Sequence Alignment

Kun-Mao Chao

Department of Computer Science and Information Engineering

National Taiwan University

Taipei, Taiwan 106.

Email: kmchao@csie.ntu.edu.tw

October 19, 2005

keywords: bioinformatics, computational molecular biology, dynamic programming, sequence comparison.

1 Introduction

In nature, even a single amino acid sequence contains all the information necessary to determine the fold of the protein. However, the folding process is still mysterious to us, and some valuable information can be revealed by sequence comparison. Take a look at the following sequence:

THETR UTHIS MOREI MPORT ANTTH ANTHE FACTS

What did you see in the above sequence? By comparing it with the words in the dictionary, we find the tokens "FACTS," "IMPORTANT," "IS," "MORE," "THAN," "THE," and "TRUTH." Then we figure out the above is the sentence "The truth is more important than the facts."

Even though we have not decoded the DNA and protein languages, the emerging flood of sequence data has provided us with a golden opportunity of investigating the evolution and function of biomolecular sequences. Sequence comparison plays a major role in this line of research, and thus becomes the most basic tool of bioinformatics.

Sequence comparison has wide applications to molecular biology, computer science, speech processing, and so on. In molecular biology, it is often used to reveal similarities among sequences, determine the residue-residue correspondences, locate patterns of conservation, study gene regulation, and infer evolutionary relationships. It helps us to fish for related sequences in databanks, such as the GenBank database. It can also be used for the annotation of genomes.

To be continued...

2 Sequence Alignment

The unfortunate scarcity of interaction between biologists and computer scientists is well illustrated by the parallel developments of dynamic-programming methods for comparing sequences. Such methods were independently discovered by biologists [22], computer scientists [29], and workers in other fields. (For a survey of the history, see [26]) The precise relationship between the methods developed by the two communities is somewhat

obscured by notational differences, most but not all of which are insignificant. Computer scientists typically make explicit the relationship between two sequences by producing a list of editing operations that converts one sequence to the other, while biologists prefer to see an alignment of the sequences. Moreover, mathematically-oriented researchers find it natural to compare sequences using a distance "metric", *i.e.*, where a sequence is at distance zero from itself and large values of the measure correspond to highly divergent sequences. On the other hand, biologists tend to think in terms of similarity scores, i.e., a sequence has a very high similarity score when compared with itself, similar sequences have a smaller, but still positive, similarity score, and a pair of unrelated sequences has a negative score. It is tempting to believe that there is some kind of "duality" between minimizing a distance measure and maximizing a similarity score that makes it immaterial which one is adopted. Indeed, this is true under certain circumstances [28], though not for "local" alignments (see below).

Besides needing flexible ways of scoring alignments, biologists typically want to compute kinds of alignments that are not equivalent to the models studied by computer scientists. Computer scientists generally consider a problem equivalent to *global* alignment, i.e., computing an optimal alignment that is required to extend from the starts of the given sequences to their ends. Biologists frequently find it more useful to seek an alignment that is highest-scoring among all alignments between an arbitrary section of the first sequence and an arbitrary section of the second sequence [27], which is called the *local* alignment problem. Probably the most useful of these variations for aligning biological sequences is that of computing "n best non-intersecting" local alignments. Care

must be taken to formalize this notion in a way that allows subtle matches lying near much stronger (but perhaps less interesting) matches to be found, while still permitting efficient computation [32]. Earlier attempts to define the "n best local alignments problem" in terms of minimizing a measure of the distance between two sequences led to substantially more cumbersome algorithms.

On the computer science side, Hirschberg [18] discovered a method for computing longest common subsequences using only linear space (space proportional to the sum of the sequence lengths) rather than the naive quadratic space (space proportional to the product of the sequence lengths). Although space, rather than time, is often the constraining factor when applying dynamic-programming techniques to biological sequences (e.g., with a DNA sequence of length 50,000, a quadratic-space method uses billions of computer memory locations), biologists didn't discover the technique for themselves. While Hirschberg's original formulation was for alignment scores that are unrealistically simple for applications in biology, Myers and Miller [21] (also [20]) extended the approach to affine gap costs. Moreover, linear-space methods have been developed for the "n best local alignment problem" [19].

