Heuristics for the Sorting Signed Permutations by Reversals and Transpositions Problem

Klairton Lima Brito[1(✉)], Andre Rodrigues Oliveira[1], Ulisses Dias[2], and Zanoni Dias[1]

[1] Institute of Computing, University of Campinas, Albert Einstein 1251, Campinas, Brazil
{klairton,andrero,zanoni}@ic.unicamp.br
[2] School of Technology, University of Campinas, Paschoal Marmo 1888, Limeira, Brazil
ulisses@ft.unicamp.br

Abstract. We present two heuristics, *Sliding Window* and *Look Ahead*, to improve solutions for the Sorting Signed Permutations by Reversals and Transpositions Problem. To assess the heuristics, we implemented algorithms described in the literature to provide initial solutions. Despite the fact that we addressed a specific problem, both heuristics can be applied to many others within the area of genome rearrangement. When time is a crucial factor, *Sliding Window* is a better choice because it runs in linear time and improves the initial solutions in 76.4% of cases. If the quality of the solution is a priority, *Look Ahead* should be preferred because it improves the initial solutions in 97.6% of cases, but it runs in $\mathcal{O}(n^3 \times alg(n))$, where $alg(n)$ is the complexity of the algorithm given as input. By using these heuristics one may find a good tradeoff between running time and solution improvement.

Keywords: Genome rearrangement · Heuristics · Reversals Transpositions

1 Introduction

Genome rearrangements affect large portions of the DNA sequence. They occur when chromosomes break at one or more locations and the pieces are reassembled in a different order. Due to the *Principle of Maximum Parsimony*, we approximate the evolutionary distance by the minimum number of events that transforms one genome into another. A Genome Rearrangement Problem aims at finding this minimum number, the so-called *rearrangement distance*.

Assuming no duplicated genes, we assign numbers to each gene to represent genomes as permutations of integers. If we know the relative orientation of the genes, we associate a sign (positive or negative) to each element of the permutation, resulting in a *signed permutation*; we omit this sign otherwise, resulting in an *unsigned permutation* (or simply *permutation*).

J. Jansson et al. (Eds.): AlCoB 2018, LNBI 10849, pp. 65–75, 2018.
https://doi.org/10.1007/978-3-319-91938-6_6

A reversal is a genome rearrangement event that inverts a segment of the genome, changing the order and the orientation of genes in this segment. A transposition swaps the position of two consecutive genome segments, keeping the order and the orientation of genes unchanged inside the segments.

To compute the distance between two genomes, we map one to the identity permutation defined as $\iota_n = (+1 \ \ldots \ +n)$ and use gene labels to map the other to an arbitrary permutation α. The goal is to transform α into ι—a sorting problem—using the minimum number of genome rearrangement events.

Hannenhalli and Pevzner [10] proved that the Sorting Signed Permutations by Reversals problem can be solved in polynomial time. Caprara [4] proved that the unsigned version is NP-hard. Bulteau and coauthors [3] proved that the Sorting Permutations by Transpositions problem is also NP-hard.

Sorting Signed Permutations by Reversals and Transpositions has unknown complexity, the same being true for the unsigned version. The best algorithm for the signed version has an approximation factor of 2 [14]. The best algorithm for the unsigned has an approximation factor of $2k$ [11], where k is the approximation factor of the algorithm used for cycle decomposition [5].

In this work, we present two heuristics to improve solutions from existing algorithms. Our heuristics produce smaller sorting sequences in the vast majority of cases when compared to those provided by the algorithms with no heuristics applied.

The paper is organized as follows. Section 2 presents notations and definitions. Section 3 details the heuristics. Section 4 shows the algorithms used to evaluate our heuristics. Section 5 reports the experiments. Section 6 concludes the manuscript.

