Algorithmica (2022) 84:784–814
https://doi.org/10.1007/s00453-021-00917-5

On the Complexity of Recognizing Wheeler Graphs

Daniel Gibney[1] · **Sharma V. Thankachan**[2]

Received: 1 September 2020 / Accepted: 16 December 2021 / Published online: 10 January 2022

Abstract

In recent years, several *compressed indexes* based on variants of the Burrows–Wheeler transform 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 in J. ACM 52(4):552–581, https://doi.org/10.1145/1082036.1082039, 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. in Theor Comput Sci 698:67–78, https://doi.org/10.1016/j.tcs.2017.06.016, 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 space-efficient way. 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 show:

- 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-NFAs for $d \geq 5$. This is in contrast to recent results demonstrating the problem can be solved in polynomial time for d-NFAs where $d \leq 2$. We also show that the recognition problem can be solved in linear time for $\sigma = 1$ on graphs without self-loops;
- 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 identify the smallest set of edges that have to be removed from a graph to obtain a Wheeler graph. We show WGV is APX-

This research is supported in part by the U.S. National Science Foundation under the Grants CCF-1703489 and CCF-2112643. The first author has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 690941.

Extended author information available on the last page of the article

hard, even when G is a DAG, implying there exists a constant $C > 1$ for which there is no C-approximation algorithm (unless P = NP). Also, conditioned on the Unique Games Conjecture, for all $C > 1$, it is NP-hard to find a C-approximation, implying WGV is not in APX;
– 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 WGV). In contrast to WGV, we give an $O(\sigma)$-approximation algorithm for the WS problem, implying it 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 question of which properties determine this problem's difficulty.

Keywords Wheeler graphs · FM-index · Burrows–Wheeler transform

1 Introduction

Within the last two decades, there has been the development of Burrows–Wheeler Transform (BWT) [9] based indices for compressing a diverse collection of data structures. This list includes labeled trees [38], certain classes of graphs [17,36], and sets of multiple strings [19,33]. These new techniques have motivated the search for a set of general conditions under which a structure can be indexed by a BWT based index, which led to the recent introduction of Wheeler graphs by Gagie et al. [20] (also see [4]). A Wheeler graph is a directed graph that has edge labels and satisfies two simple properties related to the ordering of its vertices. Although not general enough to encompass all BWT-based structures (e.g., [21]), Gagie et al. demonstrated that Wheeler graphs offer a unified way of modeling several BWT based data structures such as representations of de Bruijn graphs [8,14], generalized compressed suffix arrays [38], multi-string BWTs [34], XBWTs [17], wavelet matrices [12], and certain types of finite automata [1,6,29]. 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 [18]. Since their introduction, Wheeler graphs have been the subject of significant study. This includes the study of the languages that are accepted by automata that are also Wheeler graphs [3], as well as the extension of a technique for compression known as tunneling to the BWTs of Wheeler graphs [4]. Additionally, Wheeler graphs give us insight into the more general problem of exact pattern matching on arbitrary labeled graphs, the computational complexity of which has been studied in several recent works [13,16,22]. It is clear that not all directed edge labeled graphs are Wheeler graphs, but despite being the subject of an increasing amount of research, it remained unknown how to recognize whether a given graph is a Wheeler graph. This made the

authors of [20] explicitly pose the question of *how to efficiently recognize whether a 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, one maintains an encoding of the subgraph using the framework presented in [20] 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 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 two *optimization versions* of this problem that seek subgraphs of the given graph which are Wheeler graphs. These problems turn out to be computationally difficult as well. As a positive result, we show that, for a constant sized alphabet, the problem of finding a maximum Wheeler subgraph admits a polynomial-time algorithm that outputs a solution with size within some constant factor of optimal. We also show that the problem of recognizing Wheeler graphs is related to that of identifying the queue number of a graph. This suggests a class of graphs where the problem becomes computationally tractable, a topic investigated in the last section of this work.

1.1 Wheeler Graphs

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

Definition 1 A Wheeler graph is a directed graph with edge labels where there exists an ordering $<_\pi$ on the vertices such that for any two edges (u, v, a) and (u', v', a') the following properties hold:

Property 1 $a < a'$ implies $v <_\pi v'$ and vertices with in-degree zero are placed first in the ordering;

Property 2 $a = a'$ and $u <_\pi u'$ implies $v \leq_\pi v'$.

We consider an ordering of the vertices of the graph a *proper ordering* if it satisfies the properties of the Wheeler graph definition. See Fig. 1 for an illustration.

The following list of additional properties of Wheeler graphs can be deduced from Definition 1.

Property 3 All edges inbound to a vertex v have the same edge label.

Property 4 In a proper ordering, all vertices with the same inbound edge label are ordered consecutively.

Property 5 A vertex can 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 6 For a vertex ordering π, two edges with the same label, (u, v, a) and (u', v', a), where $u <_\pi u'$ and $v' <_\pi v$ are called a monochromatic rainbow. No monochromatic rainbows can exist in a proper ordering (see Fig. 2).

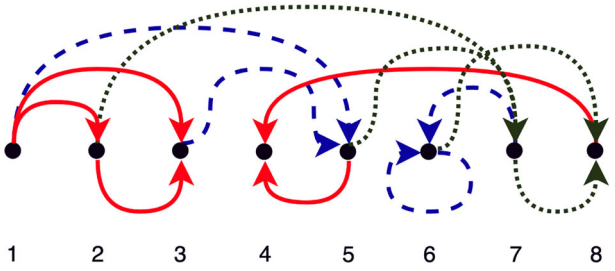

Fig. 1 A Wheeler graph with $\sigma = 3$. Ordering on edge labels: red (solid) $<$ blue (long-dash) $<$ green (short-dash) (Color figure online)

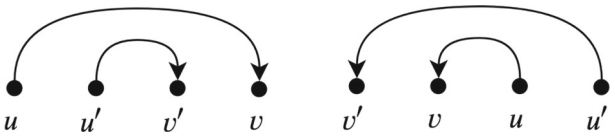

Fig. 2 In a proper ordering the above configurations cannot occur with edges that have the same label

1.2 Problem Definitions

The first question we wish to answer is: given a directed graph with edge labels, does there exist a proper ordering $<_\pi$ for its vertices? 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.

Next, we 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 Sect. 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 $\sigma = 2$. This 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 Sect. 3 we relate the notion of queue number to Wheeler graphs, allowing us to place a bound on the number of edges of any Wheeler graph.
– In Sect. 4 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 Wheeler graphs (of bounded size), 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 with the same time complexity for the optimization variants introduced in this work.
– In Sect. 5 we examine the optimization variants 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 polynomial-time algorithm whose solution size is $\Omega(1/\sigma)$ times the optimal value.
– In Sect. 6, by using PQ-trees and ideas similar to those used to detect whether a DAG is leveled-planar, we demonstrate a class of graphs where Wheeler graph recognition can be done in linear time.

2 NP-Completeness of Wheeler Graph Recognition

Theorem 1 *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 $\Theta(n)$ sources, or $\Theta(n)$ edges with the same label leaving a single vertex. In Sect. 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 [37]. Like our problem, it deals with finding a total ordering on a set of elements. The input to the Betweenness problem is a set of elements $T = \{t_1, \ldots, t_n\}$ and a collection of ordered triples $C \subseteq T^3$ called constraints. For a total ordering $<_\phi$ on T, we say a constraint $(t_1^j, t_2^j, t_3^j) \in C$ is satisfied if $t_1^j <_\phi t_2^j <_\phi t_3^j$ or $t_3^j <_\phi t_2^j <_\phi t_1^j$. The decision problem is to determine whether such a total ordering $<_\phi$ exists.

As an example, consider the input $T = \{1, 2, 3, 4, 5, 6\}$, and constraints $C = \{(5, 2, 3), (1, 5, 2), (4, 5, 6), (4, 6, 2)\}$. An ordering that satisfies the given constraints is 1, 4, 5, 6, 2, 3. An ordering that does not satisfy the given constraints is 1, 2, 3, 4, 5, 6 since it violates the constraints (5, 2, 3), (1, 5, 2), and (4, 6, 2).

2.2 Reduction from Betweenness to Wheeler Graph Recognition

Suppose we are given as input to the Betweenness problem the set $T = \{t_1, t_2, \ldots, t_n\}$, and constraints $C = \{(t_1^j, t_2^j, t_3^j) \mid 1 \leq j \leq k\}$. Construct a DAG G of size $O(nk)$ as follows:

1. Create a source vertex v_0.
2. For $1 \leq j \leq k$, $1 \leq i \leq n$, create vertex v_i^j.
3. For each constraint $(t_1^j, t_2^j, t_3^j) \in C$, create a vertex for each element, we call them w_1^j, w_2^j, and w_3^j respectively.
4. For $1 \leq i \leq n$, create the edge $(v_0, v_i^1, 1)$.
5. For $1 \leq j \leq k-1$, $1 \leq i \leq n$, create the edge $(v_i^j, v_i^{j+1}, 1)$.
6. For $1 \leq j \leq k$, $1 \leq i \leq n$, create the edge(s):
   - $(v_i^j, w_1^j, 2)$ and $(v_i^j, w_2^j, 2)$ if $t_i = t_1^j$;
   - $(v_i^j, w_2^j, 2)$ if $t_i = t_2^j$;
   - $(v_i^j, w_3^j, 2)$ and $(v_i^j, w_2^j, 2)$ if $t_i = t_3^j$.

Figure 3 provides an illustration. The intuition is that the vertices with inbound red (solid) edges labeled 1 represent the permutation of the elements in T repeated k times. The vertices with the inbound blue (dashed) edges labeled 2 represent the elements in the constraints. An ordering can be obtained from a vertex arrangement like that in Fig. 3 as follows: vertices with inbound red edges are ordered from bottom-to-top, followed by the vertices with inbound blue edges, ordered from bottom-to-top. If the ordering we obtain is a proper ordering, the arrangement in the figure will have no edges of the same color crossing. The relation that this ordering has with constraints being satisfied can be observed in Fig. 3. For example, with the top-most betweenness constraint gadget for constraint $(4, 5, 6)$, one can check that reversing the positions of the vertices for 4 and 6 with inbound blue edges will still avoid dashed blue edges crossing dashed blue edges as long as the same change happens for vertices with inbound red edges. However, any order where 5 is not between 4 and 6 will not satisfy this property. Hence, we can relate solutions to the Wheeler graph recognition problem to solutions to the Betweenness problem. We will formalize this argument next.

Lemma 1 first formalizes the way in which the vertices with inbound red edges represent a permutation being repeated k times.