2.1 Global alignment

Following its introduction by Needleman and Wunsch [22], dynamic programming has become the method of choice for "rigorous" alignment of DNA and protein sequences. For a number of useful alignment-scoring schemes, this method is guaranteed to produce an alignment of two given sequences having the highest possible score.

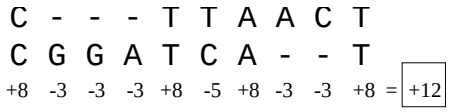

Figure 1: We assume the following simple scoring scheme: $match = 8$, $mismatch = -5$, and $g = 3$. That is, $w(a, b) = 8$ if a and b are the same; $w(a, b) = -5$ if a and b are different; and the gap penalty for each gap symbol is 3.

Given two sequences $A = \langle a_1, a_2, \ldots, a_M \rangle$, and $B = \langle b_1, b_2, \ldots, b_N \rangle$, an *alignment* of A and B is obtained by introducing dashes into the two sequences such that the lengths of the two resulting sequences are identical and no column contains two dashes. Let Σ denote the input symbol alphabet. To simplify the presentation, we employ a very simple scoring scheme as follows. A score $w(a, b)$ is defined for each $(a, b) \in \Sigma \times \Sigma$. Each gap symbol is penalized by a constant g. The score of an alignment is the sum of w scores of all columns with no dashes minus the penalties of the gaps. Figure 1 gives an example of an alignment's score. An *optimal alignment* is an alignment that maximizes the score. By global alignment we mean that both sequences are aligned globally, i.e., from their first symbols to their last.

It is quite helpful to recast the problem of aligning two sequences as an equivalent problem of finding a maximum-score path in a certain graph, as has been observed by a number of authors [7]. Figure 2 gives an example.

Let $S(i, j)$ denote the score of an optimal alignment between $\langle a_1, a_2, \ldots, a_i \rangle$, and $\langle b_1, b_2, \ldots, b_j \rangle$. With proper initializations, $S(i, j)$ can be computed by the following recurrence:

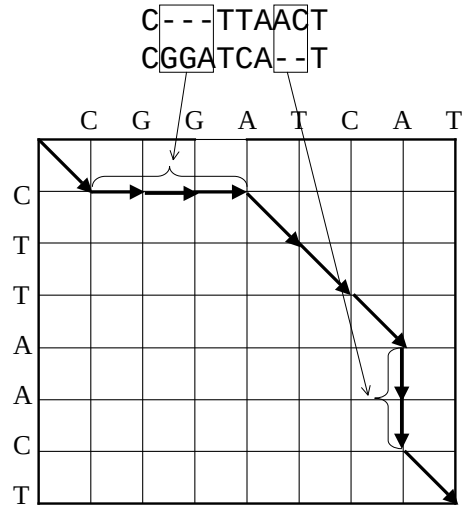

Figure 2: An alignment corresponds to a path in an alignment graph. The substitution aligned pairs, insertion aligned pairs, and deletion aligned pairs correspond to diagonal edges, horizontal edges, and vertical edges, respectively.

$$S(i,j) = \max \begin{cases} S(i-1,j) - g \\ S(i,j-1) - g \\ S(i-1,j-1) + w(a_i, b_j) \end{cases}$$

Figure 3 explains the recurrence by showing that there are three possible ways entering into the grid point (i, j), and we take the maximum of their path weights. The value $S(M, N)$ is the score of an optimal alignment between sequences A and B.

Figure 4 illustrates the tabular computation for $S(i, j)$.

Figure 5 gives an optimal alignment that is corresponding to the backtracking in Figure 4.

