UNIVERSIDADE FEDERAL FLUMINENSE

THIAGO LOPES NASCIMENTO

Paralelismo e Heurística para o Problema da Mediana por Swap

NITERÓI

2023

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

N244p Nascimento, Thiago Lopes
Paralelismo e Heurística para o Problema da Mediana por Swap / Thiago Lopes Nascimento. - 2023.
34 f.: il.

Orientador: Luís Felipe Ignácio Cunha.
Trabalho de Conclusão de Curso (graduação)-Universidade Federal Fluminense, Instituto de Computação, Niterói, 2023.

1. Bioinformática. 2. Rearranjo de genomas. 3. Mediana. 4. Swap. 5. Produção intelectual. I. Cunha, Luís Felipe Ignácio, orientador. II. Universidade Federal Fluminense. Instituto de Computação.III. Título.

CDD - XXX

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

Thiago Lopes Nascimento

Paralelismo e Heurística para o Problema da Mediana por Swap

Trabalho de Conclusão de Curso apresentado ao Departamento de Ciência da Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Bacharel em Computação.

Aprovada em 15 de dezembro de 2023.

BANCA EXAMINADORA

Prof. Luís Felipe Ignácio Cunha - Orientador, UFF

Prof. Fábio Protti, UFF

Prof. Leandro Santiago de Araújo, UFF

Niterói

2023

Paralelismo e Heurística para o Problema da Mediana por SWAP[1]

Algorithms for the swap median *problem and relation to* convexity *problems*

Thiago Lopes Nascimento[2]
Lu s Felipe Ignacio Cunha[3]

Resumo

Rearranjo de genomas sao eventos em que grandes blocos de DNA trocam pedacos durante a evolucao. A analise desse tipo de vento e uma ferramenta para entender evolucao genômica, baseado em encontrar o menor numero de reordenamento que transformam um genoma em outro. De modo geral, sao considerados mais de dois genomas e nos temos novos desa os. Problema da Mediana consiste em, dado três permutacoes e uma metrica de distância, determinar a permutacao *s* que minimize a soma de todas as distâncias entre *s* e cada entrada. O conhecimento da mediana determina ancestralidade comum relativamente perto, sendo uma ferramenta util para reconstrucao de arvores logeneticas. Diversas metricas ja foram consideradas, algumas limitadas, como *breakpoint*, e muitas outras mais gerais em sequências que sao muitas genericas que permutacoes, como *double-cut-and-join* ou *single-cut-and-join* nos DNAs. Nos trabalhamos com o problema da mediana por distância de *swap* em permutacoes. Visto que sua complexidade computacional ainda esta em aberto, nos desenvolvemos e analisamos algoritmos forca bruta (sequencial e paralelo) e heur sticas, e provamos condicoes sobre as permutacoes solucao. Alem disso, nos associamos solucoes mediana e intervalos de conjuntos convexos, proposto por Cunha e Protti, em *Journal of Computational Biology*, 2019. O conceito de grafos convexos motiva as seguintes investigacoes: Uma permutacao mediana pertence ao menor caminho entre cada par das permutacoes de entrada, i.e ela pertence a convexidade geodesica? Todas as permutacoes mediana assim como o fecho geodesico, formam todos os caminhos induzidos entre os três pares de permutacoes da entrada? Baseado nos nossos algoritmos propostos, nos somos capazes de responder essas perguntas.

---

[1] Trabalho de conclusao de curso apresentado ao curso de Bacharelado em Ciência da Computacao da Universidade Federal Fluminense como requisito parcial para conclusao do curso.

[2] Graduando do Curso de Ciência da Computacao - UFF, thiago_nascimento@id.u .br.

[3] Orientador - Instituto de Computacao - UFF, l gnacio@ic.u .br.

Palavras-chaves: Rearranjo de genomas; Mediana; Swap; Permutacoes; Convexidade.

Abstract

Genome rearrangements are events where large blocks of DNA exchange pieces during evolution. The analysis of such events is a tool for understanding evolutionary genomics, based on finding the minimum number of rearrangements to transform one genome into another. In a general scenario, more than two genomes are considered and we have new challenges. Median problem consists of, given three permutations and a distance metric, determining a permutation s that minimizes the sum of the distances between s and each input. The knowledge of the median determines relatively close common ancestry, being a useful tool to reconstruct phylogenetic trees. Several metrics have been considered, some limited, as *breakpoints*, and others more general in sequences that are even more general than permutations, as *double-cut-and-join* or *single-cut-or-join* in DNAs. We deal with median problem over *swap* distances in permutations. Since its computational complexity is open, we develop and evaluate brute force (sequential and parallel) and heuristic algorithms, and prove conditions on permutations solution. In addition, we associate median solutions and interval convex sets, proposed by Cunha and Protti, in *Journal of Computational Biology*, 2019. The concept of graph convexity foments the following investigation: Does a median permutation belong to any shortest path between one of the pairs of input permutations, i.e. does it belong to the geodesic convexity? Do all median permutations along with the geodesic closure, form all induced paths between the three pairs of input permutations? Based on our proposed algorithms, we are able to answer those questions.

Keywords: Genome Rearrangements; Median; Swap; Permutations; Convexity.

Aprovado em: 15/12/2023. Versao Final em: 18/12/2023

1. INTRODUCTION

Ancestral reconstruction is a classic task in comparative genomics. This problem is based on consensus word analysis, with vast applicability (CUNHA *et al.*, 2013; FERTIN *et al.*, 2009; PEVZNER, 2000). Genome rearrangement problems study large-scale mutations on a set of DNAs in living organisms, having been extensively studied in computational biology and computer science fields for decades. From a mathematical point of view, a genome can be represented by as strings or permutations. Watterson et al. (WATTERSON *et al.*, 1982) proposed genome rearrangement problems interpreted as transforming one permutation into another by a

minimum number of operations depending on the possible allowed rearrangements, i.e., the chosen metric. This model considers the following assumptions: the order of genes in each genome is known; all genomes we compare share the same gene set; all genomes contain a single copy of each gene; and all genomes consist of a single chromosome. So, genomes can be modeled as permutations, once each gene is encoded as an integer.

Lot of metrics related to genome rearrangements are studied, having a diverse amount of applications in computational biology (FEIJAO; MEIDANIS, 2011; SANKOFF, 1999; CUNHA; PROTTI, 2019; CUNHA *et al.*, 2020; SILVA *et al.*, 2023). Different metrics are used for the realization of measurements, for example the breakpoint distance, block-interchange distance and swap distance (CUNHA *et al.*, 2013; CUNHA *et al.*, 2020; FEIJAO; MEIDANIS, 2011; XIA *et al.*, 2018; SANKOFF; BLANCHETTE, 1997; SETZER; STEIDL; TEUBER, 2012). In addition to the chosen metric, the way the genomes can be represented must also be considered. In some cases it is better to represent genomes as permutations, as we mentioned above and also considered in this work, though, there are other models to represent genomes with more than one chromosome, using metrics as double-cut-and-join (XIA *et al.*, 2018).

For each of these metrics, it is possible to make a correlation between sets of elements, for example the closest genome problem, that consists in finding a genome that minimizes the maximum distance in relation to the input set based on the chosen metric (POPOV, 2007; CUNHA *et al.*, 2020). It is well known that closest string problem over Hamming distance is NP-complete even for binary strings, and when dealing with permutations, Cunha et al. (CUNHA *et al.*, 2020) settled NP-completeness results on closest permutation regarding breakpoints and block-interchanges.

Graphs can represent genome objects and the relationships between them. Vertices are permutations (or strings), and there exists an edge linking two vertices if the distance between them is equal to one (ERIKSSON *et al.*, 2001; KONSTANTINOVA, 2008). Graph convexity is a rich field in which several combinatorial problems and convexity types can be defined according to the application at hand (PENSO *et al.*, 2015). In general, we deal with sets of vertices to be increased step-by-step by following some prescribed rule, until the process stabilizes into a *convex set* (the \closure"). Cunha and Protti (CUNHA; PROTTI, 2019) proposed an association between graph convexity and genome rearrangement problems. This concept is viewed as permutations that could be generated by a given input set. The authors proved that the closure of a given set of strings can be determined in polynomial time when dealing with Hamming distances, whereas considering swap distance over permutations they could only certify in polynomial time if a given set is already a closure itself.

