Universität Bielefeld
Forschungsschwerpunkt Mathematisierung —
Strukturbildungsprozesse

Materialien/Preprints 102

Improving the Divide-and-Conquer Approach to Sum-of-Pairs Multiple Sequence Alignment

Jens Stoye
Andreas W. M. Dress
Sören W. Perrey

Bielefeld 1996

Classification code(s) according to the
1980 Mathematics Subject Classification of
Mathematical Reviews:

92-04
92A10
68K05

Forschungsschwerpunkt Mathematisierung —
Strukturbildungsprozesse
Universität Bielefeld
Postfach 10 01 31
D-33501 Bielefeld

Telefon: (05 21) 1 06-47 64
Fax: (05 21) 1 06-60 07

e-mail:
fsp@mathematik.uni-bielefeld.de

Improving the Divide-and-Conquer Approach to Sum-of-Pairs Multiple Sequence Alignment

Jens Stoye, Andreas W. M. Dress
Research Center for Interdisciplinary Studies on Structure Formation (FSPM)
University of Bielefeld, Postfach 10 01 31, D-33501 Bielefeld, Germany
{stoye,dress}@mathematik.uni-bielefeld.de

Sören W. Perrey
Department of Mathematics, Massey University, Palmerston North, New Zealand
S.W.Perrey@massey.ac.nz

Abstract

We consider the problem of multiple sequence alignment: given k sequences of length at most n and a certain scoring function, find an alignment that minimizes the corresponding "sum of pairs" distance score.

We generalize the divide-and-conquer technique described in [7] and [21], and present new ideas on how to use efficient search strategies for saving computer memory and accelerating the procedure for three or more sequences. Resulting running times and memory usage are shown for several test cases.

1 Introduction

Multiple sequence alignment is an important problem in computational molecular biology, and many algorithms have been presented in this area of research (see [18], [15], [5], or [13] for a survey). Since the problem of computing optimal alignments with respect to the "sum-of-pairs" criterion is NP-hard [23], many approximative algorithms have been proposed (e.g. [9], [11], [20], or [2]). Unfortunately, almost all of these methods either exhibit a prohibitive computational complexity or yield biologically unplausible results. With our algorithm, we try to contribute to improving this situation.

2 Definitions

Let us consider a finite alphabet $\mathcal{A}$, k sequences $s_1, s_2, \ldots, s_k$ over $\mathcal{A}$ of length $n_1, n_2, \ldots, n_k$, respectively, and an additional letter, say '$-$', not contained in $\mathcal{A}$ symbolizing gaps. An alignment of $s_1, s_2, \ldots, s_k$ is given by a $k \times N$ matrix $M = (m_{ij})_{1 \le i \le k, 1 \le j \le N}$ for some $N \le \sum_{i=1}^{k} n_i$ with entries $m_{ij} \in \mathcal{A} \cup \{-\}$ subject to the following constraints: it does not contain any column consisting of gaps only and, for each $i = 1, 2, \ldots, k$, the row $(m_{i1}, m_{i2}, \ldots, m_{iN})$ reproduces the sequence s_i upon eliminating all of its gap letters.

The *weighted sum of pairs* multiple sequence alignment problem can now be described as follows (cf. [4]): Given $s_1, s_2, \ldots, s_k$, and given a scoring function $D : (\mathcal{A} \cup \{-\})^2 \to \mathbf{R}$, defined on all possible pairs of letters, find an *optimal* alignment M, i.e. an alignment that minimizes

$$w(M) := \sum_{1 \le p < q \le k} \left(\alpha_{p,q} \cdot \sum_{j=1}^{N} D(m_{pj}, m_{qj}) \right),$$

where the $\alpha_{p,q}$ are sequence-dependent (non-negative) weight factors reflecting e.g. phylogenetic relationship.

In our program, the weight factors $\alpha_{p,q}$ are calculated by the optimal (distance) scores $w_{op}(s_i, s_j)$ of the pairwise alignments:

$$\alpha_{p,q} := \frac{\min_{1 \le i < j \le k} \{ w_{op}(s_i, s_j) \}}{w_{op}(s_p, s_q)},$$

so that $\alpha_{p,q} = 1$ if and only if the pair of sequences (s_p, s_q) has lowest alignment score over all possible pairs of sequences under consideration.

