

# Supplementary Material for ‘Enredo and Pecan: Genome-wide mammalian consistency based multiple alignment with paralogs’

Benedict Paten<sup>\*1†</sup>, Javier Herrero<sup>2†</sup>, Kathryn Beal<sup>2</sup>, Stephen Fitzgerald<sup>2</sup>, Ewan Birney<sup>\*2</sup>

<sup>1</sup> Center for Biomolecular Science and Engineering, University of California Santa Cruz, CA, USA <sup>3</sup> EMBL European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge, CB10 1SD, UK

Email: Benedict Paten<sup>\*</sup> - benedict@soe.ucsc.edu; Ewan Birney<sup>\*</sup> - birney@ebi.ac.uk;

<sup>\*</sup>Corresponding author

**Keywords** Large-scale genomics, Multiple-sequence alignment.

## Overview

This is the supplementary material for: Enredo and Pecan: Genome-wide mammalian consistency based multiple alignment with paralogs. You can find extended description of the ancestral repeat analysis in the document ‘ancestral\_repeats.pdf’. This document primarily contains methodological details of the Enredo and Pecan programs. We first describe the details of computing the non-overlapping GPA set for Enredo, then the process of guide tree inference for each segment-group in the Enredo segmentation, and finally we describe in some detail the Pecan alignment program used to align each segment-group. Pecan is publicly available for download from <http://www.ebi.ac.uk/~bjp/pecan/>. Enredo will shortly be available for public download.

### Enredo Genomic Point Anchor Generation

Each genomic point anchor (GPA) is made up of a set of homologous segments from the different input genomes (for this paper we have used the human, cow, dog, rat and mouse genomes as inputs). To build the set of non-overlapping GPAs we use a two stage process. In the first stage we scan and process pairwise alignments to find short segment-groups (short in terms of their length on the human genome). In the second stage we map the segments in each of these segment-groups to the input genomes to find extra homologous segment copies. We then remove segment-groups from the set whose members occur too frequently or overlap with segments in other segment-groups. For practical reasons the method is currently human centric, in common with other methodologies [Blanchette et al., 2004] [Miller et al., 2007], but still allows for paralogous alignment. We describe these two stages in detail.

To build the GPA set we superimpose BlastZ-net alignments [Schwartz et al., 2003] of the mouse, rat, dog and cow genomes to the human one, using the latter genome as a reference. We trim the alignments in order to get the longest region common to all the segments. We consider alignments containing segments from at least 4 out of these 5 genomes. When the resulting alignment is too small (shorter than 50bp in its projection on the human genome) and it contains segments from all 5 genomes, we try to get a longer alignment by removing one of the segments, the whole alignment being discarded if this doesn’t work. If, on the other hand, the alignment is too long (more than 250 bp in its projection on the human genome), we look for the best conserved region of this alignment. This is achieved by re-aligning the segments within the alignment using Pecan, and determining highly conserved regions using Gerp [Cooper et al., 2005]. We obtained 95605 GPAs using the trimming technique and another 436865 GPAs using the Gerp method, totalling 532470 GPAs. Approximately half of the GPAs have a representative segment from all 5 genomes.

We use Exonerate [Slater and Birney, 2005] to align the segments in the GPAs to all genomes. Ultimately we want GPAs containing a limited number of segments. Over-spread and over-represented GPAs create too much noise in the Enredo graph. Therefore, we remove any GPA which contains a segment either found in more than 5 different chromosomes of the same genome, more than 10 times in any single chromosome, or more than 20 times in any genome. This filtering step results in the removal of 2% of the original GPAs. The GPA set is required to be non-overlapping, i.e. no position in any input genome can occur in two distinct GPAs. To make the set of GPAs meet this requirement we use a greedy algorithm. Let  $s(A, B)$  be the number of times a segment from GPA A overlaps a segment from GPA B in all the genomes. The total overlap score for a GPA A is  $\sum_{B \neq A} s(A, B)$ . The GPA with the largest score is removed from the set and all the scores are updated accordingly. This process is repeated until no more overlapping GPAs are found, reducing the number of final GPAs by another 6%. The final anchor set contains nearly half a million GPAs.