Another relevant problem over a given input set is the Median problem, with many approaches in regard to nding ancestral genomes (PE'ER; SHAMIR, 1998; CAPRARA, 2003; BADER, 2011; HAGHIGHI; SANKOFF, 2012; CUNHA; PROTTI, 2019; CUNHA *et al.*, 2020). Given a set of three genomes and a metric, the goal is to nd a solution genome that minimizes the sum of the distances between the solution and all the input set. Our study has a focus on the swap median problem over permutations.

Permutations are sequences of distinct integers from 1 to n and a *swap operation* is an exchange between two elements on the permutation. Given two permutations with the same number of elements, the *swap distance* is the smallest number of swaps that transform one permutation into the other. The swap distance problem is solved in polynomial time, being obtained by the number of selection sort operations (CORMEN *et al.*, 2022).

Regarding the swap median problem, we are given three permutations of the same size and we want to obtain the solution permutation s that minimizes the sum of the swap distance between s and each of the input. The computational complexity of the swap median problem is still open. With this in mind, we present the following contributions:

- We develop and analyze speedups with respect to sequential and parallel brute force implementations we propose.
- We elaborate heuristics and evaluated their quality by using the brute force implementations.
- We prove necessary conditions of median solutions of a given set of input permutations.
- We relate convexity problems and swap median problem, which is equivalent to the Steiner convexity, as we discuss later.

This work is organized as follows: next, we present biological de nitions necessary for a better understanding of median problems; In Section 2, we show some preliminary results concerning the swap median problem and de nitions on graph convexity; In Section 3, we develop the brute force solutions, considering both sequential and parallel approaches; In Section 4, we develop e cient heuristics based on some properties that guarantee necessary conditions on the solution medians; In Section 5, we discuss the relation between median problem and convexity problems; In Section 6, we present speedups, quality of heuristics and how those results yield in guaranteeing properties among median and closure solutions; Section 7 presents a conclusion for further investigation.

Also note that a portion of this work was presented at SBPO 2023, Simposio Brasileiro de Pesquisa Operational, and also is under revision for the JCB, Journal of Computational Biology.

2. PRELIMINARIES

An *alphabet* is a non empty set of letters, and a *string* over is a - nite sequence of letter of . A *permutation* of size n is a particular *string* with a single occurrence of each element from 1 to n, considering that the permutation is a bijection of the set $f1;2;:::;ng$ with itself. Hence, we have the permutation = [(1) (2) (n)]. Given a metric M, two permutations and , $d_M(\ ;\ )$ is the distance between the permutations and in relation to the metric M. Since we can relabel to transform it to the *identity permutation* = $[1;2;\ ;n]$ with the operation ^{1} = , we have $d_M(\ ^{1};\ )$. Therefore, we can calculate the distance between two permutations with one of them being , denoting $d_M(\ ;\ )$ simply by $d_M(\ )$.

Swap distance Permutations can also be represented by each element followed by its image. For example, given a set $f1;2;3g$, the sequence (1 2 3) maps 1 into 2, 2 into 3, and 3 into 1, corresponding to the permutation [2 3 1]. This representation is not unique; for instance, (2 3 1) and (3 1 2) are equivalent. Permutations are composed of one or more algebraic cycles, where each *algebraic cycle* of a permutation is a representation of a domain i, followed by its image (i), followed by getting the image of (i) as the next element, i.e., ((i)), and so on. We continue this process until we reach a repeated element. This procedure uniquely de nes the permutation. We denote by $c(\ )$ the number of algebraic cycles of in relation to and a *k-cycle* is the number of elements of an algebraic cycles of . Moreover, we can de ne in a similar way the algebraic cycles of a permutation $_1$ with relation to $_2$ where both of them are distinct of the identity permutation.

For example, given $_2$ = [1 2 3 4 5] and $_1$ = [2 4 5 1 3], for the rst cycle we select the rst position of $_2$, starting a cycle with 1 and the element at the same position in $_1$, 2. The next element in this cycle is the element in $_1$ at the same position as 2 is in $_2$, selecting the number 4. Since the next step would choose the element 1, the cycle will result in (1 2 4) Using the same process as before, the next cycle will be (3 5). This way, we have $c(\ _1) = 2$, with one 3-cycle and one 2-cycle.

An exchange of elements involving elements a and b such that a and b are in the same cycle is an exchange that *breaks* the cycle in two, whereas if a and b belong to di erent cycles, the exchange of these elements *unites* the two cycles (FERTIN *et al.*, 2009). Thus, when considering the *swap* metric, the *swap* distance of a

permutation is determined as follows: $d_{swap}(\ ) = n \quad c(\ )$, where $c(\ )$ is the number of algebraic cycles of .

Biological concepts Living beings have in their cells an organic compost called DeoxyriboNucleic Acid (DNA) responsible to de ne their behaviors. In this set of information, the one responsible to generate a protein for the body that can have a variety of functions is called gene, it also has the DNA information that is passed for the next generations. These genes can be found inside the core of the cell as chromosome, they are a linear sequence having its formation and structure varying for each organism. Considering the human being, each cell contain a set of 23 pairs of chromosomes. Consequently, genome is a set of genetic information of a being. It has within itself all information capable to generate some information and utility for the being, in other words, it is the set of genes of each individual.

It is possible to represent the genome by di erent forms, like permutations. Even though this is a simple form to represent the DNA, it is still enough for the study in question since it is possible to use several di erent metrics. This simplicity to represent is a positive point since it facilitates the development of algorithms and to understand given problems.

As said before, our study is based on the swap metric (also called as Cayley operation (FERTIN *et al.*, 2009)). This measurement is based on the permutation representation of the genome and it treats a genome change by the exchange of the position between two distinct elements. The swap operation is a particular case of block-interchange operation, where in this one, instead of exchanging the position of a pair of elements each time, the exchange can happen considering two blocks of elements with any size. In biology, median problem has the goal to nd a common ancestral genome considering di erent species when knowing the genome sequencing of the input.

Note that, given the input permutations, equivalently, we can deal with the median problem relabeling one of them as the *identity permutation*, i.e. whose elements are in increasing order, and relabeling the elements of the other permutations properly. Thus, without any loss of generality, we assume that the three input permutations are in such a way that one of them is the identity.

Median problem by a metric M

The decision version of the median problem by the metric M is formally de ned in the following way:

Median by the metric M (M{Med)
INPUT: Set of permutations $f\ _1;\ _2;\ _3g$, and a not negative integer f.
QUESTION: Is there a permutation such that $\sum_{i=1;2;3} d_M(\ _i;\ ) \quad f$?

As consequence of the triangle inequality, any permutation solution of the median f for the permutations $_1; _2; _3$ satis es $f \quad \frac{d_M(_1;_2)+d_M(_1;_3)+d_M(_2;_3)}{2}$. A permutation that has the minimum sum of the distances between and the input permutations is called the *median solution* (or *median permutation*).

There are in the literature di erent studies about Median problems with di erent focus. Silva et al. (SILVA *et al.*, 2023) addressed this problem under a restricted rearrangement measure called C_4*-distance*, having developed heuristics and linear programming formulations. Shao and Moret (SHAO; MORET, 2014) considered the DCJ metric. Tannier et al. (TANNIER; ZHENG; SANKOFF, 2009) showed a number of algorithmic study with multiple multichromosomal genomes. When considering swap-med, its computational complexity is still open.

Note that, similar as distance problems, regarding median problem, we can consider $_1$ as $= [1\ 2\ \ n]$ to make the study simpler and relabel the other two permutations accordingly. For example, if $_1$ is [3 1 2], $_2$ is [2 1 3] and $_3$ is [1 3 2], we can re-label $_1$ to [1 2 3], $_2$ to [3 2 1] and $_3$ to [2 1 3].

Convexity problems

Graph convexity is a rich eld in which several combinatorial problems and convexity types can be de ned according to the application at hand (PENSO *et al.*, 2015). In general, we deal with sets of vertices to be increased step-by-step by following some prescribed rule, until the process stabilizes into a *convex set*.

A nite convexity space is a pair $(V;C)$ consisting of a nite set V and a family C of subsets of V such that $; 2 C$, $V 2 C$, and C is closed under intersections. Members of C are called *convex sets*. Let P be a collection of paths of a graph G and S a subset of V, and let $I_P : 2^{V(G)} ! 2^{V(G)}$ be a function (called *interval function*) such that:

$I_P(S) = S [fz 2 S \mid 9\ u;v 2 S$ such that z lies in an uv-path $P 2 Pg:$

Distinct choices of P lead to interval functions of quite di erent behavior. Such functions, in turn, are naturally associated with special convexity spaces (the so-called path convexities). For instance, if P contains precisely all the shortest paths in a graph then the corresponding interval function is naturally associated with the well-known geodesic convexity; if P is the collection of induced paths then the corresponding interval function is associated with the monophonic convexity; and there are many other examples in the literature.

