Automated Segmentation of DNA Sequences with Complex Evolutionary History

Broňa Brejová, Michal Burger, and Tomáš Vinař

Faculty of Mathematics, Physics, and Informatics, Comenius University
Mlynská Dolina, 842 48 Bratislava, Slovakia

Abstract. Most algorithms for reconstruction of evolutionary histories involving large-scale events such as duplications, deletions or rearrangements, work on sequences of predetermined markers, for example protein coding genes or other functional elements. However, markers defined in this way ignore information included in non-coding sequences, are prone to errors in annotation, and may even introduce artifacts due to partial gene copies or chimeric genes.
We propose the problem of sequence segmentation where the goal is to automatically select suitable markers based on sequence homology alone. We design an algorithm for this problem which can tolerate certain amount of inaccuracies in the input alignments and still produce segmentation of the sequence to markers with high coverage and accuracy. We test our algorithm on several artificial and real data sets representing complex clusters of segmental duplications.

1 Introduction

Genome rearrangements and segmental duplications, acting on long stretches of DNA, pose a significant challenge to comparative genomics. Rearrangements change the order of segments in the genome, resulting in new gene orders and new chromosomal organization. In a typical rearrangement study, we aim at computing the shortest possible number of operations transforming one genome to another or reconstructing a phylogenetic tree and ancestral gene orders (Moret et al., 2001; Bourque and Pevzner, 2002; Adam and Sankoff, 2008).

Segmental duplications increase the length of the sequences by copying genetic material to new locations, creating complex gene clusters, hotspots of evolutionary innovation (Zhang, 2003). Reconstruction of duplication events within such regions is a key to understanding their organization, function, and evolution (Benson and Dong, 1999; Elemento et al., 2002; Zhang et al., 2009; Vinar et al., 2010; Lajoie et al., 2010).

Most algorithms for these tasks do not work directly on the original sequences, but rather on predetermined markers or synteny blocks. These are intervals of the sequence such that all events in the true evolutionary history introduce breakpoints only at or between the boundaries of these intervals, but not inside them. The notion of such intervals was first introduced by Nadeau and

Taylor (1984) (conserved segments) and in this work we call them *atomic segments* or *atoms*. Splitting the sequence into atoms allows algorithm developers to abstract from modeling local sequence alignments, and instead to concentrate on larger-scale processes that reorder and duplicate blocks of segments.

Here, we introduce a new method for segmenting sequence into atoms, primarily targeted at a fine-scale analysis of relatively recent evolutionary events (for example events that happened in the last 85 My of mammalian evolution, which approximately translates to higher than 80% sequence similarity for neutrally evolving sequences). We test our method in the context of reconstruction of duplication histories; however, it is also applicable to other scenarios, including rearrangement studies. In the rest of this section, we give an overview of the methods used previously for this task, and we demonstrate examples of various problems in real data sets that we address by our work.

Related work. Many algorithms for rearrangement or duplication analysis use protein coding genes as atoms (Fitch, 1977; Benson and Dong, 1999; Moret et al., 2001; Elemento et al., 2002; Bourque and Pevzner, 2002; Bertrand and Gascuel, 2005; Lajoie et al., 2007; Adam and Sankoff, 2008; Lajoie et al., 2010). In order to do so, we need to annotate genes in the sequence and establish homology or even orthology among them. The order and strand orientation of the genes is then used as an input for further analysis.

While this is perhaps the only solution applicable to distantly related sequences that need to be aligned at the protein level, it is not universal. First of all, necessary preprocessing, including gene finding and homology or orthology detection, is difficult and can introduce errors. Even in cases where reliable ortholog sets are available, this approach is not relevant for all evolutionary scenarios. Many incomplete pseudogenes present in human and other genomes clearly show that duplications and rearrangements do not respect gene boundaries. For example, the human PRAME gene cluster contains 38 copies of the PRAME locus (preferentially expressed antigens in melanoma), but more than a third of these copies are incomplete pseudogenes (Gibbs et al., 2007). Even more problematic are *chimeric genes* that contain a breakpoint inside an intron. The PRAME gene cluster contains a chimeric gene whose protein sequence consists of two parts with different phylogenetic ancestries. Its inclusion in a phylogenetic analysis may result in a completely incorrect phylogenetic tree.

Another example, where genes are not appropriate as atoms, is the UGT1A cluster. In the human genome, this cluster contains a single alternatively-spliced gene (UDP-glucuronosyltransferase) with at least 13 unique copies of the first exon that apparently arose by segmental duplication (Bellemare et al., 2010). To analyze this sequence, we would have to use exons as atoms instead of genes, but these exons are too short for a reliable gene tree reconstruction, which is a necessary step for many methods. Therefore it would be ideal to use also some of the surrounding non-coding sequences, which have been copied together with the exons, but that requires finding atomic segments unrelated to functional annotation.