2 Preliminaries

In genome rearrangement problems, we represent a genome as an n-tuple, where each element stands for a gene or blocks of genes. Assuming no duplicated genes, the n-tuple is a permutation $\pi = (\pi_1 \pi_2 \pi_3 \ldots \pi_n)$, where $\pi_i \in \{-n, -(n-1), \ldots, -2, -1, +1, +2, \ldots, +(n-1), +n\}$ such that $|\pi_i| \neq |\pi_j| \leftrightarrow i \neq j$. The positive or negative sign of an element indicates the orientation of the gene.

The composition between two permutations $\pi = (\pi_1 \ \pi_2 \ \ldots \ \pi_n)$ and $\sigma = (\sigma_1 \ \sigma_2 \ \ldots \ \sigma_n)$ results in a new permutation: $\alpha = \pi \circ \sigma = (\pi_{\sigma_1} \ \pi_{\sigma_2} \ \ldots \ \pi_{\sigma_n})$. If $\sigma_i < 0$, then $\alpha_i = -\pi_{|\sigma_i|}$, otherwise $\alpha_i = \pi_{\sigma_i}$.

The inverse of σ is a permutation σ^{-1} such that $\sigma \circ \sigma^{-1} = \iota_n$. The inverse σ^{-1} indicates the position and orientation in σ of each element i.

A reversal reverts the order of the segment $\{\pi_i, \pi_{i+1}, ..., \pi_j\}$ and also flips the signs of the elements. Therefore, a reversal $\rho(i, j)$ applied to π leads to $\pi \circ \rho(i, j) = (+\pi_1 \ \ldots \ +\pi_{i-1} \ \underline{-\pi_j \ \ldots \ -\pi_i} \ +\pi_{j+1} \ \ldots \ +\pi_n)$.

A transposition $\tau(i, j, k)$, $1 \leq i < j < k \leq n + 1$, swaps the positions of two adjacent blocks. Therefore, a transposition $\tau(i, j, k)$ applied to π leads to $\pi \circ \tau(i, j, k) = (\pi_1 \ \ldots \ \pi_{i-1} \ \underline{\pi_j \ \ldots \ \pi_{k-1}} \ \underline{\pi_i \ \ldots \ \pi_{j-1}} \ \pi_k \ \ldots \ \pi_n)$.

The *distance* between π and σ, $d(\pi, \sigma)$, is the size of a minimum length sequence $\delta_1, \delta_2, \ldots, \delta_t$ of reversals and transpositions such that $\pi \circ \delta_1 \circ \delta_2 \ldots \delta_t = \sigma$. In this case, $d(\pi, \sigma) = t$.

Let $\iota_n = (+1 \ \ldots \ +n)$ be the identity permutation. A *sorting problem* is the distance between an arbitrary permutation $\alpha = (\alpha_1 \ \ldots \ \alpha_n)$ into ι_n. We denote the distance between α and ι_n by $d(\alpha, \iota_n) = d(\alpha)$.

The sorting problem may appear a particular case of rearrangement distance, but it has the same power of representation. Sorting α is equivalent to transforming π into σ if we consider $\alpha = \pi \circ \sigma^{-1}$. Note that $d(\pi, \sigma) = d(\pi \circ \sigma^{-1}, \sigma \circ \sigma^{-1}) = d(\alpha, \iota_n) = d(\alpha)$.

If we can sort α, we can also transform π into σ using the same sequence of operations. For example, let $\pi = (+6 \ +5 \ +1 \ +2 \ +4 \ +3)$ and $\sigma = (+2 \ -1 \ +4 \ -5 \ +3 \ +6)$, the inverse of σ is $\sigma^{-1} = (-2 \ +1 \ +5 \ +3 \ -4 \ +6)$. We compute $\alpha = \pi \circ \sigma^{-1} = (+6 \ -4 \ -2 \ +1 \ +3 \ +5)$. Applying a sorting sequence in α leads to $\alpha \circ \rho(2, 4) \circ \tau(4, 5, 6) \circ \tau(1, 2, 7) \circ \rho(1, 1) = \iota_6$. Applying the same operations in π leads to $\pi \circ \rho(2, 4) \circ \tau(4, 5, 6) \circ \tau(1, 2, 7) \circ \rho(1, 1) = \sigma$.

