Sorting by Transpositions is Difficult

Laurent Bulteau, Guillaume Fertin, Irena Rusu

Sorting by Transpositions is Difficult

Laurent Bulteau, Guillaume Fertin, Irena Rusu

Laboratoire d'Informatique de Nantes-Atlantique (LINA), UMR CNRS 6241
Université de Nantes, 2 rue de la Houssinière, 44322 Nantes Cedex 3 - France
{Laurent.Bulteau, Guillaume.Fertin, Irena.Rusu}@univ-nantes.fr

Abstract. In comparative genomics, a transposition is an operation that exchanges two consecutive sequences of genes in a genome. The transposition distance, that is, the minimum number of transpositions needed to transform a genome into another, can be considered as a relevant evolutionary distance. The problem of computing this distance when genomes are represented by permutations, called the SORTING BY TRANSPOSITIONS problem (SBT), has been introduced by Bafna and Pevzner [3] in 1995. It has naturally been the focus of a number of studies, but the computational complexity of this problem has remained undetermined for 15 years.
In this paper, we answer this long-standing open question by proving that the SORTING BY TRANSPOSITIONS problem is NP-hard. As a corollary of our result, we also prove that the following problem from [9] is NP-hard: given a permutation π, is it possible to sort π using $d_b(\pi)/3$ permutations, where $d_b(\pi)$ is the number of breakpoints of π?

Introduction

Along with reversals, transpositions are one of the most elementary large-scale operations that can affect a genome. A transposition consists in swapping two consecutive sequences of genes or, equivalently, in moving a sequence of genes from one place to another in the genome. The transposition distance between two genomes is the minimum number of such operations that are needed to transform one genome into the other. Computing this distance is a challenge in comparative genomics, since it gives a maximum parsimony evolution scenario between the two genomes.

The SORTING BY TRANSPOSITIONS problem is the problem of computing the transposition distance between genomes represented by permutations: see [16] for a detailed review on this problem and its variants. Since its introduction by Bafna and Pevzner [3,4], the complexity class of this problem has never been established. Hence a number of studies [4,9,17,19,13,5,15] aim at designing approximation algorithms or heuristics, the best known fixed-ratio algorithm being a 1.375-approximation [13]. Other works [18,9,14,21,13,5] aim at computing bounds on the transposition distance of a permutation. Studies have also been devoted to variants of this problem, by considering, for example, prefix transpositions [12,22,7] (in which one of the blocks has to be a prefix of the sequence), or distance between strings [10,11,25,24,20] (where multiple occurrences of each element are allowed in the sequences), possibly with weighted or prefix transpositions [23,6,1,2,7]. Note also that sorting a permutation by block-interchanges (i.e. exchanges of non-necessarily consecutive sequences) is a polynomial problem [8].

In this paper, we address the long-standing issue of determining the complexity class of the SORTING BY TRANSPOSITIONS problem, by giving a polynomial-time reduction from SAT, thus proving the NP-hardness of this problem. Our reduction is based on the study of transpositions that remove three breakpoints. A corollary of our result is the NP-hardness of the following problem, introduced in [9]: given a permutation π, is it possible to sort π using $d_b(\pi)/3$ permutations, where $d_b(\pi)$ is the number of breakpoints of π?

1 Preliminaries

In this paper, n denotes a positive integer. Let $[\![a\,;\,b]\!] = \{x \in \mathbb{N} \mid a \leq x \leq b\}$, and Id_n be the identity permutation over $[\![0\,;\,n]\!]$. We consider only permutations of $[\![0\,;\,n]\!]$ such that 0 and n are fixed-points. Given a word $u_1\ u_2\ \ldots\ u_l$, a *subword* is a subsequence $u_{p_1}\ u_{p_2}\ \ldots\ u_{p_{l'}}$, where $1 \leq p_1 < p_2 < \ldots < p_{l'} \leq l$. A *factor* is a subsequence of contiguous elements, i.e. a subword with $p_{k+1} = p_k + 1$ for every $k \in [\![1\,;\,l'-1]\!]$.

Definition 1 (Transposition). *Given three integers $0 < i < j < k \leq n$, the* transposition $\tau_{i,j,k}$ *over $[\![0\,;\,n]\!]$ is the following permutation:*

$$\tau_{i,j,k} = \begin{pmatrix} 0 & \cdots & i-1 & i & \cdots & k+i-j-1 & k+i-j & \cdots & k-1 & k & \cdots & n \\ 0 & \cdots & i-1 & j & \cdots & k-1 & i & \cdots & j-1 & k & \cdots & n \end{pmatrix}$$

Let π be a permutation of $[\![0\,;\,n]\!]$. The transposition distance $d_t(\pi)$ *from π to Id_n is the minimum value k for which there exist k transpositions $\tau_1, \tau_2, \ldots, \tau_k$ such that $\pi \circ \tau_k \circ \ldots \circ \tau_2 \circ \tau_1 = Id_n$.*

The transposition $\tau_{i,j,k}$ is the operation that, when it is composed with a permutation, exchanges factors with indices $i, \ldots, j-1$ and $j, \ldots, k-1$, see Figure 1a. The inverse function of $\tau_{i,j,k}$ is also a transposition: $\tau_{i,j,k}^{-1} = \tau_{i,k+i-j,k}$. See Figure 1b for an example of the computation of the transposition distance.

We consider the following problem:

SORTING BY TRANSPOSITIONS PROBLEM [3]
INPUT: A permutation π, an integer k.
QUESTION: Is $d_t(\pi) \leq k$?

Computing the transposition distance has often been linked to studying the breakpoints of a permutation. A breakpoint of π is a pair $(x-1, x)$, $x \in [\![1\,;\,n]\!]$, such that $\pi(x) \neq \pi(x-1)+1$. A transposition can decrease the number of breakpoints of a permutation, $d_b(\pi)$, by at most 3. In this paper we in fact focus on the "simpler" problem of determining whether $d_t(\pi) = d_b(\pi)/3$ for a given permutation π.