Finally, a well-known KRAB zinc finger gene family in the human genome contains more than 400 genes in 25 gene clusters spread across several chromosomes (Schmidt and Durrett, 2004; Huntley et al., 2006). Each zinc finger gene is composed of the KRAB domain and between three and forty zinc fingers. In this family, we can see duplication of whole genes, as well as duplication of zinc finger domains within genes. For both UGT1A cluster and KRAB genes, the traditional selection of atoms (i.e. first exons or zinc finger domains) requires detailed functional annotation and a complex prior knowledge about the studied region. Even with such knowledge, we may create errors due to chimeric atoms or to loose valuable information due to insufficient sequence coverage.

Recently, a new approach to segmentation based on local sequence alignments has been introduced in the context of ancestral genome reconstruction (Ma et al., 2006). Briefly, using sequence alignment tools such as blastz (Schwartz et al., 2003) or UCSC chain/net pipeline (Kent et al., 2003), they identify significant local alignments of the sequence to itself with a desired level of homology and then use boundaries of these sequence alignments as atomic segment boundaries, and regions between the boundaries as atoms. Ma et al. (2006, 2008b) developed a heuristic pipeline that creates a map of such segments considering events of 50kb or more in length, later refining the boundaries to a finer precision. Their resolution is ideal for mammalian whole-genome analyses; however for smaller-scale events (such as analysis of gene clusters), finer resolution is needed.

In our previous work, we have used a simple greedy heuristic (SGH) for this type of analysis (Vinar et al., 2010). Analyzed sequences are first divided into non-overlapping segments of size 500, and for each segment we search for sequence homologies in the rest of the sequence. The segment with the highest number of homologs and its matching homologs are then designated as atoms. All segments overlapping new atoms are removed and the whole process is repeated.

There are several advantages to methods based on local alignments. The analysis is not dependent on possibly error-prone annotations. The atoms often span longer sections of the sequence which allows more accurate determination of phylogeny of individual atom instances. Finally, thanks to potentially finer resolution of atoms, we can use assumptions of infinite sites model (Ma et al., 2008a), such as low breakpoint reuse assumption that often helps to resolve symmetries in duplication event direction (Zhang et al., 2009).

Problem statement. Our goal is to investigate systematic approaches to segmentation of sequences into atoms. The input for our problem consists of several evolutionarily related sequences or one sequence with segmental duplications. We want to find non-overlapping segments called atoms (completely or partially covering the input sequences) and divide atoms into classes so that: (a) coverage of the sequences by atoms is high, to use as much sequence information as possible, (b) the number of atoms is low, to prevent unnecessary segmentation of long atoms, (c) two atoms of the same class share high sequence similarity across their entire length, (d) two atoms of different classes or two parts of the same atom do not appear to be homologous at a chosen sequence similarity threshold.

In practice, one has to find a trade-off these goals and to accommodate imperfect results of homology detection methods. In the rest of the paper we describe our new algorithm for the segmentation problem and evaluate its performance on both simulated and real sequences.

2 Algorithm

Alignments and breakpoint mapping. The input to our algorithm is a set of evolutionarily related input sequences. We use the LASTZ program (Harris, 2007) to align each sequence to itself and to every other sequence. One possible approach to sequence segmentation is to take the resulting set of local alignments, add a breakpoint at every boundary of a local pairwise alignment and to create an atom between every two adjacent breakpoints (Fig.1). To classify atoms, we can simply create a graph with atoms as vertices and alignments between atoms as edges. Then we create one class for each connected component of this graph. This approach would work well on a perfect set of alignments, but on real data we can encounter various artifacts.

In Fig.2, we see an example where one homology was not found by the local alignment program, leading to a missing breakpoint and wrong class assignment in the resulting segmentation. To avoid this problem, we will map boundaries of every alignment through other overlapping alignments to create new breakpoints, as suggested by a dotted line in Fig.2. Mapping a breakpoint through an alignment is easy if the breakpoint is located at an aligned nucleotide. If it is located at a nucleotide aligned with a gap, we find the nearest aligned nucleotide to the left or to the right (whichever is closer) and map it according to this nucleotide.

Iterated homology mapping. In our algorithm we perform breakpoint mapping iteratively, as a newly mapped breakpoint may need to be mapped further to other regions. We call this general process *iterative homology mapping (IHM).*

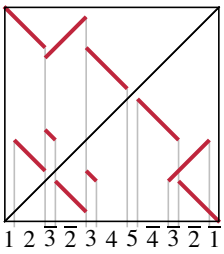

Fig. 1. Simple sequence segmentation. The figure shows a dotplot of alignments of a sequence to itself. We can consider each alignment boundary as a breakpoint; segments between neighbouring breakpoints will form atoms in classes $1, 2, 3, 4, 5$.

