Eliminating Dynamic Programming Bias in Multiple Sequence Alignment Algorithms

Luke Sheneman[1*], James A. Foster [1]

[1]Department of Biological Sciences, University of Idaho, Moscow, Idaho, USA

[*]Corresponding author

Email addresses:
LS: shen0614@uidaho.edu
JAF: foster@uidaho.edu

Abstract

Background

Most useful algorithms that align multiple molecular sequences rely on dynamic programming. In practice, dynamic programming algorithms commonly produce arithmetic ties. Conventional methods for resolving these ties produce biased results through arbitrary and deterministic strategies. These tie resolution strategies influence the final sequence alignment, especially in the context of progressive multiple sequence alignment. With progressive alignment, the search through alignment space is constrained by tie resolutions since any alignment step is dependent on how ties were resolved in previous steps.

Results

We extend conventional dynamic programming alignment algorithms to be both stochastic and unbiased when resolving arithmetic ties. We demonstrate that progressive alignment with unbiased tie resolution results in a set of possible alignments and a corresponding distribution of alignment scores. We challenge the standard paradigm of progressive multiple sequence alignment by presenting an example wherein the optimal progressive sequence alignment is achieved only by combining sub-optimal subalignments.

Conclusions

Arithmetic ties during dynamic programming are very common, and conventional algorithms are biased in how they resolve ties. We solve this problem by extending conventional dynamic programming alignment algorithms. Lastly, we propose a novel sampling approach to explore alignment distributions to find better alignments.

Background

Dynamic programming (DP) is a common, mathematically rigorous technique for solving a wide variety of optimization problems [1]. In DP, problems are broken down into independent sub-problems, and these sub-problems are individually and systematically solved and eventually combined to assemble the overall solution. DP is inherently Markovian in the sense that every incremental step of DP is dependent only on a small set of immediately previous steps (*i.e.* adjacent sub-problems). Dynamic programming is commonly used to optimally align molecular sequences with respect to specific substitution rates and gap penalties [2, 3, 4].

In pairwise sequence alignment, we use DP by constructing a two-dimensional matrix in which the characters from one sequence (or alignment profile) is oriented along one axis of the matrix with one character per matrix column or row. Likewise, the other sequence is oriented along the other axis of the matrix. DP algorithms then fill the matrix with values, progressing from one corner of the matrix to the other corner in a systematic fashion as depicted in Figure 1. The values for each cell in the matrix are determined by the three adjacent cells which were solved one step previously. At every cell, the *cost* of substituting residues/nucleotides or adding/extending a gap is assessed. One determines these costs by the choice of substitution matrix and affine

gap penalties. Every cell is assigned a value in order to maximize the local benefit (or minimize the cost) of moving from one of the three adjacent cells to the current cell, and is computed using the following recurrence relationship (1):

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_i), \\ F(i-1, j) - d, \\ F(i, j-1) - d \end{cases} \quad (1)$$

In (1), *F(i,j)* is the current cell in the DP matrix *F* at position *i*, *j*.. The cost function *s(x,y)* is a lookup into a substitution table describing the cost of substituting symbol x_i with symbol y_i while *d* represents the cost incurred for adding a gap. Substitution matrices for nucleotides typically reflect differences in transversion/transition ratios, while substitutions for proteins are commonly based on empirically-derived, probabilistic models such as PAM [5] and BLOSUM [6]. Affine gap costs are often used, which assigns a fixed gap penalty for starting a new gapped region, and a smaller penalty for extending an existing gapped region [7].

When the entire DP matrix is populated, the path which led from the starting point to the end point is uncovered through an algorithmic process known as backtracking. Backtracking identifies the valid steps at each cell which could have led to the cell being filled with the locally optimal value that it ultimately received. When using affine gap penalties on a two-dimensional matrix, correctly identifying the correct path at each cell actually involves a lookup of nine cells at every backtracking step to identify a truly valid backtracking step.

Identifying an optimal sequence alignment for a set of input sequences is known to be NP-Hard [8, 9]. Dynamic programming sequence alignment techniques compute the provably optimal alignment for *n* sequences of mean length *k* in time $O(k^n)$. In other words, optimal simultaneous multiple sequence alignment via dynamic programming quickly becomes computationally intractable as the number of input sequences increase, even when the dynamic programming matrix is evaluated only within pre-computed bounds [10]. For this reason, researchers rely on heuristic alignment algorithms to produce good alignments in a practical amount of time.

Arithmetic Ties In Dynamic Programming

Dynamic programming *ties* can and do occur. Ties happen when the value of any one cell is computable with identical values from more than one adjacent cell. When this happens, there exists more than one valid route through the backtracking path. Due to the presence of these ties, the route through the dynamic programming matrix is not a single, branchless, linear path, but a directed acyclic graph (DAG). In the presence of ties, backtracking changes from a simple linear traversal algorithm to a more complicated graph traversal algorithm.

The backtracking algorithm can break arithmetic ties arbitrarily and deterministically, or break ties in a wholly stochastic, unbiased way. In the interest of speed and

simplicity (and perhaps naiveté), most progressive multiple sequence alignment algorithms break these ties in an arbitrary fashion, severely biasing the resulting alignment. Other algorithms break the tie stochastically by assigning each valid path equal probability. For example, by giving each of two valid paths a 50% chance of consideration. However, this method of tie resolution is also flawed.

It is obvious that when ties are encountered along the backtracking path, the path ceases to be strictly linear and should really be considered a directed acyclic graph (DAG). The backtracking DAG represents a set of possible, equally optimal paths through the DP matrix. One enumerates the set of optimal pairwise alignments by exhaustively traversing all paths through the DAG. However, in most cases, traversing all possible paths through the backtracking DAG is prohibitively expensive.

Progressive Multiple Sequence Alignment

Feng and Doolittle proposed the progressive multiple sequence alignment (PMSA) method [11] as a practical alternative to the intractable simultaneous alignment of n input sequences via n-dimensional dynamic programming. PMSA represents a pragmatic trade-off between alignment accuracy and speed. While the final alignment is no longer provably optimal, it is computed quickly. Instead of using dynamic programming on all n sequences simultaneously, PMSA systematically applies *pairwise* dynamic programming to optimally align pairs of sequences and subalignments.

