Indexing sets of strings: Wheeler languages

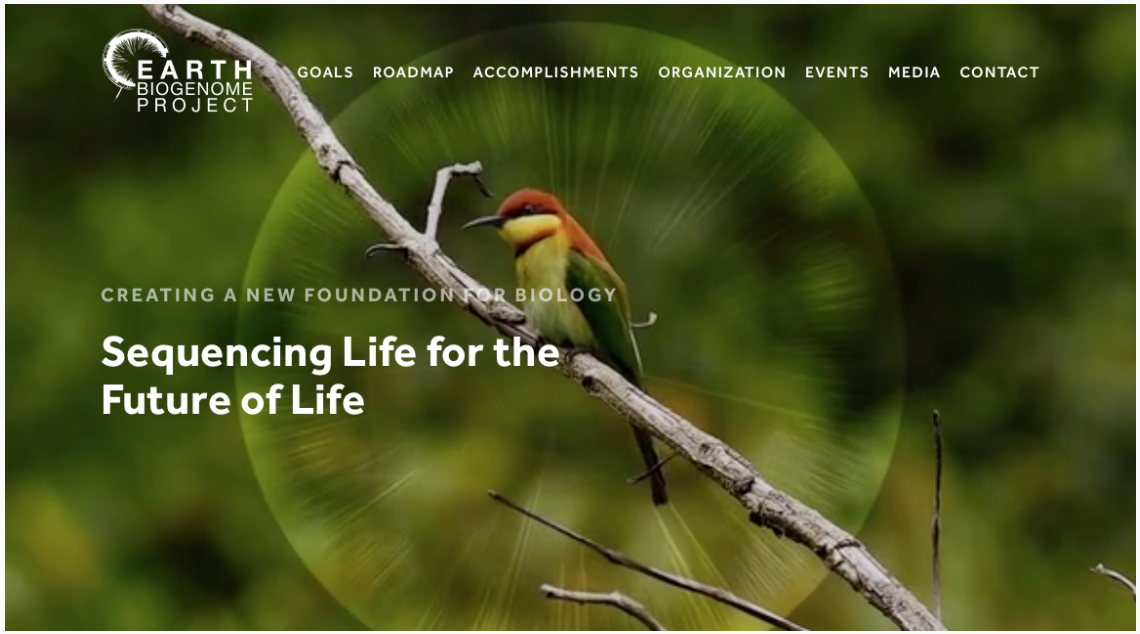
Part I

Alberto Policriti & Giovanna D'Agostino

Dpt. of Mathematics, Computer Science, and Physics, University of Udine.

Why should be care about big data(sets)?