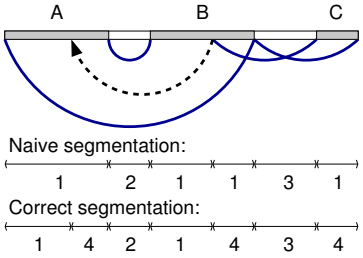

Fig. 2. Example of an input with a missing alignment. Region A is aligned to B and end of B to C, but the alignment between end of A and C is missing. The missing breakpoint in A can be mapped from B through the alignment between A and B, as suggested by the dotted line. Without mapping we get incorrect segmentation.

Another common problem is that boundaries of overlapping local alignments often do not coincide exactly, but are spread around the true breakpoint. This is caused by uncertainty of sequence alignment near alignment boundaries. The problem is even more exacerbated by mapping breakpoints through alignments, as shown in Fig.3. Breakpoints x and y in regions B and C will be mapped to region A through pairwise alignments. Although ideally they should map to the same place, due to imprecision in alignment boundary between B and D, or due to imprecision in pairwise alignment between A and B, the new breakpoint x' is at some distance from the new breakpoint y'. The new breakpoint x' is then mapped to C and y' is mapped to B, again creating pairs of nearby breakpoints. In some cases, the iterative process may create long arrays of breakpoints originating from repeatedly mapping what should have been the same breakpoints through a cycle of imprecise alignments. It is clearly not desirable to create a very short atom between every pair of such nearby boundaries.

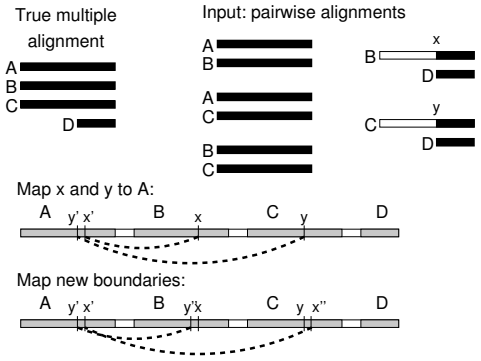

Fig. 3. Example of imprecise breakpoint mapping. Breakpoints x and y are mapped to two different positions in A, and each of them is then further mapped to a new position in B or C. Each of atoms A, B, and C is thus split into three atoms instead of two.

To avoid this problem, we select a window size W and allow at most one breakpoint within each window. This is achieved by clustering breakpoints and replacing each cluster of nearby breakpoints with a new breakpoint roughly at their center. The new breakpoints are chosen so that no two breakpoints are closer than W and the sum of squared distances between the input breakpoints and their new representatives is minimized. This can be done in $O(NW^2)$ time by a dynamic programming algorithm described in the appendix.

Outline of our approach is shown in pseudocode of Algorithm 1. We start with a set of breakpoints created from alignment endpoints. In each iteration of our algorithm, we first cluster breakpoints as described above. Each new breakpoint is then checked against all alignments, and if it is inside an alignment, it is mapped to the other sequence in the alignment. However, we do not map breakpoints that were already mapped in one of the previous iterations. Since breakpoint position may change slightly in each iteration due to the clustering process, we only map breakpoints that are at a distance of more than W from every previously mapped breakpoint. For this purpose we keep a list B_M of all previously mapped breakpoints.

Algorithm 1: Iterative homology mapping

Data: set of alignments A, window length W

$B \leftarrow$ endpoints(A) ; // current breakpoints

$B_M \leftarrow \emptyset$; // all mapped breakpoints

repeat

  $B \leftarrow$ cluster(B, W);

  $B' \leftarrow$ select_to_map(B, B_M, W) ; // get unmapped breakpoints from B

  $B'' \leftarrow$ map(B', A) ; // map B' through A

  $B \leftarrow B \cup B''$; $B_M \leftarrow B_M \cup B'$;

until $B'' = \emptyset$;

return B;

During the whole algorithm, we map at most N/W breakpoints, and therefore the algorithm terminates in at most N/W iterations. In practical instances, the number of iterations is usually quite low, ranging from two to six in the experiments reported in this paper.

Atom classification. Once the breakpoints are fixed, we want to group atoms to classes so that the atoms within a class form a cluster densely connected by alignments, and there are relatively few alignments between atoms from different classes. This can be formulated as an optimization problem, where we seek to minimize the weighted sum of the number of false positive alignments (alignments connecting atoms from two different classes) and false negatives (pairs of atoms in the same class not connected by an alignment). This is a weighted variant of the NP-complete Cluster Editing Problem (Shamir et al., 2004).

We solve this problem exactly by CPLEX software from IBM using integer linear programming (ILP) formulation (the integer program is shown in the Appendix). In order to efficiently process inputs with large numbers of atoms, we employ several simple heuristics. First of all, atoms that are in different components of the alignment connectivity graph will never be in the same class in the optimal solution of the ILP. Therefore we process each connected component separately. We also do not run ILP on components that form a clique, because the optimal solution has then cost 0. If the size of a component exceeds 200, for efficiency reasons we use a different graph clustering strategy implemented in the MCL program (Van Dongen, 2008).

