Length and Symmetry on the Sorting by Weighted Inversions Problem

Christian Baudet, Ulisses Dias, Zanoni Dias

Length and Symmetry on the Sorting by Weighted Inversions Problem

Christian Baudet[1], Ulisses Dias[2], and Zanoni Dias[2]

[1] Université Lyon I, INRIA Bamboo Team, France
`christian.baudet@univ-lyon1.fr`
[2] University of Campinas, Institute of Computing, Brazil
{`udias,zanoni`}`@ic.unicamp.br`

Abstract. Large-scale mutational events that occur when stretches of DNA sequence move throughout genomes are called genome rearrangement events. In bacteria, inversions are one of the most frequently observed rearrangements. In some bacterial families, inversions are biased in favor of symmetry as shown by recent research [6, 8, 10]. In addition, several results suggest that short segment inversions are more frequent in the evolution of microbial genomes [4,6,15]. Despite the fact that symmetry and length of the reversed segments seem very important, they have not been considered together in any problem in the genome rearrangement field. Here, we define the problem of sorting genomes (or permutations) using inversions whose costs are assigned based on their lengths and asymmetries. We present five procedures and we assess these procedure performances on small sized permutations. The ideas presented in this paper provide insights to solve the problem and set the stage for a proper theoretical analysis.

1 Introduction

Among various large-scale rearrangement events that have been proposed to date, inversions were established as the main explanation for the genomic divergence in many organisms [6, 8, 11]. An inversion occurs when a chromosome breaks at two locations, and the DNA between those locations is reversed.

In some families of bacteria, an 'X'-pattern is observed when two circular chromosomes are aligned [8, 10]. Inversions symmetric to the origin of replication (meaning that the breakpoints are equally distant from the origin of replication) have been proposed as the primary mechanism that explains the pattern [10]. The justification relies on the fact that one single highly asymmetric inversion affecting a large area of the genome could destroy the 'X'-pattern, although short inversions may still preserve it.

Darlink, Miklós and Ragan [6] studied eight *Yersinia* genomes and added evidence that symmetric inversions are "over-represented" with respect to other types of inversions. They also found that inversions are shorter than expected under a neutral model. In many cases, short inversions affect only a single gene, as observed by Lefebvre *et al.* [12] and Sankoff *et al.* [16], which contrasts with the null hypothesis that the two endpoints of an inversion occur by random and independently.

Despite the importance of symmetry and length of the reversed segment, both have been somewhat overlooked in the genome rearrangement field. Indeed, the most important result regarding inversions is a polynomial time algoritm presented by Hannenhalli and Pevzner [11] that considers an unit cost for each inversion no matter its length or symmetry. When gene orientation is not taken into account, finding the minimum number of inversions that transform one genome into the other is a NP-Hard problem [5].

Some results have considered at least one of the concepts. There is a research line that considers the total sum of the inversion lengths as the objective function of a minimization problem. Several results have been presented both when gene orientation is considered [2, 17] and when it is not [1, 3, 14]. Recently, Arruda *et al.* [2] developed a randomized approach that starts with a scenario with the minimum number of inversions, but allows the number of inversions to increase if the outcome is a reduction in the total sum of the inversion lengths.

Regarding symmetry, the first results were presented by Ohlebusch *et al* [13]. Their algorithm uses symmetric inversions in a restricted setting to compute an ancestral genome and, therefore, is not a generic algorithm to compute the rearrangement distance using only symmetric inversions. In 2012, Dias *et al.* presented an algorithm that considers only symmetric and almost-simmetric inversions [9]. They later included unitary inversions to the problem and provided a randomized heuristic to compute scenarios between two genomes that uses solely these operations [7].

Here we propose a new genome rearrangement problem that combines the concepts of symmetry and length of the reversed segments. Whereas previous works restricted the set of allowed operations by considering only inversions that satisfy constrains like symmetry or almost-symmetry [7, 9], here we allow all possible inversions. The problem we are proposing aims at finding low-cost scenarios between genomes when gene orientation is not taken into account. The results obtained are the first steps in exploring this interesting new problem.

