Sorting Finite Automata via Partition Refinement

Ruben Becker
Ca' Foscari University of Venice, Italy

Manuel Cáceres
University of Helsinki, Finland

Davide Cenzato
Ca' Foscari University of Venice, Italy

Sung-Hwan Kim
Ca' Foscari University of Venice, Italy

Bojana Kodric
Ca' Foscari University of Venice, Italy

Francisco Olivares
CeBiB – Centre for Biotechnology and Bioengineering, Santiago, Chile
Department of Computer Science, University of Chile, Santiago, Chile

Nicola Prezza
Ca' Foscari University of Venice, Italy

Abstract

Wheeler nondeterministic finite automata (WNFAs) were introduced in (Gagie et al., TCS 2017) as a powerful generalization of prefix sorting from strings to labeled graphs. WNFAs admit optimal solutions to classic hard problems on labeled graphs and languages such as compression and regular expression matching. The problem of deciding whether a given NFA is Wheeler is known to be NP-complete (Gibney and Thankachan, ESA 2019). Recently, however, Alanko et al. (Information and Computation 2021) showed how to side-step this complexity by switching to *preorders*: letting Q be the set of states and δ the set of transitions, they provided a $O(|\delta| \cdot |Q|^2)$-time algorithm computing a totally-ordered *partition* (i.e. equivalence relation) of the WNFA's states such that (1) equivalent states recognize the same regular language, and (2) the order of (the classes of) non-equivalent states is consistent with any Wheeler order, when one exists. As a result, the output is a preorder of the states as useful for pattern matching as standard Wheeler orders.

Further extensions of this line of work (Cotumaccio et al., SODA 2021 and DCC 2022) generalized these concepts to arbitrary NFAs by introducing *co-lex partial preorders*: in general, any NFA admits a partial preorder of its states reflecting the co-lexicographic order of their accepted strings; the smaller the width of such preorder is, the faster regular expression matching queries can be performed. To date, the fastest algorithm for computing the smallest-width partial preorder on NFAs runs in $O(|\delta|^2 + |Q|^{5/2})$ time (Cotumaccio, DCC 2022), while on DFAs the same task can be accomplished in $O(\min(|Q|^2 \log |Q|, |\delta| \cdot |Q|))$ time (Kim et al., CPM 2023).

In this paper, we provide much more efficient solutions to the co-lex order computation problem. Our results are achieved by extending a classic algorithm for the relational coarsest partition refinement problem of Paige and Tarjan to work with *ordered* partitions. More specifically, we provide a $O(|\delta| \log |Q|)$-time algorithm computing a co-lex total preorder when the input is a Wheeler NFA, and an algorithm with the same time complexity computing the smallest-width co-lex partial order of any DFA. In addition, we present implementations of our algorithms and show that they are very efficient also in practice.

2012 ACM Subject Classification Theory of computation → Sorting and searching; Theory of computation → Graph algorithms analysis; Theory of computation → Pattern matching

Keywords and phrases Wheeler automata, prefix sorting, pattern matching, graph compression, sorting, partition refinement

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.15

31st Annual European Symposium on Algorithms (ESA 2023).
Editors: Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman; Article No. 15; pp. 15:1–15:15

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Related Version *Full Version*: http://arxiv.org/abs/2305.05129

Supplementary Material *Software (Source Code)*:
https://github.com/regindex/finite-automata-partition-refinement

Funding *Ruben Becker, Davide Cenzato, Sung-Hwan Kim, Bojana Kodric, Nicola Prezza*: Funded by the European Union (ERC, REGINDEX, 101039208). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.
Manuel Cáceres: Funded by the Academy of Finland (grants No. 352821, 328877).
Francisco Olivares: Funded by Ph.D Scholarship 21210579, ANID, Chile.

1 Introduction

The classical *pattern matching* problem between two strings S (the text) and P (the pattern) over alphabet Σ asks to find the substrings of S matching P. Although many algorithms solving the *on-line* version of the problem exist, in many scenarios it is possible to pre-process S *off-line* into an *index* to speed up subsequent pattern matching queries. As a matter of fact, a very successful line of research dating back to the invention of suffix trees [24] and culminating with the discovery of compressed data structures [20] showed that it is possible to represent S in compact space while speeding up matching queries.

The *indexed pattern matching* problem can be generalized to collections of strings: in this case the pattern must be found as a substring in a string collection $\mathcal{S}$. A natural approach to solve this problem is to concatenate all strings in $\mathcal{S}$ into one string S and re-use the well-optimized techniques for classical pattern matching. Even though there are many successful examples of indexes following this strategy [12, 14, 19], this approach suffers from high space consumption during index construction (the input is very large), and does not scale to a more general (and interesting) scenario: the case where $\mathcal{S}$ contains an *infinite* number of strings. A solution, addressing both issues, is to represent a (potentially infinite) collection of strings using a finite-state automaton with set of states Q and transition function δ. In this new scenario, the goal of pattern matching is to locate walks in the automaton (seen as a labeled graph) spelling the query pattern P. In bioinformatics, for example, genomic collections are represented using pangenome graphs: labeled graphs encoding nucleotide variations within the collection [5, 23]. Matching a DNA sequence over a pangenome graph allows one to discover the genetic variation in a population [22].

Unfortunately, both the off-line and on-line pattern matching problems are hard to solve on labeled graphs: Equi et al. [10, 11] showed that, conditioned on OVH [25], it is not possible to design a polynomial-time algorithm to index a labeled graph such that pattern matching queries can be answered in $O(|P| + |\delta|^{\alpha}|P|^{\beta})$ time, for any constants $\alpha < 1$ or $\beta < 1$.

A successful paradigm to cope with this hardness is to study sub-classes of graphs where the problem is easier. Along these lines, Gagie et al. [13] introduced the class of *Wheeler automata*: labeled graphs admitting a total order of their states (a *Wheeler order*) which respects the underlying alphabet's order and it is propagated through pairs of equally-labeled edges. Wheeler orders generalize prefix sorting (the machinery standing at the core of the most successful string indexes) to labeled graphs and as a consequence, an index over a Wheeler automaton supports pattern matching queries in near-optimal $O(|P| \log |\Sigma|)$ time.

Wheeler automata, however, have two important limitations. First, the classes of Wheeler automata and Wheeler languages (accepted by such automata) are quite restricted. For example, Wheeler automata cannot contain proper cycles labeled with a unary string, and

moreover, Wheeler languages are star-free [2]. Second, several natural problems related to Wheeler graphs are computationally hard. For example, the simple fact of deciding if an automaton is Wheeler is NP-complete, even when the automaton is acyclic [15].

Related to the first issue, the work [8] extended state-ordering to arbitrary automata by using the concept of *co-lex partial order*. By switching from total to partial orders, the authors showed that (i) every automaton can be (partially) sorted, and that (ii) the efficiency of pattern matching on the automaton depends on the *width* (maximum size of an antichain) of such partial order. Wheeler automata are the particular case in which there exists a total co-lex order (i.e., of width one), thus enabling near-optimal time pattern matching queries.

A way to circumvent the latter limitation (hardness of computing a Wheeler order) was proposed by Alanko et al. [2] by switching to *total preorders*: the authors showed a $O(|\delta| \cdot |Q|^2)$-time *partition refinement* algorithm, which outputs a totally-ordered partition of Q such that (1) states in the same part recognize the same language, and (2) the order of the partition is consistent with *any* Wheeler order of Q. In other words, this ordered partition is a preorder of the states as useful for indexing as Wheeler orders. Recently, a similar solution was proposed by Chao et al. [4] to speed up the computation of Wheeler orders in practice; after a first polynomial-time partition refinement step, their tool runs an exponential-time solver to assign a Wheeler ordering within classes of equivalent states.