Lemma 1 *Let $G' = (V', E')$ be the subgraph of G consisting of the vertices v_0 v_1^1, v_2^1,..., v_n^1,...,v_1^2, v_2^2,..., v_n^2,..., v_1^k, v_2^k,..., v_n^k and the edges $(v_0, v_i^1, 1)$ for $1 \leq i \leq n$ and $(v_i^j, v_i^{j+1}, 1)$ for $1 \leq i \leq n$, $1 \leq j \leq k-1$. Then the Wheeler graph properties are satisfied on G' iff the ordering π on V' is of form*

$$v_0, v_{\phi(1)}^1, v_{\phi(2)}^1, \ldots, v_{\phi(n)}^1, v_{\phi(1)}^2, v_{\phi(2)}^2, \ldots, v_{\phi(n)}^2, \ldots, v_{\phi(1)}^k, v_{\phi(2)}^k, \ldots, v_{\phi(n)}^k$$

where ϕ is a permutation of $\{1, 2, \ldots, n\}$.

Proof First consider when the vertices are ordered in the form stated in the lemma. Clearly, Property 1 is satisfied. Let $(v_i^j, v_i^{j+1}, 1)$ and $(v_{i'}^{j'}, v_{i'}^{j'+1}, 1)$ be arbitrary edges.

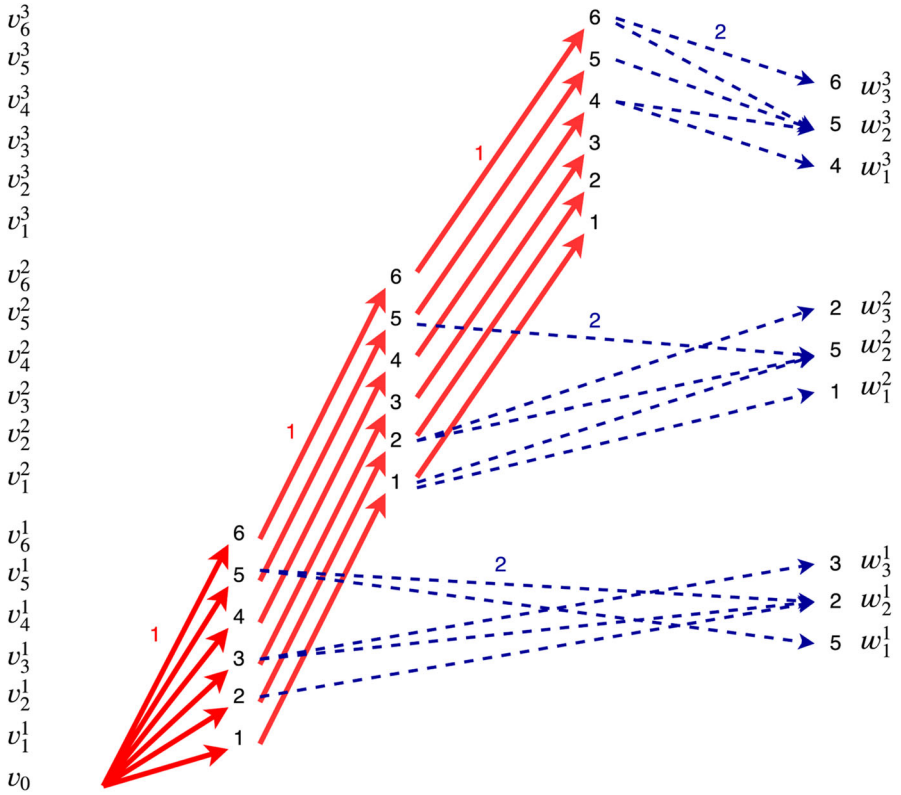

Fig. 3 An example of the reduction with input list 1, 2, 3, 4, 5, 6 and the constraints (5, 2, 3), (1, 5, 2), (4, 5, 6)

If $j < j'$, then $v_i^j <_\pi v_{i'}^{j'}$, and $v_i^{j+1} <_\pi v_{i'}^{j'+1}$. Thus the edges satisfy Property 2. The case where $j' < j$ is symmetric. Hence, we can assume $j' = j$. If we assume WLOG that $v_i^j <_\pi v_{i'}^j$, then $v_i^{j+1} <_\pi v_{i'}^{j+1}$ and Property 2 is satisfied.

Conversely, assume the Wheeler graph properties are satisfied by an ordering $<_{\pi'}$. Then, by Property 1, v_0 is ordered first. If all vertices in $\{v_i^1 \mid 1 \leq i \leq n\}$ do not appear prior to all $\{v_i^j \mid 1 \leq i \leq n\}$ where $j > 1$, then there exists some leftmost vertex v_i^j such that $v_i^j <_{\pi'} v_{i'}^1$ for some $1 \leq i, i' \leq n$. Then the edges $(v_0, v_{i'}^1, 1)$ and $(v_i^{j-1}, v_i^j, 1)$ contradict Property 2 since $v_0 <_{\pi'} v_i^{j-1}$ but $v_{i'}^1 >_{\pi'} v_i^j$. This argument can then be extended to show that the set $\{v_i^2 \mid 1 \leq i \leq n\}$ must immediately follow $\{v_i^1 \mid 1 \leq i \leq n\}$, and then repeated to prove the partial ordering

$$v_0 <_{\pi'} v_1^1, \ldots v_n^1 <_{\pi'} v_1^2, v_2^2, \ldots, v_n^2 <_{\pi'} \ldots <_{\pi'} v_1^k, v_2^k, \ldots, v_n^k$$

must hold.

Let ϕ denote the permutation applied to the vertices in $\{v_i^1 \mid 1 \leq i \leq n\}$ by the vertex ordering $<_{\pi'}$. If ϕ is not applied to the vertices in $\{v_i^2 \mid 1 \leq i \leq n\}$ by $<_{\pi'}$,

then there exists an i and an i' such that $v_i^1 <_{\pi'} v_{i'}^1$ and $v_{i'}^2 <_{\pi'} v_i^2$, causing the edges $(v_i^1, v_i^2, 1)$ and $(v_{i'}^1, v_{i'}^2, 1)$ to contradict Property 2. The same argument can then be repeated to show the same permutation ϕ applied to $\{v_i^2 \mid 1 \leq i \leq n\}$ must be applied to $\{v_i^3 \mid 1 \leq i \leq n\}$. Continuing this argument to $\{v_i^k \mid 1 \leq i \leq n\}$ shows that $<_{\pi'}$ orders the vertices in the desired form. □ □

Lemma 2 *An instance of the Betweenness problem has an ordering satisfying all of the constraints iff the graph G is a Wheeler graph.*

Proof We first assume that there exists a solution to the instance of the Betweenness problem. Let $<_\phi$ be the ordering on the elements in T such that all constraints are satisfied. We abuse notation and also use ϕ to denote the permutation where $t_{\phi(1)} <_\phi t_{\phi(2)} <_\phi \ldots <_\phi t_{\phi(n)}$. For constraint (t_1^j, t_2^j, t_3^j) let f_j be 0 if $t_1^j <_\phi t_2^j <_\phi t_3^j$ and 2 if $t_3^j <_\phi t_2^j <_\phi t_1^j$.

Order the vertices as follows:

$$v_0, v_{\phi(1)}^1, \ldots, v_{\phi(n)}^1, v_{\phi(1)}^2, \ldots, v_{\phi(n)}^2, \ldots, v_{\phi(1)}^k, \ldots, v_{\phi(n)}^k,$$
$$w_{1+f_1}^1, w_2^1, w_{3-f_1}^1, w_{1+f_2}^2, w_2^2, w_{3-f_2}^2, \ldots, w_{1+f_k}^k, w_2^k, w_{3-f_k}^k.$$

Let $<_\pi$ denote this vertex ordering. The first property of Wheeler graphs is satisfied since v_0 with in-degree zero is ordered first, all vertices with inbound edges having label 1 are next, followed by all vertices having inbound edges with label 2. Violations of Property 2 with edges having label 1 are avoided by Lemma 1. We show next that violations of Property 2 with edges having label 2 are avoided as well. Consider the constraint (t_1^j, t_2^j, t_3^j) where $t_i = t_1^j, t_h = t_2^j$, and $t_\ell = t_3^j$. Either $t_i <_\phi t_h <_\phi t_\ell$ and the vertex order has $v_i^j <_\pi v_h^j <_\pi v_\ell^j$ and $w_1^j <_\pi w_2^j <_\pi w_3^j$; or $t_\ell <_\phi t_h <_\phi t_i$ and the vertex order has $v_\ell^j <_\pi v_h^j <_\pi v_i^j$ and $w_3^j <_\pi w_2^j <_\pi w_1^j$. In both cases, for every pair of edges in $(v_i^j, w_1^j, 2)$, $(v_i^j, w_2^j, 2)$, $(v_h^j, w_2^j, 2)$, $(v_\ell^j, w_3^j, 2)$, and $(v_\ell^j, w_2^j, 2)$ Property 2 is satisfied.

In the other direction, we assume that the graph constructed in the reduction is a Wheeler graph. First, by Property 1, v_0 must be ordered before all vertices in $\{v_i^j \mid 1 \leq j \leq k, 1 \leq i \leq n\}$, followed by all vertices $\{w_i^j \mid 1 \leq j \leq k, 1 \leq i \leq 3\}$. Again by Lemma 1, to avoid violations of Property 2 for edges with label 1 there must be some permutation ϕ such that v_0 and the vertices in the first set are ordered

$$v_0, v_{\phi(1)}^1, \ldots, v_{\phi(n)}^1, v_{\phi(1)}^2, \ldots, v_{\phi(n)}^2, \ldots, v_{\phi(1)}^k, \ldots, v_{\phi(n)}^k.$$

Since there are no violations of Property 2 for blue edges, for all $1 \leq j \leq k-1$ we have that w_1^j, w_2^j, and w_3^j are all ordered before w_1^{j+1}, w_2^{j+1}, w_3^{j+1}. Furthermore, for a particular j where $t_i = t_1^j$, $t_h = t_2^j$, and $t_\ell = t_3^j$, it must be that the partial order induced on v_i^j, v_h^j, and v_ℓ^j is also induced on w_1^j, w_2^j, and w_3^j. Additionally, for there to be no violations of Property 2, v_h^j (and hence w_2^j) must lie between v_i^j and v_ℓ^j (w_1^j, w_3^j resp.). This implies that the ordering ϕ satisfies the constraint (t_1^j, t_2^j, t_3^j) in C when

ϕ is applied to T. Since this is true for all constraints in C, ϕ provides a solution to the Betweenness problem. □

Theorem 1 then follows directly from Lemma 2.