In the process of classification we also assign a strand to each atom so that they are consistent with alignments (each alignment suggests either that the two atoms should be on the same strand or on the opposite strands).

Segmentation postprocessing. In the resulting segmentation, we sometimes see a pair of atom classes a and b such that each atom of class a is always followed by an atom of class b and an atom of class b is always preceded by an atom of class a. We replace each such pair by a single atom, because there is no evidence of a breakpoint between a and b at the segmentation level. We perform such postprocessing on predicted segmentations as well as on the true segmentation in the simulated data. If the segmentation does not cover the whole sequence, the two atoms do not need to be adjacent in the sequence, as long as there are no further atoms between them.

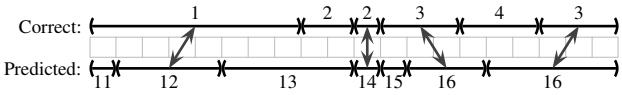

Fig. 4. An example of reciprocal best matches between two segmentations. BRMs are shown as arrows. Each atom is labeled by its class. BRM sensitivity is 4/6, specificity is 4/7. Classes 12 and 16 are correctly predicted.

Since shorter atoms, with length close to window length W, are often less accurate than longer ones, in some tests we also filter out all classes that have all atoms shorter than some threshold $T > W$.

3 Experiments

Here, we evaluate our methods in the context of duplication history reconstruction of gene clusters from several related species. We have tested our new IHM algorithm on both simulated and real data, comparing it to the simpler algorithm from Vinar et al. (2010), which we will call SGH (Simple Greedy Homology).

Measuring segmentation accuracy. If we know the true segmentation of a sequence, which is the case for artificial sequences generated from an evolutionary model, we can measure the quality of the predicted segmentation directly. To compare two segmentations, we first compute *reciprocal best matches* (BRM) between their atoms (see Fig.4). In particular, an atom from one segmentation is a BRM of an atom in another segmentation if they cover overlapping regions of the sequence and no other atom overlaps either of the two by a larger amount.

We use four quality measures comparing the predicted segmentation to the true segmentation. Let p the number of atoms in the predicted segmentation, t the number of atoms in the true segmentation, and b be the number of BRM pairs between them. We define BRM sensitivity as b/t and BRM specificity as b/p. If, for example, an algorithm splits a true atom into two predicted atoms, one of them will not have a BRM pair, and therefore the BRM specificity will decrease. Similarly, a predicted atom spanning two real atoms will lead to a decrease in the BRM sensitivity.

The other two measures focus on the correctness of atom classification. Class C_1 is correctly predicted if each of its atoms has a BRM atom in the same true class C_2 and each atom in class C_2 has a BRM atom in C_1. Class sensitivity is then c/t and specificity is c/p, where c is the number of correctly predicted classes, t the number of true classes, and p the number of predicted classes.

Data sets. We have tested our algorithms on 30 simulated data sets divided into three categories with different parameter settings (see the overview in Table 1). To produce them, we have simulated sequence evolution, allowing substitutions according to the HKY model (Hasegawa et al., 1985), short insertions and deletions, as well as large-scale deletions and duplications. The simulation started with a 100kb sequence and proceeded along the human, chimpanzee, and rhesus macaque phylogeny. The parameters of the substitution model, branch lengths of the phylogenetic tree, and rate and length distributions of short insertions and

deletions were estimated from UCSC syntenic alignments (Fujita et al., 2011) of human, chimpanzee, and macaque on the human chromosome 22. Parameters of large-scale events (duplications and deletions) were taken from Vinar et al. (2010). The Slow and Fast data sets differ in the rate of large-scale events per site, with Fast data sets using 1.5 times the slower rate. The No indel data sets were taken from Vinar et al. (2010) (first 10 out of 20 sets labeled as 300 in that paper). These sequences were generated by a simpler simulator that did not allow short insertions and deletions and also assumed that the rate of large-scale duplication and deletion is constant per sequence and does not depend on the sequence length.

Segmentation accuracy on simulated data. We have segmented all simulated data sets with our new method IHM, setting $W = 250$ and discarding atoms shorter than 500. The SGH program from Vinar et al. (2010) also produces atoms of length at least 500. On the first two categories, IHM noticeably outperforms SGH, particularly in sensitivity at both the BRM and class levels (Table 2). Since both programs work at the resolution of 500bp, they cannot predict very short atoms, which is why their sensitivity is lower than their specificity. For comparison, we also show the accuracy of the true segmentation with the atoms shorter than 500bp filtered out. Our program is very close to this ideal sensitivity. Even without filtering, IHM maintains high specificity combined with much higher sensitivity than the filtered IHM segmentation.

