Efficient sampling of transpositions and inverted transpositions for Bayesian MCMC

István Miklós[1,3], Timothy Brooks Paige[2], and Péter Ligeti[3,4]

[1] eScience Regional Knowledge Center, Eötvös Loránd University
1117 Budapest, Pázmány Péter sétány 1/c, Hungary
miklosi@ramet.elte.hu
[2] Box 786, Amherst College, Amherst MA, 01002 USA
tbpaige@gmail.com
[3] Bioinformatics group, Alfréd Rényi Institute of Mathematics, Hungarian Academy of Sciences
1053 Budapest, Reáltanoda u. 13-15, Hungary
{miklosi,ligeti}@renyi.hu
[4] Department of Computer Science, Eötvös Loránd University
1117 Budapest, Pázmány Péter sétány 1/c, Hungary
turul@cs.elte.hu

Abstract. The evolutionary distance between two organisms can be determined by comparing the order of appearance of orthologous genes in their genomes. Above the numerous parsimony approaches that try to obtain the shortest sequence of rearrangement operations sorting one genome into the other, Bayesian Markov chain Monte Carlo methods have been introduced a few years ago. The computational time for convergence in the Markov chain is the product of the number of needed steps in the Markov chain and the computational time needed to perform one MCMC step. Therefore faster methods for making one MCMC step can reduce the mixing time of an MCMC in terms of computer running time.

We introduce two efficient algorithms for characterizing and sampling transpositions and inverted transpositions for Bayesian MCMC. The first algorithm characterizes the transpositions and inverted transpositions by the number of breakpoints the mutations change in the breakpoint graph, the second algorithm characterizes the mutations by the change in the number of cycles. Both algorithms run in $O(n)$ time, where n is the size of the genome. This is a significant improvement compared with the so far available brute force method with $O(n^3)$ running time and memory usage.

1 Introduction

The differences between the order of genes in two genomes have been used as a measurement of evolutionary distance already more than six decades ago [1]. The rediscovery of inversion distance is dated back to the eighties [2, 3], and since then a large set of papers on optimization methods for genome rearrangement problems has been published. However, except the case of sorting signed permutations by inversions [4–9] or by translocations [10], only approximations [11–15] and heuristics [16] exist. Most of the methods concerning with more types of mutations either penalize all the mutations with the same weight [14], or exclude a whole set of possible mutations due to a special choice of weights [13]. (A nice exception can be found in [17].)

Above the numerous parsimony approaches that try to obtain the shortest sequence of rearrangement operations sorting one genome into the other, Bayesian Markov chain Monte Carlo methods have been introduced a few years ago. They define different models where genomes can evolve by reversals [18–20], reversals and translocations [21] or reversals, transpositions and inverted transpositions [22, 23]. It has been shown that transpositions and inverted transpositions could happen in unichromosomal genomes [24], therefore it is natural to incorporate such events into the Bayesian model. So far the available computer program for the model accommodating transpositions and inverted transpositions used $O(n^3)$ memory and had $O(n^4)$ running time per MCMC step [23]. Though this memory usage and running time allowed the analysis of short genomes (for example, Metazoan mithochondrial genomes), the program suffered memory problems with large genomes containing hundreds of genes.

We introduce two algorithms for characterizing and sampling transpositions and inverted transpositions. The first algorithm characterizes the mutations by the number of breakpoints they remove and samples from a distribution in which breakpoint-removing mutations are preferred. The second algorithm characterizes the mutations by the change in the number of cycles in the graph of desire and reality and samples from a distribution in which cycle-increasing mutations are preferred. Both algorithms run in $O(n)$ time where n is the length of the genome. Since linear running time algorithms for characterizing and sampling reversals have already been developed earlier [24, 23], an MCMC step in the reversals, transpositions and inverted transpositions accommodating model takes only $O(n^2)$ running time (the sampling algorithm might be repeated $O(n)$ times in an MCMC step), and needs only linear memory with these algorithms.