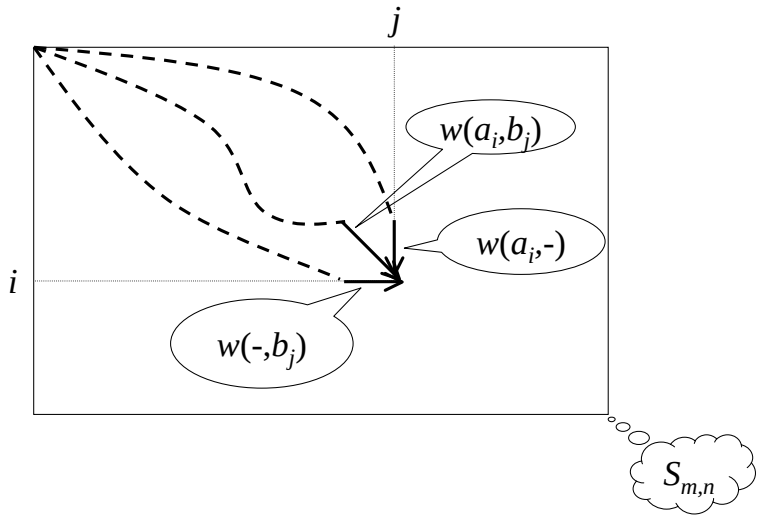

Figure 3: There are three ways entering the point i, j). In this simple scoring scheme, $w(a_i, -) = w(-, b_j) = -g$.

		C	G	G	A	T	C	A	T
	0	-3	-6	-9	-12	-15	-18	-21	-24
C	-3	8	5	2	-1	-4	-7	-10	-13
T	-6	5	3	0	-3	7	4	1	-2
T	-9	2	0	-2	-5	5	-1	-1	9
A	-12	-1	-3	-5	6	3	0	7	6
A	-15	-4	-6	-8	3	1	-2	8	5
C	-18	-7	-9	-11	0	-2	9	6	3
T	-21	-10	-12	-14	-3	8	6	4	14

Figure 4: The tabular computation. The maximum alignment score is 14. The shaded area is the trace of backtracking.

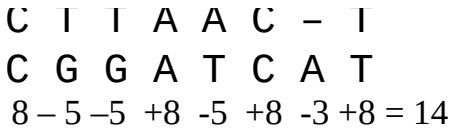

Figure 5: An optimal (global) alignment.

2.2 Local alignment

In many applications, a global (i.e., end-to-end) alignment of the two given sequences is inappropriate; instead, a local alignment (i.e., involving only a part of each sequence) is desired. In other words, one seeks a high-scoring path that need not terminate at the corners of the dynamic-programming grid [27]. The highest local alignment score can be computed as follows: :

$$S(i,j) = \max \begin{cases} 0 \\ S(i-1,j) - g \\ S(i,j-1) - g \\ S(i-1,j-1) + w(a_i, b_j) \end{cases}$$

The recurrence is quite similar to that for global alignment except the first entry "zero." For local alignment, we are not required to start from the source $(0,0)$. Therefore, if the scores of all possible paths ending at the current position are all negative, they are reset to zero because any point in the alignment graph could be the starting position of a local alignment. The largest value of $S(i,j)$ is the score of the best local alignment between sequences A and B.

Further complications arise when one seeks k best alignments, where $k > 1$. For computing an arbitrary number of non-intersecting and high-scoring local alignments, Waterman and Eggert [32] developed a very time-efficient method.

		C	G	G	A	T	C	A	T
	0	0	0	0	0	0	0	0	0
C	0	8	5	2	0	0	8	5	2
T	0	5	3	0	0	8	5	3	13
T	0	2	0	0	0	8	5	2	11
A	0	0	0	0	8	5	3	13	10
A	0	0	0	0	8	5	2	11	8
C	0	8	5	2	5	3	13	10	7
T	0	5	3	0	2	13	10	8	18

Figure 6: The maximum score is 18. The shaded area is the trace of backtracking.

```
A – C - T
A T C A T
8-3+8-3+8 = 18
```

Figure 7: An optimal local alignment.

