On recognising words that are squares for the shuffle product

Romeo Rizzi, Stéphane Vialette

On recognizing words that are squares for the shuffle product

Romeo Rizzi[a], Stéphane Vialette[b]

[a] *Computer Science Department, University of Verona, Verona, Italy*
[b] *Université Paris-Est, LIGM (UMR 8049), CNRS, UPEM, ESIEE Paris, ENPC, F-77454, Marne-la-Vallée, France*

Abstract

The shuffle of two words u and v of A^* is the language $u \sqcup\!\sqcup v$ consisting of all words $u_1v_1u_2v_2 \dots u_kv_k$, where $k \geq 0$ and the u_i and the v_i are words of A^* such that $u = u_1u_2 \dots u_k$ and $v = v_1v_2 \dots v_k$. In other words, $u \sqcup\!\sqcup v$ is the finite set of all words obtainable from merging the words u and v from left to right, but choosing the next symbol arbitrarily from u or v. A word $u \in A^*$ is a *square for the shuffle product* if it is the shuffle of two identical words (*i.e.*, $u \in v \sqcup\!\sqcup v$ for some $v \in A^*$).

Whereas it can be decided in polynomial-time whether or not $u \in v_1 \sqcup\!\sqcup v_2$ for given words u, v_1 and v_2 [J.-C. Spehner. *Le Calcul Rapide des Mélanges de Deux Mots*, Theoretical Computer Science, 1986], we show in this paper that it is **NP**-complete to determine whether or not a word u is a square for the shuffle product. The novelty in our approach lies in representing words as *linear graphs*, in which deciding whether or not a given word is a square for the shuffle product reduces to computing some *inclusion-free perfect matching*. Finally, we prove that it is **NP**-complete to determine whether or not an input word is in the shuffle of a word with its reverse.

Keywords: Combinatorics on words; Shuffle operator; Complexity.

1. Introduction

Let A be an alphabet. The *shuffle* $u \sqcup\!\sqcup v$ of words u and v over A is the finite set of all words obtainable from merging the words u and v from left to right, but choosing the next symbol arbitrarily from u or v [14]. The *iterated shuffle* of u is the language $\epsilon \cup u \cup (u \sqcup\!\sqcup u) \cup (u \sqcup\!\sqcup u \sqcup\!\sqcup u) \cup \dots$ These definitions naturally extend to languages. It is known that the shuffle product is a commutative and associative operation, which is also distributive over union. The operations of shuffle and iterated shuffle have been used by many researchers to describe

*Work partially supported ANR project BIRDS JCJC SIMI 2-2010.
URL: `romeo.rizzi@univr.it` (Romeo Rizzi), `vialette@univ-mlv.fr` (Stéphane Vialette)

sequential computation histories of concurrent programs [12]. Interestingly, it was observed in [11] that some aspects of the shuffle product bear strong similarities with genetic recombinations (a non-tree-like event that produces a child sequence by crossing two parent sequences).

For the iterated shuffle, there are basically two kinds of questions that can be addressed depending on whether or not the shuffled element is given as a part of the input. This distinction basically reduces to the two following problems:

- "*Given $u, v \in A^*$, is u in the iterated shuffle of v?*", and
- "*Given $u \in A^*$, is u in the iterated shuffle of some $v \in A^*$?*".

If we focus on only one application of the shuffle product, we are left with the two following problems:

- "*Given $u, v \in A^*$, is $u \in v \sqcup\!\sqcup v$?*", and
- "*Given $u \in A^*$, does there exist $v \in A^*$ such that $u \in v \sqcup\!\sqcup v$?*".

As we shall see soon, these two problems dramatically differ in complexity.

We briefly review the key results that arise in our context. Given words u, v_1 and v_2, it can be tested in $O\left(|u|^2/\log(|u|)\right)$ time whether or not $u \in v_1 \sqcup\!\sqcup v_2$ [21]. (To the best of our knowledge, the first $O(|u|^2)$ time algorithm appeared in [16]. This algorithm can easily be extended to check in polynomial-time whether or not a word is the shuffle of any fixed number of given words.) The shuffle $u \sqcup\!\sqcup v$ of words u and v can be computed in $O\left((|u|+|v|)\binom{|u|+|v|}{|u|}\right)$ time [19]. An improvement and generalization has been proposed in [1], where it is proved that, given words $u_1, u_2, \ldots, u_k$, the shuffle $\sqcup\!\sqcup_{i=1}^{k} u_i$ can be computed in $O\left(\binom{|u_1|+|u_2|+\ldots+|u_k|}{|u_1|,|u_2|,\ldots,|u_k|}\right)$ time.

Given words $u, v_1, v_2, \ldots, v_k \in A^*$, it is however **NP**-complete to decide whether or not $u \in \sqcup\!\sqcup_{i=1}^{k} v_i$ [16, 24], and recently the problem has been proven to be **W[2]**-hard parameterized by k [20]. This remains true even if the alphabet has size 3 [24]. Of particular interest, it is shown in [24] that this problem remains **NP**-complete even if all words $v_1, v_2, \ldots, v_k$ are identical, thereby proving that, for two words u and v, it is **NP**-complete to decide whether or not u is in the iterated shuffle of v. Again, this remains true even if the alphabet has size 3.

Strongly related is the problem of shuffling a word with its reverse. Let $u \in A^*$. It is easily seen that if there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v^R$, then u is an *Abelian square* (*i.e.*, $u = v\,v'$, where v' is a permutation of v). Of particular interest, it is shown in [17] that if u is a binary Abelian square, then there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v^R$, thereby proving that it is polynomial-time solvable to decide whether or not a binary word is the shuffle of another word with its reverse. The equivalence is, however, no longer true for larger alphabets. For example, *abcabc* is a ternary Abelian square that cannot be written as an element of $v \sqcup\!\sqcup v^R$ for any word $v \in A^*$ [17].

In this paper, our approach lies in the use of *linear graphs* (*i.e*, those graphs with sets of vertices equipped with some total order), in which deciding whether or not a given word is in the shuffle of another word with itself (or its reverse) reduces to computing some constrained perfect matching. We show that, given $u \in A^*$, it is **NP**-complete to decide whether or not u is the shuffle of some word $v \in A^*$ with itself (*i.e.*, does there exist some $v \in A^*$ such that $u \in v \sqcup\!\sqcup v$?). Notice that this result was first claimed by K. Iwama [9], but it turns out that the proof has a serious flaw [3]. Buss and Soltys [5] have also, very recently, independently solved this problem (as an answer to Erickson [8] on the Stack Exchange discussion board) using a similar constrained perfect matching approach. The two proofs use, however, different reductions, and it is worth noticing that Buss and Soltys managed to obtain the stronger result that the question remains hard for a size-9 alphabet. Furthermore, we complete the positive result of [17] by proving that, given $u \in A^*$, it is **NP**-complete to decide whether or not u is the shuffle of some word $v \in A^*$ with its reverse (*i.e.*, does there exist some $v \in A^*$ such that $u \in v \sqcup\!\sqcup v^R$?).

2. Definitions

We follow standard terminology on words [6]. Let A be an alphabet. The *empty word* is denoted ϵ. A word $v = a_1 a_2 \dots a_n \in A^n$ with $a_i \in A$ is a *subsequence* of $u \in A^*$ if there exist $n+1$, not necessarily distinct and possibly empty, words $u_1, u_2, \dots, u_{n+1} \in A^*$ such that $u_1\, a_1\, u_2\, a_2 \dots u_n\, a_n\, u_{n+1} = u$, and we write $v \preceq u$ to denote this fact. The *reverse* of $v = a_1\, a_2 \dots a_n$ with $a_i \in A$ is the word $v^R = a_n \dots a_2\, a_1$.

The *shuffle* (also sometimes referred to as the *ordinary shuffle* in the literature) of two words u and v, denoted $u \sqcup\!\sqcup v$, is the language of all words w such that $w = u_1\, v_1\, u_2\, v_2 \dots u_n\, v_n$, where $u_i, v_i \in A^*$, $u_1\, u_2 \dots u_n = u$, and $v_1\, v_2 \dots v_n = v$. It may be defined inductively on words by $u \sqcup\!\sqcup \epsilon = u$, $\epsilon \sqcup\!\sqcup u = u$, and $ua \sqcup\!\sqcup vb = (u \sqcup\!\sqcup vb)a \cup (ua \sqcup\!\sqcup v)b$. A word $u \in A^*$ is said to be a *square for the shuffle product* if it is the shuffle of two identical words (*i.e.*, $u \in v \sqcup\!\sqcup v$ for some $v \in A^*$). The *iterated shuffle* of u is the language $\epsilon \cup u \cup (u \sqcup\!\sqcup u) \cup (u \sqcup\!\sqcup u \sqcup\!\sqcup u) \cup \dots$

For two words $u = a_1\, a_2 \dots a_n$ and $v = b_1\, b_2 \dots b_n$, $a_i, b_i \in A$, of the same length, we denote their *perfect shuffle*[1] $u \sqcup\!\sqcup_p v = a_1\, b_1\, a_2\, b_2 \dots a_n\, b_n$. Note that $u \sqcup\!\sqcup_p v$ needs not equal to $v \sqcup\!\sqcup_p u$. Moreover, $(u \sqcup\!\sqcup_p v)^R = v^R \sqcup\!\sqcup_p u^R$. Abusing notation, it will be useful to allow $|u| = |v| + 1$, where $u = a_1\, a_2 \dots a_{n+1}$ and $v = b_1\, b_2 \dots b_n$, in which case we define $u \sqcup\!\sqcup_p v = a_1\, b_1\, a_2\, b_2 \dots a_n\, b_n\, a_{n+1}$.

For a graph G, we denote $\mathbf{V}(G)$ as the set of vertices and $\mathbf{E}(G)$ as the set of edges. Let $u = u_1\, u_2 \dots u_n \in A^n$ be a word on some alphabet A. The *graph associated to* u, denoted G_u, is defined by $\mathbf{V}(G_u) = \{1, 2, \dots, n\}$ and $\mathbf{E}(G_u) = \{\{i, j\} : i \neq j \ \wedge \ u_i = u_j\}$. (We write (i, j) for an edge of $\mathbf{E}(G_u)$

[1] Note that some authors - see for example [17] - use $\sqcup\!\sqcup$ to denote the perfect shuffle operator and $\bigsqcup$ (larger symbol) to denote the ordinary shuffle operator. To avoid confusion, we prefer to use $\sqcup\!\sqcup_p$ for the perfect shuffle operator.

if it is clear from the context that $i < j$, and $\{i, j\}$ otherwise.) Clearly, G_u is the disjoint union of cliques, one clique for each distinct letter. Recall that two edges of a graph are *independent* if they do not share a common vertex, and that a *matching* $\mathcal{M}$ in G is a set of pairwise independent edges. A matching is *perfect* if it covers all the vertices of the graph. In case the set of vertices is equipped with a total order, a matching $\mathcal{M}$ is said to be *inclusion-free* if there do not exist (independent) edges (i, j) and (k, ℓ) in $\mathcal{M}$ such that $i < k < \ell < j$. Similarly, a matching $\mathcal{M}$ is said to be *precedence-free* (resp. *crossing-free*) if there do not exist (independent) edges (i, j) and (k, ℓ) in $\mathcal{M}$ such that $i < j < k < \ell$ (resp. $i < k < j < \ell$). Finally, a perfect matching $\mathcal{M}$ is said to be a *tower* if it is both precedence-free and crossing-free.