### Phylogenetic tree prediction

Unlike previous methods for whole genome alignment, we include paralogs in our alignments. This has the potential to provide a great deal more information, but requires the reconstruction of phylogenetic-trees that disentangle paralog relationships. Taking each segment-group we perform phylogenetic tree inference before passing this tree to Pecan to act as a guide tree for its progressive alignment stage. This tree (with branch lengths) is also stored and is available for other down-stream analyses of the alignment segments. We have developed a script that takes as input a distance matrix for a set of sequences and a species tree which relates the sequences and which generates a phylogenetic tree using neighbour-joining (NJ) (Saitou, et al, 1987). Neighbour-joining predicts unrooted binary trees. Unfortunately, we observe that standard heuristics to predict the root are often unsatisfactory. We therefore implemented a species tree reconciliation algorithm [Page and Charleston, 1997] able to calculate for a rooted phylogenetic tree the minimum number of duplications and losses required for reconciliation with a given species tree. We then choose the root of our NJ tree as midway along the branch which requires the minimum number of duplications, and then, to break any ties, the minimum number of losses.

To generate the distance matrix for the NJ script we require an alignment between the sequences. But to get a reliable alignment using our program we need a good guide tree. To solve this chicken-and-egg problem we use an iterative approach. We start with a star tree and use Pecan to align the sequences. We then take this alignment and compute a distance matrix upon which we run the NJ-script to get a new tree

estimate. We iterate the process of alignment and tree inference until either the tree does not change between iterations, or a maximum number of iterations have elapsed (typically 3).

We use the Semphy program (Ninio, et al, 2007) for the NJ script, using the HKY (Hasegawa, et al, 1985) model with a transition to transversion ratio of 2.0 and a stationary GC content of 40% model. We use only un-gapped columns from an alignment if more than 1000 ungapped columns are available, otherwise we use all the available columns.

## **Pecan**

This section describes the Pecan multiple alignment program. The inputs to Pecan are a set of sequences and binary phylogenetic guide tree relating them. The Pecan algorithm has four main stages. In the first stage we compute a set of alignment constraints, we call this a ‘constraint map’. In the second stage we use this constraint map to compute the set of pairwise posterior match probabilities. In the third stage we modify the set of posterior match probabilities using the consistency transformed described in Equation 1 of the main paper. In the fourth and final stage we combine the transformed posterior match probabilities into a multiple alignment using a method of progressive alignment with the input guide tree.

In this methods section we first describe the computation of the initial constraint map. We then describe the computation of the pairwise posterior match probabilities using this constraint map. We omit details of third and fourth stages of the Pecan program, because they are equivalent to the final two stages of the related Probcons method [Do et al., 2005]. In practice there is an important scheme for recycling memory allocated for storing posterior match probabilities, which will be described elsewhere (Paten et al., submitted). After describing these core processes we describe the technique of transitive anchoring, used to reduce the amount of redundant pairwise posterior probability calculation. Finally we give an informal runtime analysis and describe the pair-HMM models available in Pecan and the training used to parameterise these models.

### *Building the constraint map*

We use the notation  $x_i \triangleleft y_j$  for a  $i < j$  constraint between sequences  $x$  and  $y$ . Similarly we use the notation  $x_i \trianglelefteq y_j$  for a  $i \leq j$  constraint between sequences  $x$  and  $y$ . This is consistent with the notation  $x_i \diamond y_j$  used in this paper to imply that position  $i$  in sequence  $x$  and position  $j$  in sequence  $y$  are aligned. We will also talk about alignment anchors. Alignment anchors are continuous un-gapped series of one or more aligned pairs (see green edges in Figure S1(a)), normally derived from local alignments.

For each pairwise alignment Pecan must first select a colinear and non-overlapping chain of anchor constraints around which to construct the alignment ‘band’. The alignment band being a relaxed set of constraints derived from the initial set of anchors (black lines around anchors in Figure S1(a)). During a multiple sequence alignment if free variation of the anchor chains is allowed (i.e. anchor chains are chosen without checking if they can form a single alignment with the other selected chains) then it is obvious that conflicts will occur (Figure S2). We call these conflicts ‘anchor inconsistencies’. In terms of sequence constraints, an anchor inconsistency occurs if a cycle of constraints occurs containing a constraint of the form  $x_i \triangleleft y_j$ . Avoiding these inconsistencies is important for reasons of efficiency.

The constraint map is constructed greedily, thus, starting from an initially empty set, constraints are added one by one providing they are consistent with the existing set.

The procedure has the following order:

1. Create an empty constraint map.
2. For each set of pairwise alignment parameters (progressively more sensitive):
  - (a) For each pair of sequences in ascending order of pairwise distance:
    - i. For each non-zero length gap in both sequences, between existing anchors compute a chain of monotonically increasing local alignments using the Exonerate program.
    - ii. For each local alignment check if it is consistent with the existing constraint map. If it is add it to the map.