We obtain an *extended permutation* from π by inserting two new elements: $\pi_0 = +0$ and $\pi_{n+1} = n + 1$. From now on, unless stated otherwise, permutations will be extended.

A *breakpoint* occurs in a pair π_i and π_{i+1} of π if $\pi_{i+1} - \pi_i \neq 1$, $0 \leq i \leq n$. We denote the number of breakpoints by $b(\pi)$. For $\pi = (+0 \cdot -2 \ -1 \cdot +4 \ +5 \cdot -3 \cdot +6)$, where "$\cdot$" represents a breakpoint, we have $b(\pi) = 4$. The identity permutation ι is the only with no breakpoints.

Breakpoints split a permutation into *strips*, which are maximal intervals without breakpoints. We do not add the elements π_0 and π_{n+1} to the scope of strips. For $\pi = (+0 \cdot -2 \ -1 \cdot +4 \ +5 \cdot -3 \cdot +6)$, we have three strips: $(-2 \ -1)$, $(+4 \ +5)$, and (-3).

Christie [6] created an algorithm to reduce a permutation π into a permutation $\pi_{reduced}$ such that $d(\pi) \leq d(\pi_{reduced})$. Four steps summarize the algorithm: (i) Remove the first strip if it starts with $+1$. (ii) Remove the last strip if it ends with $+n$. (iii) Replace each strip with the smallest element in it. (iv) Renumber the final sequence to obtain a valid permutation.

For example, let $\pi = (+1 \ +2 \ -9 \ -8 \ +5 \ +6 \ +7 \ +3 \ +4)$ be a permutation with four strips: $(+1 \ +2)$, $(-9 \ -8)$, $(+5 \ +6 \ +7)$, and $(+3 \ +4)$. We remove the first strip since it starts with $+1$, resulting in $(-9 \ -8)$, $(+5 \ +6 \ +7)$, and $(+3 \ +4)$. Then, we select the smallest element in each strip: $(-9 \ +5 \ +3)$. Finally, we renumber the final sequence to obtain the reduced permutation: $\pi_{reduced} = (-3 \ +2 \ +1)$.

3 Heuristics

We developed two heuristics, *Sliding Window* and *Look Ahead*, that extend previous approaches applied to unsigned permutations [7,8] and we assess them on the Sorting Signed Permutations by Reversals and Transpositions problem.

3.1 Sliding Window

Sliding Window uses a database that contains optimal sorting sequences for signed permutations of size up to nine [9]. It receives a permutation π and an algorithm alg as input and outputs a sequence of rearrangement events that sorts π.

The heuristic behaves as follows: we use alg to sort π and generate a sequence of permutations $S = [\pi^0, \ldots, \pi^z]$, such that $\pi^i \circ \delta = \pi^{i+1}$, where $\delta \in \{\rho, \tau\}$, for $0 \leq i < z$. The output is a sequence $S' = [\pi^0, \ldots, \pi^y]$, such that $y \leq z$ and $\pi^y = \pi^z$.

Initially, the heuristic picks a subsequence of permutations S^w from S that we call window. The window begins with π^i and ends with π^j, $0 \leq i < j \leq z$. The heuristic computes $\alpha = \pi^i \circ \pi^{j^{-1}}$ and reduces it to $\alpha_{reduced}$. If $\alpha_{reduced}$ has up to nine elements, we retrieve the optimal sorting sequence from the database, otherwise a smaller window S^w will be sought and slided through S.

If the optimal sequence for $\alpha_{reduced}$ is shorter than S^w, we use it to build a sequence $S^{w'}$ that sorts α. Each permutation $\alpha' \in S^{w'}$ is replaced by $\pi^j \circ \alpha'$ and the window S^w is replaced by $S^{w'}$, which improves S. Figure 1 shows a flowchart for this heuristic.