2 Preliminaries

2.1 Mathematical Description of Genome Rearrangement

Genomes are assumed to have the same gene content, and each gene is represented in one copy in both genomes. Gene orders are described as signed permutations, numbers correspond to genes, signs represent the reading direction of genes. Since mutations are actions on the signed permutation group, transforming a genome π_1 to genome π_2 is equivalent with sorting $\pi_2^{-1}\pi_1$ to the identical permutation, and thus, we are going to talk about sorting permutations instead of transforming one into another. By following the convention, a signed permutation of length n is represented as an unsigned permutation of length $2n$, $+i$ is replaced by $2i-1, 2i$, and $-i$ is replaced by $2i, 2i-1$. This unsigned permutation is then framed to 0 and $2i+1$. To properly mimic the signed permutation case, only segments $[2i+1, 2j]$ are allowed to mutate in the unsigned representation.

The graph representation of a signed permutation is called *graph of reality and desire*, whose vertexes are the numbers from 0 to $2n + 1$, and edges are the reality and desire edges. The reality edges connect every second position in the permutation starting with 0. Mutations act on the reality edges; a reversal acts on two reality edges, while a transposition or an inverted transposition on three ones. The desire edges are arcs connecting $2i$ with $2i+1$ for each i. A desire edge is unoriented if it spans even number of points otherwise it is oriented. Since each vertex has a degree of 2, the graph of desire and reality can be unequivocally decomposed into cycles. A reality edge is a breakpoint if its cycle is longer than 2.

The identity permutation has 0 breakpoints and $n + 1$ cycles, all other mutations have more breakpoints and less cycles. Therefore the sorting of a permutation is equivalent with increasing the number of cycles to $n + 1$ or decreasing the number of breakpoints to 0. Mutations can be characterized by the number of breakpoints they remove or the change in the number of cycles. We will talk about e.g. -3-b-transpositions meaning that they remove 3 breakpoints or +1-c-inversions, which increase the number of cycles by 1.

2.2 Stochastic modeling and Bayesian MCMC

Time-continuous Markov models have been the standard approaches for stochastic modeling of molecular evolution. Unlike the case of nucleic acid substitution models, modeling genome rearrangements is computationally demanding and no analytical solutions are known for transition probabilities. What we can calculate is the likelihood of a trajectory, which is the probability that a given sequence of mutations happened in a time span conditional on a set of parameters describing the model [22–24].

To sample trajectories from the posterior distribution, we apply Bayesian Markov chain Monte Carlo (MCMC) [25, 26] which is a random walk on the possible trajectories, and whose stationary distribution is the posterior distribution of trajectories. The random walk is constructed in two steps. In the first step, a new trajectory is drawn from a proposal distribution, and in the second step, the discrepancy between the proposal and the target distribution is corrected by accepting the proposal with probability

$$\min\left\{1\ ,\ \frac{P(X|Y)\pi(Y)}{P(Y|X)\pi(X)}\right\} \tag{1}$$

where P is the proposal distribution, π is the target one, X is the actual state of the chain, and Y is the proposal, and the chain remains in state X with the complement probability [25, 27]. The proposal step replaces a part of the trajectory. The new sub-trajectory is obtained step by step, each mutation is drawn from a distribution that mimics the target distribution we would like to sample from, and the new proposal is independent from the old sub-trajectory.

The mixing of the Markov chain depends on how well the proposal distribution can mimic the target distribution. When proposing a new sub-trajectory step by step, published methods measure the departure of the actual rearrangement from the rearrangement where the sub-trajectory must arrive to, and propose mutations decreasing the measurement of the departure ('good' mutations) with high probability and propose other ones ('bad' mutations) with low probability. This philosophy seems to be essential since random mutations would reach the target rearrangement with a very small probability.

Since there are $3\binom{n+1}{3}$ transpositions and inverted transpositions and $\binom{n+1}{2}$ reversals, an algorithm that spends only constant time with each possible mutation to decide its goodness will already run in $\Omega(n^3)$ time. Therefore it is not a trivial problem how to characterize and sample mutations in less time. Below we show two algorithms characterizing and sampling transpositions and inverted transpositions in linear time, a very simple for breakpoints and a more sophisticated for cycles.