2.3 NP-Completeness of Wheeler Graph Recognition on *d*-NFAs

Now we restrict the number of edges with the same label that can leave a single vertex. We adopt the terminology used by Alanko et al., and consider the problem of recognizing whether a d-NFA is also a Wheeler graph [2]. A d-NFA is defined as follows:

Definition 2 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.

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 et al., who showed that it can be recognized in polynomial time whether a 2-NFA is a Wheeler graph [2]. Their result, coupled with the observation that the reduction in Sect. 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 2.

Theorem 2 *The Wheeler Graph Recognition Problem is NP-complete for d-NFAs, $d \geq 5$.*

The strategy of the proof will be to reduce the NP-complete problem 4-NAESAT to Wheeler Graph Recognition. In 4-NAESAT each clause is of length 4, and an instance 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 to obtain a 3-NAESAT instance. The reduction is folklore knowledge, but we include it for completeness and to highlight a desired property.

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

Proof Convert the 4-NAESAT instance ψ to a 3-NAESAT instance ψ' by converting each clause (a_k, b_k, c_k, d_k) into the clauses (a_k, w_k, b_k) and $(c_k, \overline{w_k}, d_k)$ where w_k is a new variable. One can quickly check that it is always possible to find a satisfying not-all-equal assignment for both clauses, unless $a_k = b_k = c_k = d_k$. We also note that the variable used in the middle of the clauses, w_k, is used only twice in all of ψ'. □

For convenience, we define the set of 3-NAESAT instances where any variable occurring in the middle of a clause occurs at most twice in the whole Boolean formula as 3-NAESAT*. We next describe the construction of a one source DAG from an instance of 3-NAESAT*.

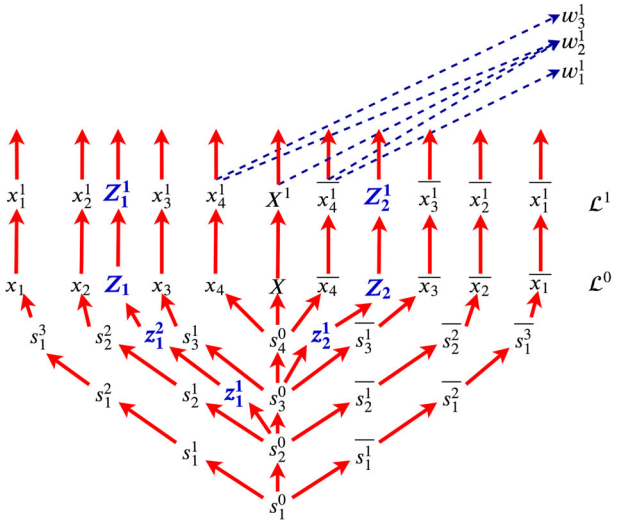

Fig. 4 Vertex Z_1 and Z_2 could be for clauses (x_1, x_2, x_3), $(x_2, \overline{x_3}, x_4)$. Each 'betweenness' constraint adds a layer. Layer for constraint $(x_4, X, \overline{x_4})$ is shown

Suppose we are given an instance ψ 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 ψ. 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 the literals, $x_1, \ldots, x_n, X, \overline{x_1}, \ldots, \overline{x_n}$ (the role of X will become clear). We will use the literals to refer to the vertices. Next, we add a structure to constrain their possible positions (see Fig. 4 for an example).

To create this structure, do as follows:

- Create the vertices $s_1^0, s_2^0, \ldots, s_n^0$.
- Create the red (solid) edges $(s_1^0, s_2^0, 1), (s_2^0, s_3^0, 1), \ldots (s_n^0, X, 1)$.
- For $1 \leq i \leq n-1$, $1 \leq j \leq n-i$, create the vertices s_i^j and $\overline{s_i^j}$.
- For $1 \leq i \leq n-1$, $1 \leq j \leq n-i$, create the red (solid) edges $(s_i^{j-1}, s_i^j, 1)$ and $(\overline{s_i^{j-1}}, \overline{s_i^j}, 1)$.
- For $1 \leq i \leq n$, create the red (solid) edges $(s_i^{n-i}, x_i, 1)$ and $(\overline{s_i^{n-i}}, \overline{x_i}, 1)$.

For clause k, denoted as (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 to add a set of *betweenness* type constraints on any proper ordering given of the vertices $\mathcal{L}^0 = \{x_1, \ldots, x_n, X, \overline{x_n}, \ldots, \overline{x_1}, Z_1, Z_2, \ldots\}$. We first add a layer of new vertices $\mathcal{L}^1 = \{x_1^1, \ldots, x_n^1, X^1, \overline{x_n^1}, \ldots, \overline{x_1^1}, Z_1^1, Z_2^1, \ldots\}$ and red (solid) edges labeled 1 from each vertex in layer $\mathcal{L}^0$ to its corresponding vertex in $\mathcal{L}^1$. We will utilize the same gadget that was used in Sect. 2.2. To add a betweenness constraint

Table 1 Possible relative orderings of a_k, b_k, c_k, Z_k, X subject to constraints (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	$a_k \dots Z_k, b_k \dots X \dots c_k$	$a_k \dots Z_k, b_k \dots X \dots c_k$

(y_1, y_2, y_3) to arbitrary vertices y_1, y_2, y_3 in $\mathcal{L}^1$, we add the vertices w_1^1, w_2^1, and w_3^1 and the blue (dashed) edges $(y_1, w_1^1, 2)$, $(y_1, w_2^1, 2)$, $(y_2, w_2^1, 2)$, $(y_3, w_3^1, 2)$, and $(y_3, w_2^1, 2)$. Additional betweenness constraints can be similarly enforced by adding a new layer $\mathcal{L}^2$ on top of $\mathcal{L}^1$ with a new gadget. Using this technique of adding a new layer for every new betweenness constraint, we next add the constraints (a_k, Z_k, b_k) and (c_k, X, Z_k) for every clause (a_k, b_k, c_k) and the constraints $(x_i, X, \overline{x_i})$ for $1 \leq i \leq n$.

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

Lemma 4 *The leveled graph G constructed as above from an instance ψ of 3-NAESAT* is a Wheeler graph iff ψ is satisfiable.*

Proof Given a truth assignment that satisfies the 3-NAESAT* instance ψ, put the vertices in $\mathcal{L}^0$ whose literals are assigned the value T (true) on the left side of X (as in Fig. 4), and the vertices whose literals are assigned F (false) on the right side of X. For example, if $x_1 = T$ and $x_2 = F$, the two left-most vertices on level $\mathcal{L}^0$ would be x_1 followed by $\overline{x_2}$. For all of the possible not-all-equal arrangements of the literals for a_k, b_k, and c_k, relative to X, we will always be able to find a place in the ordering for Z_k that respects the betweenness constraints. For instance, if the variable for b_k is x_h, this is possible because Z_k is able to 'freely pivot' around the vertex s_h in the spine of the menorah structure and find the betweenness-constraint-respecting position immediately to the left or right of x_h or $\overline{x_h}$. This can be confirmed by examining all possible cases, as is shown in Table 1. For clause (a_k, b_k, c_k), Table 1 shows all possible not-all-equal truth assignments, and the corresponding relative orderings of $\mathcal{L}^0$ we can apply to the vertices that satisfy the Wheeler graph properties.

In the other direction, assume G is a Wheeler graph so we have a proper ordering on the vertices of G. The proper ordering of the menorah structure is fixed with the exception of z_i^j vertices, the ordering duplicated across layers $\mathcal{L}^0, \mathcal{L}^1, \dots$. We will show that in a proper ordering of the vertices, the ordering given to $\mathcal{L}^0$ must have every clause in ψ getting a not-all-equal assignment when we apply the following map: vertices for a non-negated literal on the left of X in $\mathcal{L}^0$ map back to a T assignment for

Table 2 Orderings implied by all-equal assignment

Impossible 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	$a_k \ldots b_k \ldots c_k \ldots X$	$c_k \ldots b_k \ldots a_k \ldots X$
FFF	$X \ldots c_k \ldots b_k \ldots a_k$	$X \ldots a_k \ldots b_k \ldots c_k$

These make it impossible to satisfy all constraints

that variable, and vertices for a non-negated literal to the right of X in $\mathcal{L}^0$ map back to an F assignment for that variable.

Suppose to the contrary that this mapping did not provide a valid not-all-equal assignment. Then $\mathcal{L}^0$ was given an ordering where the vertices for a_k, b_k, and c_k are all either on the left or the right side of X. The possible arrangements for this are presented in Table 2. In contrast to the cases listed in Table 1, for all cases listed in Table 2, placing Z_k between a_k and b_k violates the constraint (c_k, X, Z_k), which by our reduction implies it violates Wheeler graph Property 2 as well. This contradicts the assumption that we have a proper ordering on the vertices. We conclude that a proper ordering of the vertices of G must map back to a truth assignment that gives each clause in ψ a not-all-equal assignment. □

This leaves open the complexity of the recognition problem for 3-NFAs and 4-NFAs.