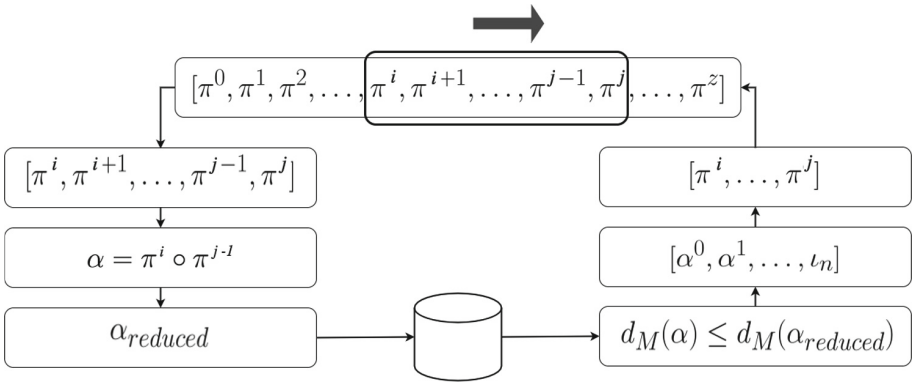

Fig. 1. Flowchart of the *Sliding Window* heuristic.

The heuristic runs in $\mathcal{O}(n + alg(n))$, where $alg(n)$ is the complexity of the algorithm given as input.

3.2 Look Ahead

Look Ahead receives a permutation π and an algorithm alg as input, and outputs a sequence of events that sorts π. The heuristic behaves as follows: we start with the permutation π as the current permutation. While the current permutation is not sorted, the heuristic assess all possible reversals and transpositions, fully investigating the neighborhood of π.

We use alg to estimate the distance of each permutation in the neighborhood of π, and we select the permutation with the shortest distance (or one of the shortest if multiple choices are available). The selected permutation will be the current permutation in the next iterative step. The process ends when we reach the identity.

Look Ahead requires a distance estimator alg to select an operation at each step. If the estimator does not work well, it negatively impacts the solution provided.

The heuristic runs in $\mathcal{O}(n^3 \times alg(n))$, where $alg(n)$ is the complexity of the algorithm given as input. Since the complexity of this heuristic is directly linked to $alg(n)$, it becomes prohibitive in cases where $alg(n)$ has a high complexity. Figure 2 shows the flowchart for this heuristic.

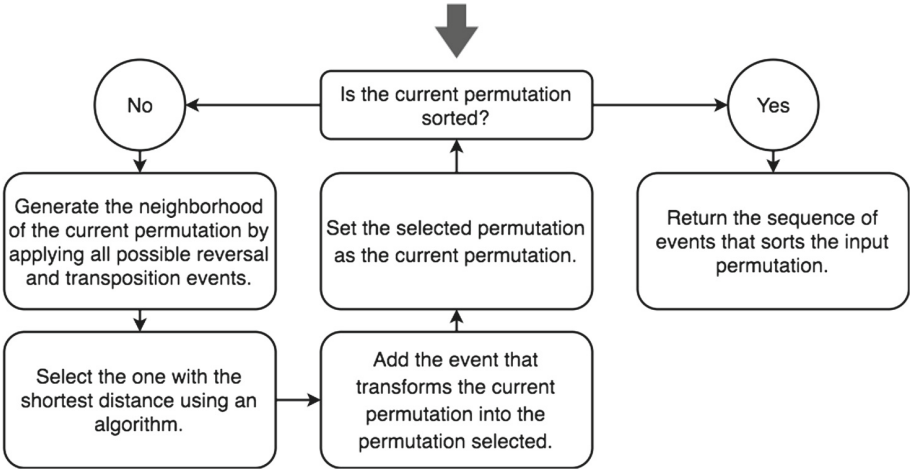

Fig. 2. Flowchart of the *Look Ahead* heuristic.

4 Algorithms Implemented to Evaluate the Heuristics

We use as input different algorithms from the literature. Some were not designed for the Sorting Signed Permutations by Reversions and Transpositions problem, but they provide a valid solution or some modifications were performed to make it valid. We employed such algorithms to verify the behavior on various situations. Table 1 shows the algorithms used as input.