The last category of data does not contain small-scale indels, which makes the segmentation problem much easier. Due to the higher number of short atoms, the sensitivity is relatively small at resolution of 500bp, but almost identical for both programs as well as for the true segmentation filtered at 500. Both programs have achieved perfect specificity on these data sets, that is, every predicted atom is covered by BRM and all predicted classes agree with the true segmentation. The marked difference is in the boundary placement accuracy. About 86% of the BRM atoms produced by IHM have both their boundaries within 50nt of the correct boundary, but this fraction is only 19% for the SGH method. This is because IHM boundaries ultimately originate in alignment endpoints, whereas SGH starts with arbitrarily placed sequence windows.

Influence of segmentation on evolutionary history reconstruction. In Vinar et al. (2010), we have used the true segmentation of simulated sequences as a starting point for evolutionary history reconstruction under a probabilistic model encompassing substitutions and large-scale duplications and deletions. In this work, we compare the accuracy of the history reconstruction when run on different seg-

Table 1. Overview of simulated data sets. Each category contains ten data sets. The table lists mean values of various statistics over these data sets, or in the case of sequence length, over all sequences from all data sets in the category combined.

	Sequence	Segmentation		No. events	
Data set	length (kb)	No. atoms	No. classes	dupl.	del.
Slow	253	153	55	28	2.6
Fast	444	385	113	56	3.8
No indels	210	611	78	25	1.2

Table 2. Accuracy of segmentation on simulated sets. TRUE500 is the true segmentation with atoms shorter than 500 filtered out. SGH500 is the segmentation created by method from Vinar et al. (2010). IHM250 is our new algorithm with $W = 250$ and IHM250.500 is the same method, only with atoms shorter than 500 filtered out. Sensitivity and specificity at the BRM and class level are computed as explained in the text.

		BRM		Class	
Set	Program	sn	sp	sn	sp
Slow	TRUE500	86%	100%	89%	100%
	SGH500	63%	97%	45%	83%
	IHM250.500	86%	100%	88%	100%
	IHM250	95%	100%	96%	100%
Fast	TRUE500	80%	100%	86%	100%
	SGH500	65%	99%	52%	91%
	IHM250.500	79%	100%	84%	100%
	IHM250	92%	100%	94%	99%
No indels	TRUE500	55%	100%	61%	100%
	IHM250.500	56%	100%	62%	100%
	SGH500	56%	100%	60%	100%
	IHM250	86%	100%	86%	98%

Table 3. The predicted number of evolutionary events by the MCMC method. For each dataset D00-D09 in the No indel series, the left column indicates the number of duplications and the right column the number of deletions. The two rows labeled History list the actual number of events in the simulated history, counting only events affecting at least one atom of length of at least 500 or 250 respectively. The remaining rows show difference between the actual count and the count observed in the maximum likelihood history predicted by the MCMC algorithm (Vinar et al., 2010) run on different segmentations.

Program	D00		D01		D02		D03		D04		D05		D06		D07		D08		D09	
History	24	2	24	0	24	1	18	1	24	1	28	1	18	2	22	3	29	1	21	0
TRUE500	0	0	0	0	0	+2	0	0	-1	0	0	0	0	0	-1	+1	-1	1	0	0
SGH500	+1	-1	0	+1	0	+2	0	0	0	0	0	0	0	0	-1	+1	0	0	0	0
IHM250.500	0	0	0	0	+1	+2	0	0	-1	0	0	0	0	0	-1	+1	0	0	0	0
History	25	2	26	0	27	1	18	1	26	1	28	1	19	2	23	3	30	1	22	0
IHM250	0	0	0	+2	+1	+2	0	+1	+1	0	0	+1	0	+1	0	0	-1	+1	0	+1

mentations using a subset of the simulated data from Vinar et al. (2010) (Table 3). We observe that in this case, using the predicted segmentation instead of the true segmentation filtered at 500bp does not have a large impact on the accuracy of the history reconstruction, yielding in most cases the same or very similar number of events.

Segmentation of primate gene clusters. Finally, we have applied our algorithm to the study of three complex primate gene clusters (PRAME, AMY and UGT1A), considered in Vinar et al. (2010). Before running the segmentation algorithms, we have masked the sequences with RepeatMasker, and then excised all masked or unknown bases, obtaining a shorter sequence without repeats.

On all three sets, SGH has produced more atoms (Table 4). A large majority of IHM atoms (81-91% in different sets) have a BRM atom in the SGH segmentation, but only 37%-76% of classes agree completely with the SGH. We have used these new segmentations to infer evolutionary history under the model of large-scale duplications and deletions.

Table 4. Results on complex primate gene clusters. Data set size, segmentation parameters, and the predicted number of duplications and deletions from the MCMC algorithm (Vinar et al., 2010). Species abbreviations: human (H), chimpanzee (C), orangutan (O), rhesus (R).