2 3-Deletion and Transposition Operations

In this section, we introduce 3DT-instances, which are the cornerstone of our reduction from SAT to the SORTING BY TRANSPOSITIONS problem, since they are used as an intermediate between instances of the two problems.

$$\begin{aligned}\pi &= (\pi_0\pi_1\ldots\pi_{i-1}\underline{\pi_i\ldots\pi_{j-1}}\,\underline{\pi_j\quad\ldots\quad\pi_{k-1}}\pi_k\ldots\pi_n)\\ \pi\circ\tau_{i,j,k} &= (\pi_0\pi_1\ldots\pi_{i-1}\underline{\pi_j\quad\ldots\quad\pi_{k-1}}\,\underline{\pi_i\ldots\pi_{j-1}}\pi_k\ldots\pi_n)\end{aligned}$$

(a)

$$\begin{aligned}\pi &= (0\,\underline{2\,4}\,\underline{3\,1}\,5)\\ \pi\circ\tau_{1,3,5} &= (0\,\underline{3\,1}\,\underline{2\,4}\,5)\\ \pi\circ\tau_{1,3,5}\circ\tau_{1,2,4} &= (0\,1\,2\,3\,4\,5)\end{aligned}$$

(b)

Fig. 1: (a) Representation of a transposition $\tau_{i,j,k}$ for $0 < i < j < k \leq n$ on a general permutation. (b) The transposition distance from $\pi = (0\ 2\ 4\ 3\ 1\ 5)$ to Id_5 is 2: it is at most 2 since $\pi \circ \tau_{1,3,5} \circ \tau_{1,2,4} = Id_5$, and it cannot be less than 2 since $d_t(\pi) \geq d_b(\pi)/3 = 5/3 > 1$.

$$\begin{aligned} I &= a_1\ c_2\ b_1\ b_2\ c_1\ a_2 \quad \text{with} \quad T = \{(a_1, b_1, c_1), (a_2, b_2, c_2)\}\\ I' &= \ \cdot\ b_2\ \cdot\ c_2\ \cdot\ a_2 \quad \text{with} \quad T' = \{(a_2, b_2, c_2)\}\end{aligned}$$

Fig. 2: Two examples of 3DT-instances of span 6. We write $I = \langle \Sigma, T, \psi\rangle$ and $I' = \langle \Sigma', T', \psi'\rangle$. I has an alphabet of size 6, $\Sigma = \{a_1, b_1, c_1, a_2, b_2, c_2\}$, hence ψ is a bijection ($\psi(a_1) = 1$, $\psi(c_2) = 2$, $\psi(b_1) = 3$, etc). I' has an alphabet of size 3, $\Sigma' = \{a_2, b_2, c_2\}$, with $\psi'(b_2) = 2$, $\psi'(c_2) = 4$, $\psi'(a_2) = 6$.

Definition 2 (3DT-instance). *A* 3DT-instance $I = \langle \Sigma, T, \psi\rangle$ *of span* n *is composed of the following elements:*

- Σ*: an alphabet of at most* n *elements;*
- $T = \{(a_i, b_i, c_i) \mid 1 \leq i \leq |T|\}$*: a set of (ordered) triples of elements of* Σ*, partitioning* Σ *(i.e. all elements are pairwise distinct, and* $\bigcup_{i=1}^{|T|}\{a_i, b_i, c_i\} = \Sigma$*);*
- $\psi : \Sigma \to [\![1\,;\,n]\!]$*, an injection.*

The domain *of* I *is the image of* ψ*, that is the set* $L = \{\psi(\sigma) \mid \sigma \in \Sigma\}$*. The* word representation *of* I *is the* n*-letter word* $u_1\ u_2 \ldots u_n$ *over* $\Sigma \cup \{\cdot\}$ *(where* $\cdot \notin \Sigma$*), such that for all* $i \in L$*,* $\psi(u_i) = i$*, and for* $i \in [\![1\,;\,n]\!] - L$*,* $u_i = \cdot$*. For* $\sigma_1, \sigma_2 \in \Sigma$*, we write* $\sigma_1 \prec \sigma_2$ *if* $\psi(\sigma_1) < \psi(\sigma_2)$*, and* $\sigma_1 \lhd \sigma_2$ *if* $\sigma_1 \prec \sigma_2$ *and* $\nexists x \in \Sigma,\ \sigma_1 \prec x \prec \sigma_2$*.*

Two examples of 3DT-instances are given in Figure 2. Note that such instances can be defined by their word representation and by their set of triples T. The empty 3DT-instance, in which $\Sigma = \emptyset$, can be written with a sequence of n dots, or with the empty word ε.

Using the triples in T, we can define a successor function over the domain L:

Definition 3. *Let* $I = \langle \Sigma, T, \psi\rangle$ *be a 3DT-instance with domain* L*. We write* $succ_I : L \to L$ *the function such that, for all* $(a, b, c) \in T$*,* $\psi(a) \mapsto \psi(b)$*,* $\psi(b) \mapsto \psi(c)$*, and* $\psi(c) \mapsto \psi(a)$*.*

Function $succ_I$ is a bijection, with no fixed-points, and such that $succ_I \circ succ_I \circ succ_I$ is the identity over L.

In the example of Figure 2, $succ_I = \begin{pmatrix}1&2&3&4&5&6\\3&6&5&2&1&4\end{pmatrix}$ and $succ_{I'} = \begin{pmatrix}2&4&6\\4&6&2\end{pmatrix}$.

