FRESCO: FLEXIBLE ALIGNMENT WITH RECTANGLE SCORING SCHEMES *

A.V. DALCA[1] AND M. BRUDNO[1,2]

1. Department of Computer Science, and 2. Donnelly Center for Cellular and Biomolecular Research, University of Toronto, Toronto, Canada
{dalcaadr, brudno}@cs.toronto.edu

While the popular DNA sequence alignment tools incorporate powerful heuristics to allow for fast and accurate alignment of DNA, most of them still optimize the classical Needleman Wunsch scoring scheme. The development of novel scoring schemes is often hampered by the difficulty of finding an optimizing algorithm for each non-trivial scheme. In this paper we define the broad class of *rectangle scoring schemes*, and describe an algorithm and tool that can align two sequences with an arbitrary rectangle scoring scheme in polynomial time. Rectangle scoring schemes encompass some of the popular alignment scoring metrics currently in use, as well as many other functions. We investigate a novel scoring function based on minimizing the expected number of random diagonals observed with the given scores and show that it rivals the LAGAN and Clustal-W aligners, without using any biological or evolutionary parameters. The FRESCO program, freely available at http://compbio.cs.toronto.edu/fresco, gives bioinformatics researchers the ability to quickly compare the performance of other complex scoring formulas without having to implement new algorithms to optimize them.

1. Introduction

Sequence alignment is one of the most successful applications of computer science to biology, with classical sequence alignment programs, such as BLAST[1] and Clustal-W[2], having become standard tools used by all biologists. These tools, developed while the majority of the available biological sequences were of coding regions, are not as effective at aligning DNA[3]. Consequently, the last ten years have seen the development of a large number of tools for fast and accurate alignment of DNA sequences. These alignments are typically not an end in themselves - they are further used as input to tools that do phylogeny inference, gene prediction, search for tran-

*This work is supported by an NSERC USRA and Discovery grants

scription factor binding sites, highlight areas of conservation, or produce another biological result.

Within the field of sequence alignment, several authors[4,5,6,7] have noted the distinction between the function that is used to score the alignment, and the algorithm that finds the best alignment for a given function. Given an arbitrary scoring scheme, it is normally easy to assign a score to an already-aligned pair of sequences, but potentially more complicated to yield a maximal-scoring alignment if the sequences are not aligned to begin with. While for some scoring schemes, such as edit distance or Needleman-Wunsch, the optimizing algorithm is simple to write once the scoring function is defined, in other cases, such as the DIALIGN scoring metric, it is trivial to score a given alignment, but the algorithm which one could use to compute the optimal alignment under the given metric may be difficult to devise. Because of this complexity, sequence alignment programs concentrate on a single scoring scheme, allowing the user to vary a range of parameters, but not the scheme itself.

Among the many DNA alignment programs developed over the last few years, most have attempted to use various heuristics to quickly optimize the Needleman-Wunsch metric. In this paper we propose algorithms and software to enable bioinformatics researchers to explore a plethora of richer scoring schemes for sequence alignments. First, we define the class of *rectangle scoring schemes*, which encompass a large number of scoring metrics, including Needleman-Wunsch, Karlin-Altschul E-value, DIALIGN, and many others. Secondly we demonstrate an efficient polynomial-time algorithm to compute the optimal alignment for an arbitrary rectangle scoring scheme, and present both provably optimal and non-optimal heuristics to speed up this search. Our algorithms are implemented in a tool, FRESCO, which can be used to investigate the efficacy of various rectangle scoring schemes. Finally, we illustrate two examples of scoring functions that produce accurate alignments without any prior biological knowledge.

2. Scoring Schemes

2.1. *Previous work*

The work on scoring an alignment is closely tied to the problem of defining a distance between two strings. Classical work on formulating such distances are due to Hamming[8] for ungapped similarities and Levenshtein[9] for similarity of sequences with gaps. The Needleman-Wunsch algorithm[10] expanded on Levenshtein's approach to allow for varying match scores, and mismatch and gap penalties. Notably, the Needleman-Wunsch algorithm, as described by the original paper, supports arbitrary gap functions and runs in $O(n^3)$ time. The special case of affine gaps being computable in quadratic time was demonstrated by Gotoh[11] in 1982. Most of the widely used DNA sequence alignment programs such as BLASTZ[12], AVID[13] and LAGAN[14] use the Needleman-Wunsch scoring scheme with affine gaps. The DIALIGN scoring scheme[4,5] is notable because it was one of the first scoring schemes that allowed for scoring not on a per-letter, but on a per-region (diagonal) basis. The score of a single diagonal was defined as the probability that the observed number of matches within a diagonal of a given length would occur by chance, and the algorithm sought to minimize the product of all these probabilities.