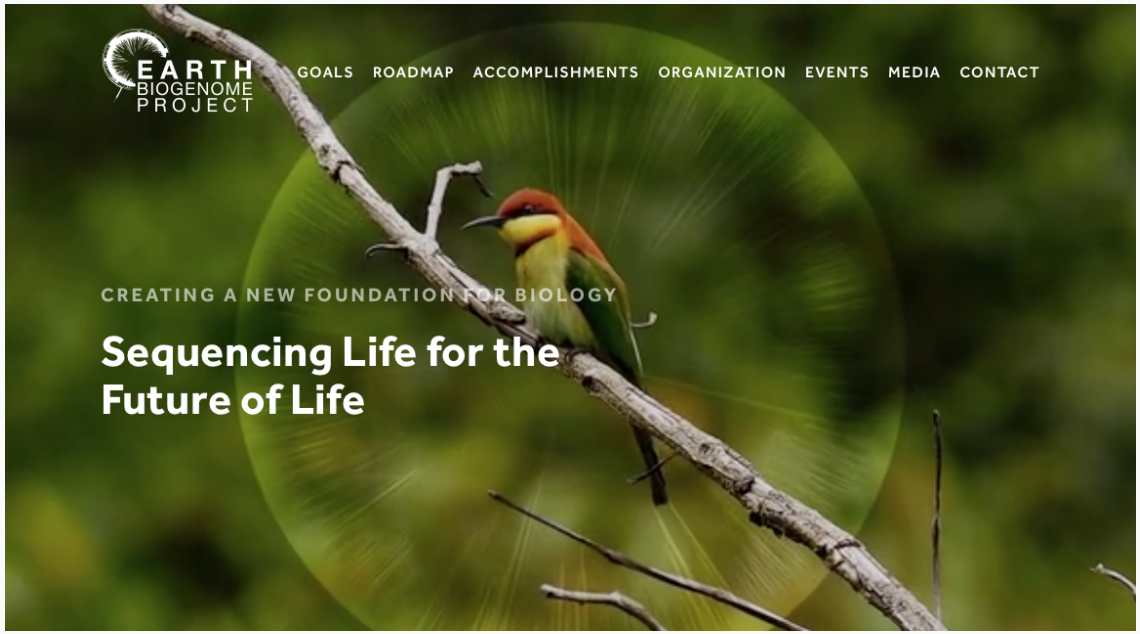
Earth Biogenome Project



Sequencing 1.5 (!) million species

Why should be care about big data(sets)?

Earth Biogenome Project



Sequencing 1.5 (!) million species

Side (?) observation

coverage increases $\Rightarrow$ *repetitiveness* icreases

Why structuring data

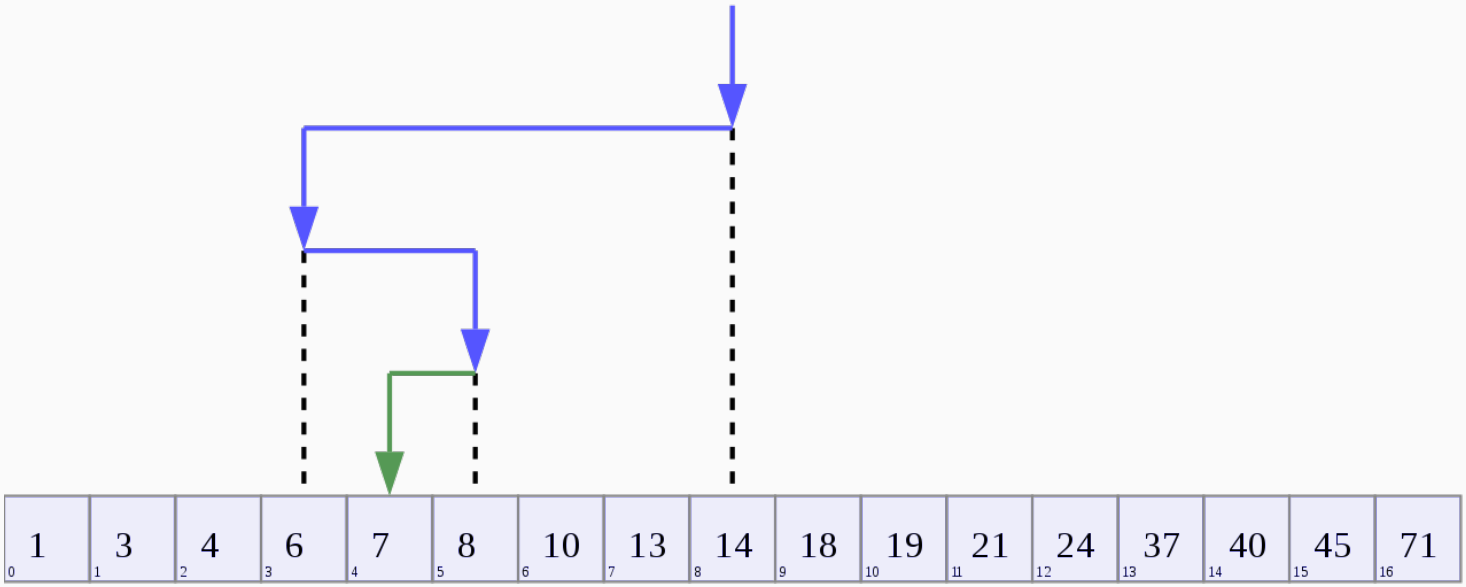
Example

Given a file S containing 1T ($\approx 2^{40}$ numbers) numbers and a number n answer to the question:

$$\underline{\text{is } n \text{ in } S?}$$

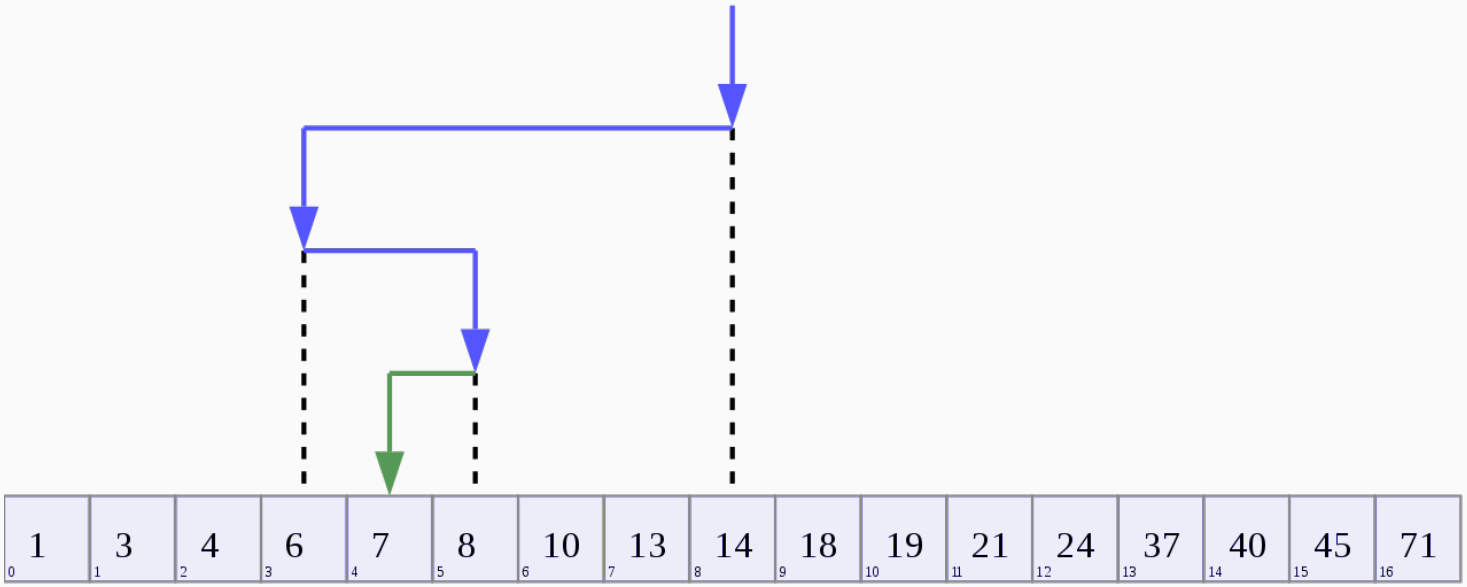
Binary Search

Search for 7:



Binary Search

Search for 7:

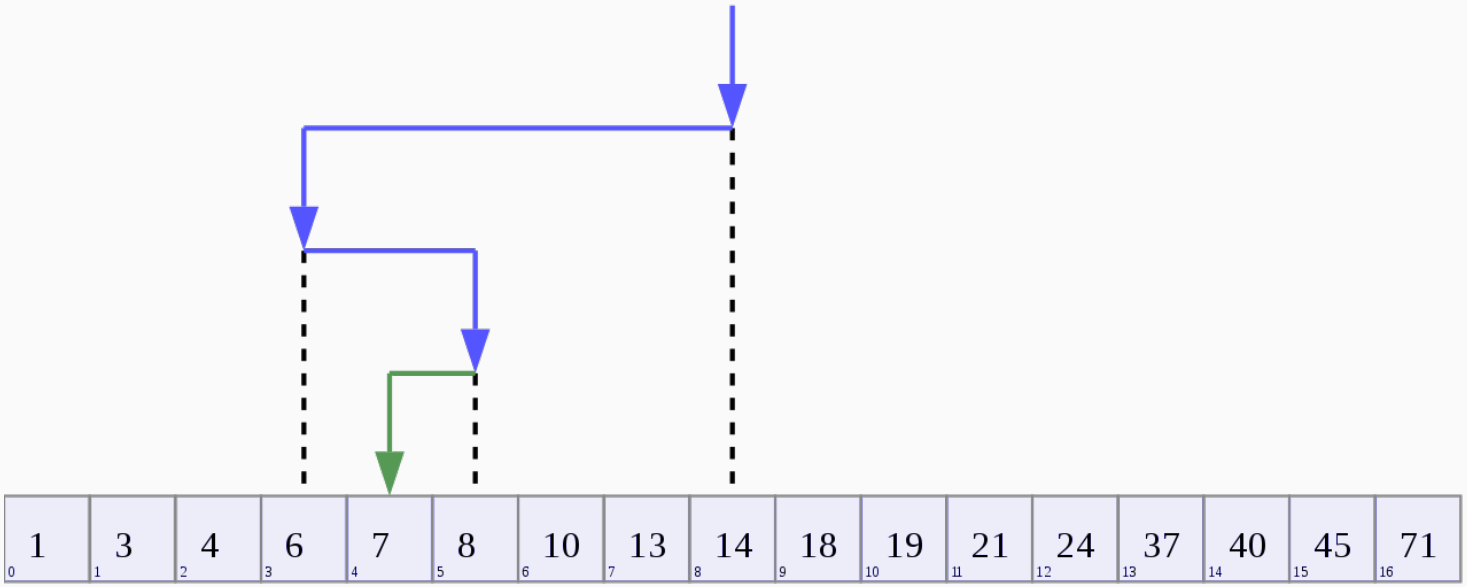


How many questions/steps/instructions?

≈ 40 ... why?

Binary Search

Search for 7:



$$2^{40} = \underbrace{2 \cdot 2 \cdot 2 \cdot \ldots \cdot 2}_{40\text{times}}$$

A different look: codes

Elements can be *turned into* (*encoded by*) binary codes:

indexes

A different look: codes

Elements can be *turned into* (*encoded by*) binary codes:

indexes

What about space?

A different look: codes

Elements can be *turned into* (*encoded by*) binary codes:

indexes

What about space?

Are there better indexes? (Better codes/encodings?)

Learned Indexes

30 Apr 2018

The Case for Learned Index Structures

Tim Kraska*
MIT
Cambridge, MA
kraska@mit.edu

Alex Beutel
Google, Inc.
Mountain View, CA
alexbeutel@google.com

Ed H. Chi
Google, Inc.
Mountain View, CA
edchi@google.com

Jeffrey Dean
Google, Inc.
Mountain View, CA
jeff@google.com

Neoklis Polyzotis
Google, Inc.
Mountain View, CA
npolyzotis@google.com

14 Oct 2019

The PGM-index: a multicriteria, compressed and learned approach to data indexing

Paolo Ferragina
University of Pisa, Italy
paolo.ferragina@unipi.it

Giorgio Vinciguerra
University of Pisa, Italy
giorgio.vinciguerra@phd.unipi.it

Abstract

ABSTRACT

The recent introduction of learned indexes has shaken the foundations of the decades-old field of indexing data structures. Combining, or even replacing, classic design elements such as B-tree nodes with machine learning models has proven to give outstanding improvements in the space footprint and time efficiency of data systems. However, these novel approaches are based on heuristics, thus they formidable results, we still miss proper algorithms and data structures that are flexible enough to work under computational constraints that vary across users, devices and applications, and possibly evolve over time.

In this paper, we restrict our attention to the case of *indexing data structures* for internal or external memory which solve the so-called *fully indexable dictionary* problem. This problem asks to store a multiset S of real keys in order to

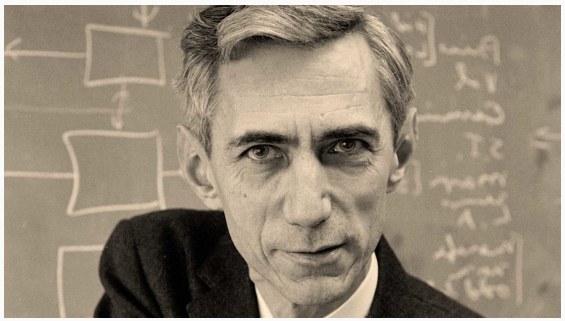
Entropy of S

How difficult is to find an element in S?

$$\mathcal{H}(S) = \log |S|$$

How much *information* is carried by each code (of an element of S)?