Figure 6 illustrates the tabular computation for local alignment.

Figure 7 gives an optimal local alignment that is corresponding to the backtracking in Figure 6.

To attain greater speed, biologists have employed the strategy of building alignments from alignment fragments [33, 34]. For example, one could specify some fragment length $k \geq 1$ and work with fragments consisting of a segment of length at least k that occurs in both sequences. With protein sequences, it might well work better to begin with inexact but high-scoring matches, such as those used by the *blast* program [2] for other purposes. In any case, algorithms that optimize the score over alignments constructed from fragments can run faster that algorithms than optimize over all possible alignments.

Alignments constructed from fragments (or often just the alignments' scores) have been very successful in initial filtering criteria within programs that search a sequence database for matches to a query sequence; database sequences whose alignment score with the query sequence falls below a threshold are ignored, and the remainder are subjected to a slower but higher-resolution alignment process. Moreover, the high-resolution process can be made more efficient by restricting the search to a "neighborhood" of the alignment-from-fragments [24, 6, 8].

The idea of filtration used in both FASTA and BLAST is based on the observation that a good alignment usually includes short identical or very similar fragments. FASTA [24] uses a multi-step approach to finding local alignments: (1)find runs of identities, and identify regions with the highest density of identities; (2)re-score using PAM matrix, and keep top scoring segments; (3)eliminate segments that are unlikely to be part of the alignment; and (4)optimize the alignment in a band.

The first version of BLAST finds ungapped alignments [2]. First, it builds a hash table for the query sequence (see Figure 8). Then it scans the database for hits. Finally, it extends hits as an ungapped alignment in both diagonal directions. New version of BLAST accelerates the process of extending hits (which consumes almost 90% of the computation time in the first version of BLAST), and is now able to deliver some important gapped alignments.

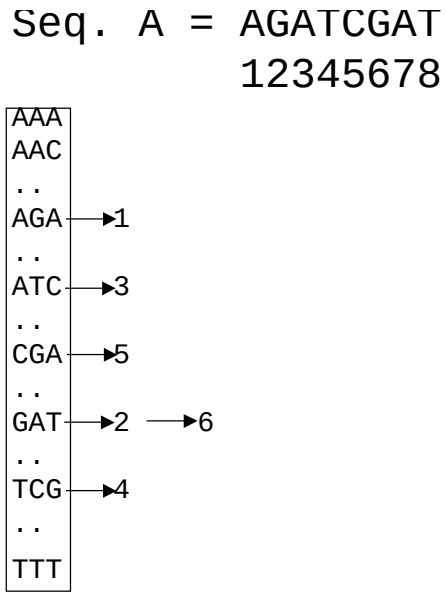

Figure 8: A hash table for finding exact matches of size 3.

2.3 Affine gap penalties

Biologists need more general measurements of sequence relatedness than are typically considered by computer scientists. The most popular formulation in the computer science literature is the "longest common subsequence problem," which is equivalent to scoring alignments by simply counting the number of exact matches. For comparing protein sequences, it is important to permit the bonus awarded for aligning two symbols to depend on the particular symbol pair [12]. For both DNA and protein sequences, it is standard to penalize a long gap (i.e., deletion from one of the sequences) less than the sum of the penalties for a set of shorter gaps of the same total length [14]. This is usually accomplished by charging $g + t \times e$ for a gap of length t. Thus the "gap-open penalty" g is assessed for every gap, regardless of length, and an additional "gap-extension penalty" e is charged for every sequence entry in the gap. Such penalties are called *affine* gap penalties (See Figure 9). Gotoh [15] showed how to efficiently compute

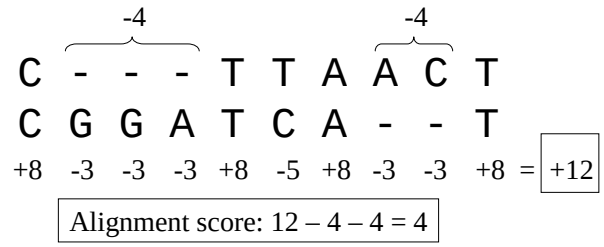

Figure 9: tba

optimal alignments under such scoring rules.