The Karlin-Altschul (KA) E-value[15], which estimates the expected number of alignments with a certain score or higher between two random strings, can be formulated not only as a confidence, but also as a scoring scheme, as was heuristically done in the OWEN program[16]. After the single best local alignment between the two sequences is found and fixed, the algorithm begins a search for the second best alignments, but restricts the location to be either before the first alignment in both sequences, or after in both. The KA E-value depends on the lengths of sequences being aligned, and because the effective sequence lengths are reduced by the first local alignment, the KA E-value of the second alignment depends on the choice of the first. The OWEN program, which uses the greedy heuristic, does not always return the optimal alignment under the KA E-value scoring scheme. Other alignment scoring schemes include scoring metrics to find an alignment which most closely matches a given evolutionary model. These can be heuristically optimized using a Markov Chain Montecarlo (MCMC) algorithm, for example the MCAlign program[17]. Alignment scoring schemes which are based on various interpretations of probabilistic models, e.g. the ProbCons alignment program that finds the alignment with the maximum expected number of correct matches, are another example. Within the context of alignments based on probabilistic models there has been work on methods to effectively learn the optimal values for the various parameters of the common alignment schemes using Expectation Maximization or other unsupervised learning algorithms[18].

2.2. *Rectangle Scoring Schemes*

In this section we will define the concept of a rectangle scoring scheme, and illustrate how some of the classic alignment algorithms are all special cases of such schemes. Consider a 2D matrix M defined under the dynamic programming paradigm, on whose axes we set the two sequences being aligned. We define a diagonal in an alignment as referring to a sequence of matched letters between two gaps. We define a diagonal's bounding rectangle as the sub-rectangle in M delimited by the previous diagonal's last match and the next diagonal's first match (Fig. 1a). Thus, a diagonal's bounding rectangle includes the diagonal itself as well as the preceding and subsequent gaps. A *rectangle scoring scheme* is one that makes use of gap and diagonal information from within this rectangle (such as the number of matches, area of the rectangle, lengths of the dimensions, etc), while the scores for all rectangles can be computed independently and then combined[a]. For example, Needleman-Wunsch[10] is one such scheme: the score of a rectangle is defined as the sum of match and mismatch scores for the diagonal, minus half of the gap penalties for the two gaps before and after the diagonal. The Karlin-Altschul E-value[15] ($E = -Kmne^{-\lambda s}$) is another example, as the E-value depends on m and n, the entire lengths of the two strings being compared. The DIALIGN scoring function is another example of a rectangle scoring scheme.

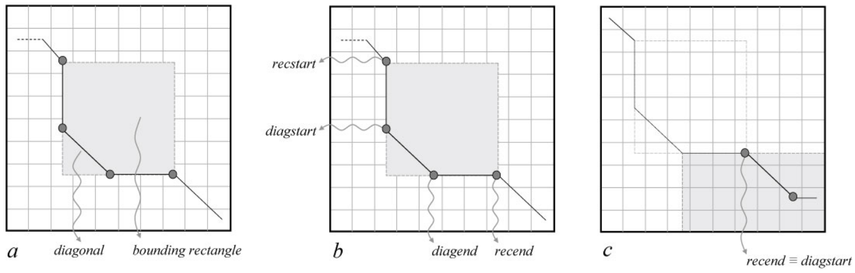

Figure 1. (a) Definition of a bounding rectangle of a diagonal - the rectangle in M delimited by the previous diagonal's last match and the next diagonal's first match. (b) Shows the 4 important points within a rectangle: *recstart* & *recend* - the start and end (top left and bottom right, respectively) points of a rectangle, and *diagstart* & *diagend* - the starting and ending points of a diagonal. (c) Note how a *recend* is equivalent to the next diagonal's *diagstart*.