These strategies – partial orders and total preorders – were finally merged by Cotumaccio [6]. As in [8], the efficiency of pattern matching depends on the width of such a partial preorder. The author described a polynomial-time algorithm computing a *colex partial preorder* of smallest width over an arbitrary nondeterministic finite-state automaton (NFA) in $O(|\delta|^2 + |Q|^{5/2})$ time. Later, this running time was improved by Kim et al. [17] in the particular case of DFAs with two algorithms running in time $O(|Q|^2 \log |Q|)$ and $O(|\delta| \cdot |Q|)$, respectively, and one algorithm running in near-optimal time $O(|\delta| \log |Q|)$ on acyclic DFAs. Within the same running times, all the above-mentioned algorithms compute also a chain partition of minimum size p of the partial preorder (by Dilworth's theorem [9], p is equal to the order's width); such a chain partition is needed to build the index described in [6]. The index can be built in linear time given as input such a chain partition, uses $(\lceil \log |\Sigma| \rceil + \lceil \log p \rceil + 2) \cdot (1 + o(1))$ bits per edge, and answers pattern matching queries on the regular language accepted by the automaton in $O(p^2 \log(p \cdot |\Sigma|))$ time per pattern's character.

Our contributions. We consider two cases of colex orders: (1) total preorders on NFAs, and (2) partial orders of minimum width on DFAs. We provide algorithms running in time $O(|\delta| \log |Q|)$ for both cases, improving the state-of-the-art cubic and quadratic algorithms to near-optimal time.

Our solution to (1) is obtained by extending a classic algorithm for the relational coarsest partition refinement problem of Paige and Tarjan [21] to work with *ordered* partitions. Our algorithm starts from the ordered partition corresponding to the states' incoming labels and, similarly to [21], iteratively refines this partition by enforcing *forward-stability*: for any two parts B (the "splitter") and D (the "split") the image $\delta_a(B)$ of B through the transition function (for any alphabet's character a) must either contain D or be disjoint with D. If this condition is not satisfied, D is split into three parts: states reached only from B, states reached only from $Q \setminus B$, and states reached from both. In addition to Paige and Tarjan's algorithm, we show how to sort these three parts consistently with the current partition's order. In fact, we show that our algorithm finds a total preorder on a class of automata being strictly larger than the Wheeler automata. We dub this class *quasi-Wheeler*, thereby extending the class of automata that can be indexed to support queries in near-optimal time.

▶ **Theorem 1.** *For an NFA $\mathcal{A} = (Q, \delta, s)$, we can compute a total preorder $\preceq$ in $O(|\delta| \log |Q|)$ time such that $\preceq$ corresponds to a Wheeler preorder if and only if $\mathcal{A}$ is quasi-Wheeler.*

Our solution to (2) relies on a further modification of the partition refinement algorithm. More specifically, we show that within the same time complexity, we can prune the automaton's transition function so that the resulting graph is a quasi-forest satisfying the following property: every state u has only one incoming walk, whose label is co-lexicographically smallest (the *infimum*) among the labels of all walks ending at u in the original DFA. Symmetrically, we compute a pruned automaton encoding the co-lexicographically largest strings (the *suprema*). We obtain our second result by plugging in the $O(|Q| \log |Q|)$-time algorithms of Kim et al. [17], which sort the infimum and suprema strings (suffix-doubling)[1] and then compute a minimum chain partition of the smallest-width co-lex order (interval graph coloring). Our result assumes that the DFA is input-consistent (all in-going transitions to a state are labeled with the same letter). This assumption is a common simplification in automata theory and it is not a limitation of our approach as every DFA can be easily transformed into an equivalent input-consistent DFA.

▶ **Theorem 2.** *For an input-consistent DFA $\mathcal{A} = (Q, \delta, s)$, we can compute a minimum chain partition of the smallest-width co-lex order in $O(|\delta| \log |Q|)$ time.*

We have implemented our partition refinement algorithms. We compared the implementation of our algorithm from Theorem 1 with a heuristic implementation from `WGT` [4] (their first polynomial time step) which, similarly to our algorithm, computes an ordered partition being consistent with any Wheeler order (although not the most refined – unlike the output of our algorithm). On random Wheeler NFAs generated with a tool from the same toolbox (`WGT`), our implementation outperforms the heuristic from `WGT` by more than two orders of magnitude. We also experimentally show that our algorithm from Theorem 2 can prune a pangenomic DFA with more than 50 million states in less than 6 minutes.

2 Notation and Preliminaries

For an integer $k \geq 1$, we let $[k] := \{1, \ldots, k\}$. Given a set U, a partition $\mathcal{P}$ of U is a set of pairwise disjoint non-empty sets $\{U_1, \ldots, U_k\}$ whose union is U. We call $U_1, \ldots, U_k$ the *parts* of $\mathcal{P}$. If we, in addition, have a (total) order of the parts, we say that it is an *ordered partition* and denote it with $\langle U_1, \ldots, U_k \rangle$. For two (ordered) partitions $\mathcal{P}$ and $\mathcal{P}'$ of U, we say that $\mathcal{P}'$ is a *refinement* of $\mathcal{P}$ if every part of $\mathcal{P}'$ is contained in a part of $\mathcal{P}$, i.e., for every $U' \in \mathcal{P}'$ there is $U \in \mathcal{P}$ with $U' \subseteq U$. As a special case, a partition is a refinement of itself.

Relations and Orders. A *relation* R over a set U is a set of ordered pairs of elements from U, i.e., $R \subseteq U \times U$. We sometimes omit U if it is clear from the context. For two elements u, v from U, we usually write uRv for $(u, v) \in R$.

A *strict partial order* over a set U is a relation that satisfies irreflexivity, asymmetry, and transitivity. If, in addition, a strict partial order satisfies connectedness it is a *strict total order*. A *total preorder* over a set U is a relation that satisfies transitivity, reflexivity, and connectedness (i.e., for all two distinct $u, v \in U$, u is in relation with v or v is in relation with u). An *equivalence relation* over a set U is a relation that satisfies reflexivity, symmetry,

[1] More precisely, it is a special case of the suffix doubling algorithm [17], which runs in the claimed time.

and transitivity. For an equivalence relation $\sim$, we use $[u]_\sim$ to denote the equivalence class of an element $u \in U$ with respect to $\sim$, i.e., $[u]_\sim := \{v \in U : u \sim v\}$. We denote with $U/_\sim$ the partition of U consisting of all equivalence classes $[u]_\sim$ for $u \in U$. In this paper, we denote strict total orders with the symbols $\prec$ and $<$, total preorders with the symbols $\preceq$ and $\leq$, and equivalence relations with the symbols $\sim$ and $\approx$.

A total preorder $\preceq$ over U induces an equivalence relation $\sim$ over U: For $u, v \in U$, define $u \sim v$ if and only if $u \preceq v$ and $v \preceq u$. Throughout the paper, $\sim$ will always refer to the equivalence induced by $\preceq$ (the order $\preceq$ will always be unambiguously defined). A total preorder $\preceq$, in addition, yields a strict total order $\prec$ on the elements of $U/_\sim$ as $[u]_\sim \prec [v]_\sim$ if and only if $u \preceq v$ and not $v \preceq u$. Throughout the paper $\prec$ will refer to the strict order induced by $\preceq$ (always unambiguously defined). A total preorder $\preceq$ over a set U can thus be represented by a unique ordered partition $\langle U_1, \ldots, U_k \rangle$, where the parts U_i represent the equivalence classes with respect to $\sim$ and their ordering represents the strict total order $\prec$.