3. Being a square for the shuffle product

This section is devoted to proving hardness of recognizing those words that are squares for the shuffle product. Notice that, as observed in [8], the special case where each letter occurs at most 4 times easily reduces to 2-SAT. This approach generalizes to general strings but does not give a polynomial-time decision procedure as clauses may contain up to $\max\{|u|_a : a \in \Sigma\} - 1$ literals.

At the heart of our approach for proving hardness is the following property. (See Fig. 1 for an illustration.)

Lemma 1. *Let $u \in A^*$ for some alphabet A, and G_u be the corresponding linear graph. Then, u is a square for the shuffle product if and only if there exists an inclusion-free perfect matching in G_u.*

PROOF. Indeed, suppose first that there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v$. Let $2n = |u|$. Fix the occurrences in u of the two copies of v and write $I^1 = \{i^1_1, i^1_2, \ldots, i^1_n\}$, $i^1_1 < i^1_2 < \ldots < i^1_n$, for the positions in u of the first copy of v and $I^2 = \{i^2_1, i^2_2, \ldots, i^2_n\}$, $i^2_1 < i^2_2 < \ldots < i^2_n$, for the positions in u of the second copy of v. It is easily seen that $\mathcal{M} = \{\{i^1_j, i^2_j\} : 1 \leq j \leq n\}$ is a subset of $\mathbf{E}(G_u)$. Furthermore, $\mathcal{M}$ is a perfect matching since $I^1 \cap I^2 = \emptyset$, and $I^1 \cup I^2 = \{1, 2, \ldots 2n\}$. It is also inclusion-free. Indeed, if it were not the case then there would exist two edges $e = \{i^1_j, i^2_j\}$ and $e' = \{i^1_k, i^2_k\}$, $j < k$, in $\mathcal{M}$ such that $i^1_j < i^1_k < i^2_k < i^2_j$. This is a contradiction since $i^2_k > i^2_j$ if $k > j$.

Conversely, let $\mathcal{M} \subseteq \mathbf{E}(G_u)$ be an inclusion-free perfect matching of G_u. Let $I^1 = \{j : \exists k > j \text{ with } (j, k) \in \mathcal{M}\}$ and $I^2 = \{k : \exists j < k \text{ with } (j, k) \in \mathcal{M}\}$. Then $|I^1| = |I^2|$. Write $I^1 = \{i^1_1, i^1_2, \ldots, i^1_n\}$ and $I^1 = \{i^2_1, i^2_2, \ldots, i^2_n\}$ with $i^1_1 < i^1_2 < \ldots < i^1_n$ and $i^2_1 < i^2_2 < \ldots < i^2_n$. Let $v = u_{i^1_1} u_{i^1_2} \ldots u_{i^1_n}$ and $v' = u_{i^2_1} u_{i^2_2} \ldots u_{i^2_n}$. We claim that $v = v'$. Indeed, suppose, aiming at a contradiction, that $v \neq v'$. Let $j - 1$ be the length of the largest common prefix of v and v'. Then it follows that there exist i^1_k and i^2_ℓ such that $i^1_j < i^1_k < i^2_j < i^2_\ell$ with $\{i^1_j, i^2_\ell\} \in \mathcal{M}$ and $\{i^1_k, i^2_j\} \in \mathcal{M}$, and hence $\mathcal{M}$ is not inclusion-free. This is the sought contradiction. □

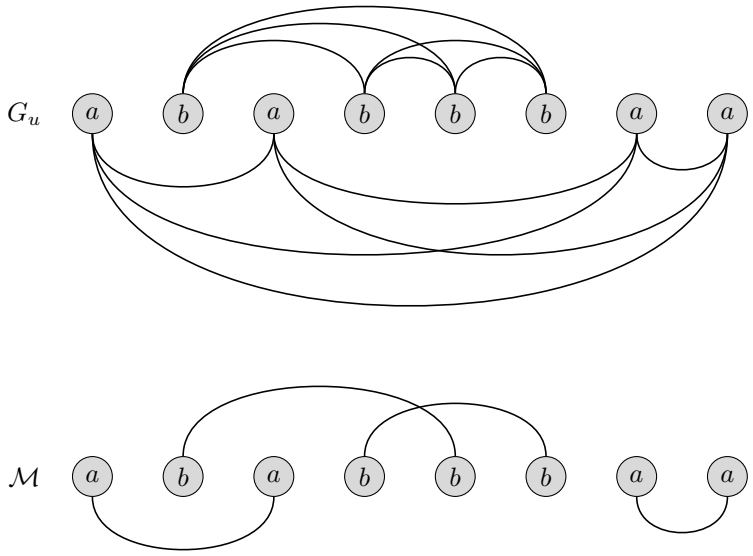


Figure 1: The linear graph G_u of $u = ababbbaa$ together with an inclusion-free perfect matching $\mathcal{M}$. The perfect matching $\mathcal{M}$ denotes $u \in v \sqcup\!\sqcup v$ for $v = abba$ and reads as $u = \begin{array}{cccccccc} a & b & & b & & & & a \\ & & a & & b & b & a & \end{array}$ with the first copy of v on top and the second on bottom.

The following easy result is used in upcoming Proposition 4. Two points are worth noting First, Proposition 2 is most probably folklore, but we do not know any reference. Second, requiring equal length input words in the proof of Proposition 4 is actually not a crucial property but it greatly simplifies the exposition by avoiding introducing length specific gadgets.

Proposition 2. *The* Longest Common Subsequence *problem is* **NP***-complete even if the input consists of binary words that are all of the same length.*

Proof. We reduce from the Longest Common Subsequence problem for binary words which is known to be **NP**-complete [15]. Let $U = \{u_1, u_2, \ldots, u_m\}$, $u_i \in \{0,1\}^*$ for $1 \leq i \leq m$, be our input collection of words, and k be a positive integer. Write $|u_i| = n_i$ for $1 \leq i \leq m$ and assume $n_1 \leq n_2 \leq \ldots \leq n_m$. Define a new collection of words $V = \{v_1, v_2, \ldots, v_m\}$, $u_i \in \{0,1\}^*$ for $1 \leq i \leq m$, as follows:

$$v_i = u_i\, 0^{n_m - k + 1}\, 1^{n_m - n_i}.$$

It is easily seen that V is composed of words that are all of the same length $2n_m - k + 1$. Define $k' = n_m + 1$.

We claim that there exists a subsequence of length k common to all words of U if and only if there exists a common subsequence of length k' common to all words of V.

Suppose first that there exists a subsequence w of length k common to all words of U. It is immediate to check that $w\,0^{n_m-k+1}$ is a word of length $k' = n_m + 1$ common to all words of V.

Conversely, suppose that there exists a subsequence w of length $k' = n_m + 1$ common to all words of V. We first observe that $v_m = u_m\,0^{n_m-k+1}$ as $1^{n_m-n_m}$ reduces to the empty word. Hence, since $k' > n_m = |u_m|$ it follows that w terminates with a sequence of 0's. Therefore we may safely assume that $w = w'\,0^{n_m-k+1}$ and that the 0^{n_m-k+1} suffix of w occurs in the $0^{n_m-k+1}\,1^{n_m-n_i}$ suffix of every v_i, $1 \leq i \leq m$. Then it follows that w' is a word of length $k' - (n_m - k + 1) = k$ that occurs as a subsequence in every u_i, $1 \leq i \leq m$. □

The next easy lemma turns out to be extremely useful for Proposition 4.

Lemma 3. *Any word $u \in \{0,1\}^{p+q}$ with $|u|_0 = p$ and $|u|_1 = q$ is a subsequence of $(0^p1)^q0^p$.*

We are now in position to prove our main result.

Proposition 4. *It is* **NP***-complete to determine whether or not a word is a square for the shuffle product.*

Proof. The problem is certainly in **NP**. To prove hardness, we propose a polynomial-time reduction from the **NP**-complete Longest Common Subsequence problem for binary words (01-LCS for short) which is defined as follows: Given a collection of words $U = \{u_1, u_2, \ldots, u_m\}$, $u_i \in \{0,1\}^*$ for $1 \leq i \leq m$, and positive integers p and q, decide whether there exists a subsequence of size $p+q$ with p letters 0 and q letters 1 common to all sequences of U [15]. According to Lemma 2, we may assume that $|u_i| = |u_j|$ for $1 \leq i < j \leq m$. According to Lemma 2, we may assume that $|u_i| = |u_j|$ for $1 \leq i < j \leq m$. We write (U, p, q) for such an instance of 01-LCS.

Let (U, p, q), $U = \{u_1, u_2, \ldots, u_m\}$, $u_i \in \{0,1\}^*$ for $1 \leq i \leq m$ and $|u_i| = |u_j| = n$ for $1 \leq i < j \leq m$, be an arbitrary instance of 01-LCS. Let us construct from this instance a word w over the $(3m+6)$-size alphabet A defined as follows:

$$A = \{0,1\} \dot\cup \{s, s'\} \dot\cup \{t, t'\} \dot\cup \{x_i, y_i, z_i : 1 \leq i \leq m\}.$$

The word target w is defined by

$$w = W_s\ W_1\ W_2\ \ldots\ W_m\ W_t,$$

where W_s, W_1, W_2, …, W_m and W_t are words in A^* (we refer to these words as our gadget words).

Let us now describe the various gadget words. The source and sink gadget words, denoted W_s and W_t respectively, are defined as follows

$$W_s = s'\ 0^{pq}\ s'\ s\ (0^p\ 1)^q\ 0^p\ s$$
$$W_t = t\ (0^p\ 1)^q\ 0^q\ t\ t'\ 0^{pq}\ t',$$

where s, s', t and t' are four letters that do not occur in any other gadget word. To shorten the exposition, we shall speak about the $(0^p\ 1)^q\ 0^p$-*factor* of W_s (resp. W_t) to designate the factor of W_s (resp. W_t) that occurs between the two occurrences of the letter s (resp. t), and about the 0^{pq}-*factor* of W_s (resp. W_t) to designate the factor of W_s (resp. W_t) that occurs between the two occurrences of the letter s' (resp. t'). For each input word $u_i = u_{i,1}\, u_{i,2}\, \ldots\, u_{i,n}$, the associated gadget word W_i is defined by

$$W_i = x_i\, W_i'\, x_i\, y_i\, W_i'\, y_i,$$

where

$$W_i' = u_{i,1}\, z_i\, u_{i_2}\, z_i\, \ldots\, u_{i,n-1}\, z_i\, u_{i,n}.$$

In other words, $W_i' = u_i \sqcup\!\sqcup_p z_i{}^n$. (Notice that letters x_i, y_i and z_i only occur in the gadget word W_i.)