BITs and SHANNONs



Let's get real

Probability and entropy of a random variable distribution

Let's get real

Probability and entropy of a random variable distribution

Definition

For X r.v. taking values $x_1, \ldots, x_n$

Let's get real

Probability and entropy of a random variable distribution

Definition

For X r.v. taking values $x_1, \ldots, x_n$

$$\mathcal{H}(X) = \sum_{i=1}^{n} p(x_i) \log(1/p(x_i)$$

$$= E[\log(1/p(x_i)] \text{ ... BITs (SHANNONs)}$$

Let's get real

Probability and entropy of a random variable distribution

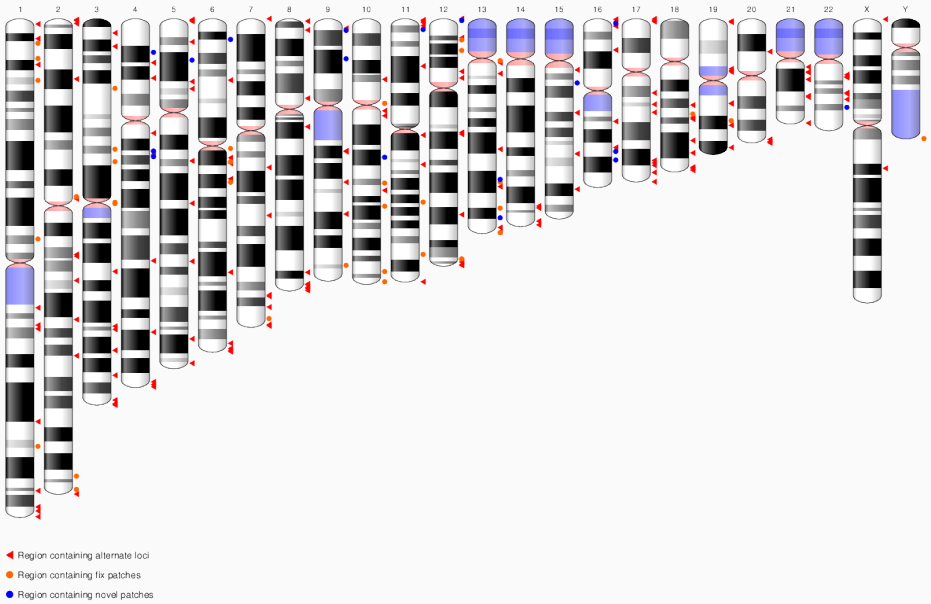
Definition

For X r.v. taking values $x_1, \ldots, x_n$

$$\mathcal{H}(X) = \sum_{i=1}^{n} p(x_i) \log(1/p(x_i)$$
$$= E[\log(1/p(x_i)] \ \ldots \ \text{BITs (SHANNONs)}$$

Then, we cannot encode a sequence S in less than
$H(S) = -\sum_{c \in \Sigma} P(c) \cdot \log_2(P(c))$ bits/symbol .

Entropy of a Genome




```
TTTCCCGTAGGTGAACCTGCGGAAGGATCATTGTCGAATCCTGCGAAGCAGACGACCTGAGAACATGTTA
AACTCACGGGCGTTCGGGACGGGGGAGACCTCGGACCGACGCCCCGAGCACGACGTCGACCGACGGTTGC
TCGTCGTGCATAAAACTAACCCCGGGCGCGGATAAGCGCCAAGGAAAACTATGAAAGGATTGCCTGCCTC
TCGATGCCCCGTACGCGGTGTGCACGGGAGGATCACGGGCGTCTGGAGAAACAAAAACGACTCTCGGCAA
CGGATATCTCGGCTCTCGCATCGATGAAGAACGTAGCGAACTGCGATACTTGGTGTGAATTGCAGAATCC
CGTGAACCATCGAGTCTTTGAACGCAAGTTGCGCCCGAAGCCATTAGGCCGAGGGCACGTCTGCCTGGGC
GTCACGCATCGCGTCGCCCCCAACACATGTCCTCCGTATCGCACGGGTGACCGGGAGGGGGCGGAAACTG
GCTTCCCGTGCTTTCGTGCGGCTGGCCTAAACACGAGTCCCTAGCGATGGACGTTGTGACAAGTGGTGGT
TGATTTCCTCAACTAGAGCGCTGTCACGCGATCCTCCATCGATTGAGGAGACTCCCACGACCCTGATGCA
```

Problems

How do we structure (e.g.) genomes for searching?