2 Definitions

Formally, a chromosome is represented as a n-tuple whose elements represent genes. If we assume no gene duplication, then this n-tuple is a permutation $\pi = (\pi_1\ \pi_2\ \dots\ \pi_n)$, $1 \leq \pi_i \leq n$ and $\pi_i \neq \pi_j \leftrightarrow i \neq j$. Because we focus on bacterial chromosomes, we assume permutations to be circular, and π_1 is the first gene after the origin of replication.

The inverse of a permutation π is denoted by π^{-1}, for which $\pi^{-1}_{\pi_i} = i$ for all $1 \leq i \leq n$. The composition between two permutations π and σ is similar to function composition in such way that $\pi \cdot \sigma = (\pi_{\sigma_1}\ \pi_{\sigma_2}\ \dots\ \pi_{\sigma_n})$.

An inversion $\rho(i, j)$, $1 \leq i \leq j \leq n$, is a rearrangement that transforms π into $\pi \cdot \rho(i, j) = (\pi_1\ \dots\ \pi_{i-1}\ \underline{\pi_j\ \pi_{j-1}\ \dots\ \pi_{i+1}\ \pi_i}\ \pi_{j+1}\ \dots\ \pi_n)$.

Given two permutations α and σ, the inversion distance $d(\alpha, \sigma)$ is the size t of the minimum sequence of operations $\rho_1, \rho_2, \ldots, \rho_t$ such that $\alpha \cdot \rho_1 \cdot \rho_2 \cdot \ldots \cdot \rho_t = \sigma$.

Let $\iota = (1\ 2\ \ldots\ n)$ be the identity permutation, sorting a permutation $\pi = (\pi_1\ \pi_2\ \ldots\ \pi_n)$ is the process of transforming π into ι and the inversion distance between them is denoted as $d(\pi,\ \iota) = d(\pi)$. Note that $\sigma \cdot \sigma^{-1} = \sigma^{-1} \cdot \sigma = \iota_n$. Therefore, the inversion distance $d(\alpha, \sigma)$ is equivalent to transform a permutation π into a permutation ι if we take $\pi = \sigma^{-1} \cdot \alpha$, because $d(\alpha, \sigma) = d(\sigma^{-1} \cdot \alpha, \sigma^{-1} \cdot \sigma) = d(\pi, \iota) = d(\pi)$. Therefore, we hereafter consider the sorting by inversions problem which aims at finding the sorting distance $d(\pi)$ for an arbitrary permutation π.

The following functions can be applied to identify any element i in the permutation π. **Position**: $p(\pi, i) = k \Leftrightarrow |\pi[k]| = i,\ p(\pi, i) \in \{1, 2, \ldots, n\}$. **Slice**: $slice(\pi, i) = \min\{p(\pi, i), n - p(\pi, i) + 1\}$, $slice(\pi, i) \in \{1, 2, \ldots, \lceil \frac{n}{2} \rceil\}$.

We will now define the cost function for each inversion $\rho(i, j)$ and then we explain two cases that arise. Our cost function is given by $cost(\rho(i, j)) = |slice(\iota, i) - slice(\iota, j)| + 1$. Figure 1 illustrates both cases.

Case 1: $i, j \le \lceil \frac{n}{2} \rceil$ or $i, j \ge \lceil \frac{n}{2} \rceil$.

In this case, the cost function can be simplified to $cost(\rho(i, j)) = abs(i - j) + 1$, which means that it is proportional to the number of elements in the reversed segment. This cost is what one would expect from a length-weighted inversion distance in such a way that larger inversions cost more than short inversions.

Case 2: $i > \lceil \frac{n}{2} \rceil$ and $j < \lceil \frac{n}{2} \rceil$, or $j > \lceil \frac{n}{2} \rceil$ and $i < \lceil \frac{n}{2} \rceil$.

In this case, the cost function is penalizing the asymmetry instead of the number of elements in the reversed segment. In effect, if the inversion $\rho(i, j)$ is perfectly symmetric (meaning that i and j are equally distant from the origin of replication), then the cost is given by $cost(\rho(i, j)) = 1$.