3 Characterizing and sampling transpositions and inverted transpositions

Fig. 1. **a)** and **b)** shows the two decision trees that the below described algorithms use to sample random mutations. At an internal node, a random decision is made only if both subtrees are non-empty. If one of the subtrees is empty, then the algorithm chooses the other subtree with probability 1. For example, on Fig. 1. **a)**, if there is no transposition or inverted transposition decreasing the number of breakpoints by 3, and there is no reversal decreasing the number of breakpoints by two, then there is no random decision at the root of the tree, the algorithm will go to the right subtree with probability 1.

3.1 Sampler based on the change of breakpoints

The first algorithm characterizes the mutations with the number of breakpoints the mutation removes. The algorithm calculates in linear time the number of transpositions and inverted transpositions for each category on Fig. 1. **a)**, namely -3-b, -2-b, -1-b and "rest" mutations, and it is able to sample from a uniform distribution for each category also in linear time.

Preprocessing For each i, we calculate $b(i)$, which is the number of breakpoints after position i; $od(i)$, which is the number of oriented desire edges going from the left end of a reality edge to the right after position i, and $ud(i)$, which is the number of unoriented desire edges going from the left end of a reality edge to the right after position i. These numbers can be trivially calculated by traversing the permutation from right to left.

Counting the mutations For each category and reality edge, we calculate in $O(1)$ time the number of mutations that fall in the given category and their leftmost reality edge is the given edge on which they act. Since there are at most

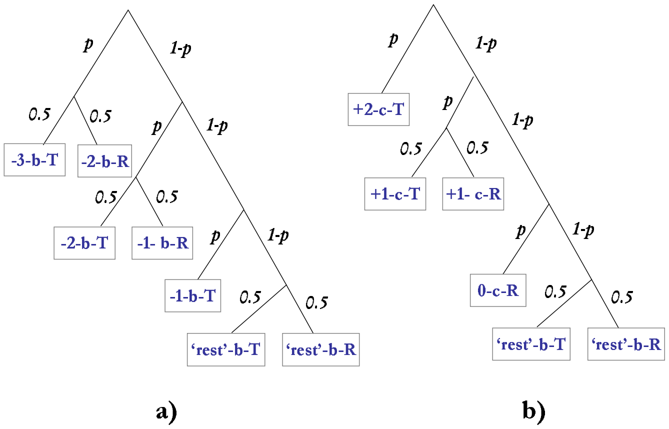


Fig. 1. Decision trees used by the introduced algorithms. T stands for transpositions and inverted transpositions, R stands for reversals. Numbers on the edges means probabilities, p is between 0.5 and 1. In practice, $p = 0.8$ gives a proposal distribution which is reasonably close to the target distribution, acceptance ratio is about $20 - 30\%$.

three such mutations for categories -3-b and -2-b (Fig. 2. **a)-d)**), and these cases can be checked in $O(1)$ time for each reality edge, this is the trivial part of the algorithm.

The maximum number of possible -1-b-transpositions and inverted transpositions is $O(n)$ for each reality edge. These mutations fall into three categories, see Fig 2. **e)**. Having known $b(i)$, $od(i)$ and $ud(i)$ in advance, the number of mutations can be calculated in constant time for each category. For example, for the third case on Fig 2. **e)**, it is $ud(i)$ minus the possible zero, one or two mutations which are actually -2-b- or -3-b-transpositions.

The number of "rest" mutations can be easily calculated if the number of -3-b, -2-b and -1-b-mutations are subtracted from the number of all possible mutations, which is $\binom{n+1-i}{2}$ each for transpositions, inverted transpositions to the right and inverted transpositions to the left.