The method employs two nested loops. The outer loops recurses over sets of local alignment parameters. The use of progressively more sensitive parameters is used successfully by a number of other programs [Sammeth et al., 2003] [Schwartz et al., 2003] [Brudno et al., 2003a] [Bray et al., 2003]. Biological sequences are frequently quite heterogeneous in their pattern of conservation. It is sensible therefore to search initially for strong anchors (such as exons) that can be found quickly before narrowing the search. The Exonerate [Slater and Birney, 2005] program is used to generate the local alignments, using the simplest affine-gap local alignment model. We chose Exonerate because it is very fast and we found it to be reasonably sensitive. Before adding anchors to the constraint map the ends are trimmed (by default 3 positions at each end). Even where the core (middle most positions) of an anchor is correct, edge wander effects [Holmes and Durbin, 1998] will often cause misalignments at the ends. By trimming both ends of each anchor the method reduces the risk of introducing misaligned constraints that might greedily prevent

correct anchors from being later added to the set. Anchors smaller than the total trim length are discarded at this point.

For efficiency it is necessary to work with the minimum possible ‘prime’ set of constraints when building the constraint map. Let  $\mathcal{L}_{1\dots n}$  denote the list of input sequences and  $C$  denote the set of pairwise constraints. Myers et al. [Myers et al., 1996] described an  $O(n^2 + n |C|)$  algorithm for computing the set of all prime constraints, where the notation  $|C|$  denotes the total number of input constraints. This algorithm requires complete recomputation when new constraints are added to the set of existing prime constraints. We use a simpler, essentially brute force search algorithm that can be progressively updated with new constraints at relatively constant cost.

In a nutshell, given an existing set of prime constraints,  $C'$ , and a new prime constraint  $x_i \triangleleft y_j$  (For simplicity we omit the details concerning constraint types and simply denote constraints as a single type,  $\triangleleft$ ), which is consistent with this set,  $C'$  is updated by checking if for every ordered pair of sequences  $(w, z)$  if a new prime constraint is implied by a chain of prime constraints of the form  $w_h \triangleleft x_i \triangleleft y_j \triangleleft z_k$  and if so adding it to  $C'$  and removing any existing redundant (now non-prime) constraints. The worst case running time of the method in the context of the pipeline given in the previous section is  $O(n^2 N \log L)$ , where  $L$  is the average length of a sequence and  $N$  is the total length of all the sequences.

For the purposes of this slightly simplified description, and as so far described,  $C'$  can be thought of as a set of  $x_i \diamond y_j$  constraints. Due to the way it is constructed it will not contain any constraints of the form  $x_i \triangleleft y_j$ , and if it contains  $x_i \triangleleft y_j$  then it will also contain  $y_j \triangleleft x_i$ . In practice extra constraints, not of this form, are necessary to ensure proper bounds on the memory management scheme, which is why we implement and describe the fully general constraint framework, rather than a system that just deals with  $x_i \diamond y_j$  constraints, as in [Morgenstern et al., 1998] [Schwartz and Pachter, 2007]. Full details of these extra constraints and proof of the above runtime bound are described in our forthcoming paper.

### *Banded posterior alignment*

After computing the constraint map Pecan computes banded posterior match probabilities for each pair of sequences in the multiple alignment. Even banded calculation of posterior match probabilities is practically unrealistic for large sequences, because memory for the entire band must be held during the two passes of the Forward and Backward algorithms. To avoid this we have found a way to break up the computation into a series of smaller alignment fragments.

An anti-diagonal line in a pairwise edit graph is a line defined by the equation  $i + j = k$ , where  $i$  and  $j$  are

indices in each of the respective sequences and  $k$  is the index of the line. The insight upon which our method is based is that one can generate a very good estimate of the backward distribution along an anti-diagonal ‘cut line’ in the edit graph. This is achieved by choosing a cut line at near the end of high confidence anchor, initialising the anchor point with probability 1 of alignment and propagating this pseudo-sum over all paths from this point back to the cut line. In effect, without forcing any position to be definitively on the final alignment, we suggest that we can estimate the probability of a line abutting an anchor using a relatively local area of alignment, this is shown visually in Figure S1(b) and described in detail below.