Table 1. Algorithms used to evaluate the heuristics.

Rearrangement problem	Code	Reference	Time	Ratio
Reversal and transposition	RSH	Rahman *et al.* [11]	$\mathcal{O}(n^3)$	$2k$
Signed reversal and transposition	WDM	Walter *et al.* [14]	$\mathcal{O}(n^3)$	2
	BRPT	Walter *et al.* [14]	$\mathcal{O}(n^2)$	3
	BRPR	Walter *et al.* [14]	$\mathcal{O}(n^2)$	3
Signed reversal	HPB	Hannenhalli and Pevzner [10]	$\mathcal{O}(n^2)$	1
		Bader *et al.* [1]	$\mathcal{O}(n)$	1
Transposition	BP	Bafna and Pevzner [2]	$\mathcal{O}(n^2)$	1.5

- RSH: An algorithm for the Sorting Unsigned Permutations by Reversals and Transpositions problem with an approximation factor of $2k$, where k is the approximation of the algorithm that decomposes π in cycles. If applied on signed permutations, it outputs valid solution with approximation factor of 2.
- WDM: An algorithm for the Sorting Signed Permutations by Reversals and Transpositions problem that guarantees an approximation factor of 2.
- BRPT: An algorithm for the Sorting Signed Permutations by Reversals and Transpositions problem with an approximation factor of 3. The algorithm greedly removes the largest number of breakpoints. In case of ties between reversals and transpositions, a transposition is chosen.
- BRPR: A variation of BRPT that favours reversals instead of transpositions.
- HPB: An exact algorithm for the Sorting Signed Permutations by Reversals problem. Since *Look Ahead* needs a distance estimation, we used a linear time algorithm that outputs only the distance. Since *Sliding Window* requires an initial sequence of rearrangement events, we used a quadratic algorithm. The implementations were provided by Tesler [12,13].
- BP: An approximation algorithm for the Sorting Unsigned Permutations by Transpositions problem with an approximation factor of $\frac{3}{2}$. To ensure a valid solution for the Sorting Signed Permutations by Reversals and Transpositions problem, we first reverse all negative strips before applying this algorithm. The final sorting sequence is composed by the reversal operations that were first applied and the result of this algorithm.

5 Results

The heuristics and the algorithms implemented from literature received the same set of permutations that were randomly generated with the maximum number of breakpoints. The sizes of permutations ranged from 10 to 500 and increased in intervals of 10 from 10 to 100, and in intervals of 50 from 150 up to 500. For each size, we created a set of 1000 permutations. We executed *Look Ahead* on permutations with size up to 100 due to the slow running time. We executed *Sliding Window* on all permutations.

