GENOMIC REARRANGEMENT ALGORITHMS

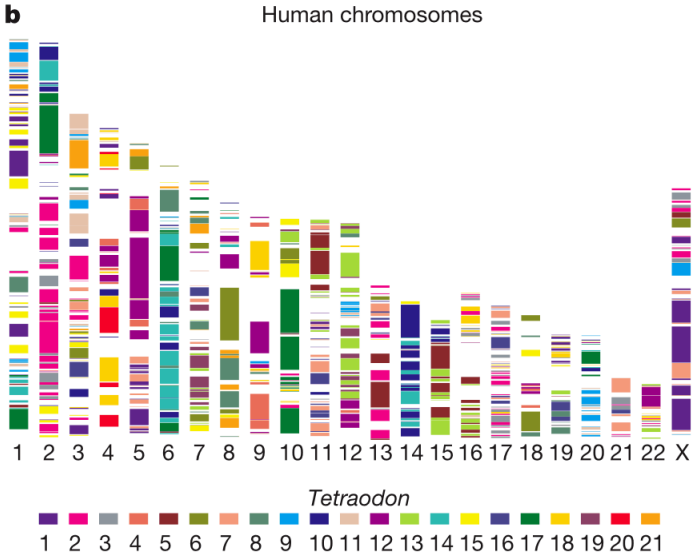
KAREN LOSTRITTO

Abstract. In this paper, I discuss genomic rearrangement. Specifically, I describe the formal representation of these genomic rearrangements as well as two basic algorithms for transforming between one genome permutation and another. I discuss the most recent advances in the field and concepts employed in these algorithms. I then propose an idea for obtaining a better understanding of the biological significance of these algorithms. Although this topic is related to the assigned synteny mapping topic, it is different in that I am assuming that the synteny map already exists, and I am exploring the calculation of genomic rearrangement and distance to yield the synteny map.

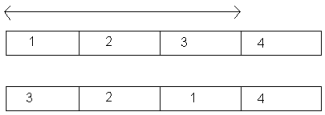
Throughout evolution, genomes have been rearranged through the breaking and rejoining of the DNA backbone which results in scrambling of the genome. However large chunks of DNA which did not contain breaks remain intact. Therefore by comparing different organisms A and B with a common ancestor, we can track what and where each chunk of DNA in organism B corresponds to in organism A. This information can be depicted using a synteny map in which we start with the karyotype of organism A and assign different colors to chunks of chromosomes to represent what chromosome they correspond to in organism B, shown in figure 1[1].

Since genes from organism A and organism B will start out in the same order in a common ancestor, the question arises of how the genome was cut apart and pasted back together to form the gene order in organism A and organism B. So given a synteny map of organism A, we ask how it can be rearranged to yield organism B. This task is quite a challenge even though in cases like mice and humans there have only been 250 genomic rearrangements between their genomes. We can define the problem of transforming between genome A and genome B by defining a set of

FIGURE 1. Synteny Map [2]



operations which we will use to transform the genome. The question is then determining the fewest number of operations necessary to complete this transformation. This number does not tell us necessarily how the transformation actually occurred in evolution, but it gives us a lower bound on the number of operations to perform the rearrangement. However determining the fewest number of rearrangements is rather difficult. An example of the operations used to define a rearrangement is the operation of inversions (reversal). One step in this rearrangement is as follows where segments 1, 2, and 3 are reversed.



In order to formalize modeling reversals, we define a permutation as the order of the chunks (or genes) in the synteny map, so in the example above, we start with permutation (1,2,3,4) and change to (3,2,1,4). In a signed permutation, the sign of the integer represents the orientation of the gene. If a permutation π is defined as

$\pi_1\pi_2\pi_3...\pi_n$ a reversal between positions i and j, p(i,j), yields the following transformation:

$p(i,j) * \pi_1..\pi_i\pi_{i+1}..\pi_j\pi_{j+1}...\pi_n \rightarrow \pi_1...\pi_j...\pi_{i+1}\pi_i\pi_{j+1}...\pi_n$

