Linear-time Minimization of Wheeler DFAs

Alanko, Jarno

IEEE

2022

Alanko , J , Cotumaccio , N & Prezza , N 2022 , Linear-time Minimization of Wheeler DFAs . in A Bilgin , MW Marcellin , J Serra-Sagrista & JA Storer (eds) , DCC 2022: 2022 DATA COMPRESSION CONFERENCE (DCC) . IEEE Data Compression Conference , IEEE , pp. 53-62 , Data Compression Conference (DCC) , Snowbird , 22/03/2022 . https://doi.org/10.1109/DCC52660.2022.000

http://hdl.handle.net/10138/349645
https://doi.org/10.1109/DCC52660.2022.00013

Linear-time Minimization of Wheeler DFAs

Jarno Alanko[1,2], Nicola Cotumaccio[3], Nicola Prezza[4]

[1]Dept. of Computer Science, University of Helsinki, Finland, jarno.alanko@helsinki.fi
[2]Faculty of Computer Science, Dalhousie University, Halifax, Canada
[3]GSSI, L'Aquila, Italy, nicola.cotumaccio@gssi.it
[4] DAIS, Ca' Foscari University, Venice, Italy, nicola.prezza@unive.it

Abstract

Wheeler DFAs (WDFAs) are a sub-class of finite-state automata which is playing an important role in the emerging field of *compressed data structures*: as opposed to general automata, WDFAs can be stored in just $\log \sigma + O(1)$ bits per edge, σ being the alphabet's size, and support optimal-time pattern matching queries on the substring closure of the language they recognize. An important step to achieve further compression is minimization. When the input $\mathcal{A}$ is a general deterministic finite-state automaton (DFA), the state-of-the-art is represented by the classic Hopcroft's algorithm, which runs in $O(|\mathcal{A}| \log |\mathcal{A}|)$ time. This algorithm stands at the core of the only existing minimization algorithm for Wheeler DFAs, which inherits its complexity. In this work, we show that the minimum WDFA equivalent to a given input WDFA can be computed in linear $O(|\mathcal{A}|)$ time. When run on de Bruijn WDFAs built from real DNA datasets, an implementation of our algorithm reduces the number of nodes from 14% to 51% at a speed of more than 1 million nodes per second.

Introduction

Minimizing deterministic finite-state automata (DFA) is a classic problem in computer science. The most well-known method solving this problem is the partition refinement algorithm of Hopcroft [1], which runs in time $O(|\mathcal{A}| \log |\mathcal{A}|)$, where $|\mathcal{A}|$ denotes the size of the input automaton. Revuz improved this to linear time for *acyclic* DFAs [2], but no linear-time algoritihm for the general case is known to date.

In this work, we consider the problem of minimizing Wheeler DFAs (WDFA): given a WDFA, compute the minimum equivalent WDFA. This is a class of automata recently introduced by Gagie et al. [3] which is gaining a lot of interest in the field of compressed data structures. A Wheeler automaton with an alphabet of size σ can be represented in just $\log \sigma + O(1)$ bits per edge, and indexed for substring queries with the addition of auxiliary succinct data structures. Practically-relevant classes of labeled graphs that are always Wheeler include disjoint paths (sets of strings) [4], trees [5], and de Bruijn graphs [6].

Wheeler DFAs are popular in indexing genomic databases. For example, the compressed index structure of the variation graph toolkit VG [7] is based on the WDFA of a compacted de Bruijn graph of the input data. Other applications include the colored de Bruijn graph indexes VARI [8] and Themisto [9]. Efficient WDFA minimization has applications in compressing the space of all these data structures. Note that regular DFA minimization algorithms are not suitable for the problem, because the minimum DFA equivalent to a given WDFA is not necessarily Wheeler (and thus it is not compressible/indexable as efficiently).

Related Work

There are elements of WDFA minimization in the variation graph indexing tools GCSA and GCSA2 [10, 11] used in the VG toolkit [7]. Due to historical reasons, in both GCSA and GCSA2 the transitions of the automata travel in the reverse direction compared to the convention in Wheeler graphs. For clarity, in this section we describe GSCA and GSCA2 using the orientation convention in Wheeler graphs.

GCSA turns a determinized acyclic variation graph G into an equivalent Wheeler DFA W by splitting states until the graph satisfies the properties of a Wheeler graph. During the process, states of W with the same incoming path label sets are merged if they correspond to the same original node in G. By construction, this merging preserves the language of the automaton, but it does not merge equivalent states that correspond to distinct nodes in the variation graph.

The GCSA2 data structure builds the WDFA of the order-k de Bruijn graph of the variation graph. A de Bruijn graph (dBg) of a string set is a graph where the nodes represents all substrings of length k ("k-mers") of the input. There is an edge from node u to node v if there is a $(k+1)$-mer in the input that is suffixed by the k-mer of u and prefixed by the k-mer of v. The label of the edge is the last character of v. In GCSA2, the dBg is indexed and minimized similarly to GCSA. As a final step, a more aggressive form of merging is applied — targeting nodes sharing some incoming suffix — such that the nodes map to the same *set* of nodes in the variation graph. This may be seen as reducing the order k of the dBg locally.