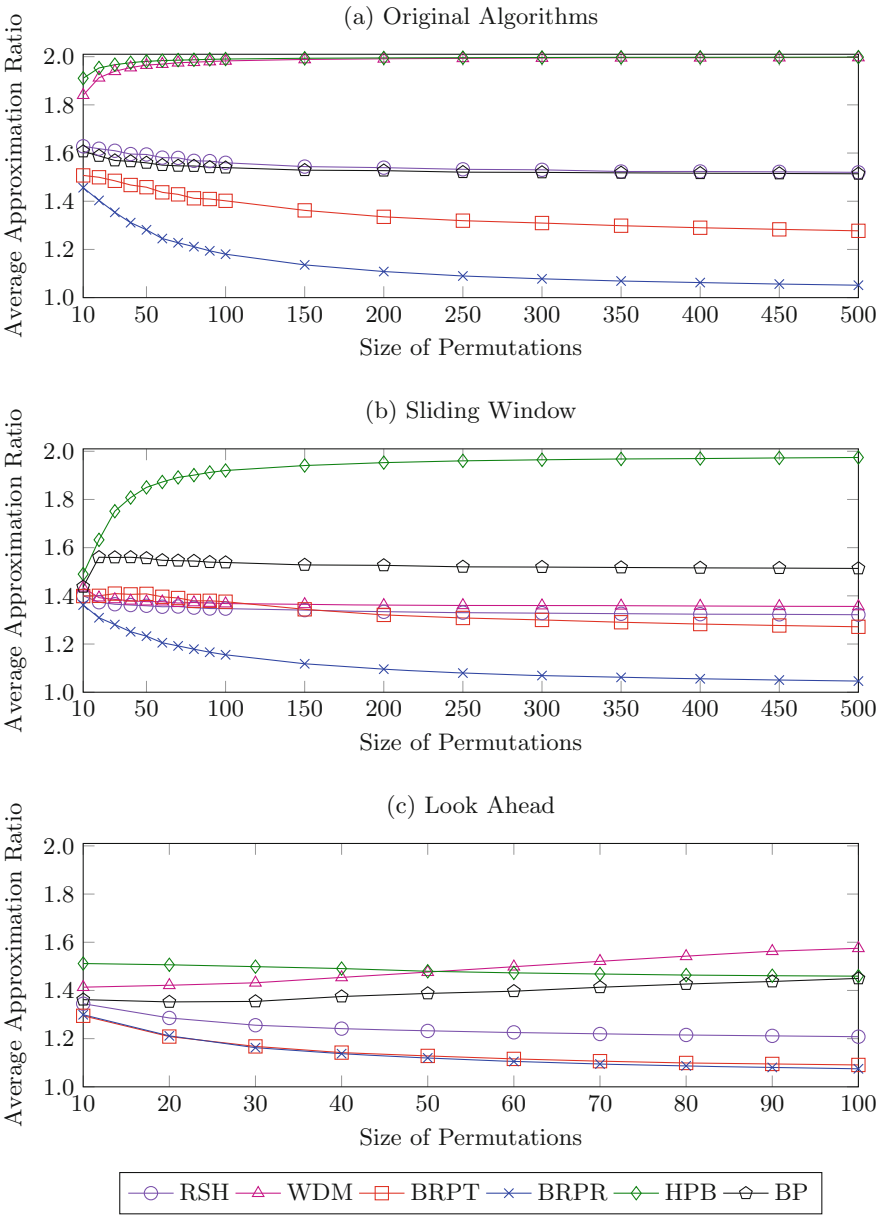

Fig. 3. Average approximation factor of each (a) original algorithm, (b) *Sliding Window*, and (c) *Look Ahead*. We can see a significant improvement in the average approximation factor in almost all the algorithms where *Sliding Window* was applied. *Look Ahead* improved the average approximation factor of all algorithms.

To compute the approximation factors, we used the lower bound $\lceil \frac{(n+1)-c(\pi)}{2} \rceil$, where $c(\pi)$ is the number of cycles in the cycle graph [14].

Figure 3 shows the average approximation factor of the original algorithms and our heuristics. Comparing the Fig. 3(a) and (b) we observe improvement in the average approximation factor in almost all the algorithms provided by *Sliding Window*. We make similar comparison with Fig. 3(a) and (c) and see a significant reduction in average approximation factor in all algorithms using *Look Ahead*. In most cases, the results provided by *Look Ahead* showed better performance than those provided by *Sliding Window*, except for the case where the WDM algorithm was used.