Sampling from each category Since we know the number of possible mutations for each category and each leftmost reality edges, we first sample the leftmost reality edge from the properly weighted distribution. For -3-b and -2-b-mutations, the number of mutations having a fixed leftmost reality edge is constant, and the algorithm can choose a random one from this constant size set.

To sample from the $O(n)$ possible -1-b-transpositions or inverted transpositions, the algorithm first chooses one of the three possible sub-cases for the selected leftmost reality edge, and depending on the chosen sub-case, it chooses a breakpoint, an oriented or unoriented desire edge that defines the corresponding mutation.

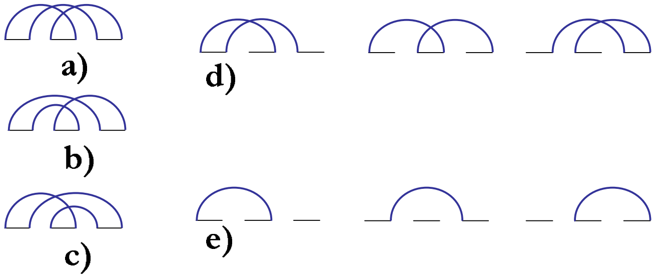


Fig. 2. Configurations on which a mutation can decrease the number of breakpoints. **a)-c):** 3-cycles on which if **a)** a transposition, **b)** an inverted transposition to the left or **c)** an inverted transposition to the right acts, the number of breakpoints decreases by 3. **d)** The three possible cases on which a transposition can decrease the number of breakpoints by two. Similarly for inverted transpositions, there are 3-3 cases derived from the 3-long cycles showed at **b)** and **c)**. **e)** The three possible situation on which a transposition can decrease the number of breakpoints by one. The empty reality edge must be a breakpoint. Similar configurations can be obtained for inverted transpositions.

To sample from the "rest" mutations, the algorithm first chooses a rightmost reality edge after fixing the leftmost reality edge. It calculates the number of "rest" mutations for each possible rightmost reality edge. This is the number of all possible mutations minus the number of -3-b, -2-b and -1-b mutations. The subtracted numbers can be calculated in $O(1)$ time, hence the number of "rest" mutations for each rightmost edge. After this, the algorithm chooses a rightmost edge from the properly weighted distribution, and finally, the algorithm chooses one from the $O(n)$ possible middle reality edges, given the fixed leftmost and rightmost edges.

3.2 Sampler based on the change of cycles

The second algorithm characterizes the mutations by the change in the number of cycles. Though this algorithm does not tell the exact number of mutations falling into a given class, it does tell for each category and for each reality edge whether or not there exists a mutation that falls into the given category and its leftmost edge is the given one. This is enough for using the decision tree on Fig. 1. **b)** and for sampling from a distribution for which the sampling probabilities can be calculated. (We would like to mention for non-experts that the ability of sampling from a distribution does not imply that sampling probabilities can be calculated, see for example [28, 26, 24].)

It is easy to show that cycle-increasing mutations act on one cycle. If three reality edges are in one cycle, they are in one of the eight possible configurations on Table 1. The idea of the algorithm is that for each configuration and reality edge, the algorithm decides whether or not there are other two reality edges to the right being in the given configuration with the third edge. If so, then the reality edge goes to a set from which the algorithm chooses a random leftmost reality edge. Once the algorithm has chosen the mutation type and the leftmost reality edge, it decides for each reality edge on the right hand side of the leftmost edge whether or not it can be together with a rightmost reality edge in a configuration that is good for the given mutation type. After choosing a random middle edge from the ensemble of possible middle edges, the algorithm finally chooses a random good leftmost edge. This method also takes only $O(n)$ time and memory.

Preprocessing The algorithm works on each cycle independently. Starting with the leftmost edge of the cycle, the algorithm traverses the cycle and stores the visiting order of reality edges, as well as the direction of the reality edges on the cycle-traversing. $\pi(i)$ tells the visit order of the reality edge in the ith position, and $pos(i)$ tells the position of the edge which was the ith in the cycle tour and $sign(i)$ tells the direction of the edge.(We will denote by plus sign the left to right direction and by minus sign the right to left direction.) These arrays can be trivially calculated in $O(n)$ time.