A core issue with the minimization in GCSA and GCSA2 is that the merging is not based on the fundamental Myhill-Nerode equivalence relation of states [12]. Instead, they on rely on the mapping to the variation graph to detect equivalent states. This acts as a proxy to Myhill-Nerode equivalence, but the method is more restrictive and may fail to detect all equivalent states.

In 2020, Alanko et al. [13] formalized the WDFA minimization problem using the Myhill-Nerode equivalence relation. They characterized the minimum WDFA of an automaton and gave an algorithm to minimize a WDFA in time $O(|\mathcal{A}| \log |\mathcal{A}|)$, using the classic Hopcroft's DFA minimization algorithm [1] as a subroutine. This algorithm stands as the fastest currently known algorithm for WDFA minimization.

In the special case of de Bruijn WDFAs, there is another minimization technique available. To turn the de Bruijn graph into a WDFA, additional technical dummy nodes need to be added: these represent prefixes of k-mers being themselves prefixes of strings in the input set. Alanko et al. [14] showed how to eliminate redundant dummy nodes from the automaton. This may change the language of the automaton but it does not change the substrings of length k of the language.

Our contributions

In this work, we improve on the WDFA minimization algorithm of Alanko et al. [13], by eliminating the dependency on Hopcroft's algorithm, and instead exploiting the special structure of a Wheeler automaton to minimize it in *linear time*.

Experimentally, we use our WDFA minimization algorithm to show that the order-

k de Bruijn WDFA contains significant redundancies. In particular, we show that on real DNA sequence data, we can compress it by up to 51%, while maintaining the language of the automaton and the Wheeler property.

Notation and Preliminaries

We denote DFAs as quintuples $\mathcal{A} = (Q, \Sigma, \delta, s, F)$, where Q is the set of states, $\Sigma = \{0, \dots, |Q|^{O(1)}\}$ is an alphabet of polynomial size, $\delta : Q \times \Sigma \to Q$ is the transition function, $s \in Q$ is the initial (start) state, and $F \subseteq Q$ is the set of final states.

Given a state of $\mathcal{A}$, we will denote with $final(u)$ the boolean predicate returning `true` if and only if $u \in F$.

Note that a DFA $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ can be interpreted as an edge-labeled graph (Q, E), where $E \subseteq Q \times Q \times \Sigma$ is the set of labeled edges $E = \{(q, q', a) \mid \delta(q, a) = q'\}$. We indicate with $|\mathcal{A}| = |Q| + |E|$ the size of this graph. We denote with $out(u) = \{a \in \Sigma \mid \delta(u, a) \in Q\}$ the set of all labels exiting u.

We do not assume δ to be complete: there could exist pairs $q \in Q$, $a \in \Sigma$ for which $\delta(q, a)$ is not defined. From a graph-theoretic perspective this means that, in the labeled graph (Q, E) underlying $\mathcal{A}$, a state q does not necessarily have one outgoing edge for each alphabet's character. We also assume that all states in $Q \setminus \{s\}$ are reachable from s, that there are no incoming edges in s, and that every state is either final or it allows to reach a final state.

The *regular language recognized by a DFA $\mathcal{A}$*, denoted $\mathcal{L}(\mathcal{A})$, is the set of all strings labeling paths in (Q, E) which connect s with some state in F. More formally, let $\hat{\delta} : Q \times \Sigma^* \to Q$ be the extension of δ defined as $\hat{\delta}(q, \epsilon) = q$, where ϵ is the empty string, and $\hat{\delta}(q, \alpha \cdot a) = \delta(\hat{\delta}(q, \alpha), a)$ for $\alpha \in \Sigma^*$.

Then, $\mathcal{L}(\mathcal{A}) = \{\alpha \in \Sigma^* \mid (\exists q \in F)(\hat{\delta}(s, \alpha) = q)\}$.

We say that an equivalence relation $\sim$ on Q is *right-invariant* if and only if for every $u, v \in Q$ such that $u \sim v$ and for every $a \in \Sigma$, $\delta(u, a)$ is defined if and only if $\delta(v, a)$ is defined, and, if so, it holds $\delta(u, a) \sim \delta(v, a)$. We denote the equivalence class of v with $[v]_\sim$.

Let $\sim$ be a right invariant equivalence relation on a DFA $\mathcal{A}$. The *quotient automaton* is defined as $\mathcal{A}_{/\sim} = (Q_\sim, \Sigma, \delta_\sim, [s]_\sim, F_\sim)$, where $Q_\sim = \{[u]_\sim \mid u \in Q\}$, $\delta_\sim([u]_\sim, a) = [v]_\sim$ if and only if $\delta(u, a) = v$, and $F_\sim = \{[u]_\sim \mid u \in F\}$. A classic result is that, since $\sim$ is right-invariant, then $A_{/\sim}$ is a well-defined DFA such that $\mathcal{L}(\mathcal{A}_{/\sim}) = \mathcal{L}(\mathcal{A})$.