Note that, depending on the context, more general scores are being considered, e.g. scores which use more sophisticated penalties for segments of consecutive gap letters or which do not penalize gaps at the beginning and/or the end of the alignment. The way our algorithm can be adapted to handle such score functions will be discussed in Section 5.

It is well known that the above mentioned multiple sequence alignment problem can be solved optimally by *dynamic programming* with a running time proportional to $(2^k - 1) \cdot \prod_{i=1}^{k} n_i$, searching for a shortest alignment path in a directed graph with $\prod_{i=1}^{k} n_i$ vertices. Faster variants and a number of speedups of dynamic programming have therefore been proposed (see for instance [14], and [4]). Unfortunately, most of these approaches still need quite large computing time and computer memory when applied to more than, say, six protein sequences of average length 300.

3 The Basic Algorithm

A first outline of our algorithm has been presented at the *Third International Conference on Intelligent Systems for Molecular Biology* in Cambridge, England (cf. [7]), a more detailed discussion of the procedure, restricted to three sequences, can be found in [21]. Here, we present the algorithm for the first time generalized for k sequences, give additional improvements which reduce the required memory by several orders of magnitude making the algorithm considerably more efficient in practice, and we give detailed results concerning running time and memory requirement.

The algorithm works according to the well-known *divide-and-conquer* principle: each of the given k sequences is being cut at an appropriately chosen site somewhere near to its midpoint, this way reducing the original alignment problem to the two subproblems of aligning the two resulting groups of prefix and suffix subsequences, respectively. These will be handled by the same procedure in a recursive manner. The recursion stops when the remaining subsequences are short enough to be aligned by a standard procedure, – in our implementation, we use `MSA` [12],[8].

A simple and effective stopping criterion for the recursion is a threshold L for the length of the shortest of the (sub)sequences under consideration. Other stopping criteria have been considered as well, but did not yield any significant improvement of our procedure.

The main problem is to find a tuple of slicing sites $(c_1, c_2, \ldots, c_k)$ so that the simple concatenation of the two optimal alignments of the prefixes $s_1(\leq c_1), s_2(\leq c_2), \ldots, s_k(\leq c_k)$ and the suffixes $s_1(> c_1), s_2(> c_2), \ldots, s_k(> c_k)$ forms an optimal alignment of the original sequences. (Here, $s_p(\leq c_p)$ denotes the prefix subsequence of s_p with indices running from 1 to c_p and $s_p(> c_p)$ denotes the suffix subsequence of s_p with indices running from $c_p + 1$ to n_p, $1 \leq p \leq k$.)

Obviously, for any fixed site c_1, there exists a $(k-1)$-tuple $(c_2, \ldots, c_k)$, such that $(c_1, c_2, \ldots, c_k)$ forms an optimal k-tuple of slicing sites. Unfortunately, finding these *ideal* slicing points requires searching the whole k-dimensional hypercube, taking the same time as the standard dynamic programming procedure. So, of course, this is not the method of choice.

Instead, our algorithm tries to find near-to-optimal, so-called *optimal* slicing sites, that are based on pairwise sequence comparisons. More precisely, we use the dynamic programming procedure which we apply to all pairs of sequences (s_p, s_q). The result-

ing score matrices for pairwise alignment give rise to *secondary matrices*

$$C_{s_p,s_q}[c_p, c_q] \;:=\; w_{op}\,(\,s_p(\leq c_p), s_q(\leq c_q)) + w_{op}\,(\,s_p(>c_p), s_q(>c_q)) - w_{op}\,(\,s_p,\, s_q\,)$$

imposed by forcing the alignment path to run through a particular vertex (c_p, c_q) $(1 \leq p < q \leq k)$. Similar matrices have been considered in the context of *dot plots* in [3] and [22]. The calculation of C_{s_p,s_q} can be performed by computing *forward* and *reverse* matrices in a similar way as it is described in [16] and [17], respectively.

Note that there exists, for every fixed $\hat{c}_p$, at least one slicing site $c_q(\hat{c}_p)$ with $C_{s_p,s_q}[\hat{c}_p, c_q(\hat{c}_p)] = 0$. This follows from the fact that the vertices on an optimal alignment path are precisely those with no additional charge and that (the projection of) every alignment path meets at least once every position of each sequence.