After this, the algorithm traverses the reality edges in reverse position order (namely, from right to left), and calculates $s\max(i) = \max_{j \geq i}\{\pi(j)|sign(j) = s\}$ both for positive and negative signs.

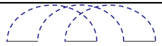
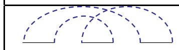
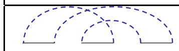
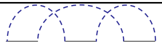
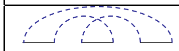
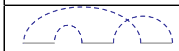
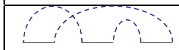
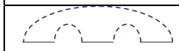
Configuration	**transposition**	**inv. trans. to the left**	**inv. trans. to the right**
	+2-c	+1-c	+1-c
	+1-c	+2-c	“rest”
	+1-c	“rest”	+2-c
	+1-c	“rest”	“rest”
	“rest”	+1-c	+1-c
	“rest”	+1-c	“rest”
	“rest”	“rest”	+1-c
	“rest”	“rest”	“rest”

Table 1. The possible configurations of three reality edges in a cycle and the category of mutations acting on them. Dotted arcs are not necessarily reality edges but alternating paths of reality end desire edges.

Existence of mutations Each configuration on Table 1 can be traversed in six possible ways, see for example on Fig. 3. how the first configuration on

Table 1 can be traversed. Eight configuration times the six possible traversing gives 48 cases, and this is the 3! possible permutations of the visiting order of the three edges multiplied by the 2^3 possible signs of the three edges. Instead of configurations and traversing, we will talk about visiting permutations and signs, there is a one-to-one correspondence between them. Therefore the problem is to tell in constant time for each permutation, sign pattern and reality edge whether or not there are other two reality edges to the right being in the given permutation and sign pattern. Any sign pattern can be discussed in a general way, the three signs will be denoted by s_1, s_2 and s_3 from left to right.

Another observation is that it is enough to give algorithms for the $1, 2, 3$, the $2, 1, 3$ and $1, 3, 2$ permutations since the cycle can be traversed with starting the tour on the leftmost edge in the other direction. This will cause a change in the permutation such that 3 and 1 will be swapped, and all signs will change to the other sign. For example, on Fig. 3. the cases on the right column will turn to the cases on the left column if the cycle is traversed in a reverse order.

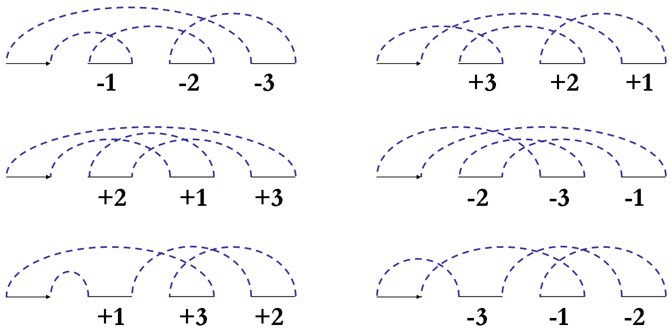


Fig. 3. The possible visiting order of three reality edges on which a transposition increases the number of cycles by two. Dotted arcs are not necessarily reality edges but alternating paths of reality and desire edges.

The $1, 2, 3$ case The $1, 2, 3$ permutation is the easy case for any signs. The algorithm traverses again the reality edges in a reverse position order, and calculates

$$s_2 \max s_3 \max(i) = \max_{j \geq i}\{\pi(j) | \pi(j) < s_3 \max(j) \ \& \ sign(j) = s_2\} \qquad (2)$$

There is a $1, 2, 3$ permutation with a good sign pattern for a position i if $sign(i) = s_1$ and $\pi(i) < s_2 \max s_3 \max(i)$.