We define a special equivalence relation $\approx_\mathcal{A}$ on Q as follows: $u \approx_\mathcal{A} v$ if and only if for every $\alpha \in \Sigma^*$ we have that $\hat{\delta}(u, \alpha) \in F$ if and only if $\hat{\delta}(v, \alpha) \in F$. In other words, $\approx_\mathcal{A}$ is the state version of the classic *Myhill-Nerode* equivalence relation [12]. Note that $\approx_\mathcal{A}$ is right-invariant. The fundamental *Myhill-Nerode Theorem* [12] states that $\mathcal{A}_{/\approx_\mathcal{A}}$ is the minimum (in the number of states) DFA recognizing $\mathcal{L}(\mathcal{A})$. The minimum DFA $\mathcal{A}_{/\approx_\mathcal{A}}$ can be computed in $O(|\mathcal{A}| \log |\mathcal{A}|)$ time by means of a classic algorithm described by Hopcroft [1].

We assume that elements of Σ are totally-sorted according to the standard integer order, which we denote here by $\preceq$ when referring to elements of Σ.

A *Wheeler DFA* (WDFA for brevity) [3, 13] $\mathcal{A}$ is a DFA for which there exists a *total* order $\leq \subseteq Q \times Q$ (called *Wheeler order*) satisfying the following three axioms (in the following, $u < v$ means $u \leq v$ and $u \neq v$):

(i) $s \leq u$ for every $u \in Q$.

(ii) If $u' = \delta(u, a)$, $v' = \delta(v, b)$, and $a \prec b$, then $u' < v'$.

(iii) If $u' = \delta(u, a)$, $v' = \delta(v, b)$, $a = b$, and $u < v$, then $u' \leq v'$.

In [13] it was showed that (1) if $\mathcal{A}$ is a WDFA, then a Wheeler order $\leq$ on $\mathcal{A}$ is uniquely determined (that is, $\leq$ is *the* Wheeler order on $\mathcal{A}$), and (2) if $\mathcal{L}$ is a regular language recognized by some WDFA, then a WDFA $\mathcal{A}'$ recognizing $\mathcal{L}$ and having the minimum number of states is unique up to isomorphism (that is, $\mathcal{A}'$ is the *minimum* WDFA recognizing $\mathcal{L}$).

Wheeler axioms imply the *input-consistency* property: if $\delta(q, a) = \delta(q', a')$, then $a = a'$. From a graph-theoretic perspective this means that, in the labeled graph (Q, E) underlying $\mathcal{A}$, all edges entering the same state bear the same label. With $\lambda(u) = a$, for $u \in Q$, we denote the unique $a \in \Sigma$ with $\delta(u', a) = u$ for all predecessors u' of u. For s we use the convention $\lambda(s) = \# \notin \Sigma$, where $\# \prec a$ for all $a \in \Sigma$.

Linear-time Minimization of WDFAs

The Wheeler DFA minimization problem was first addressed by Alanko et al. in [13]:

Problem: WDFA minimization [13]. *Given a WDFA $\mathcal{A}$, compute the smallest (minimum number of states) WDFA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') = \mathcal{L}(\mathcal{A})$.*

We note that it is not important whether or not the input WDFA is sorted (that is, whether or not the Wheeler order $\leq$ is given as a part of the input), since WDFAs can be sorted in linear time [13] (assuming a polynomial integer alphabet, as we do in the present paper).

Alanko et al. in [13] presented an algorithm solving the WDFA minimization problem in $O(|\mathcal{A}| \log |\mathcal{A}|)$ time. The main bottleneck of this algorithm is represented by a call to Hopcroft's algorithm. After the Myhill-Nerode equivalence classes of $\mathcal{A}$ ($[u]_{\approx_{\mathcal{A}}}$, for $u \in Q$) have been computed, the minimum WDFA $\mathcal{A}'$ can be derived in linear time by means of the following lemma:

Lemma 1 (minimum WDFA [13]). *Let $\mathcal{A} = (Q, \Sigma, \delta, s, F)$ be a Wheeler DFA, let $\leq$ be the Wheeler order on $\mathcal{A}$, and write $Q = \{u_1, u_2, \ldots, u_n\}$, with $u_1 < u_2 < \cdots < u_n$. Let $\equiv_{\mathcal{A}}$ be the equivalence relation on Q that puts in the same equivalence classes exactly all states belonging to the maximum runs of states $u_i, u_{i+1}, \ldots, u_{i+t}$ such that: (1) $\lambda(u_i) = \lambda(u_{i+1}) = \cdots = \lambda(u_{i+t})$, and (2) $u_i \approx_{\mathcal{A}} u_{i+1} \approx_{\mathcal{A}} \cdots \approx_{\mathcal{A}} u_{i+t}$. Then, $\equiv_{\mathcal{A}}$ is right-invariant and $\mathcal{A}_{/\equiv_{\mathcal{A}}}$ is the minimum WDFA recognizing $\mathcal{L}(\mathcal{A})$.*

In other words, in order to find the smallest WDFA one needs to identify maximal runs of consecutive states (in Wheeler order) that are Myhill-Nerode equivalent and that are reached by the same label. The bottleneck of a direct implementation of this procedure is the computation of $\approx_{\mathcal{A}}$, which in general takes $O(|\mathcal{A}| \log |\mathcal{A}|)$ time using Hopcroft's algorithm.