Definition 4. *Let* $I = \langle \Sigma, T, \psi\rangle$ *be a 3DT-instance, and* (a, b, c) *be a triple of* T*. Write* $i = \min\{\psi(a), \psi(b), \psi(c)\}$*,* $j = succ_I(i)$*, and* $k = succ_I(j)$*. The triple* $(a, b, c) \in$

T *is* well-ordered *if we have* $i < j < k$*. In such a case, we write* $\tau[a, b, c, \psi]$ *the transposition* $\tau_{i,j,k}$*.*

An equivalent definition is that $(a, b, c) \in T$ is well-ordered iff one of abc, bca, cab is a subword of the word representation of I. In the example of Figure 2, (a_1, b_1, c_1) is well-ordered in I: indeed, we have $i = \psi(a_1)$, $j = \psi(b_1)$ and $k = \psi(c_1)$, so $i < j < k$. The triple (a_2, b_2, c_2) is also well-ordered in I' ($i = \psi'(b_2) < j = \psi'(c_2) < k = \psi'(a_2)$), but not in I: $i = \psi(c_2) < k = \psi(b_2) < j = \psi(a_2)$. In this example, we have $\tau[a_1, b_1, c_1, \psi] = \tau_{1,3,5}$ and $\tau[a_2, b_2, c_2, \psi'] = \tau_{2,4,6}$.

Definition 5 (3DT-step). *Let* $I = \langle \Sigma, T, \psi \rangle$ *be a 3DT-instance with* $(a, b, c) \in T$ *a well-ordered triple. The* 3DT-step *of parameter* (a, b, c) *is the operation written* $\xrightarrow{(a,\,b,\,c)}$*, transforming* I *into the 3DT-instance* $I' = \langle \Sigma', T', \psi' \rangle$ *such that, with* $\tau = \tau[a, b, c, \psi]$*:*

$$\Sigma' = \Sigma - \{a, b, c\}; \qquad T' = T - \{(a, b, c)\}; \qquad \psi' : \begin{array}{ccc} \Sigma' & \to & [\![1\,;\,n]\!] \\ \sigma & \mapsto & \tau^{-1}(\psi(\sigma)) \end{array}.$$

If the word representation of I is $W\,a\,X\,b\,Y\,c\,Z$, then, after the 3DT-step $I \xrightarrow{(a,\,b,\,c)} I'$, the word representation of I' is $W \cdot Y \cdot X \cdot Z$. Note that a triple that is not well-ordered in I can become well-ordered in I', or vice-versa. In the example of Figure 2, I' can be obtained from I via a 3DT-step: $I \xrightarrow{(a_1,\,b_1,\,c_1)} I'$. Moreover, $I' \xrightarrow{(a_2,\,b_2,\,c_2)} \varepsilon$.

Definition 6 (3DT-collapsibility). *A 3DT-instance* $I = \langle \Sigma, T, \psi \rangle$ *is* 3DT-collapsible *if there exists a sequence of 3DT-instances* $I_k, I_{k-1}, \ldots, I_0$ *such that* $I_k = I$*,* $I_0 = \varepsilon$*, and* $\forall i \in [\![1\,;\,k]\!]\,,\ \exists (a, b, c) \in T,\ I_i \xrightarrow{(a,\,b,\,c)} I_{i-1}$.

In Figure 2, I and I' are 3DT-collapsible, since we have $I \xrightarrow{(a_1,\,b_1,\,c_1)} I' \xrightarrow{(a_2,\,b_2,\,c_2)} \varepsilon$.

3 3DT-collapsibility is NP-Hard to Decide

In this section, we define, for any boolean formula ϕ, a corresponding 3DT-instance I_ϕ. We also prove that I_ϕ is 3DT-collapsible iff ϕ is satisfiable (see Theorem 1).

3.1 Block Structure

The construction of the 3DT-instance I_ϕ uses a decomposition into blocks, defined below. Some triples will be included in a block, in order to define its behavior, while others will be shared between two blocks, in order to pass information. The former are unconstrained, so that we can design blocks with the behavior we need (for example, blocks mimicking usual boolean functions), while the latter need to follow several rules, so that the blocks can easily be arranged together.

Definition 7 (l-block-decomposition). *An* l-block-decomposition $\mathcal{B}$ *of a 3DT-instance* I *of span* n *is an* l*-tuple* $(s_1, \ldots, s_l)$ *such that* $s_1 = 0$*, for all* $h \in [\![1\,;\,l-1]\!]$*,* $s_h < s_{h+1}$ *and* $s_l < n$*. We write* $t_h = s_{h+1}$ *for* $h \in [\![1\,;\,l-1]\!]$*, and* $t_l = n$*.*

Let $I = \langle \Sigma, T, \psi \rangle$ *and* $u_1\,u_2\,\ldots u_n$ *be the word representation of* I*. For* $h \in [\![1\,;\,l]\!]$*, the subword* $u_{s_h+1}\,u_{s_h+2}\,\ldots u_{t_h}$ *where every occurrence of* $\cdot$ *is deleted is called the* block $\mathcal{B}_h$*. For* $\sigma \in \Sigma$*, we write* $block_{I,\mathcal{B}}(\sigma) = h$ *if* $\psi(\sigma) \in [\![s_h + 1\,;\,t_h]\!]$ *(equivalently, if* σ *appears in the word* $\mathcal{B}_h$*). A triple* $(a, b, c) \in T$ *is said to be* internal *if* $block_{I,\mathcal{B}}(a) = block_{I,\mathcal{B}}(b) = block_{I,\mathcal{B}}(c)$*,* external *otherwise.*

Given a 3DT-step $I \xrightarrow{(a,b,c)} I'$, the arrow notation can be extended to an l-block-decomposition $\mathcal{B}$ of I, provided at least one of the following equalities is satisfied: $block_{I,\mathcal{B}}(a) = block_{I,\mathcal{B}}(b)$, $block_{I,\mathcal{B}}(b) = block_{I,\mathcal{B}}(c)$ or $block_{I,\mathcal{B}}(c) = block_{I,\mathcal{B}}(a)$. In this case, with $\tau = \tau[a,b,c,\psi]$, the l-tuple $\mathcal{B}' = (\tau^{-1}(s_1), \ldots, \tau^{-1}(s_l))$ is an l-block-decomposition of I', and we write $(I,\mathcal{B}) \xrightarrow{(a,b,c)} (I',\mathcal{B}')$.