This method reduces the amount of computation from $O(k^n)$ in the case of optimal simultaneous alignment to $O(nk^2)$ in the case of progressive alignment. In short, PMSA represents a dramatic asymptotic runtime improvement with respect to both the average length and number of input sequences.

PMSA uses estimated evolutionary distances between all sequence pairs to construct a *guide tree.* The guide tree approximates the phylogeny of the input sequences based on the interrelatedness of the sequences. A guide tree is typically constructed using a distance-based tree inference algorithm such as Neighbor-Joining [12, 13] or Relaxed Neighbor-Joining [14]. Guide trees dictate the order of the pairwise DP alignments. By aligning similar sequences prior to distant sequences, one limits the introduction of excessive and immutable gaps early in the alignment process. As sequences are added to a progressive alignment, previously introduced gaps cannot be retroactively altered. This phenomenon in PMSA is known as “once-a-gap, always-a-gap” [11].

Arithmetic ties encountered during dynamic programming have a particularly pronounced effect in PMSA algorithms, as we discuss later.

Results

The Frequency of Ties

Ties occur frequently in dynamic programming. The frequency of ties is determined by several factors, including the content and relatedness of the sequences and alignment parameters such as choice of substitution matrix and affine gap costs. By simulating and aligning pairs of nucleotide sequences under specific evolutionary models, we demonstrate patterns of tie frequency as a function of sequence similarity.

We use distinct simulated sequence pairs stratified into different levels of sequence similarity, from low to high. The sequences are generated by simulation under the F84 model [15] of sequence evolution using the ROSE program [16]. By using compatible alignment parameters, we pairwise align the sequences with the Needleman-Wunsch algorithm and count the frequency of dynamic programming ties across different strata of sequence relatedness.

We hypothesize that DP ties become more frequent as sequence similarity decreases. The results summarized in Figure 2 provide support for this hypothesis. As sequence divergence grows, the total number of ties counted along any backtracking DAG increase.

We found that when aligning very similar sequences, the path through the backtracking DAG has fewer equally-optimal branches. By contrast, highly-divergent sequences include many ties along the DAG. This may indicate that DP matrices for highly-divergent sequences simply have less inherent "signal", and therefore DP ties occur more frequently due to chance. By contrast, closely related sequences may contain a stronger signal (*i.e.* less noise) where chance ties are less likely.

The most common type of tie involves two-way ties with the diagonal direction. Several examples of such ties are shown in Figure 1. Ties between horizontal and vertical directions are very rare, as are three-way ties between all adjacent nodes.

Figure 3 indicates that the number of DP ties through the entire matrix remains fairly constant, independent of sequence relatedness. Ultimately, we observed the size and branching of the DAG shrinks as sequence relatedness increases, despite a fairly constant overall number of ties throughout the matrix.

Removing bias from dynamic programming

Most existing dynamic programming implementations for molecular sequence alignment break ties in an arbitrary and deterministic way that introduces bias. Superficially, this appears justifiable since there seems to be no reason to prefer one equally optimal pairwise alignment over another. However, in the context of progressive multiple sequence alignment, where the particular output of one stage becomes the input to the next, choosing between equally optimal pairwise alignments becomes important.

In the case of pairwise DP alignment, ties involving the diagonal are sometimes resolved by simply choosing the diagonal path. This decision results in no immediate gap insertions. At first glance, this technique appears to reduce the total number of gaps in the alignment since diagonal moves represent substitutions and do not add gaps to an alignment. In reality, this arbitrary tie-breaking technique simply delays gap insertions until later backtracking stages, systematically biasing gap placement along the aligned sequences. Finally, there is no mathematical or biological justification to arbitrarily prefer diagonal paths over other, equally optimal paths.

Another common approach to breaking DP ties is to essentially "flip a coin" wherever a tie is encountered during backtracking. While neither arbitrary nor deterministic, this stochastic approach remains biased because there are often an unequal number of paths to get to any particular cell along the backtracking DAG. Flipping a coin during backtracking, disregarding the number of paths leading to the current cell, results in choosing some paths with higher likelihood than other equally valid paths. Since there is no objective justification for choosing one equally valid backtracking path over another, this biased tie resolution strategy also artificially constrains the alignment.

We propose an extension to traditional dynamic programming algorithms that allows for stochastic, unbiased tie resolution during backtracking. Our extended DP algorithm impacts both the matrix fill and backtracking steps. The algorithm stores not only the locally optimal value at each cell, but also a list of the valid directions that lead to that cell. We also store the total number of paths that lead to that cell. We compute the total number of paths for any particular cell as the sum of the number of paths from the adjacent cells in the valid direction list. For example, if a cell's value can be computed from two adjacent cells, then there exists a two-way tie. The number of paths leading to the current cell is the sum of the number of paths leading from the two adjacent, tied cells.

During backtracking, when a tie is encountered, we stochastically resolve the tie by choosing a valid random direction *with weights proportional to the number of paths from each direction*. By proportionally weighting our backtracking directions, all complete and valid paths to the current cell are considered with equal likelihood. An example of this extended dynamic programming algorithm is shown below in Figure 4.

The extended dynamic programming algorithm for aligning sequences *X* and *Y* using substitution matrix *s* and gap penalty *d:*

The Modified Matrix Fill Algorithm (for global alignment):

```
FOR EACH row i AND column j IN DP matrix F DO:
      F_{i,j}.val    ← max(F_{i-1,j-1} + s(X_i,Y_j), F_{i,j-1} - d, F_{i-1,j} - d)
      Fi,j.dirs  ← find_directions()  // including tied dirs
      FOR EACH direction r in F_{i,j}.dirs DO:
            F_{i,j}.npaths ← F_{i,j}.npaths + F_r.npaths
      DONE
DONE
```

The *find_directions()* function determines which cells adjacent to the current cell could produce the optimal value recorded in the cell. It constructs and records a set of equally optimal directions (*i.e.* $F_{i,j}.dirs$) leading from adjacent tied cells to the current cell. The algorithm then uses this set of directions to sum up the number of paths leading to the current cell.