[a] Currently FRESCO assumes the operation combining rectangle scores is addition or multiplication, as this is most often the case, but can be trivially modified to allow for any operation/formula.

3. Algorithm

In this section we will present an overview of the algorithm that we use to find the best alignment under an arbitrary rectangle scoring scheme. We will again make use of the 2D dynamic programming matrix M defined above, on whose axes we set the two sequences being aligned.

3.1. *Basic FRESCO Algorithm*

Given any rectangle scoring scheme, FRESCO computes an optimal alignment between two sequences. For clarity, we define *recstart* as the starting point of a rectangle and *recend* as the endpoint of a rectangle, and, similarly, *diagstart* and *diagend* points to be the starting and ending points of a diagonal (Fig. 1b). By definition of a rectangle of a given diagonal, a *recstart* is equivalent to the previous diagonal's *diagend* and a *recend* is equivalent to the next diagonal's *diagstart* (Fig. 1c).

The FRESCO algorithm can be explained within a slightly modified dynamic programming algorithm paradigm.

1. *Matrix.* First we create the dynamic programming matrix M with the two sequences on the axes.

2. *Recursion.* Here we describe the recursion relation. We iterate through the matrix M row-wise.

- **Terminology**: a *diagend* cell C can form a gap with several possible *recend* cells D (cells that come after C on the same row or column), as shown in Figure 2a. Note that this {C, D} pair can thus be part of a number of rectangles {A, B, C, D}, where A is the *recstart* and B is the *diagstart*. To view all of these, one would consider all the possible *diagstarts*, and for each, all the possible *recstarts*, as shown in Figure 2(b-d). We use this notion of a {C, D} pair and {A, B, C, D} rectangle below.
- **Invariant**: (true at the end of each step (i,j)). Let $C = M[i, j]$ and consider this cell as a possible *diagend*. We have computed, for each possible pair {C, D} described above, a rectangle representing the best alignment up to {C, D}.
- **Recursion**: Assume cell C is a *diagend*. For every cell D as described above:
    - ○ Find all the possible rectangles through {C,D}: {A, B, C, D} as described above (for every possible *diagstart* consider every possible *recstart*)
    - ○ For each rectangle $R =$ {A, B, C, D}, we will have computed the best alignment & associated score S_B up to {A, B} (via the invariant) and we can compute the score S_R of R alone via the current rectangle scoring scheme. Adding $S_T = S_B + S_R$ will give us the score of the best alignment through {A, B, C, D}.

- ○ After computing all the S_T (total) scores for each R, we take the maximum, giving us the optimal alignment & score up to {C, D}. This completes the recursion. For the purposes of recreating the alignment, we hold, for each {C, D} pair, a pointer to the optimal {A, B} choice.

3. *Computing the alignment.*

- · Let the final cell be denoted by F, $F \equiv M[m, n]$. We will have $m + n - 1$ pairs {C, F}, (where C is on the rightmost column of bottommost row) that will hold the best alignment and score up to {C, F}. Taking the maximum of these will give us the best alignment up to F. Having stored pointers from each {C, D} pair to its optimal {A, B} pair, we simply follow the pointers back through each rectangle up to $M[0, 0]$, thus recreating the alignment.

The proof of correctness is by induction and follows very similarly. The algorithm can be trivially modified to allow for unaligned regions by setting the diagonal score to the score of the maximum contiguous subsequence.

3.1.1. *Running Time and Resources*

Let the larger of the sequences be of length n. The algorithm iterates over all the points of the matrix M — $O(n^2)$ iterations. In the recursion, we look ahead at most $2n$ *recends* D and look back at no more than n *diagstarts* B. For each of these {B, C, D} sets, we search through at most $2n$ *recstarts* A. Thus we have $O(n^3)$ computation and $O(n)$ storage. Consequently we have an overall running time of $O(n^5)$ and storage of $O(n^3)$.