To search now for a good k-tuple of slicing sites, we try to estimate the *additional charge* imposed by forcing the *multiple* alignment path of the sequences through the particular vertex $(c_1, c_2, \ldots, c_k)$ in the whole (k-dimensional) box associated with the corresponding alignment problem. To this end, we use a weighted sum of secondary charges over all projections (c_p, c_q) as such an estimate: we put

$$C(c_1, c_2, \ldots, c_k) \;:=\; \sum_{1 \leq p < q \leq k} \alpha_{p,q} \cdot C_{s_p,s_q}[c_p, c_q].$$

Our proposition is now, that *optimal* slicing sites, i. e. $(c_1, c_2, \ldots, c_k)$ that minimize $C(c_1, c_2, \ldots, c_k)$, result in very good, if not optimal multiple alignments.

In conclusion, this leads to the following general procedure:

Algorithm *d&c-align* $(\, s_1, s_2, \ldots, s_k, L\,)$

`If` $\min_{1 \leq p \leq k}\{n_p\} \leq L$
  `then` return the optimal alignment of $s_1, s_2, \ldots, s_k$ (using e.g. `MSA`);
  `else` return the concatenation of
    $d\&c\text{-}align(s_1(\leq c_1), s_2(\leq c_2), \ldots, s_k(\leq c_k), L)$
    and $d\&c\text{-}align(s_1(>c_1), s_2(>c_2), \ldots, s_k(>c_k), L)$;
    `where` $(c_1, c_2, \ldots, c_k) := calc\text{-}cut(s_1, s_2, \ldots, s_k)$.

How to realize the function *calc-cut*, which computes a k-tuple of optimal slicing sites $(c_1, c_2, \ldots, c_k)$ efficiently, will be described in the following section.

4 Efficiently Calculating the Slicing Sites

In a naive implementation, the search *calc-cut* for optimal slicing sites $(c_1, c_2, \ldots, c_k)$ needs time $\mathcal{O}(k^2 n^2 + n^{k-1})$, where n is the length of the longest of the sequences $s_1, s_2, \ldots, s_k$: the computation of all pairwise secondary matrices takes $\mathcal{O}(k^2 n^2)$ time and, for given $\hat{c}_1$, all possible combinations of $c_2, \ldots, c_k$ have to be checked to find the tuple that minimizes C in $\mathcal{O}(n^{k-1})$.

We now describe how we are able to reduce this time by several orders of magnitude in the average case. Even the memory required, for the naive version $\mathcal{O}(k^2n^2)$, can be reduced in most cases by a large factor.

The main improvement is based on a simple, but very effective idea: We first precalculate an estimation $\widehat{C}$ for $C(c_1, c_2, \ldots, c_k)$ by calculating only the scores of the pairwise alignments, using the space saving technique originally invented by Hirschberg [10], where only the memory for one column $col^i_{p,q}[j] := C_{s_p,s_q}[i,j] \quad (1 \leq p < q \leq k, 1 \leq i \leq n_p, 1 \leq j \leq n_q)$ of the dynamic programming matrix is needed at a time. This precalculation allows us to prune the search space enormously: Because the additional charge $C(c_1, c_2, \ldots, c_k)$ is a sum of non-negative numbers $\alpha_{p,q} \cdot C_{s_p,s_q}[c_p, c_q]$, it is possible to exclude a tuple of slicing sites $(c_1, c_2, \ldots, c_k)$, whenever one of the summands $\alpha_{p,q} \cdot C_{s_p,s_q}[c_p, c_q]$ is larger than the minimum $\widehat{C}$ found so far. In particular, for fixed $\hat{c}_1$, any c_p with $\alpha_{1,q} \cdot C_{s_1,s_p}[\hat{c}_1, c_p] \geq \widehat{C}$ can never lead to a smaller sum C.