The Modified Backtracking Algorithm (for global alignment):

```
// start at lower-right corner
i=length(X)
j=length(Y)
WHILE i>0 AND j>0 DO:

      // vertical adjacent cell
      IF F_{i-1,j} leads to F_{i,j} AND results in F_{i,j}.val
        wt.up    ←  F_{i-1,j}.npaths   / F_{i,j}.npaths

      // horizontal adjacent cell
      IF F_{i,j-1} leads to F_{i,j-1} AND results in F_{i,j}.val
        wt.left  ←  F_{i,j-1}.npaths   / F_{i,j}.npaths

      // diagonal adjacent cell
      IF F_{i-1,j-1} leads to F_{i,j} AND results in F_{i,j}.val
        wt.diag  ←  F_{i-1, j-1}.npaths / F_{i,j}.npaths

      newdir   ←  rand_weight(wt.up, wt.left, wt.diag)

      go(newdir)
DONE
```

Here, the *length()* function computes the length of the supplied sequences, and this is used to start the backtrack at the lower-right corner of the DP matrix. Every direction has a weight that is computed by determining the proportion of total paths leading to the current cell from adjacent paths. The *rand_weight()* function stochastically breaks ties such that each direction is chosen with a likelihood proportional to its associated weight. In this way, we give every equally-optimal path through the DAG equal consideration, thus eliminating bias in tie resolution during backtracking. The *go()* function decrements *i* and/or *j* as necessary to backtrack up the DAG and matrix.

One guide tree, many alignments

The current dogma with respect to PMSA implies that there exists a one-to-one relationship between guide trees and alignments. That is, a set of input sequences, an evolutionary model (*e.g.* alignment parameters) and a guide tree entirely determine the single alignment result. However, the presence of arithmetic ties during dynamic programming in progressive alignment contradicts this dogma. The goal of a good PMSA algorithm is to search for and find the best possible multiple alignment achievable through the progressive sequence alignment method (*i.e. the globally optimal progressive alignment*).

During pairwise alignment, DP ties result in a set of equally optimal pairwise alignments. This has a *combinatorial effect* during progressive MSA, as one set of possible pairwise alignments could be aligned with yet another set of possible pairwise alignments. The choice of which optimal alignments to align next during PMSA constrains subsequent alignment steps during the progressive alignment process. Unfortunately, since both sets contain equally-optimal pairwise alignments, there exists no information to inform the choice of which optimal alignments to align at each step. While every pairwise alignment step results in a set of one or more equally-optimal alignments, the stepwise choice of alignments/profiles impacts the ability to find the globally optimal progressive alignment. Ultimately, every tie-breaking decision constrains the search for the globally optimal MSA. Since the choice of alignments at each pairwise step is uninformed, it is rare that a PMSA algorithm achieves the globally optimal PMSA for any given set of inputs.

In PMSA, a particular guide tree does not necessarily correspond to a single, deterministic multiple alignment. A guide tree, together with input sequences and alignment parameters, results in a *set* of possible complete progressive multiple alignments. This set of complete progressive alignments is rarely equally good. In reality, a single guide tree produces an entire set of possible multiple alignments and a corresponding *distribution* of alignment scores.

Figure 5 shows the output of a simulation experiment wherein we fixed the parameters to the progressive alignment algorithm and sampled the alignments produced through unbiased tie resolution as described previously. We simulated 10 sequences using ROSE under the F84 model. ROSE introduced insertion/deletion (i.e. *indels*) of different sizes in the output sequences, and output the "true" alignment. We fixed the guide tree and other alignment parameters and sampled 1,000 alignments by resolving dynamic programming ties in different ways. We scored the alignments by measuring their distance from the true alignment output by ROSE.

We align the same set of sequences using the same inputs via Clustal W [17] to demonstrate how a deterministic progressive alignment algorithm performs relative to the distribution of possible alignment scores. We find that sometimes Clustal W performs relatively well, sometimes it performs poorly, but it always misses the complete picture (the full distribution of alignments) and will almost never arrive at the best possible progressive alignment for any particular set of inputs.

A progressive alignment and dynamic programming counterexample

As mentioned, progressive multiple sequence alignment suffers from a fundamental limitation known as "once-a-gap, always-a-gap". Sequences and alignment profiles are pairwise aligned in the order dictated by a guide tree using only the local information present in the sequences at hand. Most PMSA algorithms use dynamic programming to achieve locally optimal pairwise alignments. Subsequent alignment steps cannot or do not use new information to retroactively inform and optimize the relative gap positions for sequences which have already been aligned. Resolved homology and relative gap placement are immutable between already-aligned sequences.

The key problem with the current PMSA paradigm is that optimal pairwise alignment at each step of the progressive alignment process constrains the search for the globally optimal alignment achievable through PMSA. PMSA algorithms are greedy because they always choose the locally optimal alignment at each step, even if choosing a suboptimal pairwise alignment at an earlier step may subsequently result in a globally optimal progressive alignment.

While this limitation of progressive multiple sequence alignment is well known, we present a concrete, visual example of the effect in Fig. 6. In this example, we progressively align three short DNA sequences. In once case, we align the sequences in the traditional fashion by consistently choosing from locally optimal pairwise alignments. In the other case, we vary from the optimal dynamic path slightly, resulting in a locally sub-optimal pairwise alignment but a globally superior overall progressive alignment.

In this specific example, we first pairwise align Seq1 and Seq2 and take the resulting alignment profile and align it with Seq3. There is one possible optimal pairwise alignment of Seq1 and Seq2 (i.e. there are no dynamic programming ties), and in this case the optimal backtracking path is precisely along the diagonal. If we align Seq1 and Seq2 optimally, and take the resulting alignment profile and optimally align it with Seq3, the result is globally suboptimal. This is demonstrated by aligning Seq1 and Seq2 together in a demonstrably suboptimal way by diverging slightly from the diagonal. When we take this locally suboptimal pairwise alignment and align it with Seq3, the result is globally superior to the first example.

It is infeasible to try all possible optimal and even slightly suboptimal paths in all combinations for large datasets. However, we believe this counterexample points to clear possibilities for future improvements for progressive alignment algorithms. Progressive algorithms could sample alignments, not only from optimal backtracking paths, but from paths in close proximity to the optimal path. It is possible to sample with trial and error from such alignments to incrementally approximate the globally optimum progressive alignment. Such sampling could continue as long as time permits, as determined by the user of the alignment algorithm.

