vaPLA$ is primarily intended as a reference implementation against which specialized partially local alignments can be tested and benchmarked. We anticipate at least two use cases. First, $vaPLA$ is useful to to create test cases and help debugging during the development phase of a specialized exact implementation. More importantly, since $vaPLA$ computed exact solutions for a moderate number of input sequences, it can be used to generate ground-truth data against which faster heuristics can be compared. The current version of $vaPLA$ was not implemented to yield good performance while we expect that substantial gains can be achieved by parallelization with fine grained multi-threading (Martins et al., 2001). Still it will need to be tested whether such a solution is realizable in Java.

While biological sequences tend to be rather long as compared to average word lengths we see a promising application to alignment of lexical items where prefixing and suffixing seem to play even a bigger role than in biology. Even though affixes can contain information, the root of words is most valuable when doing cross-linguistics comparisons. For conceptual examples we refer to (Retzlafl and Stadler, 2018).

The computational efforts for exact dynamic programming algorithms often can be reduced excluding subsets of matches using bounds on the achievable scores. Lossless filters for local pairwise alignments have been pioneered by Peterlongo et al. (2008, 2009). Ideas to prune the search space of the DP problem are discussed e.g. by Schroedl (2005) or Bilu et al. (2006). Quasi-alignments based on $k$-mer matches can be employed as alignment anchors also in a global context (Nagar and Hahsler, 2013). These techniques may be used not only to reduce the computational effort of the exact algorithm for input sequences of practical interest, but also might serve as starting points for constructing efficient heuristics for partially local MSA problems.

The framework of $vaPLA$ lends itself to further ex-

---

**Figure 4:** Accuracy of $vaPLA$ compared to the heuristic global alignment tools $T$-Coffee, MAFFT, and ClustalW.

**Figure 5:** Position shift error of $vaPLA$ compared to the heuristic global aligners $T$-Coffee, MAFFT, and ClustalW.
tensions. First, it is easily possible to derive a probabilistic version. This essentially entails a change in the scoring from adding score contributions to multiplying with the corresponding Boltzmann factors. The corresponding outside algorithm could easily be constructed along the lines of Höner zu Siederdissen et al. (2015). Another extension that could be realized very easily is to enforce additional constraints on state transitions. For example, it may be useful in a pattern-based application to allow the transition to and from active only concurrently, i.e., at the same position relative to remaining input sequences.

Instead of treating state transitions in an acyclic manner, it is also possible to allow multiple transitions from active back to inactive. This would allow certain (context dependent) deletions to occur at a unit cost. Such “exclusions” have rarely been considered in sequence alignment but are of some interest for structured RNAs (Schirmer and Giegerich, 2013). This idea may be of use in particular when sequences are provided with structural annotation and deletions of entire structural elements are to be scored in a special way.

Advances in computing technology now make it feasible to compute exact simultaneous solutions of alignment problems with more than two sequences. A combinatorial diversity of distinct alignment problems arises in this setting just by allowing to distinguish local and global ends separately for each input. We suspect that additional variations on the theme are of interest, e.g., requiring additional constraints on pairwise overlaps. The framework and the implementation presented here is a first step towards a systematic exploration of this largely uncharted universe of alignments, many of which we suspect will be of practical use in computational biology.

ACKNOWLEDGMENTS

NR gratefully acknowledges the hospitality of the Santa Fe Institute, where parts of this work were performed, as well as travel support by the ASU-SFI Center for Biosocial Complex Systems. The Competence Center for Scalable Data Services and Solutions (ScaDS) Dresden/Leipzig is funded by BMBF grant 01IS14014B.

REFERENCES