3 Wheeler Graphs and Queue Number

3.1 Queue Number

The concept of queue number and queue layout were introduced by Heath and Rosenberg, originally for undirected graphs in [28], and later expanded to DAGs in [27]. We describe it first for DAGs. Let the vertices of a DAG G be given a total ordering that is also a topological ordering. We will say the edges of G can be processed using a set of queues if we can iterate through the vertices in the given ordering and every time the tail of an edge is encountered that edge is enqueued in one of the queues, and when the head of that same edge is encountered, that edge is then dequeued from its assigned queue. If we assign every edge a color according to its queue, this is equivalent to the ordering not creating any monochromatic rainbows like those in the left of Fig. 2. Over all possible orderings of the vertices, there is some ordering which requires the minimum number of queues to perform this processing. That minimum number of queues is called the queue number of G. For undirected graphs, when processing the edges we make an edge enqueued the first time either of the vertices it is incident to is encountered, and dequeued when the other vertex it is incident to is encountered. Again, the minimum number of queues required to do this over all possible vertex orderings is the queue number of the graph. Figure 5 provides an illustration of the edges of an undirected graph being processed in this way. The problem of detecting whether a graph is a one-queue DAG was shown to be solvable in linear time by Heath

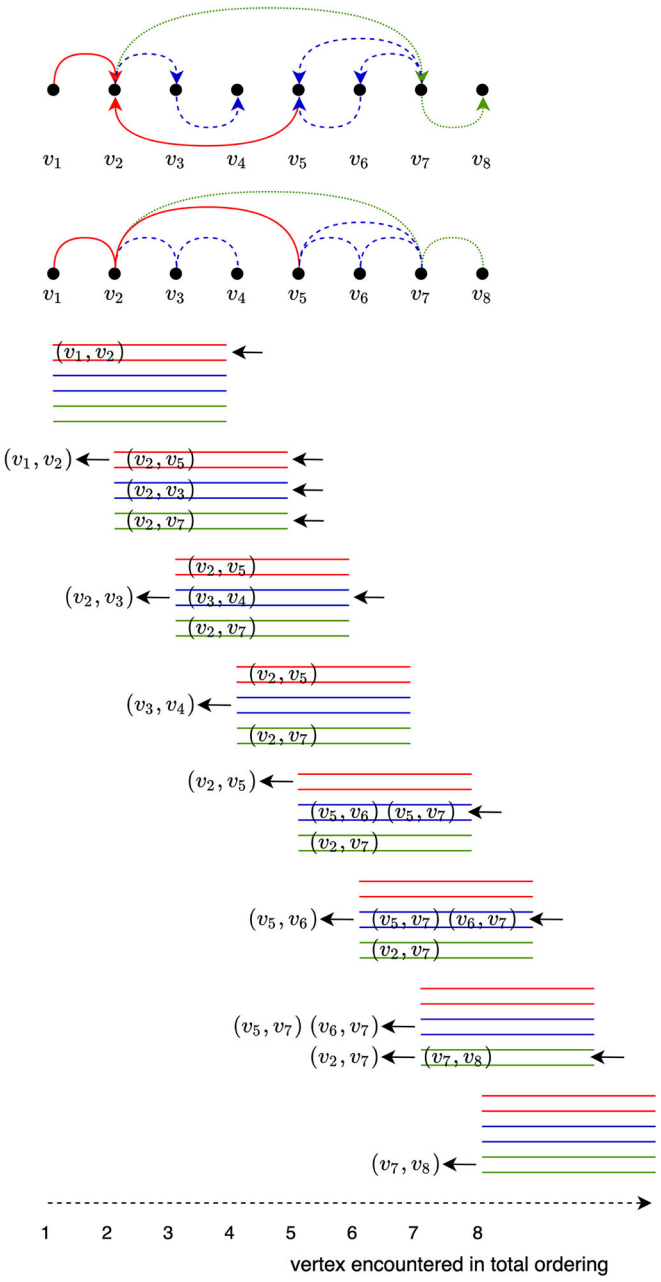

Fig. 5 (Top) A Wheeler graph with an alphabet of size three. (Middle) The same graph with edge orientations removed. (Bottom) Processing of edges with three queues. A Wheeler graph with an alphabet of size three is guaranteed to have queue number at most three. Note the undirected graph actually has queue number less than three, which can be seen by coloring the two green (dotted) edges red (solid) (Color figure online)

and Pemmaraju [26–28]. Using a few additional steps, we can extend their techniques to a specific subset of Wheeler graphs.

Theorem 3 *The Wheeler graph recognition problem can be solved in linear time for an edge alphabet of size $\sigma = 1$ on graphs without self-loops.*

Proof When $\sigma = 1$ and the graph has no self-loops, any proper Wheeler ordering is a topological ordering. The problem of finding a one-queue ordering and a proper Wheeler ordering are almost equivalent. The only difference is that for a proper Wheeler ordering all of the vertices with in-degree zero must be placed first. To overcome this, we first let $V_0 \subseteq V$ represent all vertices in V with in-degree zero. Create a new vertex u with in-degree zero and add an edge from u to each vertex in V_0. Since a valid one-queue ordering is a topological ordering, v_0 must be first in the one-queue ordering. Moreover, any vertices in the set $V - V_0$ must be in the one-queue ordering after the last position given to a vertex in V_0, otherwise a rainbow is created. Thus, the above modification ensures that one-queue orderings on V place the vertices in V_0 before any vertices in $V - V_0$, so these orders are also proper Wheeler orderings. □

We can use additional results on the queue number of undirected simple graphs to obtain an upper bound on the number of edges that can be in a Wheeler graph. The queue number of the underlying undirected graph of a Wheeler graph with alphabet size σ, is at most σ. This is since, when edge orientations are removed, the absence of monochromatic rainbows enforced by the properties of Wheeler graphs is exactly what is required for processing edges having the same color using a single queue. See Fig. 5. As a result, we get the bound presented in Theorem 4.

Theorem 4 *The number of edges in a Wheeler graph is at most* $(2\sigma+1)n-\sigma(2\sigma+1)$.

Proof The number of edges in a undirected graph with queue number at most q is bound by $2qn - q(2q+1)$ [15]. By removing self edges and the edge orientations, the Wheeler graph becomes an undirected graph with queue number at most σ. Finally, there are at most n additional edges added due to self-loops. □

4 An Exponential Time Algorithm

We can apply the encoding introduced by Gagie et al. [20] 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 having a limited search space of Wheeler graphs, and that graph isomorphism can be checked in sub-exponential time. We have the following theorem.

Theorem 5 *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 5, we need to describe the encoding of a Wheeler graph given in [20]. A Wheeler graph can be completely

specified by three bit vectors; two bit vectors O and I 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 O is of the form $0^{\ell_1}10^{\ell_2}1 \ldots 0^{\ell_n}1$ and I 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 character symbol is 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. All of these arrays are equipped with additional rank and select structures to allow for efficient traversal as is done in the FM-index [18]. In [20] an additional array that stores character counts is added. For our purposes however, the arrays O, I, and L are adequate.