The $2, 1, 3$ case The algorithm runs an index i from 1 to n and is in the rightmost position j for which $\pi(j) < i$, $sign(j) = s_2$ and $s_3 \max(j) > i$. If

$pos(i) < j$ and $sign(i) = s_1$, then there is a $2, 1, 3$ case with proper signs starting in position $pos(i)$, otherwise such configuration does not exist in that position. Knowing the $pos()$ and $s_3\max()$ arrays, it is easy to jump to the proper rightmost position until $i > s_3\max(j)$. Then the algorithm must go back to the rightmost position j for which $\pi(j) < i < s_3\max(j)$. Directly traversing back the positions would take $O(n)$ time and such traversing back might be necessary $O(n)$ times, giving the algorithm an $O(n^2)$ running time. Therefore some preprocessing is necessary.

In the preprocessing, the algorithm marks the anchor points of the $s_3\max$ threshold function (rectangles on Fig. 4.). Then for each interval between two consecutive anchor points, it traverses backward the interval, and creates the chained list of the local $s_2\min$ anchor points (black circles on Fig. 4, locality also means that it checks only points which are smaller than the right anchor $s_3\max$ value). For the local minimum, it finds on the previous chained list the first anchor point which is smaller than the actual local minimum, traversing the chain from up to down. The actual list is then augmented with the rest of the list. With up-to-down search, each anchor point is visited only once while searching, providing the $O(n)$ running time of the preprocessing algorithm.

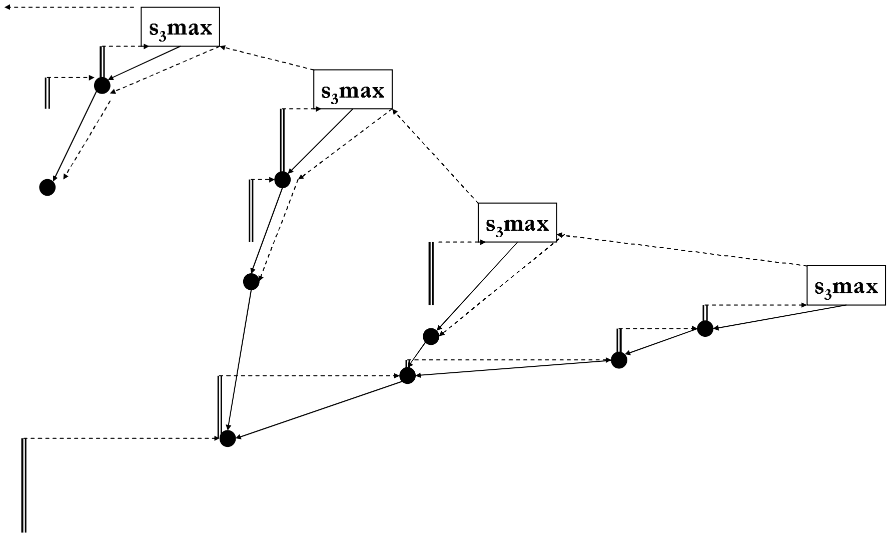


Fig. 4. Explanatory figure for the $2, 1, 3$ algorithm. For details, see the text.

Increasement of i is indicated with a double line on Fig. 4., jumping in positions is indicated with a dashed line. While there is no j for which $\pi(j) < i < s_3\max(j)$ and $sign(j) = s_2$, the algorithm remains in position 1 and marks all $pos(i)$ having no good $2, 1, 3$ configuration. The algorithm jumps positions toward the right end of the permutation whenever a good position j appears, until

$i > s_3 \max(j)$. Then it jumps to the next $s_3 \max$ anchor point to the left, and slides down on the $s_2 \min$ chained list until for the current position j, $\pi(j) < i$. Each edge of the $s_2 \min$ anchor chains is used at most once for back-traversing. To see this, suppose that the algorithm used an edge in a back-traversing, let the value of the starting $s_3 \max$ anchor point be x, and let the starting point of the edge in question be in position j, hence having value $\pi(j)$. Clearly, $\pi(j) < x$, and next time the traceback starts when $i > x$. Although the back-traversing might arrive to position j, it will stop since $i > \pi(j)$. Since the total size of the chained $s_2 \min$ anchor list is $O(n)$, the algorithm spends only $O(n)$ time with back-traversing, and hence, has only $O(n)$ running time altogether.