Discussion

It is typically too computationally expensive to enumerate all possible combinations of DP backtracking paths in order to identify the specific combination of paths resulting in the globally optimal progressive multiple sequence alignment. Breaking ties in an unbiased, stochastic manner is the only practical way to fairly explore the space of path combinations. Repeatedly and fairly sampling from the complete set of DP paths and constructing a sample distribution of alignment scores allows one to characterize the search space and choose paths which are notably better than the average alignment.

Traditional progressive alignment algorithms are greedy in the sense that only optimal pairwise alignments are considered, while it is trivial to identify cases where *locally suboptimal pairwise alignments can be used to achieve superior overall alignments*.

We recommend that future progressive alignment algorithms sample from both optimal and slightly suboptimal backtracking paths in an attempt to identify globally superior alignments.

Conclusions

We demonstrated that dynamic programming ties are common in practice. In our simulation example, we showed that the frequency of DP ties is a function of the relatedness of the input sequences. In the case of simple pairwise alignments, dynamic programming ties result in a set of equally probable and equally optimal alignments. However, in the case of progressive multiple sequence alignment, the pairwise DP ties result in a combinatorial explosion of possible outputs. Since the way in which the sets of alignments are paired and aligned during the progressive alignment step impacts the final alignment output, these distinct outputs result in a distribution of alignment scores.

Without information guiding the selection of which alignments to use at each progressive step, the globally optimal complete progressive alignment is almost never realized. We demonstrated how conventional, deterministic progressive alignment algorithms fall far short of finding the globally optimal progressive alignment, leaving considerable room for improvement to existing progressive alignment and dynamic programming approaches.

We also demonstrated an extension to the traditional DP approach that removes bias during stochastic backtracking. This modification to the common Needleman-Wunsch or Smith-Waterman algorithms is simple to implement and relatively inexpensive in both space and time.

We demonstrated that deterministic programs with arbitrary tie-breaking mechanisms produce alignments that score significantly lower than the globally optimal progressive alignment.

Finally, we presented a concrete visual example of the core limitation of progressive multiple sequence alignment: the well-known “once-a-gap, always-a-gap” phenomenon.

Methods

Dynamic programming and progressive alignment program

We used portions of the EVALYN program [18] to perform dynamic programming and progressive multiple sequence alignment. Current versions of EVALYN (written by LS) implement stochastic, unbiased dynamic programming for global sequence

alignment. We retrofitted EVALYN to provide additional diagnostic output to count arithmetic ties and produce distributions of alignments through alternative tie breaking scenarios.

The C source code for EVALYN is freely available online at http://bioinformatics.hungry.com

Sequence simulation and related experiments

We simulated nucleotide sequences in our experiments for the sake of convenience and control. Simulation also provided us with the true alignment for our sequences, which was critical in objectively scoring our alignments. We used ROSE [16] to simulate evolution and generate nucleotide sequences under an explicit model. The source code for ROSE is freely available, and can be found online.

In all of our experiments, we strived to maximize model compatibility between how sequences were generated and how those sequences were aligned. ROSE handles insertion/deletion (*indel*) events in a probabilistic fashion, where the existence and length of the indel is determined by a user-specified probability distribution. While ROSE's distribution method does not map directly to the affine gap cost model used in alignment programs, we approximated affine costs by using a Poisson distribution with λ=6. This results in a long-tailed distribution of indel lengths with an average length of 6 nucleotides while the chance of a single indel remains relatively small. The probability of additional gaps decreases slowly, thereby mimicking low-cost affine gap extensions.

In an effort to increase the realism, we simulated our nucleotide sequences with an empirically-based GC-content based on a published *c. Elegans* study [19].

We objectively scored our alignments as shown in Figure 5 using the SP scores reported from the *baliscore* program. The source code for baliscore is distributed with BAliBASE, the Benchmark Alignment Database [20]. We used baliscore to measure the distance between alignments produced with EVALYN and Clustal W against the reference alignment (*i.e. true alignment*) produced by ROSE.

Authors' contributions

LS invented and implemented the algorithmic extensions to dynamic programming, carried out all of the related simulation studies, and developed the counter-example for progressive alignment.

JAF participated in experimental design and manuscript editing.

Acknowledgements

LS was partially funded by NIH P20 RR16454 from the INBRE Program of the National Center for Research Resources. JAF was partially funded by NIH NCRR 1P20 RR16448.

References

1. Bellman R: *Dynamic Programming*. Princeton, NJ: Princeton University Press; 1957.
2. Needleman SB, Wunsch CD: A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Bio* 1970, 48:443-453.
3. Smith T, Waterman M: Identification of common molecular sequences. *J Mol Bio* 1981, 147:195-197.
4. Gotoh O: An improved algorithm for matching biological sequences. *J Mol Bio* 1982, 162:705-708.
5. Dayhoff MO, Schwartz RM, Orcutt BC: A model of evolutionary change in proteins. In *Atlas of Protein Sequence and Structure*, Volume 5, Suppl. 3, Chapter 22. Edited by Dayhoff MO. Washington, DC: National Biomedical Research Foundation; 1978:345-352.
6. Henikoff S, Henikoff J: Performance evaluation of amino acid substitution matrices. *Proteins: Structure, Function, and Genetics* 1993, 17:49-61.
7. Gotoh O: Optimal sequence alignment allowing for long gaps. *Bull Math Biol* 1990, 52:359-373.
8. Wang L, Jiang T: On the complexity of multiple sequence alignment. *J Comput Bio* 1994, 1:337-348.
9. Just W: Computational complexity of multiple sequence alignment with sp-score. *J Comput Biol* 2001, 8(6):615-623.
10. Carrillo H, Lipman DJ: The multiple sequence alignment problem in biology. *SIAM J Appl Math* 1988, 48:1073-1083.
11. Feng DF, Doolittle RG: Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J Mol Evol* 1987, 25:351-360.
12. Saitou N, Nei M: The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Mol Biol Evol* 1987, 4:406-425.
13. Studier JA, Keppler KJ: A note on the neighbor-joining algorithm of Saitou and Nei. *Mol Biol Evol* 1988, 5:729-731.
14. Evans J, Sheneman L, Foster JA: Relaxed Neighbor-Joining: A Fast Distance-Based Phylogenetic Tree Construction Method, *J Mol Evol* 2006, 62:785-792
15. Felsenstein J, Churchill GA: A hidden Markov model approach to variation among sites in rate of evolution. *Mol Biol Evol* 1996, 13:93–104.
16. Stoye J, Evers D, Meyer F: ROSE: generating sequence families, *Bioinformatics* 1998, 14(2):157-163.
17. Thompson, JD, Higgins DG, Gibson TJ: CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence

weighting, positions-specific gap penalties and weight matrix choice. *Nucleic Acids Res* 1994, 22:4673-4680.
18. Sheneman L, Foster JA: Evolving Better Multiple Sequence Alignments, Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004), Seattle, WA.
19. Argawal P, States DJ: A Bayesian evolutionary distance for parametrically aligned sequences, *J Comp Biol* 1996, 3:1-17.
20. Thompson JD, Plewniak F, Poch O: BAliBASE: A benchmark alignments database for the evaluation of multiple sequence alignment programs, *Nuc Acids Res* 1999, 27(13):2682-2690.

Figures

Figure 1 - Pairwise sequence alignment with dynamic programming

This dynamic programming matrix depicts an implementation of the Needleman-Wunsch algorithm for global sequence alignment. The cells of the matrix are initially computed based on specific substitution rates and gap penalties. The algorithm then backtracks from the lower-right to the upper-left cells to reconstruct the alignment in reverse. In this example, there are three ties along the backtracking path. These backtracking ties lead to three equally optimal pairwise sequence alignments. We used a simple substitution/gap model where matches = 1, mismatches = 0, initial gaps = -1.0 and gap extensions = -0.1.

Figure 2 - Percentage of ties in optimal path

We measured tie frequency along the DAG during backtracking, and present the number of arithmetic ties encountered during backtracking. All four distinct kinds of ties are counted separately. The figure shows averages computed over multiple replicates. The 1 Kb nucleotide sequence pairs were simulated with ROSE under the F84 model. We computed percent sequence identity post-alignment using Clustal W. Here, our DP algorithm used the following scoring system: match=+1, mismatch=0, gap open = -1, gap extend = -0.1.

Figure 3 - Percentage of ties in entire dynamic programming matrix

The total number of ties of different types present in the entire DP matrix, as a function of sequence similarity. We present the tie count averaged over multiple replicates. The sequences were simulated as in Figure 2. The full DP matrix is 1000x1000, of which up to 10% of the cells contain a tie of some type. The correlation of tie frequency to sequence similarity is significantly different from the strong linear correlation found along the actual backtracking DAG as shown previously in Figure 2.

Figure 4 - A novel extension to dynamic programming

We extend dynamic programming to allow for unbiased, stochastic backtracking. For every cell, we track the number of paths leading to that cell (shown in upper-right squares in every cell along the backtracking graph). When randomly resolving ties during backtracking, our choice is weighted based on this path count, providing equal weight to every path through the matrix. In this example, the weights are depicted on the backtracking path lines.

Figure 5 - Four examples of distributions of 1,000 alignment scores each

For each distribution, we aligned 10 simulated nucleotide sequences and resolved ties in an unbiased, stochastic fashion. Within each sequence set, the alignments were derived from the same fixed guide tree and model parameters. All differences in alignments and scores within each example are entirely due to different output due to stochastic tie breaking. In each example, we show where the alignment generated by Clustal W scored with respect to the complete distribution.

Figure 6 – A progressive multiple sequence alignment counterexample

This example clearly depicts a key limitation in progressive multiple sequence alignment. We progressively align 3 short DNA sequence fragments. In one case, we produce the typical result by following the optimal DP backtracking path during the alignment of the first two sequences. In the other case, we vary slightly (in red) from the optimal backtracking path. By wondering off of the optimal backtracking path, we get a sub-optimal pairwise alignment of the first two sequences. Aligning the third sequence with the sub-optimal alignment of the first two sequences, we find a globally better final alignment. By using alignments dictated by locally optimal backtracking paths, we significantly constrain the search for the best possible overall sequence alignment. Future PMSA algorithms could also attempt sub-optimal backtracking paths near the optimal path for a more complete search.

Additional files

Additional file 1 – ties.rose.txt

Example input file for ROSE, the program used to generate sequences and *true* alignments based on a probabilistic model of sequence evolution.

This input file was used to perform the tie counting observations in the manuscript (Figures 2 and 3). The goal of using ROSE here is to create realistic nucleotide sequences with specific characteristics, such as average length and specific levels of sequence identity.

In this example input, we produce 10 sequences with an average length of 1kbp. We use empirically-derived values for GC content, and we attempt to model indel frequencies in a realistic way which is compatible with the affine gap cost model for sequence alignment.

Additional file 2 – dists.rose.txt

Example input file for ROSE, the program used to generate sequences and *true* alignments based on a probabilistic model of sequence evolution.

This input file is used to perform the "distribution of alignments" observations in the manuscript (Figure 5). The goal of using ROSE here is to create realistic nucleotide sequences with specific characteristics, such as a specific average sequence length and specific levels of sequence identity. Simulating sequences in this way provides many useful benefits.

ROSE outputs the "true" sequence alignment of the simulated sequences. We use the true alignment to score every alignment in the distribution (by measuring the distance from every alignment in the distribution to the true alignment). We do the same with the multiple alignment produced by Clustal W to get an objective measure of alignment quality.

In this example input, we produce 10 sequences with an average length of 500 bp. We use empirically-derived values for GC content, and we attempt to model indel frequencies in a realistic way which is compatible with the affine gap cost model for sequence alignment.

.

