A New Compression Technique for Repetitive Tries

Ca' Foscari University of Venice
Department of Environmental Sciences, Informatics and Statistics

Master's Thesis Defense

Computer Science and Information Technology
Artificial Intelligence and Data Engineering

6th November 2025

Candidate: Davide Tonetto **Supervisor**: Nicola Prezza **Co-Supervisor**: Alessio Campanelli

Introduction and Motivation

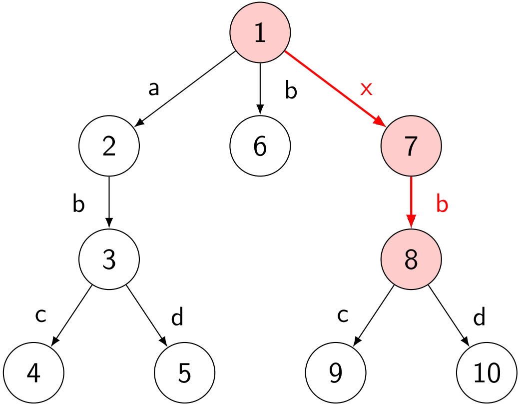


Figure 1: The language is $\{abc, abd, b, xbc, xbd\}$. The path for `xbd` is highlighted in red.

Trie

A trie is a **tree data structure** optimized for **storing strings** and performing **fast prefix-based queries**.

Prefix-Query for 'xb'

1. Start at the root.
2. Follow the edge labeled `x`.
3. From there, follow the edge labeled `b`.
4. There are two strings with prefix `xb`: `xbc` and `xbd`.

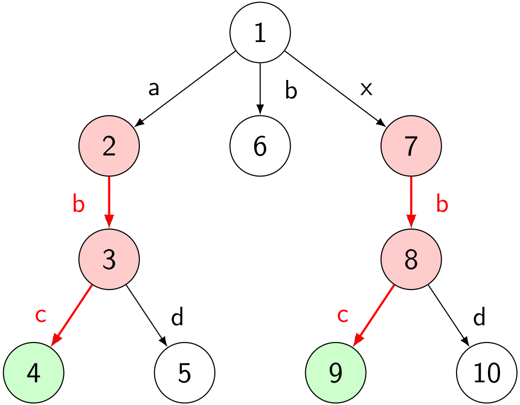


Figure 2: The language is $\{abc, abd, b, xbc, xbd\}$. The path for `xbd` is highlighted in red.

Sub-Path Query

- **Definition:** Find label sequences starting from *any* node
- **Advantage:** More flexible than prefix queries

Example for 'bc'

- Return the set of states reached by a path with label `bc`.
- In Figure 2, states 4 and 9 are returned.

Why Tries Matter

- **Purpose:** Fundamental data structures for large string sets
- **Strength:** Efficient prefix-based queries
- **Challenge:** Memory consumption can be massive

Real-World Applications

- **Text Processing:** Spell checking systems
- **Bioinformatics:** DNA/protein pattern matching
- **Databases:** String indexing and retrieval

Motivations

Why Tries Matter

- **Purpose:** Fundamental data structures for large string sets
- **Strength:** Efficient prefix-based queries
- **Challenge:** Memory consumption can be massive

Real-World Applications

- **Text Processing:** Spell checking systems
- **Bioinformatics:** DNA/protein pattern matching
- **Databases:** String indexing and retrieval

Tries, DFAs and Minimization

- **Theoretical Foundation:** A trie can be viewed as an **acyclic DFA**
- **Minimization Principle:** DFA minimization merges **Myhill–Nerode equivalent states**
- **Algorithmic Solution:** **Linear minimization** for acyclic DFAs [Revuz 1992]

(a) Trie of 2 with equivalence classes.

(b) The minimized automaton.

Figure 3: An example of automaton minimization for trie of 2.

- **Theoretical Foundation:** A trie can be viewed as an **acyclic DFA**
- **Minimization Principle:** DFA minimization merges **Myhill–Nerode equivalent states**
- **Algorithmic Solution:** **Linear minimization** for acyclic DFAs [Revuz 1992]

(a) Trie of 2 with equivalence classes.