The $\mathbf{1, 3, 2}$ case For this case, the preprocessing creates a double chained list of the numbers having sign s_3. It traverses the permutation in position order (namely, from left to right) and pulls out the numbers having sign s_3 from the chained list. The preprocessing remembers the neighbours of each number being pulled out, hence it will be possible to put back the numbers in reverse order.

After the preprocessing, the algorithm traverses the permutation in reverse position order (namely, from right to left), and puts back the visited s_3-signed numbers into the chained list. The algorithm remembers the maximal visited number with sign s_3 having to the right a greater number with s_2 sign, denoted by M. Whenever the algorithm arrives to an s_2-signed number, S, it updates M. To do this, it walks on the chained list of s_3-signed numbers till the last number in the chain that is smaller than S. For any s_1-signed number X, there is a $1, 3, 2$ configuration iff $X < M$.

Mutations with leftmost reality edge of position 1, and sampling the middle and rightmost edges The above mentioned algorithms work for the reality edge in position 1, with the notation that the given permutation patterns must be compared with the configurations in Table 1.

Once we choose a rightmost edge in position i and the type of the mutation, deciding whether or not a reality edge can be in a pattern being good for the prescribed mutation is very easy, one should only check the $s_3 \min$ and $s_3 \max$ values with the possible restriction they might not be bigger or smaller than $\pi(i)$, depending on the searched permutation pattern. Similarly, once the rightmost and middle edge have been chosen, it is very easy to find the list of possible leftmost reality edges.

Weighting the reality edges Sampling from the uniform list of possible rightmost edges might lead to a very skewed distribution where mutations on the right ends of cycles are preferred. This is because there might be significantly more mutations of a category with a leftmost reality edge at the left end of a cycle than at the right end of a cycle. Therefore some sophisticated weighting yields better distribution also in terms of acceptance ratios. This statistical issue will be discussed in another paper.

“Rest” mutations We must mention that mutations acting on more than one cycle all fall into the “rest” category. Knowing whether or not there are reality edges being in other cycles, it is trivial to decide whether or not mutations acting on different cycles and having the current reality edges as leftmost edge exists is a trivial problem.

4 Discussion

We introduced two strategies for efficient sampling of transpositions and inverted transpositions. Both algorithms run in $O(n)$ time and memory, and can be used in Bayesian MCMC. With these sampling algorithms, one MCMC step can be performed in $O(n^2)$ time and in linear memory, which is a significant improvement to the so far available algorithm having $O(n^4)$ running time and $O(n^3)$ memory.

We hope we could convince the readers that designing Markov chain Monte Carlo methods in bioinformatics is not only a statistical problem but an at least as important algorithmic problem, too.

Acknowledgments

I.M. was supported by the National Office for Research and Technology at the e-Science Regional Knowledge Centre, Eötvös Loránd University. I.M. is also supported by a Bolyai postdoctoral fellowhip. I.M. and T.B.P. thank for the Rényi Institute for hosting T.B.P. in 2005 summer. P.L. would like to thank for the Rényi Institute for hosting him as a young researcher. Zsuzsanna Márton is thanked for some very useful comments.

References

1. Sturtevant, A.H., Novitski, E.: The homologies of chromosome elements in the genus Drosophila. Genetics **26** (1941) 517–541
2. Nadau, J.H., Taylor, B.A.: Lengths of chromosome segments conserved since divergence of man and mouse. PNAS **81** (1984) 814–818
3. Palmer, J.D., Herbon, L.A.: Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence. J. Mol. Evol. **28** (1988) 87–97
4. Bader, D.A., Moret, B.M.E., Yan, M.: A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. J. Comp. Biol. **8(5)** (2001) 483–491
5. Bergeron, A.: A very elementary presentation of the Hannenhalli-Pevzner theory. In: Proceedings of CPM2001 (2001) 106–117
6. Hannenhalli, S., Pevzner, P.A.: Transforming Cabbage into Turnip: Polynomial Algorithm for Sorting Signed Permutations by Reversals. Journal of ACM **46(1)** (1999) 1–27
7. Kaplan, H., Shamir, R., Tarjan, R.: A faster and simpler algorithm for sorting signed permutations by reversals. SIAM J. Comput. **29(3)** (1999) 880–892