Non-Deterministic Finite Automata (NFAs). Let Σ denote a fixed finite and non-empty *alphabet* of *letters*. We assume that there is a strict total order $<$ on the alphabet Σ.

▶ **Definition 3** (NFA and DFA). *A* non-deterministic finite automaton (NFA) *over the alphabet Σ is an ordered triple $\mathcal{A} = (Q, \delta, s)$, where Q is the set of* states*, $\delta : Q \times \Sigma \to 2^Q$ is the* transition function*, and $s \in Q$ is the* source state*.*

A deterministic finite automaton (DFA) *over the alphabet Σ is an NFA over Σ such that $|\delta(v, a)| \leq 1$ for all $a \in \Sigma$ and $v \in V$.*

We note that the standard definition of NFAs includes also a set of final states. As we are not concerned with the accepting languages of automata in this work, we omit the final states from the definition. In what follows we consider the alphabet Σ to be fixed and thus frequently refer to NFAs without specifying the alphabet. Given an NFA $\mathcal{A} = (Q, \delta, s)$, for a state $u \in Q$ and a letter $a \in \Sigma$, we use the shortcut $\delta_a(u)$ for $\delta(u, a)$, similarly $\delta_a^{-1}(u) = \{v \ : \ \delta(v, a) = u\}$. For a set $S \subseteq Q$, we let $\delta_a(S) := \bigcup_{u \in S} \delta_a(u)$ and $\delta_a^{-1}(S) := \bigcup_{u \in S} \delta_a^{-1}(u)$.

The set of finite strings over Σ, denoted by Σ^*, is the set of finite sequences of letters from Σ. We extend the transition function from letters to finite strings in the common way, i.e, for $\alpha \in \Sigma^*$ and $v \in Q$, we define $\delta_\alpha(v)$ recursively as follows. If $\alpha = \varepsilon$, we let $\delta_\alpha(v) = \{v\}$. Otherwise, if $\alpha = a\alpha'$ for some $a \in \Sigma$ and $\alpha' \in \Sigma^*$, we let $\delta_\alpha(v) = \bigcup_{w \in \delta_a(v)} \delta_{\alpha'}(w)$.

▶ **Definition 4** (Strings Reaching a State). *Given an NFA $\mathcal{A} = (Q, \delta, s)$, for a state $v \in Q$, the* set of strings reaching v *is defined as $S_v = \{\alpha \in \Sigma^* : v \in \delta_\alpha(s)\}$.*

Equivalently, S_v is the regular language recognized by state v. For simplicity, in this work we assume that NFAs satisfy the following two properties: (1) The source state has no in-going transitions, i.e., $S_s = \{\varepsilon\}$ and every other state is reachable from the source state, i.e., $|S_v| \geq 1$ for all $v \in Q \setminus \{s\}$. (2) The automaton is *input-consistent*, i.e., for every state $v \in Q \setminus \{s\}$ it holds that $\delta_a^{-1}(v)$ is non-empty for exactly one letter $a \in \Sigma$. It is easy to see that any automaton can be transformed into an equivalent one (in the sense of the recognized language) satisfying assumptions (1) and (2). Condition (1) yields that $|Q| \leq |\delta| + 1$. Condition (2) comes at the cost of increasing the number of states and transitions by a factor of at most $|\Sigma|$. We also assume that a given NFA contains at least one transition for every letter. We denote by $\lambda(v)$ the unique label of the in-going transitions of a state $v \in Q \setminus \{s\}$, while $\lambda(s) := \varepsilon$. We also extend this notation to sets of states, that is for a set of states S, we let $\lambda(S) := \{\lambda(v) : v \in S\}$ and if $\lambda(S) = \{a\}$, we write $\lambda(S) = a$.

Forward-Stable Partitions. Alanko et al. consider *forward-stable partitions* [2, Section 4.2].

▶ **Definition 5** (Forward-Stability). *Given an NFA $\mathcal{A} = (Q, \delta, s)$ and two sets of states $S, T \subseteq Q$, we say that S is* forward-stable *with respect to T, if, for all $a \in \Sigma$, $S \subseteq \delta_a(T)$ or $S \cap \delta_a(T) = \emptyset$ holds. A partition $\mathcal{P}$ of $\mathcal{A}$'s states is* forward-stable for $\mathcal{A}$*, if, for any two parts $S, T \in \mathcal{P}$, it holds that S is forward-stable with respect to T.*

A direct consequence of forward-stability is given in the following lemma, i.e., all states in the same part of a forward-stable partition are reached by the exact same set of strings.

▶ **Lemma 6.** *Let $\mathcal{P}$ be a forward-stable partition for an NFA $\mathcal{A} = (Q, \delta, s)$ and assume that $u, v \in P$ for some part $P \in \mathcal{P}$. Then, $S_u = S_v$.*

This property can be proven easily using the definition of forward-stability, see, e.g., [2, Lemma 4.7]. Furthermore, there is a straightforward relationship between forward-stability and bisimulation, see, e.g., the work of Kanellakis and Smolka [16] or Chapter 7.3 of the book by Katoen and Baier [3]. The *coarsest* forward-stable partition for an NFA $\mathcal{A}$, i.e., the forward-stable partition with fewest parts, is identical to the partition consisting of the equivalence classes with respect to the bisimilarity relation (the unique largest bisimulation) on $\mathcal{A}^{-1}$, the automaton obtained from $\mathcal{A}$ by reversing all its transitions. We include a proof of this fact in full version of this article. This also directly yields that there is a unique forward-stable partition. We note that a reverse statement of Lemma 6 may not necessarily hold even for the coarsest forward-stable partition. More precisely, states in different parts of the coarsest forward-stable partition may have the same set of strings reaching them, see the left automaton in Figure 1. In what follows, for an NFA $\mathcal{A} = (Q, \delta, s)$ and an equivalence relation $\sim$ on Q, we write $\mathcal{A}/_\sim$ for the quotient NFA (of $\mathcal{A}$ with respect to $\sim$) obtained by collapsing the equivalence classes into single states, see [2, Definition 8] for a formal definition.

Wheeler NFAs and Quasi-Wheeler NFAs. Wheeler NFAs [13] are a special class of NFAs that can be stored compactly and indexed efficiently as they can be endowed with a specific type of strict total order, the so-called *Wheeler order*.

▶ **Definition 7** (Wheeler NFA and Wheeler order). *Let $\mathcal{A} = (Q, \delta, s)$ be an NFA. We say that $\mathcal{A}$ is a* Wheeler NFA*, if there exists a* Wheeler order $\prec$ *of Q. A Wheeler order $\prec$ of Q is a strict total order on Q such that the source state precedes all other states, i.e., $s \prec v$ for all $v \in Q \setminus \{s\}$, and, for any pair $v \in \delta_a(u)$ and $v' \in \delta_{a'}(u')$:*

(1) *If $a < a'$, then $v \prec v'$.*

(2) *If $a = a'$, $u \prec u'$, and $v \neq v'$, then $v \prec v'$.*

Clearly, not every NFA is a Wheeler NFA and, even worse, recognizing if a given NFA is Wheeler is NP-complete (for alphabet size at least 2) as shown by Gibney and Thankachan [15]. We now introduce the following problem, called OrderWheeler: Given an arbitrary NFA $\mathcal{A} = (Q, \delta, s)$ as input, the task is to compute a strict total order $\prec$ of Q with the property that $\prec$ is a Wheeler order if $\mathcal{A}$ is Wheeler. As a result of the NP-completeness of recognizing Wheeler NFAs previously mentioned, also OrderWheeler is NP-hard. This follows as checking whether a given order is indeed a Wheeler order can be done in linear time [1, Lemma 3.1]. This motivates introducing the following relaxed version of Wheeler NFAs.