With the corresponding linear graph G_w in mind, for any letter $a \in A$ occurring only twice in w, we shall write (a,a)-*edge* to designate (without any ambiguity) the unique edge connecting the two occurrences of letter a in G_w.

A schematic description of the reduction is depicted in Figure 2 and a full example involving 3 binary words in given in Appendix (subsections *Describing the* 01-LCS *instance* and *Full example for shuffled square words*).

We now claim that there exists a common subsequence with p letters 0 and q letters 1 common to all sequences of U if and only if w is a square for the shuffle product. It will be convenient to see the reduction as a flow-like procedure, where some piece of information (the common subsequence) emitted from gadget W_s (the source) propagates lossless to gadget W_t (the sink) going through all gadgets W_i, $1 \leq i \leq m$ (every such gadget being associated to an input word of our input instance of 01-LCS).

For the forward direction, suppose that there exists a common subsequence v of the words $u_1, u_2, \ldots, u_m$ with p occurrences of the letter 0 and q occurrence of the letter 1. Write $k = p+q$ and $v = v_1 v_2 \ldots v_k$. According to Lemma 1, it is enough to show that G_w has an inclusion-free perfect matching. Now, observe that v is a subsequence of both the $(0^p\ 1)^q\ 0^p$-factor of W_s and the $(0^p\ 1)^q\ 0^p$-factor of W_t (see Lemma 3). Furthermore, by hypothesis (and construction), v also occurs in each gadget word W_i', $1 \leq i \leq m$. Fix any occurrence of v as a subsequence in (i) the $(0^p\ 1)^q\ 0^p$-factor of W_s, (ii) the $(0^p\ 1)^q\ 0^p$-factor of W_t, and (iii) in every W_i' gadget word, $1 \leq i \leq m$ (if W_i' contains several occurrences of v as a subsequence, we fix any but the same occurrence in the two W_i' gadget words.) We can now turn to defining an inclusion-free perfect matching $\mathcal{M}$ of G_w. This perfect matching contains both *intra-gadget edges* (*i.e.*, edges connecting two identical letters that occur in the same gadget word), and *inter-gadget edges* (*i.e.*, edges connecting two identical letters that occur in distinct - but consecutive - gadget words).

Intra-gadget edges:

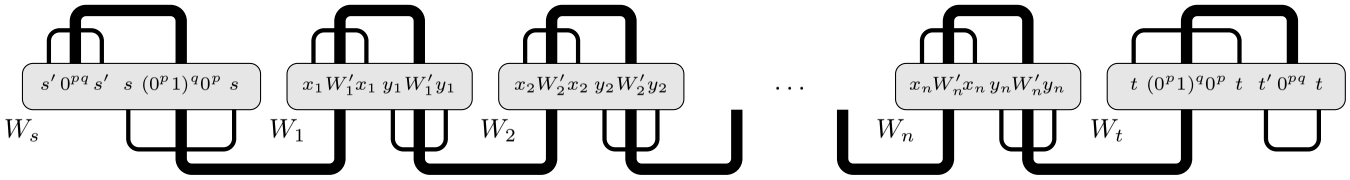


Figure 2: Schematic description of the reduction of Proposition4. It is convenient to see the reduction as a flow-like procedure, where some piece of information (the common subsequence) emitted from gadget W_s (the source) propagates lossless to gadget W_t (the sink) going through all gadgets W_i, $1 \leq i \leq m$ (every such gadget being associated to an input word of our input instance of 01-LCS). Here, bold edges denote pairwise crossing edges while thin edges emphasises some (a, a)-edges (*i.e.*, the unique edges connecting the two occurrences of some letter a in G_w).

- $\mathcal{M}$ contains (i) the (s, s)-edge, (ii) the (s', s')-edge, and (iii) pq pairwise crossing edges that connect the leftmost pq letters 0 of W_s to the pq letters 0 of the $(0^p\ 1)^q\ 0^p$-factor of W_s that do not correspond to the chosen occurrence of the common subsequence v in the $(0^p\ 1)^q\ 0^p$-factor of W_s.

- For every $1 \leq i \leq m$, $\mathcal{M}$ contains (i) the (x_i, x_i)-edge,, (ii) the (y_i, y_i)-edge, (iii) $n-1$ pairwise crossing edges connecting the $n-1$ occurrences of letter z_i in the leftmost W'_i gadget word to the $n-1$ occurrences of letter z_i in the rightmost W'_i gadget word, and (iv) $n-p-q$ pairwise crossing edges connecting the $n-p-q$ letters of the leftmost W'_i gadget word that do not correspond to the chosen occurrence of v in W'_i to the $n-p-q$ letters of the rightmost W'_i gadget word that do not correspond to the occurrence of v in W'_i.

- $\mathcal{M}$ contains (i) the (t, t)-edge, (ii) the (t', t')-edge, and (iii) pq pairwise crossing edges connecting the pq letters 0 of the $(0^p\ 1)^q\ 0^p$-factor of W_s that do not correspond to the chosen occurrence of the common subsequence v in the $(0^p\ 1)^q\ 0^p$-factor of W_t to the last pq letters 0 of W_t.

Inter-gadget edges:

- $\mathcal{M}$ contains $p+q$ pairwise crossing edges connecting the $p+q$ letters of the $(0^p\ 1)^q\ 0^p$-factor of W_s that correspond to the chosen occurrence of the common subsequence v in the $(0^p\ 1)^q\ 0^p$-factor of W_s to the $p+q$ letters of the leftmost W'_1 gadget word that correspond to the chosen occurrence of the common subsequence v in W'_1.

- For every $1 \leq i < m$, $\mathcal{M}$ contains $p+q$ pairwise crossing edges connecting the $p+q$ letters of the rightmost W'_i gadget word that correspond to the chosen occurrence of v in W'_i to the $p+q$ letters of the leftmost W'_{i+1} gadget word that correspond to the chosen occurrence of v in W'_{i+1}.

- $\mathcal{M}$ contains pq pairwise crossing edges connecting the $p+q$ letters of the rightmost W'_m gadget word that correspond to the chosen occurrence of the common subsequence v in W'_m to the $p+q$ letters of $(0^p\ 1)^q\ 0^p$-factor of W_t that correspond to the chosen occurrence of the common subsequence v.

It can be easily verified that $\mathcal{M}$ is a perfect inclusion-free matching. Indeed, all inter-gadget edges in $\mathcal{M}$ are pairwise crossing (and inter-gadget edges in $\mathcal{M}$ connect consecutive gadgets words), and no two intra-gadget edges are in inclusion.

For the reverse direction, suppose that w is a square for the shuffle product. Once again, according to Lemma 1, this amount to saying that G_w has an inclusion-free perfect matching $\mathcal{M}$. We begin with a sequence of easy observations. First, observe that the letters s, s', t, t', x_i $(1 \leq i \leq m)$, and y_i $(1 \leq i \leq m)$ occur exactly twice in w, and hence the $2m+4$ edges connecting these vertices two by two have to be in $\mathcal{M}$ since it is perfect. In other words, $\mathcal{M}$

contains the (s,s)-edge, the (s',s')-edge, the (t,t)-edge, the (t',t')-edge, and the (x_i,x_i)-edge and the (y_i,y_i)-edge for $1 \leq i \leq m$. Let us now focus on the source W_s gadget word. Since both the (s,s)-edge and the (s',s')-edge are in $\mathcal{M}$, then it follows that (i) no edge in $\mathcal{M}$ can connect two identical letters occurring in the $(0^p\ 1)^q\ 0^p$-factor of W_s, and (ii) no edge in $\mathcal{M}$ can connect two identical letters occurring in the 0^{pq}-factor of W_s. Then it follows that $\mathcal{M}$ contains pq pairwise crossing edges connecting all letters from the 0^{pq}-factor of W_s to pq letters 0 occurring in the $(0^p\ 1)^q\ 0^p$-factor of W_s (otherwise $\mathcal{M}$ would not be inclusion-free). Similar considerations apply to W_t yielding (i) no edge in $\mathcal{M}$ can connect two identical letters occurring in the $(0^p\ 1)^q\ 0^p$-factor of W_t, and (ii) no edge in $\mathcal{M}$ can connect two identical letters occurring in the 0^{pq}-factor of W_t. Then it follows that $\mathcal{M}$ contains pq pairwise crossing edges connecting pq letters 0 occurring in the $(0^p\ 1)^q\ 0^p$-factor of W_t to all letters from the 0^{pq}-factor of W_t (otherwise $\mathcal{M}$ would not be inclusion-free). We now turn to the W_i gadget words. For every $1 \leq i \leq m$, $\mathcal{M}$ has to contain both the (x_i,x_i)-edge and (y_i,y_i)-edge, and hence $\mathcal{M}$ contains $n-1$ pairwise crossing edges connecting the $n-1$ letters z_i of the leftmost W'_i gadget word to the $n-1$ letters z_i of the rightmost W'_i gadget word (otherwise one edge connecting two letters z_i would be included in the (x_i,x_i)-edge or in the (y_i,y_i)-edge).

According to the above, $p+q$ letters of W_s have to be involved in some inter-gadget edges of $\mathcal{M}$. But $\mathcal{M}$ contains the (x_1,x_1)-edge, an hence these $p+q$ inter-gadget edges are pairwise crossing and each connect a letter occurring in the $(0^p\ 1)^q\ 0^p$-factor of W_s to a letter in the leftmost W'_1 gadget word. Now, since the $2(n-1)$ occurrences of letter z_i are involved in $n-1$ pairwise crossing edges, then it follows that $\mathcal{M}$ contains $n-p-q$ pairwise crossing edges connecting the $n-p-q$ letters $u_{1,j}$ of the leftmost W'_1 gadget word that are not involved in the leftmost $p+q$ inter-gadget edges to $n-p-q$ letters $u_{1,j}$ of the rightmost W'_1 gadget word. Of particular importance, these $n-p-q$ edges have to be position preserving, *i.e.*, each edge connect a letter $u_{1,j}$ of the leftmost W'_1 gadget word to a letter $u_{1,j}$ of the rightmost W'_1 gadget word for a same position j. At this point, $p+q$ letters of the rightmost W'_1 gadget are yet to be involved in $\mathcal{M}$. Since $\mathcal{M}$ contains both the (y_1,y_1)-edge and the (x_2,x_2)-edge, the only solution is that $\mathcal{M}$ contains $p+q$ pairwise crossing edges connecting letters from the rightmost W'_1 gadget word to the leftmost W'_2 gadget word. The same process continues until $p+q$ pairwise crossing edges connecting the rightmost W'_m gadget word to the W_t sink gadget word.

It follows from the examination of $\mathcal{M}$ that the $p+q$ pairwise crossing edges connecting $p+q$ letters of the $(0^p\ 1)^q\ 0^p$-factor of W_s to $p+q$ letters of the leftmost W'_1 gadget word define a word with p letters 0 and q letters 1 that occurs as a subsequence in each input word u_i. □

It is worth noticing that S.C. Li and M. Li [13] proved that computing the largest inclusion-free matching in a linear graph is **NP**-complete. However, their quite complicated proof involves general linear graphs and not linear graphs that are unions of cliques, and hence cannot be used in the context of shuffling words (the above proposition may, however, be seen as a much simpler proof of Li and

Li's result).