So the reversal distance is then just the minimum number of reversals necessary to transform permutation π into permutation σ, and the reversal problem is computing the shortest series of reversals to transform between permutation π and permutation σ [3]. However there are many other possible operations to consider. For example, we can use transpositions to transform rearrangements. A transposition p(i,j,k) can be seen as switching two adjacent blocks in the permutation. Formally this is defined as:

$$p(i,j,k)*\pi_1..\pi_{i-1}\pi_i\pi_{i+1}...\pi_{j-1}\pi_j...\pi_{k-1}\pi_k\pi_{k+1}....\pi_n \rightarrow \pi_1..\pi_{i-1}\pi_j....\pi_{k-1}\pi_i\pi_{i+1}...\pi_{j-1}\pi_k....\pi_n$$

Similarly, the transposition distance is the number of transpositions necessary to transform π into σ. An inverted transposition is like the transposition above except that one of the switched blocks has its order reversed. If it is a signed transposition, then the sign of each reversed gene also gets switched. An example of an inverted transposition follows:

$$p(i,j,k)*\pi_1..\pi_{i-1}\pi_i\pi_{i+1}...\pi_{j-1}\pi_j...\pi_{k-1}\pi_k\pi_{k+1}....\pi_n \rightarrow \pi_1..\pi_{i-1}\pi_j....\pi_{k-1}\pi_{j-1}...\pi_{i+1}\pi_i\pi k....\pi_n$$

A block interchange works in the same way as a transposition except that the blocks to be switched are not adjacent, in other words, a transposition is a special case of a block interchange[3]. Another type of genomic rearrangement is a translocation, which exhanges segments between two different permutations (which can be seen as different chromosomes). There are prefix-prefix translocations like the following:

$$p(\pi,\sigma,i,j) * (\pi_1...\pi_i...\pi_n),(\sigma_1...\sigma_j...\sigma_n) \rightarrow (\pi_1...\pi_{i-1}\sigma_j....\sigma_n),(\sigma_1...\sigma_{j-1}\pi_i...\pi_n)$$

There are also prefix-suffix translocations where the prefix of one permutation is swapped with the suffix of another permutation. A fusion of two permutations $p(\pi, \sigma)$ is concatenating one permutation right after the other, and fission of a permutation $p(\pi, i)$ splits the permutation into two permutations, one from positions 1 to i-1 and the other from positions i to n [3]. If we add the elements 0 and n+1 to a permutation yielding $(0, \pi_1....\pi_n, n+1)$, we can then define the concept of a breakpoint as any pair (π_i, π_{i+1}) where $|\pi_{i+1} - \pi_i| \neq 1$ for any i between 0 and n. The set of breakpoints defines the idea of strips, where $\pi_i...\pi_j$ is a strip if there are no breakpoints for any pairs between i and j and (π_{i-1}, π_i) and (π_j, π_{j+1}) is a breakpoint. For example, in $(0, 3, 4, 8, 5, 6, 7, 2, 1, 9)$, the breakpoints are $(0, 3), (4, 8), (8, 5), (7, 2), (1, 9)$, and the strips are $(0), (3, 4), (8), (5, 6, 7), (2, 1), (9)$[3, 4].

Given an algorithm which can perform the transformation of one permutation to another, we ask how "good" it is, meaning how close does it come to being optimal i.e. fewest number of operations. This problem is identical to that of transforming permutation π into the identity permutation $I = (1, 2, 3, n)$ since transforming π into σ would just involve transforming π into I and σ into I and then reversing the steps for transforming σ into I. Therefore we consider the problem of transforming π into I. Our algorithm A will perform $A(\pi)$ steps to transform π into I. Then say that the optimal algorithm O, which we may not know, will perform $O(\pi)$ steps to transform π into I. Therefore $A(\pi)$ will be at least as big as $O(\pi)$. A high level of optimality would imply that $\frac{A(\pi)}{O(\pi)}$ is close to 1 (as opposed to greater than 1). We calculate the ratio for all sequences with the same length as π and find the maximum of this ratio, which basically tells us the worst case for our algorithm to differ from the optimal solution [4]. So formally we are calculating:

$$\max_{|\pi|=n} \frac{A(\pi)}{O(\pi)})$$

Assuming that we are just using reversals to transform from one rearrangement to another, I will explain the following SimpleReversal Sort algorithm which can be used to transform from genome A to genome B. Although this algorithm will

guarantee that we successfully transform between these two genomes, it is not optimal, meaning that it will not perform the transformation in the least number of steps. The following steps transform π into I. The algorithm first locates the element numbered one and performs a reversal of the segment between it and the first position, so now number one is correctly positioned. It then locates element two and performs a reversal between its position and the second position. In this way each reversal step increases by one the number of correctly positioned numbers until after n-1 steps all n numbers are in the correct order[4]. However this is by no means the fewest steps required to transform π into the identity. Here is an example of this algorithm:

$$4, 1, 2, 5, 3 \rightarrow 1, 4, 2, 5, 3 \rightarrow 1, 2, 4, 5, 3 \rightarrow 1, 2, 3, 5, 4 \rightarrow 1, 2, 3, 4, 5$$