▶ **Definition 8** (Wheeler preorders and quasi-Wheeler NFAs). *Let $\mathcal{A} = (Q, \delta, s)$ be an NFA. A Wheeler preorder $\preceq$ on Q is a total preorder on Q such that:*

- *The partition $Q/_\sim$, where $\sim$ is the equivalence relation induced by $\preceq$, is equal to the coarsest forward-stable partition of $\mathcal{A}$.*
- *The quotient automaton $\mathcal{A}/_\sim$ is a Wheeler NFA with the strict total order $\prec$ induced by $\preceq$ on the equivalence classes with respect to $\sim$.*

We say that $\mathcal{A}$ is a quasi-Wheeler NFA*, if there exists a* Wheeler preorder $\preceq$ *on Q.*

See the beginning of this section for the formal definition of $\sim$ and $\prec$. From the point of view of indexing, quasi-Wheeler NFAs are as useful as Wheeler NFAs: note that the quotient automaton $\mathcal{A}/_\sim$ in the definition above generates the same language as the original NFA $\mathcal{A}$ due to the properties of the forward-stable partition, see Lemma 6. However, $\mathcal{A}/_\sim$ is a Wheeler NFA and can thus be stored compactly and indexed efficiently. We next note that Corollary 4.11 in the paper by Alanko et al. [2] directly yields the following lemma.

▶ **Lemma 9.** *Let $\mathcal{A} = (Q, \delta, s)$ be a Wheeler NFA, then $\mathcal{A}$ is also a quasi-Wheeler NFA.*

Furthermore, the set of quasi-Wheeler NFAs is strictly larger than the set of Wheeler NFAs as there exist non-Wheeler NFAs $\mathcal{A}$ for which the quotient automaton $\mathcal{A}/_\sim$ is Wheeler, e.g., the NFA and its corresponding quotient NFA in the center and on the right of Figure 1.

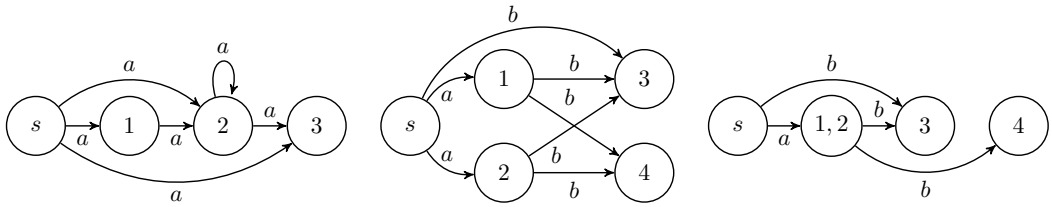

Figure 1 Left: An NFA for which $\mathcal{P} = \langle\{s\}, \{1\}, \{2\}, \{3\}\rangle$ is the coarsest forward-stable partition, yet $S_2 = S_3$. Center: A non-Wheeler NFA (the two states 3 and 4 cannot be ordered) that is quasi-Wheeler as witnessed by the Wheeler preorder $\preceq$ corresponding to the ordered partition $\langle\{s\}, \{1, 2\}, \{3\}, \{4\}\rangle$. Right: Quotient automaton $\mathcal{A}/_\sim$, where $\mathcal{A}$ is the automaton in the middle and $\sim$ is the equivalence relation induced by the Wheeler preorder $\preceq$. Notice that $\mathcal{A}/_\sim$ is Wheeler with the strict total order $\prec$ induced by $\preceq$ on the equivalence classes with respect to $\sim$.

An important advantage of quasi-Wheeler NFAs over classical Wheeler NFAs is that the former can be recognized in polynomial time, which is implicit in the Forward Algorithm of Alanko et al. [2]. Indeed, the Forward Algorithm receives a Wheeler NFA as input and, in $O(|\delta||Q|^2)$ time, outputs a Wheeler order of the quotient automaton $\mathcal{A}/_\sim$, where $\sim$ is the equivalence relation induced by the returned partition. However, this algorithm actually solves a slightly more general problem, since their output partition is guaranteed to correspond to a Wheeler order of $\mathcal{A}/_\sim$ whenever $\mathcal{A}$ is quasi-Wheeler rather than Wheeler. In other words Alanko et al. gave a polynomial time algorithm for the following problem that we call PreOrderWheeler: Given an arbitrary NFA $\mathcal{A} = (Q, \delta, s)$, the task is to compute a total preorder $\preceq$ of Q with the property that $\preceq$ is a Wheeler preorder if $\mathcal{A}$ is quasi-Wheeler. Their polynomial time algorithm for PreOrderWheeler, the *Forward Algorithm*, yields a polynomial-time recognition algorithm for quasi-Wheeler NFAs as follows: Given an arbitrary input NFA $\mathcal{A}$, run the Forward Algorithm to compute the Wheeler preorder $\preceq$ given by an ordering of the forward-stable partition. The output partition is guaranteed to be the coarsest forward-stable partition. Compute the quotient automaton with respect to this partition and check if it is Wheeler (in polynomial time by [1, Lemma 3.1]) when endowed with the induced strict total order $\prec$.

3 Partition Refinement for Wheeler Preorders of NFAs

In this section we provide an algorithm that solves PreOrderWheeler in $O(|\delta| \log |Q|)$ time. Recall that the Forward Algorithm of Alanko et al. [2] has running time $O(|\delta| \cdot |Q|^2)$. We achieve the nearly-linear time complexity by using the partition refinement framework of Paige and Tarjan [21]. It is in fact clear that this framework can be used to compute a forward-stable partition. Our notion of forward-stability corresponds to stability with respect to $|\Sigma|$ relations, defined as $E_a := \{(u, v) \in Q^2 : u \in \delta_a(v)\}$ for $a \in \Sigma$, in their terminology [21, Section 3]. Our main contribution here is to extend the framework so as to compute an *ordered* partition (and thus a preorder), while maintaining the nearly-linear running time.

We proceed with a description of the partition refinement algorithm by Paige and Tarjan. The algorithm maintains two partitions $\mathcal{P}$ and $\mathcal{X}$, here $\mathcal{P}$ is an input partition to be refined and $\mathcal{X}$ is such that (1) $\mathcal{P}$ is a refinement of $\mathcal{X}$ and (2) every part of $\mathcal{P}$ is stable with respect to every part of $\mathcal{X}$. Initially, $\mathcal{X}$ is the partition with a single part containing all elements. Until $\mathcal{P}$ becomes stable, i.e., every part of $\mathcal{P}$ is stable with respect to every part of $\mathcal{P}$, which is witnessed by the fact that $\mathcal{P} = \mathcal{X}$, the algorithm iteratively chooses a *compound part* S from $\mathcal{X}$, i.e., a part form $\mathcal{X}$ that consists of multiple parts in $\mathcal{P}$. Then a "splitter" B is chosen as one of the parts in $\mathcal{P}$ contained in S and every part in $\mathcal{P}$ is refined using B by doing a so-called "three-way split" that we detail in the next section. The idea behind the three-way split is essentially to make $\mathcal{P}$ stable with respect to B and $S \setminus B$ at the same time. In fact, the choice of B in the algorithm of Paige and Tarjan is crucial. It is fundamental to always choose a block B such that $|B| \leq |S|/2$. This is the essential property that yields the nearly linear running time using the observation that every element is contained in at most logarithmically many splitters. This property on the size of B is achieved in Paige and Tarjan's implementation by inspecting the first two elements of the list of all parts from $\mathcal{P}$ contained in S and then choosing B to be the smaller one (in size).