In this study, we propose Steiner convexity, which is the collection of vertices of *Steiner trees* with three permutations as input, along the vertices of shortest paths between the input. The *Steiner tree* with respect to three input permutations consists in nding a permutation (a vertex in the corresponding graph) that minimizes the total weight of its edges (FAMPA; LEE; MACULAN, 2016). In other words, a median solution of a given set of three permutations belongs

to a solution of the Steiner problem. Based on this, the Steiner convexity yields the collection of all elements that form possible solutions for the Steiner tree, considering a properly metric.

As a consequence of the just presented de nitions, if the geodesic interval convex set ($I_G(S)$) is equal to the monophonic interval convex set ($I_M(S)$), then they are equal to the Steiner interval convex set ($I_S(S)$). Overall, $I_G(S)$ $I_S(S)$ $I_M(S)$.

3. BRUTE FORCE ALGORITHMS FOR SWAP MEDIAN

In order to evaluate the quality of the proposed algorithms of Section 4, rstly, we develop brute force algorithms, sequential and parallel strategies. Initially, Algorithm 1 presents the sequential one:

Algorithm 1: *SeqBruteForce*$_{swap}$(; ;)

```
   input : Three permutations ; ; with n elements, where = [1; 2; ; n]
   output: Median permutation
 1 lower bound = (d_swap(π) + d_swap(π,σ) + d_swap(σ)) / 2
 2 best = d_swap( ) + d_swap( ) /* Take as solution candidate */
 3 for candidate in Permut(n!) do
       /* Permut(n!) has n! permutations */
 4     solution = d_swap(candidate) + d_swap(candidate; ) + d_swap(candidate; )
 5     if solution == lower bound then
 6         best = solution
 7         return best
 8     else if solution < best then
 9         best = solution
10 return best
```

Clearly, Algorithm 1 has exponential complexity, since there are $O(n!)$ possible permutations to check for a given input with three permutations of size n. A possible way to parallelize such approach is by searching the solution permutation as follows:

1. For each *thread* i, create all permutations that start with i;
2. Execute Algorithm 1 in each thread i changing Line 3 by the set of permutations that starts with i.

Note that this approach has a complexity gain, since each *thread* executes $O((n \quad 1)!)$ checks. Besides, since we made a partition of the universe space, we ensure that two *threads* do not create a same candidate.

Furthermore, the parallel approach can be generalized by creating for each thread the permutations that start with a combination of $k < n$ elements. Note

that the parallelism above, we have settled $k = 1$. In this new scenario, there would be $\binom{n}{k}$ threads and each thread would generate $k!(n - k)!$ permutations.

4. HEURISTICS FOR SWAP MEDIAN

We develop greedy approaches by considering: algebraic cycles (*Heuristic by common cycles*); elements we can ensure their correct positions (*Heuristic by good columns*); elements chosen in each position being one of three candidates of the input (*Heuristic by choosing best of three per column*); permutations generated in optimum paths between every pair of input (*Heuristic by grid*); new input being permutations belonging to the half of the distance for each pair of permutations of the input (*Heuristic by half distance*).

Heuristic by common cycles In order to transform one permutation into another one with minimum swaps, each operation must be done breaking cycles. Hence, looking for the median solution considering the input $_1$; $_2$; $_3$, a possible approach is by starting at one of three permutations, $_1$, searching for an operation that at same time breaks cycle with respect to each $_2$ and $_3$. If it is possible, we apply this swap and continue the search. Next, we do the same starting with $_2$, and nally, starting with $_3$. This procedure is fully described in Algorithm 2.

Algorithm 2: Common cycles (; ;)

```
   input : Three permutations ; ;
   output: Permutations ŝ median candidate
 1 i  0; j  0; k  0
 2 while 9 swap t such that t breaks a cycle and
     approach  to  do
 3     t
 4     i  i + 1
 5 while 9 swap t such that t breaks a cycle and
     approach  to  do
 6     t
 7     j  j + 1
 8 if  ==  then
 9     ŝ
10     ŝ is a candidate solution
11 else any swap t of  to , and vice-versa, unite
   cycles
12     while 9 swap t such as t unite cycles and
         approach a  and  do
13         t
14         k  k + 1
15     x  i + j + k + minfd( ; ) +
         d( ; );d( ; ) + d( ; );d( ; ) + d( ; )g
16     if x == i + j + k + d( ; ) + d( ; ) then
17         ŝ
18     else if x == i + j + k + d( ; ) + d( ; )
         then
19         ŝ
20     else
21         ŝ
22     ŝ is a candidate solution
```

Heuristic by good columns

A *column* is de ned as the elements in a same position over the 3 permutations. Hence, given three input permutations ; ; , a column i has elements i, (i) and (i). In order to obtain a median solution, a re nement can be done when the inputs satis es some restrictions. Following, we deal with the case where in a column at least two elements the same.

Theorem 1. *Given three permutations as an input of the* swap median *problem, if there is a column i with at least two elements equal to a, then in any median solution, the element a is in position i.*

Proof. Let s be a candidate solution in which there is a column i at the input with two elements equal to a and s do not have a at position i. Starting from s, we obtain s' by setting at position i the element a, and next we show how to obtain s' and how it corresponds to a better solution than s. Consider a column i in such a way that, without loss of generality, and agree and do not agree with a (at this column has the element b). Besides, the median candidate s has $c = s(i) \neq (i)$. Hence:

Case 1) $s(i) \neq (i)$.

Case 1.1) Element a in s disagrees with . Changing at s the element c with a, we have that, overall, the new permutation decreases at least 2 units (1 unit to and 1 unit to). This way we are getting closer the permutation to and to at the same time. Furthermore, we increase at most 1 unit in relation to (since happened swap at this change from s to s' that could worse the solution to eventually). Therefore, s' is a better solution than s.

Case 1.2) Element a in s agrees with . Changing at s the element c with a, we have that, overall, the new permutation decrease into at least 2 units (1 to and 1 to) and increase 1 unit to (since a starts to disagree at s' with), but also it does not get worse than 1 unit, since, equal to the case above, we only applied one swap, so the distance increases by one unit.

Case 2) $s(i) = (i)$.

Case 2.1) Element a in s disagrees with . When changing in s the element c with the element a, we decrease by at least 2 units (1 to and 1 to) and increase 1 unit for (1 swap).

Case 2.2) Element a in s agrees with . When changing in s the element c with a, the new permutation s' increases 1 unit the distance to , again we only left one more unit distant for having applied 1 swap and decrease by at least 2 units (1 to and 1 to).

We have obtained in all cases a permutation whose sum of distances is the best one when the solution has element a in position i. This concludes the proof. □

Given an input, when there is a column with two elements being the same, then we call such a column as a *good column*. Based on Theorem 1, we can make a

safe reduction when there is a good column. Thus, when considering a column with exactly two elements with the same value, without loss of generality, let the permutations and with the element a at position i, and in relation to the permutation , let $(a) = b$.

Let i be the position that $(i) = a$, where $i \neq a$. The median permutation has at the position a the element a and when creating a permutation s candidate to be the solution, we consider $s(i) = b$. Note that, the permutation s in relation to creates an algebraic cycle of size 2, and applying one swap in s we make two elements right in relation to . The cycle of size 2 is created because $s(i) = b$, $s(a) = a$, $(i) = a$ and $(a) = b$.

Next, we summarize this strategy. Consider (in general) $_1$; $_2$; $_3$ the input permutations:

Step 1: For each column with at least 2 elements being the same, put this element in the solution candidate we are creating:

Step 1.1 : For each column that has 2 common elements (both equals to a at the permutations $_1$ and $_2$), take the third element b (let b be the element at this column in permutation $_3$) that is distinct from both, and at the solution, put b at the position where a appears in $_3$. If there is a collision, consider: case i) create one permutation; or case ii) create all permutations separately. Let c be the number of columns with at least two equal elements. In the end, there are at least $n - 2c$ columns to allocate the remaining elements in each candidate.