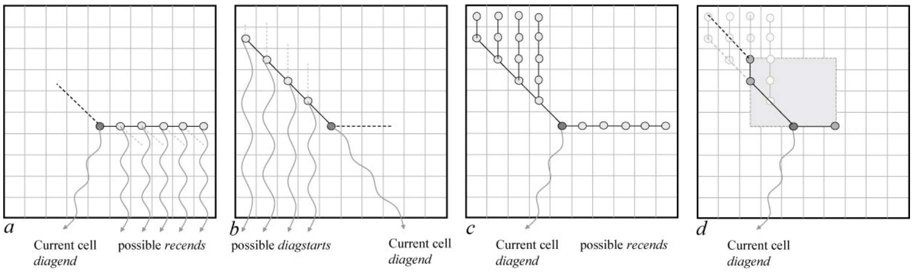

Figure 2. The figure illustrates the search, described in section 3.1, for the best rectangle assuming the current point acts as a *diagend*. For the current cell being considered (dark gray), referred to as C, (a) shows possible *recends* D; and hence pairings {C, D}. (D could also be on the same column as C). (b) illustrates the possible *diagstarts* (B) considered for each of these {C, D}. For each {B, C, D} set we have possibilities such as those shown in (c), all of which form rectangles {A, B, C, D} that go through the diagend C we begin with. We choose the optimal of these rectangles, as shown in (d).

3.2. *FRESCO Speed Ups*

Under most scoring schemes, a large portion of the calculations above become redundant. We have built into FRESCO several optional features that take advantage of the properties of possible scoring functions with the aim of lowering the time and storage requirements for the algorithm. These can be separated into two categories: optimal (the resulting alignment is still optimal) and heuristic (without optimality guarantees).

Optimal

- ○ *Pareto Efficiency.* Most relevant scoring schemes will score a specific diagonal's rectangle lower if its length or width are larger than another rectangle with the same diagonal (and if the other parameter is the same). Given this likely property, we have implemented an optional feature in FRESCO whereby, for each set {B, C, D}, we will have eliminated any points A (*recstarts*) where we have another closer A with a better overall score. This defines a pareto-efficient set[19]. While it is difficult to predict the exact size of this reduction, empirically, we observed that about order *logn recstarts* of the originally available $O(n)$ are retained, allowing us to reduce the running time and space requirements by almost a factor of n. This holds for both unrelated and highly similar sequences.

- ○ *SMAWK.* Given that the scoring function has the same concavity with respect to the rectangle area throughout (i.e. the function is always concave, or always convex), we can further speed up the alignment using the SMAWK[20] algorithm. In the recursion, we can reduce the number of rectangles we look at if we change the order of the iterations: first we consider pairs of *diagbegin* and *diagend* points {B, C}, and then compute the total scores at all relevant *recends* (Ds) and *recbegins* (As). When the computation is done in this manner, we can view this as the search for all of the column minima of a matrix $D[NxN]$, where each row corresponds to a particular *recbegin* point, each column corresponds to a *recend* point, and the cell $D[i, j]$ is the score of the path that enters the given diagonal through *recbegin* point i and exists it through recend point j. This matrix has been previously used in literature, and is commonly known as the DIST matrix[21]. If the scoring function is either concave or convex, the DIST matrix is totally monotone, and all of its column minima can be found in time linear in the number of columns and rows using the SMAWK algorithm. This optimization decreases the computation time for each possible *diagend* to $O(n^2)$, speeding up the overall alignment by $O(n)$.

Because the user may be interested in exploring non-uniform scoring schemes we have made both SMAWK and Pareto-efficency optional features in FRESCO, which can be turned on or off using compile-time options. However, with both the Pareto-efficency and the SMAWK speedups, the overall running time, originally $O(n^5)$, is observed to grow as $n^3 logn$ when both speed-ups are enabled. The observed running times are summarized in Figure 3.

Heuristic (Non-Optimal)

We also introduce two speed ups that, while not guaranteeing an optimal overall score, have been observed to work well in practice.

○ *Maximum diagonal length.* Since one key parameter that limits the running time of our algorithm is having to compute diagonals of all possible lengths, we have added an optional limit on the length of the diagonal, forcing each long diagonal to be scored as several shorter ones. For many scoring schemes this does not greatly affect the final alignment, while the running time is reduced by O(n). This improvement was also employed in the DIALIGN program[5].