Even more general models for quantifying sequence relatedness have been proposed. For example, it is sometimes useful to have the penalty for adding a symbol to a gap depend on the position of the gap within the sequence [16], which is motivated by the observation that insertions in certain regions of a protein sequence can be much more likely than at other regions. Another generalization is to let the incremental gap cost $\delta_i = c_{i+1} - c_i$, where a k-symbol gap costs c_k, be a monotone function of i, e.g., $\delta_1 \geq \delta_2 \geq \cdots$ [31, 20]. There is some evidence that monotonic gap-extension penalties incorrectly model nature in certain circumstances [23].

Selection of the scoring parameters is often a major factor affecting the usefulness of the computed alignments, since it determines which sequence regions will be considered non-aligning (e.g. because of negative scores) and what relationships will be assigned between aligning regions. Appropriateness of scoring parameters depends on several factors, including evolutionary distance between the species being compared.

2.4 Space-saving strategies

A dynamic-programming strategy for sequence alignment first proposed in 1975 by Dan Hirschberg can be adapted to yield a number of extremely space-efficient algorithms [18]. Specifically, these algorithms align two sequences using only "linear space", i.e., an amount of computer memory that is proportional to the sum of the lengths of the two sequences being aligned.

We briefly describe Hirschberg's linear-space alignment algorithm; the algorithm delivers an explicit optimal alignment, not merely its score. Readers can refer to [7] for more space-saving strategies. First, make a linear-space "forward" score-only pass, stopping at the middle row, i.e., row $mid = \lfloor M/2 \rfloor$. Then make a linear-space "backward" score-only pass, again stopping at the middle row. Thus, for each point along the middle row, we now have the optimal score from $(0, 0)$ to that point and the optimal score from that point to (M, N). Adding those numbers gives the optimal score over all paths from $(0, 0)$ to (M, N) that pass through that point. A sweep along the middle row, checking those sums, determines a point (mid, j) where an optimal path crosses the middle row. This reduces the problem to finding an optimal path from $(0, 0)$ to (mid, j) and an optimal path from (mid, j) to (M, N), which is done recursively.

Figure 10(A) shows the two subproblems and each of their "subsubproblems". Note that regardless of where the optimal path crosses the middle row, the total of the sizes of the two subproblems is just half the size of the original problem, where problem size is measured by the number of nodes. Similarly, the total sizes of all subsubproblems is a fourth the original size. Letting T be the size of the original, it follows that the total

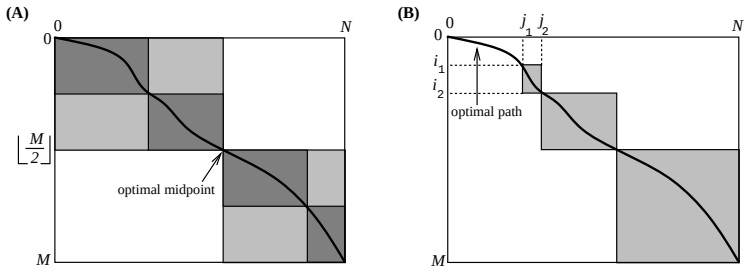

Figure 10: Hirschberg's linear-space approach.

sizes of all problems, at all levels of recursion, is at most $T + \frac{1}{2}T + \frac{1}{4}T \ldots = 2T$. Since computation time is directly proportional to the problem size, this approach will deliver an optimal alignment in about twice the time needed to compute merely its score.