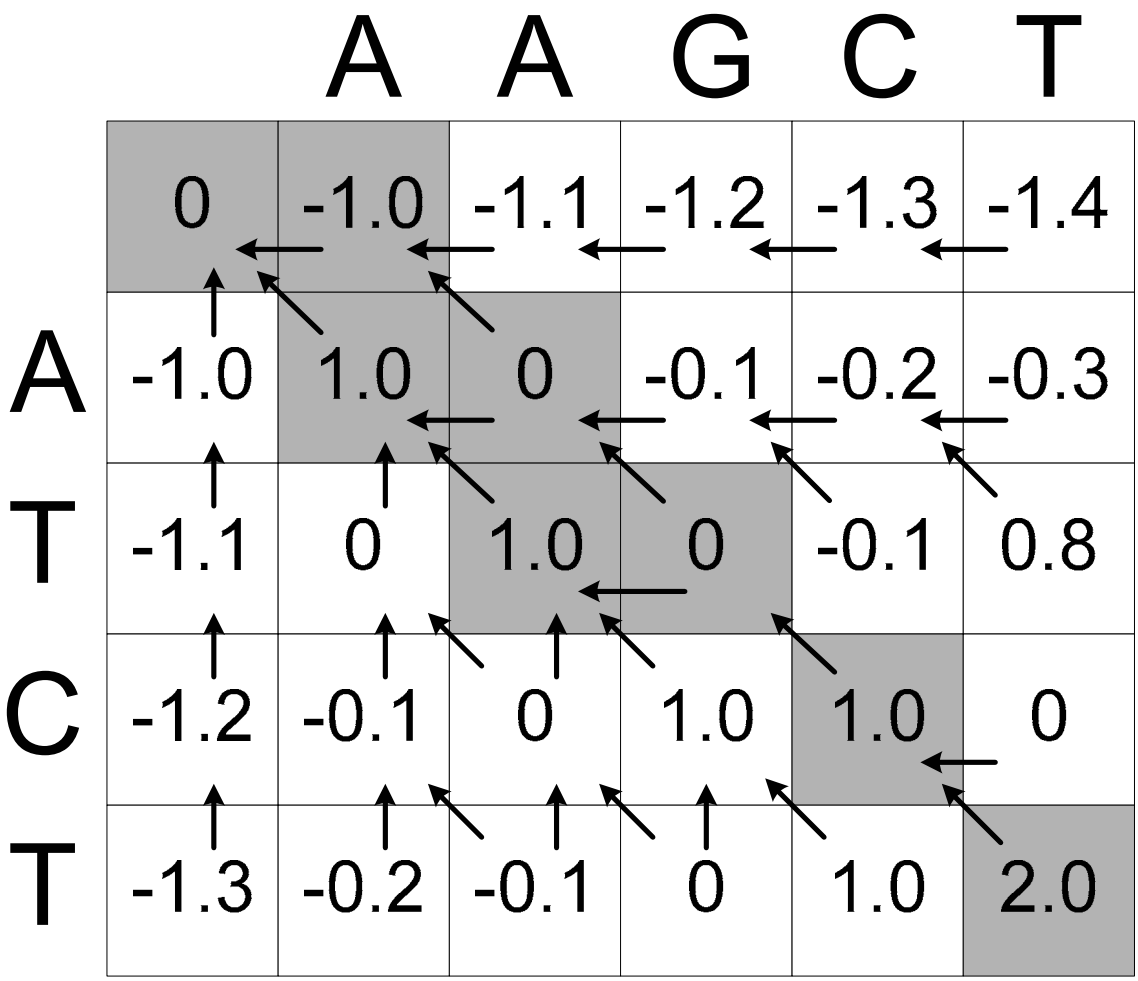

A A G C T
A
T
C
T
0 -1.0 -1.1 -1.2 -1.3 -1.4
-1.0 1.0 0 -0.1 -0.2 -0.3
-1.1 0 1.0 0 -0.1 0.8
-1.2 -0.1 0 1.0 1.0 0
-1.3 -0.2 -0.1 0 1.0 2.0

Figure 1

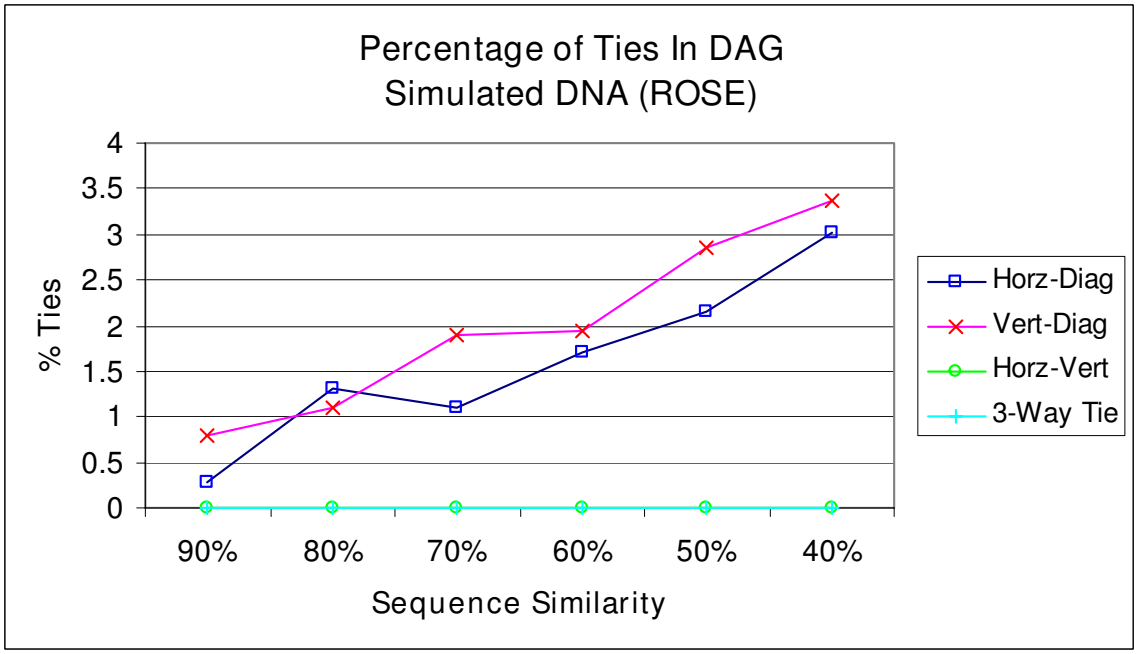

Percentage of Ties In DAG
Simulated DNA (ROSE)
% Ties
4
3.5
3
2.5
2
1.5
1
0.5
0
90%
80%
70%
60%
50%
40%
Sequence Similarity
Horz-Diag
Vert-Diag
Horz-Vert
3-Way Tie