The algorithm has the following basic outline:

1. From the global constraint map  $C'$  take the sub-set of constraints for the two considered sequences,  $x$  and  $y$ , compute a ‘relaxed’ alignment band, such for each  $x_i \diamond y_j$  in  $C'$  two constraints,  $x_{i-k} \triangleleft = y_{j+k}$  and  $y_{j-k} \triangleleft = x_{i+k}$  are introduced, where  $k$  is a positive integer parameter. This in effect makes an envelope of some constant width around the anchors, joined by larger ‘squares’ of unanchored alignment.
2. Find a set of approximately evenly spaced anchor-constraints,  $B$ , including the end point  $(n, m)$  of the alignment and a subset of  $C'$ .
3. For each member  $b(i, j)$  of  $B$ , in ascending order:
  - (a) Compute the backward algorithm between the point  $b(i, j)$  and the anti-diagonal cut line  $p_k$  defined by  $i + j - l$ , where the size of  $l$  defines the gap between the anchor and the cut line.
  - (b) Calculate the total probability of the sub matrix up to  $b(i, j)$  by treating  $b(i, j)$  as the end point of the alignment.
  - (c) Calculate posterior match probabilities for all points between the lines  $p_k$  and  $p_{k-1}$  where  $p_{k-1}$  is the previous cut line or anti-diagonal line intersecting the point  $(0, 0)$ . Store only those pairs for which  $P(x_i \diamond y_j \mid \theta)$  is greater than a given threshold (by default 0.01).
  - (d) Store the forward variables for the line  $p$  to be used in the computation of the next fragment.

Thus starting with an alignment band the method breaks up the dynamic programming matrix into a series of small fragments. The points in  $B$  are chosen to intersect the middle portions of anchors (where the confidence will on average be highest) using a greedy (and non-symmetric) parse of the anchors that for

efficiency ensures an even spacing between points. It should be noted that the use of cut-points makes the forward-backward calculations non-symmetric, in that inverting the sequences would potentially produce a different result. In practice however, the approximation has little if any effect on the final alignment.

### *Transitive anchoring*

One of the main drawbacks of consistency alignment programs is the quadratic cost of computing all the underlying pairwise alignments. We utilise triangulation between sequences in a method called ‘transitive anchoring’, which aims to find new highly probable paths in a third, unseen alignment using the presence of two existing sets of posterior match probabilities.

For a pair of sequences  $x, y$  and an out-group sequence  $z$ , we define

$$D_{x,y|z} = \{(x_i \diamond y_j) \mid P(x_i \diamond z_k) > t \text{ and } P(y_j \diamond z_k) > t\},$$

where  $t$  is the probability threshold. We call the members of  $D_{x,y|z}$  ‘transitive anchors’ and more generally the process of refining an initial alignment band with their inclusion ‘transitive anchoring’. For a set of sequences  $\mathcal{L}$  and pair-of sequences in that set  $x, y$  let  $D_{x,y}$  be

$$D_{x,y} = \bigcup_{z \in \mathcal{L} - \{x,y\}} D_{x,y|z}.$$

We construct an outer perimeter constraint envelope around  $D_{x,y}$ , thus the upper right side of the perimeter will be defined by the pairs in the set

$$R = \{(x_{i1} \diamond y_{j1}) \mid \forall (x_{i2} \diamond y_{j2}) \in D_{x,y} \text{ if } i1 \leq i2 \text{ then } j2 > j1\}$$

and similarly the pairs in the lower left side of the perimeter will be defined by

$$L = \{(x_{i1} \diamond y_{j1}) \mid \forall (x_{i2} \diamond y_{j2}) \in D_{x,y} \text{ if } j1 \leq j2 \text{ then } i2 > i1\}.$$

Given these sets, the constraints for the bands can then be projected outwards, thus

$$R' = \{(y_{j-k} \triangleleft x_{i+k}) \mid x_i \diamond y_j \in R\}$$

are the transitive anchor constraints for the upper-right perimeter and

$$L' = \{(x_{i-k} \triangleleft y_{j+k}) \mid x_i \diamond y_j \in L\}.$$



are the transitive anchor constraints for the lower-left perimeter. In this scenario, therefore, it unnecessary to trust any single transitive anchor to contain all the probable alignment paths.

The number of sequence triangles within which a transitive anchoring relationship can be established scales as  $\binom{n}{3}$ . For each triangle the same method is used for choosing the out-group sequence as when there are just three sequences: for each triangle, pick the sequence which is not contained in the pairwise alignment (of the three) with longest branch-length. To see that this method produces a consistent overall ordering of alignments, i.e. that these relationships can specify a linear ordering amongst the pairs without any cycles, simply observe that if the set of pairwise alignments for a group of sequences are placed in ascending order by branch length then this ordering will be consistent with the ordering of alignments for each and every triangle. Currently every informant triangle is used to generate transitive anchors, as this can be calculated as a by-product of the consistency transformation. However, this is not a pre-requisite and thus the method need not scale cubically with the number of sequences.