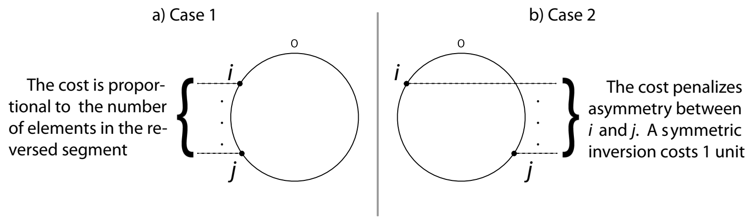
a) Case 1
b) Case 2
o
o
i
j
i
j
The cost is proportional to the number of elements in the reversed segment
The cost penalizes asymmetry between i and j. A symmetric inversion costs 1 unit

Fig. 1. Effect of the cost function when (a) $i, j \le \lceil \frac{n}{2} \rceil$ or $i, j \ge \lceil \frac{n}{2} \rceil$ and (b) $i > \lceil \frac{n}{2} \rceil$ and $j < \lceil \frac{n}{2} \rceil$, or $j > \lceil \frac{n}{2} \rceil$ and $i < \lceil \frac{n}{2} \rceil$.

3 Algorithms

This section presents several algorithms that take advantage from the characteristics of the cost functions.

3.1 Greedy Function

This algorithm uses a greedy function in order to estimate how good an inversion might be. The greedy function uses two concepts: breakpoints and slice-misplaced pairs.

Definition 1. *Breakpoints: let us extend the permutation π to include the elements $\pi_0 = 0$ and $\pi_{n+1} = n + 1$. The pair π_i and π_{i+1}, for $0 \leq i \leq n$, is a breakpoint if $|\pi_{i+1} - \pi_i| \neq 1$. We use $b(\pi)$ to represent the number of breakpoints in a permutation and $\Delta_b(\pi, \rho) = b(\pi \cdot \rho) - b(\pi)$ to represent the variation in the number of breakpoints caused by an inversion ρ.*

Definition 2. *Slice-Misplaced pairs: let π_i, π_j be two elements in π such that $slice(\pi, \pi_i) > slice(\pi, \pi_j)$. We say that π_i, π_j corresponds to a slice-misplaced pair if $slice(\iota, \pi_i) < slice(\iota, \pi_j)$. We use $m(\pi)$ to represent the number of slice-misplaced pairs in π and $\Delta_m(\pi, \rho) = m(\pi \cdot \rho) - m(\pi)$ to represent the variation in the number of misplaced-pairs caused by an inversion ρ.*

The identity permutation ι is the only permutation with no breakpoints. Therefore, an inversion that decreases the number of breakpoints indirectly leads up to the identity permutation. The identity permutation also has no misplaced pairs, so we will create a function that combines both concepts in order to estimate how good an inversion is.

Since an inversion can affect only two breakpoints, we know that $\Delta_b(\pi, \rho) \in \{-2, -1, 0, 1, 2\}$. The variation in the number of slice-misplaced pairs Δ_m is not well-behaved like Δ_b. Thus, we decided to favour breakpoints reduction in our greedy function: $h(\pi, \rho) = \Delta_b(\pi, \rho) + \frac{\Delta_m(\pi, \rho)}{n^2}$. Therefore, the benefit of an inversion is given by $\delta(\pi, \rho) = \frac{h(\pi, \rho)}{cost(\rho)}$. We construct a sequence of inversions that sorts π by iteratively adding an inversion with the best benefit among all possible inversions.

3.2 Left or Right Heuristic

We first divide the elements in π in two groups. The first group refers to the elements that are in slices classified as `sorted` and the second group comprises those elements that are in `unsorted` slices. A slice s is in the `sorted` group if $p(\pi, s) = s$, $p(\pi, n - s + 1) = n - s + 1$, and the slices $\{1, 2, \ldots, s - 1\}$ are also in the `sorted` group. Otherwise, s is in the `unsorted` group.

First, the *Left or Right* heuristic selects the least slice in the `unsorted` group. Then, we determine the element that should be moved first to that slice: the left or the right. The left (right) side is composed of the elements which are in positions that have indices bigger (lower) than the middle position.