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 'YES'
    end if
  end if
end for
return 'NO'
```

Psuedocode for our algorithm is given in 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 $|S| \leq 2^{O(e+n)+e\log\sigma}$. The Wheeler graph corresponding to an 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 a be the edge label and j be the index of the label a in L that 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 $i = rank_1(O, select_0(O, j)) + 1$. 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, a) with label a in G, then $f(u)$ and $f(v)$ are also adjacent via an edge $(f(u), f(v), a)$ with label a in G'. Using ideas similar to those presented by Miller in [35], this problem can be reduced in polynomial time to checking whether two undirected graphs are isomorphic.

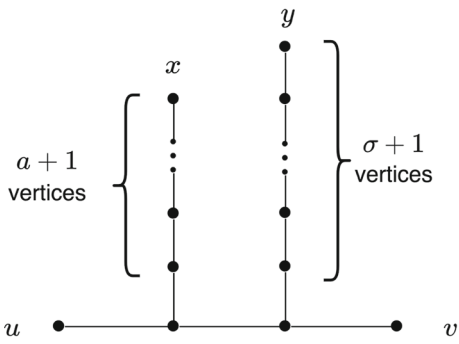

Fig. 6 An a-gadget replacing directed labeled edge (u, v, a)

Lemma 5 *The problem of checking whether the directed 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 Define the transformation α from the directed edge labeled graph G to an undirected graph $\alpha(G)$ as follows: For every directed edge (u, v, a) replace it with the a-gadget in Fig. 6. We will show that there exists an edge-label-preserving isomorphism from $V(G)$ to $V(G')$ if and only if there exists a (standard) isomorphism between $\alpha(G)$ and $\alpha(G')$.

We first assume that there exists an edge-label-preserving isomorphism f from $V(G)$ to $V(G')$. This implies that when α is applied to G' the same gadget is used to replace the edge $(f(u), f(v), a)$ as the gadget used to replace the edge (u, v, a) in G. Therefore, the function f can be naturally extended to an isomorphism $\tilde{f}$ on the vertices of $\alpha(G)$ providing an isomorphism between $\alpha(G)$ and $\alpha(G')$.

Now, consider the case where $\tilde{f}$ is an isomorphism between $\alpha(G)$ and $\alpha(G')$. We wish to show that G and G' must be related by an edge-label-preserving isomorphism. Let $n' = |V(\alpha(G))|$. We define a n'-tuple of numbers for each vertex $v \in V(\alpha(G))$ as $\beta(v) = (c_1, c_2, \ldots, c_{n'})$ where c_i is the number of vertices with graph distance i from v, i.e., minimum path length measured in edges. In Fig. 6, $\beta(x) = (1, 1, \ldots, 1, 2, \ldots)$ where the leading 1's are repeated $a + 1$ times. Also, $\beta(y) = (1, 1, \ldots, 1, 2, \ldots)$ where the leading 1's are repeated $\sigma + 1$ times. For example, when $\sigma = 1$, we have $\beta(y) = (1, 1, 2, \ldots)$. Notice first that $\beta(v) = \beta(\tilde{f}(v))$, i.e., $\beta(v)$ is invariant under $\tilde{f}$. Now observe that for any vertex $u \in V(G)$ of degree d we have that $\beta(\alpha(u)) = (d, 2d, \ldots)$ (where $\alpha(u)$ denotes the vertex u is mapped onto when α is applied to G). It follows that any vertex which is an x vertex of an a-gadget is mapped by $\tilde{f}$ onto an x vertex of an a-gadget. Similarly, any vertex which is a y vertex of an a-gadget is mapped by $\tilde{f}$ onto a y vertex of an a-gadget. Hence, a-gadgets are mapped by $\tilde{f}$ onto a-gadgets. This also implies that vertices in $V(\alpha(G))$ originally in G are mapped by $\tilde{f}$ onto vertices in $V(\alpha(G'))$ that were originally in $V(G')$. If we restrict $\tilde{f}$ to only the vertices originally in $V(G)$, then this restriction provides us with an edge-label-preserving isomorphism between G and G'. The reduction clearly takes polynomial time. □

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 $\alpha(G)$ [5]. The total time complexity of Algorithm 1 is the number of bit strings tested, multiplied by the time it takes to validate whether the bit string encodes a Wheeler graph G' and decode it, convert G and G' to undirected graphs $\alpha(G)$ and $\alpha(G')$, and 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.

5 Optimization Variants of Wheeler Graph Recognition

5.1 The Wheeler Graph Violation Problem is APX-Hard

In this section we show that obtaining an approximate solution to the WGV problem whose objective value comes within some constant factor of the optimal solution's objective value 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 that need to be removed in order to make the directed graph a DAG. A slightly different formulation proves more useful for us. Letting $F_\pi = \{(v_i, v_j) \in E \mid v_j <_\pi v_i\}$, we have the following:

Lemma 6 (Younger [39]) *Determining a minimum feedback arc set for $G = (V, E)$ is equivalent to finding an ordering $<_\pi$ on V for which $|F_\pi|$ is minimized.*

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

Definition 3 (*Minimum Feedback Arc Set (FAS)*) The input is a set $T = \{t_1, t_2, \ldots, t_n\}$ of n numbers and a set of k inequalities of the form $t_i < t_j$. This task is to compute an ordering $<_\pi$ on T such that the number of inequalities violated is 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 [10]. 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 linear time.

On the other hand, it has been shown that FAS is APX-hard, meaning that every problem in APX is reducible to it [31]. 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 6 *The WGV problem is APX-hard.*

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 7 (Guruswami et al. [24]) *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 7, proves the other main result of this section:

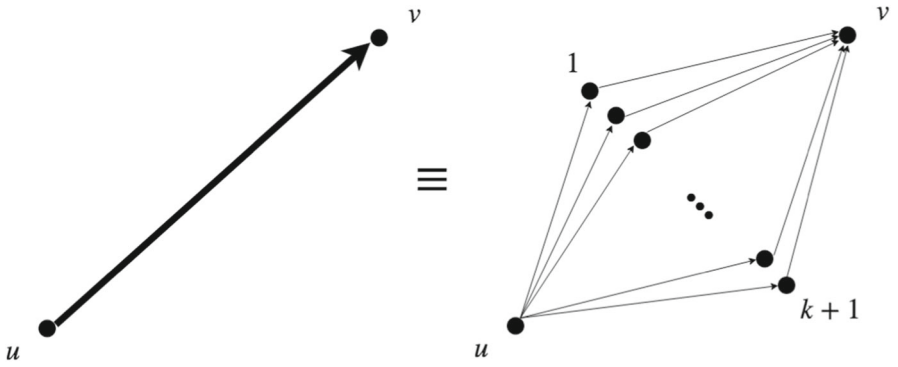

Fig. 7 A heavy(bold) edge in Fig. 8 is actually $k+1$ subdivided edges

Theorem 7 *Conditioned on the Unique Games Conjecture, for every constant $C \geq 1$, it is NP-hard to find a C-approximation to WGV, implying WGV is not in APX.*

5.2 The Reduction from 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 a as $k+1$ subdivided edges between u and v each with label a. That is, a heavy edge between u and v with label a consists of the edges (u, w_i, a) and (w_i, v, a) for $1 \leq i \leq k+1$. See Fig. 7 for an illustration. Heavy edges are useful to us, as a violation of Property 2 involving two heavy edges will require $k+1$ edges to be removed if the ordering is to be maintained. We use the following steps to create a graph (which is a DAG):

- Create a vertex v_0.
- For $1 \leq j \leq 2k-1$, $1 \leq i \leq n+1$, create vertex v_i^j.
- 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.
- For $1 \leq i \leq n+1$, create heavy edges $(v_0, v_i^1, 1)$.
- For $1 \leq i \leq n+1$, $1 \leq j \leq 2k-2$, create heavy edges $(v_i^j, v_i^{j+1}, 1)$.
- Create heavy edge $(v_0, w_1^1, 2)$.
- For $1 \leq j \leq 2k-1$,
  - if j is odd, create the heavy edge $(v_{n+1}^j, w_2^{\frac{j+1}{2}}, 2)$,
  - if j is even, create the heavy edge $(v_{n+1}^j, w_1^{\frac{j}{2}+1}, 2)$.
- For $1 \leq j \leq 2k-1$, $1 \leq i \leq n$,
  - if j is odd, create the regular (not heavy) edge:
    - $(v_i^j, w_1^{\frac{j+1}{2}}, 2)$ if $t_i = t_1^{\frac{j+1}{2}}$,
    - $(v_i^j, w_2^{\frac{j+1}{2}}, 2)$ if $t_i = t_2^{\frac{j+1}{2}}$.

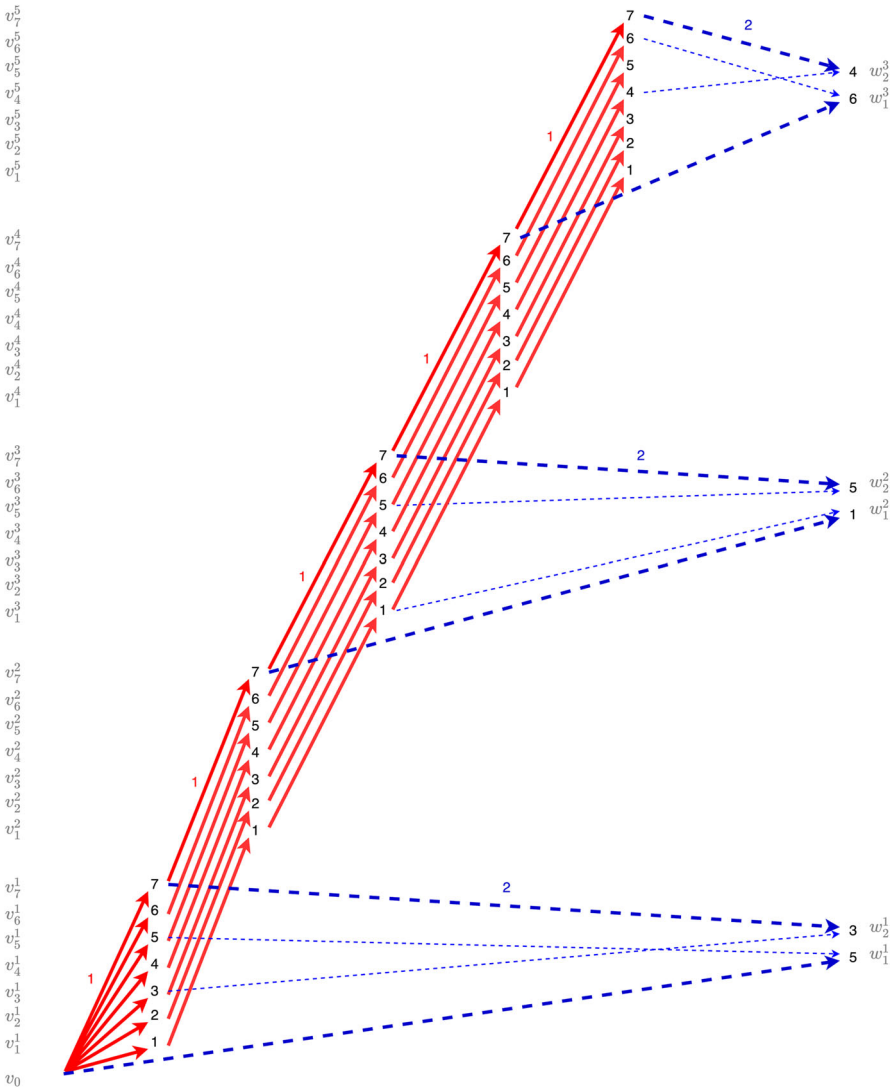

Fig. 8 Reduction from FAS to WGV where $T = \{1, 2, 3, 4, 5, 6\}$ and the inequalities are $5 < 3$, $1 < 5$, and $6 < 4$

An example of the reduction is given in Fig. 8. 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 edges labeled 2 represent the elements in each inequality. We will show that this is an approximation preserving reduction.

Let E' be an optimal solution to WGV and $G' = (V, E \backslash E')$. Let $<_\pi$ represent a proper ordering on the vertices of G'. Lemma 8 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 $1 \leq j \leq 2k-1$), the ordering for the vertices in Fig. 8 is fixed.

Lemma 8 *Let ϕ represent a permutation of the set $\{1, 2, \ldots, n+1\}$. Any ordering $<_\pi$ which provides a solution to the constructed instance of WGV with at most k edges violating the Wheeler graph properties orders the vertices in the form*

$$v_0, v_{\phi(1)}^1, v_{\phi(2)}^1, \ldots v_{\phi(n+1)}^1, \ldots v_{\phi(1)}^{2k-1}, v_{\phi(2)}^{2k-1}, \ldots v_{\phi(n+1)}^{2k-1}, w_1^1, w_2^1, w_1^2, w_2^2, \ldots w_1^k, w_2^k.$$

Proof The ordering given from the statement of the lemma would require at most k edges to be removed to satisfy the Wheeler graph properties, so we know that $|E'| \leq k$. If any of the w vertices is placed before a v vertex, that would cause at least $k+1$ edges to need to be removed, and is hence sub-optimal. Similarly, v_0 must be placed first in the ordering.

Again by Lemma 1, if the v vertices are not ordered in the form

$$v_0, v_{\phi(1)}^1, v_{\phi(2)}^1, \ldots, v_{\phi(n+1)}^1, \ldots,$$
$$v_{\phi(1)}^{2k-1}, v_{\phi(2)}^{2k-1}, \ldots, v_{\phi(n+1)}^{2k-1},$$

this will cause a violation of Property 2. However, since these are now heavy edges, this will require at least $k+1$ edges to be removed in order to satisfy the Wheeler graph properties.

For the w vertices, the vertex w_1^1 must be ordered before w_2^1, else the heavy edges $(v_0, w_1^1, 2)$ and $(v_{n+1}^1, w_2^1, 2)$ would cause the removal of at least $k+1$ edges to be necessary. The vertex w_2^1 must be ordered before w_1^2, else the heavy edges $(v_{n+1}^1, w_2^1, 2)$ and $(v_{n+1}^2, w_1^2, 2)$ would cause the removal of at least $k+1$ edges to be necessary. This argument can be repeated until w_1^k and w_2^k, proving that the ordering has the desired form. □

Let $f(x)$ refer to the reduction described above applied to an instance x of FAS. Hence, $f(x)$ is an instance of WGV. We also refer to the optimal solution to either of these problems as $\text{OPT}(\cdot)$, and the objective value (or cost) of a solution as $val(\cdot)$. For FAS, $val(\cdot)$ is the number of violated inequalities. For WGV, $val(\cdot)$ is the minimum number of edges that need to be removed to obtain a Wheeler graph when the ordering given by the solution is applied.

Lemma 9 *Given an instance x of FAS, a solution y' to the instance $f(x)$ of WGV that has $val(y') = \ell \leq k$ yields a solution to x with ℓ violated inequalities.*