As it turns out, this choice of B however interferes with maintaining the order of the parts such as to satisfy the properties of the Wheeler preorder. We solve this issue by choosing the smaller (in size) among the first and last block (in the sense of the ordered partition) contained in the first compound block (rather than the smaller out of the first two blocks, as in Paige and Tarjan's algorithm). This choice both guarantees that $|B| \leq |S|/2$ and enables us to maintain a consistent ordering between the parts resulting from a split step.

Algorithm. We proceed with a description of our algorithm. A pseudo-code formulation can be found in Algorithm 1. First, note that an input-consistent NFA has a natural partition of its states into the $|\Sigma|+1$ parts $\{Q_a\}_{a \in \Sigma \cup \{\varepsilon\}}$, where $Q_a := \{v \in Q : \lambda(v) = a\}$ for $a \in \Sigma \cup \{\varepsilon\}$. Property (1) of Wheeler orders already defines the ordering of these parts: any Wheeler preorder of the NFA's states has to satisfy that u precedes v if u's in-coming letter is smaller than v's. Hence, the ordering of the $|\Sigma|+1$ parts has to be $Q_\varepsilon, Q_{a_1}, \ldots Q_{a_k}$, where we assume that $\Sigma = \{a_1, \ldots, a_k\}$ with $a_1 < \ldots < a_k$. Following this observation, the ordered partition $\mathcal{P}$ in the algorithm is initialized to $\mathcal{P} := \langle Q_\varepsilon, Q_{a_1}, \ldots Q_{a_k} \rangle$.

As in the original partition refinement framework, the algorithm then keeps splitting the parts of the partition using so called *splitters* which are themselves parts in the partition. By splitting, we mean the operation of making all parts of the partition forward-stable with respect to the splitter. As in Paige and Tarjan's algorithm, to maintain the set of splitters, the algorithm also maintains another ordered partition $\mathcal{X}$ such that $\mathcal{P}$ is a refinement of $\mathcal{X}$. Initially $\mathcal{X}$ has a unique part that is equal to Q and the algorithm will maintain the invariant that $\mathcal{P}$ is forward-stable with respect to each part from $\mathcal{X}$. As before, we call the

Algorithm 1 Ordered Partition Refinement.

Input : NFA $\mathcal{A} = (Q, \delta, s)$
Output : Ordered Partition $\mathcal{P}$ of Q that corresponds to Wheeler preorder if and only if $\mathcal{A}$ is quasi-Wheeler

`// * initialization * //`
$\mathcal{P} := \langle Q_\varepsilon, Q_{a_1}, ..., Q_{a_k} \rangle$, $\mathcal{X} := \langle Q \rangle$
while $\mathcal{X} \neq \mathcal{P}$ **do**
  `// * get splitter` B `* //`
  $S :=$ first block in $\mathcal{X}$ consisting of multiple blocks in $\mathcal{P}$
  $B :=$ smaller of first and last block from $\mathcal{P}$ contained in S
    `// * variable` B `contains this block even if split *//`
  `// * update` $\mathcal{X}$ `* //`
  if B *is first block* **then** replace S in $\mathcal{X}$ with $B, S \setminus B$ **else** with $S \setminus B, B$
  `// * split blocks in` $\mathcal{P}$ `using` B `* //`
  for $D \in \mathcal{P}$ **do**
    `// * split` D `* //`
    $D_1 := D \cap \delta_a(B)$, $D_2 := D \setminus D_1$, where $a = \lambda(D)$
    $D_{11} := D_1 \cap \delta_a(S \setminus B)$, $D_{12} := D_1 \setminus D_{11}$
    `// * update` $\mathcal{P}$ `* //`
    if B *first block* **then** replace D in $\mathcal{P}$ with non-empty sets from D_{12}, D_{11}, D_2
    else with non-empty sets from D_2, D_{11}, D_{12}
    `// * continue for behind newly inserted blocks in` $\mathcal{P}$ `* //`
return $\mathcal{P}$

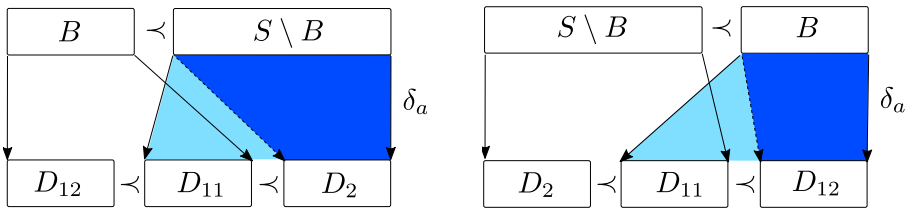

Figure 2 The two cases of our "ordered" three-way split of D into D_2, D_{11} and D_{12} using splitter B contained in the compound part S. Here, D_{11} is the part of D that is reached both by B and $S \setminus B$, D_{12} is the part of D reached by B but not by $S \setminus B$, and D_2 is the part of D reached by $S \setminus B$ but not by B. The size of the splitter B is at most half of the size of the compound block S containing B. On the left, B is the first part in S (in the ordered partition), on the right it is the last part. On the left, the order of $B, S \setminus B$ gets propagated forward, the resulting order within D is D_{12}, D_{11}, D_2. On the right, the order of $S \setminus B, B$ gets propagated forward, the resulting order within D is D_2, D_{11}, D_{12}. The two shades of blue indicate the pruning of edges explained in Section 4.

parts of $\mathcal{X}$ that contain more than a single part from $\mathcal{X}$ *compound parts*. The algorithm then iteratively takes the first (in the order of the ordered partition) compound block S from $\mathcal{P}$ and defines the splitter B as the smaller (in size) of the first and last (in the order of the ordered partition) block in $\mathcal{P}$ contained in S.

Once the splitter B is defined, the algorithm aims to split each part D of the partition $\mathcal{P}$ (including B itself) using B. As we aim at making the partition forward-stable with respect to B we would want to split D into $D_1 := D \cap \delta_a(B)$ and $D_2 := D \setminus D_1$. To implement the three-way split, we however want to further refine D by $S \setminus B$. Recall that S was a part in $\mathcal{X}$ that contained B and that $\mathcal{P}$ is already forward-stable with respect to S by the invariant. The forward-stability of D with respect to S yields that all states in D_2 are reached by $S \setminus B$ and thus D is decomposed into only three parts $D_{11} := D_1 \cap \delta_a(S \setminus B)$, $D_{12} := D_1 \setminus D_{11}$, and $D_2 := D \setminus D_1$ when splitting both with B and $S \setminus B$. This three-way split can be implemented with work proportional to the number of edges leaving B (rather than B and $S \setminus B$). Together with the choice of B being the smaller out of the first and the last block contained in S, this is the main property that allows to prove the nearly linear running time bound of the algorithm. The order in which the sets D_{11}, D_{12}, and D_2 are put in $\mathcal{P}$ replacing D depends on whether B was the first or the last block in S and is chosen so as to satisfy property 2 of Wheeler orders. Intuitively, the order of B and $S \setminus B$ is propagated forward to D_{11}, D_{12}, and D_2, see Figure 2 for an illustration of this "ordered" three-way split.

The algorithm keeps splitting the parts in $\mathcal{P}$ in this way until $\mathcal{X} = \mathcal{P}$, i.e., until there are no compound blocks left. Stopping at this point is correct by the invariant that the partition $\mathcal{P}$ is forward-stable with respect to each part from $\mathcal{X} = \mathcal{P}$, and, thus, forward-stable for $\mathcal{A}$.