We also notice that in the proof of Proposition 4, some letters occur exactly twice in the constructed word w. Clearly, in this case, w cannot be the shuffle of $k \geq 3$ identical copies of some word $v \in A^*$. We have thus proved the following.

Proposition 5. *It is* **NP***-complete to decide whether or not a word $u \in A^*$ is in the iterated shuffle of some word $v \in A^*$ with $u \neq v$.*

Another easy easy corollary of Proposition 4 is worth mentioning.

Proposition 6. *For a palindromic word $u \in A^*$, it is* **NP***-complete to determine whether or not there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v$. This remains true even if one restricts v to be palindromic as well.*

Proof. Let $u \in A^*$ and let a be any letter not in A. Consider the palindromic word $w = uaau^R$. We claim that u is a square for the shuffle product if and only if w is a square for the shuffle product.

Suppose first that u is a square for the shuffle product. Then, there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v$. We check at once that $w \in uau^R \sqcup\!\sqcup uau^R$, and hence w is a square for the shuffle product.

Conversely, suppose that w is a square for the shuffle product. Since $a \notin A$, then $w \in xay \sqcup\!\sqcup xay$ for some words $x, y \in A^*$. Therefore, $u \in x \sqcup\!\sqcup x$ (and $u^R \in y \sqcup\!\sqcup y$), and hence u is a square for the shuffle product. □

Anticipating Section 4, we observe that Proposition 7 can be rephrased as follows.

Proposition 7. *For a palindromic word $u \in A^*$, it is* **NP***-complete to determine whether or not there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v^R$. This remains true even if one restricts v to be palindromic as well.*

The case of binary alphabets (in a slightly altered question) is considered in [4]. For a word u, let $f(u)$ be the largest integer m such that there exist a word v of length m such that u contains a subsequence in $v \sqcup\!\sqcup v$. Let $f(n, A) = \min\{f(u) : u \text{ is of length } n, \text{ over alphabet } A\}$. It is shown in [4] that $2f(n, \{0, 1\}) = n - o(n)$ using the regularity lemma for words. In other words, any binary word of length n can be split into two identical subsequences and, perhaps, a remaining subsequence of length $o(n)$. A similar result is proven for k identical subsequences of a word over an alphabet with at most k letters. An additional outcome of Lemma 1 is worth mentioning in this context.

Proposition 8. *Let $u \in A^*$. There is a polynomial-time approximation scheme (PTAS) for computing the longest subsequence of u that is a square for the shuffle product.*

Proof. Jiang gave a polynomial-time approximation scheme for computing the largest inclusion-free matching in a linear graph [10]. The result now follows from Lemma 1. □

4. Being the shuffle of a word with its reverse

As we mentioned in Section 1, for a given u over some binary alphabet A, it is polynomial-time solvable to determine whether or not there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v^R$. Indeed, if there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v^R$, then u is an *Abelian square* (*i.e.*, $u = v\,v'$, where v' is a permutation of v). Furthermore, if u is a binary Abelian square, then there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v^R$ [17]. The equivalence is, however, no longer true for larger alphabets (the words *abcabc* is an example of a ternary Abelian square that cannot be written as an element of $v \sqcup\!\sqcup v^R$ for any word v). As a complementary result to [17], we use again linear graphs to show that the problem is **NP**-complete for large alphabets. We need the following equivalence which can be seen as the analogous of Lemma 1 (see Fig. 3 for an illustration.); the lemma corrects a mistake in the former version of the paper [18].

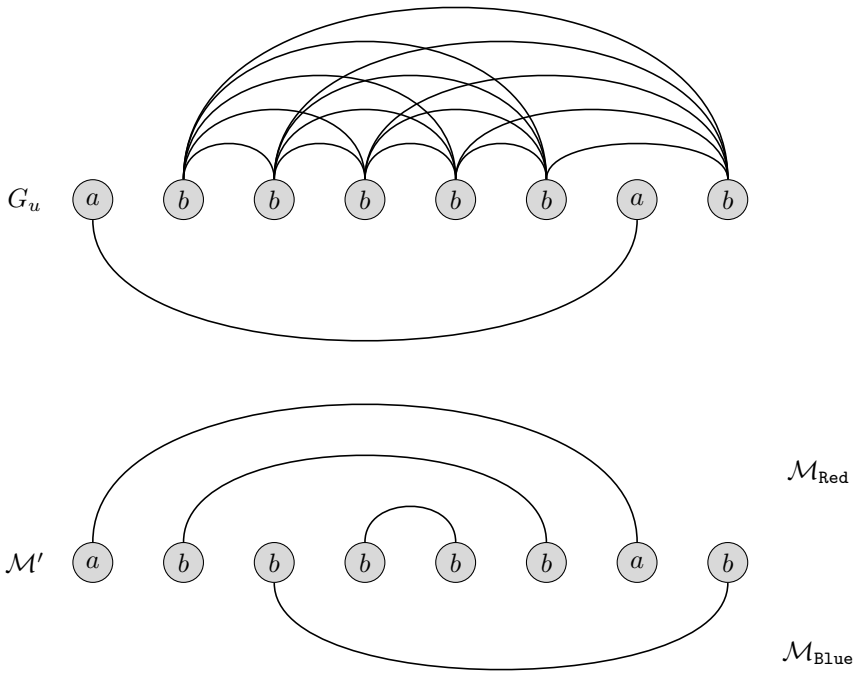


Figure 3: The linear graph G_u of $u = abbbbbab$ together with and a precedence-free perfect matching $\mathcal{M}'$ that can be partitioned into two towers. The perfect matching $\mathcal{M}'$ denotes $u \in v \sqcup\!\sqcup v^R$ for $v = abba$ and reads as $u = \overset{a\quad b\qquad b\qquad\qquad\qquad b}{\underset{\qquad b\qquad\quad b\quad b\quad a}{\qquad\qquad\qquad\qquad\qquad\qquad}}$ with the first copy of v on top and the second on bottom.

Lemma 9. *Let $u \in A^{2n}$ for some alphabet A, and G_u be the corresponding linear graph. Then, there exists $v \in A^n$ such that $u \in v \sqcup\!\sqcup v^R$ if and only if there exists a precedence-free perfect matching in G_u that can be partitioned into two towers.*

Proof. Suppose that there exists $v = v_1 v_2 \dots v_n \in A^n$ such that u is in the shuffle of v with its reverse. Then the first half of u must contain some prefix of v, say $v_1 v_2 \dots v_k$, and the second half of u must contain the remaining suffix of v, say $v_{k+1} v_{k+2} \dots v_n$. Then it follows that the second half of u must contain (in the remaining positions) some prefix of v, reversed. But a straightforward counting argument shows that that this prefix must be $v_1 v_2 \dots v_k$. Therefore, the first half of u must contain the remaining symbols of v, reversed. This shows that the first half of u is just $v_1 v_2 \dots v_k$ shuffled with $(v_{k+1} v_{k+2} \dots v_n)^R$, and the second half of u is just $v_k v_{k+1} \dots v_n$ shuffled with $(v_1 v_2 \dots v_k)^R$. Construct a perfect matching $\mathcal{M}$ of G_u as follows: Join every letter of $v_1 v_2 \dots v_k$ in the first half of u to the corresponding letter of $(v_1 v_2 \dots v_k)^R$ in the second half of u (call this set $\mathcal{M}_{\texttt{Red}}$), and every letter of $(v_{k+1} v_{k+2} \dots v_n)^R$ in the first half of u to the corresponding letter of $v_{k+1} v_{k+2} \dots v_n$ in the second half of u (call this set $\mathcal{M}_{\texttt{Blue}}$). Clearly, $\mathcal{M}$ is precedence-free (every edge connect a letter in the first half of u to a letter in the second half of u) and can partitioned into two towers ($\mathcal{M}_{\texttt{Red}}$ and $\mathcal{M}_{\texttt{Blue}}$)

Conversely, suppose that there exists a precedence-free perfect matching $\mathcal{M}$ in G_u that can be partitioned into two towers, say $\mathcal{M}_{\texttt{Red}}$ and $\mathcal{M}_{\texttt{Blue}}$ with $\mathcal{M} = \mathcal{M}_{\texttt{Red}} \dot\cup \mathcal{M}_{\texttt{Blue}}$. Since $\mathcal{M}$ is precedence-free, every edge of $\mathcal{M}$ connects a letter in the first half of u to a letter in the second half of u. Let us say that a letter of u is `Red` (resp. `Blue`) if is part of an edge in $\mathcal{M}_{\texttt{Red}}$ (resp. $\mathcal{M}_{\texttt{Blue}}$) so that we may define $u_{\texttt{Red}}$ (resp. $u_{\texttt{Blue}}$) to be the subsequence of u made of all `Red` (resp. `Blue`) letters. Let $u^1_{\texttt{Red}}$ (resp. $u^2_{\texttt{Red}}$) be the subsequences of u made of all `Red` letters in the first (resp. second) half of u, and $u^1_{\texttt{Blue}}$ (resp. $u^2_{\texttt{Blue}}$) be the subsequences of u made of all `Blue` letters in the first (resp. second) haft of u. We claim that $u^1_{\texttt{Red}}\, u^2_{\texttt{Blue}} = (u^1_{\texttt{Blue}}\, u^2_{\texttt{Red}})^R$, thereby proving the lemma as $u \in (u^1_{\texttt{Red}}\, u^2_{\texttt{Blue}}) \sqcup\!\sqcup (u^1_{\texttt{Blue}}\, u^2_{\texttt{Red}})$ is immediate by construction. Indeed, since $\mathcal{M}_{\text{Red}}$ and $\mathcal{M}_{\text{Blue}}$ are towers, we have $u^1_{\texttt{Red}} = (u^2_{\texttt{Red}})^R$ and $u^1_{\texttt{Blue}} = (u^2_{\texttt{Blue}})^R$, and hence $u^1_{\texttt{Red}}\, u^2_{\texttt{Blue}} = (u^2_{\texttt{Red}})^R\, (u^1_{\texttt{Blue}})^R = (u^1_{\texttt{Blue}}\, u^2_{\texttt{Red}})^R$. □

We now turn to proving hardness. Whereas the general idea of the reduction is the same as in the proof of Proposition 4, the proof turns out to be a little bit more complex.

Proposition 10. *For a word $u \in A^*$, it is* **NP***-complete to determine whether or not there exists $v \in A^*$ such that $u \in v \sqcup\!\sqcup v^R$.*

Proof. The problem is certainly in **NP**. Again, to prove hardness, we propose a polynomial-time reduction from the **NP**-complete Longest Common Subsequence problem for binary words (01-LCS): Given a collection of words $U = \{u_1, u_2, \dots, u_m\}$, $u_i \in \{0,1\}^*$ for $1 \le i \le m$, and positive integers p and q, decide whether there exists a subsequence of size $p + q$ with p letters 0 and

q letters 1 common to all sequences of U [15]. According to Lemma 2, we may assume that $|u_i| = |u_j|$ for $1 \leq i < j \leq m$. We write (U, p, q) for such an instance of 01-LCS.