With this in mind, a tuple of optimal slicing sites can be calculated as follows:

Function *calc-cut* $(\, s_1, s_2, \ldots, s_k \,)$

1. Fix $\hat{c}_1 := \lceil \frac{n_1}{2} \rceil$;
2. calculate and save columns $col^{\hat{c}_1}_{1,q}[j] := C_{s_1,s_q}[\hat{c}_1, j] \quad (2 \leq q \leq k, 1 \leq j \leq n_q)$;
3. locate slicing sites $\hat{c}_2, \ldots, \hat{c}_k$ such that $col^{\hat{c}_1}_{1,q}[\hat{c}_q] = 0 \quad (2 \leq q \leq k)$;
4. calculate the estimate

$$\widehat{C} := \sum_{1 \leq p < q \leq k} \alpha_{p,q} \cdot C_{s_p,s_q}[\hat{c}_p, \hat{c}_q] = \sum_{2 \leq p < q \leq k} \alpha_{p,q} \cdot C_{s_p,s_q}[\hat{c}_p, \hat{c}_q],$$

   where to calculate the entries $C_{s_p,s_q}[\hat{c}_p, \hat{c}_q]$ $(2 \leq p < q \leq k)$ only the $\mathcal{O}(n)$ memory for one column $col^{\hat{c}_1}_{p,q}$ is needed at a time, since no part of the matrices C_{s_p,s_q} has to be saved.

5. Calculate lower and upper borders l_q and u_q such that $\alpha_{1,q} \cdot col^{\hat{c}_1}_{1,q}[j] \geq \widehat{C}$ for all $j < l_q$ and for all $j > u_q$ $(2 \leq q \leq k)$. The intermediate segment $col^{\hat{c}_1}_{1,q}[l_q], \ldots, col^{\hat{c}_1}_{1,q}[u_q]$ forms the *relevant part* of each column $col^{\hat{c}_1}_{1,q}$.
6. Given these borders, compute and save the *relevant parts* of the matrices C_{s_p,s_q}, defined by $C_{s_p,s_q}[i,j]$ with $l_p \leq i \leq u_p$ and $l_q \leq j \leq u_q$.
7. Search for better slicing sites $(\hat{c}_1, c_2(\hat{c}_1), \ldots, c_k(\hat{c}_1))$ within the relevant parts of the columns $col^{\hat{c}_1}_{1,q}$ and of the matrices C_{s_p,s_q}. Thereby, the sum C can be computed step by step and the search can be stopped, if an intermediate result is larger than $\widehat{C}$. During this search, with decreasing values of $\widehat{C}$, the relevant part of the columns $col^{\hat{c}_1}_{1,q}$ can possibly be further reduced, diminishing the search space even more.

As described in [21], neither the input order of the sequences nor a deviation from the restriction imposed by fixing $\hat{c}_1$ on exactly $\lceil \frac{n_1}{2} \rceil$ have a significant influence on the behavior of the procedure.

Obviously, the worst case time and space complexity of this approach still remains $\mathcal{O}(k^2n^2 + n^{k-1})$ and $\mathcal{O}(k^2n^2)$, respectively, for the (very improbable) case where the borders l_p and u_p can never be increased or decreased, respectively. But for biological sequences, the effect is enormous. Especially, for calculating the first tuple of slicing sites in the recursion, which takes far the longest time of all cut-site computations, the reduction from n to the length $r := \max_{p=2,\ldots,k}\{u_p - l_p + 1\}$ of the longest of the remaining relevant parts of the columns, is usually greater than $100 : 1$ for small k, yielding memory savings for the matrices of at least four orders of magnitude and reducing the expected time and space complexity to $\mathcal{O}(k^2n^2 + r^{k-1})$ and $\mathcal{O}(kn + k^2r^2)$, respectively. More detailed results concerning exact running times and memory usage are presented in Section 6.

5 Further Improvements

In the actual implementation of *d&c-align*, the program `MSA` is called for every single group of subsequences containing at least one subsequence of length at most L. It would be more elegant to calculate all the slicing sites first (which is possible without the knowledge of any of the sub-alignments) and then call `MSA` only once with the extra information concerning the slicing sites. Although `MSA` is able to handle fixed parts of the alignment of length one or longer, it still needs to be extended so as to accept fixed slicing sites, which, from this point of view, can be seen as fixing an alignment of zero length. Upon our request, the authors of `MSA` are working on such an extension [19].