Analysis. The following lemma states that the claimed invariant, that every part of $\mathcal{P}$ is forward-stable with respect to every part of $\mathcal{X}$, holds. This follows immediately from the same property of the framework by Paige and Tarjan (the proof of this lemma can be found in the full version).

▶ **Lemma 10.** *At the beginning of every iteration of the while loop in Algorithm 1, it holds that every part of the ordered partition $\mathcal{P}$ is forward-stable with respect to every part of $\mathcal{X}$.*

We will now prove the core technical result of this section: The ordered partition refinement algorithm in fact computes a partition that corresponds to a Wheeler preorder.

▶ **Lemma 11.** *Assume that Algorithm 1 is called on a quasi-Wheeler NFA $\mathcal{A} = (Q, s, \delta)$. Then, at any step of the algorithm, the partition $\mathcal{P} = \langle Q_1, \dots, Q_k \rangle$ agrees with every Wheeler order $\prec$ of the quotient automaton $\mathcal{A}' := \mathcal{A}/_{\mathcal{P}'}$, where $\mathcal{P}'$ denotes the coarsest forward-stable partition for $\mathcal{A}$. That is, if $i < j, u \in P_i, v \in P_j$ then $u \prec v$ for every Wheeler order $\prec$ of $\mathcal{A}'$.*

Proof. The initial partition $\mathcal{P} = \langle Q_\varepsilon, Q_{a_1}, \dots, Q_{a_k} \rangle$ agrees with any Wheeler order $\prec$ of $\mathcal{A}'$. We will show by induction over the number of steps of the while loop that this is the case in any step of the algorithm. Assume that this is true before some intermediate iteration and let us denote with $\mathcal{P}$ the ordered partition at that point. Now, let S be the compound block and let B be the splitter chosen in that iteration. Let us call $\mathcal{P}'$ the ordered partition after refining $\mathcal{P}$ with B and $S \setminus B$, i.e., at the end of that iteration. We will prove that $\mathcal{P}'$ agrees with any Wheeler order $\prec$. As in the algorithm, for $D \in \mathcal{P}$, assume that $a = \lambda(D)$ and let $D_1 = D \cap \delta_a(B)$, $D_2 = D \setminus D_1$, and $D_{11} = D_1 \cap \delta_a(S \setminus B)$ and $D_{12} = D_1 \setminus D_{11}$. Now let $u, v \in D$. If u and v are contained in the same set out of the three sets D_{12}, D_{11}, D_2, nothing is to be shown. Hence, assume that u and v are in two different sets out of the three sets. There are three cases: (1) $u \in D_{11}$ and $v \in D_2$, (2) $u \in D_{12}$ and $v \in D_2$, and (3) $u \in D_{12}$ and $v \in D_{11}$. Notice first that in all three cases, there exists $x \in B$ with $u \in \delta_a(x)$. Now, recall that D is stable with respect to S due to Lemma 10 and thus there exists $y \in S \setminus B$ with $v \in \delta_a(y)$ in all three cases as well. Let us now first assume that B is the first block in S in line 5. Then, as B precedes all blocks in $S \setminus B$ in $\mathcal{P}$ and $\mathcal{P}$ is consistent with any Wheeler

order of $\mathcal{A}'$, we have that $x \prec y$ for any Wheeler order of $\mathcal{A}'$. Then, property 2 of Wheeler orders implies that $u \prec v$ for any Wheeler order of $\mathcal{A}'$. In summary, as the algorithm replaces D by D_{12}, D_{11}, D_2 in this case, we deduce that $\mathcal{P}'$ again agrees with each Wheeler order of $\mathcal{A}'$. If, instead B is the last block in S in line 5, then B succeeds all blocks in $S \setminus B$ in $\mathcal{P}$ and thus we can analogously deduce $v \prec u$ from property 2 of Wheeler orders and as the algorithm replaces D by D_2, D_{11}, D_{12}, we deduce that $\mathcal{P}'$ agrees with every Wheeler order of $\mathcal{A}'$ also in this case. ◀

It remains to argue that the ordered partition refinement algorithm in fact runs in the claimed running time bound of $O(|\delta| \log |Q|)$. The two main ingredients are as follows: (1) We always choose the smaller out of the first and the last part contained in S as B and thus $|B| \leq |S|/2$. As a result every state is in at most logarithmically many splitters. (2) We can implement the algorithm in such a way that the work done in a refinement step with splitter B is proportional to the size of B and the number of out-going transitions from B. This can be argued completely analogously as done by Paige and Tarjan. We prove our algorithm running time and describe the main data structure details in the full version. In summary, we obtain the following theorem.

▶ **Theorem 1.** *For an NFA $\mathcal{A} = (Q, \delta, s)$, we can compute a total preorder $\preceq$ in $O(|\delta| \log |Q|)$ time such that $\preceq$ corresponds to a Wheeler preorder if and only if $\mathcal{A}$ is quasi-Wheeler.*

4 Partition Refinement for Width-Optimal Co-lex Orders of DFAs

Cotumaccio and Prezza [8] propose a way of sidestepping the fact that there may not be a Wheeler order for a given NFA. They show how to index general NFAs using *partial orders*. For this purpose, they define co-lex orders for NFAs as follows.

▶ **Definition 12** (Co-lex Order). *Let $\mathcal{A} = (Q, \delta, s)$ be an NFA. A* co-lex order *for $\mathcal{A}$ is a strict partial order $\prec$ of Q such that the source state precedes all other states, i.e., $s \prec v$ for all $v \in Q \setminus \{s\}$, and, for any pair $v \in \delta_a(u)$ and $v' \in \delta_{a'}(u')$:*
(1) *If $a < a'$, then $v \prec v'$.*
(2) *If $a = a'$, $v \prec v'$, and $u \neq u'$, then $u \prec u'$.*

The *width* of a strict partial order $\prec$ is defined as the size of the largest antichain, i.e. set of pairwise *incomparable* states, where two distinct states $u, v \in Q$ are said to be incomparable if neither $u \prec v$ nor $v \prec u$ holds. Finding a smallest-width co-lex order of $\mathcal{A}$ has been of particular interest for constructing efficient indexes for pattern matching on automata [8, 6, 7, 17]. We provide more context on co-lex orders in the full version. For DFAs, the following order is known to be the smallest-width co-lex order [8].

▶ **Definition 13.** *Let $\mathcal{A} = (Q, \delta, s)$ be a DFA. The relation $\prec_{\mathcal{A}}$ over Q is defined as follows. For $u, v \in Q$, $u \prec_{\mathcal{A}} v$ holds if and only if $\alpha < \beta$ for all $\alpha \in S_u$ and $\beta \in S_v$.*

In the above definition, the alphabet order $<$ over Σ is extended to the co-lexicographical order on strings. For the purpose of the co-lex order $\prec_{\mathcal{A}}$, a state $v \in Q$ can thus be represented solely using upper and lower bounds on S_v. In particular, for a state $v \in Q$, let $\inf S_v$ and $\sup S_v$ be the greatest lower bound (infimum) and the least upper bound (supremum) of S_v, respectively, which are possibly left-infinite strings. We refer the reader to the full version for a formal definition of these concepts. As shown by Kim et al. [17], the co-lex order $\prec_{\mathcal{A}}$ can be characterized using the co-lex order of $\mathrm{IS}_{\mathcal{A}} := \{\inf S_v : v \in Q\} \cup \{\sup S_v : v \in Q\}$.