Compared with the SimpleReversalSort algorithm presented above, the following algorithm uses the concept of breakpoints to obtain a closer to optimal solution. First we extend our permutation by including $\pi_0 = 0$ and $\pi_{n+1} = n + 1$. Since the identity permutation has no breakpoints, one can view the task of transforming π into the identity as equivalent to eliminating all breakpoints from π. Since any given reversal is only affecting neighbors at the two endpoints of the reversal, at the most, only two breakpoints can be removed. Therefore the number of reversals necessary to transform to the identity, $r(\pi)$ will be greater than or equal to the number of breakpoints divided by 2, $\frac{b(\pi)}{2}$. So the algorithm will start with our permutation π and continually eliminate breakpoints through reversals until there are none remaining [4]. Two important issues that arise in regards to this algorithm are whether we can always perform a reversal which decreases the number of breakpoints and then how close to optimal this algorithm is. In order to prove that we can continually reduce the number of breakpoints, we start with the case where there is at least one decreasing strip such as (321) or (65). (Note: Strips of length 1 are usually defined as decreasing, except in the case of the first and last

element which are always increasing.) If you find the smallest element k which is in a decreasing strip (which implies it is the last element in that strip), then element k-1 will be the last element in an increasing strip so performing a reversal after the position of k-1 and after the position of k will cause k-1 and k to be joined together thus decreasing the number of breakpoints. However in the case of all increasing strips, we first reverse one of the strips so that we now have at least one decreasing strip and our original logic applies [4]. An example of this is as follows. In the permutation, 0541237689, four is the smallest element in a decreasing strip implying that three is at the end of an increasing strip and therefore performing a reversal between these elements yields 0543217689. Continuing in this way yields 0123457689 and then 0123456789, the identity. We want to find a bound for how optimal this algorithm is. So as we saw in the previous proof, we need at most two reversals to eliminate one breakpoint. Therefore our algorithm will take maximally $2b(\pi)$ to eliminate all the breakpoints. Given an optimal running time of $r(\pi)$, the ratio of interest is $\frac{2b(\pi)}{r(\pi)}$ and because we know that $r(\pi) \geq \frac{b(\pi)}{2}$, we have

$$\frac{2b(\pi)}{r(\pi)} \leq \frac{2b(\pi)}{\frac{b(\pi)}{2}} = 4$$

An approximation ratio of 4 is far from the best algorithm in the field. A great deal of work has been done on determining algorithms which give the minimum number of rearrangements (for a certain subset of rearrangement types) in order to transform one genome into another. The best approximation for rearranging unsigned permutations by reversals is 1.375 [4]. For signed permutations, sorting by reversals can be carried out in $O(n^2)$ time, elaboration in next paragraph [5]. However if we are only interested in calculating the number of reversals necessary to do a transformation without knowing the series of reversals, there exists a linear time algorithm which uses stacks [6].

The $O(n^2)$ algorithm for sorting signed permutations by reversals was constructed by Kaplan et al. Although I will not discuss the algorithm in detail, I will highlight some of the main concepts used in the algorithm which relies heavily

on graph theory. First a breakpoint graph is constructed. The graph has vertices $0, \pi_1, \pi_2 ... \pi_n, n+1$ and the edges are black and gray. Black edges will be (π_i, π_j) such that $|i-j| = 1$ while $|\pi_i - \pi_j| > 1$, in other words adjacent nonconsecutive elements i.e. breakpoints. The gray edges are (π_i, π_j) such that $|\pi_i - \pi_j| = 1$ while $|i-j| > 1$, in other words non adjacent consecutive elements. We define cycles to be those cycles (in the normal sense) which have edges of alternating colors. We define a reversal as being proper if $\Delta b(\pi, \rho) - \Delta c(\pi, \rho) = -1$ where $\Delta b(\pi, \rho)$ is the change in the number of breakpoints after applying the reversal and $\Delta c(\pi, \rho)$ is the change in the number of cycles after applying the reversal. An oriented edges is therefore an edge in which a reversal acting on it is proper; otherwise it is unoriented. An overlap graph, $OV(\pi)$, is created from the breakpoint graph, and the concept of a hurdle is then derived. The goal is to eliminate hurdles in the graph, and safe reversals are those which do not create new hurdles. The algorithm proceeds by first finding the overlap graph and then clearing the hurdles. Then the following step is repeated until π becomes the identity: a safe reversal is performed and π and $OV(\pi)$ are updated [5].