(b) The minimized automaton.

Figure 3: An example of automaton minimization for trie of 2.

- **Theoretical Foundation:** A trie can be viewed as an **acyclic DFA**
- **Minimization Principle:** DFA minimization merges **Myhill–Nerode equivalent states**
- **Algorithmic Solution:** **Linear minimization** for acyclic DFAs [Revuz 1992]

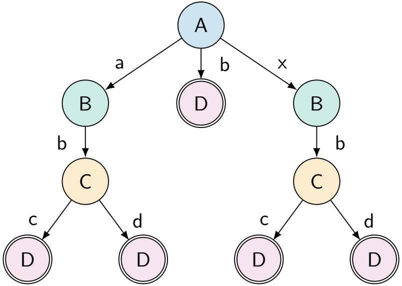
(a) Trie of 2 with equivalence classes.

(b) The minimized automaton.

Figure 3: An example of automaton minimization for trie of 2.

Tries, DFAs and Minimization

- **Theoretical Foundation:** A trie can be viewed as an **acyclic DFA**
- **Minimization Principle:** DFA minimization merges **Myhill–Nerode equivalent states**
- **Algorithmic Solution:** **Linear minimization** for acyclic DFAs [Revuz 1992]

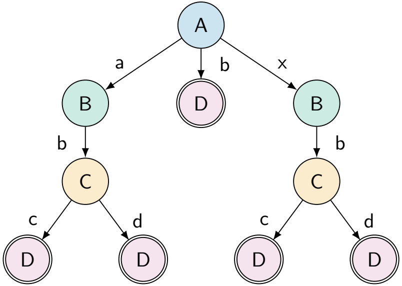


(a) Trie of 2 with equivalence classes.

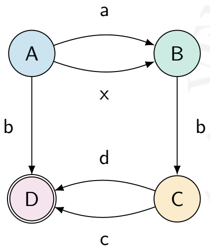
(b) The minimized automaton.

Figure 3: An example of automaton minimization for trie of 2.

- **Theoretical Foundation:** A trie can be viewed as an **acyclic DFA**
- **Minimization Principle:** DFA minimization merges **Myhill–Nerode equivalent states**
- **Algorithmic Solution:** **Linear minimization** for acyclic DFAs [Revuz 1992]

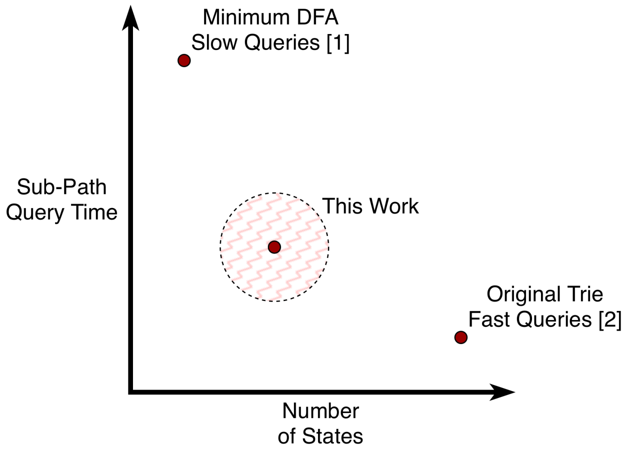


(a) Trie of 2 with equivalence classes.



(b) The minimized automaton.

Figure 3: An example of automaton minimization for trie of 2.