Figure 10(B) shows a typical point in the alignment process. The initial portion of an optimal path will have been determined, and the current problem is to report the aligned pairs along an optimal path from (i_1, j_1) to (i_2, j_2).

2.5 Multiple sequence alignment

Simultaneous alignment of several sequences is among the most important problems in computational molecular biology [17]. In particular, accurate multiple alignment is critical for the use of DNA sequence comparisons to study gene regulation. In spite of the plethora of existing ideas and methods for multiple alignment, no available program seemed well-suited to the needs.

When simultaneously aligning more than two sequences, it is hoped that knowledge of appropriate parameters for pairwise alignment to lead immediately to appropriate settings for the multiple-alignment scoring parameters. Thus, it is desirable that a scoring scheme for multiple alignments that is intimately related to the pairwise alignment scores.

Of course, it is also necessary that the approach be amenable to a multiple-alignment algorithm that is reasonably efficient with computer resources, i.e., time and space.

To attain this tight coupling of pairwise and multiple alignment scores at reasonable computational expense, the algorithm has adopted the sum-of-pairs substitution scores and quasi-natural gap costs, as described by Altschul [1]. Some notation will help for a precise description of these ideas. Assume given are S_1, S_2, ..., S_m, each of which is a sequence of "letters." A *quasi-alignment* of those sequences is an $m \times n$ array of letters and dashes, such that removing dashes from row i leaves the sequence S_i for $1 \leq i \leq m$. An *alignment* is a quasi-alignment containing no null columns, i.e., columns consisting entirely of dashes. For each pair of sequences, say S_i and S_j, rows i and j of the m-way alignment constitute a pairwise quasi-alignment of S_i and S_j; removing any null columns produces a pairwise alignment of these sequences.

To score an m-way alignment α given appropriate parameters for the score, say $Score_{i,j}$, for pairwise alignments between each sequence pair (S_i, S_j), it is natural to define

$$Score(\alpha) = \sum_{i<j} Score_{i,j}(\alpha_{i,j})$$

where $\alpha_{i,j}$ is the pairwise alignment of S_i and S_j induced by α. However, as explained in detail by Altschul, this scoring scheme results in undesirable algorithmic complexity. Altschul further observed that this complexity can be reduced dramatically if for every pair of rows of the m-way alignment an additional gap penalty is assessed for each "quasi-gap", defined as follows. Fix a pair of rows and consider a gap, G, in the corresponding pairwise quasi-alignment, i.e., a run of consecutive gap symbols occurring in one of the

rows (the run should be extended in both directions until it hits a letter or the end of the sequence). If at least one dash in G is aligned with a letter in the other row, then G corresponds to a gap in the pairwise alignment (i.e., after discarding null columns), and hence is penalized. The other possibility is that every dash in G is aligned with a dash in the other sequence. If the gap in the other sequence starts strictly before and ends strictly after G, then G is called a *quasi-gap* and may be penalized; if either end of G is aligned to an end of the gap in the other sequence, then G is not penalized. See Figure 11.

In summary, a multiple alignment is scored as follows. For each pair of rows, say rows i and j, fix appropriate substitution scores $\sigma_{i,j}$ and a gap cost $g_{i,j}$. Then the score for the multiple alignment is determined by the above SP equation, where each $Score_{i,j}(\alpha_{i,j})$ is found by adding the *sigma* values for non-null columns of the pairwise quasi-alignment, and subtracting a gap penalty g for each gap and for each "progressive" quasi-gap (defined below).

For k sequences of length n, a straightforward mutliple sequence alignment algorithm runs in $O(n^k)$ time. In fact, it has been shown to be NP-Complete by Wang and Jiang [30]. Therefore, the exact multiple alignment algorithms for many sequences are not feasible. Some approximation algorithms are given. For example, Bafna et al. [4] gave an algorithm with approximation ratio $2 - l/k$ for any fixed l.