To make this choice, we use an auxiliary function f_1 which determines the total weight to put a given element in its right place. The function considers only inversions ρ such that $cost(\rho) \leq 2$. The minimum number of such inversions that is necessary to move the element onto its right position is counted and the weight is computed.

If the slice has only one element that does not belong to it, we just find the element that should be in that slice and perform an inversion to place it closer to its right position. We use inversions ρ whose $cost(\rho) \leq 2$ following the principle of always applying non-expensive movement.

If the slice has two elements that are misplaced, we compute the total weight to place the left element and the total weight to place the right element. Then, we choose the element that has the smallest total weight. In case of tie, we move the right element.

3.3 Lock Heuristic

This heuristic adds a new step that will be called before using the *Left or Right* heuristic. We first search for a `lock` inversion, which puts an element directly into its final position in the `unsorted` slice with the least value.

We also consider the possibility of `indirect lock` inversions that put the element in its final position in two steps: first, it moves the element to its final slice, but in the opposite side. Then, it moves the element to its right place with a perfectly symmetric inversion. In the case where the opposite side is already `locked` (opposite side has already the right element), we consider moving the element one slice above and then applying an almost-symmetric inversion.

An auxiliary function f_2 is used to evaluate the inversions. This function takes the elements π_i and π_j that are in the endpoints of the inversion $\rho(i, j)$ and uses the function f_1 to compute the total weight to put them in their final positions. We compute the total weight before and after applying the inversions and compute the gain. If the gain is positive (*i.e.*, the total weight decreased) and this gain is equal to or bigger than the inversion weight, $\rho(i, j)$ is considered a valid inversion.

The heuristic evaluates the inversions for placing the right and left elements, and then applies the inversion with least cost. If a lock inversion does not exist, then the *Left or Right* heuristic is used.

3.4 Reduce Heuristic

This heuristic computes all possible inversions that remove at least one breakpoint and is considered valid by the function f_2 as described in Section 3.3. We filter the inversions having minimum weight according to f_2, and if more than one inversion remains, we select one that removes two breakpoints. If no inversion reduces the number of breakpoints, we apply the *Left or Right* heuristic (Section 3.2). This new heuristic is called *Reduce* heuristic.

It is also possible to combine the *Reduce* heuristic with the *lock* heuristic to generate other approach called *Lock or Reduce*. This new approach searches for a valid lock inversion ρ_1 and for a valid inversion ρ_2 which reduces the number of breakpoints. If just one of ρ_1 or ρ_2 exists, the heuristic simply applies it. If both exist, the heuristic selects the one that has the smallest weight (in case of tie, apply ρ_1). Finally, if none exists, the *Left or Right* heuristic is used.

4 Experimental Results

We generated a dataset with all possible instance π, such that $|\pi| \leq 12$. Since the permutations in the dataset are small, we were able to compute a minimum cost solution for comparison purposes. The minimum cost solution was calculated using a graph structure G_n, for $n \in \{1, 2, \ldots, 12\}$. We define G_n as follows. A permutation π is a vertex in G_n iff π has n elements. Let π and σ be two vertices in G_n, we build an edge from (π, σ) iff there is an inversion ρ that transforms π into σ. The weight assigned to this edge is $cost(\rho)$. Finally, we calculate the shortest path from ι to each vertex in G_n using a variant of Dijkstra's algorithm for the single-source shortest-paths problem. This variant gives us the minimum cost to sort permutations in G_n, as well as an optimum scenario of inversions.

Let $heuristic\_cost(\pi)$ be the cost for a sorting sequence of π and $cost(\pi)$ be the optimum cost, we can compute the approximation ratio as $\frac{heuristic\_cost(\pi)}{cost(\pi)}$. The first graph in Figure 2 shows how often each heuristic returns the optimum cost. The second and third graphs exhibit, respectively, the average and maximum ratios observed among permutations of same size.