- not one string but many strings
- string of characters (not numbers)
- we search for substrings (not single characters)
- ...

Problems

How do we structure (e.g.) genomes for searching?

- not one string but many strings
- string of characters (not numbers)
- we search for substrings (not single characters)
- ...

How do we exploit repetitiveness?

Key point

We ordered data in order to apply binary search.

What about DNA?

Ordering a string (of DNA)?

Suffixes

1	AGGTTGCCAGTGT
2	GGTTGCCAGTGT
3	GTTGCCAGTGT
4	TTGCCAGTGT
5	TGCCAGTGT
6	GCCAGTGT
7	CCAGTGT
8	CAGTGT
9	AGTGT
10	GTGT
11	TGT
12	GT
13	T

1	AGGTTGCCAGTGT
9	AGTGT
8	CAGTGT
7	CCAGTGT
6	GCCAGTGT
2	GGTTGCCAGTGT
12	GT
10	GTGT
3	GTTGCCAGTGT
13	T
5	TGCCAGTGT
11	TGT
4	TTGCCAGTGT

Ordered suffixes (can be compressed): Burrows-Wheeler

AGGTTGCCAGTGT

AGTGT

CAGTGT

CCAGTGT

GCCAGTGT

GGTTGCCAGTGT

GT

GTGT

GTTGCCAGTGT

T

TGCCAGTGT

TGT

TTGCCAGTGT

Ordered suffixes (can be compressed): Burrows-Wheeler

$
AGGTTGCCAGTGT$
AGTGT$
CAGTGT$
CCAGTGT$
GCCAGTGT$
GGTTGCCAGTGT$
GT$
GTGT$
GTTGCCAGTGT$
T$
TGCCAGTGT$
TGT$
TTGCCAGTGT$

Ordered suffixes (can be compressed): Burrows-Wheeler

$AGGTTGCCAGTGT
AGGTTGCCAGTGT$
AGTGT$AGGTTGCC
CAGTGT$AGGTTGC
CCAGTGT$AGGTTG
GCCAGTGT$AGGTT
GGTTGCCAGTGT$A
GT$AGGTTGCCAGT
GTGT$AGGTTGCCA
GTTGCCAGTGT$AG
T$AGGTTGCCAGTG
TGCCAGTGT$AGGT
TGT$AGGTTGCCAG
TTGCCAGTGT$AGG

Ordered suffixes (can be compressed): Burrows-Wheeler

$AGGTTGCCAGTGT

AGGTTGCCAGTGT$

AGTGT$AGGTTGCC

CAGTGT$AGGTTGC

CCAGTGT$AGGTTG

GCCAGTGT$AGGTT

GGTTGCCAGTGT$A

GT$AGGTTGCCAGT

GTGT$AGGTTGCCA

GTTGCCAGTGT$AG

T$AGGTTGCCAGTG

TGCCAGTGT$AGGT

TGT$AGGTTGCCAG

TTGCCAGTGT$AGG

Definition (Burrows-Wheeler Transform)

BWT(AGGTTGCCAGTGT$) = T$CCGTATAGGTGG

Proposition

The BWT of a text can be (easily) inverted and compressed.

Space

How much space are we using?

booster

The BWT is a *booster* for compression

An Analysis of the Burrows-Wheeler Transform

Giovanni Manzini
Dipartimento di Informatica, Università del Piemonte Orientale, Italy.

JACM 2001

run-length encoding

Space

How much space are we using?

booster

The BWT is a *booster* for compression

An Analysis of the Burrows-Wheeler Transform

Giovanni Manzini
Dipartimento di Informatica, Università del Piemonte Orientale, Italy.

JACM 2001

run-length encoding

Space

How much space are we using?

booster

The BWT is a *booster* for compression

An Analysis of the Burrows-Wheeler Transform

Giovanni Manzini
Dipartimento di Informatica, Università del Piemonte Orientale, Italy.