Our Algorithm

In this section we present a linear-time algorithm for the WDFA minimization problem. The idea behind our algorithm is to use the characterization of the minimum WDFA provided by Lemma 1 and exploit the following observation: since classes of $\equiv_{\mathcal{A}}$ form intervals in the Wheeler order, it is sufficient to identify *borders* (u_i, u_{i+1}) between these intervals in order to reconstruct $\equiv_{\mathcal{A}}$ (and thus the minimum WDFA). Let $u_1 < u_2 < \cdots < u_n$ be the Wheeler order. The borders between classes of $\equiv_{\mathcal{A}}$ can be found efficiently by exploiting the properties stated in the following two lemmas:

Lemma 2. *For any string α, if $v = \hat{\delta}(u_i, \alpha)$, $v' = \hat{\delta}(u_{i+1}, \alpha)$ are both defined and $v \neq v'$, then $v = u_j$ and $v' = u_{j+1}$ for some $1 \leq j < n$, that is, v and v' are also adjacent in the Wheeler order.*

Proof. Without loss of generality, we can assume that $\alpha = a \in \Sigma$ is a character (the claim will follow by extension). By Wheeler Axiom (iii) and since $v \neq v'$, it must be $v < v'$. Assume, for a contradiction, that $v = u_j$ and $v' = u_{j'}$ with $j' > j + 1$. Let therefore v'' be any node such that $v < v'' < v'$. By Wheeler Axiom (ii), it must be $\lambda(v) = \lambda(v'') = \lambda(v') = a$. Let therefore u' be any a-predecessor of v'': $\delta(u', a) = v''$. Since $\mathcal{A}$ is a DFA and $v \neq v'' \neq v'$, it must be the case that $u_i \neq u' \neq u_{i+1}$. Since we also know that u_i and u_{i+1} are adjacent in Wheeler order, we therefore have two cases: either $u' < u_i$ or $u' > u_{i+1}$. If $u' < u_i$, then by Wheeler Axiom (iii) it must be $v'' < v$, a contradiction. Similarly, if $u' > u_{i+1}$ then by Wheeler Axiom (iii) it must be $v'' > v'$, again a contradiction. □

The second property follows directly from the right-invariance of $\equiv_{\mathcal{A}}$:

Lemma 3. *If $u_j \not\equiv_{\mathcal{A}} u_{j+1}$ (i.e. (u_j, u_{j+1}) is a border), then all pairs (u_i, u_{i+1}) such that $\hat{\delta}(u_i, \alpha) = u_j$ and $\hat{\delta}(u_{i+1}, \alpha) = u_{j+1}$ for some string $\alpha \in \Sigma^*$, are also such that $u_i \not\equiv_{\mathcal{A}} u_{i+1}$ (that is, they are borders).*

These properties naturally suggest a linear-time reachability algorithm for the problem: first, mark all "base-case" borders (below we formalize this notion). Then, mark also all borders (u_i, u_{i+1}) that lead to a marked "base-case" border (u_j, u_{j+1}) through some string α, that is, such that $\hat{\delta}(u_i, \alpha) = u_j$ and $\hat{\delta}(u_{i+1}, \alpha) = u_{j+1}$. Lemma 2 guarantees that for every string α states $\hat{\delta}(u_i, \alpha)$ and $\hat{\delta}(u_{i+1}, \alpha)$, if distinct, are indeed adjacent in Wheeler order.

In order to formalize this reasoning, let us define the *border graph* of a WDFA:

Definition 4 (border graph of a WDFA)**.** *Let $\mathcal{A}$ be a WDFA. The border graph of $\mathcal{A}$ is the (unlabeled) graph $\mathcal{B}(\mathcal{A}) = (B, Z)$ where $B = \{(u_i, u_{i+1}) \mid 1 \leq i < n, \lambda(u_i) = \lambda(u_{i+1})\}$ and $Z = \{((u_i, u_{i+1}), (u_j, u_{j+1})) \in B \times B \mid u_i = \delta(u_j, \lambda(u_i)) \ \wedge \ u_{i+1} = \delta(u_{j+1}, \lambda(u_i))\}$.*

In other words, an edge of $\mathcal{B}(\mathcal{A})$ exists between borders (u_i, u_{i+1}) and (u_j, u_{j+1}) whenever u_i (respectively, u_{i+1}) can be reached by u_j (respectively, u_{j+1}) by an edge labeled $a = \lambda(u_i) = \lambda(u_{i+1})$.