Let (U, p, q), $U = \{u_1, u_2, \ldots, u_m\}$, $u_i \in \{0,1\}^*$ for $1 \leq i \leq m$ and $|u_i| = |u_j| = n$ for $1 \leq i < j \leq m$, be an arbitrary instance of 01-LCS. Let $Z = \{z_{i,j} : 1 \leq i \leq m \ \wedge \ 1 \leq j \leq n\}$ be an alphabet of $m(n+1)$ new letters, and, for every $1 \leq i \leq m$, define the word $z_i = z_{i,1}\, z_{i_2}\, \ldots\, z_{i,n+1}$ of length n.

Let us construct from this instance a word w over the $(mn(n+3)+7)$-size alphabet A defined as follows:

$$\begin{aligned} A = &\{0,1\} \,\dot\cup \\ &\{r, b_1, b_2, t_1, t_2\} \,\dot\cup \\ &\{x_i, y_i : 1 \leq i \leq m\} \,\dot\cup \\ &\{z_{i,j} : 1 \leq i \leq m \ \wedge \ 1 \leq j \leq n+1\}. \end{aligned}$$

The word w is defined by

$$w = V_t\, V_m\, V_{m-1}\, \ldots\, V_1\, V_s\, W_s\, W_1\, W_2\, \ldots W_m\, W_t$$

where V_s, W_s, V_t, W_t, $W_1, W_2, \ldots, W_m$, $V_1, V_2, \ldots, V_m$ are words in A^* (we refer to these words as our gadget words).

Let us now describe the various gadget words. The two source gadget words, denoted V_s and W_s, are defined as follows:

$$\begin{aligned} V_s &= r\ b_1\ 0^{pq}\ b_2 \\ W_s &= r\ b_2\ (0^p\ 1)^q\ 0^p\ x_1\ b_1 \end{aligned}$$

where r, b_1 and b_2 are three letters that do not occur in any other gadget word (x_1 will appear soon in gadget word V_1). The two sink gadget words, denoted V_t and W_t, are defined as follows:

$$\begin{aligned} V_t &= t_1\ t_2\ (0^p\ 1)^q\ 0^p \\ W_t &= t_1\ y_m\ 0^{pq}\ t_2 \end{aligned}$$

where t_1 and t_2 are two letters that do not occur in any other gadget word (y_m will appear soon in gadget word V_m). For each input word $u_i = u_{i,1}\, u_{i,2}\, \ldots\, u_{i,n}$, the two associated gadget word, denoted V_i and W_i, are defined by:

$$\begin{aligned} V_i &= x_i\ y_i\ (z_i \sqcup\!\sqcup_p u_i)^R \\ W_i &= \begin{cases} z_1 \sqcup\!\sqcup_p u_1 & \text{if } i = 1 \\ x_i\ y_{i-1}\ (z_i \sqcup\!\sqcup_p u_i) & \text{if } n > 1 \end{cases} \end{aligned}$$

Notice that letters x_i, y_i and $z_{i,j}$, $1 \leq j \leq n+1$, only occur in the gadget words

V_i and W_i. By construction we have

$$\begin{aligned}
|V_s| &= pq + 3 \\
|W_s| &= pq + p + q + 4 \\
|V_t| &= pq + p + q + 2 \\
|W_t| &= pq + 3 \\
|V_i| &= 2n + 3 \quad (1 \leq i \leq m) \\
|W_1| &= 2n + 1 \\
|W_i| &= 2n + 3 \quad (2 \leq i \leq m),
\end{aligned}$$

and hence $|w| = 4pq + 2(p + q) + 2m(2n + 3) + 10$. Furthermore,

$$\begin{aligned}
|V_t\ V_m\ V_{m-1}\ \ldots\ V_1\, V_s| &= |W_s\, W_1\ W_2\ \ldots W_m\ W_t| \\
&= pq + p + q + m(2n + 3) + 5 \\
&= \frac{|w|}{2}
\end{aligned}$$

(*i.e.*, $V_t\ V_m\ V_{m-1}\ \ldots\ V_1\, V_s$ is the first half of w and $W_s\, W_1\ W_2\ \ldots W_m\ W_t$ is the second half of w).

As in the proof of Proposition 4, with the corresponding linear graph G_w in mind, for any letter $a \in A$ occurring only twice in w, we shall write (a, a)-*edge* to designate (without any ambiguity) the unique edge connecting the two occurrences of letter a in G_w.

A schematic description of the reduction is depicted in Figure 4 and a full example involving 3 binary words in given in Appendix (subsections *Describing the* 01-LCS *instance* and *Full example for shuffled square words with reverse*).

We now claim that there exists a common subsequence with p letters 0 and q letters 1 common to all sequences of U if and only if there exists $v \in \{0, 1\}^*$ such that $w \in v \sqcup\!\sqcup v^R$. Albeit less obvious than in Proposition 4, it will be nevertheless convenient to see the reduction as a flow-like procedure, where some piece of information (the common subsequence) emitted from the source (the V_s and W_s word gadgets), propagates "*lossless*" to the target (the V_t and W_t gadgets) going through all gadgets V_i and W_i, $1 \leq i \leq m$ (every such pair of gadgets being associated to an input word of our input instance of 01-LCS). Note that "*lossless*" has here to be understood in the broadest sense of the term since, as we shall see soon, dealing with precedence-free perfect matchings that can be partitioned into two towers challenge us to consider in the constructed word w both a input string and its reverse, and hence both the common subsequence (viewed as some data) and its complement with respect to each input words.

For the forward direction, suppose that there exists a common subsequence v of the words $u_1, u_2, \ldots, u_m$ with p occurrences of the letter 0 and q occurrence of the letter 1. Write $k = p + q$ and $v = v_1 v_2 \ldots v_k$. According to Lemma 9, it is enough to show that G_w has a precedence-free perfect matching in G_w that can be partitioned into two towers. Now, observe that v is a subsequence of both the $(0^p\ 1)^q\ 0^p$-factor of W_s and the $(0^p\ 1)^q\ 0^p$-factor of V_t. Furthermore, by hypothesis (and construction),

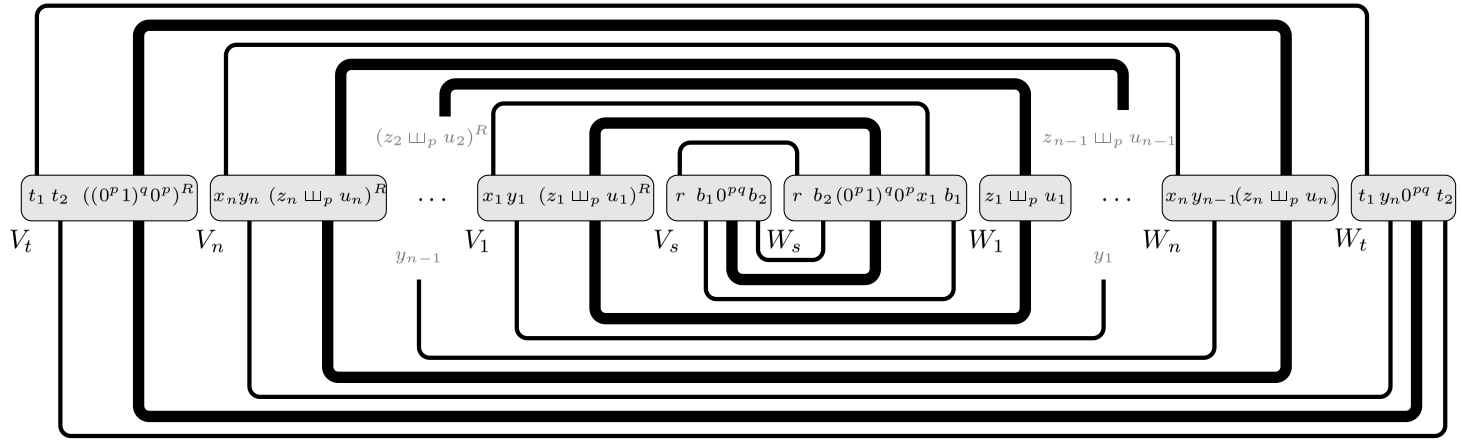


Figure 4: Schematic description of the reduction of Proposition10. It is convenient to see the reduction as a flow-like procedure, where some piece of information (the common subsequence) emitted from gadget V_s (the source) propagates lossless to gadget W_t (the sink) going through all gadgets W_s, V_t, V_i, W_i, $1 \leq i \leq m$ (every pair of gadget $(V_i, W_i$ being associated to an input word of our input instance of 01-LCS). Here, bold edges denote pairwise included edges while thin edges emphasises (a, a)-edge (*i.e.*, the unique edge connecting the two occurrences of some letter a in G_w).

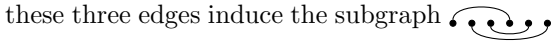
- v^R occurs in each gadget word V_i, $1 \leq i \leq m$,
- v occurs in each gadget word W_i, $1 \leq i \leq m$.

Fix any occurrence of v as a subsequence in (i) the $(0^p\ 1)^q\ 0^p$-factor of W_s, (ii) the $(0^p\ 1)^q\ 0^p$-factor of W_t, and (iii) in every W_i gadget word, $1 \leq i \leq m$. Furthermore, fix any occurrence of v^R as a subsequence in every V_i gadget word, $1 \leq i \leq m$. We can now turn to defining a precedence-free perfect matching $\mathcal{M}$ in G_w that can be partitioned into two towers. For the sake of presentation, let us use the colours `Red` and `Blue`. The matching $\mathcal{M}$ is precisely defined as follows:

- $\mathcal{M}$ contains the (r, r)-edge. These two edges are coloured `Red`.
- $\mathcal{M}$ contains the (b_1, b_1)-edge and the (b_2, b_2)-edge. These two edges are coloured `Blue`.
- $\mathcal{M}$ contains pq edges that connect the pq letters 0 of V_s to the pq letters 0 of the $(0^p\ 1)^q\ 0^p$-factor of W_s that do not correspond to the chosen occurrence of the common subsequence v in the $(0^p\ 1)^q\ 0^p$-factor of W_s. These pq edges form a tower and are coloured `Blue`.
- $\mathcal{M}$ contains the (x_i, x_i)-edge for $1 \leq i \leq m$. These m edges form a tower (by construction) and are coloured `Red`.
- $\mathcal{M}$ contains the (y_i, y_i)-edge for $1 \leq i \leq m$. These m edges form a tower (by construction) and are coloured `Blue`.
- $\mathcal{M}$ contains the $p+q$ edges that connect the p letters 0 and q letters 1 that correspond to the chosen occurrence of the common subsequence v in the $(0^p\ 1)^q\ 0^p$-factor of W_s to the p letters 0 and q letters 1 that correspond to the chosen occurrence of the common subsequence v^R in V_1. These m edges form a tower (by construction) and are coloured `Red`.
- $\mathcal{M}$ contains the $p + q$ edges that connect the p letters 0 and q letters 1 that correspond to the chosen occurrence of the common subsequence v in the $(0^p\ 1)^q\ 0^p$-factor of W_i, $1 \leq i \leq m-1$ to the p letters 0 and q letters 1 that correspond to the chosen occurrence of the common subsequence v^R in V_{i+1}. These $p + q$ edges form a tower (by construction) and are coloured `Red`.
- For every $1 \leq i \leq m$, $\mathcal{M}$ contains the $n + 1$ $(z_{i,j}, z_{i,j})$-edges together with $n-p-q$ edges that connect that connect the p letters 0 and q letters 1 that do not correspond to the chosen occurrence of the common subsequence v^R in V_i to the p letters 0 and q letters 1 that do not correspond to the chosen occurrence of the common subsequence v in W_i. These $2n-p-q+1$ edges form a tower (by construction) and are coloured `Blue`.