A heuristic approach was proposed by Feng and Doolittle [13]. It iteratively merges the most similar pairs of sequences/alignments, as illustrated by Figure 12, following the principle "once a gap, always a gap." The time for progressive alignment in most cases

seq1 : **G C − T C**
seq2 : **A − − − C**
seq3 : **G − A T C**

	match	mismatch	gap pair	gap	quasi−gap	Score
seq1 : **G C − T C** *seq2* : **A − − − C**	1	1	2	1	1	−7
seq1 : **G C − T C** *seq3* : **G − A T C**	3	0	2	2	0	−4
seq2 : **A − − − C** *seq3* : **G − A T C**	1	1	2	1	0	−4
						−15

Figure 11: Scoring a multiple alignment. All pairs of rows are scored as described in the text (all quasi-gaps are penalized) and the multiple alignment's score is the sum of the pairwise scores. The notion of a "quasi-gap" is defined in the text; it is introduced to reduce the complexity of computing a highest-scoring multiple alignment. For this example, pairwise alignments are scored by: match = 1, mismatch = -1, gap pair (one symbol is "-") = 0.5 and gap penalty = -3. Null columns (both entries "-") always score 0. (In general, a different scoring scheme can be used for each pair of rows.) The alignments shown are not intended to be optimal.

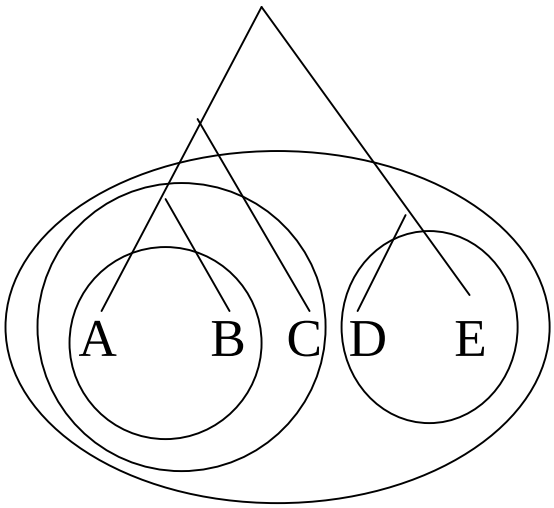

Figure 12: Progressive alignment.

is roughly the order of the time for computing all pairwise alignment, i.e., $O(k^2n^2)$.

Acknowledgements

The work was partially done while the author was visiting the Institute for Mathematical Sciences, National University of Singapore in 2002. The visit was supported by the Institute and by a grant (No. 01/1/21/19/217) from BMRC-NSTB of Singapore. The author would like to thank Drs. Louis Chen and Louxin Zhang for their support and encouragement.

References

[1] S.F. Altschul, Gap costs for multiple sequence alignment. *J. theor. Biol.* 138 (1989), 297-309.

[2] S.F. Altschul, W. Gish, W. Miller, E. Myers and D. Lipman, A basic local alignment search tool. *J. Mol. Biol.* 215 (1990), 403–410.

[3] S.F. Altschul, T. L. Madden, A.A. Schäffer, J. Zhang, Z. Zhang, W. Miller and D. Lipman, Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research* 25 (1997), 3389–3402.

[4] V. Bafna, E.L. Lawler, and P.A. Pevzner, Approximation algorithms for multiple sequence alignment. *Theoretical Computer Science* 182 (1997), 233–244.

[5] J. Bentley, *Programming Pearls*, Addison-Wesley, Reading, Massachusetts, 1986.

[6] K.-M. Chao, W. R. Pearson, and W. Miller, Aligning two sequences within a specified diagonal band. *CABIOS* 8 (1992), 481–487.

[7] K.-M. Chao, R.C. Hardison, and W. Miller, Recent developments in linear-space alignment methods: a survey. *Journal of Compuational Biology* 1 (1994), 271–291.

[8] K.-M. Chao, and W. Miller, Linear-space algorithms that build local alignments from fragments. *Algorithmica* 13 (1995), 106–134.