JACM 2001

run-length encoding

Long repetitions?

Searching the $BWT(T)$ ...

... is like searching the suffix array (via the L-F property)

Search

Searching the $BWT(T)$ ...

... is like searching the suffix array (via the L-F property)

The *BWT* *orders* suffixes.

What about graphs?

Theoretical Computer Science 698 (2017) 67–78

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Wheeler graphs: A framework for BWT-based data structures ☆

Travis Gagie [a], Giovanni Manzini [b,c,*], Jouni Sirén [d]

[a] *Diego Portales University and CEBIB, Santiago, Chile*
[b] *University of Eastern Piedmont, Alessandria, Italy*
[c] *IIT-CNR, Pisa, Italy*
[d] *Wellcome Trust Sanger Institute, Cambridge, UK*

What about graphs?

A B S T R A C T

The famous Burrows–Wheeler Transform (BWT) was originally defined for a single string but variations have been developed for sets of strings, labeled trees, de Bruijn graphs, etc. In this paper we propose a framework that includes many of these variations and that we hope will simplify the search for more.

We first define *Wheeler graphs* and show they have a property we call *path coherence*. We show that if the state diagram of a finite-state automaton is a Wheeler graph then, by its path coherence, we can order the nodes such that, for any string, the nodes reachable from the initial state or states by processing that string are consecutive. This means that even if the automaton is non-deterministic, we can still store it compactly and process strings with it quickly.

We then rederive several variations of the BWT by designing straightforward finite-state automata for the relevant problems and showing that their state diagrams are Wheeler graphs.

What about graphs?

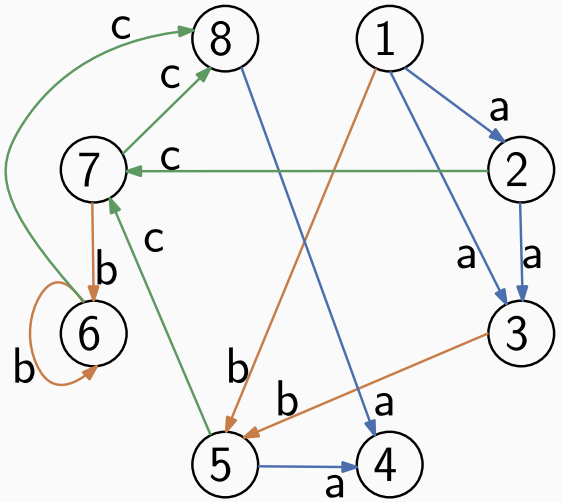
Definition 1. G is a *Wheeler graph* if there is an ordering of the nodes such that nodes with in-degree 0 precede those with positive in-degree and, for any pair of edges $e = (u, v)$ and $e' = (u', v')$ labeled a and a' respectively, the following monotonicity properties hold:

$$\begin{aligned} a \prec a' &\implies v < v' \,, \\ (a = a') \wedge (u < u') &\implies v \leq v' \,. \end{aligned}$$

What about graphs?

Definition 1. G is a *Wheeler graph* if there is an ordering of the nodes such that nodes with in-degree 0 precede those with positive in-degree and, for any pair of edges $e = (u, v)$ and $e' = (u', v')$ labeled a and a' respectively, the following monotonicity properties hold:

$$a \prec a' \implies v < v',$$
$$(a = a') \wedge (u < u') \implies v \leq v'.$$



Co-lexicographic order and path-coherence

Definition 2. G is *path coherent* if there is a total order of the nodes such that for any consecutive range $[i, j]$ of nodes and string α, the nodes reachable from those in $[i, j]$ in $|\alpha|$ steps by following edges whose labels for α when concatenated, themselves form a consecutive range.

Lemma 3. *If G is a Wheeler graph by an ordering π on its nodes then it is path coherent by π.*

- Ordered *sets of* strings

Co-lexicographic order and path-coherence

Definition 2. G is *path coherent* if there is a total order of the nodes such that for any consecutive range $[i, j]$ of nodes and string α, the nodes reachable from those in $[i, j]$ in $|\alpha|$ steps by following edges whose labels for α when concatenated, themselves form a consecutive range.