Key Problem

- Tries may contain **large, identical subtrees**

[1] Equi et al. "Graphs cannot be indexed in polynomial time for sub-quadratic time string matching, unless SETH fails"
[2] Ferragina et al. "Compressing and Indexing Labeled Trees, with Applications"

Theoretical Background

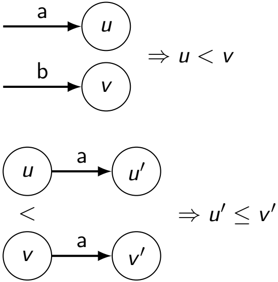
Wheeler Automata [Gagie et al. 2017]

Wheeler Axioms

- The Wheeler axioms defines a total order of the states

(i)

(ii)



Wheeler DFAs (WDFAs)

- If all the states of a DFA are comparable it is called Wheeler DFA

(i) $\underset{\epsilon}{s} < \underset{a}{\{q_1, q_3\}} < \underset{b}{q_2}$

(ii) $\underset{\epsilon}{s} < \underset{a}{q_1} < \underset{a}{q_3} < \underset{b}{q_2}$

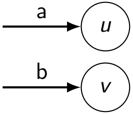
This special ordering allows for highly efficient queries

Wheeler Automata [Gagie et al. 2017]

Wheeler Axioms

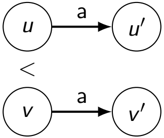
- The Wheeler axioms defines a total order of the states

(i)



$\Rightarrow u < v$

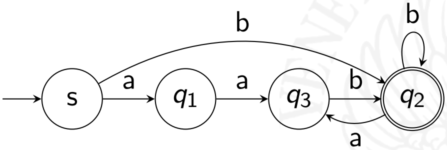
(ii)



$\Rightarrow u' \leq v'$

Wheeler DFAs (WDFAs)

- If all the states of a DFA are comparable it is called Wheeler DFA



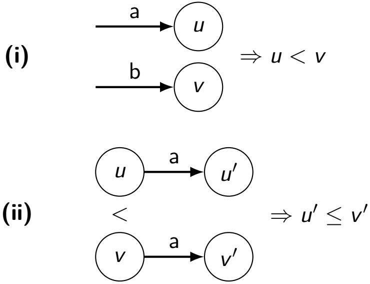
(i) $\underset{\epsilon}{s} < \underset{a}{\{q_1, q_3\}} < \underset{b}{q_2}$

(ii) $\underset{\epsilon}{s} < \underset{a}{q_1} < \underset{a}{q_3} < \underset{b}{q_2}$

This special ordering allows for highly efficient queries

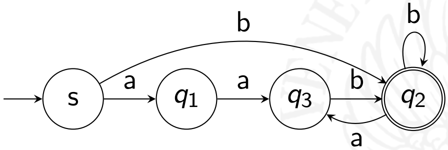
Wheeler Axioms

- The Wheeler axioms defines a total order of the states



Wheeler DFAs (WDFAs)

- If all the states of a DFA are comparable it is called Wheeler DFA



(i) $\underset{\epsilon}{s} < \underset{a}{\{q_1, q_3\}} < \underset{b}{q_2}$

(ii) $\underset{\epsilon}{s} < \underset{a}{q_1} < \underset{a}{q_3} < \underset{b}{q_2}$

This special ordering allows for highly efficient queries

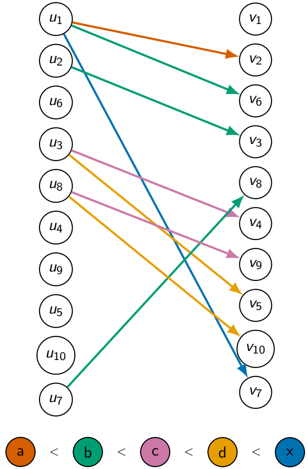


Figure 4: Bipartite representation for trie of 2.

Bipartite Representation

- ✓ Edge labels appear in alphabetical order from top to bottom.
- ✓ No two edges bearing the same label may cross.

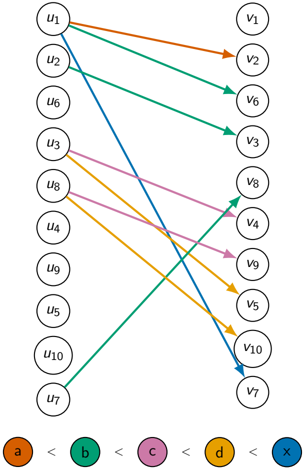
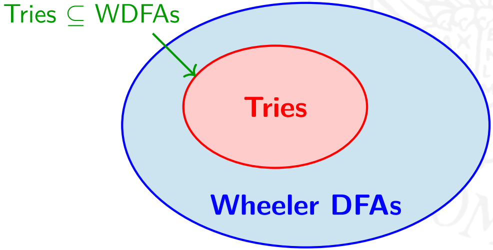


Figure 4: Bipartite representation for trie of 2.