○ *Banded Alignment.* We have also added an option to FRESCO which forces the rectangle scoring scheme to act only within a band in the matrix M around an already-computed alignment. Because most genome sequence alignment tools are going to agree overall on strong areas of similarity, banded alignment heuristics have commonly been used to improve on an existing alignment. Since FRESCO allows the testing of abilities of various scoring schemes, this improvment technique may be of particular interest when used with FRESCO. We have performed empirical tests by running FRESCO within a band around the optimal alignment to investigate the running time, and empirically observed a running time linear in n.

Figure 3 displays the running time of FRESCO using various optimization techniques for sequences of length 100 to 1000 nucleotides.

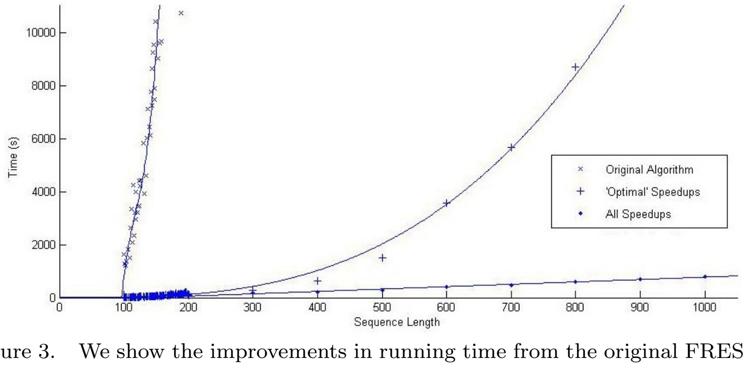

Figure 3. We show the improvements in running time from the original FRESCO algorithm, indicated by (x) and modeled by a polynomial, to the running time of FRESCO with the Pareto and Ranges (SMAWK) utilities on, indicated by (+) and modeled by a polynomial, and finally applying all speedups described in the text (including band size of 20 bp, maximum diagonal length of 30 bp), and resulting in linear running time (•).

4. Results

4.1. *Functions allowed*

The main power of the FRESCO tool is its ability to create alignments dictated by any rectangle scoring scheme. This will allow researchers to test schemes based on any motivations, such as evolution-based or statistical models. Since the creation of a new algorithm is not required for each of these schemes, we now have the ability to quickly compare the performance of complex scoring schemes. We have investigated traditional scoring schemes and aligners Clustal-W and LAGAN, against two novel scoring functions based on a parameter-less global E-value, described below.

4.2. *Example function & performance*

Given a diagonal and its bounding rectangle, the global E-value is the expected number of diagonals within this rectangle with equal or higher score. We calculate this by computing, for every possible diagonal in the rectangle, the probability that it has a score higher than the one in our diagonal, and summing these indicator variables. Note that our global E-value is different from the Karlin-Altschul statistic. To compute the global E-value we first define a random variable corresponding to the score of matching two random (non-homologous) letters. The expected value of this random variable (referred to as R below) is determined by computing the frequency of all nucleotides in the input strings, and for all 16 possible pairings multiplying the score of a match by the product of the frequencies. The variance (V) of the variable is the sum of the squared differences from the expectation. We model a diagonal of length d as a set of repeated, independent samplings of this random variable. The probability $g(s, d)$ that the sum of these d trials has a $score > s$ can be approximated as the integral of the tail of a Gaussian, with mean R_d and variance V_d:

$$g(s,d) = \int_s^\infty \frac{1}{V_d\sqrt{2\pi}} e^{-\frac{(x-R_d)^2}{2V_d^2}} dx \qquad (1)$$