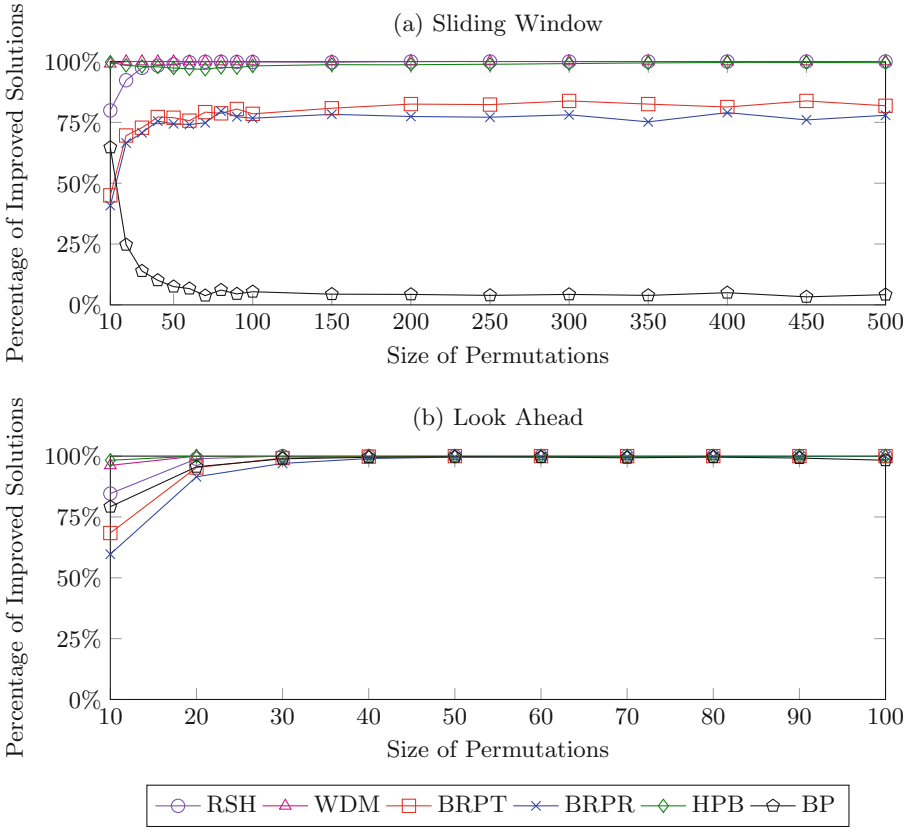

Fig. 4. Percentage of sorting sequences that have been improved using (a) *Sliding Window* and (b) *Look Ahead*. We can see that in almost all cases *Sliding Window* improved the initial sorting sequence of a significant amount of permutations. The only case in which this behavior was not observed was when we used the BP algorithm. *Look Ahead* improved the initial sorting sequence of a significant amount of permutations. For all permutations with size greater than 10, this value exceeded 90%.

Figure 4 shows the percentage of permutations where the original sorting sequence was improved by Sliding Window and *Look Ahead*, respectively.

Applying *Sliding Window* in permutations of size 10 up to 100, we obtained an improvement in 75.6% of cases, and for permutations of size 150 up to 500, we obtained an improvement in 77.4% of cases. *Look Ahead* was executed with permutations up to size 100 and improved the original sorting sequence in 97.6% of cases. All algorithms presented significant improvements.

Table 2 reports the average running time in seconds. The abbreviations ALG, SW, LA, represents the original algorithm, *Sliding Window*, and *Look Ahead*. We see that *Sliding Window* runs extremely fast, whereas *Look Ahead* is more time-consuming.

Table 2. Average running time in seconds. In all cases, *Sliding Window* outputs a solution in less than 0.1 s. *Look Ahead* is more time-consuming, but it runs fast when an algorithm with low time complexity like HBP is used.

Algorithm	Size of permutations				
	100			500	
	ALG	SW	LA	ALG	SW
RSH	0.003	0.016	11378.694	0.048	0.085
WDM	0.003	0.017	15330.229	0.047	0.092
BRPT	0.001	0.007	1676.322	0.006	0.021
BRPR	0.001	0.007	1445.922	0.005	0.019
HPB	0.003	0.009	72.803	0.029	0.049
BP	0.002	0.005	3941.291	0.025	0.035

Table 3. Average approximation factor provided by the original algorithms and our heuristics. *Look Ahead* significantly improved the average approximation factor of all algorithms. *Sliding Window* showed better results when applied to specific algorithms for the Sorting Signed Permutations by Reversals and Transpositions Problem.