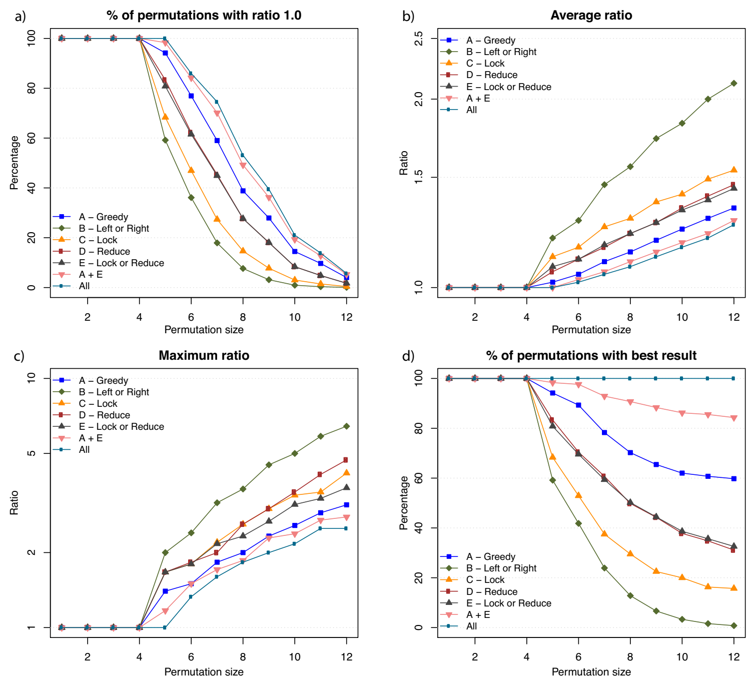

Fig. 2. In (a) we show how often each heuristic returns a minimum cost solution. In (b) and (c) we show the average and the maximum ratio, respectively. In (d) we show how often each heuristic succeeds in providing the best answer among all the heuristics.

The graphs in figures 2(a), 2(b) and 2(c) are maximum or average values. Thus, they may not answer the question: for a single instance π, is there any algorithm that is likely to provide the best answer? The fourth graph discuss this question by assessing the number of times each algorithm provides the least costly sequence. We observe that *Greedy* leads to the best results. In fact, this heuristic is consistently better than the other heuristics in every aspects we plot in Figure 2. The heuristics *Reduce* and *Lock or Reduce* are very similar when we consider how often each one returns the optimum cost and the average ratio. However, the *Lock or Reduce* heuristic seems more appropriate for those permutations that are hard to optimize, since the maximum ratio curve for *Lock or Reduce* (shown in the third graph) is significantly better than that for *Reduce*.

We added new curves to see if one could take advantage of using more than one heuristic other than *Greedy*. The curve labeled as `A+E` selects for each instance the less costly result between those produced by the *Greedy* heuristic and the *Lock or Reduce* heuristic. As we can see, using both heuristics is consistently better than using solely the *Greedy* heuristic in every aspect we are studying.

A final test checks if any profit is gained from running all possible heuristic, which is what we consider in the curve `All`. The increase in performance produced by `All` is consistent when compared to `A+E`. In every graph, the `All` curves are far from the others, which leads to the conclusion that using a combination of more than one heuristic accomplishes satisfactory results.

5 Conclusions

In this work, we have defined a new genome rearrangement problem based on the concepts of symmetry and length of the reversed segments in order to assign a non-unit cost for each inversion. The problem we are proposing aims at finding low-cost scenarios between genomes when gene orientation is not taken into account. We have provided the first steps in exploring this problem.

We presented five heuristics and we assessed their performances on small sized permutations. The ideas used in order to develop these heuristics together with the experimental results set the stage for a proper theoretical analysis.

As in other inversion sorting problems, we would like to know the complexity of determining the distance between any two genomes using only the operations we defined. That seems to be a difficult problem that we intend to keep studying. We plan to design approximation algorithms and more effective heuristics.