### *Runtime analysis*

The inputs to Pecan are a list of input sequences  $\mathcal{L}_{1...n}$  of average length  $L$  and total length  $N$ . The cost of computing the constraint map scales  $O(n^2 N \log L)$ . The cost of computing the set of constrained pairwise alignments scales  $O(\binom{n}{2} L)$ . The cost of computing the consistency transform partially eclipses these first two processes, scaling as  $O(n^3 L)$ . In theory Pecan therefore scales by  $O(n^3 L + n^2 N \log L)$ , in practice the coefficients associated with the computation of the pairwise alignments mean it scales closer to  $O(\binom{n}{2} L)$  for moderate numbers ( $<30$ ) of sequences.

### **Training and alignment models**

A key to getting accurate alignments is choosing the correct parameters for the alignment models. For Pecan these models are pair-HMMs. Training a model in the presence of concrete correct examples can be performed by calculating from the data the maximum likelihood parameters for the model  $\theta$  that maximises the value of the likelihood function  $P(D | \theta)$ , where  $D$  is the training data. In the absence of correct examples procedures, the most principled thing to do is to sum across the likelihood of all possible paths (the missing data). For alignment this is generally the situation, where a few hand curated or structurally derived databases exist for ‘correct’ protein alignments [Bahr et al., 2001] [Thompson et al., 2005] [Edgar, 2004], and little or nothing exists for non-coding genomic alignments. We therefore use a method of blind iterative training, known generally as Expectation-Maximisation or Baum-Welch training

for HMMs [Baum, 1972], to attempt to maximise the likelihood of our alignments given a model.

We implement a middle way between Viterbi and Baum-Welch training: a constrained, banded forward-backward approach, the arguments for this being analogous to those used for banded alignment generally. Thus for parts of the alignment in which there is confidence sequence anchors are introduced to constrain the path of the alignment, and in regions in which the path is unclear greater marginalisation over different possibilities is allowed.

Two models were trained, one with a single pair of indel states corresponding to an affine gap model, and one with an additional pair of indel states (double affine model). Affine gap functions are known to unduly penalise long gaps, adding the second pair of indel states therefore allows the better discrimination of long gaps from shorter ones, without resorting to using a dramatically more computationally expensive methodology.

To reduce the dimensionality of the search space we only trained the transitions of the models, assuming that the emissions parameters can be set satisfactorily by using standard nucleotide-substitution models (HKY by default [Hasegawa et al., 1985] with a stable GC frequency of 40% and a transition to transversion ratio of 2).

For training we took the human-dog sequence pairs from three ENCODE regions, ENm005 (Beta-globin locus, 0.5 mega-bases of human sequence), ENm010 (the major Homeobox gene cluster (HoxA), 1.0 mega-bases of human sequence) and ENm001 (the region containing the cystic fibrosis transmembrane conductance regulator (CFTR) gene, 1.8 mega-bases of human sequence).

Regions were chosen because the sequences of interest appeared not to have been subject to a significant number of rearrangement events (rearrangement information was taken from a shuffle-Lagan [Brudno et al., 2003b] data-set provided by George Asimenos). The regions were thus colinear along most of their lengths, although some of the sequences were in the reverse orientation, in which case they were just flipped appropriately. Using sequences that had not been largely rearranged, it was reasoned, would help prevent bias from being introduced by needing to use a pre-processing program, at the risk of using regions that were in some way otherwise unusual.

Figure S3 shows the results of the training. The three different regions show quite close agreement. There is some variance in the gap-opening parameters, we suspect a proportion of the spread in these parameters relates to the variance in the amounts of un-alignable sequence, which is affected by small scale sequence rearrangements, partial assembly and the degree to which the sequences picked are accurately globally alignable along their length. In future we plan to explore scaling the parameters with sequence distance,

though currently the double affine model with the parameters from the human-dog CFTR region (the largest one) are used as the default parameters for Pecan.

## Figures

### Figure S1 - Banded alignment

Diagram (a) shows an edit graph for two sequences (X and Y). Anchors, representing un-gapped alignments are shown as green diagonal lines. Around these anchors a relaxed alignment band is computed (shown as thin black lines). Diagram (b) shows the structure of a cut-point in the matrix. The chosen hidden cut-point is chosen as a position near the middle of an anchor. This point is treated as if the alignment terminates at this position. The forward variables are calculated up to the marked cut line. Backward variables are calculated from the hidden cut-point backwards. Posterior probabilities are calculated for positions in the edit graph contained within the band preceding the cut line.