Figure 2

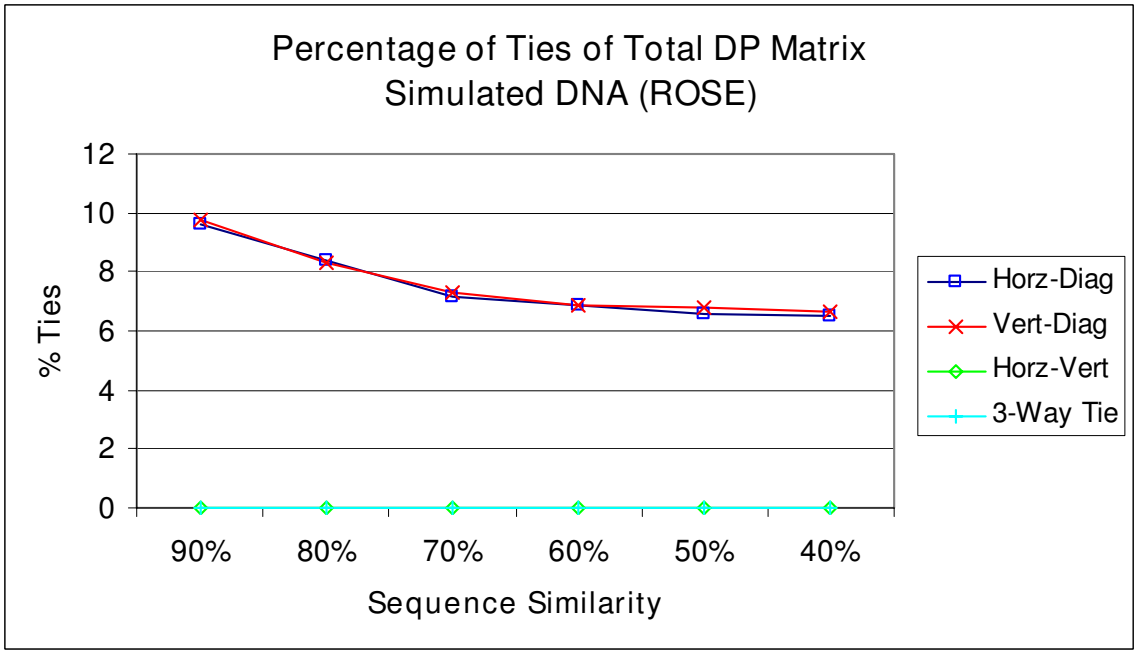

Percentage of Ties of Total DP Matrix
Simulated DNA (ROSE)
% Ties
12
10
8
6
4
2
0
90%
80%
70%
60%
50%
40%
Sequence Similarity
Horz-Diag
Vert-Diag
Horz-Vert
3-Way Tie

Figure 3

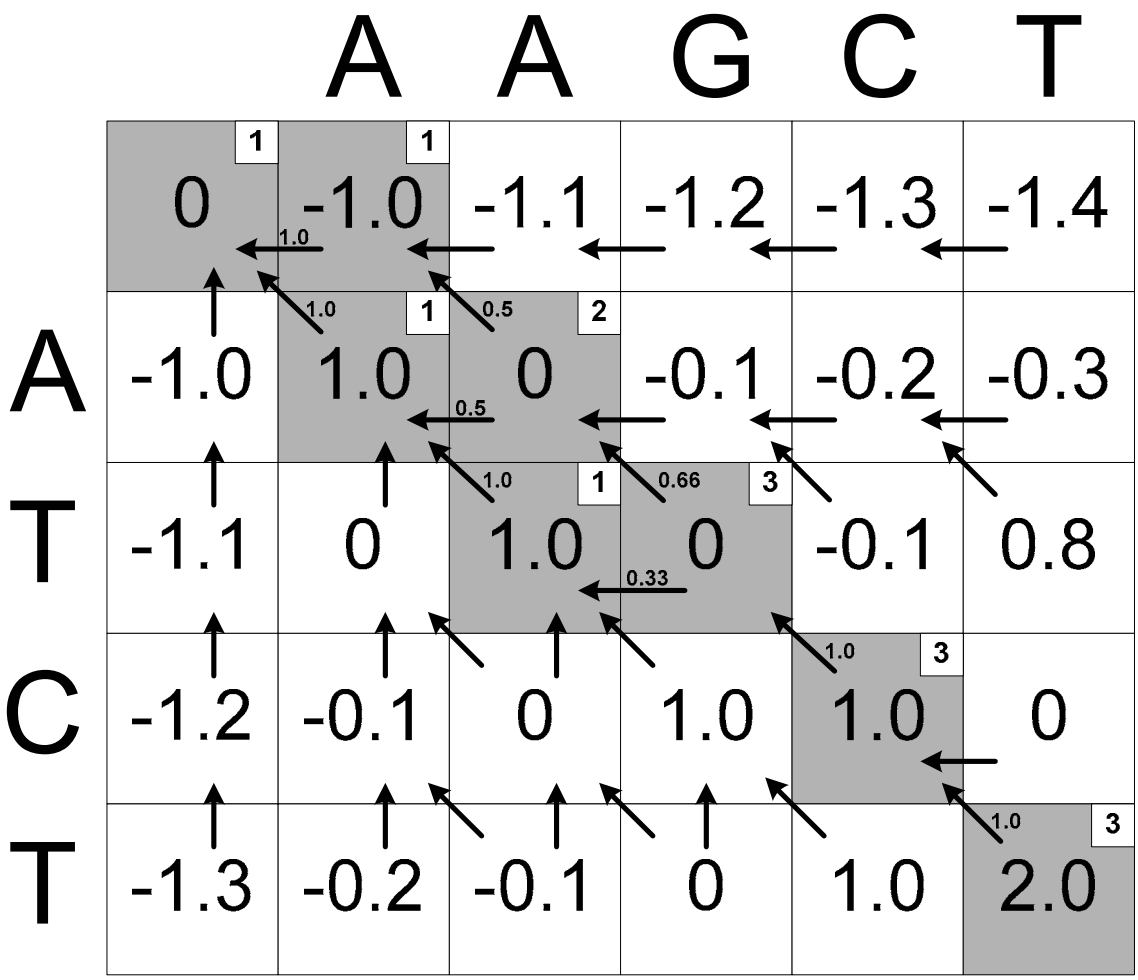

A A G C T
A T C T
1 1
0 -1.0 -1.1 -1.2 -1.3 -1.4
1.0
1.0 1 0.5 2
-1.0 1.0 0 -0.1 -0.2 -0.3
0.5
1.0 1 0.66 3
-1.1 0 1.0 0 -0.1 0.8
0.33
1.0 3
-1.2 -0.1 0 1.0 1.0 0
1.0 3
-1.3 -0.2 -0.1 0 1.0 2.0