In general, $\mathcal{B}(\mathcal{A})$ may contain cycles. With the next lemma we put a bound to the size of $\mathcal{B}(\mathcal{A})$, by showing that the maximum out-degree in the graph is at most 1.

Lemma 5. *$\mathcal{B}(\mathcal{A})$ has at most $n-1$ edges and $n-1$ vertices, where n is the number of states of $\mathcal{A}$. Moreover, $\mathcal{B}(\mathcal{A})$ can be constructed in $O(|\mathcal{A}|)$ time given $\mathcal{A}$ as input.*

Proof. Clearly, $|B| \leq n-1$ since elements of B are pairs of adjacent (in Wheeler order) states of $\mathcal{A}$.

We now show that for every $(u_i, u_{i+1}) \in B$, there exists at most one $(u_j, u_{j+1}) \in B$ such that $((u_i, u_{i+1}), (u_j, u_{j+1})) \in Z$. Indeed, if $u_r, u_s \in Q$ are any states such that $u_i = \delta(u_r, a)$ and $u_{i+1} = \delta(u_s, a)$, where $a = \lambda(u_i) = \lambda(u_{i+1})$, then from $u_i < u_{i+1}$ and from Wheeler Axiom (iii) it follows $u_r < u_s$ (equality cannot hold, $\mathcal{A}$ being a DFA), and $(u_r, u_s) \in B$ if and only if r is the largest integer such that $u_i = \delta(u_r, a)$, s is the smallest integer such that $u_{i+1} = \delta(u_s, a)$, $s = r+1$ and $\lambda(u_r) = \lambda(u_s)$. In other words, $|Z| \leq |B| \leq n-1$.

Finally, $\mathcal{B}(\mathcal{A})$ can be built in $O(|\mathcal{A}|)$ time as follows. Consider the list $u_1 < \cdots < u_n$ of $\mathcal{A}$'s states, sorted in Wheeler order (sorting WDFAs takes linear time [15]). For each (u_i, u_{i+1}) with $\lambda(u_i) = \lambda(u_{i+1})$ and each letter a labeling a transition leaving u_i, let $v = \delta(u_i, a)$ and $v' = \delta(u_{i+1}, a)$ (note that the outgoing edges of each node can be sorted in linear time by their label to speed up this operation). If both v and v' exist and are distinct, they must indeed be adjacent in Wheeler order by Lemma 2: $v = u_j$ and $v' = u_{j+1}$, for some $1 \leq j < n$. Then, insert in $\mathcal{B}(\mathcal{A})$ an edge $((u_j, u_{j+1}), (u_i, u_{i+1}))$. □

We describe our minimization algorithm as Algorithm 1. In Line 1, we compute in linear time the Wheeler order on $\mathcal{A}$ by using the algorithm of Alanko et al. [13]. In line 2 we compute the border graph $(B, Z) = \mathcal{B}(\mathcal{A})$ of $\mathcal{A}$. This is done in linear time, see Lemma 5. In Lines 3-5, we mark base-case nodes: for every pair of adjacent nodes, if they are not both final/not final, or if their sets of outgoing labels are not equal then they cannot be Myhill-Nerode equivalent (and thus $\equiv_{\mathcal{A}}$-equivalent). This step takes time proportional to the number of edges of $\mathcal{A}$. In Line 6 we perform a linear-time visit of $\mathcal{B}(\mathcal{A})$ starting from the nodes that have been marked in Lines 3-5. During this visit, we mark every visited node. In Lines 7-9 we compute the equivalence classes of $\equiv_{\mathcal{A}}$. In Line 8, the predicate *marked*$((u_i, u_{i+1}))$ returns `true` if and only if $(u_i, u_{i+1}) \in B$ has been marked in the previous lines. Procedure make_equivalent(u_i, u_{i+1}) at Line 9 records that nodes u_i, u_{i+1} belong to the same equivalence class of $\equiv_{\mathcal{A}}$. To conclude, at Line 10 we return the quotient automaton $\mathcal{A}_{/\equiv_{\mathcal{A}}}$ which, by Theorem 6, is the minimum WDFA recognizing $\mathcal{L}(\mathcal{A})$. The WDFA $\mathcal{A}_{/\equiv_{\mathcal{A}}}$ can be computed in linear time by collapsing each equivalence class of $\equiv_{\mathcal{A}}$ (intervals in Wheeler order) into one state and deduplicating equally-labeled edges exiting the same equivalence class.

In Figure 1 we pictorially show how Algorithm 1 minimizes a WDFA.

Theorem 6. *Let $\mathcal{A}$ be a WDFA. Algorithm 1 computes the minimum WDFA recognizing $\mathcal{L}(\mathcal{A})$ in $O(|\mathcal{A}|)$ time.*

Proof. Complexity follows from the algorithm's description: all steps take linear time.