Lemma 3. *If G is a Wheeler graph by an ordering π on its nodes then it is path coherent by π.*

- Ordered *sets of* strings
- Align to the right

Co-lexicographic order and path-coherence

Definition 2. G is *path coherent* if there is a total order of the nodes such that for any consecutive range $[i, j]$ of nodes and string α, the nodes reachable from those in $[i, j]$ in $|\alpha|$ steps by following edges whose labels for α when concatenated, themselves form a consecutive range.

Lemma 3. *If G is a Wheeler graph by an ordering π on its nodes then it is path coherent by π.*

- Ordered *sets of* strings
- Align to the right
- *Intervals* in the collection of states
- *Intervals* in the collection of strings

Co-lexicographic order and path-coherence

Definition 2. G is *path coherent* if there is a total order of the nodes such that for any consecutive range $[i, j]$ of nodes and string α, the nodes reachable from those in $[i, j]$ in $|\alpha|$ steps by following edges whose labels for α when concatenated, themselves form a consecutive range.

Lemma 3. *If G is a Wheeler graph by an ordering π on its nodes then it is path coherent by π.*

- Ordered *sets of* strings
- Align to the right
- *Intervals* in the collection of states
- *Intervals* in the collection of strings
- (Infinity)

Automata Toolkit

- minimization $\Rightarrow$ many function coarsest partition problem (non-deterministic case: handle with care!)

Automata Toolkit

- minimization $\Rightarrow$ many function coarsest partition problem (non-deterministic case: handle with care!)
- Myhill-Nerorde Theorem $\Rightarrow$ States of the minimum DFA accepting $\mathcal{L}$ are sets strings

Automata Toolkit

- minimization $\Rightarrow$ many function coarsest partition problem (non-deterministic case: handle with care!)
- Myhill-Nerorde Theorem $\Rightarrow$ States of the minimum DFA accepting $\mathcal{L}$ are sets strings
- non-determinism can be eliminated $\Rightarrow$ power-construction (exponential blow-up)

New results

Regular Languages meet Prefix Sorting *

Jarno Alanko[†] Giovanna D'Agostino[‡] Alberto Policriti[§] Nicola Prezza[¶]

New results

1. We show that Wheeler languages are the natural version of regular languages endowed with the co-lexicographic ordering: when sorted, the prefixes of strings belonging to a Wheeler language are partitioned into a *finite* number of *intervals*, each formed by elements from a single Myhill-Nerode equivalence class. In regular languages, those *intervals* are replaced with general *sets*.

New results

2. We show that every Wheeler NFA (WNFA) with n states admits an equivalent Wheeler DFA (WDFA) with at most $2n - 1 - |\Sigma|$ states (Σ being the alphabet) that can be computed in $O(n^3)$ time. This is in sharp contrast with general NFAs (where the blow-up could be exponential).

New results

3. Let d-NFA denote the class of NFAs with at most d equally-labeled edges leaving any state. We show that the problem of recognizing and sorting Wheeler d-NFAs is in P for $d \leq 2$. A recent result from Gibney and Thankachan [12] shows that the problem is NP-complete for $d \geq 5$. Our result almost completes the picture, the remaining open cases being $d = 3$ and $d = 4$.

New results

4. We provide an online incremental algorithm that, when fed with an acyclic Wheeler DFA's nodes in any topological order, can dynamically compute the co-lexicographic rank of each new incoming node among those already processed with just logarithmic delay.

5. We improve the running time of (4) to linear in the offline setting for arbitrary WDFAs.

New results

6. Given a Wheeler DFA $\mathcal{A}$ of size n, we show how to compute, in $O(n \log n)$ time, the smallest Wheeler DFA recognizing the same language as $\mathcal{A}$. If $\mathcal{A}$ is acyclic, running time drops to $O(n)$.

7. Given *any acyclic DFA* $\mathcal{A}$ of size n, we show how to compute, in $O(n + m \log m)$ time, the smallest Wheeler DFA $\mathcal{A}'$, of size m, recognizing the same language as $\mathcal{A}$.

Question

Decidability

Can we decide whether $\mathcal{L}$ is Wheeler?