- $\mathcal{M}$ contains the $p+q$ edges that connect the p letters 0 and q letters 1 that correspond to the chosen occurrence of the common subsequence v in the $(0^p\ 1)^q\ 0^p$-factor of W_m to the p letters 0 and q letters 1 that correspond to the chosen occurrence of the common subsequence v^R in the $(0^p\ 1)^q\ 0^p$-factor of W_t. These $p+q$ edges form a tower (by construction) and are coloured `Red`.
- $\mathcal{M}$ contains the pq letters 0 that do not correspond to the chosen occurrence of the common subsequence v^R in the $(0^p\ 1)^q\ 0^p$-factor of W_s to the pq letters 0 if V_t. These pq edges form a tower (by construction) and are coloured `Blue`.
- $\mathcal{M}$ contains the (t_1, t_1)-edge. This edge is coloured `Red`.
- $\mathcal{M}$ contains the (t_2, t_2)-edge. This edge is coloured `Blue`.

It is a tedious simple matter to check that $\mathcal{M}$ is a precedence-free perfect matching in G_w that can be partitioned into two towers (`Red` and Blue above).

For the reverse direction, suppose that there exists v such that $w \in v \sqcup\!\sqcup v^R$. Once again, according to Lemma 9, this amount to saying that G_w has a precedence-free perfect matching $\mathcal{M}$ that can be decomposed into two towers. Let us colour the edges of first tower with the colour `Red` and the edges of the second tower with the colour `Blue`. We begin with a sequence of easy observations. First, observe that $\mathcal{M}$ being precedence-free, every edge in $\mathcal{M}$ has to join a letter in the first half of w to a letter in the second half of w (the first half of w terminates at the first occurrence of the letter b_2 in V_s). We now observe that the letters r, b_1 and b_2 occur exactly twice in w, and hence the 3 edges connecting these vertices two by two have to be in $\mathcal{M}$ since it is perfect. In other words, $\mathcal{M}$ contains the (r, r)-edge, the (b_1, b_1)-edge, and the (b_2, b_2)-edge. But these three edges induce the subgraph  as w contains the subsequence $r\, b_1\, b_2\, r\, b_2\, b_1$. Then it follows that the the (b_1, b_1)-edge and the (b_2, b_2)-edge are part of the same tower in $\mathcal{M}$ and hence are coloured with the same colour, say `Blue`, and that the (r, r)-edge is part of the other tower in $\mathcal{M}$, and hence is coloured with the other colour, say `Red`. Let us now focus on V_s in its entirety. Recall that $V_s = r\ b_1\ 0^{pq}\ b_2$ and that b_2 is the last letter of the first half of w. Therefore, since $\mathcal{M}$ is a precedence-free matching no two letters 0 of V_s are connected by an edge in $\mathcal{M}$. We show that every letter 0 of V_s is connected to a letter 0 of W_s in $\mathcal{M}$. Indeed, any edge of $\mathcal{M}$ involving a letter 0 of V_s is crossing with the (r, r)-edge of $\mathcal{M}$ and hence the pq letters 0 of V_s are involved in pq `Blue` pairwise stacking edges in $\mathcal{M}$. But the (b_1, b_1)-edge is also coloured `Blue`, and hence every letter 0 of V_s is connected to a letter 0 of W_s in $\mathcal{M}$ (b_1 is indeed the last letter of W_s). Turning now to W_s, p letters 0 and q letters 1 of the $0^p(10^p)^q$-factor are not involved in the above-mentioned `Blue` pairwise stacking edges of $\mathcal{M}$. But W_s is in the second half of w and hence neither two letters 0 of W_s nor two letters 1 of W_s are connected by an edge in $\mathcal{M}$. Furthermore, any edge of $\mathcal{M}$ involving any of these $p+q$ letters is crossing with the (b_1, b_1)-edge that is coloured `Blue`, and hence has to be coloured `Red`.

Now, thanks to the (x_1, x_1)-edge (the letter x_1 occurs exactly twice in w, and hence the (x_1, x_1)-edge has to be in $\mathcal{M}$ since it is perfect), we see that these $p+q$ letters of W_s are connected in $\mathcal{M}$ to letters of V_1. We now turn to V_1. By construction the $n+1$ letters $z_{1,j}$ have to be involved in $n+1$ pairwise stacking edges with the $n+1$ letters $z_{1,j}$ in W_1. But these $n+1$ edges together with the (y_1, y_1)-edge are all crossing with the (x_1, x_1)-edge that is coloured `Red`, and hence have to coloured `Blue`. Now, thanks to the `Blue` (y_1, y_1)-edge, the remaining $n-p-q$ letters 0 or 1 of V_1 are also involved in this `Blue` stacking with letters in W_1. Furthermore, thanks to the $z_{1,j}$ letters, these edges match position at position. The same process continues until $p+q$ pairwise nested edges connecting the rightmost V_t gadget word to the W_t sink gadget word.

It follows from the examination of $\mathcal{M}$ that the $p+q$ pairwise nested edges connecting $p+q$ letters of the $(0^p\ 1)^q\ 0^p$-factor of W_s to $p+q$ letters of the V_1 gadget word define a word with p letters 0 and q letters 1 that occurs as a subsequence in each input word u_i. □

It is worth mentioning that if we drop the partition into 2 towers constraint in Lemma 9, we are left with a polynomial-time solvable problem as finding a maximum size precedence-free matching in a linear graph is polynomial-time solvable [7, 22, 23] (this problem is nothing but Abelian square recognition).

5. Conclusion and Open problems

In this paper we have used a (union of cliques) linear graph framework to show that it is **NP**-complete to recognize those words that squares for the shuffle product. Using the same framework, we have proved that recognizing those words that are the shuffle of another word with its reverse is also **NP**-complete.

There are a number of further directions of investigation in this general subject. We mention one open problem that is, in our opinion, the most interesting. How hard is the problem of detecting squares for the shuffle product for bounded alphabet words? It is proved in [5] that the problem is **NP**-complete for an alphabet with 9 symbols (it is claimed that this can be improved to 7 letters). Notice that it is claimed without proof in [2] (Fact 2 Subsection 2.2) that detecting squares for the shuffle product is **NP**-complete for binary words. This result – that would be an important improvement over [5] and Proposition 4 – is yet to be confirmed.

References

[1] C. Allauzen, *Calcul efficace du shuffle de k mots*, Tech. report, Institut Gaspard Monge, Université de Marne-la-Vallée, 2000.

[2] H. Aoki, R. Uehara, and K. Yamazaki, *Expected length of longest common subsequences of two biased random strings and its application*, Tech. Report 1185, RIMS Kokyuroku, 2001.

[3] K. Iwama (author of [9]), 2012, Personal communication.

[4] M. Axenovich, Y. Person, and S. Puzynina, *A regularity lemma and twins in words*, J. Comb. Theory, Ser. A **120** (2013), no. 4, 733–743.

[5] S. Buss and M. Soltys, *Unshuffling a square is NP-hard*, Journal of Computer and System Sciences **80** (2014), no. 4, 766–776.

[6] C. Choffrut and J. Karhumäki, *Combinatorics of words, in g. rozenberg and a. salomaa (eds), handbook of formal languages*, Springer-Verlag, 1997.

[7] C. Erdong, Y. Linji, and Y. Hao, *Improved algorithms for largest cardinality 2-interval pattern problem*, Journal of Combinatorial Optimization **13** (1983), 263–275.

[8] J. Erickson, *How hard is unshuffling a string?*, http://cstheory.stackexchange.com/q/34 (version: 2010-12-01).

[9] K. Iwama, *Unique decomposability of shuffled strings: A formal treatment of asynchronous time-multiplexed communication*, Proc. 15th Annual ACM Symposium on Theory of Computing (STOC), Boston, Massachusetts, USA, ACM, 1983, pp. 374–381.

[10] M. Jiang, *A PTAS for the weighted 2-interval pattern problem over the preceding-and-crossing model*, 1st Annual International Conference on Combinatorial Optimization and Applications (COCOA'07), Xi'an, Shaanxi, China (A. Dress, Y. Xu, and B. Zhu, eds.), Lecture Notes in Computer Science, vol. 4616, 2007, pp. 378–387.

[11] J.D. Kececioglu and D. Gusfield, *Reconstructing a history of recombinations from a set of sequences*, Discrete Applied Mathematics **88** (1998), no. 1-3, 239–260.

[12] T. Kimura, *An algebraic system for process structuring and interprocess communication*, Proc. 8th annual ACM symposium on Theory of computing (STOC), Hershey, Pennsylvania, USA (A.K. Chandra, D. Wotschke, E.P. Friedman, and M.A.Harrison, eds.), ACM, 1976, pp. 92–100.

[13] S.C. Li and M. Li, *On two open problems of 2-interval patterns*, Theoretical Computer Science **410** (2009), no. 24-25, 2410–2423.

[14] M. Lothaire, *Applied combinatorics on words*, Series: Encyclopedia of Mathematics and its Applications, no. 105, Cambridge university press, 2005.

[15] D. Maier, *The complexity of some problems on subsequences and supersequences*, Journal of the ACM **25** (1978), no. 2, 322–336.

[16] A. Mansfield, *On the computational complexity of a merge recognition problem*, Discrete Applied Mathematics **5** (1983), 119–122.

[17] D. Henshall N. Rampersad and J. Shallit, *Shuffling and unshuffling*, Bulletin of the EATCS (2012), 131–142.

[18] R. Rizzi and S. Vialette, *On recognizing words that are squares for the shuffle product*, Computer Science - Theory and Applications - 8th International Computer Science Symposium in Russia, CSR 2013, Ekaterinburg, Russia (Arseny M. Shur Andrei A. Bulatov, ed.), Lecture Notes in Computer Science, vol. 7913, Springer, 2013, pp. 235–245.

[19] J.-C. Spehner, *Le calcul rapide des melanges de deux mots*, Theoretical Computer Science (1986), 171–203.

[20] R. van Bevern, R. Bredereck, L. Bulteau, C. Komusiewicz, N. Talmon, and G.J. Woeginger, *Precedence-constrained scheduling problems parameterized by partial order width*, Discrete Optimization and Operations Research - 9th International Conference, DOOR 2016, Vladivostok, Russia, September 19-23, 2016, Proceedings, 2016, pp. 105–120.