An NP hard problem is one in which any other problem in the class NP (problems with polynomial time verifiers) has a polynomial time reduction to it. It was found that the problem of unsigned permutation rearrangement by reversals is NP-hard [3]. The problem of reversals can be extended to include chromsomes which do not necessarily contain the same set of genes. In this case, we want to determine how many reversals , insertions, and deletions of gene groups need to be performed to transform one signed permutation into the other. An $O(n^2)$ algorithm was found to do this[7].

We can then look at results for rearrangement by transpositions. Algorithms with running time $O(n^2)$ and $O(n^4)$ with an approximation ratio of 1.5 have been developed, but it is not known whether there is a polynomial -time algorithm to solve the transposition problem. Certain restrictions (like on which blocks can be

swapped) can be imposed on transpositions in order to solve the problem or at least to improve the approximation ratio [3].

In more realistic cases, we may want to consider rearrangements by some combination of the given rearrangement types. For example, we can look at calculating rearrangements for transpositions and reversals. Unsigned permutation rearrangments with these two types of operations have approximation ratio 3, and for signed permutations, the approximation ratio is 2 [8]. The true evolutionary distance is the number of transpositions, reversals and inverted transpositions which will be necessary to transform one permutation into another [3].

The issue also arises of weighting different kinds of genomic rearrangements. So far, we have been trying to find the minimum number of genomic rearrangement events to transform one permutation into another. However in finding this minimum number we may want some rearrangement events to count more because they are less likely to occur. Since it has been found that transpositions occur with half the frequency of reversals, we weight them twice as much in order to internalize the lower probability of their occurence. For the problem of calculating the minimum weight series of transpositions, fusions and fissions to transfer between two permutations, an $O(n^2)$ algorithm was found [9].

So far we have not considered algorithms for translocations. For the signed case the translocation problem has been solved with a polynomial $O(n^3)$ as well as a $O(n^2 \log n)$ algorithm [3]. For the unsigned case, a solution has not been found, but there is a polynomial-time approximation algorithm which has ratio 2, and for cases of swapping equal length blocks, runs in $O(n)$ time[10]. If we combine translocations with reversals, fusions, and fissions for the signed permutation case, there exists an $O(n^4)$ algorithm [3].

A complication to the genomic rearrangement problem is to consider the case of having more than one copy of certain genes, in which case we create a string whose characters represent genes on the chromosome. We assume that the number of copies of each gene is the same for both strings. We wish to compute the minimum

number of reversals to transform between two strings. It was proved that sorting strings by reversals is always NP-hard, regardless of the number of different types of genes. The same was proved for sorting strings by block interchanges [11].

Another measure of distance between two genomes is syntenic distance. This measure of distance is quite different from others because we ignore the ordering of the genes on the chromosome and thus just represent each chromosome by an unordered set of genes. A genome is then a collection of chromosomes. Given two chromosomes A and B we can fuse them creating $A \cup B$. A fission of chromosome A results in A_1 and A_2 where $A_1 \cup A_2 = A$. A translocation of chromosomes A and B, where $A_1 \cup A_2 = A$ and $B_1 \cup B_2 = B$ results in $A_1 \cup B_2$ and $B_1 \cup A_2$. The syntenic distance is therefore defined as the fewest number of fusions, fissions, and translocations to transform one genome into another. It was proved that computing the syntenic distance between two genomes is NP-hard [12].

We can now ask ourselves what the purpose is of determining these minimum number of genomic rearrangements and different distances between chromosomes and genomes. Well, the object is to use these numbers as measures of distance when constructing a phylogenetic tree. The phylogenetic tree problem is defined as follows. Given a tree T and l leaves where each leaf is one of l chromosomes, we want to assign a chromosome to each internal node such that the distances between all nodes in the tree are minimized. Formally we want to minimize the weight $w(T) = \sum_{(x,y)\in T} d(x, y)$ where (x, y) is an edge in the tree and d is chosen from the distance metrics. Heuristics have been designed to solve this problem [3].