[9] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, the MIT Press, Cambridge, Massachusetts, 1994.

[10] R.F. Doolittle, ed. *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, Methods in Enzymology, 183, Academic Press, New York, 1990.

[11] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, San Francisco, 1979.

[12] D.F. Feng, M. S. Johnson and R. F. Doolittle, Aligning amino acid sequences: comparison of commonly used methods. *J. Mol. Evol.* 21 (1985), 112–125.

[13] D.F. Feng, and R.F. Doolittle, Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution* 25 (1987), 351–360.

[14] W.M. Fitch, and T. F. Smith, Optimal sequence alignments. *Proc. Natl. Acad. Sci. USA* 80 (1983), 1382–1386.

[15] O. Gotoh, An improved algorithm for matching biological sequences. *J. Mol. Biol.* 162 (1982), 705–708.

[16] M. Gribskov, R. Luthy, and D. Eisenberg, Profile analysis. In R.F. Doolittle (ed.) *Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, Methods in Enzymology, 183, Academic Press, New York, (1990), 146–159.

[17] F.G. Higgins, J.D. Thompson, and T.J. Gibson, Using CLUSTAL for multiple sequence alignment. *Methods in Enzymology* 266 (1996), 383–402.

[18] D.S. Hirschberg, A linear space algorithm for computing maximal common subsequences. *Comm. Assoc. Comput. Mach.* 18 (1975), 341–343.

[19] X. Huang, and W. Miller A time-efficient, linear-space local similarity algorithm. *Advances in Applied Mathematics* 12 (1991), 337–357.

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

[21] E. Myers, and W. Miller Optimal alignments in linear space. *CABIOS* 4 (1988), 11–17.

[22] S.B. Needleman, and C. D. Wunsch, A general method applicable to the search for similarities in the amino acid sequences of two proteins. *J. Mol. Biol.* 48 (1970), 443–453.

[23] S. Pascarella, and P. Argos Analysis of insertions/deletions in protein structures. *J. Mol. Biol.* 224 (1992), 461–471.

[24] W.R. Pearson, and D. Lipman, Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci USA* 85 (1988), 2444–2448

[25] P.A. Pevzner, *Computational Molecular Biology: An Algorithmic Approach*, The MIT Press, Cambridge, 2000.

[26] D. Sankoff, and J. B. Kruskal (eds) *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparisons*, Addison-Wesley, Reading, Massachusetts, 1983.

[27] Smith, T. F. and M. S. Waterman, Identification of common molecular sequences. *J. Mol. Biol.* 147 (1981), 195–197.

[28] T.F. Smith, M. S. Waterman and W. M. Fitch, Comparative biosequence metrics. *J. Mol. Evol.* 18 (1981), 38–46.

[29] R.A. Wagner, and M. J. Fischer, The string-to-string correction problem. *J. Assoc. Comput. Mach.* 21 (1974), 168–173.

[30] L. Wang, and T. Jiang, On the complexity of multiple sequence alignment. *J. Comp. Biol.* 1 (1994), 337–348.

[31] M.S. Waterman, Efficient sequence alignment algorithms. *J. theor. Biol.* 108 (1984), 333–337.

[32] M.S. Waterman, and M. Eggert, A new algorithm for best subsequence alignments with application to tRNA-rRNA comparisons. *J. Mol. Biol.* 197 (1987), 723–728.

[33] W. Wilbur, and D. Lipman, Rapid similarity searches of nucleic acid and protein data banks. *Proc. Nat. Acad. Sci. USA* 80 (1983), 726–730.

[34] W. Wilbur, and D. Lipman, The context dependent comparison of biological sequences. *SIAM J. Appl. Math.* 44 (1984), 557–567.

[35] B.Y. Wu, G. Lancia, V. Bafna, K.-M. Chao, R. Ravi, and C.Y. Tang, A polynomial time approximation scheme for minimum routing cost spanning trees. *SIAM J. Comput.* 29 (2000), 761–778.