Definition 8 (Variable). *A* variable *A of a 3DT-instance $I = \langle \Sigma, T, \psi \rangle$ is a pair of triples $A = [(a,b,c),(x,y,z)]$ of T. It is* valid *in an l-block-decomposition $\mathcal{B}$ if*

(i) $\exists h_0 \in [\![1\,;\,l]\!]$ *such that* $block_{I,\mathcal{B}}(b) = block_{I,\mathcal{B}}(x) = block_{I,\mathcal{B}}(y) = h_0$
(ii) $\exists h_1 \in [\![1\,;\,l]\!]$, $h_1 \neq h_0$, *such that* $block_{I,\mathcal{B}}(a) = block_{I,\mathcal{B}}(c) = block_{I,\mathcal{B}}(z) = h_1$
(iii) *if* $x \prec y$, *then we have* $x \lhd b \lhd y$
(iv) $a \prec z \prec c$

For such a valid variable A, the block $\mathcal{B}_{h_0}$ containing $\{b,x,y\}$ is called the source *of A, and the block $\mathcal{B}_{h_1}$ containing $\{a,c,z\}$ is called the* target *of A. For $h \in [\![1\,;\,l]\!]$, the variables of which $\mathcal{B}_h$ is the source (resp. the target) are called the* output *(resp. the* input*) of $\mathcal{B}_h$. The 3DT-step $I \xrightarrow{(x,y,z)} I'$ is called the* activation *of A (it requires that (x,y,z) is well-ordered).*

Note that, for any valid variable $A = [(a,b,c),(x,y,z)]$ in $(I,\mathcal{B})$, we have, according to condition (i), $block_{I,\mathcal{B}}(x) = block_{I,\mathcal{B}}(y)$, thus its activation can be written $(I,\mathcal{B}) \xrightarrow{(x,y,z)} (I',\mathcal{B}')$.

Property 1. Let $(I,\mathcal{B})$ be a 3DT-instance with an l-block-decomposition, and A be a variable of I that is valid in $\mathcal{B}$, $A = [(a,b,c),(x,y,z)]$. Then (x,y,z) is well-ordered iff $x \prec y$; and (a,b,c) is not well-ordered.

Definition 9 (Valid context). *A 3DT-instance with an l-block-decomposition $(I,\mathcal{B})$ is a* valid context *if the set of external triples of I can be partitioned into valid variables.*

Let B be a block in a valid context $(I,\mathcal{B})$ (in which $B = \mathcal{B}_h$, for some $h \in [\![1\,;\,l]\!]$), and $(I,\mathcal{B}) \xrightarrow{(d,e,f)} (I',\mathcal{B}')$ be a 3DT-step such that, writing $B' = \mathcal{B}'_h$, we have $B' \neq B$. Then, depending on the triple (d,e,f), we are in one of the following three cases:

– (d,e,f) is an internal triple of B. We write: $\boxed{B} \text{ — } (d,e,f) \text{ → } \boxed{B'}$
– $(d,e,f) = (x,y,z)$ for some output $A = [(a,b,c),(x,y,z)]$ of B. We write: $\boxed{B} \text{ ═ } A \text{ ⟹ } \boxed{B'}$
– $(d,e,f) = (x,y,z)$ for some input $A = [(a,b,c),(x,y,z)]$ of B. We write: $\boxed{B} \text{ —— } A \text{ ⟶ } \boxed{B'}$

The graph obtained from a block B by following exhaustively the possible arcs as defined above (always assuming this block is in a valid context) is called the *behavior graph* of B. Figure 3 illustrates the activation of a valid variable A.

3.2 Basic Blocks

We now define four basic blocks: copy, and, or, and var. They are studied independently in this section, before being assembled in Section 3.3. Each of these blocks is defined by

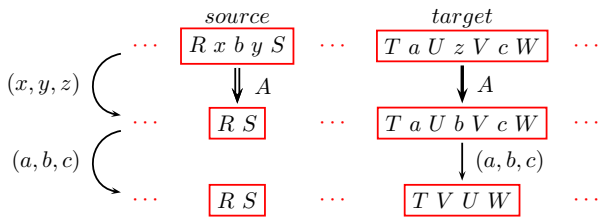

Fig. 3: Activation of a valid variable $A = [(a, b, c), (x, y, z)]$. It can be followed by the 3DT-step $\xrightarrow{(a,b,c)}$, impacting only the target block of A. Dot symbols ($\cdot$) are omitted. We denote by R, S, T, U, V, W some factors of the source and target blocks of A: the consequence of activating A is to allow U and V to be swapped in the target of A.

a word and a set of triples. We distinguish internal triples, for which all three elements appear in a single block, from external triples, which are part of an input/output variable, and for which only one or two elements appear in the block. Note that each external triple is part of an input (resp. output) variable, which itself must be an output (resp. input) of another block, the other block containing the remaining elements of the triple.

We then compute the behavior graph of each of these blocks (it is given here for the block copy, see Figure 4, and in the full version for the other blocks): in each case, we assume that the block is in a valid context, and follow exhaustively the 3DT-steps that can be applied to it. It must be kept in mind that for any variable, it is the state of the source block which determines whether it can be activated, whereas the activation itself affects mostly the target block. It can be verified that each output (resp. input) variable of these blocks satisfy the constraints (i) and (iii) (resp. (ii) and (iv)) of Definition 8.

The block copy This block aims at duplicating a variable: any of the two output variables can only be activated after the input variable has been activated. See Figure 4 for the behavior graph of this block.
Input variable: $A = [(a, b, c), (x, y, z)]$.
Output variables: $A_1 = [(a_1, b_1, c_1), (x_1, y_1, z_1)]$ and $A_2 = [(a_2, b_2, c_2), (x_2, y_2, z_2)]$.
Internal triple: (d, e, f).
Definition:

$$[A_1, A_2] = \mathsf{copy}(A) \quad = \quad a\ y_1\ e\ z\ d\ y_2\ x_1\ b_1\ c\ x_2\ b_2\ f$$

Property 2. In a block $[A_1, A_2] = \mathsf{copy}(A)$ in a valid context, the possible orders in which A, A_1 and A_2 can be activated are (A, A_1, A_2) and (A, A_2, A_1).