Step 2 : For the remaining positions, execute a greedy heuristic: Allocate one of the three possible elements that are at the columns not lled by the previous steps. Consider all cases. Or,

Step 3 : Execute a brute force algorithm when there are few columns left to be lled. A brute force can be in parallel, in relation to each candidate returned permutation from Step 1.1 and each candidate of Step 1.1 creates a new thread.

Consider, for example, the input $_1 =$, $_2 = [2; 3; 4; 5; 6; 7; 8; 9; 11; 10; 1]$ and $_3 = [7; 3; 6; 4; 2; 10; 11; 8; 1; 9; 5]$. After Step 1 above, the candidate median permutation is initially built by the columns with common elements.

1	2	3	4	5	6	7	8	9	10	11
2	3	4	5	6	7	8	9	11	10	1
7	3	6	4	2	10	11	8	1	9	5
	3		4				8		10	

After Step 1.1 there are some collisions, generating four candidates to be the solution, we list one per line after the input:

1	2	3	4	5	6	7	8	9	10	11
2	3	4	5	6	7	8	9	11	10	1
7	3	6	4	2	10	11	8	1	9	5
	3	2	4			9	8		10	
	3	5	4		9		8		10	
	3	2	4		9		8		10	
	3	5	4			9	8		10	

In relation to the remaining positions, following Step 2 we use a greedy approach by allocating for each position one of the three input permutations and for the four candidates initially generated, return the best solution. Another possible solution is obtained when the number of columns to be lled is reduced. Eventually we can use a brute force algorithm, a sequential or a parallel one by the number of candidate solutions by the collisions returned in Step 1.1.

Note that *heuristic by good columns* only works when there are columns with coinciding values in at least two permutations. Therefore, in Theorem 2 we analyse how often is an instance to have good columns. A permutation it called *chaotic* if $(i) \neq i$, for $i = 1; \ldots; n$ (GRAHAM *et al.*, 1989; BONA, 2022). Thus, a *non-chaotic* permutation is a permutation that satis es $(i) = i$, for some $i \in \{1; \ldots; n\}$.

Theorem 2. *An input instance for the* median *problem has a high probability to have good columns.*

Proof. We want to know how many inputs have coinciding elements in a column. Since one of the permutations in the input is the identity permutation:

i) A way to count the number of inputs that agree values in a column is that one of the other two permutations be *non-chaotic*, in other words, to have at least one element i at position i.

ii) Another way is the case of coinciding columns among permutations that are not the identity.

The number of chaotic permutations of size n can be obtained by the closest integer to $\frac{n!}{e}$ (BONA, 2022). Hence, the number of non-chaotic permutations of size n is $\frac{n!(e-1)}{e} \approx n!0{:}63$. In other words, approximately 63% of the permutations of size n are non-chaotic.

Let us analyze Case i), i.e. how likely is the three permutations { the identity , and { having at least one element i at position i. By the Multiplicative Principle, we have $\frac{n!(e-1)}{e}(n! - 2)$, subtracting two units for the chances that the third permutation to not be nor .

Let us analyze Case ii). It considers the cases that and agree in some column but not with . Since we are interested in agreement of with , this amount

is analogous to case i) when we are interested in and another one. Therefore, the count is the same as the previous case. This implies the amount of instances be 2 $\frac{n!(e\ \ 1)}{e}$:($n!$ 2) .

In order to obtain the probability of having favorable inputs for the heuristic, we have what follows. Sample space: $\frac{n!}{2}^{1}$ n^{2n}, once we only need two permutations for the input, because the identity permutation is xed; Event: 2 $\frac{n!(e\ \ 1)}{e}$:($n!$ 2) 1:26(n^{2n} $2n^{n}$) $n^{2n}(1\ \ \frac{2}{n^{n}})$. Therefore, with high probability the input have good columns, once we remove a small portion of $\frac{2}{n^{n}}$ from the possible events. □

Furthermore, even greater re nement can be done when the input permutations follow more constraints, as proved in Theorem 3.

Theorem 3. *Let C be a column with the elements a, c, c for the input $_1$, $_2$ and $_3$, respectively. Let C^0 be the column where the element c appears on $_1$. If a belongs at C^0, then there is a solution that the element a belongs to the column C^0.*

Proof. The median solution maximizes the number of algebraic cycles, it is the one that has the greatest number of cycles while having each cycle be as small as possible. Considering the Theorem 2, the element c must belong to the column C. Let us consider the three possible cases for the column C^0.

Case 1) The element a belongs to C^0. In this case, when comparing the elements from the columns C and C^0, there are 3 algebraic cycles of size 1, 1 cycle of size 2 and 1 cycle of size greater than or equal to 2. It can be viewed by the following cycles we have created: $_1$ with respect to has a 2-cycle (c a); $_2$ with respect to has a 1-cycle (c) and a k-cycle, for k 2, (b a); $_3$ with respect to has two 1-cycles (a) and (c).

	C^0	C
$_1$	c	a
$_2$	b	c
$_3$	a	c
	a	c

Case 2) The element b belong to C^0. In this case when comparing the elements from the columns C and C^0, there are 3 cycles of size 1, 1 cycle of size greater or equal to 2 and 1 cycle of size greater or equal to 3. It can be viewed by the following cycles we have created: $_1$ with respect to has a k-cycle, k 3, (c b a); $_2$ with respect to has two 1-cycles (b) and (c); $_3$ with respect to has a 1-cycle (c) and a k-cycle, for k 2, (a b).

	C^0	C
1	c	a
2	b	c
3	a	c
	b	c

Case 3) The element d di erent from a and b belong to C^0. In this case when comparing the elements from the columns C and C^0, there are 2 cycles of size 1, 2 cycle of size greater or equal to 2 and 1 cycle of size greater or equal to 3. It can be viewed by the following cycles we have created: $_1$ with respect to has a k-cycle, k 3, (c d a); $_2$ with respect to has a 1-cycle (c) and a k-cycle, for k 2, (b d); $_3$ with respect to has a 1-cycles (c) and a k-cycle, for k 2, (a d).

	C^0	C
1	c	a
2	b	c
3	a	c
	d	c

Note that all elements apart from a, b, c and d are not changed in any of the 3 cases above. However, the cycles involving the element d in Case 3 increase by one unit in relation to Case 2, $_2$ compared to (due to the element b), at least 1 unit in relation to Case 1, $_1$ compared to (due to the elements c and a), and at least 1 unit in relation to Case 1, $_3$ compared to pi (due to the element a). Likewise, in Case 2, $_1$ compared to adds at least 1 unit compared to Case 1. Therefore, Case 1 is the best of the options for a median permutation. □

Heuristic by choosing best of three per column

An eventual generalization of Theorem 1 and Theorem 3 is the following statement:

Conjecture: There is a median permutation that each position i of the solution has one of the elements from column i of the input.

This is a natural conjecture and we have example of instances that satis es this proposal, as the example below, once its solution satis es each position to be equal to some element from the input in that position. The solution is obtained can be easily veri ed by a brute force procedure (parallel algorithm developed in Section 3):

1	2	3	**4**	5
4	**3**	**2**	5	**1**
5	4	1	3	2
5	3	2	4	1

Even though this is an interesting approach, because that the search space to execute a brute force reduces from $n!$ to 3^n, it only induces to a heuristic, since we have instances that all solutions have at least one column whose element is distinct from each of the three input permutations. An example that refutes the previous conjecture is given next, whose all median permutations were obtained by a brute force procedure (parallel algorithm developed in Section 3):

1	2	3	4	5	6	7	8	9	10	**11**
2	3	4	5	6	7	8	9	11	10	**1**
7	3	6	4	2	10	11	8	1	9	**5**
1	3	6	4	5	7	11	8	9	10	**2**

Besides the solution presented above, where the last column do not have a common element to any of the inputs, all other solutions have this same behavior.

An in nite family of instances that refute the previous conjecture can be obtained as follows: create an instance in such a way that we force an element b to be at distinct columns, where in each column that b appears, there are some other two common elements.

Based on this construction, any median solution of these instances, the columns where b appears do not have b at the median solution. See below a situation where this happens to the element 2. As consequence of Theorem 1, we know that at the rst, second and fth columns, any solution has the elements 1, 3 and 5, respectively:

1	**2**	3	4	5	6	7	8	9	10	11
2	3	4	6	5	7	8	9	11	10	1
1	3	6	4	**2**	10	11	8	7	9	5
1	3		4	5			8		10	

Although a median solution do not necessarily have in a position one of the three candidates of the input, we have cases where the solution satis es each column to be common to some input element. Hence, based on this, we can consider a possible candidate solution permutation to be obtained as follows:

1. Take as solution a permutation that maximize the number of columns with value equal do any of the input.

2. If there is more than one possibility, take the best one as a candidate median solution.

Recall that in Theorem 1 we have presented a case where we ensure the position of an element for any solution permutation. Continuing this approach, some more positions can be guaranteed for a pair of elements that appear in two columns, as stated next in Theorem 4.