Proof By Lemma 8, we can assume the ordering of the vertices given by y' is of the form stated in Lemma 8, and is completely determined by the ordering given to $v_1^1, v_2^1, \ldots, v_n^1, v_{n+1}^1$. Ignore the vertex v_{n+1}^1 and apply the remaining ordering to T. Any edge that has to be removed is one of the two edges in a pair $(v_{i_1}^j, w_1^{\frac{j+1}{2}}, 2)$ and $(v_{i_2}^j, w_2^{\frac{j+1}{2}}, 2)$, where $t_{i_1} = t_1^j$ and $t_{i_2} = t_2^j$ for a constraint (t_1^j, t_2^j). Since $w_1^j <_\pi w_2^j$, this removal is only necessary if $v_{i_2}^j <_\pi v_{i_1}^j$. Hence, if one these edges must be

removed, it implies that in our solution to x, $t_{i_2} < t_{i_1}$, and our solution does not satisfy the inequality $t_1^j < t_2^j$. On the other hand, if Property 2 holds for the edges $(v_{i_1}^j, w_1^{\frac{j+1}{2}}, 2)$ and $(v_{i_2}^j, w_2^{\frac{j+1}{2}}, 2)$, then in our solution for x, $t_{i_1} < t_{i_2}$, and the inequality $t_1^j < t_2^j$ is satisfied. □

The next lemma is an immediate consequence of Lemma 9.

Lemma 10 *Given an instance x of FAS, a C-approximation to the solution OPT($f(x)$) yields a C-approximation to the solution OPT(x).*

Theorem 6 follows from Lemma 10 and Theorem 7 follows from Lemmas 7 and 10.

5.3 The Wheeler Subgraph Problem is in APX

The dual problem to WGV is the problem of finding the largest subgraph of $G = (V, E)$ which is a Wheeler graph. This problem (defined in Sect. 1.2) is called the 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 arborescences, where an arborescence is a directed, rooted tree with all maximal paths starting at the root.

Lemma 11 *There exists a linear time $\Theta(1)$-approximation algorithm for WS when the alphabet size is $\sigma = 1$.*

Proof We consider G to be at least weakly connected. In the case where G is not weakly connected, the approximate solutions can be combined to obtain a solution for G with a $\Theta(1)$-approximation factor. First, remove all singleton vertices (vertices with in-degree and out-degree zero), and let n' be the number of remaining vertices. By doing this we know that the number of edges in the remaining graph is at least $n' - 1$. Next, remove any edges that are self-loops. Let G' denote the resulting graph and V^+ denote the set of vertices with out-degree greater than zero in G'. There are two cases:

Case: $|V^+| \leq n'/2$: In this case, take a branching $\mathcal{F}$ such that each vertex with in-degree greater than zero is included in some arborescence whose root is in V^+. This is always possible, as can be shown using induction on the number of vertices not in V^+. In particular, if you take a vertex u not in V^+, since there are no singleton vertices, u has in-degree greater than zero. Applying the inductive hypothesis to the graph $G' - \{u\}$, you get that u has some edge from a vertex in $G' - \{u\}$, which can be used to add u to an arborescence whose root is in V^+. Let $|\mathcal{F}|$ denote the total number of arborescences in $\mathcal{F}$. Since $|V^+| \leq n'/2$, 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 that are distance i from some root in L_0. Because these are trees, we can order the vertices within the levels in such a way that the leveling is planar. For our purposes, we say the levels increase from left to right. We claim that $\mathcal{F}$ is a Wheeler graph and that we can obtain

a proper ordering $<_\pi$ for the vertices of $\mathcal{F}$ from this leveling. To obtain the proper ordering, start with L_0 and read the order of the vertices on each level from the bottom to the top, then proceed right to the next level.

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 4, the optimal number of edges, denoted $|E^*|$ (including the $O(n')$ self-loops we removed earlier) is $O(n')$. Hence, the ratio of the optimal solution value over the branching solution value is bounded by a constant. In particular, $|E^*|/e(\mathcal{F}) \leq O(n')/(n'/2) = O(1)$. The construction of the branching, the planar leveling, and the extraction of $<_\pi$, can all be done in linear time.

Case $|V^+| > n'/2$: Here we first select a vertex $u \in V^+$ and take an edge $(u, v, 1)$ in G' to be included in our solution. If u is no longer the tail of any unselected edges, remove u from V^+. If $v \in V^+$, we remove from v from V^+. We then continue taking vertices from V^+ until it is empty. This creates a graph which is a collection of stars, and hence a Wheeler graph. Moreover, since every step adds an edge to our solution and removes at most two vertices from V^+, the resulting graph has at least $|V^+|/2 > n'/4$ edges. This gives us a solution with an approximation ratio of $|E^*|/|V^+| < O(n')/(n'/4) = O(1)$.

In both cases, we obtain an approximate solution with $\Theta(|E^*|)$ edges. □

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

$$|E^*| = \sum_{a=1}^{\sigma} |E_a^*| \leq \sigma \cdot \max_a |E_a^*| \leq \sigma \cdot \max_a |E'_a|.$$

Applying the result for $\sigma = 1$ (Lemma 11), we can approximate $\max_a |E'_a|$ with a solution having $\alpha \cdot \max_a |E'_a|$ edges for some constant $\alpha \leq 1$. Since

$$\frac{\alpha}{\sigma}|E^*| \leq \alpha \max_a |E'_a| \leq \max_a |E'_a| \leq |E^*|,$$

the solution provides a $\Omega(1/\sigma)$-approximation.

Theorem 8 *There exists a linear time $\Omega(1/\sigma)$-approximation algorithm for WS.*

We close this section by noting that the algorithm presented in Sect. 4 also provides us with an exponential time solution to the two optimization problems considered here in Sect. 5. 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 5 with the addition of one e term in the exponent. We have shown the following:

Theorem 9 *The WGV problem and WS problem for an input $G = (V, E)$ with $n = |V|$, $e = |E|$, and σ the size of the edge label alphabet, can be solved in time $2^{e \log \sigma + O(n+e)}$.*

6 A Class of Graphs with Linear Time Solution for Recognition

As mentioned earlier, it was shown by Alanko et al. [2] that there exists an algorithm that solves the recognition problem on 2-NFAs in linear time. Their algorithm works by reducing the recognition problem to a 2-SAT instance, which can then be efficiently solved. However, this approach fails to generalize for d-NFAs where $d > 2$. Here we allow for arbitrary levels of non-determinism, but we place rather stringent conditions on the graphs so that our techniques will work. It is also important to note that the motivation of pattern matching on graphs is not particularly well suited for this class of graphs (one of the main motivations of the work of Alanko et al.). We will see that these graphs can be easily converted into equivalent (from the pattern matching perspective) DFAs, which are trees, and hence Wheeler graphs. Instead, we take the viewpoint that these are ordering problems, where the edges in conjunction with Properties 1 and 2 form constraints, and the vertices need to be ordered in a way as to satisfy these constraints. The below characteristics make this ordering problem solvable in polynomial time.

We let V_0 denote the set of vertices with in-degree zero. We require that the graph G must have at least one vertex with in-degree zero, making V_0 non-empty. We also insist that all vertices in G must be reachable from some vertex in V_0. The following definitions describe the additional characteristics we require in order for our algorithm to work.

Definition 4 We consider a graph G to have *full-spectrum-outputs* if for every vertex v of out-degree greater than zero, every label appears on an edge leaving from v.

Definition 5 A graph G has the *unique-string-traversal property* if for every vertex v, all walks from V_0 to v form the same string when the walk's edge labels are concatenated.

Definition 6 A graph G is *prefix-free* if for every vertex v having out-degree zero, the string obtained by concatenating edge labels when traversing from V_0 to v is not a proper prefix of any string obtained by concatenating edge labels on a walk from V_0.

As a consequence of the unique-string-traversal property and all vertices being reachable from V_0, the graph G must also be a DAG. Indeed, a cyclic graph would have a least one vertex v such that walks of different lengths start in V_0 and end at v.

In Fig. 9 we see a simple example of two graphs that satisfy all of the stated conditions, however one is a Wheeler graph and the other is not. Furthermore, it can be seen from the reductions used in Sect. 2.2 that even when the input graphs satisfy the unique-string-traversal property and are prefix-free, the recognition problem remains NP-hard. We leave open whether the problem is NP-hard when restricted to instances that have full-spectrum-outputs and do not have the unique-string-traversal property and are not prefix-free.

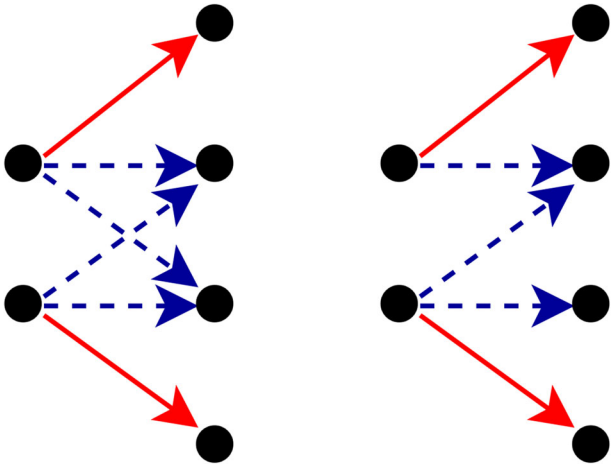

Fig. 9 On the left is an example of a small graph that has full-spectrum-outputs and the unique-string-traversal property, but is not a Wheeler graph. On the right is an example of a small graph that has all three properties and is a Wheeler graph

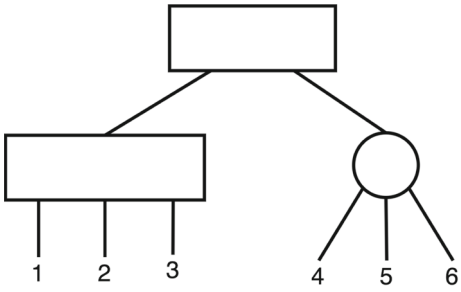

Fig. 10 Here p-nodes are represented by circles and q-nodes by rectangles. In this PQ-Tree the order on the leaves 1, 2, 3 can be reversed to 3, 2, 1, the leaves 4, 5, 6 can be permuted arbitrarily, and the order of the sets of leaves {1, 2, 3} and {4, 5, 6}, can be swapped

The stated conditions make the recognition problem tractable through the use of techniques similar to those used to detect leveled-planar DAGs. Before presenting our solution, we introduce an essential data structure, and the process by which it is used to detect whether a DAG is leveled-planar.

6.1 PQ-Trees

PQ-trees were introduced by Booth and Lueker for the purpose of solving the consecutive ones problem [7] and have since found applications in a wide range of problems, including planarity detection, detecting interval graphs, and graph embedding [7,11,25,30,32]. PQ-trees represent a set of possible orderings of the leaves which are subject to certain constraints. These constraints specify that some subset of the leaves must be contiguous in the ordering. The trees are made up of three types of nodes, p-nodes, q-nodes, and leaves. The p-nodes allow for arbitrary permutations of their child nodes, whereas q-nodes only allow for the reversal of the ordering on their