By Lemma 1, our claim will follow if we prove that $u_i \not\equiv_{\mathcal{A}} u_{i+1}$ if and only if either $\lambda(u_i) \neq \lambda(u_{i+1})$ or (u_i, u_{i+1}) is marked in $\mathcal{B}(\mathcal{A})$.

($\Leftarrow$) Suppose that $\lambda(u_i) \neq \lambda(u_{i+1})$. Then, $u_i \not\equiv_{\mathcal{A}} u_{i+1}$ follows immediately by definition of $\equiv_{\mathcal{A}}$. The other case to consider is when (u_i, u_{i+1}) is marked. Then, by

Algorithm 1: minimize($\mathcal{A}$)

```
input  : A WDFA $\mathcal{A}$
output: The minimum WDFA $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}')$
1  $< \leftarrow$ sort($\mathcal{A}$); // Compute Wheeler order $u_1 < \cdots < u_n$ of $\mathcal{A}$
2  $(B, Z) \leftarrow$ border_graph($\mathcal{A}, <$); // Compute border graph $\mathcal{B}(\mathcal{A})$ of $\mathcal{A}$
3  for $(u_i, u_{i+1}) \in B$ do
4      if $\mathit{out}(u_i) \neq \mathit{out}(u_{i+1}) \lor \mathit{final}(u_i) \neq \mathit{final}(u_{i+1})$ then
5          mark($(u_i, u_{i+1})$); // Mark a "base-case" node of $\mathcal{B}(\mathcal{A})$
6  mark_reachable($B, Z$); // Propagate "base-case" marked nodes
7  for $i = 1, \ldots, n-1$ do
8      if $\Big($not marked$\big((u_i, u_{i+1})\big)\Big) \land \lambda(u_i) = \lambda(u_{i+1})$ then
9          make_equivalent($u_i, u_{i+1}$); // Record that $u_i \equiv_{\mathcal{A}} u_{i+1}$
10 return $\mathcal{A}_{/\equiv_{\mathcal{A}}}$; // Compute and return quotient automaton
```

the definition of $\mathcal{B}(\mathcal{A})$ this means that there exists a pair (u_j, u_{j+1}) (possibly, $i = j$) that was marked in Line 5 (in particular, $u_j \not\approx_{\mathcal{A}} u_{j+1}$ and so $u_j \not\equiv_{\mathcal{A}} u_{j+1}$) such that (u_i, u_{i+1}) is reachable from (u_j, u_{j+1}) in $\mathcal{B}(\mathcal{A})$. In turn, by the definition of $\mathcal{B}(\mathcal{A})$ this implies that there exists a string α such that $\hat{\delta}(u_i, \alpha) = u_j$ and $\hat{\delta}(u_{i+1}, \alpha) = u_{j+1}$. By Lemma 3, this implies that $u_i \not\equiv_{\mathcal{A}} u_{i+1}$.

($\Rightarrow$) Conversely, suppose $u_i \not\equiv_{\mathcal{A}} u_{i+1}$. Then, either $\lambda(u_i) \neq \lambda(u_{i+1})$ (and the claim follows), or $\lambda(u_i) = \lambda(u_{i+1})$. In the latter case, by definition of $\equiv_{\mathcal{A}}$ it must be $u_i \not\approx_{\mathcal{A}} u_{i+1}$. Then, there must exist two states v, v' such that $\hat{\delta}(u_i, \alpha) = v$ and $\hat{\delta}(u_{i+1}, \alpha) = v'$ for some string α, with either $\mathit{out}(v) \neq \mathit{out}(v')$ or $\mathit{final}(v) \neq \mathit{final}(v')$ (in particular, $v \neq v'$). Indeed, let α' be a shortest string witnessing that $u_i \not\approx_{\mathcal{A}} u_{i+1}$. If $\hat{\delta}(u_i, \alpha')$ and $\hat{\delta}(u_{i+1}, \alpha')$ are both defined, let $v = \hat{\delta}(u_i, \alpha')$, $v' = \hat{\delta}(u_{i+1}, \alpha')$ and $\alpha = \alpha'$ (in this case $\mathit{final}(v) \neq \mathit{final}(v')$). If exactly one between $\hat{\delta}(u_i, \alpha')$ and $\hat{\delta}(u_{i+1}, \alpha')$ is not defined, then α' is not the empty string, so we can write $\alpha' = \alpha'' a$ with $\alpha'' \in \Sigma^*$ and $a \in \Sigma$, where both $\hat{\delta}(u_i, \alpha'')$ and $\hat{\delta}(u_{i+1}, \alpha'')$ are defined (by the minimality of α'), so let $v = \hat{\delta}(u_i, \alpha'')$, $v' = \hat{\delta}(u_{i+1}, \alpha'')$ and $\alpha = \alpha''$ (in this case $\mathit{out}(v) \neq \mathit{out}(v')$). From Lemma 2, v and v' must be adjacent in Wheeler order, i.e. $v = u_j$ and $v' = u_{j+1}$ for some $1 \leq j < n$. This implies that (i) (u_j, u_{j+1}) is marked in Line 5 and (ii) (u_i, u_{i+1}) is reachable from (u_j, u_{j+1}) in $\mathcal{B}(\mathcal{A})$. Finally, (i) and (ii) imply that (u_i, u_{i+1}) is marked during the visit of $\mathcal{B}(\mathcal{A})$ in Line 6. □