Theorem 4. *Let C be a column with three elements where two of them are the elements a, b, for any input. Let C' be another column containing the element a. If b also belongs to C', then any solution contains a and b either in columns C and C', resp, or in columns C' and C, resp.*

Proof. Since the median solution maximizes the number of algebraic cycles, it is the one that has the greatest number of cycles while having each cycle be as small as possible. Let us consider all possible cases for the columns C and C', while fixing that the elements a and b appears once at C and C'.

Case 1) The elements in C and C' at are different from all inputs. In this case, when comparing the elements from the columns C and C', at least will have 6 algebraic cycles of size greater than or equal to 2.

It can be viewed by the following cycles we have created:

Six k-cycle, for k 2, (a e), (b f), (b e), (a f), (c e) and (d f).

	C'	C		C'	C		C'	C
1	a	b		a	b		a	d
2	b	a	or	b	d	or	b	a
3	c	d		c	a		c	b
	e	f		e	f		e	f

Case 2) The elements in C at is equal to the element at 3 but different in C'. In this case, when comparing the elements from the columns C and C', the comparison between 1 with respect to and 2 with respect to are the same as the above, though when 3 with respect to there is an algebraic cycle of size 1 and one algebraic cycle of size greater than or equal to 2. Making a total of 5 algebraic cycles of size greater than or equal to 2 and 1 algebraic cycle of size 1.

It can be viewed by the following cycles we have created:

Five k-cycle, for k 2, (a c), (b e), (b c), (a e) and (d e) and one 1-cycle (c).

	C'	C		C'	C		C'	C
1	a	b		a	b		a	d
2	b	a	or	b	d	or	b	a
3	c	d		c	a		c	b
	c	e		c	e		c	e

Case 3) The elements in both C and C' at are equal to the elements at 3. In this case, when comparing the elements from the columns C and C', the

comparison between $_1$ with respect to and $_2$ with respect to are the same as the above, though when $_3$ with respect to there are 2 algebraic cycle of size 1. Making a total of 4 algebraic cycles of size greater than or equal to 2 and 2 algebraic cycle of size 1.

It can be viewed by the following cycles we have created:

Four k-cycle, for k 2, (a c), (b d), (b c) and (a d) and two 1-cycle (c) and (d).

	C^0	C		C^0	C		C^0	C
1	a	b		a	b		a	d
2	b	a	or	b	d	or	b	a
3	c	d		c	a		c	b
	c	d		c	d		c	d

Case 4) The elements in C at is a and the element in C^0 at is b. In this case, there are a total of 2 cycles of size 1, 1 cycle of size 2 and 2 cycles of size greater than or equal to 2.

It can be viewed by the following cycles we have created:

Two k-cycle, for k 2, (c a) and (d b), one 2-cycle (b a) and two 1-cycle (a) and (b).

	C^0	C		C^0	C
1	a	b		a	d
2	b	a	or	b	a
3	c	d		c	b
	a	b		a	b

Case 4.1) If in all three inputs there is not a permutation with a at C^0 and b at C.

In this case, there are a total of 2 cycles of size 1 and 2 cycles of size greater than or equal to 3.

It can be viewed by the following cycles we have created:

Two k-cycle, for k 3, (c a b) and (d b a) and two 1-cycle (a) and (b).

	C^0	C
1	a	b
2	b	d
3	c	a
	a	b

Note that all elements apart from a, b, c, d, e and f are not changed in any of the 4 cases above. Also note that even if a and b appears at di erent permutations,

meaning they do not need to be at ^{1} and ^{2} at this order the algebraic cycles will remain the same. If this happens, this new case will have the same solution as one of the cases showed before. When analyzing all cycles, Case 4 shows the best results, having more cycles with less size than all other possible cases, with 2 cycles of size 1, 1 cycle of size 2 and 2 cycles of size greater than or equal to 2. Even if the instance falls into case 4.1, the elements a and b are the best possible candidates for the columns C and C^0. Therefore, Case 4 is the best option for a median permutation. □

The result above leads us to construct in nite families of instances in such a way that it is possible to decompose it into pairs of columns where each pair satis es the hypothesis of Theorem 4.

Moreover, Theorem 4 implies that if there are three columns $C; C^0; C^{00}$ where C and C^0 have a and b in common, and C^0 and C^{00} have b and c in common, then any solution must have b in column C^0, and so a must be in column C and c must be in column C^{00}.

Heuristic by grid

While analyzing all heuristics above, we noticed that the solution would be at a shortest path between at least two inputs permutations. Given three input permutations $;$ $;$ (to simplify, rewrite them as A, B and C), take a shortest path between A and B, for each possible solution look for the shortest path between this solution and C. See the full strategy below:

Take an optimal path between A and B, assume that from A to B we passed through the permutations A_1, A_2, A_3, , $A_d = B$, where d is the distance between A and B. Now consider A_1 and try to get closer to C. In this approach, we can leave the optimal path from A to B, but we are in control of having gone one unit away. It continues approaching C, that is, from A_1 to C we pass through A_{11}, A_{12}, A_{13}, , $A_{1d'}$, where d^0 is the distance between A_1 to C. These are the steps to get from A_1 to C. Each of these permutations can, in addition to getting closer to C, move away or get closer to B. We try to take paths that get closer to B whenever possible.

At each moment, the candidate is calculated with the previous one and in the end we have the best among A_1, A_{11}, A_{12}, , $A_{1d'}$ as a candidate for the solution. After that, do the same as A_2 going to C, that is, follow the paths A_2, A_{21}, A_{22}, , $A_{2d''}$, where d^{00} is the distance between A_2 and C. Likewise, get the best of all the ones we have obtained so far. Then, do the same for A_3, and so on. For each A_i, for $i = 1;$ $;d$, we have the possible solutions for A_{ij}, for j going from 1 to the distance from A_i to C.

Since the swap distance is at most n 1, then we have $(n$ 1) $(n$ 1) $= O(n^2)$

candidate median solutions after these steps. In the end, we choose the best of this set. This is a heuristic because we are only considering an optimal path from A to B and then the path from A to B going to C. So, in the end we are considering a candidate solution that is necessarily on an optimal path.

Note that we are taking the initial basic path as an optimal path from A to B, then we have to do it from A to C and then from B to C, as well. Thus, in the end we have obtained $O(n^2 + n^2 + n^2) = O(n^2)$ candidate median solutions.

This algorithm can also be parallelized, just make each thread to start from one of the input, A, B, or C.

Heuristic by half distance

This strategy is similar to the heuristic above. Now, take a permutation midway from a shortest path from and , call it $_1$, do the same from to , $_1$ and from to , $_1$. After that, the candidate solution is the best one between $;\ _1;\ ;\ _1;\ ;\ _1$. Now, consider the new input being $_1;\ _1;\ _1$ and repeat the previous process by taking permutations in the midway of shortest paths. Repeat this process until we do not decrease the solution anymore, or until a number of steps were made.

Figure 1 illustrates this heuristic after the rst iteration. Note that the distance between each pair of permutations do not need to be the same and likely will not be in any iteration. Even though it looks interesting, in most cases new interactions tends to create a candidate that is further from the input, so having a greater distance. Also, apart from the rst iteration, all the other candidates do not belong to the shortest path from any pair of the original inputs, as shown in Figure 1, $_2$ do not necessarily belong to a shortest path between and .

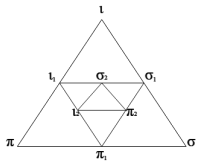


Figure 1 { Illustration of two iterations of *heuristic by half distance* algorithm.

5. CONVEXITY AND MEDIAN PERMUTATIONS

Firstly, we found ourselves with two questions.

First question: Does a median permutation always belong to a shortest path between at least two input permutations?

Second question: If the rst question is true, can we do nd a median permutation with a brute force approach in a feasible time?

In the following, we investigate those questions by using algebraic cycles of permutations (Subsection 5.1) and by using the concept of convexity in graphs (Subsection 5.2).