The block and This block aims at simulating a conjunction: the output variable can only be activated after both input variables have been activated.
Input variables: $A_1 = [(a_1, b_1, c_1), (x_1, y_1, z_1)]$ and $A_2 = [(a_2, b_2, c_2), (x_2, y_2, z_2)]$.
Output variable: $A = [(a, b, c), (x, y, z)]$.
Internal triple: (d, e, f).
Definition:

$$A = \mathsf{and}(A_1, A_2) \quad = \quad a_1\ e\ z_1\ a_2\ c_1\ z_2\ d\ y\ c_2\ x\ b\ f$$

Property 3. In a block $A = \mathsf{and}(A_1, A_2)$ in a valid context, the possible orders in which A, A_1 and A_2 can be activated are (A_1, A_2, A) and (A_2, A_1, A).

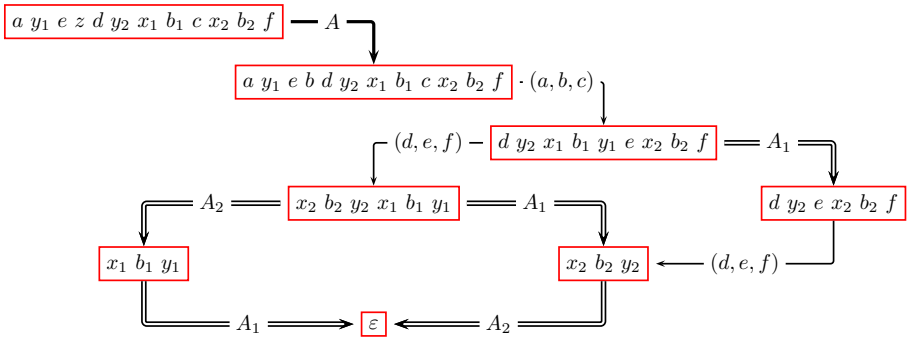

Fig. 4: Behavior graph of the block $[A_1, A_2] = \mathsf{copy}(A)$.

The block or This block aims at simulating a disjunction: the output variable can be activated as soon as any of the two input variables is activated.
Input variables: $A_1 = [(a_1, b_1, c_1), (x_1, y_1, z_1)]$ and $A_2 = [(a_2, b_2, c_2), (x_2, y_2, z_2)]$.
Output variable: $A = [(a, b, c), (x, y, z)]$.
Internal triples: (a', b', c') and (d, e, f).
Definition:

$$A = \mathsf{or}(A_1, A_2) \quad = \quad a_1\ b'\ z_1\ a_2\ d\ y\ a'\ x\ b\ f\ z_2\ c_1\ e\ c'\ c_2$$

Property 4. In a block $A = \mathsf{or}(A_1, A_2)$ in a valid context, the possible orders in which A, A_1 and A_2 can be activated are (A_1, A, A_2), (A_2, A, A_1), (A_1, A_2, A) and (A_2, A_1, A).

The block var This block aims at simulating a boolean variable: in a first stage, only one of the two output variables can be activated. The other needs the activation of the input variable to be activated.
Input variable: $A = [(a, b, c), (x, y, z)]$.
Output variables: $A_1 = [(a_1, b_1, c_1), (x_1, y_1, z_1)]$, $A_2 = [(a_2, b_2, c_2), (x_2, y_2, z_2)]$.
Internal triples: (d_1, e_1, f_1), (d_2, e_2, f_2) and (a', b', c').
Definition:

$$[A_1, A_2] = \mathsf{var}(A) \quad = \quad d_1\ y_1\ a\ d_2\ y_2\ e_1\ a'\ e_2\ x_1\ b_1\ f_1\ c'\ z\ b'\ c\ x_2\ b_2\ f_2$$

Property 5. In a block $[A_1, A_2] = \mathsf{var}(A)$ in a valid context, the possible orders in which A, A_1 and A_2 can be activated are (A_1, A, A_2), (A_2, A, A_1), (A, A_1, A_2) and (A, A_2, A_1).

With such a block, if A is not activated first, one needs to make a choice between activating A_1 or A_2. Once A is activated, however, all remaining output variables are activable.

Assembling the blocks copy, and, or, var.

Definition 10 (Assembling of basic blocks). *An* assembling of basic blocks $(I, \mathcal{B})$ *is composed of a 3DT-instance I and an l-block-decomposition $\mathcal{B}$ obtained by the following process. Create a set of variables $\mathcal{A}$. Define $I = \langle \Sigma, T, \psi \rangle$ by its word representation, as a concatenation of l factors $\mathcal{B}_1\ \mathcal{B}_2\ \ldots\ \mathcal{B}_l$ and a set of triples T, where each $\mathcal{B}_h$ is one of the blocks $[A_1, A_2] = \mathsf{copy}(A)$, $A = \mathsf{and}(A_1, A_2)$, $A = \mathsf{or}(A_1, A_2)$ or $[A_1, A_2] = \mathsf{var}(A)$, with $A_1, A_2, A \in \mathcal{A}$ (such that each $X \in \mathcal{A}$ appears in the input of exactly one block, and in the output of exactly one other block); and where T is the union of the set of internal triples needed in each block, and the set of external triples defined by the variables of $\mathcal{A}$.*

Lemma 1. *Let I' be a 3DT-instance with an l-block-decomposition $\mathcal{B}'$, such that $(I', \mathcal{B}')$ is obtained from an assembling of basic blocks $(I, \mathcal{B})$ after any number of 3DT-steps. Then $(I', \mathcal{B}')$ is a valid context. Moreover, if the set of variables of $(I', \mathcal{B}')$ is empty, then I' is 3DT-collapsible.*

The above lemma justifies the assumption that each block is in a valid context to derive Properties 2 to 5. An assembling of basic blocks is 3DT-collapsible iff there exists a total order, satisfying these properties, in which all its variables can be activated.