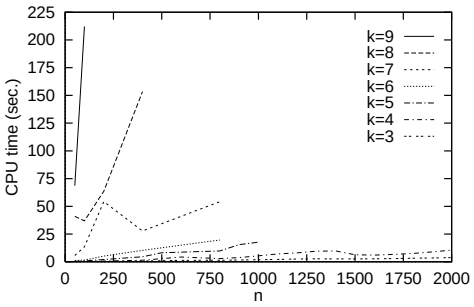
Another advantage expected from this approach is that, with such a feature, `MSA`'s ability of dealing with biologically more reasonable *quasi-natural* gap costs [1] and with a score function, which does not penalize gaps at the beginning and/or the end of the alignment, will be carried over automatically to our procedure.

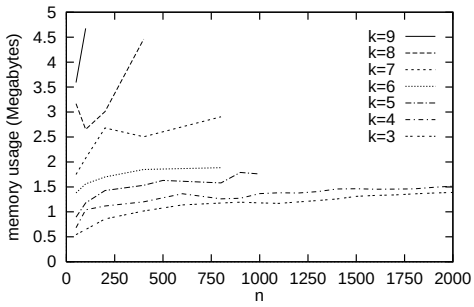
To improve the quality of the alignments further, we propose a *windowing approach* to correct the obtained alignments in the proximity of division sites. We suggest to choose a window width W depending on the threshold L. Laying such a window across each slicing site, one may search for an optimal re-alignment of the corresponding subsequences, again using `MSA`.

6 Results

From earlier measurements, we obtained a value of $L := 40$ for the stopping criterion to guarantee near-to-optimal results: the score of the alignments calculated in the following test cases differs by at most 1 percent from the score of the optimal alignment (in cases where we were able to check this with `MSA`).

To obtain the exact time and memory usage, our algorithm has been applied to several sets of sequences, obtained by a stochastical mutation process on random sequences of different lengths n and alphabet size 20, denoting the set of amino acids. The sequences have a pairwise identity between 15 and 25 percent. All shown results are averages over 10 different sequence sets. As a scoring function D we used the PAM-250 distance matrix [6] with positive entries between 0 and 24, as it is implemented in `MSA`.





As can be seen from these results, very fast calculation of high-quality alignments with rather moderate memory usage is possible with *d&c-align* for up to seven sequences. As examinations of several executions of the function *calc-cut* have shown, the still rather long running time for eight and more sequences depends (in addition to the longer `MSA`-runs) on the estimate $\widehat{C}$, which is not good enough in these cases.

k	3	4	5	6	7	8	9
average $\frac{r}{n}$	0.009	0.023	0.073	0.170	0.242	0.389	0.608

Finally, we have investigated whether the alignment scores improve upon using the windowing approach mentioned above. The following table shows for four sets of random sequences with different k and n the relative error of our score (i.e. the percentage of the optimal score by which the score obtained by our algorithm exceeds the optimal one) for different values of L and W.

k	n	L	$W=0$	$W=\frac{1}{4}L$	$W=\frac{1}{2}L$	$W=L$
3	100	60	0.00%	0.00%	0.00%	0.00%
		40	0.12%	0.00%	0.00%	0.00%
		20	0.47%	0.35%	0.25%	0.25%
3	300	60	0.21%	0.18%	0.18%	0.08%
		40	0.21%	0.18%	0.18%	0.17%
		20	0.36%	0.35%	0.32%	0.17%
4	100	60	0.28%	0.12%	0.12%	0.00%
		40	0.28%	0.18%	0.17%	0.09%
		20	0.34%	0.18%	0.18%	0.17%
5	100	60	0.41%	0.41%	0.41%	0.41%
		40	0.46%	0.42%	0.42%	0.42%
		20	0.69%	0.69%	0.50%	0.50%

7 Acknowledgements

The authors wish to thank Robert Giegerich and Udo Tönges for helpful comments on an earlier version of this paper. Part of this work is supported by the German Ministry for Education, Science, Research, and Technology (BMBF) under grant number 01 IB 301 B4.

References

[1] S. F. Altschul. Gap Costs for Multiple Sequence Alignment. *J. theor. Biol.*, 138:297–309, 1989.

[2] V. Bafna, E. L. Lawler, and P. A. Pevzner. Approximation Algorithms for Multiple Sequence Alignment. In Crochemore, M. and Gusfield, D., editor, *Combinatorial Pattern Matching, 5th Annual Symposium, CPM 94, Asilomar, CA, USA, June 5-8, 1994. Proceedings*, number 807 in Lecture Notes in Computer Science, pages 43–53, Berlin, 1994. Springer Verlag.