5.1. EXPONENTIAL NUMBER OF OPTIMUM PATHS

Actually, even if the rst question is true, a brute force algorithm is not e cient, as we show below.

Using the algebraic cycles, we can analyze how many sequences there are in which any operation that is applied from two elements of a same cycle. This is important to focus because in all instances we analyzed, the median solution we have obtained belongs to a shortest path of at least one pair of the input, as we describe in Section 6. So, if we could prove that the number of permutations in optimum paths was a polynomial function, then it could be a nice approach to develop and try to analyze the rst question. However, in Theorem 5 shows the opposite with respect to the second question, even if the rst question is true.

Theorem 5. *There is an exponential number of permutations belonging to all shortest paths with respect to an instance of the swap median problem.*

Proof. Let , and be the input permutations of size n. For each pair of such permutations, there is a corresponding algebraic cycle, one from to , from to and from to . Consider the algebraic cycle from to to be made only of cycles of size 2. Hence, there are $\frac{n}{2}$ 2-cycles. Since the swaps applied in any optimum path breaks a cycle, then there are $(\frac{n}{2})!$ sequences of swaps, i.e. optimum paths, in order to transform into .

Note that in each of those sequences, there is at least a permutation not contained in another optimum path. In general, there are $\frac{n}{2}$ operations to be applied in such a way that all cycles become of 1-cycles, where with $(\frac{n}{2})!$ we obtain the number of possible choices of the $\frac{n}{2}$ 2-cycles. Each new sequence add at least a distinct permutation, otherwise two distinct sequences would have all permutations the same, this way this two permutations would not be distinct.

Thus, there is an exponential number of permutations belonging to the shortest paths from to , which is su cient to guarantee an exponential number of permutations also from to and from to . This concludes the proof. □

In order to answer the rst proposed question of this section, we separated all instances in three di erent types.

Let , and be the inputs and S the median:

Recall that, as consequence of the triangle inequality, there is a lower bound for the problem, where $S \quad \frac{d_M(\ ;\ )+d_M(\ ;\ )+d_M(\ ;\ )}{2}$. Then, comparing to such lower bound, we have the following cases:

Type 1) S is equal to the lower bound. In this case, the median permutation belongs to the shortest path between the input.

If S does not correspond to the lower bound, then there are two options:

Type 2) It may be that the solution permutation is on the optimal path of at least one pair and at most two pairs among and , and , and and .

Type 3) It may be that the solution permutation is not on the optimal path between and , between and nor between and . Thus, the permutation solution will be at least 3 units above the lower bound.

Hence, in order to conclude that the rst question is false, we need to nd an instance that, accordingly to these types, can be classi ed as Type 3.

For all instances of sizes 8, 9 and 10, we used a brute force approach (Algorithm 1) and found out that there are no instance of Type 3, most of them are of Type 1. We also tried the same for some instances of sizes 11 and 12. Since the sample size is larger (there are $n!$ $n!$ possible instances), it is not possible to apply Algorithm 1 for all of them. Hence, we selected some instances randomly, and while there were more instances of Type 2, there was not a single one of Type 3.

For sizes greater than 12, we have used the *heuristic by grid* strategy. Hence, for sizes between 13 and 25, we selected 1; 000 instances of each size and have chosen those that the solution found by the heuristic did not belong to any shortest path. Curiously, this only happened 4 times, 1 instance of size 17, 1 instance of size 20 and 2 instances of size 25.

For the one of size 17 we used both Theorem 1 and Theorem 3 to reduce the amount of empty columns, once we could ensure elements in some positions, and we obtained an instance of size 11, Thus, by applying Algorithm 1, we found that this instance is in fact of Type 2.

For the remaining three other instances, with Theorem 1 and Theorem 3 we could not reduce the number of empty columns enough, so the parallelism of Algorithm 1 would not return a median permutation in a feasible time.

We keep as open question if median permutations are always on shortest paths between pairs of input permutations or if there is any instance classi ed as Type 3.

5.2. CONVEXITY AND MEDIAN SOLUTIONS

In order to investigate the rst question, we return the focus on convexity problems, presented in the Section 2.

Note that if the rst question is true, then all median solutions belong to the geodesic convex set, i.e. they belong to the interval function $I_G(S)$, for G

being geodesic convexity, S consists of the median input of three permutations. Hence, this would imply that geodesic interval convex set is equal to the Steiner interval convex set.

First, Figure 2 shows two graphs that satisfy the triangular inequality (necessary condition to de ne any metric, where for any pair of vertices a and b, the distance between them is at most the distance between a and c plus the distance between c and b, for c being another vertex). In order to represent the geodesic interval convex set, which is the collection of the shortest path of the graph, the vertices returned do not necessarily contain the Steiner interval convex set.

Figure 2(Left) M is a median solution, which returns the sum of the distances equal to 6. Note that M does not belong to geodesic interval convex set associated to $;\ ;\ $. However, any of those input vertex is also a median solution, then, the geodesic interval convex set contains a solution for the median problem, even do not containing all median solutions.

Despite that, Figure 2(Right) illustrates an even more general case, where M is a median solution, with value equal to 9, and no input vertex is a median solution, neither any other vertex of the graph. Therefore, overall, median solutions do not belong to geodesic solutions for graphs that satisfy the triangular inequality.

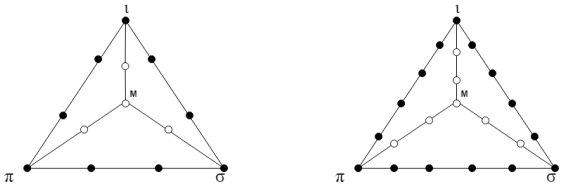


Figure 2 { Two examples of geodesic convex sets in bold of graphs with input being $;\ ;\ $, where the median solutions do not necessarily belong to it.

A *cube graph* Q_n is the graph with 2^n vertices, each of them represents a binary string with n bits, and two vertices are adjacent if they have Hamming distance equal to 1, where the *Hamming distance* between two binary strings s and r of the same length is de ned as the number of mismatched positions between s and r. Thus, cube graphs represent the Hamming metric on binary strings. Now, take a look at the monophonic convexity, where the interval set is made of all induced paths of the graph, the cube graph Q_4 of Figure 3 shows an example where the geodesic interval convex set is a proper subset of the monophonic interval convex set.

In Figure 3, consider vertices A and C as input. Note that $A;B;C$ and $A;D;C$ are the unique paths belonging to both geodesic and monophonic interval convex sets, while $A;E;F;G;C$ belongs to the monophonic convexity.

For another example, now considering permutations, take the *Exchange of adjacent elements* graph. This graph represents operations which are special type of

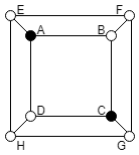


Figure 3 { Cube graph Q_4.

swaps, where given a permutation , only consecutive elements of can be swapped in . In other words, in we can only apply a swap between 1 and 2, 2 and 1 or 3, 3 and 2 or 4, and so on.

Figure 4(Left) represents the *Exchange of adjacent elements* graph for permutations of size 4. This operation is an example where the geodesic and monophonic convexity are not the same sets. Assume an input being permutations of an equal face, such as (2 4 1 3) and (4 2 1 3), then its geodesic interval set ll the other four vertices of such a face, whereas its monophonic interval set add of other faces, such as (1 4 2 3)*;* (1 4 3 2)*;* (2 4 3 1)*;* (2 3 4 1)*;* (3 2 4 1)*;* (4 2 3 1)*;* (4 3 2 1).

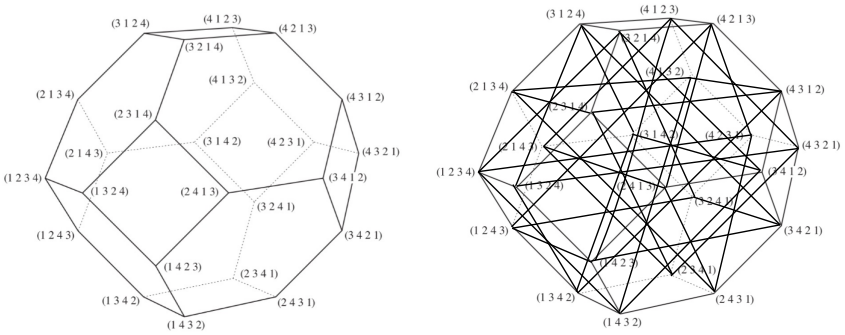


Figure 4 { (Left) *Exchange of adjacent elements* graph for permutations of size 4. (Right) *Swap* graph for permutations of size 4.