		Sequences		Segmentation			No. events	
Set	Program	species	lengths (kb)	atoms	classes	coverage	dup.	del.
PRAME	SGH500	H,R	373,92	454	57	60%	77	21
	IHM250.500			288	51	74%	41	17
AMY	SGH500	H,R	105,89	118	21	82%	14	10
	IHM250.500			90	17	98%	12	8
UGT1A	SGH500	H,C,O	90,87,108	138	17	55%	12	11
	IHM250.500			134	23	76%	11	17

The IHM segmentations consistently lead to fewer events of both kinds (duplications and deletions), with the exception of deletions in the UGT1A cluster. In this case, the IHM segmentation contains four atoms that are unique to the orangutan region. Since the history reconstruction does not allow insertions, these four segments are explained as deletions in the human-chimpanzee lineage. On the other hand, the SGH does not predict atoms with only a single occurrence, and thus these four deletions are not included in the history.

4 Conclusion

We have introduced the problem of automated segmentation of sequences with complex evolutionary histories and proposed an accurate and efficient algorithm. Unlike marker based methods, our algorithm can be used on sequence alone and can be easily incorporated into sequence analysis pipelines. The method is suitable as a preprocessing step for duplication history reconstruction, and it can also be applied to sequences with other large-scale events.

Our method consists of two stages: finding breakpoints and atom classification. Errors introduced in the first stage cannot be resolved later on. One would ideally solve both stages simultaneously. There are two obstacles to this course. First, we need to optimize various contradictory criteria (coverage, the number of false positives and false negatives, etc.). We can either combine them into a single objective function by weights, or we can put a constraint on some criteria by thresholds and optimize the remaining ones, but choosing the weights and thresholds is difficult to do in a principled way. The second problem is that even with breakpoints fixed, the problem of atom classification is already NP-hard (Shamir et al., 2004). Nonetheless, it is possible to investigate heuristics or approximation approaches to tackle this problem.

One can go even further and combine the segmentation and reconstruction of evolutionary histories. Scoring of potential segmentations is then implied directly by the underlying evolutionary model. Such approaches were attempted in the duplication scenario (Zhang et al., 2009; Song et al., 2010). Nonetheless, segmentation makes the history reconstruction problems cleaner and allows us

to filter out problematic regions of the sequence (such as shorter atoms in our experiments).

Another, perhaps less ambitious, avenue for improvement, is in the combination of segmentation and multiple alignment. Ideally all atoms of the same class would form a high-quality multiple alignment. Our algorithm relies solely on pairwise alignments, yet multiple alignments of longer homologous regions could help us to map breakpoints more consistently.

Finally, our method was targeted at recently diverged atoms, where it is possible to recognize homology at a nucleotide sequence level. The question of extending our approach to more distant sequences remains open.

Acknowledgements. We would like to thank Webb Miller and Giltae Song for collaboration on various gene cluster problems. This research was supported by Marie Curie grants IRG-224885 and IRG-231025 to TV and BB, and VEGA grant 1/0210/10.

Bibliography

Adam, Z. and Sankoff, D. (2008). The ABCs of MGR with DCJ. *Evolutionary Bioinformatics Online*, 4:69–74.

Bellemare, J., Rouleau, M., Girard, H., Harvey, M., and Guillemette, C. (2010). Alternatively spliced products of the UGT1A gene interact with the enzymatically active proteins to inhibit glucuronosyltransferase activity in vitro. *Drug Metabolism and Disposition*, 38(10):1785–1789.

Benson, G. and Dong, L. (1999). Reconstructing the duplication history of a tandem repeat. In *Intelligent Systems for Molecular Biology (ISMB)*, pages 44–53.

Bertrand, D. and Gascuel, O. (2005). Topological rearrangements and local search method for tandem duplication trees. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(1):15–28.

Bourque, G. and Pevzner, P. A. (2002). Genome-scale evolution: reconstructing gene orders in the ancestral species. *Genome Research*, 12(1):26–36.

Elemento, O., Gascuel, O., and Lefranc, M.-P. (2002). Reconstructing the duplication history of tandemly repeated genes. *Molecular Biology and Evolution*, 19(3):278–278.

Fitch, W. M. (1977). Phylogenies constrained by the crossover process as illustrated by human hemoglobins and a thirteen-cycle, eleven-amino-acid repeat in human apolipoprotein A-I. *Genetics*, 86(3):623–624.

Fujita, P. A. et al. (2011). The UCSC Genome Browser database: update 2011. *Nucleic Acids Research*, 39(D):D876–882.

Gibbs, R. A. et al. (2007). Evolutionary and biomedical insights from the rhesus macaque genome. *Science*, 316(5822):222–224.

Harris, R. (2007). *Improved pairwise alignment of genomic DNA*. PhD thesis, Pennsylvania State University.

Hasegawa, M., Kishino, H., and Yano, T. (1985). Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. *Journal of Molecular Evolution*, 22(2):160–164.

