From First Principles to the Burrows and Wheeler Transform and Beyond, via Combinatorial Optimization

Raffaele Giancarlo* Antonio Restivo† Marinella Sciortino‡

Abstract

We introduce a combinatorial optimization framework that naturally yields a class of optimal word permutations. Our framework provides the first formal quantification of the intuitive idea that the longer the context shared by two symbols in a word, the closer those symbols should be to each other in a linear order of the symbols. The Burrows and Wheeler transform [6], and the compressible part of its analog for labelled trees [10], are special cases in the class. We also show that the class of optimal word permutations defined here is identical to the one identified by Ferragina et al. for compression boosting [9]. Therefore, they are all highly compressible. We also investigate more general classes of optimal word permutations, where relatedness of symbols may be measured by functions more complex than context length. In this case, we establish a non-trivial connection between word permutations and Table Compression techniques presented in Buchsbaum et al. [5], on one hand, and a universal similarity metric [17] with uses in Clustering and Classification [8]. Unfortunately, for this general problem, we provide instances that are MAX-SNP hard, and therefore unlikely to be solved or approximated efficiently. The results presented here indicate that, contrary to folklore, the key feature of the Burrows and Wheeler transform seems to be the existence of efficient algorithms for its computation and inversion, rather than its compressibility. Finally, for completeness, we also provide solution to an open problem implicitly posed in [6] regarding the computation of the transform.

1 Introduction

The Burrows-Wheeler Transform [6] (`bwt` for short) has changed the way in which fundamental tasks for word processing and data retrieval, such as compression and indexing, are designed and engineered (see e.g. [9, 11, 12, 13, 21]). In their original paper, Burrows and Wheeler [6] give intuition why $\texttt{bwt}(w)$, a particular permutation of a word w, is more easily compressible than the word itself: symbols that are preceded (actually succeeded) by the same context are grouped together so that a relatively weak locally adaptive scheme, such as Move to Front [2], can achieve compression ratio comparable to the best compression algorithms. Ferragina et al. [9] have investigated this remarkable property of the transform and formalized it in terms of compression boosting. Following the same intuition as Burrows and Wheeler, Ferragina et al. [10] have defined `xbw`, a generalization of `bwt` to labelled trees. It consists of two parts: a binary word, encoding the topology of the tree, and a word that gives a rearrangement of the labels of

*Dipartimento di Matematica ed Applicazioni, Università di Palermo, Italy. E-mail: `raffaele@math.unipa.it`. Partially supported by the Italian MIUR FIRB project "Bioinformatica per la Genomica e la Proteomica" and by MIUR FIRB Italy-Israel project "Pattern Matching and Discovery in Discrete Structures, with applications to Bioinformatics".

†Dipartimento di Matematica ed Applicazioni, Università di Palermo, Italy. E-mail: `restivo@math.unipa.it`. Partially supported by the Italian MIUR PRIN project "Automi e Linguaggi Formali: Aspetti Matematici ed Applicativi", by the Italian MIUR FIRB project "Bioinformatica per la Genomica e la Proteomica" and by MIUR FIRB Italy-Israel project "Pattern Matching and Discovery in Discrete Structures, with applications to Bioinformatics".

‡Dipartimento di Matematica ed Applicazioni, Università di Palermo, Italy. E-mail: `mari@math.unipa.it`. Partially supported by the Italian MIUR PRIN project "Automi e Linguaggi Formali: Aspetti Matematici ed Applicativi", by the Italian MIUR FIRB project "Bioinformatica per la Genomica e la Proteomica" and by MIUR FIRB Italy-Israel project "Pattern Matching and Discovery in Discrete Structures, with applications to Bioinformatics".

the tree, according to context information. The interested reader is referred to section 3 for an example and [10] for the non-trivial algorithmic details.

To date, it is still open whether `bwt` and `xbw` are unique highly compressible and efficiently computable and invertible word permutations. For instance, Ferragina et al. [9] defined a class of word permutation in terms of their realization via a suffix tree [22]. Each permutation can then be effectively compressed via boosting, the `bwt` being the most studied case. Preliminary results in [10] indicate that analogous results could hold for `xbw`. In a sense, Ferragina et al. gave a characterization of highly compressible word permutations in terms of a single algorithmic paradigm, highlighting that the key feature of `bwt` is not its compressibility, but rather the existence of efficient algorithms for its computation and inversion.

Here we take a different avenue. Given a list of words X, we associate the symbols of a word w to the words in X so that the first symbol corresponds to the first word and so on. We take the length of the longest prefix two words have in common as a simple quantification of their degree of relatedness. We then consider the problem of permuting the words in X so that a global measure of relatedness is maximized. This approach naturally leads to the identification of Hamiltonian paths of maximum cost in a suitably defined graph. Those paths induce a permutation on the set X and therefore on the associated word w. We refer to those word permutations as *optimal*. We can show:

(A) When the list X is taken to be the list of cyclic shifts of a word w, $\texttt{bwt}(w)$ is an optimal permutation. When the list X is taken to be the "contexts" associated to the labels of a tree (see Figure 3 for an example), the non-binary part of `xbw` is an optimal permutation. That is, the known instances of compressible word permutations are in the class we identify from a simple notion of relatedness of words.

(B) When the list X is taken to be the list of cyclic shifts of a word w, we show that the set of optimal word permutations is equal to the word permutations mappable to a suffix tree defined in [9]. Therefore, using the results in [9], our optimal word permutations are all highly compressible. Preliminary results in [10] indicate that analogous results may hold for `xbw`.

The degree of relatedness of two words may be formalized by functions more sophisticated than the length of their common prefix, and indeed two classes of functions are available in the literature [14]: similarity and distance functions (e.g. [5, 7, 14, 17]). Similarity functions are somewhat to be preferred to distance functions, as outlined in [14]. We now briefly discuss two functions that have found uses in Compression and Classification. Ming Li et al. have proposed a theoretical foundation for similarity metrics between two words, which is based on Kolmogorov Complexity [17]. In real applications, a metric stemming from that theory is realized by data compression algorithms $\mathcal{C}$. Indeed, the basic idea is that if two words are related, then their concatenation should compress better than each word by itself. Cilabresi and Vitanyi [8] have used the normalized compression distance (NCD for short), coming from that theory, for classification and clustering. Motivated by Table Compression applications, Buchsbaum et al. [5] have investigated notions of relatedness of words, based on classic Information Theoretic notions. The basic idea is the same: measure relatedness of words via compression. Cilabresi and Vitanyi use hierarchical clustering for classification based on the NCD metric, while the approach to Table Compression proposed by Buchsmaum et al. can be thought of as a partitional clustering algorithm. The metric they use to measure relatedness is strikingly similar to the NCD metric and it has been formalized in [4] in terms of notions of combinatorial dependency and later refined in information theoretic terms [5]. Since `bwt` is commonly thought of as a transform that clusters together symbols that are related, we are motivated to look at more general instances of our combinatorial optimization framework to show:

(C) There exist similarity functions measuring the relatedness of words so that the problem of computing optimal word permutations based on them is MAX-SNP hard [24] and therefore unlikely to admit a

			F					L
			↓					↓
		0	a	a	b	r	a	c
I	→	1	a	b	r	a	c	a
		2	a	c	a	a	b	r
		3	b	r	a	c	a	a
		4	c	a	a	b	r	a
		5	r	a	c	a	a	b

(a)

			F						L
			↓						↓
I	→	0	\$	a	b	r	a	c	a
		1	a	b	r	a	c	a	\$
		2	a	c	a	\$	a	b	r
		3	a	\$	a	b	r	a	c
		4	b	r	a	c	a	\$	a
		5	c	a	\$	a	b	r	a
		6	r	a	c	a	\$	a	b

(b)

Figure 1: (a) The matrix of all cyclic shifts of the word $w = abraca$. The pure transformation is $\texttt{bwt}(w) = caraab$, the last column of the matrix. The symbol I is a pointer to the row where the word appears and it is essential to recover w from $\texttt{bwt}(w)$. (b) The matrix of all cyclic shifts of the word $w = abraca\$$. The augmented transform is $\$\text{-}\texttt{bwt}(w) = arcaab$, obtained by removing the end-marker symbol from the last column of the matrix. The pointer has the same role as in (a).

polynomial time approximation scheme [1]. Particularly relevant are three measures we define, one based on the LZ77 parsing rule [26], on which compressors like gzip are based, the other based on the LZ77 parsing rule and on the NCD metric and the last one based on the classic Shortest Common Superstring [14] problem. As a consequence, we also establish a non-trivial connection between the Burrows and Wheeler transform, optimal word permutations, Table Compression as formalized in [5] and Clustering by Compression as formalized in [8]. We remark that those hardness results do not extend to the case of distance functions, whose computational complexity remains open. In fact, we limit ourselves to discuss only an important special case that can be solved efficiently.

Apart from the intrinsic theoretic interest of the above results, they also bring to light that, contradicting folklore, the key feature of the `bwt` is not its compressibility, property shared by many other word permutations we constructively identify starting from first principles, but it consists of the efficient algorithms that allow for its fast computation and inversion.

Once that we have derived the `bwt` from first principles, we also solve an open problem implicitly posed in the original paper by Burrows and Wheeler, where they defined the transform of a word w in terms of its cyclic shifts. An example, from which the definition can be inferred, is given in Figure 1(a). We refer to such a transform as *pure* and keep the notation $\texttt{bwt}(s)$.

However, they also pointed out that, for efficiency reasons, the transform of word w would be best computed by sorting the suffixes of the augmented word $w\$$, where \$ is an end-marker symbol not appearing elsewhere in the word. We refer to such a transform as *augmented* and denote it as $\$\text{-}\texttt{bwt}(w)$. An example is given in Figure 1(b). As the two examples illustrate, $\texttt{bwt}(w) \neq \$\text{-}\texttt{bwt}(w)$. Since efficiency matters in data compression, the augmented transform has become the *de facto* standard definition in the literature. In algorithmic terms, it is known how to compute the pure transform by sorting the *cyclic shifts* of w, while the augmented transform can be computed by *sorting only* the suffixes of $w\$$.

(D) We show that the computation of the pure transform can be reduced to the sorting of *only* the suffixes of s. The reduction takes linear time, uses s *read only* memory cells and $|s| + c$, c constant, work memory cells and does not use sorting at all. This is accomplished by using the self-evident (and most of the times overlooked) correspondence between Lyndon words [18], conjugacy classes and the pure transform. As a special case, we show that the introduction of the extra symbol in w is a clever way to obtain the reduction for free.

2 Notation and Basic Definitions

Let A be a finite alphabet and let $\leq$ be a linear order relation defined on the characters of A, from which a lexicographic order relation on A^* can be induced. In what follows, for our examples, we will assume the standard alphabetic ordering. Let \$ be an end-marker symbol not in the alphabet and assume that \$ is lexicographically smaller than any character in A. Given a word x, let $suff(x)$ be the set of all of its suffixes. A set of words X is *prefix-free* if no word in X is a proper prefix of any other word in X.

A word $v \in A^*$ is *primitive* if $v = w^k$ implies $v = w$ and $k = 1$. Notice that we are using a somewhat more constrained definition of primitivity with respect to the standard one in the literature [23]. It is well known that every word $v \in A^*$ can be written in a unique way as a power of a primitive word, i.e., there exists a unique primitive word w and a unique integer k such that $v = w^k$. Let `period`(v) be an algorithm that, given in input v, returns that integer $k \geq 1$. Several of such type of algorithm can be obtained via standard word matching tools [14] and they are linear time and use only $|v|+c$ work memory cells, c constant.

Two words $x, y \in A^*$ are *conjugate* if $x = uv$ and $y = vu$ for some $u, v \in A^*$. It is easy to see that conjugacy is an equivalence relation. Moreover, for later use, we point out that, for primitive words, all words in the same conjugacy class are distinct [18]. A *Lyndon word* is a primitive word which is also the minimum in its conjugacy class, with respect to the lexicographic order relation. Let `min`(v) be an algorithm that, given in input a primitive word, returns an integer j, $1 \leq j \leq |v|$, indicating where the minimum of its cyclic shifts starts. That is, it computes the Lyndon word corresponding to v. An algorithm for this task that takes linear time and uses $|v| + c$ work memory cells, c constant, and no sorting, can be found in [19].

Let s be a word over the alphabet $A = \{a_1, \ldots, a_h\}$ and, for each $a_i \in A$, let n_i be the number of occurrences of a_i in s. Throughout this paper we assume that all logarithms are taken to the base 2 and $0 \log 0 = 0$. The *0-th order empirical entropy* of the word s is defined as $H_0(s) = -\sum_{i=1}^{h} (n_i/|s|) \log(n_i/|s|)$. It is well known that H_0 is the maximum compression we can achieve using a fixed codeword for each alphabet symbol. We can achieve a greater compression if the codeword we use for each symbol depends on the k symbols preceding it, i.e., its length k context, since the maximum compression is now bounded by the k-th order entropy $H_k(s)$ (see [21] for the formal definition).

For highly compressible words, $|s|\, H_k(s)$ fails to provide a reasonable bound to the performance of compression algorithms (see discussion in [9, 21]). For that reason, [21] introduced the notion of *0-th order modified empirical* entropy:

$$H_0^*(s) = \begin{cases} 0 & \text{if } |s| = 0 \\ (1 + \lfloor \log |s| \rfloor)/|s| & \text{if } |s| \neq 0 \text{ and } H_0(s) = 0 \\ H_0(s) & \text{otherwise.} \end{cases} \quad (1)$$

Note that for a non-empty word s, $|s|H_0^*(s)$ is at least equal to the number of bits needed to write down the length of s in binary. The *k-th order modified empirical entropy* H_k^* is then defined in terms of H_0^* as the maximum compression we can achieve by looking at *no more than* k symbols preceding the one to be compressed.

In the data compression literature, it is costumary to work with symbol contexts that precede the symbol and the various definitions of entropy just given comply to that rule. However, as observed in [9], it is more convenient to work with symbol contexts that actually succeed a given symbol, when one deals with the `bwt`. Information measures analogous to the ones given above can be provided also in this case [9]. Here it is of interest $\overleftarrow{H}_k^*$, the analog of $H_k^*(w)$. In fact, it can be shown [9] that $H_k^*(w^R) = \overleftarrow{H}_k^*(w)$, where w^R is the reverse of w.

3 Relatedness of Words and Optimal Word Permutations

In this section we start to describe the combinatorial optimization framework briefly outlined in the Introduction. Additional results, mainly negative, are presented in section 4. In particular, we derive `bwt` and `xbw` as special cases of optimal word permutations, a notion we introduce here. We then show that such optimal word permutations are all compressible with good performance guarantee, measured in terms of the k-th order modified empirical entropy of a word.

3.1 The Burrows and Wheeler Transform and Optimal Word Permutations

Let $X = (x_1, \ldots, x_n)$ be a list of words and let $w = a_1 \cdots a_n$ be yet another word. Associate the positions of w to the words in X, from left to right. In what follows, we implicitly assume such an association. Notice that any permutation of X induces a permutation of w, i.e., a new word w'. We are interested in permutations of X such that the stronger the relation between two words, the closer they should be in such an order. As a measure of relatedness of two words x and y, we consider first the length of their longest common prefix, denoted $|lcp(x, y)|$. Additional measures are discussed in section 4.

Problem 1 Find a permutation $X_\pi = (x_{i_1}, x_{i_2}, \cdots, x_{i_n})$ of the list X so that $LCP(X_\pi) = \Sigma_{j=1}^{n-1} |lcp(x_{i_j}, x_{i_{j+1}})|$ is maximum. We denote the corresponding permutation of w as *optimal* with respect to the longest common prefix measure.

Problem 1 naturally corresponds to find a Hamiltonian path of maximum cost in an undirected, complete and weighted graph G_X, where the vertices are the elements of X and each edge (x_i, x_j) is weighted by $|lcp(x_i, x_j)|$. Figure 2 gives an example of G_X. In this case, finding a solution is computationally easy:

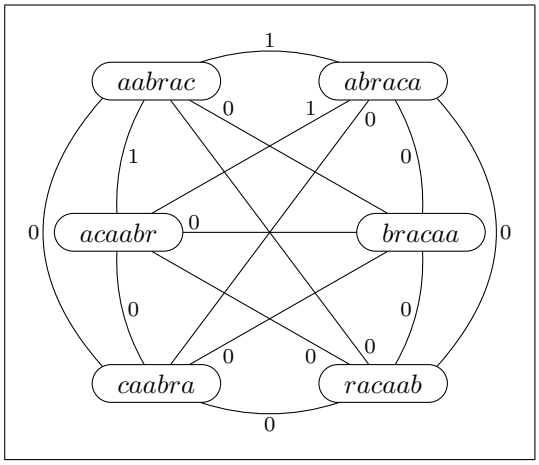

Figure 2: Undirected, complete and weighted graph G_X, where X is the list of all conjugates of the word *abraca*

Theorem 1 *Consider any path in G_X that is also a spanning tree of maximum cost in G_X. Then, it is also a Hamiltonian path of maximum cost. In particular, C_{lex}, corresponding to the lexicographic non-decreasing order of the words in X, is such an optimal path and provides a solution to Problem 1.*

Proof: For each node u in G_X, let $s(u)$ be the word corresponding to u. Given a spanning tree T_{opt} of maximum cost in G_X, we transform it into C_{lex}, with no change in total cost. The result then follows

because the cost of C_{lex} must be at least equal to that of a Hamiltonian path of maximum cost. We limit ourselves to present the first step of the transformation.

Let u be the node corresponding to the lexicographically smallest element of X and let $v_1, v_2, \ldots, v_s$ be its neighbors. Moreover, let $s(w)$ be the word lexicographically closest to $s(u)$. Now, we break the edges connecting u to its neighbors and connect it to w. In addition, we also connect w to the former neighbors of u (with the possible exception of w itself). Using the definition of lexicographic order, it can be shown that this transformation yields a new tree of cost exactly equal to that of T_{opt} (else T_{opt} could not be optimal). In the new tree, u and w are neighbors and u has no other incoming edges. We can now apply the same reasoning to w by considering only its neighbors in T_{opt} (possibly excluding u). ∎

We now show that both the `bwt` of a word and (part of) the `xbw` of a tree are optimal permutations, with respect to the *lcp* measure, of a suitably chosen word w and list X. To this end, we need to recall the definition of `xbw` for a labelled tree $\mathcal{T}$ [10]. Rather than being formal, we resort to an example given in Figure 3.

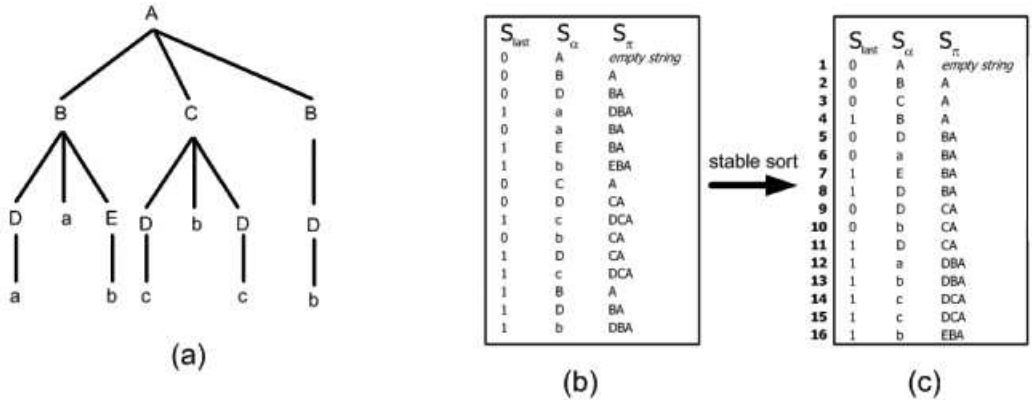

Figure 3: (a) A labelled tree $\mathcal{T}$. (b) The word S_{last} encodes the structure of the tree. The triplet in each row encodes information about a node u. In particular, the letter in S_α is the label of u and the word in S_π is its context, i.e., the word obtained by concatenating the labels on the path from $parent(u)$ to the root. The triplets also establish a natural correspondence between symbols of S_α and words in S_π. (c) $\texttt{xbw}(\mathcal{T}) = (S_{last}, S'_\alpha)$ is obtained after a stable sort of the triples in (b).

Given a word $w = a_1 a_2 \ldots a_n$, let $sh(w, i)$ be the word corresponding to the cyclic shift of w starting at i, $1 \leq i \leq n$, and let $SH(w)$ be the list of those shifts, given in increasing order of i and starting with $i = 2$. For each position i of w, associate to it $sh(w, i+1)$, where arithmetic is mod n.

Let G_w be the undirected, complete and weighted graph where the vertices are the integers in $\{1, \ldots, n\}$ and each edge (i, j) is weighted by $|lcp(sh(w, i+1), sh(w, j+1))|$, as showed in Figure 4.

Corollary 1 *Let $\mathcal{T}$ be a labelled tree and w be a word. Consider the lists of words S_π and $SH(w)$ and the correspondence of their elements to S_α and w, respectively. Then, the permutations S'_α and $\texttt{bwt}(w)$ of S_α and w correspond to optimal Hamiltonian paths in the graphs G_{S_π} and $G_{SH(w)}$, respectively. Therefore, they are both optimal word permutations with respect to the longest common prefix measure.*

Proof: For both graphs the result is implied by Theorem 1. ∎

Remark 1 With reference to Figure 4, we point out that there are maximum cost Hamiltonian paths not corresponding to $\texttt{bwt}(w)$. In particular, the Hamiltonian paths of maximum cost (5 3 6 4 1 2), (5 3 6 1 4 2)

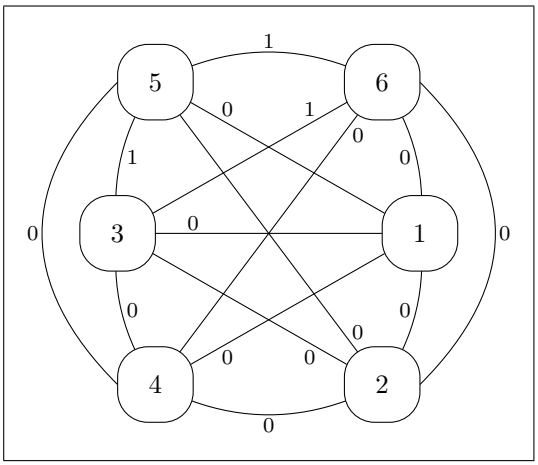

Figure 4: Graph G_w where $w = abraca$

and (6 5 3 4 1 2) correspond to the word permutations *craaab*, *craaab* and *acraab*, respectively, while the transform is *caraab*. We also notice that *craaab* gives a longer run of identical symbols than *caraab*. The former corresponds to the alphabet ordering $a < c < b < r$ while the latter to the standard lexicographic order. Therefore, the alphabet ordering one chooses to derive the transform may play a role in the actual compression of the original word.

3.2 High Order Entropy Compressibility of Optimal Permutations

We now show that optimal permutations for a single word w yield high order entropy compression, with the `bwt` being the most studied case. As a matter of fact, we show a rather strong result. Indeed, Ferragina et al. [9] introduce the notion of permutations realized by a suffix tree [22], showing that those permutations can be compressed to satisfy high order entropy bounds via boosting techniques. We show that all and only the optimal permutations of a word, introduced in the previous subsection, are realized by a suffix tree. In other words, Ferragina et al. characterize compressible permutations in algorithmic terms. Here we show that exactly those permutations are optimal in terms of a very intuitive notion of relatedness of words.

The results here apply to a single word, where `bwt` is a special case. As for `xbw`, its characterization in terms of entropy of a tree is under investigation in [10]. However, preliminary studies by those authors indicate that results analogous to the ones reported here must hold also for `xbw`.

Let $\mathcal{T}_w$ be the lexicographically sorted suffix tree of a word w. That is, a visit of the leaves of $\mathcal{T}_w$ from left to right gives the suffixes of w in lexicographic order.

A permutation $a_{i_1}a_{i_2}\cdots a_{i_n}$ of w is realized by $\mathcal{T}_w$ if there exists another suffix tree $\mathcal{T}$ for w (necessarily isomorphic to $\mathcal{T}_w$) such that the leaf associated to the suffix starting at position $i_j + 1$ is the j-th leaf in a left to right scan of the leaves of the tree $\mathcal{T}$ (see Figure 5 for an example).

Lemma 1 *All and only the optimal permutations of w are realized by $\mathcal{T}_w$.*

Proof: The result trivially holds when $|w| = 1$. So, we assume $|w| > 1$. Moreover, we give a proof for the case in which $SH(w)$ is a list made out of distinct words. The straightforward details for the extension of our proof to the case of non-distinct words are left to the reader.

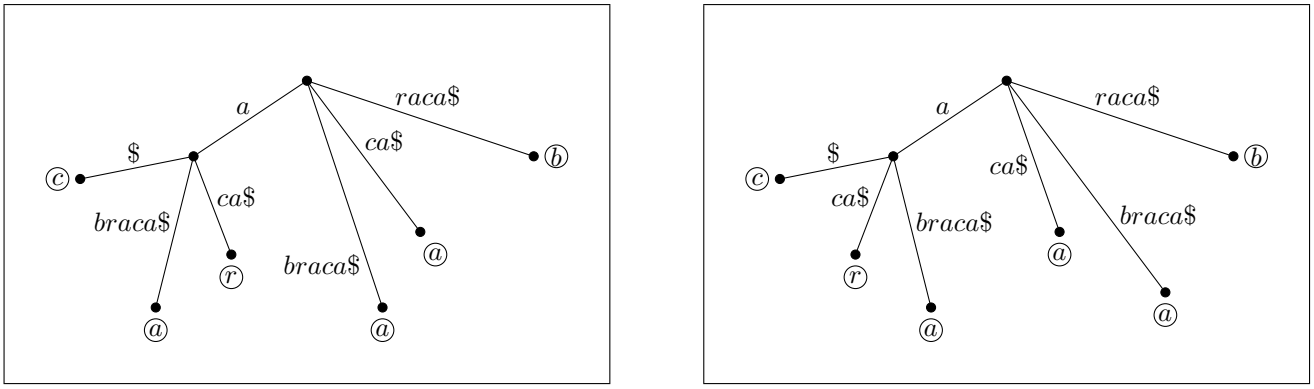

Figure 5: (a) The suffix tree $\mathcal{T}_w$ of the word $w = abraca$. We assume that a character \$ is added to each suffix of w, so that no suffix can be prefix of any other suffix. (b) The tree $\mathcal{T}$ isomorphic to $\mathcal{T}_w$ shows that the permutation *craab* is realized by $\mathcal{T}_w$.

The proof consists of two steps: (A) given an optimal permutation w' of w, we show how to build a compacted trie [15] T for the words in the list $SH(w)$, so that visiting its leaves from left to right gives a permutation of the elements in $SH(w)$ according to w'; (B) given a compacted trie T for the list $SH(w)$, we show how to obtain an optimal Hamiltonian path in $G_{SH(w)}$. Then, the result follows since any permuted instance of $\mathcal{T}_w$ can be obtained by appropriately pruning a compacted trie T for the list in $SH(w)$.

Proof of (A). Let $sh(w, i_1), \cdots, sh(w, i_{|w|})$ be the cyclic shifts of w arranged according to w'. That linear order corresponds to a Hamiltonian path H of maximum cost in $G_{SH(w)}$. Let r be the root of the trie T we need to build. Moreover, let $lcp_{\max}$ be the prefix of maximum length common to all words in $SH(w)$. Notice that $|lcp_{\max}|$ is the minimum weight on the edges of $G_{SH(w)}$. We remove all edges of minimum weight, to obtain a partition of $G_{SH(w)}$ into s connected components $P_1, \cdots, P_s$. Each P_i corresponds to a graph analogous to $G_{SH(w)}$ consisting of the suffixes of words in $SH(w)$ that still have a non-empty prefix in common, once $lcp_{\max}$ is removed from the words in $SH(w)$. By the optimality of H, that partition into connected components induces a partition of H into s paths $H_1, \cdots, H_s$, where each H_i is a Hamiltonian path of maximum cost for P_i. If $|lcp_{\max}| > 0$, we create a new node v as child of the root of the trie T and label the edge (r, v) with $lcp_{\max}$. We scale the costs on the edges of each P_i by subtracting $lcp_{\max}$ and recursively apply the above reasoning to each scaled P_i consisting of at least two nodes and with v playing the role of r in the trie T.

If $|lcp_{\max}| = 0$, we create s nodes $u_1, \cdots, u_s$ as children of r and assign them to $P_1, \cdots, P_s$, respectively. Then, we apply the above reasoning recursively as in the previous case.

Proof of (B). Consider a compacted trie T and let $u_1, \cdots, u_{|w|}$ be the Hamiltonian path obtained by taking, from left to right, the words corresponding to the leaves of T. Reasoning by induction on the length of the Hamiltonian path one can show the path is of maximum cost. Indeed, it holds that $u_1, \cdots, u_i$ is a maximum Hamiltonian path for the problem instance restricted to words $s(u_1), \cdots, s(u_i)$, $1 < i \leq |w|$. ■

Theorem 2 *Given a word w, it is possible to compress each of its optimal permutations, so that the length of the resulting binary word is bounded by $\lambda|s|\overleftarrow{H}^*_k(w) + \log_2|s| + \eta|w| + g_k$ for any $k \geq 0$; where g_k is a constant independent of $|w|$ and λ and η depend on the data compression algorithm one uses.*

Proof: The proof comes from Lemma 1 and results detailed in Section 5.2 of [9]. ∎

4 The General Problem and Its Computational Complexity

Problem 1, defined in Section 3 in terms of the cost function $LCP(X_\pi)$, can be rephrased in very general terms by picking a generic function r, not necessarily symmetric, quantifying the relatedness of two words and by defining a total cost function, analogous to $LCP(X_\pi)$, to be optimized. We provide a formal definition:

Problem 2 Given a list of words X, find a permutation $X_\pi = (x_{i_1}, x_{i_2}, \cdots, x_{i_n})$ of the list X so that $COST(X_\pi) = \Sigma_{j=1}^{n-1} r(x_{i_j}, x_{i_{j+1}})$ is optimized. We denote the corresponding permutation of w as *optimal* with respect to the r measure of relatedness.

The function r can be either a similarity or a distance function. As we show next, such a general problem is MAX-SNP hard [24] for similarity functions. That is, no polynomial time approximation scheme exists for it, unless P=NP [1]. Indeed, we consider three similarity functions for which we can provide hardness results. We then consider instances of the general problem for the case in which relatedness of words is measured by a distance function. Although we are not able to provide complexity results in this case, we discuss an important special case.

4.1 Similarity Functions

Recall from the Introduction that the functions we consider here are all motivated by uses of compression techniques in establishing the relatedness of words. The three functions we define here are a direct extension of definitions in [5], [8, 17] and [14], respectively.

In order to introduce the first measure, we need to recall the LZ77 parsing rule [16], which is used by compressors like gzip. Consider a word z, and, if $|z| \geq 1$, let z^- denote the prefix of z of length $|z| - 1$. If $|z| \geq 2$, then define $z^{--} = (z^-)^-$.

LZ77 parses z into *phrases*, each a factor of z. Assume that LZ77 has already parsed the prefix $z_1 \cdots z_{i-1}$ of z into phrases $z_1, \ldots, z_{i-1}$, and let z' be the remaining suffix of z. LZ77 selects the i'th phrase z_i as the longest prefix of z' that can be obtained by adding a single character to a factor of $(z_1 \cdots z_{i-1} z_i)^-$. Therefore, z_i has the property that z_i^- is a factor of $(z_1 z_2 \cdots z_{i-1} z_i)^{--}$, but z_i is not a factor of $(z_1 z_2 \cdots z_{i-1} z_i)^-$. This recursive definition is sound [16].

After parsing z_i, LZ77 outputs an encoding of the triplet (p_i, ℓ_i, α_i), where p_i is the starting position of z_i^- in $z_1 z_2 \cdots z_{i-1}$; $\ell_i = |z_i| - 1$; and α_i is the last character of z_i. The length of the encoding is linear in the number of phrases $ph(z)$ in the parsing of z. As it is to be expected, when a word x is related (in terms of factors) to a word y, we have that $ph(yx) \leq ph(x) + ph(y)$. In practice, using gzip as a compressor, we would have that the concatenated words would compress better than each word separately. In order to model such a notion of relatedness, we define a function $lz(y, x) = \min(ph(yx), ph(x) + ph(y))$. Now, Problem 2 becomes the following:

Problem 3 Find a permutation $X_\pi = (x_{i_1}, x_{i_2}, \cdots, x_{i_n})$ of the list X so that $LZ_{77}(X_\pi) = \Sigma_{j=1}^{n-1} lz(x_{i_j}, x_{i_{j+1}})$ is minimum. We denote the corresponding permutation of w as *optimal* with respect to the LZ77 measure.

Our second notion of relatedness comes from the NCD metric defined in [8]. Given a data compressor $\mathcal{C}$, $\mathrm{NCD}(y, x) = (\mathcal{C}(yx) - \min(\mathcal{C}(x), \mathcal{C}(y)))/\max(\mathcal{C}(x), \mathcal{C}(y))$. In order to prove hardness results, we model this function in terms of the LZ77 parsing rule and pick $\mathcal{C} = ph$. We have:

Problem 4 Find a permutation $X_\pi = (x_{i_1}, x_{i_2}, \cdots, x_{i_n})$ of the list X so that $TNCD(X_\pi) = \Sigma_{j=1}^{n-1} NCD(x_{i_j}, x_{i_{j+1}})$ is minimum, where $\mathcal{C} = ph$. We denote the corresponding permutation of w as *optimal* with respect to the Normalized Compression Distance with the LZ77 compression measure.

The third notion of relatedness of words comes from Shortest Common Superstring [14]. Given two words x and y, let $ov(x, y)$ be the longest suffix of x equal to a prefix of y. In this case, Problem 2 becomes the following:

Problem 5 Find a permutation $X_\pi = (x_{i_1}, x_{i_2}, \cdots, x_{i_n})$ of the list X so that $OV(X_\pi) = \Sigma_{j=1}^{n-1} |ov(x_{i_j}, x_{i_{j+1}})|$ is maximum. We denote the corresponding permutation of w as *optimal* with respect to the longest common overlap measure.

In order to prove hardness results, we use the standard definitions of L-reduction and MAX-SNP [24]. Let A and B be two optimization (minimization or maximization) problems. Let $cost_A(y)$ be the cost of a solution y to some instance of A; let $opt_A(x)$ be the cost of an optimum solution for an instance x of A; and define analogous metrics for B. *A L-reduces* to B if there are two polynomial-time computable functions f and g and constants $\alpha, \beta > 0$ such that:

(1) Given an instance a of A, $f(a)$ is an instance of B such that $opt_B(f(a)) \leq \alpha \cdot opt_A(a)$;

(2) Given a solution y to $f(a)$, $g(y)$ is a solution to a such that $|cost_A(g(y)) - opt_A(a)| \leq \beta |cost_B(y) - opt_B(f(a))|$.

The composition of two L-reductions is also an L-reduction. A problem is MAX-SNP hard [24] if every problem in MAX-SNP can be L-reduced to it. If A L-reduces to B, then if B has a polynomial-time approximation scheme (PTAS), so does A. A MAX-SNP hard problem is unlikely to have a PTAS [1].

We first show that Problem 3 is MAX-SNP hard. The proof is a subtle variation of one given by Buchsbaum et al. (see section 5 in [5]) for problems related to Table Compression. Here we establish a non-trivial connection between optimal word permutations with the LZ77 measure and Table Compression problems.

Consider TSP(1,2), the traveling salesman problem on a complete graph where each distance is either 1 or 2. An instance of TSP(1,2) can be specified by a graph H, where the edges of H connect those pairs of vertices with distance 1. The problem remains MAX-SNP hard if we further restrict the problem so that the degree of each vertex in H is bounded by some arbitrary but fixed constant [25] and with the additional constraint that no vertex has only one outgoing cost-1 edge [5]. This result holds for both symmetric and asymmetric TSP(1,2); i.e., for both undirected and directed graphs H. We assume that H is directed and we are interested in the version of TSP(1,2) where a Hamiltonian path of minimum cost is required. In fact, in what follows, TSP(1,2) denotes a Hamiltonian path. We also assume that n_h and m_h are the number of nodes and edges in H, respectively.

We associate a set $S(H)$ of words to the vertices and edges of H; $S(H)$ will be the input to Problem 3. Each vertex v engenders three symbols: v, v', and $\$_v$. Let $w_0, \ldots, w_{d-1}$ be the vertices on the edges out of v in H, in some arbitrary but fixed cyclic order. For $0 \leq i < d$ and mod-d arithmetic, we say that edge (v, w_i) *cyclicly precedes* edge (v, w_{i+1}). The $d + 1$ words we associate to v and these edges are: $e(v, w_i) = (v' w_{i-1})^4 v' w_i$, for $0 \leq i < d$ and mod-d arithmetic; and $s(v) = v^4 (v')^5 \$_v$. That $d \neq 1$ implies that $w_i \neq w_{i+1}$, when $d \neq 0$.

To prove MAX-SNP hardness, we first show how to transform a TSP(1,2) solution for H into a solution to Problem 3 with input $S(H)$. We then show how to transform in polynomial time a solution to Problem 3 into a TSP(1,2) solution of a certain cost. We use the intermediate step of transforming the

first solution into a canonical form of at most the same cost. The canonical form solution will correspond to the required TSP(1,2) tour.

Given two words x and y, let $\epsilon_y(x)$ be the number of phrases in which x is parsed, once that it is preceded by y. Lemmas 2–4 provide a few needed facts about the parsing of words in $S(H)$.

Lemma 2 *Let x and y be any two words in $S(H)$. Then, $lz(y,x) = 3 + \epsilon_y(x)$.*

Proof: For any word y in $S(H)$, notice that it is parsed in three phrases and that the last phrase ends with the word. Therefore, its contribution to $lz(y,x)$ is 3 in both arguments of the min function. Moreover, the fact that a phrase must always end with y immediately implies that $\epsilon_y(x) \leq ph(x) = 3$. ∎

Lemma 3 *Consider $s(v) \in S(H)$, for any vertex v in H. Let $e(q,v)$ be the word associated to some edge (q,v) in H. We have that $\epsilon_y(s(v)) = 2$, if $y = e(q,v)$; otherwise, it is 3 for any other word y in $S(H)$.*

Proof: The proof is analogous to Lemma 6.3 in [5] and therefore omitted. ∎

Lemma 4 *Consider $e(v,w) \in S(H)$, for any edge (v,w) in H. Let (v,z) be the edge that cyclicly precedes (v,w). We have that $\epsilon_y(e(v,w)) = 1$, when $y = e(v,z)$; $\epsilon_y(e(v,w)) = 2$, when $y = s(v)$; and it is at least 2 for any other $y \in S(H)$.*

Proof: The proof is analogous to Lemma 6.4 in [5] and therefore omitted. ∎

We now provide a polynomial time algorithm that, given any permutation $S(H)_\pi$ of $S(H)$ transforms it into one of a canonical form $S(H)_{st}$. The cost of this latter is no higher than that of the former and it naturally corresponds to a set of node-disjoint paths in H. The algorithm is essentially the one given in [5] (section 6.1).

Algorithm STANDARD

P1 1. Place each vertex v of H in a single-vertex path. If $s(v)$ is the first word in $S(H)_\pi$, label v terminal; otherwise, label v nonterminal.

2. While there exists a path with nonterminal left end point, pick one such end point v and process it as follows. Let $x(u)$ be the word (associated to either vertex u or to one of its outgoing edges) that precedes $s(v)$ in $S(H)_\pi$. If $x(u)$ ends in a symbol other than v, label vertex v terminal. Otherwise, $(u,v) \in H$, so connect u to v, and, for each edge $(u,w) \in H$, $u \neq w$, such that $s(w)$ is immediately preceded by $e(u,w)$, declare w terminal. (This guarantees that Phase One actually builds paths.)

P2 Let $A_1, \ldots, A_t$ be the paths obtained at the end of Phase One. We transform each path A_j into a sublist of words Y_j. If A_j consists of a single vertex v, then Y_j consists of $s(v)$ followed by all the $e(v,w_j)$'s arranged in cyclic order.

Otherwise, A_j contains more than one vertex. Initially Y_j is empty. For each edge (u,v) in order in the path, we append to Y_j: $s(u)$ followed by all of its $e(u,w_j)$'s, in cyclic order ending with $e(u,v)$. When there are no more edges to process, the last vertex of the path is processed as in the singleton-vertex case.

P3 We now concatenate $Y_1, \cdots, Y_t$ in that order to obtain $S(H)_{st}$

Lemma 5 *In polynomial time, Algorithm STANDARD transforms $S(H)_\pi$ into a permutation $S(H)_{st}$ of no higher cost.*

Proof: That Algorithm STANDARD runs in polynomial time follows immediately from the specification.

Consider the cost function $LZ_{77}(S(H)_\pi)$. By Lemma 2, it can be written as $3(n_h + m_h - 1) + \Sigma_{i=2}^{n_h+m_h} \epsilon_{x_{i-1}}(x_i)$. The same fact holds for $LZ_{77}(S(H)_{st})$. So, in order to compare the two cost functions, we can charge each word with its ϵ cost in both permutations. We then show that, amortized over the entire list, the sum of the ϵ costs of words in $LZ_{77}(S(H)_{st})$ is no higher than the corresponding one in $LZ_{77}(S(H)_\pi)$.

To this end, it suffices to concentrate on a single sublist Y_j. We point out that the proof follows quite closely that of Lemma 6.5 in [5] and it is given here for convenience of the reader. Consider the path, $(v_1, v_2, \ldots, v_r)$ from which Y_j is derived. Let $d(v)$ be the out-degree of any vertex v. $Y_j = (s(v_1), e(v_1, w_1^1), \cdots, e(v_1, w_{d(v_1)}^1), \cdots, s(v_r), e(v_r, w_1^r), \cdots, e(v_r, w_{d(v_r)}^r))$, where the w_j^i's are the neighbors in cyclic order out of v_i and, for $1 \leq i < r$, we assume without loss of generality that $w_{d(v_i)}^i = v_{i+1}$.

For any word $x \in S(H)$, let pr_1 and pr_2 be the predecessors of x in $S(H)_\pi$ and $S(H)_{st}$, respectively. By Lemma 3, for $2 \leq i \leq r$, $\epsilon_{pr_2}(s(v_i)) = 2$, which is optimal. By Lemma 4, for $1 \leq i \leq r$ and $2 \leq j \leq d(v_i)$, $\epsilon_{pr_2} e(v_i, w_j^i) = 1$, which is optimal. We thus need only consider $\epsilon_{pr_2}(s(v_1))$ and, for $1 \leq i \leq r$, $\epsilon_{pr_2}(e(v_i, w_1^i))$.

The words $e(v_i, w_1^i)$, for $1 \leq i \leq r$, each have an ϵ value of 2 in Y_j, by Lemma 4. There must be some $e(v_i, x)$ that is not immediately preceded by its cyclic predecessor in $S(H)_\pi$, and this instance of $e(v_i, x)$ also has an ϵ value of at least 2 in $S(H)_\pi$, by Lemma 4. This accounts for the first $e(\cdot)$ words immediately following each $s(\cdot)$ word in Y_j.

Finally, if $s(v_1)$ is not immediately preceded by some $e(v, v_1)$ in $S(H)_\pi$, we are done, for the ϵ values of $s(v_1)$ is 3 in both lists, by Lemma 3. Otherwise, consider the maximal sequence $e(v, w_a), e(v, w_{a+1}), \cdots e(v, w_{a+\ell} = v_1), s(v_1)$ in $S(H)_\pi$, where the w_a's are cyclicly ordered neighbors of v. Because STANDARD declared v_1 to be terminal, there was another edge (v, y) such that $e(v, y)$ immediately preceded $s(y)$ in $S(H)_\pi$, which STANDARD used to connect v and y in some path. This engenders an analogous maximal chain of $e(v, \cdot)$ words followed by $s(y)$ in $S(H)_\pi$.