Using this same instance of permutations of size 4, but now considering the swap operation, we see that these sets are not the same either, as presented in Figure 4(Right). Consider the paths between (3 4 1 2) and (4 3 2 1). The geodesic interval convex set consists of *f*(3 4 1 2)*;* (4 3 2 1)*;* (4 3 1 2)*;* (3 4 2 1)*g* while monophonic interval convex set consists of these previous elements and some more, such as (2 4 3 1) and (2 4 1 3).

Considering the Steiner convexity, which contains all possible solutions for the Steiner tree problem, the solution for the problem is then a median permutation. Thus, a possible way to investigate such a convexity is to ask if it is equal to geodesic or monophonic or either if it is a proper subset of the monophonic interval convex set.

6. OBTAINING MEDIAN PERMUTATIONS IN PRACTICE

The experiments were made in a computer with 32 GB RAM and 66 GB HDD. The tests happened at a virtual machine with a common KVM having 4GiB Ram and 8 Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz. First, for the calculations and the execution of the algorithms, were generated 100 instances for each of the 4 sizes of permutations, from 8 to 11 elements. For each instance were created three distinct permutations and one of them is the identity permutation, with out loss of generality, as mentioned in Section 2. After this experiment, we used the parallel brute force approach to obtain the median solutions for all instances of sizes from 8 to 11 in order to find the types the solutions are, and used *heuristic by grid* for permutations of sizes from 13 to 25, creating 1; 000 instances per each size.

The implementation for this experiment was made in C using the library Pthread for the parallelism in the brute force approach. We used the library Open Mp with the heuristic by grid. All implementations can be found in github.

All heuristics described in Section 4 were made so we could compare the time execution and the quality of the results by the brute force approaches. Next, we show graphics with the average time of the sequential approach and the parallel one using two, four and eight threads (Figure 5), as well as their speedup graphics (Figure 6). By the speedup graphic (Figure 6), it is possible to notice that the results show a sublinear scaling.

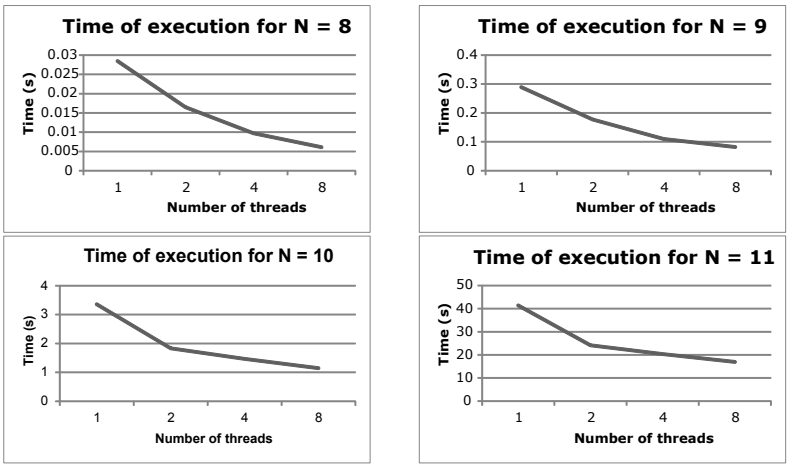


Figure 5 { Graphics with the average time of execution for 100 permutations of size 8 until 11.

For the heuristics, we made a comparison between the execution time and its quality of solution (Figure 7). For all inputs, *heuristic by common cycles* showed a different result compared to the brute force only once, by returning one unit further the exact solution.

The *heuristic by good columns* generated optimum results in all cases, since at worst it executes a brute force after ensure columns based on Theorem 1 and

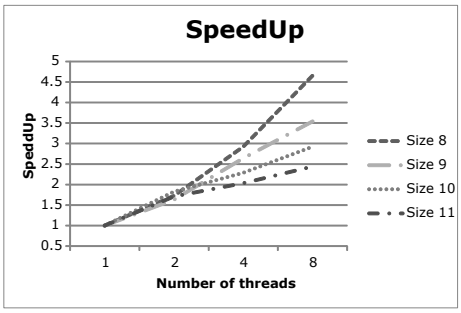


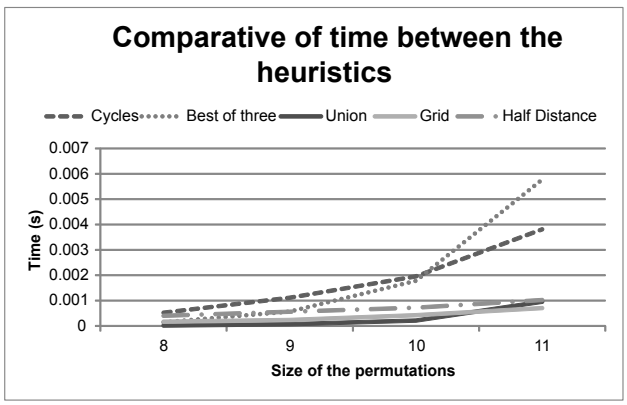
Figure 6 { Comparative graphic with the speedup for each size.

Theorem 3, though with a time limiting based on necessity. Its implementation was made considering the risk of happening a collision after Step 1.1, considering both Cases i and ii of that heuristic. Since this is based on the amount of good columns, we also counted the number of times it happened. Considering all 400 inputs, in this experiment, only 8 inputs did not have any good column, corroborating Theorem 2.

Even though it had a good result and its average time was close to brute force with eight threads, all heuristics, except the *heuristic by half distance*, had better quality results with a better time execution.

Considering the *heuristic by grid*, the median permutation was indeed obtained in all cases and in an e cient time execution. As we have proved it has quadratic time complexity. In the scenario of 400 instances of size 8 to 12, the solution were always correct, but when trying to look for an instance that do not belong tho a shortest path, this is not true for all instances.

Considering the *heuristic by half distances*, the tests show that in each iteration the best candidate would be even worse than the previous one. Even thought this is a fast heuristic, the solution is not the best one in general.



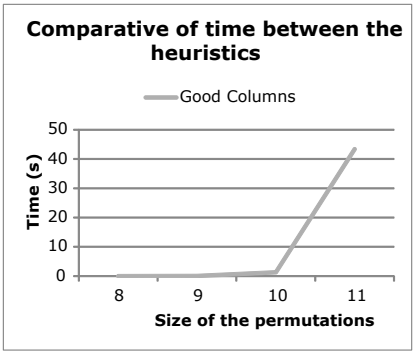


Figure 7 { Graphic comparing the execution time of the heuristics and the *heuristic by good column*.

Table 1 shows how many times each heuristic had the correct answer when comparing to the brute force. The lines represent the permutation sizes and the

columns are the select heuristics.

	Cycles	Good Column	Best of three	Best of three with column	Grid	Half Distance
8	100%	100%	76%	92%	100%	57%
9	100%	100%	45%	74%	100%	57%
10	99%	100%	42%	66%	100%	50%
11	100%	100%	34%	49%	100%	56%

Table 1 { Percentage of success of each heuristic when comparing to the exact solution returned by the brute force approach.

Both Table 1 and graphics of Figure 7 show the e ciency of each heuristic. It is important to note that even though *heuristic by good columns* and *heuristic best of three* had some issues when used individually because the rst one may need a lot of time to be conclude (we may call brute force approach for empty columns) and the second one by the possible non optimum result, both of them used together return a better result than *heuristic by best of three* while it uses less time than *heuristic by good column*.

	Cycles	Good Column	Best of three	Best of three with column	Grid	Half Distance
8	0	0	0,26	0,29	0	1,56
9	0	0	0,62	0,28	0	1,37
10	0,17	0	0,65	0,35	0	1,6
11	0	0	0,77	0,57	0	1,88

Table 2 { Average results of the di erence between the heuristic and brute force.

In Table 2 it is possible to notice that even though the heuristics do not always have the ideal result, its average tends to be low.

Besides this experiment explained above, we also used the brute force approach to see how many instances were of Type 1 (if the solution is equal to the lower bound, as consequence of the triangle inequality), Type 2 (if it is does not correspond to the lower bound, but sill belong to a shortest path between at least one and at most two pairs of the input) or Type 3 (if the solution does not belong to the shortest path between any pair of input permutations).

For this test, we applied the brute force for all instances for sizes 8, 9, 10 and 11. Considering these tests, we did not nd any example for Type 3, and for most of the cases, the solution is of Type 1. Since sizes greater than 10 require a

much longer execution time, we have chosen to use the *heuristic by grid* to obtain candidate solutions. This experiment was explained at the end of Subsection 5.1.