3.3 Construction of I_ϕ

Let ϕ be a boolean formula, over the boolean variables $x_1, \ldots, x_m$, given in conjunctive normal form: $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_\gamma$. Each clause C_c ($c \in [\![1\,;\,\gamma]\!]$) is the disjunction of a number of literals, x_i or $\neg x_i$, $i \in [\![1\,;\,m]\!]$. We write q_i (resp. $\bar{q}_i$) for the number of occurrences of the literal x_i (resp. $\neg x_i$) in ϕ, $i \in [\![1\,;\,m]\!]$. We also write $k(C_c)$ for the number of literals appearing in the clause C_c, $c \in [\![1\,;\,\gamma]\!]$. We can assume that $\gamma \geq 2$, that for each $c \in [\![1\,;\,\gamma]\!]$, we have $k(C_c) \geq 2$, and that for each $i \in [\![1\,;\,m]\!]$, $q_i \geq 2$ and $\bar{q}_i \geq 2$ (otherwise, we can always add clauses of the form $(x_i \vee \neg x_i)$ to ϕ, or duplicate the literals appearing in the clauses C_c such that $k(C_c) = 1$). In order to distinguish variables of an l-block-decomposition from $x_1, \ldots, x_m$, we always use the term *boolean variable* for the latter.

The 3DT-instance I_ϕ is defined as an assembling of basic blocks: we first define a set of variables, then we list the blocks of which the word representation of I_ϕ is the concatenation. It is necessary that each variable is part of the input (resp. the output) of exactly one block. Note that the relative order of the blocks is of no importance. We simply try, for readability reasons, to ensure that the source of a variable appears before its target, whenever possible. We say that a variable *represents* a term, i.e. a literal, clause or formula, if it can be activated only if this term is true (for some fixed assignment of the boolean variables), or if ϕ is satisfied by this assignment. We also say that a block *defines* a variable if it is its source block.

The construction of I_ϕ is done as follows (see Figure 5 for an example):
Create a set of variables:

- For each $i \in [\![1\,;\,m]\!]$, create $q_i + 1$ variables representing x_i: X_i and X_i^j, $j \in [\![1\,;\,q_i]\!]$, and $\bar{q}_i + 1$ variables representing $\neg x_i$: $\bar{X}_i$ and $\bar{X}_i^j$, $j \in [\![1\,;\,\bar{q}_i]\!]$.
- For each $c \in [\![1\,;\,\gamma]\!]$, create a variable Γ_c representing the clause C_c.
- Create $m + 1$ variables, A_ϕ and A_ϕ^i, $i \in [\![1\,;\,m]\!]$, representing the formula ϕ.

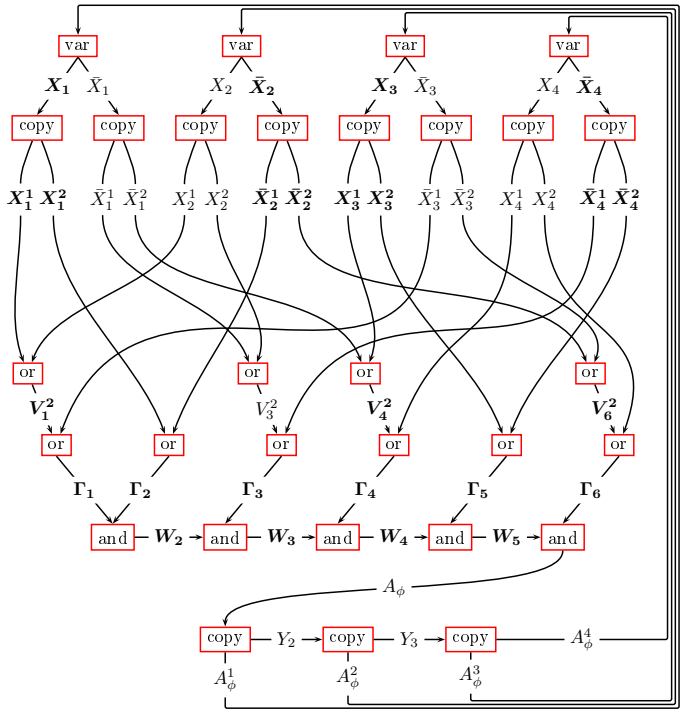

Fig. 5: Schematic diagram of the blocks defining I_ϕ for $\phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee \neg x_4) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (x_3 \vee \neg x_4) \wedge (\neg x_2 \vee \neg x_3 \vee x_4)$. For each variable, we draw an arc between its source and target block. Note that ϕ is satisfiable (e.g. with the assignment $x_1 = x_3 = true$ and $x_2 = x_4 = false$). A set of variables that can be activated before A_ϕ is in bold, they correspond to the terms being true in ϕ for the assignment $x_1 = x_3 = true$ and $x_2 = x_4 = false$.

– We also use a number of intermediate variables, with names U_i^j, $\bar{U}_i^j$, V_c^p, W_c and Y_i.

Start with an empty 3DT-instance ε, and add blocks successively:

– For each $i \in [\![1\,;\,m]\!]$, add the following $q_i + \bar{q}_i - 1$ blocks defining the variables X_i, X_i^j ($j \in [\![1\,;\,q_i]\!]$), and $\bar{X}_i$, $\bar{X}_i^j$ ($j \in [\![1\,;\,\bar{q}_i]\!]$):

$$[X_i, \bar{X}_i] = \mathsf{var}(A_\phi^i);$$

$$[X_i^1, U_i^2]{=}\mathsf{copy}(X_i);\ [X_i^2, U_i^3]{=}\mathsf{copy}(U_i^2);$$
$$\ldots\ [X_i^{q_i-2}, U_i^{q_i-1}]{=}\mathsf{copy}(U_i^{q_i-2});\ [X_i^{q_i-1}, X_i^{q_i}]{=}\mathsf{copy}(U_i^{q_i-1});$$
$$[\bar{X}_i^1, \bar{U}_i^2]{=}\mathsf{copy}(\bar{X}_i);\ [\bar{X}_i^2, \bar{U}_i^3]{=}\mathsf{copy}(\bar{U}_i^2);$$
$$\ldots\ [\bar{X}_i^{\bar{q}_i-2}, \bar{U}_i^{\bar{q}_i-1}]{=}\mathsf{copy}(\bar{U}_i^{\bar{q}_i-2});\ [\bar{X}_i^{\bar{q}_i-1}, \bar{X}_i^{\bar{q}_i}]{=}\mathsf{copy}(\bar{U}_i^{\bar{q}_i-1}).$$