[21] J. van Leeuwen and M. Nivat, *Efficient recognition of rational relations*, Information Processing Letters **14** (1982), no. 1, 34–38.

[22] S. Vialette, *On the computational complexity of* 2*-interval pattern matching problems*, Theoretical Computer Science **312** (2004), no. 2-3, 223–249.

[23] ———, *Two-interval pattern problems*, Encyclopedia of Algorithms (M.-Y. Kao, ed.), Springer, 2008, pp. 985–989.

[24] M.K. Warmuth and D. Haussler, *On the complexity of iterated shuffle*, Journal of Computer and System Sciences **28** (1984), no. 3, 345–358.

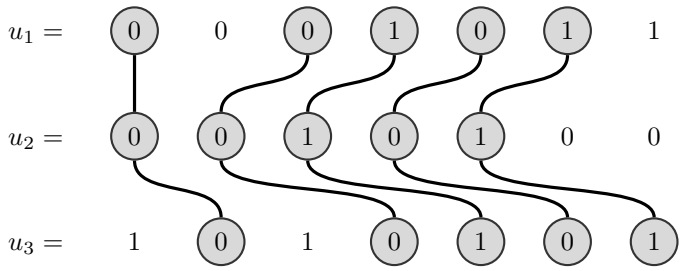
Appendix

This appendix is devoted to illustrating the proofs of Proposition 4 and Proposition 10. In the subsection *Describing the* 01-LCS *instance* we describe a specific instance of the 01-LCS problem. In the subsection *Full example for shuffled square words* (resp. *Full example for shuffled square words with reverse*) we fully construct exhibit the construction as described in Proposition 4 (resp. Proposition 10). Notice that we do use the same specific 01-LCS instance i, the two example constructions.

Describing the 01-LCS *instance*

Recall that the Longest Common Subsequence for binary words (written 01-LCS for short) is defined as follows: Given a collection of words $U = \{u_1, u_2, \ldots, u_m\}$, $u_i \in \{0,1\}^*$ for $1 \leq i \leq m$, and a positive integer k, decide whether there exists a subsequence of size k common to all sequences of U [15]. Without loss of generality, we may assume that $|u_i| = |u_j|$ for $1 \leq i < j \leq m$, and that that we are looking for a common subsequence with p letters 0 and q letters 1, $k = p + q$. We write (U, p, q) for such an instance of the 01-LCS problem.

In what follows, we consider the specific instance of the 01-LCS problem $U = \{u_1, u_2, u_3\}$, where $u_1 = 0001011$, $u_2 = 0010100$ and $u_3 = 1010101$. For $p = 3$ (*i.e.*, number of letters 0 is the sought solution) and $q = 2$ (*i.e.*, number of letters 1 is the sought solution), a solution is given by $w = 00101$. Aiming at better illustrating the reductions used in Proposition 4 and Proposition 10, we fix an arbitrary occurrence of the solution $w = 00101$ in each input word u_1, u_2 and u_3 as follows:



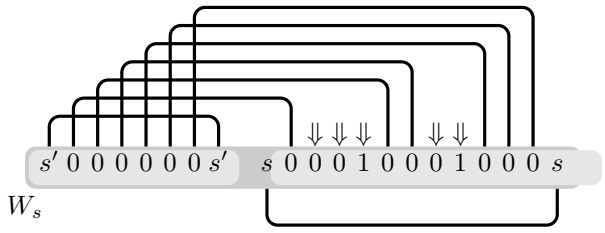


Figure 5: The intra-gadget edges in gadget W_s for the solution 00101. Double arrows designate free letters (*i.e.* letters that are not involved in these intra-gadget edges).

Full example for shuffled square words

We present the solution for the reduction used in Proposition 4 for the specific instance of the 01-LCS problem $U = \{u_1, u_2, u_3\}$, where $u_1 = 0001011$, $u_2 = 0010100$ and $u_3 = 1010101$, with $p = 3$ and $q = 2$. The occurrence of the solution in each input string u_1, u_2 and u_3 is assumed to be the one given in Subsection *Describing the* 01-LCS *instance*.

We have decomposed the whole construction into a sequence of five figures describing the various parts:

- Figure 5: the intra-gadget edges in gadget word W_s,
- Figure 6: the inter-gadget edges for gadget words W_s and W_1 (gadget word W_s is not fully represented) together with the intra-gadget edges in gadget word W_1,
- Figure 7: the inter-gadget edges for gadget words W_1 and W_2 (gadget word W_1 is not fully represented) together with the intra-gadget edges in gadget word W_2,
- Figure 8: the inter-gadget edges for gadget words W_2 and W_3 (gadget word W_2 is not fully represented) together with the intra-gadget edges in gadget word W_3, and
- Figure 9: the inter-gadget edges for gadget words W_s and W_1 (gadget word W_3 is not fully represented) together with the intra-gadget edges in gadget word W_t.

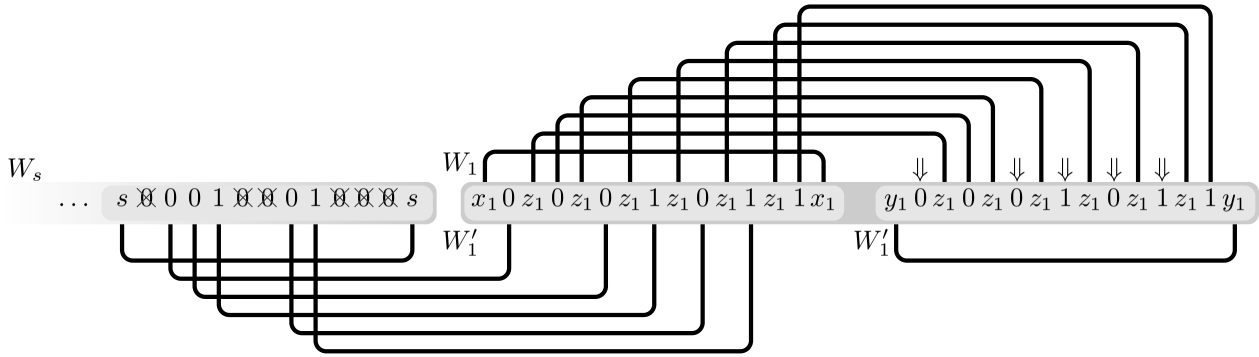


Figure 6: The inter-gadget edges for gadget words W_s and W_1 (gadget word W_s is not fully represented) together with the intra-gadget edges in gadget word W_1. The (s, s)-edge is an intra-edge of gadget word W_s (and hence part of Figure 5) but is shown for the sake of clarity. The crossed letters in (partial) gadget word W_s denote those letters that are involved in the intra-gadget edges described in Figure 5. Double arrows designate free letters (*i.e.* letters that are not involved in these the intra-gadget edges for gadget words W_s and W_1.

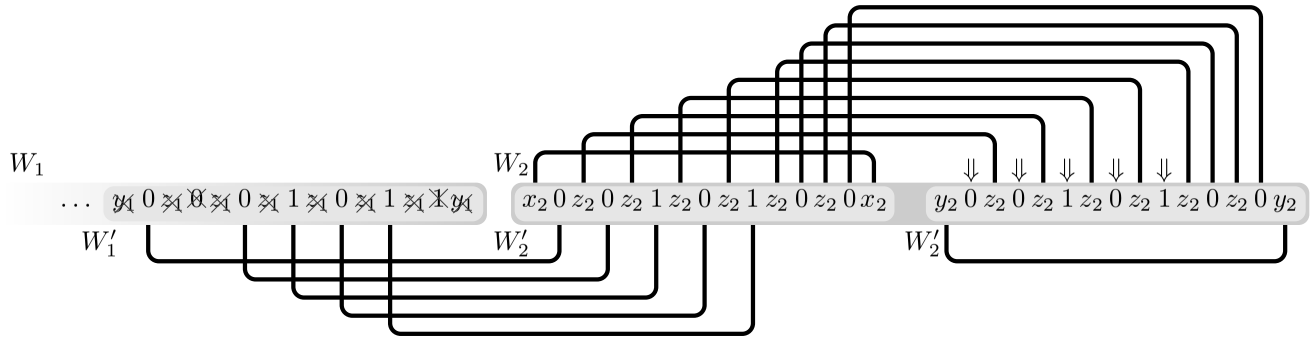


Figure 7: The inter-gadget edges for gadget words W_1 and W_2 (gadget word W_1 is not fully represented) together with the intra-gadget edges in gadget word W_2. The crossed letters in (partial) gadget word W_1 denote those letters that are involved in the intra-gadget edges described in Figure 5. Double arrows designate free letters (*i.e.* letters that are not involved in these the intra-gadget edges for gadget words W_1 and W_2.

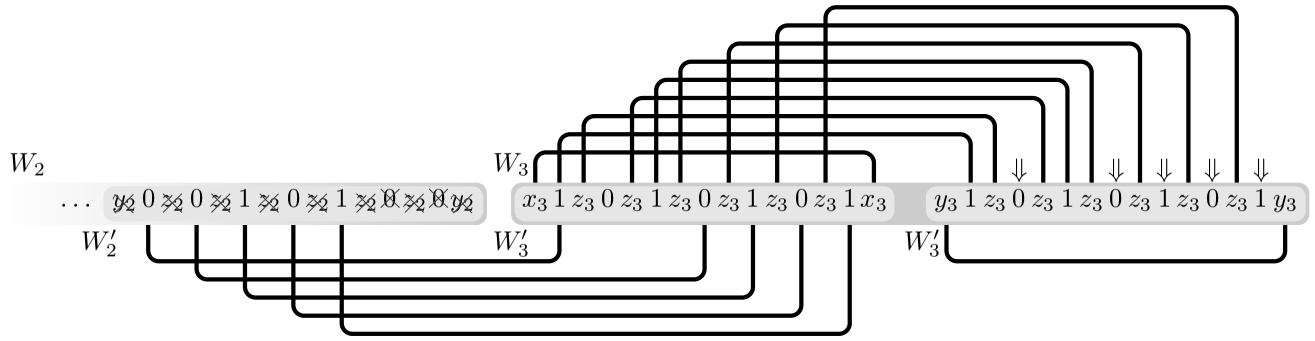


Figure 8: The inter-gadget edges for gadget words W_2 and W_3 (gadget word W_1 is not fully represented) together with the intra-gadget edges in gadget word W_3. The crossed letters in (partial) gadget word W_2 denote those letters that are involved in the intra-gadget edges described in Figure 7. Double arrows designate free letters (*i.e.* letters that are not involved in these the intra-gadget edges for gadget words W_2 and W_3.