Note that $g(s, d)$ is also the expected value of the variable which indicates whether or not a particular diagonal of length d has a score higher than s. The expected number of diagonals within a rectangle with a given score or higher is equal (by linearity of expectation) to the sum of expectations of indicator variables corresponding to individual diagonals, yelding the formula

$$E = \sum_{i=1}^{min(m,n)} g(s,i) + |m-n| g(s,d) \tag{2}$$

The E-values for the individual rectangles can be combined in a variety of ways, leading to various alignment qualities. Below we will demonstrate results for two ways of combining the functions:

$$E - Value\ I : \sum_{i=1}^{N} \log(\frac{2}{E_i} + \epsilon) = \prod_{i=1}^{N} (\frac{2}{E_i} + \epsilon) = \epsilon^N + 2\prod_{i=1}^{N} E_i^{-1} \tag{3}$$

$$E - Value\ II : \sum_{i=1}^{N} \log(\log(\frac{1}{E_i} + \epsilon)) = \prod_{i=1}^{N} \log(\frac{1}{E_i} + \epsilon) \tag{4}$$

Where ϵ is used to avoid asymptotic behaviour. We used $\epsilon = 0.1$.

Performance
The evaluation of DNA alignment accuracy is a difficult problem, without a clear solution. In this paper we have chosen to simulate the evolution of DNA sequence, and compare the alignments generated by each program with the "gold standard" produced by the program that evolved the sequences. We used ROSE[22] to generate sequences of length 100-200 nucleotides from a wide range of evolutionary distances and ratios of insertions/deletions (indels) to substitution, using a Jukes-Cantor[23] model and equal nucleotide frequency probability (See Table 1). The evolved sequences were aligned with FRESCO using several E-value based scoring functions (described above), as well as with the Clustal-W and LAGAN aligners, with default parameters. The accuracy of each alignment was evaluated both on a per-nucleotide basis with the program described in Pollard et al, 2004[24], as well as based on how closely the number of indels in the generated alignments matched the number of indels in correct alignments. The results are summarized in Figure 4. While the per nucleotide accuracy of the LAGAN aligner is best, the E-value II function we have defined manages to top the ClustalW aligner in accuracy and estimate the indel ratio better than both LAGAN and ClustalW in most tests, without using any biological or evolutionary knowledge. It is important to note that the improvement of the global E-value over ClustalW becomes more pronounced with greater evolutionary distance.

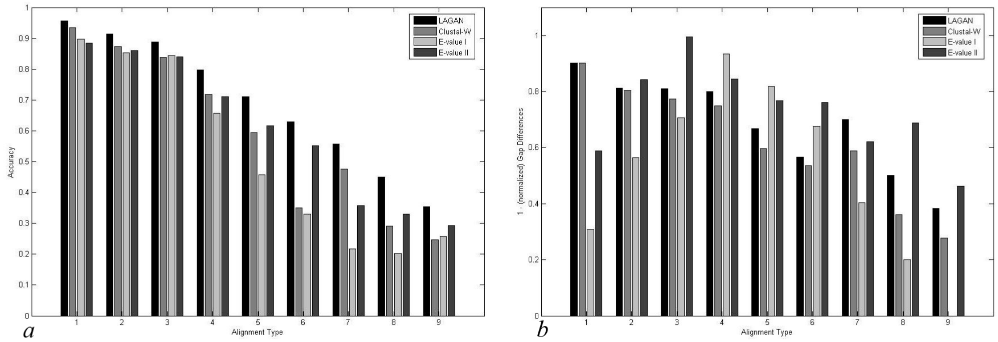

Figure 4. We evaluated the E-value scoring functions on a set of ROSE-generated alignments based on the accuracy (a) and 1 - the gap frequency difference (b) between the observed and evolved alignment, and compared with results from the LAGAN and ClustalW aligners. For alignment types 1-9, evolutionary distance 0.25, 0.50 & 0.75 subs/site, from left to right, we tried three indel per substitution ratios 0.06, 0.09, and 0.12 each. While the accuracy of the E-value II scheme fell between LAGAN and ClustalW, the indel ratio is in general better (than both aligners) with the E-value II function. The details of the analysis are included in the appendices.

Table 1. Summary of evolutionary parameters used to generate test data. Sequences were evolved using three different evolutionary distances (substitutions per site), each with three different indel to substitution ratios.