child nodes. The leaves represent the actual elements whose ordering we are interested in. See Fig. 10 for an example.

A universal PQ-tree is a p-node, v, where all of the leaves are v's children. The ϵ-tree, T_ϵ is a special tree which represents the empty set of orderings. We can take the intersection of two PQ-trees in time proportional to the sum of their two tree sizes [7]. The resulting PQ-tree represents the intersection of the orderings represented by each PQ-tree. Deletion of a leaf can be done in constant time.

6.2 Detecting Leveled-Planar DAGs

Detecting whether a DAG is leveled-planar is an important sub-step in determining whether a graph is a one-queue DAG [26,27]. A DAG, $G = (V, E)$, is leveled-planar if and only if it has a leveling (a partition of V into sets $V_1, V_2, \ldots, V_\ell$ where edges exists only from V_i to V_{i+1}, $1 \leq i \leq \ell - 1$), and there exists an ordering of vertices for each level that when combined provide a planar layout of G. An algorithm for determining if a DAG is leveled-planar is given in [27]. We describe the most relevant portions of this algorithm below.

The restrictions we place on our recognition problem allow for a simplified version of their algorithm that only works when every vertex is reachable from the first level of the leveling. We start with a leveling $V_1, \ldots, V_\ell$ of G. The idea is to process the leveling from left-to-right. For $1 \leq i \leq \ell$, we will build a PQ-tree T_i whose leaves represent the vertices in V_i. The PQ-tree T_i captures all possible orderings of vertices in V_i that still permit a leveled-planar layout of the subgraph induced by $V_1, \ldots, V_i$. To start, we make T_1 the universal PQ-tree whose leaves represent the vertices in V_1. For $i > 1$, we construct the tree T_i based on the PQ-tree T_{i-1} and the edges between V_{i-1} and V_i.

The key to being able to do this is the IDENTIFY operation, whose implementation can be found in [27]. The IDENTIFY operation takes four arguments: a PQ-tree T, two leaves x and y in T, and a new leaf z. IDENTIFY transforms T into a new PQ-tree, T', that contains z and does not contain the leaves x and y. The PQ-tree T' represents the subset of the permutations represented by T where x and y are adjacent. For these permutations the leaves are modified so that x and y are now a single leaf z. If no such permutations exist, T' is made into the ϵ-tree.

Equipped with the IDENTIFY operation, we can next describe how to obtain T_i from T_{i-1}. The PQ-tree T_i for level V_i is obtained from the PQ-tree T_{i-1} for level V_{i-1} as follows: Start by making T_i identical to T_{i-1}. For a vertex $u \in V_{i-1}$, if u has no out neighbors then u is deleted. If a vertex $u \in V_{i-1}$ has only a single neighbor $v \in V_i$, then replace u with a new leaf $v[u]$. If u has out neighbors $v_1, \ldots, v_i$, then u is made into a p-node with children $v_1[u], \ldots, v_i[u]$. This is repeated for all $u \in V_{i-1}$. Then for a fixed v, all nodes of the form $v[\cdot]$ are merged into a single node using the IDENTIFY operation. If at any point the ϵ-tree is returned, we stop and declare that the graph is not leveled-planar. A technical detail is that the order in which this operation must be applied is governed by the ordering of the nodes in V_i within the initial leveling. We refer the reader to [27] for details. Our algorithm for detecting Wheeler graphs under

the stated restrictions will build on this algorithm. For convenience, we will call the combined steps that create T_i from T_{i-1} *pushing*.

6.3 Linear Time Solution

The basic approach to solving this problem is to use a depth-first search, treating sets of vertices as a single vertex. These vertex sets will have PQ-trees pushed across them in a similar fashion as was done in the solution described above. The situation is slightly more complicated here as we have multiple edge types. This results in a tree structure, rather than a path of vertex sets. We will label the vertices representing vertex sets with capital letters and label the PQ-tree for a vertex set $W \subseteq V$ as T_W.

We split the algorithm into two parts. The first part is to create a tree where vertex sets play the role of vertices. It is a depth-first search using the edges between neighborhoods as connecting edges. The pseudocode is given in Algorithm 2. Let $N_a(W)$ denote the set of neighbors of the set W connected by an edge with label a. The function CREATEVERTEX takes a set of vertices and creates a new instance of a vertex class that can maintain pointers to its parent, children, internal vertices, and a string. Lemma 12 can be proven by applying induction to the number of edge labels, σ.

Algorithm 2 CreateNeighborhoodGraph

Require: Vertex set W with adjacency information
1: **function** CREATENEIGHBORHOODGRAPH(W):
2: **for all** $a \in [\sigma]$ **do**
3: **if** $N_a(W) \neq \emptyset$ **then**
4: $W_a \leftarrow$ CREATEVERTEX($N_a(W)$)
5: $W_a.parent \leftarrow W$
6: $W_a.string \leftarrow a \circ W.string$ ▷ Concatenate
7: $W.children$.ADD(CREATENEIGHBORHOODGRAPH(W_a))
8: **end if**
9: **end for**
10: **return** W
11: **end function**

Lemma 12 *If the given graph G is a Wheeler graph, in a proper ordering, the vertex sets obtained as above are ordered by the lexicographical ordering of their strings.*

An example of a tree obtained from Algorithm 2 is shown in Fig. 11. The vertex sets are disjoint due to the unique-string-traversal property. During Algorithm 2, we can identify if the graph satisfies the unique-string-traversal property by checking that every vertex in V gets included into exactly one vertex set. The prefix-free property can be easily checked as well.

Moving forward, the next portion of the algorithm is a recursive procedure that starts with the set of vertices having in-degree zero. Pseudocode is given in Algorithm 3. Letting V' be the vertices processed prior to reaching W, we assume inductively that the PQ-tree T_W represents all orderings of W such that if we fixed any one of these orderings there still exists a proper ordering of the vertices in V'. Then, after

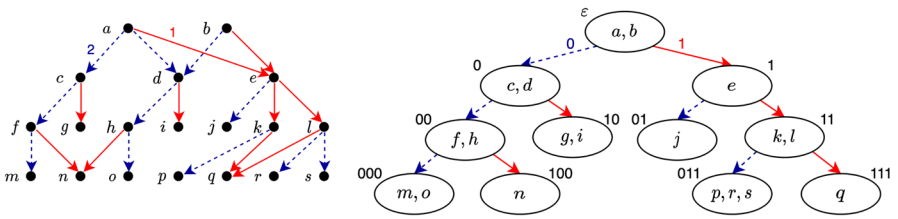

Fig. 11 On the right is the tree resulting from Algorithm 2 applied to the Wheeler graph shown on the left. For the graph on the left, red (solid) edges correspond to edges labeled 1, and blue (dashed) edges correspond to edges labeled 2. For the tree, an oval in the tree corresponds to a set of vertices in the Wheeler graph. The labels for these vertices are shown inside each oval. For each set of vertices inside an oval, the strings obtained by concatenating the edge labels on the path from the source is the same. These strings are shown to the side of each oval within the tree. In the tree, the edge colors indicated which type of edge was taken at each step along a path to that set

performing the first line of the for-loop, the PQ-tree T_{W_1} represents all orderings of W_1 such that if we fixed any one of these orderings there still exists a proper ordering of the vertices in $V' \cup W$. After performing the second line in the for-loop, T_{W_1} now represents all orderings of W_1 such that if we fixed any one of these orderings there still exists a proper ordering of the vertices in $V' \cup W$ and vertices that are descendants of W_1. After completing the third line in the loop, T_W represents all orderings of W such that if we fixed any one of them there still exists a proper ordering of the vertices in $V' \cup W_1$ and any descendants of W_1. We repeat this process for each of W's children. When finally returned, T_W represents all orderings of W such that there exists working orderings on V' and all descendants of W. The pseudocode for the whole algorithm is given in Algorithm 4.

The full-spectrum-output and prefix-free conditions are necessary to apply this algorithm. We need that every vertex in W maps onto some vertex in each of W's children. Thanks to these properties, when the PQ-tree T_{W_i} gets pushed back from a child W_i and the new PQ-tree T_W is created, all the vertices in W are also leaves in T_W, and we can take the intersection with the previous PQ-tree for W.

Algorithm 3 Propagating PQ-Trees

Require: PQ-Tree T_W and corresponding vertex set W.
1: **function** PROPAGATEPQTREES(T_W, W):
2: **if** $W.children = \emptyset$ **then**
3: **return** T_W
4: **end if**
5: **for all** $W_i \in W.children$ **do**
6: $T_{W_i} \leftarrow$ PUSH(T_W, W_i) ▷ Push PQ-Tree down to child.
7: $T_{W_i} \leftarrow$ PROPAGATEPQTREES(T_{W_i}, W_i) ▷ Recursively apply to children
8: $T_W \leftarrow T_W \cap$ PUSH(T_{W_i}, W) ▷ Push PQ-tree up from child and take intersection
9: **end for**
10: **return** T_W
11: **end function**

Algorithm 4 Detecting Wheeler graphs

Require: full-spectrum $G = (V, E)$ with unique-string-traversal/prefix-free properties.
1: **function** DETECTWHEELERGRAPH(G):
2: Let V_0 denote the set of all in-degree zero vertex in G
3: $V_0 \leftarrow$ CREATEVERTEX(V_0)
4: $V_0.string \leftarrow$ "ϵ"
5: $V_0 \leftarrow$ CREATENEIGHBORHOODGRAPH(V_0)
6: Construct T_{V_0}, the universal tree with leaves V_0
7: **if** PROPAGATEPQTREES(T_{V_0}, V_0) $\neq T_\epsilon$ **then**
8: **return** 'YES'
9: **else**
10: **return** 'NO'
11: **end if**
12: **end function**

Time Complexity: Each set of edges between two vertex sets has PQ-trees pushed across it twice. These pushes can be done in time proportional to the number of edges. In addition, all intersections can be done in time proportional to the number of vertices. As a result of these two facts, the overall algorithm can be performed in linear time. We have demonstrated the following:

Theorem 10 *It can be determined in linear time if a directed, edge labeled graph that has full-spectrum-outputs, is prefix-free, and has the unique-string-traversal property, is a Wheeler graph.*

7 Discussion and Open Problems

We have shown that recognizing Wheeler graphs is indeed a hard problem in general. We have also shown a special case where the recognition problem can be performed efficiently. The most important directions to expand this research appear to be in identifying more classes of graphs where this can be done in polynomial time. We can also ask for improvements on the algorithms presented here. Specifically, we ask:

- Is the Wheeler graph recognition problem NP-complete for 3-NFA and 4-NFA?
- For which other classes of graphs can Wheeler graph recognition be done efficiently?
- Is there a fixed-parameter-tractable exponential time algorithm for any of the hard problems given in this paper?
- Can we provide a better approximation algorithm for the optimization variants?

Constructive answers to these questions will contribute to our knowledge on finding vertex orderings "close" to that required for a Wheeler graph. It will aid in our ability to apply BWT based indices to various structures, as well as our ability to find useful compressible subgraphs.

Acknowledgements This research is supported in part by the U.S. National Science Foundation under the Grants CCF-1703489 and CCF-2112643. The first author has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 690941. We thank Travis Gagie and Nicola Prezza for introducing this problem to us. We also thank

the anonymous reviewers of ESA 2019, where a preliminary version of this paper was published [23]. The author would also like to acknowledge the BIRDS project (Bioinformatics and Information Retrieval Data Structures Analysis and Design) and the Workshop on Compression, Text and Algorithms 2018.

Funding: This research is supported in part by the U.S. National Science Foundation under the Grants CCF-1703489 and CCF-2112643. The first author has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 690941.

Declarations

Conflict of interest The authors declare that they have no conflicts/competing interests.

Consent for publication The authors consent for this work to be published in Algorithmica.

References

1. Aho, A.V., Corasick, M.J.: Efficient string matching: an aid to bibliographic search. Commun. ACM **18**(6), 333–340 (1975). https://doi.org/10.1145/360825.360855
2. Alanko, J., D'Agostino, G., Policriti, A., Prezza, N.: 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, pp. 911–930 (2020). https://doi.org/10.1137/1.9781611975994.55
3. Alanko, J., D'Agostino, G., Policriti, A., Prezza, N.: Wheeler languages. Inf. Comput. (2021). https://doi.org/10.1016/j.ic.2021.104820
4. Alanko, J.N., Gagie, T., Navarro, G., Benkner, L.S.: Tunneling on wheeler graphs. In: Data Compression Conference, DCC 2019, Snowbird, UT, USA, March 26–29, 2019, pp. 122–131 (2019). https://doi.org/10.1109/DCC.2019.00020
5. Babai, L., Luks, E.M.: Canonical labeling of graphs. In: Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25–27 April, 1983, Boston, Massachusetts, USA, pp. 171–183 (1983). https://doi.org/10.1145/800061.808746
6. Belazzougui, D.: Succinct dictionary matching with no slowdown. In: Combinatorial Pattern Matching, 21st Annual Symposium, CPM 2010, New York, NY, USA, June 21–23, 2010. Proceedings, pp. 88–100 (2010). https://doi.org/10.1007/978-3-642-13509-5_9
7. Booth, K.S.: Pq-tree algorithms. Technical report, California University, Livermore (USA). Lawrence Livermore Laboratory (1975)
8. Bowe, A., Onodera, T., Sadakane, K., Shibuya, T.: Succinct de bruijn graphs. In: Algorithms in Bioinformatics—12th International Workshop, WABI 2012, Ljubljana, Slovenia, September 10–12, 2012. Proceedings, pp. 225–235 (2012). https://doi.org/10.1007/978-3-642-33122-0_18
9. Burrows, M., Wheeler, D.J.: A block-sorting lossless data compression algorithm. SRC Research Report (1994)
10. Chen, J., Liu, Y., Lu, S., O'Sullivan, B., Razgon, I.: A fixed-parameter algorithm for the directed feedback vertex set problem. J. ACM **55**(5), 21:1-21:19 (2008). https://doi.org/10.1145/1411509.1411511
11. Chiba, N., Nishizeki, T., Abe, S., Ozawa, T.: A linear algorithm for embedding planar graphs using pq-trees. J. Comput. Syst. Sci. **30**(1), 54–76 (1985). https://doi.org/10.1016/0022-0000(85)90004-2
12. Claude, F., Navarro, G., Pereira, A.O.: The wavelet matrix: an efficient wavelet tree for large alphabets. Inf. Syst. **47**, 15–32 (2015). https://doi.org/10.1016/j.is.2014.06.002
13. Cotumaccio, N., Prezza, N.: On indexing and compressing finite automata. In: D. Marx (ed) Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10–13, 2021, pp. 2585–2599. SIAM (2021). https://doi.org/10.1137/1.9781611976465.153
14. De Bruijn, N.G.: A combinatorial problem. Koninklijke Nederlandse Akademie v. Wetenschappen **49**(49), 758–764 (1946)
15. Dujmovic, V., Wood, D.R.: On linear layouts of graphs. Discrete Math. Theor. Comput. Sci. **6**(2), 339–358 (2004). (http://dmtcs.episciences.org/317)
16. Equi, M., Grossi, R., Mäkinen, V., Tomescu, A.I.: On the complexity of string matching for graphs. In: C. Baier, I. Chatzigiannakis, P. Flocchini, S. Leonardi (eds) 46th International Colloquium on

Automata, Languages, and Programming, ICALP 2019, July 9–12, 2019, Patras, Greece, LIPIcs, vol. 132, pp. 55:1–55:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPIcs.ICALP.2019.55
17. Ferragina, P., Luccio, F., Manzini, G., Muthukrishnan, S.: Compressing and indexing labeled trees, with applications. J. ACM **57**(1), 4:1-4:33 (2009). https://doi.org/10.1145/1613676.1613680
18. Ferragina, P., Manzini, G.: Indexing compressed text. J. ACM **52**(4), 552–581 (2005). https://doi.org/10.1145/1082036.1082039
19. Ferragina, P., Venturini, R.: The compressed permuterm index. ACM Trans. Algorithms **7**(1), 10:1-10:21 (2010). https://doi.org/10.1145/1868237.1868248
20. Gagie, T., Manzini, G., Sirén, J.: Wheeler graphs: a framework for bwt-based data structures. Theor. Comput. Sci. **698**, 67–78 (2017). https://doi.org/10.1016/j.tcs.2017.06.016
21. Ganguly, A., Shah, R., Thankachan, S.V.: 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, pp. 397–407 (2017). https://doi.org/10.1137/1.9781611974782.25
22. Gibney, D., Hoppenworth, G., Thankachan, S.V.: Simple reductions from formula-sat to pattern matching on labeled graphs and subtree isomorphism. In: H.V. Le, V. King (eds) 4th Symposium on Simplicity in Algorithms, SOSA 2021, Virtual Conference, January 11–12, 2021, pp. 232–242. SIAM (2021). https://doi.org/10.1137/1.9781611976496.26
23. Gibney, D., Thankachan, S.V.: 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, pp. 51:1–51:16 (2019). https://doi.org/10.4230/LIPIcs.ESA.2019.51
24. Guruswami, V., Manokaran, R., Raghavendra, P.: 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, pp. 573–582 (2008). https://doi.org/10.1109/FOCS.2008.51
25. Haeupler, B., Tarjan, R.E.: Planarity algorithms via pq-trees (extended abstract). Electr. Not. Discrete Math. **31**, 143–149 (2008). https://doi.org/10.1016/j.endm.2008.06.029
26. Heath, L.S., Pemmaraju, S.V.: Stack and queue layouts of directed acyclic graphs: Part II. SIAM J. Comput. **28**(5), 1588–1626 (1999). https://doi.org/10.1137/S0097539795291550
27. Heath, L.S., Pemmaraju, S.V., Trenk, A.N.: Stack and queue layouts of directed acyclic graphs: Part I. SIAM J. Comput. **28**(4), 1510–1539 (1999). https://doi.org/10.1137/S0097539795280287
28. Heath, L.S., Rosenberg, A.L.: Laying out graphs using queues. SIAM J. Comput. **21**(5), 927–958 (1992). https://doi.org/10.1137/0221055
29. Hon, W., Ku, T., Shah, R., Thankachan, S.V., Vitter, J.S.: Faster compressed dictionary matching. Theor. Comput. Sci. **475**, 113–119 (2013). https://doi.org/10.1016/j.tcs.2012.10.050
30. Jiang, H., Chauve, C., Zhu, B.: Breakpoint distance and pq-trees. In: Combinatorial Pattern Matching, 21st Annual Symposium, CPM 2010, New York, NY, USA, June 21–23, 2010. Proceedings, pp. 112–124 (2010). https://doi.org/10.1007/978-3-642-13509-5_11
31. Kann, V.: On the approximability of np-complete optimization problems. Ph.d. thesis, Royal Institute of Technology Stockholm (1992)
32. Landau, G.M., Parida, L., Weimann, O.: Gene proximity analysis across whole genomes via PQ trees[1]. J. Comput. Biol. **12**(10), 1289–1306 (2005). https://doi.org/10.1089/cmb.2005.12.1289
33. Mantaci, S., Restivo, A., Rosone, G., Sciortino, M.: 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, pp. 178–189 (2005). https://doi.org/10.1007/11496656_16
34. Mantaci, S., Restivo, A., Rosone, G., Sciortino, M.: An extension of the Burrows–Wheeler transform. Theor. Comput. Sci. **387**(3), 298–312 (2007). https://doi.org/10.1016/j.tcs.2007.07.014
35. Miller, G.L.: Graph isomorphism, general remarks. J. Comput. Syst. Sci. **18**(2), 128–142 (1979). https://doi.org/10.1016/0022-0000(79)90043-6
36. Novak, A.M., Garrison, E., Paten, B.: A graph extension of the positional Burrows–Wheeler transform and its applications. Algorithms Mol. Biol. **12**(1), 18:1-18:12 (2017). https://doi.org/10.1186/s13015-017-0109-9
37. Opatrny, J.: Total ordering problem. SIAM J. Comput. **8**(1), 111–114 (1979). https://doi.org/10.1137/0208008

38. Sirén, J., Välimäki, N., Mäkinen, V.: Indexing graphs for path queries with applications in genome research. IEEE ACM Trans. Comput. Biol. Bioinf. (TCBB) **11**(2), 375–388 (2014)
39. Younger, D.: Minimum feedback arc sets for a directed graph. IEEE Trans. Circuit Theory **10**(2), 238–245 (1963)

Authors and Affiliations

Daniel Gibney[1] · **Sharma V. Thankachan**[2]

✉ Daniel Gibney
daniel.j.gibney@gmail.com

Sharma V. Thankachan
sharma.thankachan@ucf.edu

[1] School of Computational Science and Engineering, Georgia Institute of Technology, CODA Tech Square, 756 West Peachtree Street, NW, 12th Fl., Station S1257A, Atlanta, GA 30308, USA

[2] Department of Computer Science, University of Central Florida, 4000 Central Florida Blvd., Orlando, FL 32816-2362, USA