To recall the experiment, we randomly select 1; 000 instances for each size from 13 to 25 and if the solution is Type 3, we would use a brute force approach to make sure if the solution is indeed of Type 3 or not. Considering this experiment, there were only 4 candidate median solutions to be of Type 3 and in one of these candidates we veri ed to be of Type 2 by the brute force algorithm.

7. FURTHER WORK

As future works, we want to investigate some questions proposed in this paper, such as: a merge between the proposed *heuristic by good columns* and then the *heuristic by common cycles* for the remaining empty columns; analyze the parallel implementation of the *heuristic by good column* when a collision happens, as we have shown in Step 1.1 case ii); investigate the parallel brute force approach by creating k threads, for several values of k, consisting of xing the rst k elements of the permutation, and concatenating those elements with the other n k elements; Fully answer the question regarding the Steiner convexity to prove whether for the swap median problem the geodesic and monophonic convexity are the same sets or not; ideally, nd an instance to represent Type 3 instances, as mentioned in Subsection 5.2. This is an instance where a median permutation does not belong to the shortest path between any pair of input. We searched into 13; 000 instances with the *heuristic by grid* ranging 1; 000 instances for each size from 13 to 25 and only 4 of them were we considered possible candidates to be of Type 3. However, by analyzing with a brute force algorithm, we concluded that one of them is of Type 2. The instance we have proved to be Type 2 is:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
8	15	9	16	3	7	1	2	6	10	12	5	14	17	11	4	13
10	3	9	1	15	2	6	12	14	4	11	5	17	16	8	7	13

The exact answer of the parallel brute force algorithm could be obtained without computing all 17! median permutation candidates because columns 3; 10; 11; 12 and 17 satisfy Theorem 1 and column 13 satis es Theorem 3. Hence, we have reduced to an instance with permutations of size 12, and could conclude that it is of Type 2.

The other possible candidates to be of type 3 are:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
9	14	17	20	4	12	2	13	11	15	1	5	6	10	16	7	18	19	8	3
16	1	3	12	2	20	4	14	6	5	8	7	15	19	13	10	18	9	11	17

and:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
9 5 21 6 25 15 24 22 4 11 7 19 10 13 18 3 1 23 16 14 2 12 8 17 20
10 4 23 12 19 13 15 8 24 25 18 21 6 7 1 5 16 20 3 22 17 14 9 2 11

and nally:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
25 15 6 14 5 9 16 3 19 4 10 17 13 2 11 8 21 20 12 18 22 23 7 1 24
10 14 20 11 16 18 3 25 24 1 23 15 12 9 5 17 7 22 8 19 4 5 13 21 2

Bibliography

BADER, M. The transposition median problem is NP-complete. *Theor. Comput. Sci.*, Elsevier, v. 412, n. 12-14, p. 1099{1110, 2011.

BONA, M. *Combinatorics of Permutations*. [S.l.]: CRC Press, 2022.

CAPRARA, A. The reversal median problem. *INFORMS J. Comput.*, INFORMS, v. 15, n. 1, p. 93{113, 2003.

CORMEN, T. H. *et al. Introduction to algorithms*. [S.l.]: MIT press, 2022.

CUNHA, L. F. I. *et al.* On the computational complexity of closest genome problems. *Discret. Appl. Math.*, North-Holland, v. 274, p. 26{34, 2020.

CUNHA, L. F. I. *et al.* Advancing the transposition distance and diameter through lonely permutations. *SIAM J. Discret. Math.*, Society for Industrial and Applied Mathematics, v. 27, n. 4, p. 1682{1709, 2013.

CUNHA, L. F. I.; PROTTI, F. Genome rearrangements on multigenomic models: Applications of graph convexity problems. *J. Comput. Biol.*, Mary Ann Liebert, Inc., publishers 140 Huguenot Street, 3rd Floor New . . . , v. 26, n. 11, p. 1214{1222, 2019.

ERIKSSON, H. *et al.* Sorting a bridge hand. *Discrete Math.*, v. 241, n. 1-3, p. 289{300, 2001.

FAMPA, M.; LEE, J.; MACULAN, N. An overview of exact algorithms for the euclidean steiner tree problem in n-space. *Int. Trans. Oper. Res.*, Wiley Online Library, v. 23, n. 5, p. 861{874, 2016.

FEIJAO, P.; MEIDANIS, J. Scj: a breakpoint-like distance that simpli es several rearrangement problems. *IEEE/ACM Trans. Comput. Biol. Bioinform.*, IEEE, v. 8, n. 5, p. 1318{1329, 2011.

FERTIN, G. *et al. Combinatorics of genome rearrangements*. [S.l.]: MIT press, 2009.

GRAHAM, R. L. *et al.* Concrete Mathematics: a Foundation for Computer Science. *Computers in Physics*, American Institute of Physics, v. 3, n. 5, p. 106{107, 1989.

HAGHIGHI, M.; SANKOFF, D. Medians seek the corners, and other conjectures. In: SPRINGER. *BMC bioinformatics*. [S.l.], 2012. v. 13, p. 1{7.

KONSTANTINOVA, E. Some problems on Cayley graphs. *Linear Algebra and its Applications*, Elsevier, v. 429, n. 11, p. 2754{2769, 2008.

PENSO, L. D. *et al.* Complexity analysis of P_3-convexity problems on bounded-degree and planar graphs. *Theor. Comput. Sci.*, Elsevier, v. 607, p. 83{95, 2015.

PEVZNER, P. *Computational Molecular Biology: An Algorithmic Approach*. [S.l.]: MIT press, 2000.

PE'ER, I.; SHAMIR, R. The median problems for breakpoints are NP-complete. In: *Elec. Colloq. on Comput. Complexity*. [S.l.: s.n.], 1998. v. 71, n. 5.

POPOV, V. Y. Multiple genome rearrangement by swaps and by element duplications. *Theor. Comput. Sci.*, Elsevier, v. 385, n. 1-3, p. 115{126, 2007.

SANKOFF, D. Genome rearrangement with gene families. *Bioinformatics*, Oxford University Press, v. 15, n. 11, p. 909{917, 1999.

SANKOFF, D.; BLANCHETTE, M. The median problem for breakpoints in comparative genomics. In: SPRINGER. *Computing and Combinatorics: Annual International Conference, COCOON'97, Proceedings 3*. [S.l.], 1997. p. 251{263.

SETZER, S.; STEIDL, G.; TEUBER, T. On vector and matrix median computation. *J. Comput. Appl. Math.*, Elsevier, v. 236, n. 8, p. 2200{2222, 2012.

SHAO, M.; MORET, B. M. On the dcj median problem. In: SPRINGER. *Combinatorial Pattern Matching: 25th Annual Symposium, CPM 2014, Moscow, Russia, June 16-18, 2014. Proceedings 25*. [S.l.], 2014. p. 273{282.

SILVA, H. *et al.* Algorithms for the genome median under a restricted measure of rearrangement. *RAIRO-Oper. Res.*, v. 57, n. 3, p. 1045{1058, 2023.

TANNIER, E.; ZHENG, C.; SANKOFF, D. Multichromosomal median and halving problems under di erent genomic distances. *BMC Bioinformatics*, BioMed Central, v. 10, n. 1, p. 1{15, 2009.

WATTERSON, G. A. *et al.* The chromosome inversion problem. *J. Theor. Biol.*, Elsevier, v. 99, n. 1, p. 1{7, 1982.

XIA, R. *et al.* A median solver and phylogenetic inference based on double-cut-and-join sorting. *J. Comput. Biol.*, Mary Ann Liebert, Inc. 140 Huguenot Street, 3rd Floor New Rochelle, NY 10801 USA, v. 25, n. 3, p. 302{312, 2018.

Acknowledgement

Firstly, I would like to thank my family for all the support given during all my time at the university, helping me in every aspect that I needed during this ve years of my life.

I also would like to thank my advisor, Lu s Felipe Ignacio Cunha, for guiding me in my studies to make this thesis a little bit easier for me. He was very important in this period for not only helping me nd this topic of research and also helping me understand better everything that I need and I hopefully will continue this research in my graduate degree.

Lastly, I want to thank for every friend that helped me in this last ve years of my life. Not only I want to thank those that were very important to make all classes more interesting, but also for those that were by me side in every problem that I had, personal ones and academic ones.