Type	1	2	3	4	5	6	7	8	9
Subs Per Site	0.25	0.25	0.50	0.50	0.50	0.75	0.75	0.75	0.75
Indel/Subs	0.06	0.09	0.12	0.06	0.09	0.12	0.06	0.09	0.12

5. Discussion

In this paper we generalize several schemes that have been previously used to align genomes into a single, more general class of *rectangle scoring schemes*. We have developed FRESCO, a tool that can find the optimal alignment for two sequences under any scoring scheme from this large class. While the tool we have built only allows for alignment of short sequences, and is not usable for whole genomes (it is many-fold slower than anchored aligners such as LAGAN and AVID), we believe that it should enable bioinformaticians to explore a large set of schemas, and once they find one that fits their needs, it will be possible to write a faster, specialized program for that scoring scheme. In this paper we provide an example of a rectange scoring function that incorporates no biological knowledge but performs on par with popular alignment algorithms, and we believe that even more accurate schemas can be found using the FRESCO tool.

6. Implementation and Supplementary Information

FRESCO was developed solely in C. The scoring scheme is supplied as a '.c' file, in which we allow a definition of the scoring function (in C code) as well as any pre-computations and global variables necessary for the scheme. A script to test the FRESCO results against other aligners or the *true* alignment is written to aid in comparing scoring schemes, implemented in a combination of perl and shell scripts. All are available at http://compbio.cs.toronto.edu/fresco. At this same address one can find an appendix and the generated datasets used in the results section.

References

1. S. F. Altschul, T. L. Madden, A. A. Schffer, J. Zhang, Z. Zhang, W. Miller and D. J. Lipman, *Nucleic Acids Res* **25**, 3389 (1997).
2. J. D. Thompson, D. G. Higgins and T. J. Gibson, *Nucleic Acids Res* **22**, 4673 (1994).
3. C. M. Bergman and M. Kreitman, *Genome Res* **11**, 1335 (2001).
4. B. Morgenstern, A. Dress and T. Werner, *Proc Natl Acad Sci* **93**, 12098 (1996).
5. B. Morgenstern, *Bioinformatics* **16**, 948 (2000).
6. C. Notredame, D. G. Higgins and J. Heringa, *J Mol Biol* **302**, 205 (2000).
7. C. Do, M. Brudno and S. Batzoglou, *Nineteenth National Conference on Artificial Intelligence AAAI* (2004).
8. R. Hamming, *Bell System Technical Journal* **26**, 147 (1950).
9. V. I. Levenshtein, *Soviet Physics Doklady* **10**, p. 707 (1966).
10. S. B. Needleman and C. D. Wunsch, *J Mol Biol* **48**, 443 (1970).
11. O. Gotoh, *J Mol Biol* **162**, 705 (1982).
12. S. Schwartz, Z. Zhang, K. A. Frazer, A. Smit, C. Riemer, J. Bouck, R. Gibbs, R. Hardison and W. Miller, *Genome Res.* **10**, 577 (2000).
13. N. Bray, I. Dubchak and L. Pachter, *Genome Res.* **13**, 97 (2003).
14. M. Brudno, C. B. Do, G. M. Cooper, M. F. Kim, E. Davydov, E. D. Green, A. Sidow and S. Batzoglou, *Genome Res* **13**, 721 (2003).
15. S. Karlin and S. F. Altschul, *Proc Natl Acad Sci* **87**, 2264 (1990).
16. M. A. Roytberg, A. Y. Ogurtsov, S. A. Shabalina and A. S. Kondrashov, *Bioinformatics* **18**, 1673 (2002).
17. P. D. Keightley and T. Johnson, *Genome Res.* **14**, 442 (2004).
18. C. Do, S. Gross and S. Batzoglou, *Tenth Annual International Conference on Computational Molecular Biology (RECOMB)* (2006).
19. M. J. Osborne and A. Rubinstein, *A Course in Game Theory* 1994.
20. A. Aggarwal, M. M. Klawe, S. Moran, P. Shor and R. Wilber, *Algorithmica* **2**, 195 (1987).
21. J. P. Schmidt, *SIAM J. Comput.* **27**, 972 (1998).
22. J. Stoye, D. Evers and F. Meyer, *Bioinformatics* **14**, 157 (1998).
23. T. H. Jukes and C. R. Cantor, *Evolution of Protein Molecules* 1969.
24. D. A. Pollard, C. M. Bergman, J. Stoye, S. E. Celniker and M. B. Eisen, *BMC Bioinformatics* **5** (2004).