Figure 4

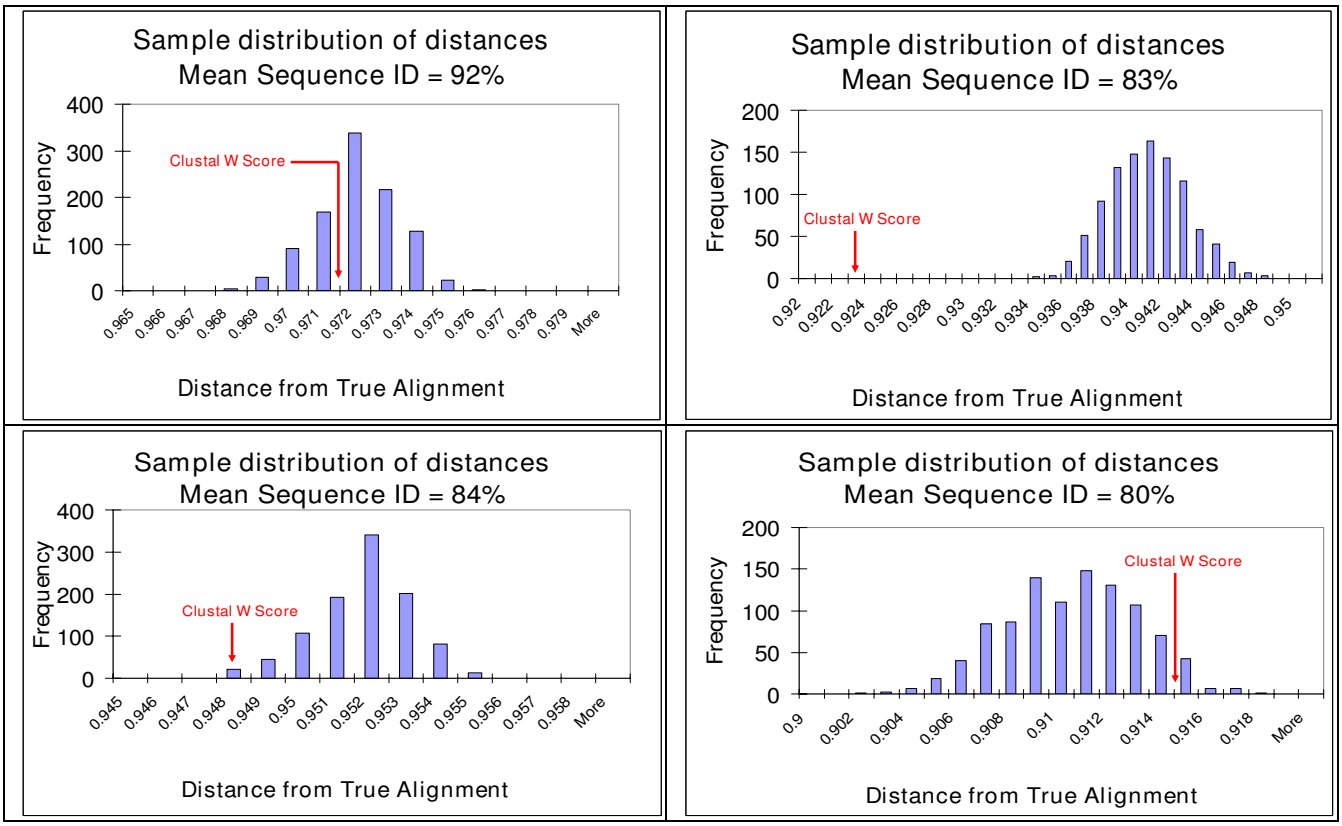
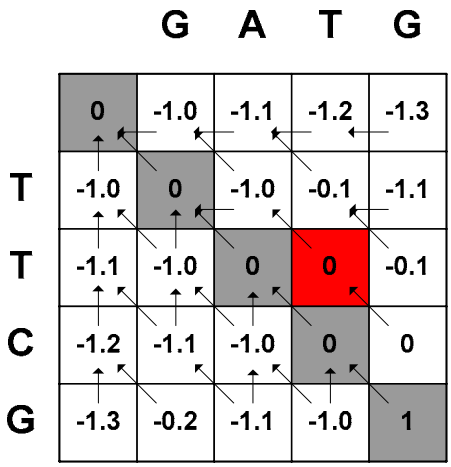

Sample distribution of distances
Mean Sequence ID = 92%
Frequency
Clustal W Score
Distance from True Alignment
Sample distribution of distances
Mean Sequence ID = 83%
Frequency
Clustal W Score
Distance from True Alignment
Sample distribution of distances
Mean Sequence ID = 84%
Frequency
Clustal W Score
Distance from True Alignment
Sample distribution of distances
Mean Sequence ID = 80%
Frequency
Clustal W Score
Distance from True Alignment

Figure 5



Optimal Path

Variation from Optimal Path

Match	= +1.0
Mismatch	= +0.0
Gap Open	= -1.0
Gap Extend	= -0.1

Sequences To Align:

Seq1: GATG
Seq2: TTCG
Seq3: ACCG

Guide Tree:
((Seq1,Seq2), Seq3)

(seq1,seq2) optimal alignment

GATG
TTCG

SP Score = 1.0

(seq1,seq2) sub-optimal alignment

GAT-G
TT-CG

SP Score = -1.0

PMSA with optimal backtrace

GATG-
TTCG-
-ACCG

SP Score = *-1.0*

PMSA with *suboptimal* backtrace

GAT-G
TT-CG
AC-CG

SP Score = *0.0*

Figure 6

Additional files provided with this submission:

Additional file 1: ties.rose.txt, 4K
http://www.biomedcentral.com/imedia/6247052931306617/supp1.txt
Additional file 2: dists.rose.txt, 4K
http://www.biomedcentral.com/imedia/1118424360130661/supp2.txt