– For each $c \in [\![1\,;\,\gamma]\!]$, let $C_c = \lambda_1 \vee \lambda_2 \vee \ldots \vee \lambda_k$, with $k = k(C_c)$. Let each λ_p, $p \in [\![1\,;\,k]\!]$, be the j-th occurrence of a literal x_i or $\neg x_i$, for some $i \in [\![1\,;\,m]\!]$ and $j \in [\![1\,;\,q_i]\!]$ (resp. $j \in [\![1\,;\,\bar{q}_i]\!]$). We respectively write $L_p = X_i^j$ or $L_p = \bar{X}_i^j$. Add the following $k-1$ blocks defining Γ_c:

$$V_c^2 = \mathsf{or}(L_1, L_2);\ V_c^3 = \mathsf{or}(V_c^2, L_3);$$
$$\ldots\ V_c^{k-1} = \mathsf{or}(V_c^{k-2}, L_{k-1});\ \Gamma_c = \mathsf{or}(V_c^{k-1}, L_k).$$

– Since $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_l$, add the following $l-1$ blocks:

$$W_2{=}\mathsf{and}(\Gamma_1, \Gamma_2);\ W_3{=}\mathsf{and}(W_2, \Gamma_3);$$
$$\ldots\ W_{l-1}{=}\mathsf{and}(W_{l-2}, \Gamma_{l-1});\ A_\phi{=}\mathsf{and}(W_{l-1}, \Gamma_l).$$

– The m copies $A_\phi^1, \ldots, A_\phi^m$ of A_ϕ are defined with the following $m-1$ blocks:

$$[A_\phi^1, Y_2] = \mathsf{copy}(A_\phi);\ [A_\phi^2, Y_3] = \mathsf{copy}(Y_2);$$
$$\ldots [A_\phi^{m-2}, Y_{m-1}] = \mathsf{copy}(Y_{m-2});\ [A_\phi^{m-1}, A_\phi^m] = \mathsf{copy}(Y_{m-1}).$$

Theorem 1. *Let ϕ be a boolean formula, and I_ϕ the 3DT-instance defined above. The construction of I_ϕ is polynomial in the size of ϕ, and ϕ is satisfiable iff I_ϕ is 3DT-collapsible.*

4 Sorting by Transpositions is NP-Hard

In order to transfer our result from 3DT-collapsibility to SORTING BY TRANSPOSITIONS, we need a notion of equivalence between 3DT-instances and permutations, which is introduced here.

Definition 11. *Let $I = \langle \Sigma, T, \psi \rangle$ be a 3DT-instance of span n with domain L, and π be a permutation of $[\![0\,;\,n]\!]$. We say that I and π are equivalent, and we write $I \sim \pi$, if:*

$$\begin{array}{ll} & \pi(0) = 0, \\ \forall v \in [\![1\,;\,n]\!] - L, & \pi(v) = \pi(v-1) + 1, \\ \forall v \in L, & \pi(v) = \pi(succ_I^{-1}(v) - 1) + 1. \end{array}$$

There is no guarantee that any 3DT-instance I has an equivalent permutation π (for example, no permutation is equivalent to $I = a_1\ a_2\ b_1\ b_2\ c_1\ c_2$). However, coming back to our example in Figure 2, we have $I \sim \pi = (0\,5\,2\,1\,4\,3\,6)$, and $I' \sim \pi' = (0\,1\,4\,5\,2\,3\,6)$. More generally, with the following theorem, we show that such a permutation can always be found in the special case of assemblings of basic blocks, which is the case we are interested in.

Theorem 2. *Let I be a 3DT-instance of span n with $\mathcal{B}$ an l-block-decomposition such that $(I, \mathcal{B})$ is an assembling of basic blocks. Then there exists a permutation π_I, computable in polynomial time in n, such that $I \sim \pi_I$.*

With an equivalence $I \sim \pi$, each breakpoint of π can be associated to an element of Σ via ψ, and the triples of breakpoints that may be resolved by a single transposition correspond to the well-ordered triples of T. Moreover, applying such a transposition on π corresponds to operating a 3DT-step on I. These properties, which lead to the following theorem, can be seen on the previous example as summarized below:

$$\begin{array}{ccccc} I & \xrightarrow{(a_1,b_1,c_1)} & I' & \xrightarrow{(a_2,b_2,c_2)} & \varepsilon \\ \pi & \xrightarrow{\circ\,\tau_{1,3,5}} & \pi' & \xrightarrow{\circ\,\tau_{2,4,6}} & Id_6 \\ d_b(\pi) = 6 & & d_b(\pi') = 3 & & d_b(Id_6) = 0 \end{array}$$

Theorem 3. *Let $I = \langle \Sigma, T, \psi \rangle$ be a 3DT-instance of span n with domain L, and π be a permutation of $[\![0\,;\,n]\!]$, such that $I \sim \pi$. Then I is 3DT-collapsible iff $d_t(\pi) = |T| = d_b(\pi)/3$.*

With the previous theorem, we now have all the necessary ingredients to prove the main result of this paper.

Theorem 4. *The* SORTING BY TRANSPOSITIONS *problem is NP-hard.*

Proof. The reduction from SAT is as follows: given any instance ϕ of SAT, create a 3DT-instance I_ϕ, being an assembling of basic blocks, which is 3DT-collapsible iff ϕ is satisfiable (Theorem 1). Then create a permutation π_{I_ϕ} equivalent to I_ϕ (Theorem 2). The above two steps can be achieved in polynomial time.