Experimental Results

We implemented our algorithm and made the source available at the repository `github.com/nicolaprezza/dBg-min`. Our tool takes as input a `fasta` or `fastq` dataset, builds the corresponding de Bruijn graph in BOSS format [6], and runs our minimization algorithm on it. The tool also integrates an implementation of

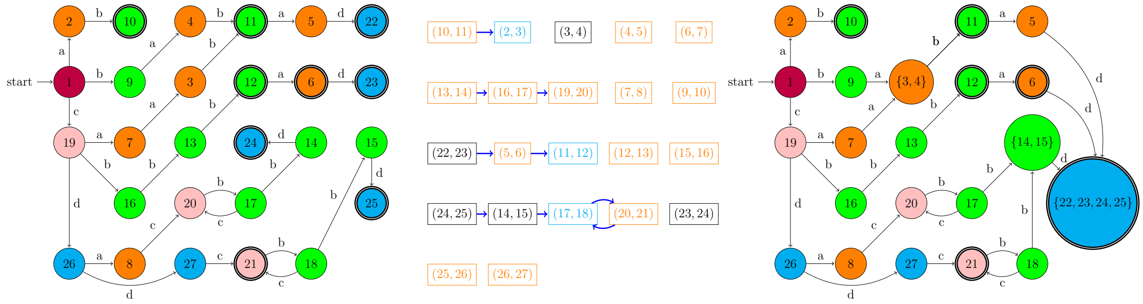

Figure 1: *Left*: a sorted WDFA $\mathcal{A}$ (node labels indicate the Wheeler order). States reached by the same label have the same color. *Center*: the border graph $\mathcal{B}(\mathcal{A})$ built at Line 2 of Algorithm 1. Nodes marked at Line 5 of the algorithm are orange, nodes marked at Line 6 are light blue. *Right*: the minimum WDFA recognizing $\mathcal{L}(\mathcal{A})$. Borders not marked (colored) in $\mathcal{B}(\mathcal{A})$ have been collapsed.

Alanko et al.'s strategy for pruning unnecessary dummy nodes [14] (disabled by default). We tested our implementation on de Bruijn graphs of order $k = 28$ built upon five real DNA datasets, see Table 1: three collections of genomes downloaded from the Pizza&Chili corpus (`pizzachili.dcc.uchile.cl/repcorpus.html` — *Saccharomyces cerevisiae*, *Haemophilus influenzae*, and *Saccharomyces paradoxus*), and short reads sequenced from *Escherichia coli* (`www.ebi.ac.uk/ena/browser/view/ERR022075`) and *Human* (`www.ebi.ac.uk/ena/browser/view/SRX7829390`) genomes. All experiments were run on a workstation with 128 GiB of RAM and a Intel(R) Xeon(R) W-2245 CPU, running Ubuntu Linux and using one thread.

The three `fasta` datasets contain just one sequence and thus the BOSS representation [6] of their de Bruijn graph has at most k dummy nodes. As a result, the pruning algorithm of Alanko et al. [14] does not compress these graphs. The remaining two `fastq` datasets, on the other hand, generate de Bruijn graphs that, as we show in Table 2, are already compressible with Alanko et al.'s algorithm. Table 2 shows the results we obtained by running our WDFA minimization algorithm on the de Bruijn graphs built from the original datasets, and — just on the `fastq` datasets — on the de Bruijn graphs pruned with Alanko et al.'s algorithm.

`dataset`	`seqs`	`bases`
cere.fasta	1	461,286,644
influenza.fasta	1	154,804,605
para.fasta	1	429,265,758
ecoli.fastq	45,440,200	4,589,460,200
human.fastq	63,917,134	6,455,630,534

Table 1: Datasets on which we built the de Bruijn graphs used as inputs for our algorithm. The columns show, respectively: name of the dataset, number of input sequences, and number of DNA bases (i.e. characters on the alphabet $\{A, C, G, T\}$).