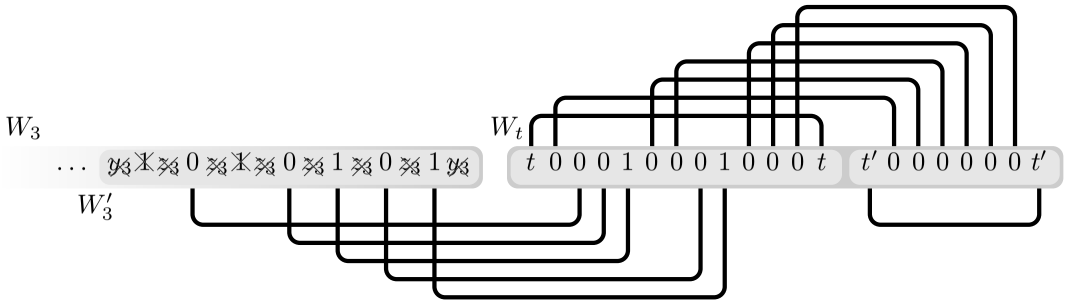


Figure 9: The inter-gadget edges for gadget words W_3 and W_t (gadget word W_3 is not fully represented) together with the intra-gadget edges in gadget word W_t. The crossed letters in (partial) gadget word W_3 denote those letters that are involved in the intra-gadget edges described in Figure 8.

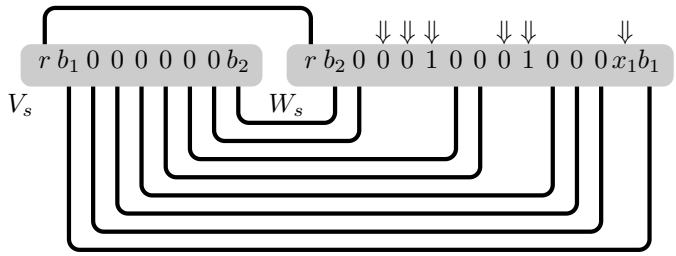


Figure 10: The edges connecting gadget words V_s and W_s. Double arrows designate free letters in gadget word W_s (*i.e.* those letters of W_s that are not involved in these edges connecting gadget words V_s and W_s).

Full example for shuffled square words with reverse

We present the solution for the reduction used in Proposition 10 for the specific instance of the 01-LCS problem $U = \{u_1, u_2, u_3\}$, where $u_1 = 0001011$, $u_2 = 0010100$ and $u_3 = 1010101$, with $p = 3$ and $q = 2$. The occurrence of the solution in each input string u_1, u_2 and u_3 is assumed to be the one given in Subsection *Describing the* 01-LCS *instance*.

We have decomposed the whole construction into a sequence of five figures describing the various parts:

- Figure 10: the edges connecting gadget words V_s and W_s,
- Figure 11: the edges connecting gadget words W_s and V_1 and the edges connecting gadget words V_1 and W_1,
- Figure 12: the edges connecting gadget words W_1 and V_2 and the edges connecting gadget words V_2 and W_2,
- Figure 13: the edges connecting gadget words W_2 and V_3 and the edges connecting gadget words V_3 and W_3, and
- Figure 14: the edges connecting gadget words W_3 and V_t and the edges connecting gadget words V_t and W_t.

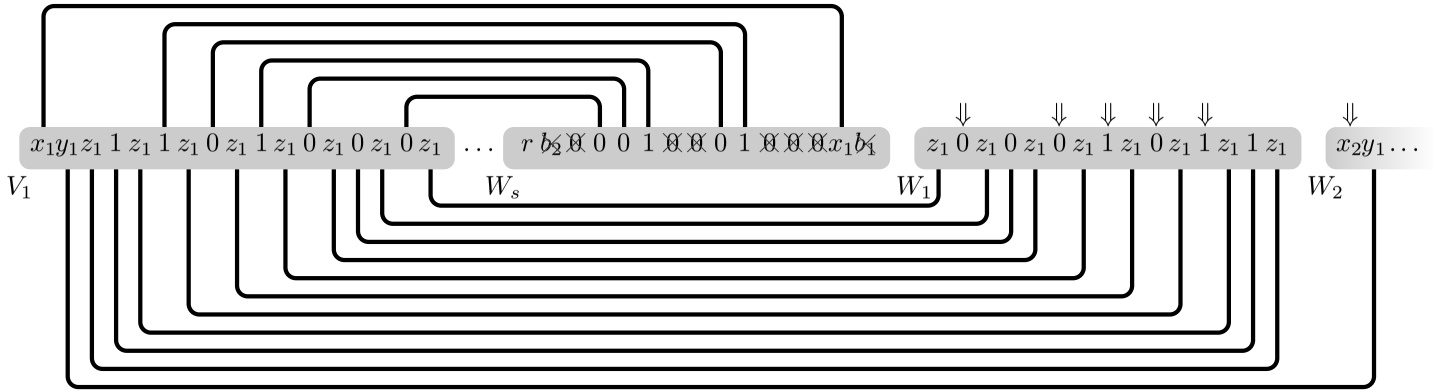


Figure 11: The edges connecting gadget words W_s and V_1 and the edges connecting gadget words V_1 and W_1. The crossed letters in gadget word W_s denote those letters that are involved in Figure 10. Double arrows designate free letters in gadget word W_1 (*i.e.* those letters of W_1 that are not involved in these edges connecting gadget words V_1 and W_1). Notice that gadget word W_2 is partially represented to show the (y_1, y_1)-edge.

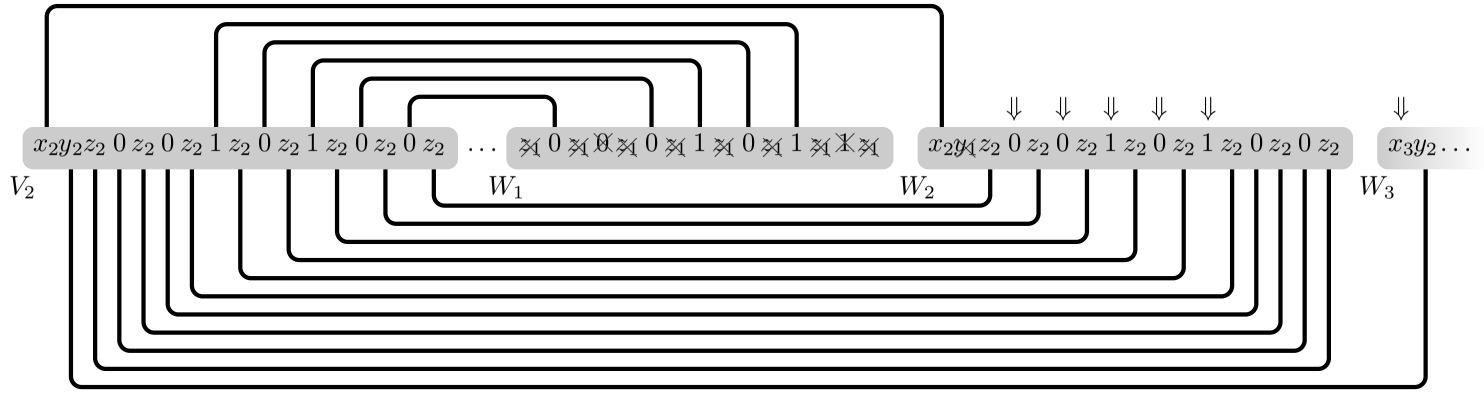


Figure 12: The edges connecting gadget words W_1 and V_2 and the edges connecting gadget words V_2 and W_2. The crossed letters in gadget word W_1 denote those letters that are involved in Figure 11. Double arrows designate free letters in gadget word W_2 (*i.e.* those letters of W_2 that are not involved in these edges connecting gadget words V_2 and W_2). Notice that gadget word W_3 is partially represented to show the (y_2, y_2)-edge.

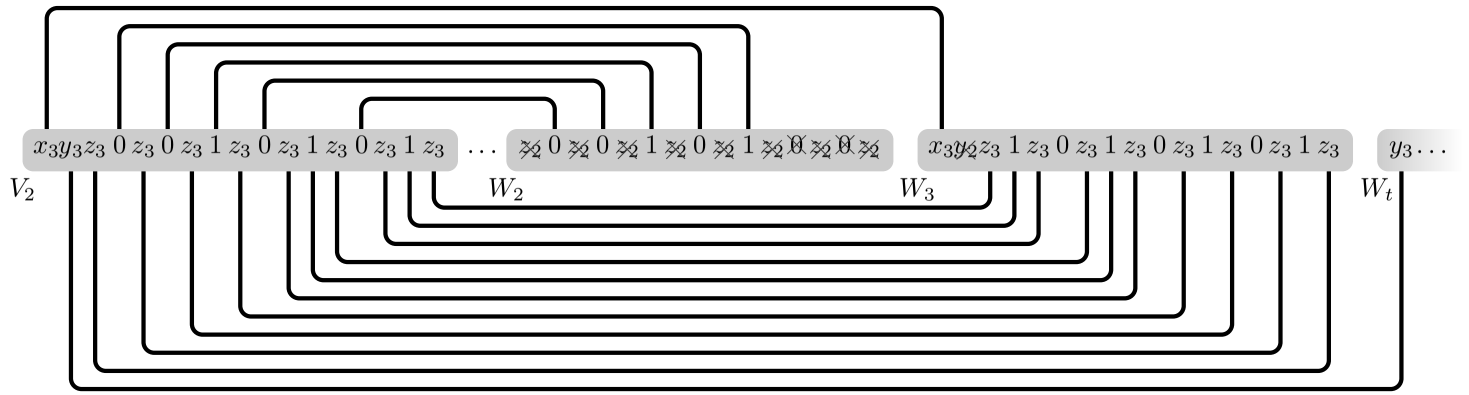


Figure 13: The edges connecting gadget words W_2 and V_3 and the edges connecting gadget words V_3 and W_3. The crossed letters in gadget word W_1 denote those letters that are involved in Figure 12. Double arrows designate free letters in gadget word W_3 (*i.e.* those letters of W_3 that are not involved in these edges connecting gadget words V_3 and W_3). Notice that gadget word W_t is partially represented to show the (y_3, y_3)-edge.

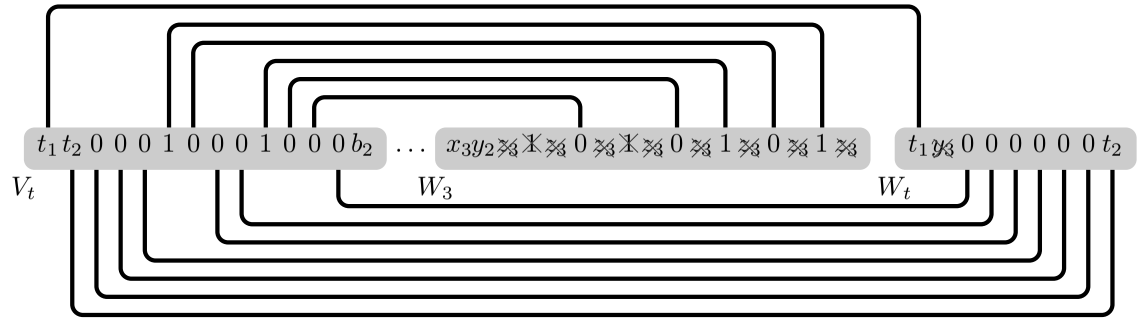


Figure 14: The edges connecting gadget words W_3 and V_t and the edges connecting gadget words V_t and W_t. The crossed letters in gadget word W_3 denote those letters that are involved in Figure 13.