Huntley, S. et al. (2006). A comprehensive catalog of human KRAB-associated zinc finger genes: insights into the evolutionary history of a large family of transcriptional repressors. *Genome Research*, 16(5):669–677.

Kent, W. J. et al. (2003). Evolution's cauldron: duplication, deletion, and rearrangement in the mouse and human genomes. *Proc Natl Acad Sci U S A*, 100(20):11484–9.

Lajoie, M., Bertrand, D., and El-Mabrouk, N. (2010). Inferring the evolutionary history of gene clusters from phylogenetic and gene order data. *Molecular Biology and Evolution*, 27(4):761–762.

Lajoie, M., Bertrand, D., El-Mabrouk, N., and Gascuel, O. (2007). Duplication and inversion history of a tandemly repeated genes family. *Journal of Computational Biology*, 14(4):462–468.

Ma, J., Ratan, A., Raney, B. J., Suh, B. B., Miller, W., and Haussler, D. (2008a). The infinite sites model of genome evolution. *Proc of the National Academy of Science USA*, 105(38):14254–61.

Ma, J., Ratan, A., Raney, B. J., Suh, B. B., Zhang, L., Miller, W., and Haussler, D. (2008b). DUPCAR: reconstructing contiguous ancestral regions with duplications. *Journal of Computational Biology*, 15(8):1007–1007.

Ma, J., Zhang, L., Suh, B. B., Raney, B. J., Burhans, R. C., Kent, W. J., Blanchette, M., Haussler, D., and Miller, W. (2006). Reconstructing contiguous regions of an ancestral genome. *Genome Research*, 16(12):1557–1565.

Moret, B. M., Wang, L. S., Warnow, T., and Wyman, S. K. (2001). New approaches for reconstructing phylogenies from gene order data. *Bioinformatics*, 17(S1):S165–173.

Nadeau, J. H. and Taylor, B. A. (1984). Lengths of chromosomal segments conserved since divergence of man and mouse. *Proceedings of the National Academy of Science USA*, 81(3):814–818.

Schmidt, D. and Durrett, R. (2004). Adaptive evolution drives the diversification of zinc-finger binding domains. *Molecular Biology and Evolution*, 21(12):2326–2329.

Schwartz, S. et al. (2003). Human-mouse alignments with BLASTZ. *Genome Research*, 13(1):103–107.

Shamir, R., Sharan, R., and Tsur, D. (2004). Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182.

Song, G., Zhang, L., Vinar, T., and Miller, W. (2010). CAGE: combinatorial analysis of gene-cluster evolution. *Journal of Computational Biology*, 17(9):1227–1232.

Van Dongen, S. (2008). Graph clustering via a discrete uncoupling process. *SIAM Journal on Matrix Analysis and Applications*, 30:121.

Vinar, T., Brejova, B., Song, G., and Siepel, A. C. (2010). Reconstructing histories of complex gene clusters on a phylogeny. *Journal of Computational Biology*, 17(9):1267–1279.

Zhang, J. (2003). Evolution by gene duplication: an update. *Trends in Ecology and Evolution*, 18(6):292–298.

Zhang, Y., Song, G., Vinar, T., Green, E. D., Siepel, A., and Miller, W. (2009). Evolutionary history reconstruction for Mammalian complex gene clusters. *Journal of Computational Biology*, 16(8):1051–1060.

Appendix

In this appendix we describe details of our methods that were omitted from the main text for space reasons.

Alignment preprocessing and breakpoint mapping. In the preprocessing stage of the algorithm we discard alignments shorter than W and split each input alignment into multiple parts if there are any gaps longer than W; boundaries of such gaps will be breakpoints at our chosen resolution.

In the main algorithm, we map a breakpoint through a particular alignment only if the alignment extends at least $W/3$ on both sides of the breakpoint. The reason is that an alignment boundary nearby a breakpoint likely represents an imprecise copy of the same breakpoint.

Breakpoint clustering. The goal of breakpoint clustering is to take a set of breakpoints that were created either from alignment endpoints or by mapping previously identified breakpoints through alignments, and to replace groups of nearby breakpoints by a single new breakpoint so that no two breakpoints are closer than W.

The input to our problem is an integer sequence X of length N where $X[i] \geq 0$ is the number of input breakpoints at position i. We are seeking a sequence of breakpoints $Y = y_1, \ldots y_k$ such that $y_j \geq y_{j-1} + W$. Let $d(i, Y)$ be the distance from i to the closest breakpoint in Y, that is, $d(i, Y) = \min_j |y_j - i|$. Our goal is to find Y minimizing the sum of squared distances for the input breakpoints $\sum_i X[i] d(i, Y)^2$.