Algorithm	Permutation size				
	100			500	
	ALG	SW	LA	ALG	SW
RSH	1.559	1.347	1.208	1.520	1.322
WDM	1.982	1.368	1.575	1.997	1.356
BRPT	1.402	1.376	1.091	1.277	1.272
BRPR	1.180	1.155	1.075	1.052	1.046
HPB	1.990	1.920	1.459	1.998	1.974
BP	1.540	1.539	1.450	1.514	1.514

Table 3 shows a comparison between the average approximation factor of the original algorithms and our heuristics. The abbreviations ALG, SW, LA, represents the original algorithm, *Sliding Window*, and *Look Ahead*.

6 Conclusion

The heuristics presented in this work significantly improved the sorting sequence provided by several algorithms known in the literature for the Sorting Signed Permutations by Reversals and Transpositions Problem. The heuristics *Sliding Window* and *Look Ahead* improved the sorting sequence in 76.4% and 97.6% of cases, respectively.

These heuristics can be applied in scenarios with different needs. If time is a crucial factor, the *Sliding Window* stands out since it presents good results and suffer less variation in execution time when permutation size increases. If time is not a priority, then *Look Ahead* is a better fit, presenting more remarkable results.

The next step is to use these heuristics on variants of the Sorting Signed Permutations by Reversals and Transpositions problem and check if it is possible to obtain results similar to those shown in this work.

Acknowledgments. The authors acknowledge the support from CAPES, the International Cooperation Program CAPES/COFECUB Foundation under grant 831/15, the CNPq under grants 425340/2016-3, 400487/2016-0, 140466/2018-5, and 138219/2016-8 and also the São Paulo Research Foundation (FAPESP) under grants 2013/08293-7, 2014/19401-8, and 2015/11937-9.

References

1. 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. Comput. Biol. **8**, 483–491 (2001)
2. Bafna, V., Pevzner, P.A.: Sorting by transpositions. SIAM J. Discrete Math. **11**(2), 224–240 (1998)
3. Bulteau, L., Fertin, G., Rusu, I.: Sorting by transpositions is difficult. SIAM J. Comput. **26**(3), 1148–1180 (2012)
4. Caprara, A.: Sorting permutations by reversals and eulerian cycle decompositions. SIAM J. Discrete Math. **12**(1), 91–110 (1999)
5. Chen, X.: On sorting unsigned permutations by double-cut-and-joins. J. Comb. Optim. **25**(3), 339–351 (2013)
6. Christie, D.A.: Genome rearrangement problems. Ph.D. thesis, Department of Computing Science, University of Glasgow (1998)
7. Dias, U., Dias, Z.: Extending Bafna-Pevzner algorithm. In: Proceedings of the 1st International Symposium on Biocomputing (ISB 2010), pp. 1–8. ACM, New York (2010)
8. Dias, U., Galvão, G.R., Lintzmayer, C.N., Dias, Z.: A general heuristic for genome rearrangement problems. J. Bioinform. Comput. Biol. **12**(3), 26 (2014)

9. Galvão, G.R., Dias, Z.: An audit tool for genome rearrangement algorithms. J. Exp. Algorithmics **19**, 1–34 (2014)
10. Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. J. ACM **46**(1), 1–27 (1999)
11. Rahman, A., Shatabda, S., Hasan, M.: An approximation algorithm for sorting by reversals and transpositions. J. Discrete Algorithms **6**(3), 449–457 (2008)
12. Tesler, G.: Efficient algorithms for multichromosomal genome rearrangements. J. Comput. Syst. Sci. **65**(3), 587–609 (2002)
13. Tesler, G.: GRIMM: genome rearrangements web server. Bioinformatics **18**(3), 492–493 (2002)
14. Walter, M.E.M.T., Dias, Z., Meidanis, J.: Reversal and transposition distance of linear chromosomes. In: Proceedings of the 5th International Symposium on String Processing and Information Retrieval (SPIRE 1998), Santa Cruz de La Sierra, Bolivia, pp. 96–102. IEEE Computer Society (1998)