8. Siepel, A.: An algorithm to find all sorting reversals. In: Proceedings of RECOMB2002 (2002) 281–290
9. Tannier, E., Sagot, M.-F.: Sorting by reversals in subquadratic time. In.: Proccedings of the 15th CPM, Lecture Notes in COmputer Science (2004) 1–13.
10. Hannenhalli, S.: Polynomial algorithm for computing translocation distance between genomes. In: Proceedings of CPM1996 (1996) 168–185
11. Bafna, V., Pevzner, A.: Sorting by transpositions. SIAM J. Disc. Math. **11(2)** (1998) 224–240
12. Berman, P., Hannenhalli, S., Karpinski, M.: 1.375-Approximation Algorithm for Sorting by Reversals. In: Proceedings of ESA2002 (2002) 200–210
13. Eriksen, N.: (1+ε)-approximation of sorting by reversals and transpositions. In: Proceedings of WABI2001, LNCS **2149** (2001) 227–237
14. Gu, Q-P., Peng, S., Sudborough, H.I.: A 2-Approximation Algorithm for Genome Rearrangements by Reversals and Transpositions. Theor. Comp. Sci. **210(2)** (1999) 327–339
15. Kececioglu, J.D., Sankoff, D.: Exact and Approximation Algorithms for Sorting by Reversals, with Application to Genome Rearrangement. Algorithmica **13(1/2)** (1995) 180–210
16. Blanchette, M., Kunisawa, T., Sankoff, D: Parametric genome rearrangement. Gene **172** (1996) GC11–GC17
17. Bader, M., Ohlebusch, E.: Sorting by weighted reversals, transpositions and inverted transpositions. Proceedings of RECOMB2006, Lecture Notes in Bioinformatics **3909** (2006) 563–577.
18. Larget, B., Simon, D.L., Kadane, B.J.: Bayesian phylogenetic inference from animal mitochondrial genome arrangements. J. Roy. Stat. Soc. B. **64(4)** 681–695
19. York, T.L., Durrett, R., Nielsen, R.: Bayesian estimation of inversions in the history of two chromosomes. J. Comp. Biol. **9** (2002) 808–818
20. Larget B, Simon DL, Kadane JB, Sweet D.: A Bayesian analysis of metazoan mitochondrial genome arrangements Mol. Biol. Evol. **22(3)** (2005) 486–495
21. Durrett, R., Nielsen, R., York, T.L.: Bayesian estimation of genomic distance. Genetics **166** (2004) 621–629
22. Miklós, I.: MCMC Genome Rearrangement. Bioinformatics **19** (2003) ii130–ii137
23. Miklós, I., Ittzés, P., Hein, J.: ParIS genome rearrangement server. Bioinformatics **21(6)** (2005) 817-820.
24. Miklós, I., Hein, J.: Genome rearrangement in mitochondria and its computational biology. Proceedings of the 2nd RECOMB Satellite Workshop on Computational Genomics, Lecture Notes in Bioinformatics **3388** (2005) 85–96.
25. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equations of state calculations by fast computing machines. J. Chem. Phys. **21(6)** (1953) 1087–1091
26. Liu, J.S.: Monte Carlo strategies in scientific computing. Springer Series in Statistics, New-York. (2001)
27. Hastings, W.K.: Monte Carlo sampling methods using Markov chains and their applications. Biometrika **57(1)** (1970) 97–109
28. von Neumann, J.: Various techniques used in connection with random digits. National Bureau of Standards Applied Mathematics Series **12** (1951) 36–38.