### Figure S2 - Cartoons illustrating anchor inconsistencies.

Cartoons illustrating anchor inconsistencies. Sequences are represented as black lines, blue blocks represent sub-sequences and the bi-directional arrows represent anchor relationships between the linked blocks. (a) Anchor inconsistency (red arrow crossing a black arrow), causing failure of partial ordering. (b) Transitive anchor inconsistency, the anchor highlighted red is not inconsistent with another anchor between the two sequences, however it breaks the partial ordering with respect to the alignments to the other sequence.

### Figure S3 - Results of Baum-Welch training affine and double affine model using three genomic regions.

(a) Affine model. (b) Double affine model. (c) Training results for affine model. (d) Training results for double affine model. Key: T/P is trained parameter, E is error (calculated from five runs, starting with random initial parameters).

## Tables

### **Table S1 - Pairwise Comparison of the Average Affects the Inclusion of Transitive Anchoring has on Nine-way Alignments of Simulated Sequences.**

The effects of enabling transitive anchoring on the numbers of cells computed and total time taken (seconds) to compute an alignment, on average. Two program configurations were examined, with and without the inclusion of initial anchoring on repeats. Alignments were then run with and without transitive anchoring enabled.

### **Table S2 - A Pairwise Sequence Comparison of the Average Sensitivity and Specificity of different Alignment Programs on a Simulated Data-Set.**

A comparison of different alignment algorithms on 50 50k simulated 9 way alignments. Sensitivity is defined as the proportion of the total residue pairs in the original simulated alignment present in the generated alignment. The specificity conversely represents the proportion of the total residue pairs in the generated alignment not present in the original simulated alignment.

## Bibliography

- Bahr, A., Thompson, J. D., Thierry, J. C., and Poch, O., 2001. BALiBASE (Benchmark Alignment dataBASE): enhancements for repeats, transmembrane sequences and circular permutations. *Nucleic Acids Res*, **29**:323–326.
- Baum, L. E., 1972. An equality and associated maximisation technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, **3**:1–8.
- Blanchette, M., Kent, W. J., Riemer, C., Elnitski, L., Smit, A. F. A., Roskin, K. M., Baertsch, R., Rosenbloom, K., Clawson, H., Green, E. D., *et al.*, 2004. Aligning multiple genomic sequences with the threaded blockset aligner. *Genome Res*, **14**(4):708–15.
- Bray, N., Dubchak, I., and Pachter, L., 2003. AVID: A global alignment program. *Genome Res*, **13**:97–102.
- Brudno, M., Do, C. B., Cooper, G. M., Kim, M. F., Davydov, E., Green, E. D., Sidow, A., and Batzoglou, S., 2003a. LAGAN and Multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA. *Genome Res*, **13**:721–731.
- Brudno, M., Malde, S., Poliakov, A., Do, C. B., Couronne, O., Dubchak, I., and Batzoglou, S., 2003b. Glocal alignment: finding rearrangements during alignment. *Bioinformatics*, **19**:i54–i62.
- Cooper, G. M., Stone, E. A., Asimenos, G., Program, N. C. S., Green, E. D., Batzoglou, S., and Sidow, A., 2005. Distribution and intensity of constraint in mammalian genomic sequence. *Genome Res*, **15**(7):901–13.
- Do, C. B., Mahabhashyam, M. S. P., Brudno, M., and Batzoglou, S., 2005. Probcons: Probabilistic consistency-based multiple sequence alignment. *Genome Res*, **15**(2):330–40.
- Edgar, R. C., 2004. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res*, **32**:1792–1797.
- Hasegawa, M., Kishino, H., and Yano, T., 1985. Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. *J Mol Evol*, **22**:160–174.
- Holmes, I. and Durbin, R., 1998. Dynamic programming alignment accuracy. *J Comput Biol*, **5**:493–504.
- Miller, W., Rosenbloom, K., Hardison, R. C., Hou, M., Taylor, J., Raney, B., Burhans, R., King, D. C., Baertsch, R., Blankenberg, D., *et al.*, 2007. 28-way vertebrate alignment and conservation track in the ucsc genome browser. *Genome Res*, **17**(12):1797–808.
- Morgenstern, B., Frech, K., Dress, A., and Werner, T., 1998. DIALIGN: finding local similarities by multiple sequence alignment. *Bioinformatics*, **14**:290–294.
- Myers, G., Selznick, S., Zhang, Z., and Miller, W., 1996. Progressive multiple alignment with constraints. *J Comput Biol*, **3**:563–572.
- Page, R. D. and Charleston, M. A., 1997. From gene to organismal phylogeny: reconciled trees and the gene tree/species tree problem. *Mol Phylogenet Evol*, **7**(2):231–40.
- Sammeth, M., Morgenstern, B., and Stoye, J., 2003. Divide-and-conquer multiple alignment with segment-based constraints. *Bioinformatics*, **19**:II189–II195.
- Schwartz, A. S. and Pachter, L., 2007. Multiple alignment by sequence annealing. *Bioinformatics*, **23**(2):e24–9.
- Schwartz, S., Kent, W. J., Smit, A., Zhang, Z., Baertsch, R., Hardison, R. C., Haussler, D., and Miller, W., 2003. Human-mouse alignments with BLASTZ. *Genome Res*, **13**:103–107.
- Slater, G. S. C. and Birney, E., 2005. Automated generation of heuristics for biological sequence comparison. *BMC Bioinformatics*, **6**:31.
- Thompson, J. D., Koehl, P., Ripp, R., and Poch, O., 2005. BALiBASE 3. *Proteins*, **61**:127–136.

