On the Complexity of BWT-Runs Minimization via Alphabet Reordering

Jason W. Bentley
Department of Mathematics, University of Central Florida, Orlando, FL, USA
jason.bentley@ucf.edu

Daniel Gibney
Department of Computer Science, University of Central Florida, Orlando, FL, USA
https://www.cs.ucf.edu/~dgibney/
daniel.j.gibney@gmail.com

Sharma V. Thankachan
Department of Computer Science, University of Central Florida, Orlando, FL, USA
http://www.cs.ucf.edu/~sharma/
sharma.thankachan@ucf.edu

Abstract

The Burrows-Wheeler Transform (BWT) has been an essential tool in text compression and indexing. First introduced in 1994, it went on to provide the backbone for the first encoding of the classic suffix tree data structure in space close to entropy-based lower bound. Within the last decade, it has seen its role further enhanced with the development of compact suffix trees in space proportional to "r", the number of runs in the BWT. While r would superficially appear to be only a measure of space complexity, it is actually appearing increasingly often in the time complexity of new algorithms as well. This makes having the smallest value of r of growing importance. Interestingly, unlike other popular measures of compression, the parameter r is sensitive to the lexicographic ordering given to the text's alphabet. Despite several past attempts to exploit this fact, a provably efficient algorithm for finding, or approximating, an alphabet ordering which minimizes r has been open for years.

We help to explain this lack of progress by presenting the first set of results on the computational complexity of minimizing BWT-runs via alphabet reordering. We prove that the decision version of this problem is NP-complete and cannot be solved in time $poly(n) \cdot 2^{o(\sigma)}$ unless the Exponential Time Hypothesis fails, where σ is the size of the alphabet and n is the length of the text. Moreover, we show that the optimization variant is APX-hard. In doing so, we relate two previously disparate topics: the optimal traveling salesperson path of a graph and the number of runs in the BWT of a text. In addition, by relating recent results in the field of dictionary compression, we illustrate that an arbitrary alphabet ordering provides an $O(\log^2 n)$-approximation. Lastly, we provide an optimal linear-time algorithm for a more restricted problem of finding an optimal ordering on a subset of symbols (occurring only once) under ordering constraints.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases BWT, NP-hardness, APX-hardness

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.15

Related Version A full version of the paper is available at https://arxiv.org/abs/1911.03035.

Funding Supported in part by the U.S. National Science Foundation (NSF) under CCF-1703489.

Acknowledgements We would like to thank the reviewers for their valuable feedback and Chandra Chekuri for his helpful correspondence.

1 Introduction and Related Work

The Burrows-Wheeler Transform (BWT) is an essential building block in the fields of text compression and indexing with a myriad of applications in bioinformatics and information retrieval [24, 25, 26, 30]. Since it first arose in 1994 [6], it has been utilized to provide the popular compression algorithm bzip2 and has been adapted to provide powerful compressed text indexing data structures, such as the FM-index [12]. Hence, improvements to the algorithmic aspects of this transformation and related data structures can have a significant impact on the research community.

The BWT of a text $T[1,n]$, denoted by $BWT(T)$ is a reversible transformation which can be defined as follows: sort the circular shifts of T in lexicographical order and place the sorted circular shifts in a matrix. By reading the last column of this matrix from top to bottom we obtain $BWT(T)$. To make the transformation invertible a new symbol \$ (lexicographically smaller than others) is appended to T prior to sorting the circular shifts. See Figure 1 for an example. Historically, the BWT was introduced for the purpose of text compression [6], where its effectiveness is based on symbols with shared preceding context forming long *runs* (maximal unary substrings).

```
F                L

$   mississipp   i
i   $mississip   p
i   ppi$missis   s
i   ssippi$mis   s
i   ssissippi$   m
m   ississippi   $
p   i$mississi   p
p   pi$mississ   i
s   ippi$missi   s
s   issippi$mi   s
s   sippi$miss   i
s   sissippi$m   i
```

Figure 1 Column L shows the BWT of *mississippi*. The number of runs $r = 9$.

Recently, the number of runs "r" in the BWT has become of increasing interest. This can be attributed to the fact that many modern text collections are highly repetitive, which makes their compression effective via the BWT followed by Run-Length encoding (i.e., in space proportional to r). This raised an interesting question: can we also index the text in space propositional to r? Note that the FM-index needs space proportional to n (i.e., $\approx n \log \sigma$ bits, where σ is the alphabet size). The data-structure community has made great strides in answering this question [3, 5, 14, 21, 23, 31]. The first such index was developed by Mäkinen and Navarro in 2005 [28]. However, it lacked the ability to efficiently locate the occurrences of a pattern within space $\tilde{O}(r)$. After a decade of related research [29, 14], we now have fully functional suffix trees in space proportional to r, developed by Gagie et al. [15]. Also note that the recent optimal BWT construction algorithm for highly repetitive texts is parameterized by r [19]. A technique reducing the value of this parameter r would have a significant impact on a large body of work.

A natural way to minimize r is to change the lexicographic ordering assigned to symbols of the alphabet. To demonstrate that this can have an impact on r, consider as an example the text *mississippi* with the usual ordering $\$ < i < m < p < s$ where $r = 9$, but with the ordering $\$ < s < i < p < m$ we have $r = 8$. In fact, there exist string families in which r differs by a factor of $\Omega(\log n)$ for different orderings. This problem of reordering the alphabet is clearly fixed-parameter tractable in alphabet size σ and has a trivial $O(\sigma!\, n)$ time solution. This may be adequate when σ is small as in DNA sequences. However, this is far from satisfactory from a theoretical point of view, or even from a practical point when the alphabet is slightly larger, such as in protein sequences, natural language texts, ascii texts, etc.

1.1 Related Work

The work in 2018 on block sorting based transformations by Giancarlo et al. gives a theoretical treatment of alphabet ordering in the context of the Generalized BWT [16]. It was shown that for any alphabet ordering, r is at most twice the number of runs in the original text, a result which then holds for the standard BWT as well. Note however that this gives no lower bound on r, and thus gives no results on the approximability of the run minimization problem. There have been multiple previous attempts to develop other approaches to alphabet ordering. In bioinformatics, the role of ordering on proteins was considered in [34] with approaches evaluated experimentally. Similar heuristic approaches evaluated through experiments were done in [1]. Researchers have also considered more restricted versions of this problem. For example, one can try to order a restricted subset of the alphabet, or limit wherein the ordering symbols can be placed. On this problem, heuristics have been utilized. Software tools like BEETL utilize these techniques to handle collections of billions of reads [8]. Another related work in [7] shows, how to permute a given set of strings in linear time, such that the number of runs in the BWT of the (long) string obtained by concatenating the input strings, separated by the same delimiter symbol is minimized.

Even more recently, a work by Giancarlo *et al.*, considered the case where ordering is assigned to the nodes of a string's suffix tree, to minimize the number of runs in the BWT [17]. Interestingly, this problem can be solved in polynomial time. Although their technique can potentially minimize the number of runs in the BWT to an even greater extent than modifying the ordering on the alphabet, it also requires storing the order for each of these nodes, which can require more space. We leave open the problem of finding a trade-off between the strategy of ordering the alphabet and ordering the nodes of the suffix tree.

Given the lack of success with attacking the main problem from the upper bound side, perhaps it is best to approach the problem from the perspective of lower bounds and hardness. To this end, we show why a provably efficient algorithm has been evasive.

2 Problem Definitions and Our Results

Let Σ denotes the alphabet and $\sigma = |\Sigma|$. A run in a string T is a maximal unary sub-string. Let $\rho(T)$ be the number of runs in T.

▶ **Problem 1** (Alphabet Ordering (AO)). *Given a string $T[1, n]$ and an integer t, decide whether there exists an ordering of the symbols in its alphabet such that $\rho(BWT(T)) \leq t$.*

▶ **Theorem 2.** *The alphabet ordering problem is NP-complete and its corresponding minimization problem is APX-hard.*

The problem can be solved in $n \cdot \sigma! = n \cdot 2^{O(\sigma \log \sigma)}$ time naively. However, any significant improvement seems unlikely as per the Exponential Time Hypothesis (ETH) [27].

▶ **Corollary 3.** *Under ETH, AO cannot be solved in time poly$(n) \cdot 2^{o(\sigma)}$.*

It is known that $\rho(BWT(T))$ can be lower bounded by the size of string attractor γ, a recently proposed compressibility measure [22]. Kempa and Kociumaka showed that $\rho(BWT(T))$ can be upper bounded by $O(\gamma \log^2 n)$ [20]. However, γ is independent of the alphabet ordering and the following result is immediate.

▶ **Corollary 4.** *Any alphabet ordering is an $O(\log^2 n)$-approximation for AO.*

We also introduce a specialization of AO, one where we impose more constraints on the ordering given to alphabet symbols.

▶ **Problem 5** (Constrained Alphabet Ordering (CAO))**.** *Given a set of d strings $T_1, \ldots, T_d$ of total length N, find an ordering π on the symbols $\$_i$ $(1 \leq i \leq d)$ such that $\$_{\pi(1)} \prec \$_{\pi(2)} \ldots \prec \$_{\pi(d)} \prec 0 \ldots \prec \sigma - 1$ and $\rho(BWT(T_1\$_1T_2\$_2 \ldots T_d\$_d))$ is minimized.*

We call $\$_1, \$_2, \ldots, \$_d$ *special symbols.* In Section 5.3, we provide an example where an optimal ordering of special symbols removes a factor of $\Omega(\log_\sigma d)$ in the number of runs, demonstrating that this can be a worthwhile preprocessing step. We refer to [8] for an immediate use case in bioinformatics, where the input is a large collection of DNA reads.

▶ **Theorem 6.** *The constrained alphabet ordering problem can be solved in linear time.*

In the full version [4] of this paper, we extend these hardness results to the related problem of ordering source vertex on Wheeler graphs. Wheeler graphs are a recently introduced class of graphs which allow for BWT based indexing [2, 13, 18].

3 Preliminaries: L-reductions

Our inapproximability results use L-reductions [9]. We will be reducing a problem A, with some known inapproximability results, to a new problem B. We will use the following notation:

- $\mathrm{OPT}_A(x)$ denotes the cost of an optimal solution to the instance x of Problem A.
- $c_A(y)$ denotes the cost of a solution y to an instance x of Problem A (suppressing the x in the notation $c_A(x, y)$).
- Since all problems presented here are minimization problems the approximation ratio can be written as $R_A(x, y) = \frac{c_A(y)}{\mathrm{OPT}_A(x)}$, which is ≥ 1.
- Let $f_A(x) = x'$ be a mapping of an instance x of Problem A to instance x' of Problem B.
- Let y' be a solution to instance $x' = f_A(x)$ and $g_B(y') = y$ be the mapping of a solution y' to a solution y for instance x.

Taking x, y, x' y' as above, an L-reduction is defined by the pair of functions (f_A, g_B), computable in polynomial time, such that there exist constants $\alpha, \beta > 0$, where for all x and y the following two conditions hold:

$$\mathrm{OPT}_B(f_A(x)) \leq \alpha\, \mathrm{OPT}_A(x) \quad \text{and} \quad c_A(g_B(y')) - \mathrm{OPT}_A(x) \leq \beta\Big(c_B(y') - \mathrm{OPT}_B(f_A(x))\Big).$$

As a result, $R_B(x', y') = 1 + \varepsilon$ implies $R_A(x, y) \leq 1 + \alpha\beta\varepsilon = 1 + O(\varepsilon)$. L-reductions preserve APX-hardness [32].

4 Hardness of Alphabet Ordering

We will demonstrate a sequence of L-reductions from the $(1,2)$-TSP Cycle problem, where the aim is to find a Hamiltonian cycle of minimum weight through an undirected *complete graph* on n vertices where all edges have weights either 1 or 2. The $(1,2)$-TSP Cycle problem is APX-hard, even with only $\Theta(n)$ edges of weight 1 [33]. The first reduction is to $(1,2)$-TSP Path, where the goal is to find a Hamiltonian path of minimum weight, rather than a cycle.

▶ **Lemma 7.** *(1,2)-TSP Path is APX-hard, even with only $\Theta(n)$ edges of weight 1.*

Proof. We will give an approximation preserving reduction from $(1,2)$-TSP to $(1,2)$-TSP Path. By the APX-hardness of $(1,2)$-TSP Cycle, we obtain Lemma 7.

Let x be the input graph G for $(1,2)$-TSP Cycle and let f_A map the graph G to an identical graph G'. Let g_B map the $(1,2)$-TSP Path y' given to G' to the cycle in G obtained by connecting the end points of the path with an edge of weight at most 2. Hence the cost $c_B(y')$ is always at most the cost $c_A(g_B(y'))$. At the same time, the weight $\mathrm{OPT}_A(x)$ of an optimal cycle in G is bound above by the weight $\mathrm{OPT}_B(f_A(x))$ of an optimal path in G' plus 2. Thus, $c_B(y') \leq c_A(g_B(y'))$ and $\mathrm{OPT}_A(x) \leq \mathrm{OPT}_B(f_A(x)) + 2$. Therefore,

$$\frac{\mathrm{OPT}_B(f_A(x))}{c_B(y')} \leq 1+\varepsilon \implies \frac{\mathrm{OPT}_A(x)}{c_A(g_B(y'))} \leq \frac{\mathrm{OPT}_B(f_A(x))+2}{c_B(y')} \leq 1+\varepsilon+\frac{2}{n} \leq 1+O(\varepsilon). \blacktriangleleft$$

We proceed to present our reduction which consists of two phases.

4.1 Reduction Phase 1

Given a complete graph on n vertices and $m = \Theta(n)$ edges of weight 1 as input to (1,2)-TSP Path, remove all edges of weight 2. We call the resulting graph G. Construct the incidence matrix for G (a row for each edge, and a column for each vertex, where the two 1's in a row indicate which two vertices are incident to the edge for that row). Then add 2ℓ rows of all 0's to bottom of the matrix, where $\ell = 4m$. Next, add two additional columns c_s and c_t where $c_s[i] = 1$ if $i \in \{m+2, m+4, \ldots, m+2\ell\}$ and 0 otherwise, and $c_t[i] = 1$ if $i \in \{m+1, m+3, \ldots, m+2\ell-1\}$ and 0 otherwise (see Figure 2). We call this matrix M.

$$\begin{array}{ccccc|c}
c_s & & & & c_t & \\
0 & ? & \dots & ? & 0 & \\
0 & ? & \dots & ? & 0 & m \\
\vdots & \vdots & \vdots & \vdots & \vdots & \\
0 & ? & \dots & ? & 0 & \\
\hline
0 & 0 & \dots & 0 & 1 & \\
1 & 0 & \dots & 0 & 0 & \\
0 & 0 & \dots & 0 & 1 & \\
1 & 0 & \dots & 0 & 0 & 2\ell \\
\vdots & \vdots & \vdots & \vdots & \vdots & \\
0 & 0 & \dots & 0 & 1 & \\
1 & 0 & \dots & 0 & 0 &
\end{array}$$

Figure 2 The modified incidence matrix for the graph G. Each of the first m rows is for an edge. The bottom $2\ell = 8m$ rows are added as are the outer two most columns.

We now present an intermediate problem that we call Column Ordering (CO), which is: given a matrix M constructed as above, find an optimal ordering on the columns so as to *minimize the number of runs in its linearization.* We will use M_π to denote the matrix M with the ordering π applied to its columns and $L(M_\pi)$ to denote the string obtained by concatenating the rows of M_π from top to bottom. We call $L(M_\pi)$ the linearization of M_π.

Next, we describe the function which maps solutions of our instance of Column Ordering back to a solution of $(1,2)$-TSP Path. Ignoring the added columns c_s and c_t, the ordering π induces a collection of disjoint paths in G, which we call P, where two vertices form an edge if their columns are adjacent and there exists a row with 1's in both columns. Given P we create a (1,2)-TSP Path by connecting the paths in P with $|P|-1$ edges of weight 2. Note that this can be done in linear time.

▶ **Lemma 8.** *If c_s and c_t are the first and last columns of M_π respectively, then the cost of our CO solution is $\rho(L(M_\pi)) = 2m_1 + 4(m - m_1) + 2\ell + 1 = 4m - 2m_1 + 2\ell + 1$, where m_1 is the number of rows whose edges are in the collection of paths P. The corresponding cost of the solution to (1,2)-TSP Path is $m_1 + 2(n - 1 - m_1) = 2(n-1) - m_1$.*

Proof. Ignoring the first run of $L(M_\pi)$ for the moment, every row in M_π corresponding to an edge in P contributes two runs to $\rho(L(M_\pi))$ (e.g. $0\dots0110\dots0$). Any row whose edge is not in P and not in the bottom 2ℓ rows, contributes four (e.g. $0\dots010\dots010\dots0$) and there are $m - m_1$ such (rows) edges. The extra 2ℓ rows in total contribute 2ℓ runs. Adding the "+1" term for the start of $L(M_\pi)$ gives the desired expression. The second statement follows from the TSP Path having m_1 edges of weight 1 and the $n-1$ edges in total needed to form a Hamiltonian path. ◀

▶ **Lemma 9.** *If c_s and c_t are not the first and last columns respectively, then the solution to CO is sub-optimal.*

Proof. If c_t is first and c_s is last, then one extra run is contributed over c_s being first and c_t last, while maintaining the rest of the ordering to be the same. In any configuration where either c_s or c_t are not ends of the matrix, the bottom rows will contribute at least 3ℓ runs. Letting m_1^* denote the optimal number of edges of P, then the optimal $\rho(L(M_{\pi^*}))$ is $4m - 2m_1^* + 2\ell + 1 < 4m + 2\ell \le 3\ell$. Note that the first inequality is strict since we can always find at least one edge for P. ◀

It is immediate from Lemmas 8 and 9 that an optimal solution for CO is one which maximizes m_1, and this provides an optimal solution for (1,2)-TSP Path. We now must show that our reduction is also an L-reduction. Lemmas 10 and 11 consider the two possible cases.

▶ **Lemma 10.** *If c_s and c_t are the first and last columns respectively in a solution to CO, then the L-reduction conditions hold.*

Proof. By Lemmas 8 and 9, the optimal cost for the instance of CO can be expressed as $4m - 2m_1^* + 2\ell + 1$ and the optimal cost for the instance of (1,2)-TSP Path as $2(n-1) - m_1^*$. To prove Condition (i), we need to show there exists an $\alpha > 0$ such that

$$4m - 2m_1^* + 2\ell + 1 \le \alpha(2(n-1) - m_1^*)$$

Since $m = \Theta(n)$ there exists a constant $C > 1$, such that for n large enough $m \le Cn$. The left hand side can be bounded above by $4Cn - 2m_1^* + 8Cn + 1 = 12Cn - 2m_1^* + 1$ (recall $\ell = 4m$). Since $m_1^* \le n - 1$ it is easy to find such an α for $n \ge 2$. Below is the inequality for Condition (ii), which is true for $\beta \ge 1/2$.

$$(2(n-1) - m_1) - (2(n-1) - m_1^*) \le \beta\Big((4m - 2m_1 + 2\ell + 1) - (4m - 2m_1^* + 2\ell + 1)\Big)$$ ◀

▶ **Lemma 11.** *If c_s and c_t are not the first and last columns respectively in a solution to CO, the L-reduction conditions still hold.*

Proof. Condition (i) holds since the optimal solution values to the overall problem have not changed. For Condition (ii), we consider the two scenarios:

- **Scenario 1**: c_s or c_t are not at the far ends of M_π. Then the cost of the solution for CO, which is at least 3ℓ, exceeds the cost for any solution considered in Lemma 10. Furthermore, any corresponding solution for (1,2)-TSP Path has already been considered in Lemma 10, where now the right-hand is larger than it was in Lemma 10.
- **Scenario 2**: c_t is the first column of M_π and c_s is the last. Then, again, we have already considered a solution in Lemma 10 which has solution cost one less for CO and yet had the same solution cost for (1,2)-TSP Path.

This completes the proof. ◀

4.2 Reduction Phase 2

Given the matrix M as constructed in Phase 1 from G, we will now construct a string T as input to the problem AO. It is easier to describe T in terms of its substrings, which are created by iterating through the matrix M as follows:

- For $1 \leq j \leq n+2$, $1 \leq i \leq m+2\ell$: if $M_{i,j} = 1$ output the substring $10^{i+1}2C_j$
- For $1 \leq j \leq n+2$: output the substring $0^{m+2\ell+2}2C_j$
- Append to each substring created above a unique $\$_i$ symbol ($1 \leq i \leq 2m+2\ell+n+2$).

The string T is the concatenation of these substrings in any order and $|T| = O(n^2)$. The alphabet set Σ is $\{0,1,2\} \cup \{C_1, C_2, \ldots, C_{n+2}\} \cup \{\$_1, \$_2, \ldots, \$_{2m+2\ell+n+2}\}$ and $\sigma = \Theta(n)$.

Given a solution π to this instance of AO we use the relative ordering given to the C_i symbols as the ordering for the columns of M_π. For the analysis of why this works, we define some properties that we would like $BWT(T)$ and π to have. For any symbol $a \in \Sigma$ we will call the maximal set of indices where the F column of the sorted circular shift matrix has only a's as the *a-block*. Our goal will be to"simulate" the linearization of $L(M_\pi)$ within the 0-block of $BWT(T)$. We let C_s and C_t denote the symbols for columns c_s and c_t respectively.

The following are the key properties that an optimal solution π^* will have:

1. For a fixed j, all C_j symbols are placed adjacently in $BWT(T)$;
2. All 2 symbols are placed adjacently in $BWT(T)$;
3. The symbol 2 is adjacent to the symbol 0 in the ordering;
4. The $\$_i$ symbols are ordered in such a way as to minimize the number of runs of 1 in the 0-block of $BWT(T)$.
5. The symbols C_s and C_t are both positioned at the beginning and end respectively of the alphabet ordering given to the C_i symbols.

The 0-block of $BWT(T)$ will consist of 0's, 1's, and $\$_i$ symbols. All $\$_i$ symbols will be adjacent within the 0-block. This is since the $\$_i$ symbols succeeded by 0, are all succeeded by the substring $0^{m+2\ell+2}2$ and every occurrence of $0^{m+2\ell+2}2$ preceded by a $\$_i$ symbol (when T is viewed as a circular string). Let r_0 denote the number of runs created in the 0-block of $BWT(T)$, minus the number of $\$_i$ symbols in the 0-block of $BWT(T)$.

▶ **Lemma 12.** *Unless all of the above properties hold, the solution to AO is suboptimal.*

Proof. If any of Properties 1–3 are violated, we can exchange our solution with one which maintains the value r_0 but reduces the runs created in other blocks. This is since the alphabet ordering can be modified to have these properties, while at the same time maintaining the relative orderings of symbols within the 0-block. In the case of Property 4, given that

Properties 1–3 hold, modifying the solution so that the property holds can only decrease r_0, while it maintains the number of runs created in other blocks. Assuming properties 1–4 hold, there are two possibilities, either C_s and C_t are extremal or they are not.

- In the case of being extremal, if $C_s < C_t$, then by Property 4, the $2\ell = 8m$ instances of 1's in the bottom 2ℓ rows of M_π shall correspond to $4m$ runs of two consecutive 1's in the 0-block of $BWT(T)$. The upper rows of M_π shall correspond to at most $2m$ runs of 1's in the 0-block of $BWT(T)$. Hence, in the 0-block there are at most $6m+1$ runs of 1's making at most $6m+2$ runs of zeros to surround them, so that $r_0 \leq 12m+3$. In the case where $C_t < C_s$, one additional run of 1's is created over the same configuration where the positions of C_s and C_t are swapped.
- In the case of them not being extremal, considering only the last 2ℓ rows of M_π, there are $8m$ runs of lonely 1's in the 0-block of $BWT(T)$, and at least $8m+1$ runs of 0's to surround them, leading to $r_0 \geq 16m+1$.

This completes the proof. ◀

As mentioned earlier, we aim to have a substring of $BWT(T)$ within the 0-block which is the same as $L(M_\pi)$ except for the lengths of its runs, i.e., the number of runs will be the same. We will call this substring the simulation of $L(M_\pi)$.

▶ **Lemma 13.** *If all Properties 1–5 hold, then $r_0 = \rho(L(M_\pi))-1$ and $\rho(BWT(T)) = r_0+\sigma-1$.*

Proof. We will first show that when Properties 1–5 hold, $r_0 = \rho(L(M_\pi)) - 1$, i.e., that the simulation works. Within the 0-block of $BWT(T)$, row i is simulated by the characters preceding each substring $0^{i+1}2$. Note that they all appear consecutively in the 0-block. Within the simulation of the i^{th} row, if the value of the j^{th} column of M_π is 0, then the characters preceding substrings of the form $0^{i+1}2C_j$ are all 0. If the value of the j^{th} column of M is 1, then there exists a single substring of the form $0^{i+1}2C_j$ preceded by a 1, and the remaining substrings of the form $0^{i+1}2C_j$ are all preceded by 0. Note that all characters preceding $0^{i+1}2C_j$ are consecutive within the i^{th} row, however, the unique \$'s following each substring allow the characters following each $0^{i+1}2C_j$ to have their orders swapped. Because of Property 5, in the column ordering of M_π there will never be a run of more than two consecutive 1's in $L(M_\pi)$. Hence, when Property 4 is applied, we know that 1's which would are adjacent in $L(M_\pi)$ are adjacent in the 0-block. Combining all these observations gives us that $L(M_\pi)$ is successfully simulated within the 0-block. The "-1" term in the expression for r_0 arises due to Property 2. This is since the 0 symbol in 0-block of $BWT(T)$ that is adjacent to the 2-block does not contribute a run. We have shown $r_0 = L(M_\pi) - 1$.

Finally, the fact that $\rho(BWT(T)) = r_0 + \sigma - 1$ follows from Properties 1–3 which cause every symbol except 1 to contribute exactly one run to $\rho(BWT(T))$ outside of the simulation (1's first appearance is within the simulation). ◀

▶ **Lemma 14.** *If all Properties 1–5 hold, the L-reduction conditions are satisfied.*

Proof. By Lemma's 12 and 13 we have the optimal cost for AO being $r_0^* + \sigma - 1$ and optimal cost for CO as $r_0^* + 1$. For Condition (i) note that $\sigma = \Theta(n)$ and because there are at most 5 runs created by each row, $m + 2\ell \leq r_0^* \leq 5(m+2\ell)$, so that $r_0^* = \Theta(n)$. Hence, we can find an α such that $r_0^* + \sigma - 1 \leq \alpha(r_0^* + 1)$. For Condition (ii), we have $(r_0 + 1) - (r_0^* + 1) \leq \beta((r_0 + \sigma - 1) - (r_0^* + \sigma - 1))$ with $\beta = 1$. ◀

▶ **Lemma 15.** *If any of Properties 1–5 are violated, the L-reduction conditions are satisfied.*

Proof. Condition (i) is satisfied since optimal values for the overall problem are unchanged. For Condition (ii), if any of the first four properties are violated, we have already shown in Lemma 14 that the inequality holds in the harder case where $\rho(L(M_\pi))$ has the same value but the overall number of runs in $BWT(T)$ is less. If the first four properties hold and the fifth property does not hold, there are two cases. In the first case, if C_t is ordered first and C_s last, then swapping C_s and C_t modifies both sides of the inequality for Condition (ii) by the same amount. In the second case, if either C_s or C_t are not ordered first or last, the left hand side of the inequality in Condition (ii), that is $\Big(\rho(L(M_\pi)) - \rho(L(M_{\pi^*}))\Big)$, will be large, as this corresponds to the columns c_s and c_t not being first or last. However, the right-hand side $\Big((r_0 + \sigma - 1) - (r_0^* + \sigma - 1)\Big)$ will be large as well, perhaps even larger as there may exist runs of three of four 1's in $L(M_\pi)$ that cannot be simulated in the 0-block of $BWT(T)$. In particular, $r_0 \geq \rho(L(M_\pi)) - 1$ and $\rho(L(M_{\pi^*})) = r_0^* + 1$, so that with $\beta = 1$

$$\rho(L(M_\pi)) - \rho(L(M_{\pi^*})) \leq (r_0 + 1) - \rho(L(M_{\pi^*})) \leq \beta\Big((r_0 + \sigma - 1) - (r_0^* + \sigma - 1)\Big). \quad ◀$$

We have shown an L-reduction from (1,2)-TSP Path to AO. This combined with Lemma 7 completes the proof for Theorem 2.

4.3 Proof of Corollary 3

Assuming ETH, there exists no $2^{o(n)}$ time algorithm for Hamiltonian Path Problem [10]. Our reduction allows us to determine the minimum number of paths in G needed to cover all the vertices and can hence solve Hamiltonian Path. This can be done by first constructing an incidence matrix for G and then applying the rest of the reduction as in Section 4. Since the alphabet size σ is linear in n and $|T| = \Theta(n^2)$, an $|T|^{O(1)} \cdot 2^{o(\sigma)}$ time algorithm for AO would imply an $2^{o(n)}$ time algorithm for Hamiltonian Path, a contradiction.

5 Constrained Alphabet Ordering

5.1 Reducing to a Simpler Problem

Recall that we wish to find an ordering on the special symbols $\$_1, \ldots, \$_d$ such that the number of runs in the BWT of $T = T_1\$_1 \ldots T_d\$_d$ is minimized and the \$ symbols are lexicographically before other symbols. We will consider our alphabet to be over integers that are bounded by $N^{O(1)}$, where $N = |T|$. Let s be an arbitrary substring of T without \$ symbols. The symbols in T which are followed by $s\$_i$ will form a contiguous portion of $BWT(T)$. However, their ordering within that contiguous portion is determined by the relative ordering given to $\$_i$ symbols. Hence, we can arrange the symbols within this portion of $BWT(T)$ so that identical symbols are placed adjacently.

For example, let $c_1s\$_1$, $c_2s\$_2$, ..., $c_ts\$_t$ be substrings of T. The symbols c_1, c_2, ... c_t will be contiguous in $BWT(T)$ in some order. Now, suppose that $c_2 = c_4 = c_7$. By rearranging the $\$_2$, $\$_4$, and $\$_7$ to be adjacent within the relative ordering of the \$ symbols, we can make c_2, c_4, and c_7 appear consecutively. Taking this one step further, we can also change the relative ordering of $\$_2$, $\$_4$, and $\$_7$, so that if the substrings $\alpha c_2 s\$_2$, $\beta c_4\$_4$, and $\alpha s\$_7$ occur in T, then the two α's will be adjacent in the contiguous portion of $BWT(T)$ corresponding to the substrings $c_2s\$_2$, $c_4s\$_4$, and $c_7s\$_7$.

Hence, the set of symbols $B_s = \{x \mid xs\$_i \text{ is a substring of } T \text{ for some } i \in [1, d]\}$ can be modeled as a tuple where each symbol appears only once within the tuple. Along with each symbol x in B_s, we will maintain a set $\Delta_s^x = \{\$_i \mid xs\$_i \text{ is a substring of } T\}$. We will arrange all non-empty tuples B_s in the lexicographic ordering of s. As such, these tuples can be constructed by first assigning any ordering to the \$ symbols (where they are lexicographically first in the alphabet) and then using the longest common prefix (LCP) between consecutive suffixes in lexicographic order. These values are obtained directly from the longest common prefix array. The suffix array and longest common prefix array can both be constructed in linear time assuming an integer alphabet of size $N^{O(1)}$ [11]. We will define the problem of ordering the symbols within these tuples as a new problem.

▶ **Problem 16** (Tuple Ordering (TO)). *Given a list of tuples $t_1, \ldots, t_q$ in a fixed order, each containing a subset of symbols from Σ, order the symbols in each tuple such that the total number of runs in the string formed by their concatenation $t_1 \cdot t_2 \cdot \ldots \cdot t_q$ is minimized (not considering '(', ')' and commas, of course).*

We will show that TO can be solved in linear time. To map solutions of TO back to solutions of CAO, a tuple for B_s needs to maintain pointers to each tuple B_{xs}, where x is a symbol. Then given a solution to TO, we start with the tuple for B_ε. The ordering given to symbols within this tuple provides us with a partial ordering on the \$ symbols. The symbols in Δ_ε^x associated with the first symbol x within the tuple are ordered before the symbols Δ_ε^y associated with the second symbol y, etc. Then for a symbol x, the tuple for B_x provides a refinement of this partial ordering. In particular, it provides a partial ordering on Δ_ε^x. To recover the total ordering on \$ symbols, we recursively refine the partial ordering at our current tuple by examining all of the tuples which the current tuple points to. Note that this works since for a given tuple for B_s, the sets Δ_s^x are disjoint. The time required to recover this solution is proportional to N.

5.2 Solving the Tuple Ordering Problem in Linear Time

We show how to reduce the TO problem to the single-source shortest path problem on a DAG G, which is constructed as follows. For each tuple t_i, create two sets of vertices L_i and R_i, both of size $|t_i|$, such that for each symbol $c \in t_i$, there exists a vertex with label c in L_i as well as in R_i. Between each pair of vertices $u \in L_i$ and $v \in R_i$, where the label of u is not equal to the label of v, create a directed edge of weight 1 from u to v. If $|t_i| = 1$, then create a directed edge of weight 1 from the unique vertex in L_i to the unique vertex in R_i. For each R_i and L_{i+1} $(1 \leq i \leq q-1)$, and each pair $u \in R_i$ and $v \in L_{i+1}$, create a directed edge from u to v, with weight 1 if they have the same label, and weight 2 otherwise. Finally, create a start vertex s and directed edges of weight 1 from s to each vertex in L_1, and an end vertex e with directed edges of weight 1 from each vertex in R_q to e. See Figure 3 for an illustration.

Clearly, the shortest path from s to e is the one with the fewest edges of weight 2, and this path gives us a tuple ordering which minimizes the number of runs created by the tuples. To obtain this ordering, for a tuple t_i, place as the left-most symbol the label of the vertex used in L_i within the shortest path, and the right-most symbol the label of the vertex used in R_i within the shortest path. The other symbols can be ordered arbitrarily. Because G a DAG, this shortest path can be found in time proportional to the number of edges, which is $O(\sigma^2 q)$. Next, we show how to solve this in time proportional to the number of vertices of G.

Rather than constructing the edges in G, we can work from left-to-right maintaining the shortest path from s to the vertices in our current level of G, either L_i or R_i. Suppose our current level is L_i and we wish to extend the solution to the level R_i. Assuming $|t_i| \geq 2$, we identify the vertices v_1 and v_2 in L_i with the first and second shortest paths (they may have the same length) from s, respectively. For each vertex u in R_i, if the label of u is not the same as the label for v_1, we make the shortest path to u the path from s to v_1, then the edge

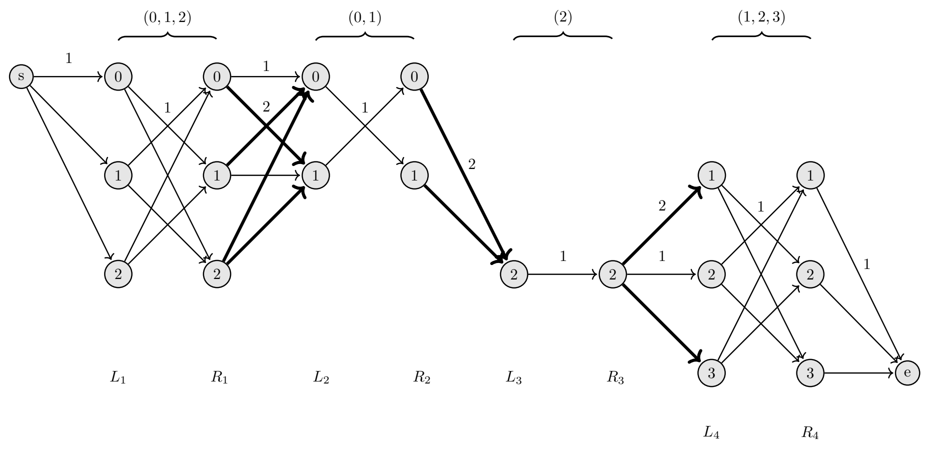

Figure 3 The graph G constructed for the tuple ordering instance $(0, 1, 2), (0, 1), (2), (1, 2, 3)$.

from v_1 to u, otherwise we make it the path from s to v_2, then the edge from v_2 to u. If $|t_i| = 1$, we make the shortest path from s to u the path from s to the unique vertex v in L_i, then the edge from v to the unique vertex u. To extend a solution from R_i to L_{i+1}, we first identify the vertex v_1 in R_i with the shortest path from s. For each vertex u in L_{i+1}, if a vertex with matching label v_u exists in R_i, we take as the shortest path to u the shorter of the following two paths: (i) the path from s to v_1, then from v_1 to u, or (ii) the path from s to v_u, then from v_u to u. If no such vertex with matching label exists in R_i, take as the shortest path from s to u the path from s to v_1, then from v_1 to u.

5.3 An Example of the Effectiveness of CAO

Lastly, we provide an example where the \$ symbol ordering greatly reduces the number of runs in the BWT. Let d be the number of strings and n the length of the strings. It is possible for a set of special symbols to be ordered such that the number of runs is $\Omega(nd)$. Let $\sigma = 2$ and $d = \sigma^n$. Consider the d distinct binary strings concatenated with special symbols in lexicographic order. Under the ordering $\$_1 < \$_2 \dots < \$_d$, the string $BWT(T)$ alternates between the \$'s, 0's, and 1's, yielding $\Omega(nd)$ runs. On the other hand, for this same case, arranging the \$'s in the optimal ordering will give $O(d)$ runs in total. This is since for any substring s of T, the contiguous section of $BWT(T)$ containing the characters preceding $s\$_i$ for $i \in [1, d]$ contains at most the start of two runs. For example, with $n = 3$, we would have $T = 000\$_1001\$_2010\$_3011\$_4100\$_5101\$_6110\$_7111\$_8$. The number of runs in $BWT(T)$ under the naive ordering $\$_1 < \$_2 < \dots < \$_8$, is 32 with $BWT(T) = 01010101010101\$_8\$_101\$_2\$_3010101\$_4\$_501\$_6\$_7$. The number of runs using an optimal ordering $\$_3 < \$_5 < \$_2 < \$_7 < \$_4 < \$_6 < \$_1 < \$_8$ is 19 with $BWT(T) = 00001111110001\$_8\$_101\$_2\$_3001110\$_4\$_501\$_6\$_7$.

References

1 Jürgen Abel. Post BWT stages of the burrows-wheeler compression algorithm. *Softw., Pract. Exper.*, 40(9):751–777, 2010. doi:10.1002/spe.982.

2 Jarno Alanko, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Regular languages meet prefix sorting. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 911–930, 2020. doi:10.1137/1.9781611975994.55.

3 Hideo Bannai, Travis Gagie, et al. Online lz77 parsing and matching statistics with rlbwts. In *Annual Symposium on Combinatorial Pattern Matching (CPM 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

4 Jason Bentley, Daniel Gibney, and Sharma V. Thankachan. On the complexity of bwt-runs minimization via alphabet reordering. *CoRR*, abs/1911.03035, 2019. `arXiv:1911.03035`.

5 Christina Boucher, Travis Gagie, Alan Kuhnle, Ben Langmead, Giovanni Manzini, and Taher Mun. Prefix-free parsing for building big bwts. *Algorithms for Molecular Biology*, 14(1):13, 2019.

6 Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm. *SRC Research Report*, 124, 1994.

7 Bastien Cazaux and Eric Rivals. Linking BWT and XBW via aho-corasick automaton: Applications to run-length encoding. In Nadia Pisanti and Solon P. Pissis, editors, *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy*, volume 128 of *LIPIcs*, pages 24:1–24:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.CPM.2019.24`.

8 Anthony J. Cox, Markus J. Bauer, Tobias Jakobi, and Giovanna Rosone. Large-scale compression of genomic sequence databases with the Burrows–Wheeler transform. *Bioinformatics*, 28(11):1415–1419, May 2012. `doi:10.1093/bioinformatics/bts173`.

9 Pierluigi Crescenzi. A short guide to approximation preserving reductions. In *Proceedings of Computational Complexity. Twelfth Annual IEEE Conference*, pages 262–273. IEEE, 1997.

10 Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. Lower bounds based on the exponential-time hypothesis. In *Parameterized Algorithms*, pages 467–521. Springer, 2015.

11 Martin Farach-Colton, Paolo Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *J. ACM*, 47(6):987–1011, 2000. `doi:10.1145/355541.355547`.

12 Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 390–398, 2000. `doi:10.1109/SFCS.2000.892127`.

13 Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for bwt-based data structures. *Theor. Comput. Sci.*, 698:67–78, 2017. `doi:10.1016/j.tcs.2017.06.016`.

14 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-time text indexing in bwt-runs bounded space. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1459–1477, 2018. `doi:10.1137/1.9781611975031.96`.

15 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in bwt-runs bounded space. *J. ACM*, 67(1), January 2020. `doi:10.1145/3375890`.

16 Raffaele Giancarlo, Giovanni Manzini, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. Block sorting-based transformations on words: Beyond the magic BWT. In *Developments in Language Theory - 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings*, pages 1–17, 2018. `doi:10.1007/978-3-319-98654-8_1`.

17 Raffaele Giancarlo, Giovanni Manzini, Giovanna Rosone, and Marinella Sciortino. A new class of searchable and provably highly compressible string transformations. In Nadia Pisanti and Solon P. Pissis, editors, *30th Annual Symposium on Combinatorial Pattern Matching, CPM 2019, June 18-20, 2019, Pisa, Italy*, volume 128 of *LIPIcs*, pages 12:1–12:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.CPM.2019.12`.

18 Daniel Gibney and Sharma V. Thankachan. On the hardness and inapproximability of recognizing wheeler graphs. In *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany.*, pages 51:1–51:16, 2019. `doi:10.4230/LIPIcs.ESA.2019.51`.

19 Dominik Kempa. Optimal construction of compressed indexes for highly repetitive texts. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1344–1357, 2019. `doi:10.1137/1.9781611975482.82`.

20 Dominik Kempa and Tomasz Kociumaka. Resolution of the burrows-wheeler transform conjecture. *CoRR*, abs/1910.10631, 2019. arXiv:1910.10631.

21 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 827–840, 2018. doi:10.1145/3188745.3188814.

22 Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: string attractors. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 827–840. ACM, 2018. doi:10.1145/3188745.3188814.

23 Alan Kuhnle, Taher Mun, Christina Boucher, Travis Gagie, Ben Langmead, and Giovanni Manzini. Efficient construction of a complete index for pan-genomics read alignment. In *Research in Computational Molecular Biology - 23rd Annual International Conference, RECOMB 2019, Washington, DC, USA, May 5-8, 2019, Proceedings*, pages 158–173, 2019. doi:10.1007/978-3-030-17083-7_10.

24 Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome biology*, 10(3):R25, 2009.

25 Heng Li and Richard Durbin. Fast and accurate long-read alignment with burrows–wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

26 Ruiqiang Li, Chang Yu, Yingrui Li, Tak-Wah Lam, Siu-Ming Yiu, Karsten Kristiansen, and Jun Wang. Soap2: an improved ultrafast tool for short read alignment. *Bioinformatics*, 25(15):1966–1967, 2009.

27 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of the EATCS*, 105:41–72, 2011. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/92.

28 Veli Mäkinen and Gonzalo Navarro. Succinct suffix arrays based on run-length encoding. In *Combinatorial Pattern Matching, 16th Annual Symposium, CPM 2005, Jeju Island, Korea, June 19-22, 2005, Proceedings*, pages 45–56, 2005. doi:10.1007/11496656_5.

29 Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki. Storage and retrieval of individual genomes. In *Research in Computational Molecular Biology, 13th Annual International Conference, RECOMB 2009, Tucson, AZ, USA, May 18-21, 2009. Proceedings*, pages 121–137, 2009. doi:10.1007/978-3-642-02008-7_9.

30 Gonzalo Navarro. *Compact data structures: A practical approach.* Cambridge University Press, 2016.

31 Tatsuya Ohno, Kensuke Sakai, Yoshimasa Takabatake, I Tomohiro, and Hiroshi Sakamoto. A faster implementation of online rlbwt and its application to lz77 parsing. *Journal of Discrete Algorithms*, 52:18–28, 2018.

32 Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. Syst. Sci.*, 43(3):425–440, 1991. doi:10.1016/0022-0000(91)90023-X.

33 Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Math. Oper. Res.*, 18(1):1–11, 1993. doi:10.1287/moor.18.1.1.

34 Lianping Yang, Guisong Chang, Xiangde Zhang, and Tianming Wang. Use of the burrows–wheeler similarity distribution to the comparison of the proteins. *Amino acids*, 39(3):887–898, 2010.