Finally, set $k = d_b(\pi_{I_\phi})/3 = n/3$: ϕ is satisfiable iff $d_t(\pi_{I_\phi}) = k$ (Theorem 3).

Corollary 1. *The following decision problem from [9] is also NP-complete: given a permutation π of $[\![0\,;\,n]\!]$, is the equality $d_t(\pi) = d_b(\pi)/3$ satisfied?*

Conclusion

In this paper we have proved that the SORTING BY TRANSPOSITIONS problem is NP-hard, thus answering a long-standing question. However, a number of questions remain open. For instance, does this problem admit a polynomial-time approximation scheme? We note that the reduction we have provided does not answer this question, since it is not a linear reduction. Indeed, by our reduction, if a formula ϕ is not satisfiable, it can be seen that we have $d_t(\pi_{I_\phi}) = d_b(\pi_{I_\phi})/3 + 1$.

Also, do there exist some relevant parameters for which the problem is fixed parameter tractable? A parameter that comes to mind when dealing with the transposition distance is the size of the factors exchanged (e.g., the value $\max\{j-i, k-j\}$ for a transposition $\tau_{i,j,k}$). Does the problem become tractable if we bound this parameter? In fact, the answer to this question is no if we bound only the size of the smallest factor, $\min\{j-i, k-j\}$: in our reduction, this parameter is upper bounded by 6 for every transposition needed to sort π_{I_ϕ}, independently of the formula ϕ.

References

1. A. Amir, Y. Aumann, G. Benson, A. Levy, O. Lipsky, E. Porat, S. Skiena, and U. Vishne. Pattern matching with address errors: Rearrangement distances. *J. Comput. Syst. Sci.* 75(6):359-370 (2009)
2. A. Amir, Y. Aumann, P. Indyk, A. Levy, and E. Porat. Efficient computations of ℓ_1 and ℓ_∞ rearrangement distances. In Nivio Ziviani and Ricardo A. Baeza-Yates, editors, *SPIRE*, volume 4726 of *Lecture Notes in Computer Science*, pages 39–49. Springer, 2007.
3. V. Bafna and P. A. Pevzner. Sorting permutations by transpositions. In *SODA*, pages 614–623, 1995.
4. V. Bafna and P. A. Pevzner. Sorting by transpositions. *SIAM J. Discrete Math.*, 11(2):224–240, 1998.
5. M. Benoît-Gagné and S. Hamel. A new and faster method of sorting by transpositions. In B. Ma and K. Zhang, editors, *CPM*, volume 4580 of *Lecture Notes in Computer Science*, pages 131–141. Springer, 2007.

6. D. Bongartz. *Algorithmic Aspects of Some Combinatorial Problems in Bioinformatics*. PhD thesis, RWTH Aachen University, Germany, 2006.
7. B. Chitturi and I. H. Sudborough. Bounding prefix transposition distance for strings and permutations. In *HICSS*, page 468. IEEE Computer Society, 2008.
8. D. A. Christie. Sorting permutations by block-interchanges. *Inf. Process. Lett.*, 60(4):165–169, 1996.
9. D. A. Christie. *Genome Rearrangement Problems*. PhD thesis, University of Glasgow, Scotland, 1998.
10. D. A. Christie and R. W. Irving. Sorting strings by reversals and by transpositions. *SIAM J. Discrete Math.*, 14(2):193–206, 2001.
11. G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *SODA*, pages 667–676, 2002.
12. Z. Dias and J. Meidanis. Sorting by prefix transpositions. In A. H. F. Laender and A. L. Oliveira, editors, *SPIRE*, volume 2476 of *Lecture Notes in Computer Science*, pages 65–76. Springer, 2002.
13. I. Elias and T. Hartman. A 1.375-approximation algorithm for sorting by transpositions. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 3(4):369–379, 2006.
14. H. Eriksson, K. Eriksson, J. Karlander, L. J. Svensson, and J. Wästlund. Sorting a bridge hand. *Discrete Mathematics*, 241(1-3):289–300, 2001.
15. J. Feng and D. Zhu. Faster algorithms for sorting by transpositions and sorting by block interchanges. *ACM Transactions on Algorithms*, 3(3), 2007.
16. G. Fertin, A. Labarre, I. Rusu, É. Tannier, and S. Vialette. *Combinatorics of genome rearrangements*. The MIT Press, 2009.
17. Q.-P. Gu, S. Peng, and Q. M. Chen. Sorting permutations and its applications in genome analysis. *Lectures on Mathematics in the Life Science*, 26:191–201, 1999.
18. S. A. Guyer, L. S. Heath, and J. P. Vergara. Subsequence and run heuristics for sorting by transpositions. Technical report, Virginia State University, 1997.
19. T. Hartman and R. Shamir. A simpler and faster 1.5-approximation algorithm for sorting by transpositions. *Inf. Comput.*, 204(2):275–290, 2006.
20. P. Kolman and T. Waleń. Reversal distance for strings with duplicates: Linear time approximation using hitting set. In T. Erlebach and C. Kaklamanis, editors, *WAOA*, volume 4368 of *Lecture Notes in Computer Science*, pages 279–289. Springer, 2006.
21. A. Labarre. New bounds and tractable instances for the transposition distance. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 3(4):380–394, 2006.
22. A. Labarre. Edit distances and factorisations of even permutations. In D. Halperin and K. Mehlhorn, editors, *ESA*, volume 5193 of *Lecture Notes in Computer Science*, pages 635–646. Springer, 2008.
23. X.-Q. Qi. *Combinatorial Algorithms of Genome Rearrangements in Bioinformatics*. PhD thesis, University of Shandong, China, 2006.
24. A. J. Radcliffe, A. D. Scott, and A. L. Wilmer. Reversals and transpositions over finite alphabets. *SIAM J. Discret. Math.*, 19:224–244, May 2005.
25. D. Shapira and J. A. Storer. Edit distance with move operations. In A. Apostolico and M. Takeda, editors, *CPM*, volume 2373 of *Lecture Notes in Computer Science*, pages 85–98. Springer, 2002.