6 Acknowledgments

This work was supported by a Postdoctoral Fellowship from FAPESP to UD (number 2012/01584-3), by project fundings from CNPq to ZD (numbers 306730/2012-0, 477692/2012-5 and 483370/2013-4), and by French Project ANR MIRI BLAN08-1335497 and the ERC Advanced Grant SISYPHE to CB. The authors also thank the Center for Computational Engineering and Sciences at Unicamp for financial support through the FAPESP/CEPID Grant 2013/08293-7.

References

1. T. S. Arruda, U. Dias, and Z. Dias. Heuristics for the sorting by length-weighted inversion problem. In *Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics*, pages 498–507, 2013.
2. T. S. Arruda, U. Dias, and Z. Dias. Heuristics for the sorting by length-weighted inversions problem on signed permutations. In *Algorithms for Computational Biology*, volume 8542 of *Lecture Notes in Computer Science*, pages 59–70. Springer International Publishing, 2014.
3. M. A. Bender, D. Ge, S. He, H. Hu, R. Y. Pinter, S. Skiena, and F. Swidan. Improved bounds on sorting by length-weighted reversals. *Journal of Computer and System Sciences*, 74(5):744 – 774, 2008.
4. M. Blanchette, T. Kunisawa, and D. Sankoff. Parametric genome rearrangement. *Gene*, 172(1):C11–17, Jun 1996.
5. A. Caprara. Sorting permutations by reversals and Eulerian cycle decompositions. *SIAM Journal on Discrete Mathematics*, 12(1):91–110, 1999.
6. A. E. Darling, I. Miklós, and M. A. Ragan. Dynamics of genome rearrangement in bacterial populations. *PLoS Genetics*, 4(7):e1000128, 2008.
7. U. Dias, C. Baudet, and Z. Dias. Greedy randomized search procedure to sort genomes using symmetric, almost-symmetric and unitary inversions. In *Proceedings of the International Conference on Bioinformatics, Computational Biology and Biomedical Informatics*, BCB'13, pages 181–190, New York, NY, USA, 2013. ACM.
8. U. Dias, Z. Dias, and J. C. Setubal. A simulation tool for the study of symmetric inversions in bacterial genomes. In *Comparative Genomics*, volume 6398 of *Lecture Notes in Computer Science*, pages 240–251. Springer-Verlag, 2011.
9. Z. Dias, U. Dias, J. C. Setubal, and L. S. Heath. Sorting genomes using almost-symmetric inversions. In *Proceedings of the 27th Symposium On Applied Computing (SAC'2012)*, pages 1–7, Riva del Garda, Italy, 2012.
10. J. A. Eisen, J. F. Heidelberg, O. White, and S. L. Salzberg. Evidence for symmetric chromosomal inversions around the replication origin in bacteria. *Genome Biology*, 1(6):research0011.1–0011.9, 2000.
11. S. Hannenhalli and P. A. Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM*, 46(1):1–27, 1999.
12. J. F. Lefebvre, N. El-Mabrouk, E. Tillier, and D. Sankoff. Detection and validation of single gene inversions. *Bioinformatics*, 19 Suppl 1:i190–196, 2003.
13. E. Ohlebusch, M. I. Abouelhoda, K. Hockel, and J. Stallkamp. The median problem for the reversal distance in circular bacterial genomes. In *Proceedings of Combinatorial Pattern Matching*, pages 116–127, 2005.
14. R. Y. Pinter and S. Skiena. Genomic sorting with length-weighted reversals. *Genome Informatics*, 13:2002, 2002.
15. D. Sankoff. Short inversions and conserved gene cluster. *Bioinformatics*, 18(10):1305–1308, 2002.
16. D. Sankoff, J. F. Lefebvre, E. Tillier, A. Maler, and N. El-Mabrouk. The distribution of inversion lengths in bacteria. In *Comparative Genomics*, volume 3388 of *Lecture Notes in Computer Science*, pages 97–108. Springer Berlin Heidelberg, 2005.
17. F. Swidan, M. Bender, D. Ge, S. He, H. Hu, and R. Pinter. Sorting by length-weighted reversals: Dealing with signs and circularity. In *Combinatorial Pattern Matching*, volume 3109 of *Lecture Notes in Computer Science*, pages 32–46. Springer Berlin Heidelberg, 2004.