▶ **Lemma 14** ([17, Theorem 10]). *Let $\mathcal{A} = (Q, \delta, s)$ be a DFA. Then, for any two distinct states $u, v \in Q$, $u \prec_{\mathcal{A}} v$ holds if and only if $\sup S_u \leq \inf S_v$.*

We note that, for the purpose of indexing, we also need a minimum chain partition of $\prec_{\mathcal{A}}$, i.e., a minimum-size partition of Q such that the states in each part are totally ordered under $\prec_{\mathcal{A}}$. As shown by Kim et al. [17], such a chain partition can be computed in linear time via a greedy algorithm for the interval graph coloring problem, provided that the set $\mathrm{IS}_{\mathcal{A}}$ (implicitly defining an interval in co-lex order for each state) has been computed and sorted.

In this section, we give an algorithm for co-lex sorting the set $\mathrm{IS}_{\mathcal{A}}$ that runs in nearly-linear time $O(|\delta| \log |Q|)$ on any input-consistent DFA. This is a significant improvement with respect to the best previously-known algorithms that run in $O(|\delta| \cdot |Q|)$ and $O(|Q|^2 \log |Q|)$ time [17].

Outline. Given an input DFA $\mathcal{A} = (Q, \delta, s)$, we first compute a pruned automaton $\mathcal{A}^{\inf} = (Q, \delta^{\inf}, s)$ that *encodes* $\{\inf S_v : v \in Q\}$ in a sense that walking back starting at a state $v \in Q$ on the pruned transition $\delta^{\inf}$ yields its infimum string $\inf S_v$. Similarly, we compute a pruned automaton $\mathcal{A}^{\sup} = (Q, \delta^{\sup}, s)$ that encodes the supremum strings. As we will see, computing these sets takes $O(|\delta| \log |Q|)$ time each. On these two pruned automata, we compute the co-lex order of $\mathrm{IS}_{\mathcal{A}}$ in $O(|Q| \log |Q|)$ time in a similar way as done by Kim et al. [17], from which the smallest-width co-lex order of $\mathcal{A}$ (with a minimum chain partition) can be computed in linear time.

Pruning Algorithm. The algorithm is identical to Algorithm 1 with the only difference that we insert an additional pruning step, Algorithm 2, before Line 8 of the algorithm. See Figure 2 for an illustration: If a state in D is reached from both B and $S \setminus B$, we only keep the edges from the smaller (in the sense of the ordered partition) of the two. This corresponds to changing δ_a such that the blue portion of the figure shrinks to the dark blue one.

Algorithm 2 Pruning Step for Algorithm 1 inserted before Line 8.

```
// * prune A * //
```
if *B is the first block* **then** delete transitions from $S \setminus B$ to D_1 **else** from B to D_1.

We inductively show that Algorithm 2 outputs a pruned automaton encoding the set of infima $I := \{\inf S_v : v \in Q\}$. For an integer $i \geq 1$, let $\delta^{(i)}$ be the pruned transition at the end of the i-th iteration of the while loop. Let us analogously denote with $\mathcal{X}^{(i)}$ and $\mathcal{P}^{(i)}$ the state of the ordered partitions $\mathcal{X}$ and $\mathcal{P}$ in the algorithm at the end of the i-th iteration. For convenience, we also define $\delta^{(0)} = \delta$, $\mathcal{X}^{(0)} = \langle Q \rangle$, and $\mathcal{P}^{(0)} = \langle Q_\varepsilon, Q_{a_1}, \ldots, Q_{a_k} \rangle$. Lemma 10 and the definition of the pruning step yield the following invariant on forward-stability.

▶ **Observation 15.** *At the end of every iteration i of the while loop in Algorithm 1 when run together with the pruning step, it holds that every part of the ordered partition $\mathcal{P}^{(i)}$ is forward-stable with respect to every part in $\mathcal{X}^{(i)}$ in the automaton $\mathcal{A}^{(i)} = (Q, \delta^{(i)}, s)$.*

Intuitively speaking, the pruning step removes transitions that do not originate from the co-lexicographically smallest part in the sorted partition. Hence, we obtain the following lemma (the proof is deferred to the full version).

▶ **Lemma 16.** *Let $v \in Q$ be a state with $a = \lambda(v)$ and let $A, A' \in \mathcal{X}^{(i)}$ for some $i \geq 0$ be such that $v \in \delta_a^{(i)}(A) \cap \delta_a(A')$. Then either (i) $A = A'$ or (ii) A precedes A' in $\mathcal{X}^{(i)}$.*

Let $\mathcal{P}^*$ and δ^* be the ordered partition and the pruned transition, respectively, obtained at the end of the algorithm's execution. From Lemma 16, we can see that for every state $v \in Q \setminus \{s\}$, there exists a unique part $A \in \mathcal{P}^*$ such that $v \in \delta^*_{\lambda(v)}(A)$. Combining this with Observation 15, we obtain that a unique string can be obtained by walking backwards through the pruned transitions δ^*. For $u \in Q$, let α^*_u be the longest (or possibly left-infinite) string that can be obtained in this way starting from u. Since every transition comes from a co-lex smallest part, we can obtain the following lemma (the proof is included in the full version).

▶ **Lemma 17.** *For every $u \in Q$, $\alpha^*_u = \inf S_u$.*

It is worth noting that some states possibly have more than one in-going transitions after the termination of the algorithm. Nevertheless, Lemma 17 still holds when we choose any of them and remove the others. From this observation, we can assume that every state (except the source state) in the pruned automaton $\mathcal{A}^{\inf}$ has exactly one in-going transition.

Similarly, we can compute the pruned automaton for the set of suprema $\{\sup S_v : v \in Q\}$. The only difference is that we start with the partition $\mathcal{P}'^{(0)} = \langle Q_{a_k}, \ldots, Q_{a_1}, Q_\varepsilon \rangle$ with the reversed order. It is simple to see the greatest string can be computed with this setting.

Regarding the time complexity, notice that the partition refinement algorithm iterates through the same partitions as if it were run on $\mathcal{A}^{\inf}$ in the first place. Additional time is taken for the pruning step for deleting transitions. This work can be done in $O(1)$ time per transition. Once a transition is deleted, it will never be considered in the rest of the execution and hence the additional work amortizes to $O(|\delta|)$. Consequently, the asymptotic time complexity of the partition refinement algorithm with pruning remains $O(|\delta| \log |Q|)$.

Computing Co-Lex Order of $\mathrm{IS}_{\mathcal{A}}$. Once the two pruned automata $\mathcal{A}^{\inf}$ and $\mathcal{A}^{\sup}$ are obtained, we can easily compute the co-lex order of $\mathrm{IS}_{\mathcal{A}}$ in $O(|Q| \log |Q|)$ time using the suffix doubling algorithm [17], which extends the well-known prefix-doubling algorithm [18]. Instead of accessing via integer indexes for the doubling procedure, each state keeps a pointer referring to another (possibly the same) state that is 2^k hops away along its backward walk. We describe all details of this algorithm in the full version. In summary, we conclude this section with the following theorem.

▶ **Theorem 2.** *For an input-consistent DFA $\mathcal{A} = (Q, \delta, s)$, we can compute a minimum chain partition of the smallest-width co-lex order in $O(|\delta| \log |Q|)$ time.*

5 Experimental results

We implemented our partition refinement algorithms in `C++` and made it available at `https://github.com/regindex/finite-automata-partition-refinement.git`. We compared the algorithm from Theorem 1 to the only (to the best of our knowledge) other available tool for computing a sorted partition consistent with any Wheeler order of the input NFA, the renaming heuristic from `WGT` [4]. For a fair comparison, we only run the first part of the `WGT` recognizer and remove the exponential time search that is used to subsequently compute a Wheeler order of the states ending up in equivalence classes.[2] We used the Wheeler graph generator included in `WGT` to generate 7 random input Wheeler NFAs with

[2] As observed empirically, the partition computed by `WGT` is in some cases coarser than the partition computed by our algorithm and is, thus, not necessarily forward-stable.