[3] D. R. Boswell and A. D. McLachlan. Sequence Comparison by Exponentially-Damped Alignment. *Nucl. Acids Res.*, 12(1):457–464, 1984.

[4] H. Carrillo and D. Lipman. The Multiple Sequence Alignment Problem in Biology. *SIAM J. Appl. Math.*, 48(5):1073–1082, 1988.

[5] S. C. Chan, A. K. C. Wong, and D. K. Y. Chiu. A Survey of Multiple Sequence Comparison Methods. *Bull. Math. Biol.*, 54(4):563–598, 1992.

[6] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A Model of Evolutionary Change in Proteins. In M. O. Dayhoff, editor, *Atlas of Protein Sequence and Structure*, volume 5, suppl. 3, pages 345–352. National Biomedical Research Foundation, Washington, D.C., 1979.

[7] A. W. M. Dress, G. Füllen, and S. W. Perrey. A Divide and Conquer Approach to Multiple Alignment. In *Proc. of the Third Conference on Intelligent Systems for Molecular Biology, ISMB 95*, pages 107–113. AAAI Press, Menlo Park, CA, USA, 1995.

[8] S. K. Gupta, J. D. Kececioglu, and A. A. Schäffer. Improving the Practical Space and Time Efficiency of the Shortest-Paths Approach to Sum-of-Pairs Multiple Sequence Alignment. *J. Comp. Biol.*, 2(3):459–472, 1995.

[9] D. Gusfield. Efficient Methods for Multiple Sequence Alignment with Guaranteed Error Bounds. *Bull. Math. Biol.*, 55(1):141–154, 1993.

[10] D. S. Hirschberg. A Linear Space Algorithm for Computing Maximal Common Subsequences. *Communications of the ACM*, 18(6):341–343, 1975.

[11] J. Kececioglu. The Maximum Weight Trace Problem in Multiple Sequence Alignment. In Apostolico, A. and Crochemore, M. and Galil, Z. and Manber, U., editor, *4th Annual Symposium, CPM 93, Padova, Italy, June 2-4, 1993. Proceedings*, number 684 in Lecture Notes in Computer Science, pages 106–119, Berlin, 1993. Springer Verlag.

[12] D. J. Lipman, S. F. Altschul, and J. D. Kececioglu. A Tool for Multiple Sequence Alignment. *Proc. Natl. Acad. Sci. USA*, 86:4412–4415, 1989.

[13] M. A. McClure, T. K. Vasi, and W. M. Fitch. Comparative Analysis of Multiple Protein-Sequence Alignment Methods. *Mol. Biol. Evol.*, 11(4):571–592, 1994.

[14] M. Murata, J. S. Richardson, and J. L. Sussman. Simultaneous Comparison of Three Protein Sequences. *Proc. Natl. Acad. Sci. USA*, 82:3073–3077, 1985.

[15] E. W. Myers. An Overview of Sequence Comparison Algorithms in Molecular Biology. Technical Report TR 91-29, University of Arizona, Tucson, Department of Computer Science, 1991.

[16] E. W. Myers and W. Miller. Optimal Alignments in Linear Space. *CABIOS*, 4(1):11–17, 1988.

[17] D. Naor and D. L. Brutlag. On Near-Optimal Alignments of Biological Sequences. *J. Comp. Biol.*, 1(4):349–366, 1994.

[18] D. Sankoff and J. B. Kruskal, editors. *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison.* Addison-Wesley, Reading, MA, USA, 1983.

[19] A. A. Schäffer. Personal communication.

[20] J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: Improving the Sensitivity of Progressive Multiple Sequence Alignment through Sequence Weighting, Position-Specific Gap Penalties and Weight Matrix Choice. *Nucl. Acids Res.*, 22(22):4673–4680, 1994.

[21] U. Tönges, S. W. Perrey, J. Stoye, and A. W. M. Dress. A General Method for Fast Multiple Sequence Alignment. *Gene*, 172:GC33–GC41, 1996.

[22] M. Vingron and P. Argos. Determination of Reliable Regions in Protein Sequence Alignments. *Protein Engng.*, 3(7):565–569, 1990.

[23] L. Wang and T. Jiang. On the Complexity of Multiple Sequence Alignment. *J. Comp. Biol.*, 1(4):337–348, 1994.