This problem can be solved by dynamic programming. For every prefix of sequence X, we compute the cost of the optimal solution $A[i]$ with an added constraint that the rightmost breakpoint in Y is at position i. To compute $A[i]$, we consider all possible positions p of the previous breakpoint. From our constraints, $p \leq i - W$, but we will also assume that $p \geq i - 2W + 1$, because if p was smaller, we could add one more breakpoint which would be at the distance of at least W from both i and p. All input breakpoints to the left of p will have as their closest breakpoint either p or some breakpoint even further to the left. To compute $A[i]$, we therefore need to compute distances only for points between p and i as follows:

$$A[i] = \min_p A[p] + \sum_{j=p+1}^{i-1} X[j] \min\{(i-j)^2, (p-j)^2\}.$$

For each i, we also keep $B[i]$ which is the value of p at which the minimum was obtained. Cost of the optimal solution for the whole input is the smallest value among $A[N], \ldots, A[N+W-1]$. Then we can use array B to recover the sequence of breakpoints in the optimal solution. Finally, we prune out those breakpoints from Y that are not the closest breakpoint for any input breakpoint. This change does not have any impact on the optimality of the resulting set.

The running time of this algorithm is $O(NW^2)$ and can be further improved if the set of input breakpoints is sparse. For example, the sum over j in the recurrence formula is done only for values where $X[j]$ is positive. Similarly, if a gap between successive input breakpoints is much greater than W (e.g. greater than $4W$), we can split the input into two parts at this gap, and solve each part independently, saving $O(W^2)$ time for every position which is not sufficiently close to any input breakpoint.

Atom classification. Once we fix the position of breakpoints, we need to partition the resulting atoms to equivalence classes such that all atoms in the same class are assumed to be homologs. We also need to assign a strand to each atom: each alignment suggests either that the two atoms should be on the same strand or on the opposite strands. The input to the classification problem are thus two sets A_+ and A_-, where A_+ is the sets of pairs of atoms connected by an alignment between the same strands and A_- between the opposite strands. We consider two atoms to be connected by an alignment if the alignment covers at least 50% of the length of both atoms.

We formulate the atom classification problem as an integer linear program as follows. For every pair of atoms i and j, we define binary variables $x_{i,j}$ and $y_{i,j}$. In the optimal solution, the value of $x_{i,j}$ is one if and only if atoms i and j are in the same class on the same strand, and $y_{i,j}$ is one if and only if i and j are in the same class on the opposite strands. Our goal is to minimize a weighted sum of false positives and false negatives in the input alignment sets

$$\sum_{(i,j)\in A_+} (1 - x_{i,j}) + \sum_{(i,j)\in A_-} (1 - y_{i,j}) + \sum_{(i,j)\notin A_+} w_F x_{i,j} + \sum_{(i,j)\notin A_-} w_F y_{i,j}$$

where w_F is the weight of false negatives (we use $w_F = 0.6$). We optimize this sum under a set of linear constraints enforcing that the variables correspond to a valid classification.

The first set of constraints enforces symmetry, i.e. $x_{i,j} = x_{j,i}$ and $y_{i,j} = y_{j,i}$. In order to save memory and computation, we actually only use variables $x_{i,j}$ and $y_{i,j}$ for $i < j$, and thus these constraints are not needed. In addition, each atom has a unique strand, which means that $x_{i,j} + y_{ij} \leq 1$. Relation "to be in the same class on the same strand" needs to be transitive, that is, for all distinct i, j and k we have a constraint $x_{i,j} + x_{j,k} \leq 1 + x_{i,k}$. For pairs of atoms on different strands, we have a modified version of transitivity that correctly propagates strand signs: $x_{i,j} + y_{j,k} \leq 1 + y_{i,k}$, $y_{i,j} + x_{j,k} \leq 1 + y_{i,k}$,, and $y_{i,j} + y_{j,k} \leq 1 + x_{i,k}$.

The number of transitivity constraints is cubic in the number of atoms, making the ILPs rather large, and memory and time intensive for larger numbers of atoms. In practice, we solve this problem by several simple heuristics. First of all, atoms that are in different components of the alignment connectivity graph will never be in the same class in the optimal solution of the ILP. Therefore we process each connected component separately. We also identify components, where

the optimal solution has cost 0. Such components must form a clique, and their atoms can be partitioned into two strands so that the strand of each alignment is consistent with this partition. ILP is not run on such perfect components.

If the size of a component exceeds 200, we use a different graph clustering strategy implemented in the MCL program (Van Dongen, 2008) with inflation option set to 4. For each atom, we include two copies as vertices, one for each strand. Each alignment corresponds to two edges, one connecting a plus strand atom to its counterpart and one connecting a minus strand atom. Ideally, both strands are assigned to classes consistently in the MCL output. If this is not the case, we use a simple heuristics to obtain a consistent classification. In particular, we merge classes until both copies of each atom are in the same class and we greedily flip strand of each original MCL cluster whenever two copies of the same atom have been assigned to the same strand.