$|Q| \in \{5^6 \cdot 2^i : i = 0, \ldots, 6\}$ and $|\delta| = 3|Q|$ (we cannot generate much denser graphs since for Wheeler graphs $\delta = O(|\Sigma| \cdot |Q|)$ and here $|\Sigma| = 5$). Our experiments were run on a server with Intel(R) Xeon(R) W-2245 CPU @ 3.90GHz with 8 cores and 128 gigabytes of RAM running Ubuntu 18.04 LTS 64-bit.

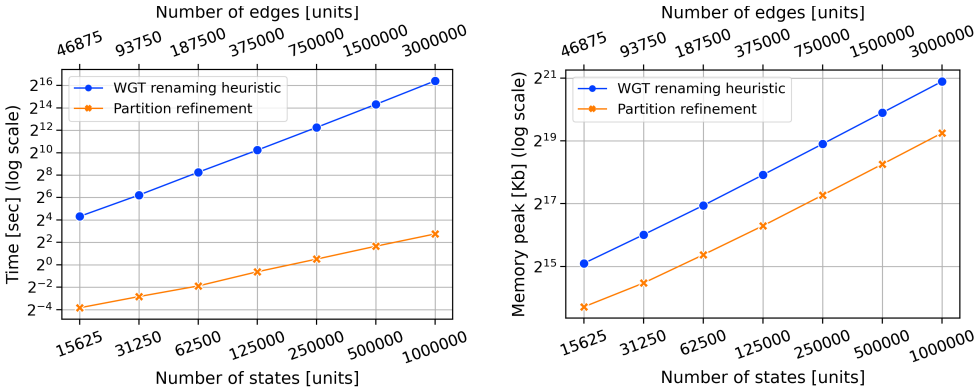

Figure 3 CPU time (left) and memory peak (right) for sorting seven Wheeler NFAs using our partition refinement algorithm and the renaming heuristic contained in the `WGT` recognizer software. Datasets generated using `WGT` specifying seven different combinations of number of states and edges.

Figure 3 shows the running time and peak memory consumption of both implementations. As expected, our partition refinement implementation shows a slight super-linear behavior confirming the $O(|\delta| \cdot \log |Q|)$ worst-case running time of the algorithm. On the other hand, `WGT` shows a quadratic behavior, which is better than the cubic $O(|\delta| \cdot |Q|^2)$ bound for the Forward Algorithm of Alanko et. al [2], but between $\approx 100\times$ to $\approx 10000\times$ slower than our implementation. In terms of peak memory, both implementations behave linearly with respect to the automaton's size with a $3\times$ advantage in favor of our implementation. On the largest input instance containing one million states, our implementation computes the Wheeler preorder in about seven seconds. In a second experiment, we show that our implementation of the algorithm from Theorem 2 can prune a pangenomic DFA containing over 51 million states and 53 million edges in 355 seconds with a peak memory of 33.2 Gb.

References

1 Jarno Alanko, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Regular languages meet prefix sorting. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 911–930. SIAM, 2020.

2 Jarno Alanko, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Wheeler languages. *Information and Computation*, 281:104820, 2021. doi:10.1016/j.ic.2021.104820.

3 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

4 Kuan-Hao Chao, Pei-Wei Chen, Sanjit A Seshia, and Ben Langmead. WGT: Tools and algorithms for recognizing, visualizing and generating Wheeler graphs. *bioRxiv*, 2022. doi:10.1101/2022.10.15.512390.

5 Computational Pan-Genomics Consortium. Computational pan-genomics: status, promises and challenges. *Briefings in bioinformatics*, 19(1):118–135, 2018.

6 Nicola Cotumaccio. Graphs can be succinctly indexed for pattern matching in $O(|E|^2 + |V|^{5/2})$ time. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *Data Compression Conference, DCC 2022, Snowbird, UT, USA, March 22-25, 2022*, pages 272–281. IEEE, 2022. doi:10.1109/DCC52660.2022.00035.

7 Nicola Cotumaccio, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza. Co-lexicographically ordering automata and regular languages. Part I, 2022. doi:10.48550/arXiv.2208.04931.

8 Nicola Cotumaccio and Nicola Prezza. On indexing and compressing finite automata. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2585–2599. SIAM, 2021. doi:10.1137/1.9781611976465.153.

9 Robert P. Dilworth. A Decomposition Theorem for Partially Ordered Sets. *Annals of Mathematics*, 51(1):161–166, 1950. doi:10.2307/1969503.

10 Massimo Equi, Veli Mäkinen, and Alexandru I. Tomescu. Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless seth fails. In *Proceedings of the 47th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, pages 608–622, 2021. doi:10.1007/978-3-030-67731-2_44.

11 Massimo Equi, Veli Mäkinen, Alexandru I Tomescu, and Roberto Grossi. On the complexity of string matching for graphs. *ACM Transactions on Algorithms*, 19(3):1–25, 2023.

12 Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. Lz77-based self-indexing with faster pattern matching. In Alberto Pardo and Alfredo Viola, editors, *LATIN 2014: Theoretical Informatics*, pages 731–742, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

13 Travis Gagie, Giovanni Manzini, and Jouni Sirén. Wheeler graphs: A framework for BWT-based data structures. *Theoretical computer science*, 698:67–78, 2017.

14 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Optimal-Time Text Indexing in BWT-runs Bounded Space. In *Proceedings of the 2018 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1459–1477, 2018. doi:10.1137/1.9781611975031.96.

15 Daniel Gibney and Sharma V. Thankachan. On the hardness and inapproximability of recognizing wheeler graphs. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 51:1–51:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

16 Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Inf. Comput.*, 86(1):43–68, 1990.

17 Sung-Hwan Kim, Francisco Olivares, and Nicola Prezza. Faster Prefix-Sorting Algorithms for Deterministic Finite Automata. In *34th Annual Symposium on Combinatorial Pattern Matching (CPM 2023)*, pages 16:1–16:16, 2023. doi:10.4230/LIPIcs.CPM.2023.16.

18 Udi Manber and Gene Myers. Suffix Arrays: A New Method for on-Line String Searches. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 319–327, 1990. doi:10.1137/0222058.

19 Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki. Storage and retrieval of highly repetitive sequence collections. *Journal of computational biology : a journal of computational molecular cell biology*, 17:281–308, March 2010. doi:10.1089/cmb.2009.0169.

20 Gonzalo Navarro. *Compact Data Structures – A practical approach*. Cambridge University Press, 2016. ISBN 978-1-107-15238-0. 536 pages.

21 Robert Paige and Robert Endre Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987. doi:10.1137/0216062.

22 Jouni Sirén, Jean Monlong, Xian Chang, Adam M Novak, Jordan M Eizenga, Charles Markello, Jonas A Sibbesen, Glenn Hickey, Pi-Chuan Chang, Andrew Carroll, et al. Pangenomics enables genotyping of known structural variants in 5202 diverse genomes. *Science*, 374(6574):abg8871, 2021.

23 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*, 11(2):375–388, 2014. doi:10.1109/TCBB.2013.2297101.

24 Peter Weiner. Linear pattern matching algorithms. In *Switching and Automata Theory, 1973. SWAT'08. IEEE Conference Record of 14th Annual Symposium on*, pages 1–11. IEEE, 1973.

25 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005.