Discussion Our implementation proved to be extremely fast, processing over one million nodes per second. The results show that WDFA minimization is indeed a relevant compression strategy for de Bruijn graphs: our algorithm reduced the number

dataset	in ($\times 10^6$)	out ($\times 10^6$)	reduction	time (s)	nodes/s ($\times 10^6$)
cere.fasta	19.004	15.756	17.1%	17	1.118
influenza.fasta	6.469	4.792	25.9%	5	1.294
para.fasta	28.178	22.556	19.9%	26	1.084
ecoli.fastq	449.92	220.47	51%	398	1.130
human.fastq	650.51	438.68	32.6%	600	1.084
ecoli_pruned	317.173	201.940	36%	291	1.089
human_pruned	449.991	387.627	13.8%	431	1.044

Table 2: Performance of our minimization algorithm on the de Bruijn graphs built over the datasets of Table 1 (first four rows) and on the pruned de Bruijn graphs [14] of the two fastq datasets (last two rows). The order of all de Bruijn graphs is $k = 28$. The columns show, respectively: name of the dataset, number of nodes in the input de Bruijn graph, number of nodes in the output (minimized) de Bruijn graph, percentage of nodes removed by the minimization algorithm, running time (construction of the de Bruijn graph in BOSS format is not counted towards the running time), and number of processed nodes per second.

of nodes of the original de Bruijn graphs from 17.1% to 51%, and from 13.8% to 36% when the graph was previously pre-processed with the pruning algorithm of Alanko et al. [14]. Interestingly, these results indicate that, while the two algorithms target mostly different sources of redundancy, minimization targets some dummy nodes as well. The combination of the two algorithms reduced the number of nodes by 55.1% on the `ecoli.fastq` dataset, and by 40.4% on the `human.fastq` dataset. We leave it as an interesting open problem to show whether or not the combination of the two algorithms is optimal, in the sense that it generates the smallest WDFA containing all (and only the) labeled paths of the input de Bruijn graph.

References

[1] John Hopcroft, "An n log n algorithm for minimizing states in a finite automaton," in *Theory of machines and computations*, pp. 189–196. Elsevier, 1971.

[2] Dominique Revuz, "Minimisation of acyclic deterministic automata in linear time," *Theoretical Computer Science*, vol. 92, no. 1, pp. 181–189, 1992.

[3] Travis Gagie, Giovanni Manzini, and Jouni Sirén, "Wheeler graphs: A framework for BWT-based data structures," *Theoretical Computer Science*, vol. 698, pp. 67 – 78, 2017, Algorithms, Strings and Theoretical Approaches in the Big Data Era (In Honor of the 60th Birthday of Professor Raffaele Giancarlo).

[4] Sabrina Mantaci, Antonio Restivo, Giovanna Rosone, and Marinella Sciortino, "An extension of the burrows–wheeler transform," *Theoretical Computer Science*, vol. 387, no. 3, pp. 298–312, 2007.

[5] Paolo Ferragina, Fabrizio Luccio, Giovanni Manzini, and S. Muthukrishnan, "Compressing and indexing labeled trees, with applications," *J. ACM*, vol. 57, no. 1, Nov 2009.

[6] Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya, "Succinct de Bruijn Graphs," in *Algorithms in Bioinformatics*, Ben Raphael and Jijun Tang, Eds., Berlin, Heidelberg, 2012, pp. 225–235, Springer Berlin Heidelberg.

[7] Erik Garrison, Jouni Sirén, Adam M Novak, Glenn Hickey, Jordan M Eizenga, Eric T Dawson, William Jones, Shilpa Garg, Charles Markello, Michael F Lin, et al., "Variation graph toolkit improves read mapping by representing genetic variation in the reference," *Nature biotechnology*, vol. 36, no. 9, pp. 875–879, 2018.

[8] Martin D Muggli, Alexander Bowe, Noelle R Noyes, Paul S Morley, Keith E Belk, Robert Raymond, Travis Gagie, Simon J Puglisi, and Christina Boucher, "Succinct colored de bruijn graphs," *Bioinformatics*, vol. 33, no. 20, pp. 3181–3187, 2017.

[9] Tommi Mäklin, Teemu Kallonen, Jarno Alanko, Ørjan Samuelsen, Kristin Hegstad, Veli Mäkinen, Jukka Corander, Eva Heinz, and Antti Honkela, "Genomic epidemiology with mixed samples," *bioRxiv*, pp. 2020–04, 2021.

[10] 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*, vol. 11, no. 2, pp. 375–388, 2014.

[11] Jouni Sirén, "Indexing variation graphs," in *2017 Proceedings of the ninteenth workshop on algorithm engineering and experiments (ALENEX)*. SIAM, 2017, pp. 13–27.

[12] Anil Nerode, "Linear automaton transformations," *Proceedings of the American Mathematical Society*, vol. 9, no. 4, pp. 541–544, 1958.

[13] Jarno Alanko, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza, "Regular languages meet prefix sorting," in *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, Shuchi Chawla, Ed. 2020, pp. 911–930, SIAM.

[14] Jarno Alanko, Bahar Alipanahi, Jonathen Settle, Christina Boucher, and Travis Gagie, "Buffering updates enables efficient dynamic de bruijn graphs," *Computational and structural biotechnology journal*, vol. 19, pp. 4067—4078, 2021.

[15] Jarno Alanko, Giovanna D'Agostino, Alberto Policriti, and Nicola Prezza, "Wheeler languages," *CoRR*, vol. abs/2002.10303, 2020.