	Cells			Time		
	No Transitive Anchors	Transitive Anchors	% Difference	No Transitive Anchors	Transitive Anchors	% Difference
Repeat Anchoring	136990669	115982936	15.3	188.8	165.7	12.3
No Repeat Anchoring	438151388	208706597	52.4	393.5	224.6	42.9

Table 1:

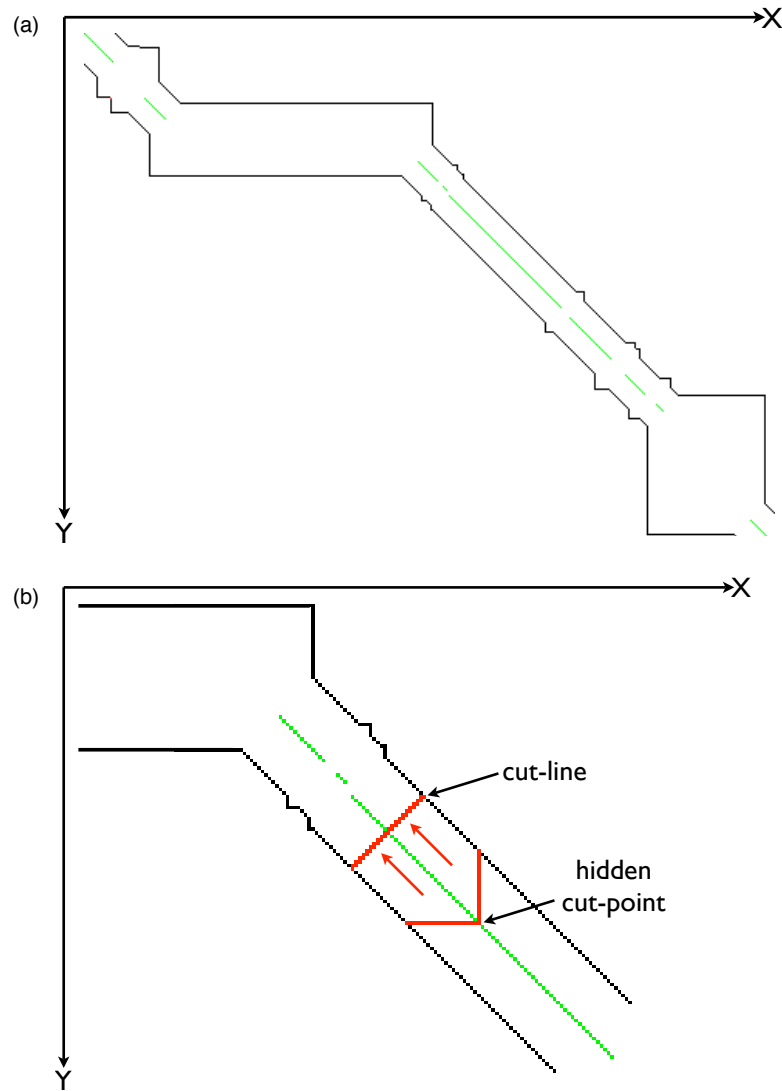


Figure 1:

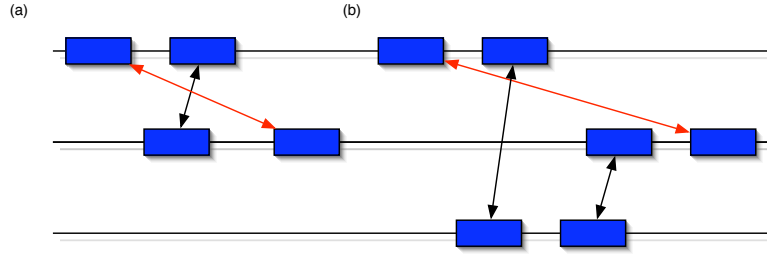


Figure 2:

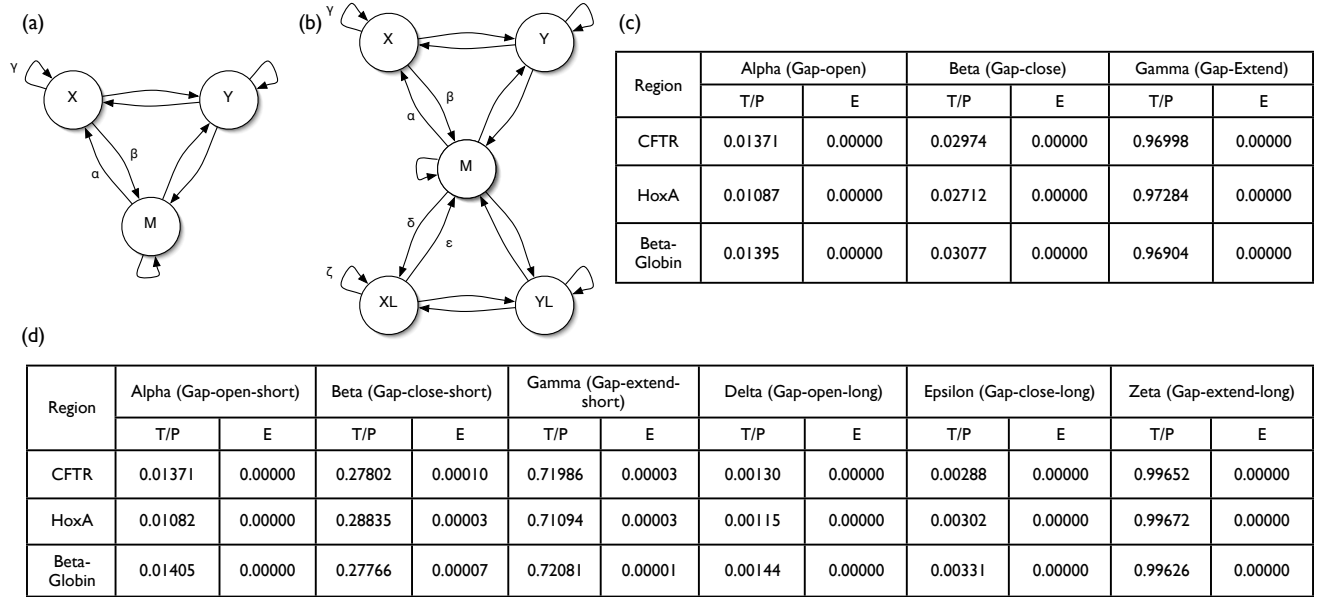


Figure 3:



	Pecan		TBA		MAVID		MLAGAN		DIALIGN		CLUSTAL		DIALIGN/CHAOS	
	Sensitivity	Specificity	Sensitivity	Specificity	Sensitivity	Specificity	Sensitivity	Specificity	Sensitivity	Specificity	Sensitivity	Specificity	Sensitivity	Specificity
Human-Mouse	0.844	0.83	0.724	0.819	0.095	0.233	0.697	0.681	0.662	0.631	0.354	0.267	0.046	0.159
Human-Cow	0.939	0.952	0.89	0.924	0.564	0.785	0.918	0.899	0.866	0.865	0.415	0.353	0.314	0.498
Human-Dog	0.955	0.951	0.869	0.928	0.547	0.802	0.921	0.909	0.875	0.869	0.418	0.378	0.287	0.584
Human-Baboon	0.997	0.997	0.991	0.995	0.995	0.994	0.994	0.994	0.991	0.991	0.983	0.978	0.966	0.975
Pig-Cow	0.978	0.970	0.947	0.963	0.963	0.940	0.966	0.946	0.928	0.923	0.799	0.731	0.455	0.573
Cow-Cat	0.962	0.965	0.941	0.948	0.693	0.875	0.945	0.933	0.913	0.903	0.696	0.626	0.359	0.582
Mouse-Rat	0.968	0.960	0.952	0.956	0.954	0.936	0.961	0.945	0.819	0.859	0.831	0.778	0.286	0.545

Table 2: