On the Hardness and Inapproximability of Recognizing Wheeler Graphs

Daniel Gibney
Department of Computer Science, University of Central Florida, Orlando FL, USA
http://cs.ucf.edu/~dgibney/
Daniel.Gibney@ucf.edu

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

In recent years several *compressed indexes* based on variants of the Burrows-Wheeler transformation have been introduced. Some of these are used to index structures far more complex than a single string, as was originally done with the FM-index [Ferragina and Manzini, J. ACM 2005]. As such, there has been an increasing effort to better understand under which conditions such an indexing scheme is possible. This has led to the introduction of Wheeler graphs [Gagie et al., Theor. Comput. Sci., 2017]. Gagie et al. showed that de Bruijn graphs, generalized compressed suffix arrays, and several other BWT related structures can be represented as Wheeler graphs, and that Wheeler graphs can be indexed in a way which is space efficient. Hence, being able to recognize whether a given graph is a Wheeler graph, or being able to approximate a given graph by a Wheeler graph, could have numerous applications in indexing. Here we resolve the open question of whether there exists an efficient algorithm for recognizing if a given graph is a Wheeler graph. We present:

- The problem of recognizing whether a given graph $G = (V, E)$ is a Wheeler graph is NP-complete for any edge label alphabet of size $\sigma \geq 2$, even when G is a DAG. This holds even on a restricted, subset of graphs called d-NFA's for $d \geq 5$. This is in contrast to recent results demonstrating the problem can be solved in polynomial time for d-NFA's where $d \leq 2$. We also show the recognition problem can be solved in linear time for $\sigma = 1$;
- There exists an $2^{e \log \sigma + O(n+e)}$ time exact algorithm where $n = |V|$ and $e = |E|$. This algorithm relies on graph isomorphism being computable in strictly sub-exponential time;
- We define an optimization variant of the problem called Wheeler Graph Violation, abbreviated WGV, where the aim is to remove the minimum number of edges in order to obtain a Wheeler graph. We show WGV is APX-hard, even when G is a DAG, implying there exists a constant $C \geq 1$ for which there is no C-approximation algorithm (unless P = NP). Also, conditioned on the Unique Games Conjecture, for all $C \geq 1$, it is NP-hard to find a C-approximation;
- We define the Wheeler Subgraph problem, abbreviated WS, where the aim is to find the largest subgraph which is a Wheeler Graph (the dual of the WGV). In contrast to WGV, we prove that the WS problem is in APX for $\sigma = O(1)$;

The above findings suggest that most problems under this theme are computationally difficult. However, we identify a class of graphs for which the recognition problem is polynomial time solvable, raising the open question of which parameters determine this problem's difficulty.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Burrows–Wheeler transform, string algorithms, suffix trees, NP-completeness

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.51

Related Version A full version of the paper is available at https://arxiv.org/abs/1902.01960.

Funding was received from the U.S. National Science Foundation (NSF) under CCF-1703489, European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 690941.

Acknowledgements We would like to thank T. Gagie and N. Prezza for their valuable feedback.

1 Introduction

Within the last two decades, there has been the development of Burrows Wheeler Transform (BWT) [7] based indices for compressing a diverse collection of data structures. This list includes labeled trees [31], certain classes of graphs [14, 29], and sets of multiple strings [16, 26]. This has motivated the search for a set of general conditions under which a structure can be indexed by a BWT based index, and consequently the introduction of Wheeler graphs. A Wheeler graph is a directed graph with edge labels which satisfies two simple axioms related to the ordering of its vertices. They were introduced by Gagie et al. [17] (also see [2]). Although not general enough to encompass all BWT-based structures (e.g., [18]), the authors demonstrated that Wheeler graphs offer a unified way of modeling several BWT based data structures such as representations of de Bruijn graphs [6, 10], generalized compressed suffix arrays [31], multistring BWTs [27], XBWTs [14], wavelet matrices [9], and certain types of finite automaton [1, 5, 24]. They also showed that there exists an encoding of a Wheeler graph $G = (V, E)$ which requires only $2(e + n) + e \log \sigma + \sigma \log e + o(n + e \log \sigma)$ bits where σ is the size of the edge label alphabet, $e = |E|$, and $n = |V|$. This encoding allows for the efficient traversal of multiple edges while processing characters in a string, using an algorithm similar to the backward search in the FM-index [15]. Since their introduction Wheeler graphs have been further developed using techniques such as tunneling by Alanko et al. [2]. Also, ideas closely coupled to Wheeler graphs have shown up in the recent work by Equi et al. for exact pattern matching on graphs [12, 13]. However, not all directed edge labeled graphs are Wheeler graphs, and thus not all directed edge labeled graphs can have such techniques applied to them. The authors of [17] posed the question of *how to reasonably recognize whether a given graph is a Wheeler graph.*

The question is of both theoretical and practical value, as it might be the first step before attempting to apply some compression scheme to a given graph. For example, one could use the existence of a *Wheeler subgraph* to encode a graph. To do so, you could maintain an encoding of the subgraph using the framework presented in [17] in addition to an adjacency list of the edges not included in the encoding. Depending on the size of the subgraph, such an encoding might provide a large space savings at the cost of a modest time trade-off while traversing the graph. This concept also motivates the portion of the paper where we look at *optimization versions* of this problem that seek subgraphs of the given graph which are Wheeler graphs. Unfortunately for practitioners of such a method, this problem turns out to be computationally difficult as well. As a positive result, we show that the problem of finding a maximal Wheeler subgraph admits a polynomial time algorithm which has solution size within some constant factor of optimal for constant alphabet size. We also show that the problem of recognizing Wheeler graphs is similar to that of identifying the queue number of a graph, indicating a class of graphs where the problem becomes computationally tractable.

1.1 Wheeler Graphs

The notation (u, v, k) is used for the directed edge from u to v with label k. We will assume the usual ordering on the edge labels which come from the alphabet $\{1, 2, ..., \sigma\}$.

▶ **Definition 1.** *A Wheeler graph is a directed graph with edge labels where there exists an ordering π on the vertices such that for any two edges (u, v, k) and (u', v', k'):*

(i) $k < k' \implies v <_\pi v'$;

(ii) $(k = k') \wedge (u <_\pi u') \implies v \leq_\pi v'$.

In addition, vertices with in-degree zero must be placed first in the ordering.

We consider an ordering of the vertices of the graph a *proper* ordering if it satisfies the axioms of the Wheeler graph definition. See Figure 1 for an illustration. One critical property of Wheeler graphs is called *path coherence*. This property is characterized by the fact that if you start at any consecutive range of vertices under the proper ordering π, and traverse the graph following edge labels matching the characters in a string P, then when finished processing P the vertices ended on will form a consecutive range. This property is key to allowing the efficient traversal of multiple edges simultaneously, as well as achieving a compressed representation of the graph.

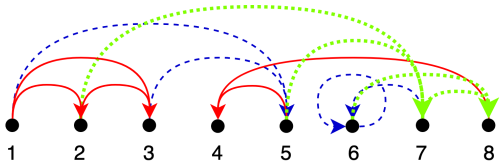

Figure 1 An example of a Wheeler graph with $\sigma = 3$. The ordering on the edge labels is: red (solid) < blue (long-dash) < green (short-dash).

The following list of properties for a Wheeler graph can be deduced from Definition 1.

▶ **Property 1.** *All edges inbound to a vertex v have the same edge label.*

▶ **Property 2.** *In a proper ordering all vertices with same inbound edge label are ordered consecutively.*

▶ **Property 3.** *It is possible for a vertex to have multiple outbound edges with the same label. It is also possible for a vertex to have more than σ inbound or outbound edges.*

▶ **Property 4.** *We call two edges (u, v, k) and (u', v', k) with the same label a rainbow if $u < u'$ and $v' < v$. No rainbows can exist in a proper ordering (see Figure 2).*

▶ **Property 5.** *For $\sigma = 1$, ignoring self-loops, any proper ordering is also a topological ordering. The vertices with self-loops must be placed last in the ordering otherwise a rainbow is created.*

1.2 Problem Definitions

The first question we wish to answer is given a directed graph with edge labels, does a proper ordering π exist? We define this problem formally as the following.

▶ **Problem 1** (Wheeler Graph Recognition). *Given a directed edge labeled graph $G = (V, E)$, answer "YES" if G is a Wheeler graph and "NO" otherwise.*

Although we do not demand it here, ideally, a solution to the above problem would also return a proper ordering.

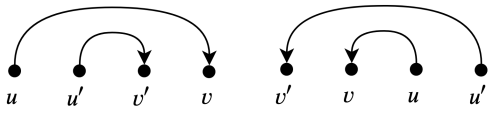

Figure 2 In a proper ordering all of the above configurations cannot occur.

Motivated by the compression of general graphs (which are not necessarily Wheeler graphs), we next define two optimization versions of Problem 1 where we seek to find Wheeler subgraphs.

▶ **Problem 2** (Wheeler Graph Violation (WGV)). *Given a directed edge labeled graph $G = (V, E)$ identify the smallest $E' \subseteq E$ such that $G' = (V, E \backslash E')$ is a Wheeler graph.*

We also consider the dual of this problem.

▶ **Problem 3** (Wheeler Subgraph (WS)). *Given a directed edge labeled graph $G = (V, E)$ identify the largest $E'' \subseteq E$ such that $G'' = (V, E'')$ is a Wheeler graph.*

1.3 Our Contribution

- In Section 2 we show that the problem of recognizing whether a given graph is a Wheeler graph is NP-complete even for an edge alphabet of size $\sigma = 2$. The result holds even when the input is a directed acyclic graph (DAG) and when the number of edges leaving a vertex with the same label is at most five. In the full version of this paper [19] we show that for $\sigma = 1$ the recognition problem can be reduced to that of determining if a DAG has queue number one which can be solved in linear time [22].
- In Section 3 we provide an exponential time algorithm which solves the recognition problem on a graph $G = (V, E)$ in time $2^{O(n+e \log \sigma)}$ where $n = |V|$ and $e = |E|$. It uses the idea of enumerating through all possible encodings (of bounded size) of Wheeler graphs, and the fact that we can test whether there exists an isomorphism between two undirected graphs in sub-exponential time. This technique also gives us exact algorithms for the optimization problems presented in this paper which run in time $2^{O(n+e \log \sigma)}$.
- Section 4 examines the optimization versions of this problem called Wheeler Graph Violation (WGV) and Wheeler Subgraph (WS). We show via a reduction of the Minimum Feedback Arc Set problem that the Wheeler Graph Violation problem is APX-hard, and assuming the Unique Games Conjecture, cannot be approximated within a constant factor. This holds even when the graph is a DAG. On the other hand, we show that the Wheeler Subgraph problem is in the complexity class APX for $\sigma = O(1)$. We do so by providing a poly-time algorithm whose solution size is $\Omega(1/\sigma)$ times the optimal value.
- Using PQ-trees and ideas similar to those used in detecting if the queue number of a DAG is one, we demonstrate a class of graphs where Wheeler graph recognition can be done in linear time (see full version [19]).

2 NP-completeness of Wheeler Graph Recognition

▶ **Theorem 2.** *The Wheeler Graph Recognition Problem is NP-complete for any $\sigma \geq 2$.*

We first show a simple reduction from the Betweenness problem to Wheeler Graph Recognition. Although straightforward, it requires graphs with either $O(n)$ sources or $O(n)$ edges with the same label leaving a single vertex. In Section 2.3, by expanding on the techniques used in the first reduction we show that even if these quantities are limited to at most five the recognition problem remains NP-complete.

2.1 The Betweenness Problem

The Betweenness Problem was established as NP-complete by Opatrný in 1979 [30]. Like our problem, it deals with finding a total ordering on a set of elements subject to some constraints. The input to the Betweenness problem is a list of distinct elements $T = t_1, \ldots, t_n$ and a collection of $k < n^3$ ordered triples of $(t_1^1, t_2^1, t_3^1), (t_1^2, t_2^2, t_3^2), \ldots (t_1^k, t_2^k, t_3^k)$ where every element

in a triple is in T. The list T should be placed into a total ordering with the property that for each of the given triples the middle item in the triple appears somewhere between the other two items. The items of each triple are not required to be consecutive in the total ordering. The decision problem is determining if such an ordering is possible.

As an example, consider the input $T = 1, 2, 3, 4, 5, 6$ and the triples: $(5, 2, 3)$, $(1, 5, 2)$, $(4, 5, 6)$, $(4, 6, 2)$. A total ordering which satisfies the given triples is $1, 4, 5, 6, 2, 3$. An ordering which does not satisfy the given triples is $1, 2, 3, 4, 5, 6$ since it violates the triples $(5, 2, 3)$, $(1, 5, 2)$, and $(4, 6, 2)$.

2.2 Reduction from Betweenness to Wheeler Graph Recognition

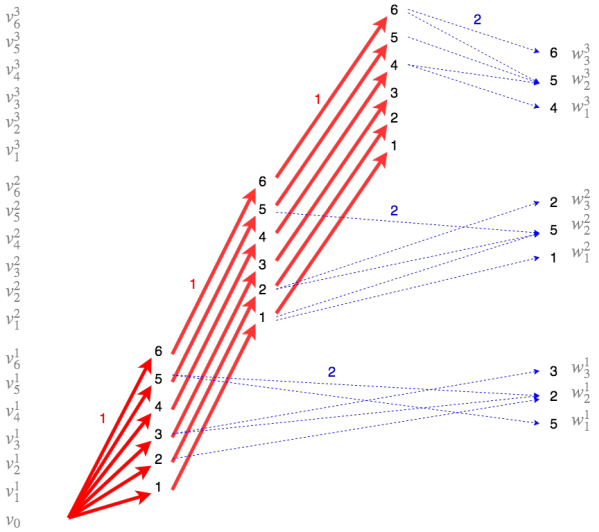

Figure 3 An example of the reduction with input list 1,2,3,4,5,6 and triples $(5, 2, 3)$,$(1, 5, 2)$,$(4, 5, 6)$.

Suppose we are given as input to the Betweenness Problem the list $t_1, t_2, \ldots, t_n$ and triples $(t_1^1, t_2^1, t_3^1), (t_1^2, t_2^2, t_3^2), \ldots, (t_1^k, t_2^k, t_3^k)$. We construct a DAG of size $O(nk)$ as follows:

- Create a source vertex v_0 and vertices v_i^j for $1 \leq i \leq n$ and $1 \leq j \leq k$.
- For each triple (t_1^j, t_2^j, t_3^j) create a vertex for each element of the triple, we call them w_1^j, w_2^j, and w_3^j respectively.
- Create the edges $(v_0, v_i^1, 1)$ and edges $(v_i^j, v_i^{j+1}, 1)$ for $1 \leq i \leq n,\ 1 \leq j \leq k-1$.
- Create the following edges:
  - $(v_i^j, w_1^j, 2)$ if $t_i = t_1^j$
  - $(v_i^j, w_2^j, 2)$ if $t_i = t_2^j$
  - $(v_i^j, w_3^j, 2)$ if $t_i = t_3^j$
  - $(v_i^j, w_2^j, 2)$ if $t_i = t_1^j$
  - $(v_i^j, w_2^j, 2)$ if $t_i = t_3^j$

▶ **Lemma 3.** *An instance of the Betweenness problem has an ordering satisfying all of the constraints iff the graph constructed as above is a Wheeler graph.*

Proof Sketch. The intuition is that the vertices with inbound edge label 1 represent the permutation of the elements in T. The edges labeled 1 force the permutation to be duplicated k times, once for each constraint. The vertices with the inbound edge label 2 represent the

elements in each triple. The edges with label 2 enforce that the only valid orderings of the vertices representing elements in T are orderings that satisfy the Betweenness constraints. This is enforced by having no edges labeled 2 which are crossing in the figure. The detailed proof can be found in the full version of this paper [19]. ◀

Theorem 2 follows from Lemma 3. The fact that being a Wheeler graph implies (arched) level planarity with respect to each edge label is the key to the reduction.

The Wheeler graph recognition problem can be solved in linear time for an alphabet of size one. This follows from relating the notion of queue number to Wheeler graphs, and previous results which give a linear time algorithm for finding a one-queue DAG [21, 22, 23]. This also gives an upper bound on the number of edges which can be in a Wheeler graph [11]. The proof can be found in the full version of this paper [19].

▶ **Theorem 4.** *The Wheeler graph recognition problem can be solved in linear time for an edge alphabet of size $\sigma = 1$.*

▶ **Theorem 5.** *For $\sigma = 1$, the number of edges in a Wheeler graph is $O(n)$.*

2.3 NP-completeness of Wheeler Graph Recognition on d-NFA's

Now we restrict the number of edges with the same label that can leave a single vertex. We adopt the terminology used by Alanko, Policriti and Prezza and consider the problem of recognizing whether a d-NFA is also a Wheeler graph [3]. A d-NFA is defined as follows:

▶ **Definition 6.** *A d-NFA G is an NFA where the number of edges with the same character leaving a vertex is at most d. We refer to the value d as the non-determinism of G.*

We emphasize that here an NFA contains a single start state, from which we assume each vertex is reachable. The results in this section are in contrast to the recent work of Alanko, Policriti and Prezza who showed that it can be recognized in polynomial time whether a 2-NFA is a Wheeler graph [3]. Their result coupled with the observation that the reduction in Section 2 requires a $n^{\Theta(1)}$-NFA suggests an interesting question about what role non-determinism plays in the tractability of Wheeler graph recognition. To this end, we prove Theorem 7.

▶ **Theorem 7.** *The Wheeler Graph Recognition Problem is NP-complete for d-NFA's, $d \geq 5$.*

The strategy of the proof is to reduce the NP-complete problem 4-NAESAT to Wheeler Graph Recognition. In 4-NAESAT each clause is of length 4, and an expression is satisfiable iff there exists a truth assignment such that each clause contains both a true literal and a false literal. We start with 4-NAESAT, rather than 3-NAESAT, to obtain a 3-NAESAT instance with the special property highlighted by Lemma 8.

▶ **Lemma 8.** *An instance ϕ of 4-NAESAT can be reduced in poly-time to an instance ϕ' of 3-NAESAT where a variable occurring in the middle of a clause appears at most twice in ϕ'.*

Proof. Convert the 4-NAESAT instance ϕ to a 3-NAESAT instance ϕ' by converting each clause (a_k, b_k, c_k, d_k) into the clauses (a_k, w_k, b_k) and $(c_k, \overline{w_k}, d_k)$. Both clauses have a satisfying not-all-equal assignment iff it is not the case that $a_k = b_k = c_k = d_k$. We note that the variable used in the middle of the clauses, w_k, is used only twice in ϕ'. ◀

For convenience, we define a case of 3-NAESAT where each variable occuring in the middle occurs at most twice, we call this 3-NAESAT*. We next describe the construction of a one source DAG from an instance of 3-NAESAT*.

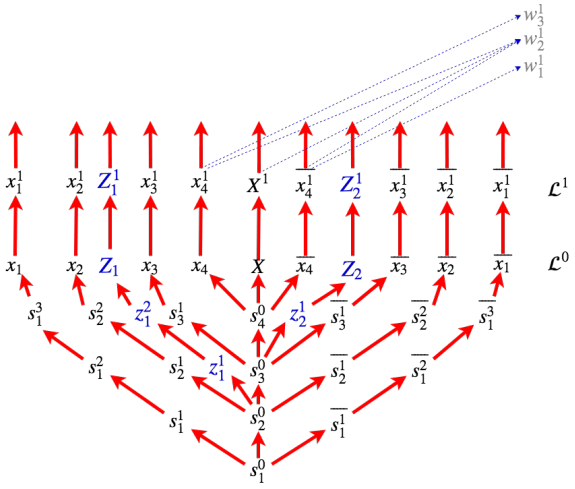

Figure 4 Vertex Z_1 and Z_2 could be for clauses (x_1, x_2, x_3), $(x_2, \overline{x_3}, x_4)$. Each "betweenness" constraint adds a layer. $(x_4, X, \overline{x_4})$ constraint shown.

Suppose we are given an instance ϕ of 3-NAESAT* with variables $x_1, x_2, \ldots, x_n$ and the clauses (a_k, b_k, c_k) where we assume a_k, b_k, c_k can represent either a Boolean variable or its negation. We create a single source DAG G based on ϕ. The first step creates a *menorah like* structure which allows for the vertices representing x_i and $\overline{x_i}$ to swap places in G, but otherwise fixes the positions of the vertices. We begin by adding the vertices which represent our variables, $x_1, \ldots, x_n, X, \overline{x_1}, \ldots, \overline{x_n}$; (the role of X will become clear). Next, we add the structure to constrain their possible positions.

Add to G the vertices:

- $s_1^0 \ldots, s_n^0$;
- For $1 \leq i \leq n-1$, $1 \leq j \leq n-i$: s_i^j and $\overline{s_i^j}$;

Add to G the red edges:

- $(s_1^0, s_2^0, 1), \ldots (s_n^0, X, 1)$;
- For $1 \leq i \leq n-1$, $2 \leq j \leq n-i$: $(s_i^{j-1}, s_i^j, 1)$ and $(\overline{s_i^{j-1}}, \overline{s_i^j}, 1)$;
- For $1 \leq i \leq n-1$: $(s_i^0, s_i^1, 1)$ and $(s_i^0, \overline{s_i^1}, 1)$;
- For $1 \leq i \leq n$: $(s_i^{n-i}, x_i, 1)$ and $(\overline{s_i^{n-i}}, \overline{x_i}, 1)$;

For clause k, denoted (a_k, b_k, c_k), we add a vertex Z_k. Suppose the middle variable of the clause, b_k, is x_h (positive or negated), then we add the vertices z_k^j for $1 \leq j \leq n-h$, and red edges $(s_h^0, z_k^1, 1), (z_k^1, z_k^2, 1) \ldots (z_k^{n-h}, Z_k, 1)$.

Now we wish add a set of *betweenness* type constraints on any proper ordering given of the vertices $\mathcal{L}^0 = \{x_1, \ldots, X, \overline{x_n} \ldots \overline{x_1}, Z_1, Z_2, \ldots\}$. Given a constraint (y_1, y_2, y_3) we insist y_2 be between y_1 and y_3 in the ordering. This can be enforced by adding a layer of new vertices $\mathcal{L}^1 = \{x_1^1, \ldots, X^1, \overline{x_n^1} \ldots \overline{x_1^1}, Z_1^1, Z_2^1, \ldots\}$ with red edges labeled 1 from vertices in layer $\mathcal{L}^0$ to their corresponding vertices in $\mathcal{L}^1$. We use the same gadget that was used in Section 2. Consider adding a betweenness on the vertices y_1, y_2, y_3 in $\mathcal{L}^0$. Add the vertices w_1^1, w_2^1, and w_3^1 and the blue edges $(y_1^1, w_1^1, 2)$, $(y_2^1, w_2^1, 2)$, $(y_3^1, w_3^1, 2)$, $(y_1^1, w_2^1, 2)$ and $(y_3^1, w_2^1, 2)$. Additional betweenness constraints can be similarly enforced by adding a new layer on top of $\mathcal{L}^1$ with a new gadget. Add the betweenness constraints $(x_i, X, \overline{x_i})$ for $1 \leq i \leq n$ fixing X, and betweenness constraints (a_k, Z_k, b_k) and (c_k, X, Z_k) for every clause (a_k, b_k, c_k).

Table 1 Possible relative orderings of a_k, b_k, c_k, Z_k, X subject to (a_k, Z_k, b_k) and (c_k, X, Z_k).

Possible Orderings (a_k has variable x_j and c_k has variable x_h)		
$a_k b_k c_k$	$j < h$	$h < j$
FFT	$c_k \dots X \dots b_k, Z_k \dots a_k$	$c_k \dots X \dots b_k, Z_k \dots a_k$
FTF	$b_k, Z_k \dots X \dots c_k \dots a_k$	$b_k, Z_k \dots X \dots a_k \dots c_k$
TFF	$a_k \dots \overline{b_k}, Z_k \dots X \dots b_k \dots c_k$	$a_k \dots \overline{b_k}, Z_k \dots X \dots b_k \dots c_k$
FTT	$c_k \dots b_k \dots X \dots \overline{b_k}, Z_k \dots a_k$	$c_k \dots b_k \dots X \dots \overline{b_k}, Z_k \dots a_k$
TFT	$a_k \dots c_k \dots X \dots Z_k, b_k$	$c_k \dots a_k \dots X \dots Z_k, b_k$
TTF	$a_k \dots Z_k, b_k \dots X \dots c_k$	$a_k \dots Z_k, b_k \dots X \dots c_k$

Table 2 Orderings implied by all-equal assignment are not possible while satisfying constraints.

(**Not**) Possible Orderings (a_k has variable x_j and c_k has variable x_h)		
$a_k b_k c_k$	$j < h$	$h < j$
TTT	$a_k \dots b_k \dots c_k \dots X$	$c_k \dots b_k \dots a_k \dots X$
FFF	$X \dots c_k \dots b_k \dots a_k$	$X \dots a_k \dots b_k \dots c_k$

Before proving the correctness of the reduction, we make the observation that because any variable occurring in the middle of a clause occurs as most **twice** in the whole expression, the maximum number of edges leaving a vertex s_i^0 is bounded by $3 + \mathbf{2} = 5$. All of the other vertices have at most three edges with the same label leaving them.

▶ **Lemma 9.** *The leveled graph G constructed as above from an instance ϕ' of 3-NAESAT* is a Wheeler graph iff ϕ' is satisfiable.*

Proof. Given a truth assignment that satisfies the 3-NAESAT* instance ϕ', put the vertices in $\mathcal{L}^0$ whose variables are assigned the value T on the left side of X in Figure 4, and the vertices whose variables are assigned F on the right side of X. For example, if $x_1 = T, x_2 = F$, the two left-most vertex on level $\mathcal{L}^0$ would be x_1 followed by $\overline{x_2}$. Now, for clause (a_k, b_k, c_k) we have the possible not-all-equal truth assignments and relative orderings of $\mathcal{L}^0$ which satisfy the Wheeler graph axioms in Table 1. This shows that a Wheeler graph ordering of the vertices is possible by placing Z_k's correctly given the truth assignment.

In the other direction, if G is a Wheeler graph then the ordering of the menorah structure is fixed with the exception of z_i^j vertices and the ordering duplicated across layers $\mathcal{L}^0, \mathcal{L}^1, \dots$. We will show the ordering given to $\mathcal{L}^0$ must have every clause getting a not-all-equal assignment. Suppose to the contrary that $\mathcal{L}^0$ was given an ordering where either a_k, b_k, c_k are all on the left(true) or the right side(false) of X. Then we have the options in Table 2.

In all cases listed in Table 2, placing Z_k between a_k and b_k violates the constraint (c_k, X, Z_k), implying we violate a Wheeler graph constraint as well, a contradiction. Hence, if G is a Wheeler graph, a valid ordering for $\mathcal{L}^0$ implies a valid truth assignment for ϕ'. ◀

This leaves open the complexity of the recognition problem for 3-NFA's and 4-NFA's.

3 An Exponential Time Algorithm

We can apply the encoding introduced by Gagie et al. [17] to develop exponential time algorithms to solve all of the problems presented in this paper. The idea is to enumerate over all possible encodings of Wheeler graphs with the proper number of vertices, edges, and labels, checking whether the encoding is isomorphic with the given graph. This idea exploits

the fact that having such a space efficient encoding also implies have a limited search space of Wheeler graphs. The graph isomorphism can be checked efficiently enough to maintain the desired time complexity. The results are summarized in the following two theorems.

▶ **Theorem 10.** *Recognizing whether $G = (V, E)$ is a Wheeler graph can be done in time $2^{e \log \sigma + O(n+e)}$, where $n = |V|$, $e = |E|$, and σ is the size of the edge label alphabet.*

Before describing the algorithm that proves Theorem 10 we need to describe the encoding of a Wheeler graph given in [17]. A Wheeler graph can be completely specified by three bit vectors. Two bit vectors I and O both of length $e + n$ and a bit vector L of length $e \log \sigma$. We assume that the vertices of the Wheeler graph G are listed in a proper ordering $x_1 <_\pi x_2 <_\pi \ldots <_\pi x_n$. The array I is of the form $0^{\ell_1}10^{\ell_2}1 \ldots 0^{\ell_n}1$ and O is of the form $0^{k_1}10^{k_2}1 \ldots 0^{k_n}1$. Here ℓ_i is the out-degree of x_i whereas k_i is the in-degree of x_i. The array L indicates which of the e character symbols are assigned to each edge. Specifically, the i^{th} character in L gives us the label of the edge corresponding to the i^{th} zero in O. In [17] an additional C array is added, and these arrays are equipped with additional rank and select structures to allow for efficient traversal as is done in the FM-index [15]. For our purposes, however, the arrays O, I, and L are adequate.

The outline of the algorithm is given below as Algorithm 1. It essentially enumerates all bit vectors of a given length, checks whether or not the bit vector encodes a valid Wheeler graph, and if so then checks whether the encoding matches our given graph G. Let S represent the set of all possible encodings we wish to check. Note that $|S| \leq 2^{2(e+n)+e \log \sigma}$.

Algorithm 1 IdentifyWheelerGraph(G).

```
for all (O, I, L) ∈ S do
    if (O, I, L) defines a valid wheeler graph G' then
        convert G to undirected graph α(G)
        convert G' to undirected graph α(G')
        if α(G) and α(G') are isomorphic then
            return "Wheeler Graph"
        end if
    end if
end for
return "Not a Wheeler Graph"
```

The Wheeler graph corresponding to the encoding can be extracted by working from right to left reading the array I. For each zero in I, we know which symbol should be on the inbound edge going into the corresponding vertex. We only need to decide where the edge's tail was. Let k be the edge label and j be the index of the label k in L which is furthest to the right in L and yet to be used. If no such j exists we reject the encoding. When assigning the tail for an edge, take as the tail the vertex x_i where $i = rank_1(O, select_0(O, j))$. We call the graph constructed in this way G'.

We now wish to check whether G' and G are the same graphs only with a reordering of the vertices, that is, G' is the result of applying an isomorphism to G. Unlike the typical isomorphism for labeled graphs, where a bijection between the symbols on the edge alphabet is all that is required, here we wish for the adjacency and the label on the edge to be preserved in the mapping between G and G'. Specifically, we wish to know if there exists a bijective

function $f : V(G) \to V(G')$, such that if $u, v \in V(G)$ are adjacent via an edge (u, v, k) with label k in G, then $f(u)$ and $f(v)$ are also adjacent via an edge $(f(u), f(v), k)$ with label k in G'. Using ideas similar to those presented by Miller in [28], this problem can be reduced in polynomial time to checking whether two undirected graphs are isomorphic.

▶ **Lemma 11.** *Checking whether the direct edge labeled graph G' is edge label preserving isomorphic to G can be reduced in polynomial time to checking if two undirected graphs are isomorphic.*

Proof. See full version [19]. ◀

The final step in this algorithm is to check whether $\alpha(G)$ and $\alpha(G')$ are isomorphic. Using well established techniques this can be done in time $2^{\sqrt{n'}+O(1)}$ where n' is the number of vertices in $\alpha(G)$ [4]. The total time complexity of Algorithm 1 is the number of bit strings tested, multiplied by the time it takes to (1) validate whether the bit string encodes a Wheeler graph G' and decode it, (2) convert G and G' to undirected graphs $\alpha(G)$ and $\alpha(G')$, and (3) test whether $\alpha(G)$ and $\alpha(G')$ are isomorphic. This yields an overall time complexity of $|S|n^{O(1)}2^{\sqrt{n+2e(\sigma+1)}+O(1)}$, i.e., $2^{e \log \sigma + O(n+e)}$ for Algorithm 1.

4 Optimization Variants to Wheeler Graph Recognition

4.1 The Wheeler Graph Violation Problem is APX-hard

In this section, we show that obtaining an approximate solution to the WGV problem that comes within a constant factor of the optimal solution is NP-hard. We do this through a reduction that shows that WGV is at least as hard as solving the Minimum Feedback Arc Set problem (FAS). FAS in its original formulation is phrased in terms of a directed graph where the objective is to find the minimum number of edges which need to be removed in order to make the directed graph a DAG. A slightly different formulation proves more useful for us. Letting $F_\pi = \{(v_i, v_j) \in E \mid \pi(v_i) > \pi(v_j)\}$ we have the following:

▶ **Lemma 12** (Younger [32]). *Determining a minimum feedback arc set for $G = (V, E)$ is equivalent to finding an ordering π on V for which $|F_\pi|$ is minimized.*

From this, we can present the equivalent formulation of FAS.

▶ **Definition 13** (Minimum Feedback Arc Set (FAS)). *The input is a list $T = t_1 t_2 \dots t_n$ of n numbers and a set of k inequalities of the form $t_i < t_j$. This task is to compute an ordering π on T so that the number of inequalities violated in minimized.*

Interestingly, we could not have used FAS for proving that the Wheeler graph recognition problem is NP-complete, as FAS is fixed-parameter tractable in terms of the size of the feedback arc set [8]. Indeed, setting the size of the feedback arc-set to zero is equivalent to checking if the given graph is a DAG and the problem becomes solvable in polynomial time.

On the other hand, it has been shown that FAS is APX-hard, meaning that every problem in APX is reducible to it [25]. It also implies, assuming NP $\neq$ P, that there is a constant $C \geq 1$ such that there is no polynomial time algorithm which provides a C-approximation. The reduction provided in this section implies:

▶ **Theorem 14.** *The WGV problem is APX-hard.*

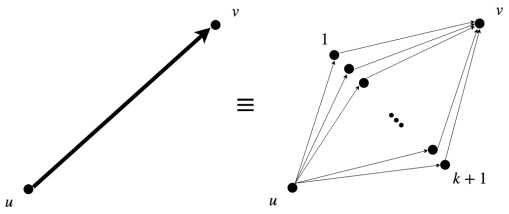

Figure 5 A bold edge in Figure 6 is actually $k+1$ subdivided edges.

In addition, Guruswami et al. demonstrated that assuming the Unique Games Conjecture holds, and NP $\neq$ P, there is no constant $C \geq 1$ such that a polynomial-time algorithm's approximate solution to FAS is always a factor C from the optimal solution. We state this as a lemma.

▶ **Lemma 15** (Guruswami et al. [20]). *Conditioned on the Unique Games Conjecture, for every $C \geq 1$, it is NP-hard to find a C-approximation to FAS.*

An approximation preserving reduction from FAS to WGV combined with Lemma 15 proves the other main result of this section:

▶ **Theorem 16.** *Conditioned on the Unique Games conjecture, for every constant $C \geq 1$, it is NP-hard to find a C-approximation to WGV.*

4.2 The Reduction of FAS to WGV

Let $T = t_1, t_2, \ldots, t_n$ and inequalities $t_1^1 < t_2^1, t_1^2 < t_2^2, \ldots, t_1^k < t_2^k$ be the input to FAS.

We define a heavy edge between the vertices u and v with label ℓ as $k+1$ subdivided edges between u and v each with label ℓ. That is, a heavy edge between u and v with label ℓ consists of the edges (u, w_i, ℓ) and (w_i, v, ℓ) for $1 \leq i \leq k+1$. See Figure 5 for an illustration. Use the following steps to create a graph (which is a DAG):

- Create a vertex v_0 and vertices v_i^j for $1 \leq i \leq n+1$ and $1 \leq j \leq k$.
- For each inequality $t_1^j < t_2^j$ create a vertex for both t_1^j and t_2^j, labeled w_1^j and w_2^j, respectively.
- Create heavy edges $(v_0, v_i^1, 1)$ for $1 \leq i \leq n+1$ and heavy edges $(v_i^j, v_i^{j+1}, 1)$ for $1 \leq i \leq n+1$, $1 \leq j \leq k-1$.
- Create heavy edges $(v_0, w_1^1, 2)$, and heavy edges $(v_{n+1}^j, w_2^j, 2)$ and $(v_{n+1}^j, w_1^{j+1}, 2)$ for $1 \leq j \leq k-1$, and heavy edge $(v_{n+1}^k, w_2^k, 2)$.
- Add the regular (not heavy) edges $(v_i^j, w_1^j, 2)$ if $t_i = t_1^j$, and $(v_i^j, w_2^j, 2)$ if $t_i = t_2^j$ for $1 \leq i \leq n$, $1 \leq j \leq k$.

An example of the reduction is given in Figure 6. The intuition is that the vertices with an inbound heavy edge labeled 1 represent the permutation of the elements in T. The heavy edges labeled 1 force the permutation to be duplicated k times, once for each constraint. The vertices with the inbound edge label 2 represent the elements in each inequality. We will show that this is an approximation preserving reduction.

Let E' be a solution to WGV and $G' = (V, E \backslash E')$ and let π represent a proper ordering on the vertices of G'. Lemma 17 indicates that, other than permuting the ordering found on the vertices v_i^j for $1 \leq i \leq n$ (with the ordering duplicated for $1 \leq j \leq k$), the ordering for the vertices in Figure 6 is fixed.

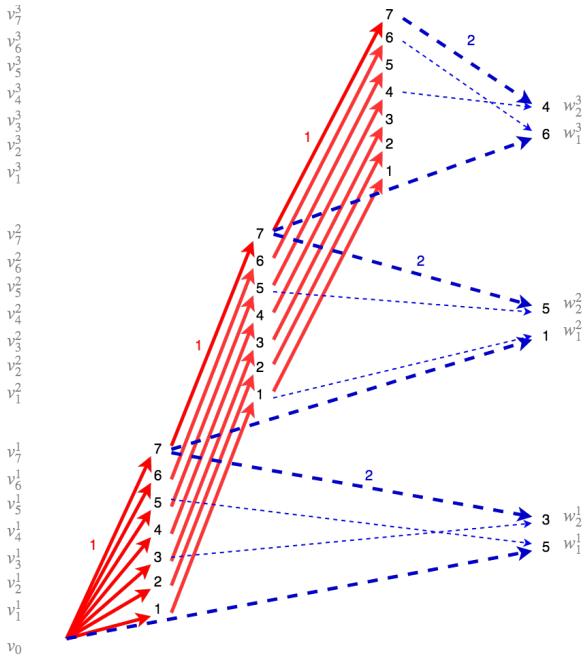

Figure 6 An example of the reduction from FAS to WGV where $T = 1, 2, 3, 4, 5, 6$ and the set of inequalities is $5 < 3$, $1 < 5$, and $6 < 4$.

▶ **Lemma 17.** *Let ϕ represent a permutation of the set $[n+1]$. Any ordering π which provides a solution to the WGV instance is of the form*

$$v_0, v^1_{\phi(1)}, v^1_{\phi(2)}, \dots v^1_{\phi(n+1)}, \dots v^k_{\phi(1)}, v^k_{\phi(2)}, \dots v^k_{\phi(n+1)}, w^1_1, w^1_2, w^2_1, w^2_2, \dots w^k_1, w^k_2.$$

Proof. See full version [19]. ◀

Let $f(x)$ refer to the reduction described above applied to an instance x of FAS creating an instance $f(x)$ of WGV. We also refer to the solution to either of these problems as $\mathrm{OPT}(\cdot)$, and $val(\cdot)$ as the cost function. For instance x of FAS $val(x)$ is the number of violated inequalities and for an instance $f(x)$ of WGV $val(f(x))$ it is the number of violating edges.

▶ **Lemma 18.** *Given an instance x of FAS, a solution(or sub-optimal solution) to the instance $f(x)$ of WGV that has $\ell \leq k$ axiom violating edges yields a solution(or sub-optimal solution) to x with ℓ violated inequalities. The converse holds as well.*

Proof. See full version [19]. ◀

▶ **Lemma 19.** *Given an instance x of FAS, a C-approximation to the solution $OPT(f(x))$ yields a C-approximation to the solution $OPT(x)$.*

Proof. By Lemma 18 any (sub-optimal) solution with objective value $C \cdot val(\mathrm{OPT}(f(x)))$ to $f(x)$, gives us a (sub-optimal) solution to x with the same objective value, $C \cdot val(\mathrm{OPT}(x))$. ◀

Theorem 14 follows from Lemma 19 and Theorem 16 follows from Lemma 19 and Lemma 15.

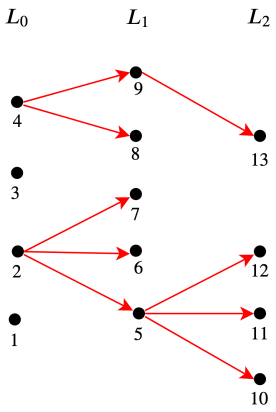

Figure 7 Arborescences have their roots aligned in level L_0. The relative ordering for each type of vertex can be read from top to bottom, left to right.

4.3 The Wheeler Subgraph Problem is in APX

The dual problem to WGV is the problem of finding the largest subgraph of G which is a Wheeler graph. This problem (defined in Section 1.2) is called Wheeler Subgraph problem, abbreviated WS. Unlike WGV, this problem yields a $\Theta(1)$-approximate solution for constant σ.

We first prove the result for $\sigma = 1$. We then apply this result to get an approximation for $\sigma > 1$. The proof for $\sigma = 1$ uses a branching of a directed graph. A branching is a set of arborescence where an arborescence is a directed, rooted tree where all edges point away from the root. A branching is spanning in that every vertex in V is included exactly one arborescence in the branching.

▶ **Lemma 20.** *There exists a linear time $\Theta(1)$-approximation algorithm for WS when the alphabet set size σ is one.*

Proof. Let V_0 be the set of sources in G (vertices with in-degree zero). There are two cases:

Case $|V_0| \leq n/2$: Take a branching $\mathcal{F}$ of the input graph G such that each vertex with in-degree greater than zero is included in some non-singleton arborescence whose root is a source vertex in V_0. Let $|\mathcal{F}|$ denote the total number of arborescences in $\mathcal{F}$. Since $|V_0| \leq n/2$, it follows that $|\mathcal{F}| \leq n/2$ as well.

We create a planar leveling $(L_0, L_1, \ldots)$ of $\mathcal{F}$ by aligning all roots of the branching on level L_0 in an arbitrary order. Then set L_i to be all of the vertices which are distance i from some root in L_0. Because these are trees, we can order the vertices in levels in such a way that the leveling is planar (and for the purpose of visualization say left to right as in Figure 7).

We claim that $\mathcal{F}$ is a Wheeler graph and that we can obtain a proper ordering π for the vertices of $\mathcal{F}$ from this leveling. Starting with V_0, we order the vertices on each level from the bottom to top before proceeding right to the next level. One can check that the Wheeler graph axioms are satisfied.

The number of edges in $\mathcal{F}$, denoted $e(\mathcal{F})$, is equal to $n - |\mathcal{F}|$. And, since $|\mathcal{F}| \leq n/2$, we have that $e(\mathcal{F}) \geq n/2$. At the same time, by Theorem 5 the optimal number of edges, denoted $|E^*|$, is $\Theta(n)$. The ratio of the optimal solution value over the branching solution value is bounded. In particular, $|E^*|/e(\mathcal{F}) \leq \Theta(n)/(n/2) = \Theta(1)$. The construction of the branching, the planar leveling, and the extracting π can all be done in linear time.

Case $|V_0| > n/2$: Take one outbound edge from each vertex in V_0. We obtain a Wheeler graph with $|V_0| > n/2$ edges. This gives us a solution with an approximation ratio of $|E^*|/|V_0| < \Theta(n)/(n/2) = \Theta(1)$.

In either case, we have an approximate solution with $\tilde{e}$ edges where $\tilde{e} \in \Theta(|E^*|)$. ◀

Next, we consider when $\sigma > 1$. Suppose $G^* = (V, E^*)$ is the optimal solution for G. Then $E^* = E_1^* \cup E_2^* \dots E_\sigma^*$ where $E_k^* = \{(u, v, k) \in E^*\}$. Let $G_k = (V, E_k)$ where $E_k = \{(u, v, k) \in E\}$ and let $G'_k = (V, E'_k)$ be the optimal solution for G_k. Then, since $|E_k^*| \leq |E'_k|$ we have

$$|E^*| = \sum_{k=1}^{\sigma} |E_k^*| \leq \sigma \cdot \max_k |E_k^*| \leq \sigma \cdot \max_k |E'_k|.$$

Applying the result for $\sigma = 1$ (Lemma 20), we can approximate $\max_k |E'_k|$ with a solution having $\tilde{e} = \alpha \cdot \max_k |E'_k|$ edges for some constant $\alpha \leq 1$. Therefore,

$$\frac{\alpha}{\sigma}|E^*| \leq \alpha \max_k |E'_k| = \tilde{e} \leq \max_k |E'_k| \leq |E^*|.$$

So the solution provides $\Omega(1/\sigma)$-approximation for G as well.

▶ **Theorem 21.** *There exists a linear time $\Omega(1/\sigma)$-approximation algorithm for WS.*

As a final result, we note that the algorithm presented in Section 3 also provides us with an exponential time solution to the two optimization problems defined in Section 4. The solution is to iterate over all possible subsets of edges in E, take the corresponding induced subgraph, and apply Algorithm 1 to identify if the induced subgraph is isomorphic to a Wheeler graph. For both the WGV and WS problems the optimal solution is the encoding with the fewest edges removed. The resulting time complexity is the same as in Theorem 10 with the addition of one e term in the exponent. We have shown the following:

▶ **Theorem 22.** *The WGV problem and WS problem for an input $G = (V, E)$ with $n = |V|$, $e = |E|$ and σ is the size of the edge label alphabet can be solved in time $2^{e \log \sigma + O(n+e)}$.*

References

1. Alfred V. Aho and Margaret J. Corasick. Efficient String Matching: An Aid to Bibliographic Search. *Commun. ACM*, 18(6):333–340, 1975. doi:10.1145/360825.360855.
2. Jarno Alanko, Travis Gagie, Gonzalo Navarro, and Louisa Seelbach Benkner. Tunneling on Wheeler Graphs. *CoRR*, abs/1811.02457, 2018. arXiv:1811.02457.
3. Jarno Alanko, Alberto Policriti, and Nicola Prezza. On Prefix-Sorting Finite Automata, 2019. arXiv:1902.01088.
4. László Babai and Eugene M. Luks. Canonical Labeling of Graphs. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 171–183, 1983. doi:10.1145/800061.808746.
5. Djamal Belazzougui. Succinct Dictionary Matching with No Slowdown. In *Combinatorial Pattern Matching, 21st Annual Symposium, CPM 2010, New York, NY, USA, June 21-23, 2010. Proceedings*, pages 88–100, 2010. doi:10.1007/978-3-642-13509-5_9.
6. Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn Graphs. In *Algorithms in Bioinformatics - 12th International Workshop, WABI 2012, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 225–235, 2012. doi:10.1007/978-3-642-33122-0_18.
7. Michael Burrows and David J Wheeler. A block-sorting lossless data compression algorithm, 1994.

8 Jianer Chen, Yang Liu, Songjian Lu, Barry O'Sullivan, and Igor Razgon. A fixed-parameter algorithm for the directed feedback vertex set problem. *J. ACM*, 55(5):21:1–21:19, 2008. doi:10.1145/1411509.1411511.

9 Francisco Claude, Gonzalo Navarro, and Alberto Ordóñez Pereira. The wavelet matrix: An efficient wavelet tree for large alphabets. *Inf. Syst.*, 47:15–32, 2015. doi:10.1016/j.is.2014.06.002.

10 Nicolaas Govert De Bruijn. A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen*, 49(49):758–764, 1946.

11 Vida Dujmovic and David R. Wood. On Linear Layouts of Graphs. *Discrete Mathematics & Theoretical Computer Science*, 6(2):339–358, 2004. URL: http://dmtcs.episciences.org/317.

12 Massimo Equi, Roberto Grossi, and Veli Mäkinen. On the Complexity of Exact Pattern Matching in Graphs: Binary Strings and Bounded Degree. *CoRR*, abs/1901.05264, 2019. arXiv:1901.05264.

13 Massimo Equi, Roberto Grossi, Alexandru I. Tomescu, and Veli Mäkinen. On the Complexity of Exact Pattern Matching in Graphs: Determinism and Zig-Zag Matching. *CoRR*, abs/1902.03560, 2019. arXiv:1902.03560.

14 Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan. Compressing and indexing labeled trees, with applications. *J. ACM*, 57(1):4:1–4:33, 2009. doi:10.1145/1613676.1613680.

15 Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.

16 Paolo Ferragina and Rossano Venturini. The compressed permuterm index. *ACM Trans. Algorithms*, 7(1):10:1–10:21, 2010. doi:10.1145/1868237.1868248.

17 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.

18 Arnab Ganguly, Rahul Shah, and Sharma V. Thankachan. pBWT: Achieving Succinct Data Structures for Parameterized Pattern Matching and Related Problems. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 397–407, 2017. doi:10.1137/1.9781611974782.25.

19 Daniel Gibney and Sharma V. Thankachan. On the Hardness and Inapproximability of Recognizing Wheeler Graphs, 2019. arXiv:1902.01960.

20 Venkatesan Guruswami, Rajsekar Manokaran, and Prasad Raghavendra. Beating the Random Ordering is Hard: Inapproximability of Maximum Acyclic Subgraph. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 573–582, 2008. doi:10.1109/FOCS.2008.51.

21 Lenwood S. Heath and Sriram V. Pemmaraju. Stack and Queue Layouts of Directed Acyclic Graphs: Part II. *SIAM J. Comput.*, 28(5):1588–1626, 1999. doi:10.1137/S0097539795291550.

22 Lenwood S. Heath, Sriram V. Pemmaraju, and Ann N. Trenk. Stack and Queue Layouts of Directed Acyclic Graphs: Part I. *SIAM J. Comput.*, 28(4):1510–1539, 1999. doi:10.1137/S0097539795280287.

23 Lenwood S. Heath and Arnold L. Rosenberg. Laying out Graphs Using Queues. *SIAM J. Comput.*, 21(5):927–958, 1992. doi:10.1137/0221055.

24 Wing-Kai Hon, Tsung-Han Ku, Rahul Shah, Sharma V. Thankachan, and Jeffrey Scott Vitter. Faster compressed dictionary matching. *Theor. Comput. Sci.*, 475:113–119, 2013. doi:10.1016/j.tcs.2012.10.050.

25 Viggo Kann. *On the approximability of NP-complete optimization problems.* PhD thesis, Royal Institute of Technology Stockholm, 1992.

26 Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. An Extension of the Burrows Wheeler Transform and Applications to Sequence Comparison and Data Compression. In *Combinatorial Pattern Matching, 16th Annual Symposium, CPM 2005, Jeju Island, Korea, June 19-22, 2005, Proceedings*, pages 178–189, 2005. doi:10.1007/11496656_16.

27 Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino. An extension of the Burrows-Wheeler Transform. *Theor. Comput. Sci.*, 387(3):298–312, 2007. doi:10.1016/j.tcs.2007.07.014.

28 Gary L. Miller. Graph Isomorphism, General Remarks. *J. Comput. Syst. Sci.*, 18(2):128–142, 1979. doi:10.1016/0022-0000(79)90043-6.

29 Adam M. Novak, Erik Garrison, and Benedict Paten. A graph extension of the positional Burrows-Wheeler transform and its applications. *Algorithms for Molecular Biology*, 12(1):18:1–18:12, 2017. doi:10.1186/s13015-017-0109-9.

30 Jaroslav Opatrny. Total Ordering Problem. *SIAM J. Comput.*, 8(1):111–114, 1979. doi:10.1137/0208008.

31 Jouni Sirén, Niko Välimäki, and Veli Mäkinen. Indexing graphs for path queries with applications in genome research. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 11(2):375–388, 2014.

32 D Younger. Minimum feedback arc sets for a directed graph. *IEEE Transactions on Circuit Theory*, 10(2):238–245, 1963.