I would like to propose an interesting extension and evaluation of these algorithms. I think that it would be beneficial to perform a comparison among several of these genomic rearrangement algorithms. Firstly, select a set of algorithms to be tested, and then select several organisms for which the gene order and synteny map is known enabling us to perform the algorithms to determine the series of genome rearrangements. So I would select the first algorithm in my set of algorithms and perform a neighbor joining / clustering algorithm as follows. Find the distance

between all pairs in the set of genomes where distance is defined by the selected algorithm. Then for the two genomes with the least distance between them, join them together, and create an internal node as their common ancestor. This internal node could be some intermediate in the transformation from one to the other. Then recalculate distance between pairs where now the closest two genomes are replaced by their ancestor genome. Keep adding genomes to the tree and adding internal nodes until a full phylogenetic tree is formed. Repeat this process for the other algorithms under consideration. Then compare them to see which algorithm yielded the minimum distances between certain pairs of genomes. Which algorithms yield similar results? Does an algorithm which involves several types of genomic rearrangements yield a tree which is an intermediate of trees by algorithms of each of the genome rearrangements separately? Or does having several different types of rearrangements allow for a much different tree with lower evoluationary distances between genomes?

I would then compare these trees to a standard phylogenetic tree involving these organisms and see which of the algorithm trees best represents the standard tree? I would expect that the trees constructed through the algorithms would have a shorter distance between organisms than would the standard tree. The reason is that our algorithms calculate the minimum number of operations for transformation but while that's interesting computationally, it doesn't necessarily reflect the true number of rearrangements in biology. So it would be interesting to see which tree best reflected the standard tree which is hopefully close to reality. I would imagine that a weighted algorithm involving several different rearrangement types would be close to the standard tree; however it's possible that this may cause a too low estimate for divergence time. I think that two items of interest in comparing trees would be the divergence time between different genomes and the phylogenetic tree architecture. Even if one method predicts a greater divergence time than another, do they still give the same tree structure, in other words, are the same two genomes the closest relatives even if the actual predicted closeness is different? I think

that testing out the algorithms in this or a similar way would be beneficial in our understanding of the usefulness of these algorithms to model the reality of genomic rearrangements.

References

[1] Griffiths, A., Gelbart, W., Lewontin, R., Miller, J., Modern Genetic Analysis. New York: W.H. Freeman and Company, 2002.

[2] *http://scienceblogs.com/pharyngula/upload/2006/06/human-tetraodon$_s$yntenyı$_g$.gif*

[3] Li, Z., Wang, L. Algorithmic Approaches for Genome Rearrangement: A Review. *IEEE Transactions onSystems, Man, and Cybernetics*,36(2006):636-645.

[4] Jones, N., Pevzner, P. An Introduction to Bioinformatics Algorithms. Cambridge, MA: MIT Press, 2004.

[5] Kaplan, H., Shamir,R., Tarjan, R. A Faster and Simpler Algorithm for Sorting Signed Permutations by Reversals. *SIAM Journal on Computing*, 29(2000):880-892.

[6] Bader et al. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Proc. 7th Int. Workshop Algorithms and Data Structures* 2001: 365-376.

[7] El-Mabroul, N., Sankoff, D. Hybridization and genome rearrangement. *Proc. 10th Annu. Symp. Combinatorial Pattern Matching*, (1848)1999: 78-87.

[8] Walter, M., Dias, Z., Meidanis, J. Reversal and Transposition Distance of Linear Chromosomes. *String Processing and Information Retrieval: A south American Symposium*1998: 96-102.

[9] Dias, Z., Meidanis, J. Genome rearrangements distance by fusion, fission, and transposition is easy. *Proc. 8th Int. Symp. String Processing and Information Retrieval*, 2001: 250-253.

[10] Kececioglu, J., Ravi, R. Of Mice and Men: Algorithms for Evolutionary Distances Between Genomes with Translocation *Proc. 6th Annual ACM-SIAM Symp. Discrete Algorithms*, 1995: 604-613.

[11] Christie, D., Irving, R. Sorting strings by reversals and by transpositions. *SIAM journal on discrete math*, 14(2001): 193-206.

[12] DasGupta, B., Jiang, T., Kannan, S., Li, M., Sweedyk, Z. On the complexity and approximation of syntenic distance *Proc. 1st Annual Int. Conf. research in Computational Molecular Biology*, 1997: 99-108.