Thus, there are at least two words $e(v, \cdot)$ not immediately preceded in the input by their cyclic predecessors; Lemma 4 implies each has an ϵ value of at least two in $S(H)_\pi$. We can charge the extra phrase in the ϵ value of $s(v_1)$ in Y_j against one of them, leaving the other for the extra phrase in the ϵ value of the word $e(v, \cdot)$ immediately following $s(v)$ in some $Y_{j'}$. ∎

Lemma 6 *A list Y_j output by STANDARD, corresponding to a path $(v_1, \ldots, v_r)$, has a total cost of $6r + 4\sum_{i=1}^r d(v_i) + 1$, $2 \leq j \leq t$, and $6r + 4\sum_{i=1}^r d(v_i) - 2$, for $j = 1$.*

Proof: We first give a proof for $j > 1$. By Lemma 3, each $s(\cdot)$ word has an ϵ value of 2, except $s(v_1)$, which has an ϵ value of 3, contributing a value of $2r + 1$ to the cost function. Lemma 4 implies that each $e(\cdot)$ has an ϵ value of 1, except each following an $s(\cdot)$, which has an ϵ value of 2, contributing a value of $r + \sum_{i=1}^r d(v_i)$ to the cost function. Moreover, by Lemma 2, we have also to consider an additional cost of $3(r + \sum_{i=1}^r d(v_i))$.

An analogous analysis holds for Y_1, except that $s(v_1)$ does not contribute to the ϵ costs. ∎

Theorem 3 *Problem 3 is MAX-SNP hard.*

Proof: Consider the graph H defined at the beginning of the section and let k be the minimum number of cost-2 edges that suffice to form a TSP(1,2) solution. Then the cost of the solution is $n_h - 1 + k$. Associating words to vertices and edges of H, as discussed above, we argue that the optimal cost for those words is $4m_h + k + 6n_h - 3$. The reduction is linear, since $m_h = O(n_h)$ by the assumption of bounded out-degree.

Assume that the TSP(1,2) solution with k cost-2 edges is the path $v_1, v_2, \ldots, v_{n_h}$. Then in polynomial time we can construct a corresponding standard permutation of the form output by STANDARD which, by Lemma 6, costs $4m_h + k + 6n_h - 3$.

For the converse, assume that we are given a permutation $S(H)_\pi$ of cost $4m_h + k + 6n_h - 3$. By Lemma 5, we can transform it in polynomial time into a standard permutation $Y_1, \ldots, Y_t$ of no higher cost. Recall that to each Y_j we can associate a path of H. Let $v_1, v_2, \ldots, v_{n_h}$ be the ordering of the vertices of H corresponding to the standard permutation. Then, H cannot be missing more than k of the edges (v_i, v_{i+1}), or else, by Lemma 6, the cost of the standard permutation would exceed $4m_h + k + 6n_h - 3$. ∎

We now consider the NCD measure.

Theorem 4 *Problem 4 is MAX-SNP hard.*

Proof: Consider the NCD measure for $\mathcal{C} = ph$ on any two words y and x in the set $S(H)$ defined earlier. Using a proof analogous to the one of Lemma 2, we have that $\mathrm{NCD}(y, x) = \epsilon_y(x)/3$. Now the proof follows along the same lines as the MAX-SNP hardness proof of Problem 3. ∎

Finally, we consider the overlap measure.

Theorem 5 *Problem 5 is MAX-SNP hard.*

Proof: The result comes directly from Blum et al. [3]. To be more precise, their reduction is for a minimization problem where the function to be optimized is the length of the common superstring of a set of words. However, they also notice that their reduction works as well for the maximization problem where the function to be optimized is $OV(X_\pi)$. This is known as the compression measure for the shortest common superstring problem. ∎

4.2 Distance Measures

The notion of relatedness of words can also be naturally formulated in terms of distances. We now briefly discuss word permutations in regard to some simple distance functions.

Given two words x and y, let $lcf(x, y)$ be the longest common factor that x and y have in common. Following Choffrut [7], we define two distance functions on words: $d_f(x, y) = |x| + |y| - 2|lcf(x, y)|$ and $d_p(x, y) = |x| + |y| - 2|lcp(x, y)|$. Problem 2 becomes the following:

Problem 6 Find a permutation $X_\pi = (x_{i_1}, x_{i_2}, \cdots, x_{i_n})$ of the list X so that $D_f(X_\pi) = \Sigma_{j=1}^{n-1} |d_f(x_{i_j}, x_{i_{j+1}})|$ is minimum. We denote the corresponding permutation of w as *optimal* with respect to the longest common factor distance. We can define an analogous problem for the d_p distance.

The computational complexity of both versions of Problem 6 is open. In particular, it is very easy to show that by changing the weights of G_X, defined in section 3, from *lcp* to distance d_p and maximum into minimum, C_{lex} is no longer optimal as a Hamiltonian path (and therefore it cannot be optimal as a spanning tree). The list $X = \{a, c, bb\}$ gives the required counterexample. However, for the important special case of a list composed of words of equal length, C_{lex} is again optimal:

Corollary 2 *Let X be a list of words of equal length. Consider the graph G_X, defined in section 3, where edges are weighted according to the distance function d_p. Then, C_{lex} is a minimum spanning tree and a minimum Hamiltonian path and therefore a solution to Problem 6.*

Proof: It is sufficient to notice that, since all words in X are of equal length, we work with a distance $d'(u,v) = -|lcp(u,v)|$. So, the result comes from Theorem 1 by substituting maximum with minimum. ■

Remark 2 We point out that, in order to derive `bwt` from our framework, we can use distance d_p and Corollary 2. However, an analogous result cannot hold for xbw.

5 Computation of Pure BWT Via Lyndon Words

We show how to compute $\texttt{bwt}(v)$ for a word v by sorting only the suffixes of a related word y. This latter word can be identified in linear time and no sorting. The following result states that when v is periodic, we can limit ourselves to compute `bwt` of its primitive root. That allows us to restrict attention to primitive words.

Proposition 1 *[20] Let $v = w^k$, for some integer $k > 1$, and $\texttt{bwt}(w) = a_0 a_1 \dots a_{|w|}$. Then $\texttt{bwt}(v) = a_0^k \dots a_{|w|}^k$.*

Let A_p^* be the set of primitive words in A^*. It is natural to define a set $\mathcal{P} \subseteq A_p^*$ such that, for any $v \in \mathcal{P}$, the permutation of $1, 2, \cdots, |v|$ induced by the lexicographic sort of its cyclic shifts is the same as the one induced by the lexicographic sort of its suffixes. Therefore, the `bwt` for the words in $\mathcal{P}$ can be computed via a lexicographic sort of their suffixes. The characterization of $\mathcal{P}$ is of some combinatorial interest. To this end, we need to introduce some notation.

Given two words u and v in A^*, if u is a prefix of v, we denote by $s(u,v)$ the corresponding suffix, i.e., the word z such that $uz = v$. Similarly, if v is a suffix of u, we denote by $p(u,v)$ the corresponding prefix, i.e., the word z such that $u = zv$.

Lemma 7 *A primitive word w belongs to P if and only if, for any $x, y \in suff(w)$, if x is prefix of y then $p(w,x) < s(x,y)p(w,y)$.*

Proof: We need two preliminary remarks: (a) if $suff(x)$ is prefix-free then the condition of the lemma is trivially satisfied; (b) if $x, y \in suff(w)$ and x is a prefix of y, then the three words $p(w,x)$, $p(w,y)$ and $s(x,y)$ are defined and the condition in the statement of the lemma has a meaning.

In order to prove the lemma, let vu and $v'u'$ be two different cyclic shifts of w. This means that $w = uv = u'v'$. We need to show that $v < v'$ if and only if $vu < v'u'$. In the non-trivial case that v is a prefix of v', by the condition in the statement of the lemma, we have $vu = vp(w,v) < vs(v,v')p(w,v') = v'u'$. ■

Lemma 8 *Let w be a Lyndon word. Then $w \in \mathcal{P}$. Moreover, there exist words in $\mathcal{P}$ that are not Lyndon.*

Proof: Let $w = uv = u'v'$ be a Lyndon word, where $|v| \neq |v'|$. We show that $v' < v$ if and only if $v'u' < vu$, which implies $w \in \mathcal{P}$. This is enough since, as pointed out earlier, no two cyclic shifts of a primitive word can be equal.

Assume that $v'u' < vu$. Let i the first position in which $v'u'$ and vu have a different character. There are two different cases to consider:

1. $|v'| < |v|$. If $i \leq |v'|$ then it trivially follows that $v' < v$. Otherwise, v' is a prefix of v, so $v' < v$.
2. $|v'| > |v|$. Assume that $i > |v|$. So, $vu = vydx$ and $v'u' = vycz$ where y, x, z are words in A^* and $c, d \in A$ with $c < d$. It follows that the cyclic shift $yczv$ should be smaller that $ydxv = uv = w$, contradicting the fact that w is a Lyndon word. When $i \leq |v|$, it trivially follows that $v' < v$.

Assume now that $v' < v$. We have again two cases:

1. v' is a prefix of v. That is, $vu = v'yu$. If $v'u'$ is greater then vu then $v'u' = v'xz$, where $|x| = |y|$ and $xz > yu$. So, the cyclic shift $yuv' < xzv' = u'v' = w$, contradicting the assumption that w is a Lyndon word.

2. v' is not a prefix of v. It immediately follows that $v'u' < vu$.

To conclude the proof, we show there exist words in $\mathcal{P}$ that are not Lyndon words. For instance, the word *braacad* is not a Lyndon word. Yet, that word is in $\mathcal{P}$.

■

Theorem 6 *Given a word v, there exists a linear time algorithm that reduces the computation of* $\texttt{bwt}(v)$ *to sorting only the suffixes of w, where w is the primitive root of v. It uses $|v| + c$ work memory cells, c constant, and no sorting.*

Proof: The reduction consists of the following steps. First compute the integer k such that $v = w^k$, via algorithm `period`. Use algorithm $\texttt{min}(w)$ to compute the Lyndon word y corresponding to w and then $\texttt{bwt}(y)$ via a sort of its suffixes. The definition of $\mathcal{P}$, Lemma 8 and the identity $\texttt{bwt}(y) = \texttt{bwt}(w)$ guarantee correctness for the case $k = 1$. When $k > 1$ we also need to expand $\texttt{bwt}(y)$ according to Fact 1. ■

So far, we have not discussed the decoding process; i.e., how to go from $\texttt{bwt}(v)$ to v. In order to invert the transform, we need to know the rank i of the word v in the lexicographic sort of its cyclic shifts (see [6] for details). We limit ourselves to discuss the case in which v is primitive, leaving its extension to the periodic case to the reader. The following well known fact is useful (see [18]). Here we provide an alternative short proof:

Proposition 2 *A word $v \in A_p^*$ is Lyndon if and only if it is lexicographically smaller than all of its suffixes.*

Proof: The if part is the only non-trivial one. Since v is Lyndon, it is the minimum among all of its cyclic shifts. By Lemma 8, it is in $\mathcal{P}$ and the definition of that set concludes the proof.

■

Now, in the reduction of Theorem 6, we know that the word y has rank one in the lexicographic sort of its suffixes, by Proposition 2, and therefore of its cyclic shifts, by definition of $\mathcal{P}$ and Lemma 8. However, based on those facts and the output of algorithm `min`, it is a simple exercise (left to the reader) to infer the rank of v needed for proper decoding.

Finally, we point out that the use of \$ as an end-marker symbol to compute $\$\text{-}\texttt{bwt}(v)$ via a lexicographic sort of its suffixes is a very elegant way to avoid the reduction in Theorem 6. Indeed, $\$v$ is already a Lyndon word and therefore in $\mathcal{P}$. As for decoding, $\$v$ has rank one in the sorted list of its suffixes (Proposition 2) and, from that, we can recover v.

6 Conclusions and Open Problems

We have introduced the notion of optimal word permutations and we have shown that both `bwt` and the compressible part of `xbw` are optimal with respect to a very simple and natural notion of relatedness of words. We have also discussed more sophisticated measures of relatedness between words and we have shown negative results on the efficient computability of optimal word permutations for some of those measures. The major open problem stemming from this research is to establish whether `bwt` and `xbw` are the unique highly compressible, easily computable and invertible optimal word permutations, among the ones characterized in section 3.

References

[1] A. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45:501–555, 1998.

[2] J. Bentley, D. Sleator, R. Tarjan, and V. Wei. A locally adaptive compression scheme. *Communications of the ACM*, 29(4):320–330, 1986.

[3] A. Blum, M. Li, J. Tromp, and M. Yannakakis. Linear approximation of shortest superstrings. *Journal of the ACM*, 41:630–47, 1994.

[4] A. L. Buchsbaum, D. F. Caldwell, K. W. Church, G. S. Fowler, and S. Muthukrishnan. Engineering the compression of massive tables: An experimental approach. In *Proc.11th ACM-SIAM Symp. on Discrete Algorithms*, pages 175–84, 2000.

[5] A.L. Buchsbaum, G.S. Fowler, and R. Giancarlo. Improving table compression with combinatorial optimization. *Journal of the ACM*, 50:825–851, 2003.

[6] M. Burrows and D. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.

[7] C. Choffrut. On some combinatorial properties of metrics over a free monoid. In L. J. Cummings, editor, *Combinatorics on Words. Progress and Perspectives*, pages 247–255. Academic Press, 1983.

[8] R. Cilabresi and P.M.B. Vitanyi. Clustering by compression. *IEEE Transactions on Information Theory*, 51:1523–1545, 2005.

[9] P. Ferragina, R. Giancarlo, G. Manzini, and M. Sciortino. Boosting textual compression in optimal linear time. *Journal of the ACM*, 52:688–713, 2005.

[10] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Structuring labeled trees for optimal succinctness, and beyond. In *Proc. of the 45th Annual IEEE Symposium on Foundations of Computer Science*, pages 198–207, 2005.

[11] P. Ferragina and G. Manzini. Indexing compressed text. *Journal of the ACM*, 52(4):552–581, 2005.

[12] R. Grossi, A. Gupta, and J. Vitter. High-order entropy-compressed text indexes. In *Proc. 14th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA '03)*, pages 841–850, 2003.

[13] R. Grossi and J. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35:378–407, 2005.

[14] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[15] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, second edition, 1998.

[16] R. Kosaraju and G. Manzini. Compression of low entropy strings with Lempel–Ziv algorithms. *SIAM Journal on Computing*, 29(3):893–911, 1999.

[17] M. Li, X. Chen, X. Li, B. Ma, and P.M.B. Vitanyi. The similarity metric. *IEEE Transactions on Information Theory*, 50:3250–3264, 2004.

[18] M. Lothaire. *Combinatorics on Words*, volume 17 of *Encyclopedia of Mathematics*. Addison-Wesley, Reading, Mass., 1983. Reprinted in the *Cambridge Mathematical Library*, Cambridge University Press, 1997.

[19] M. Lothaire. *Applied Combinatorics on Words*, volume 105 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2005.

[20] S. Mantaci, A. Restivo, and M. Sciortino. Burrows-Wheeler transform and Sturmian words. *Informat. Proc. Lett.*, 86:241–246, 2003.

[21] G. Manzini. An analysis of the Burrows-Wheeler transform. *Journal of the ACM*, 48(3):407–430, 2001.

[22] E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, 1976.

[23] F. Mignosi and A. Restivo. Periodicity. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, pages 237–274. Cambridge University Press, 2002.

[24] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43(3):425–40, 1991.

[25] C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

[26] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transaction on Information Theory*, 23:337–343, 1977.