Bipartite Representation

- ✓ Edge labels appear in alphabetical order from top to bottom.
- ✓ No two edges bearing the same label may cross.



Wheeler Automata

Pros	Cons
Linear time queries	Very few automata are Wheeler

Solution:

- Move to **partial orders**
- A partial order can be split in chains
- The minimum number of chains is the width (p)

Important Result:

- Increasing p can yield exponential compression [Manzini et al. 2024]

Wheeler Automata

Pros	Cons
Linear time queries	Very few automata are Wheeler

Solution:

- Move to **partial orders**
- A partial order can be split in chains
- The minimum number of chains is the width (p)

Important Result:

- Increasing p can yield exponential compression [Manzini et al. 2024]

Wheeler Automata

Pros	Cons
Linear time queries	Very few automata are Wheeler

Solution:

- Move to **partial orders**
- A partial order can be split in chains
- The minimum number of chains is the width (p)

Important Result:

- Increasing p can yield exponential compression [Manzini et al. 2024]

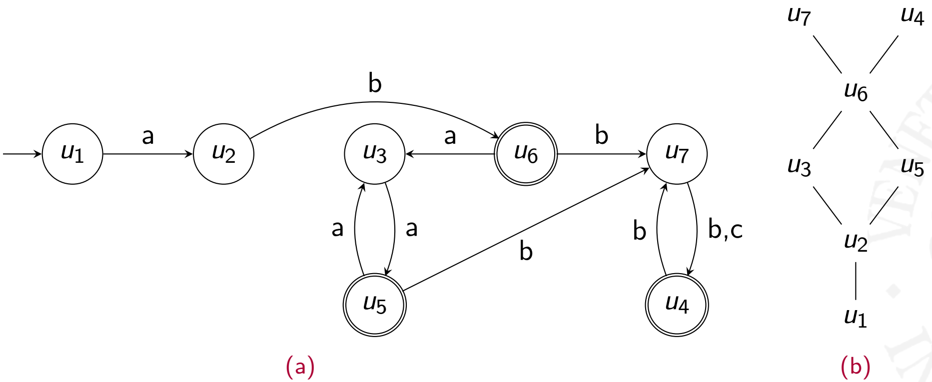


Figure 5: An example of (a) a 2–sortable DFA and (b) the corresponding Hasse diagram of its partial order.

p–sortable Automata Index [Cotumaccio et al. 2021]

Main Result

There exists a **compressed data structure** for p-sortable automata that supports **subpath queries**.

Key Properties

- **Efficiency:** Efficient subpath query time
- **Generality:** Works for both DFA and NFA
- **Scalability:** Performance and space depend on p

p–sortable Automata Index [Cotumaccio et al. 2021]

Main Result

There exists a **compressed data structure** for p-sortable automata that supports **subpath queries**.

Key Properties

- **Efficiency:** Efficient subpath query time
- **Generality:** Works for both DFA and NFA
- **Scalability:** Performance and space depend on p

Proposed Tries Compression Scheme

Our Approach

Partial Minimization Strategy

Produce a **smaller, equivalent automaton** that remains **indexable** by allowing **controlled partial merging** of equivalent subtrees.

Result

Partial Minimization

$\Downarrow$

p–sortable automata

Optimal balance: compression + query efficiency

Key Benefits

- **Space:** Significant memory reduction
- **Time:** Efficient query processing
- **Flexibility:** Adjustable compression level

Our Approach

Partial Minimization Strategy

Produce a **smaller, equivalent automaton** that remains **indexable** by allowing **controlled partial merging** of equivalent subtrees.

Result

Partial Minimization

$\Downarrow$

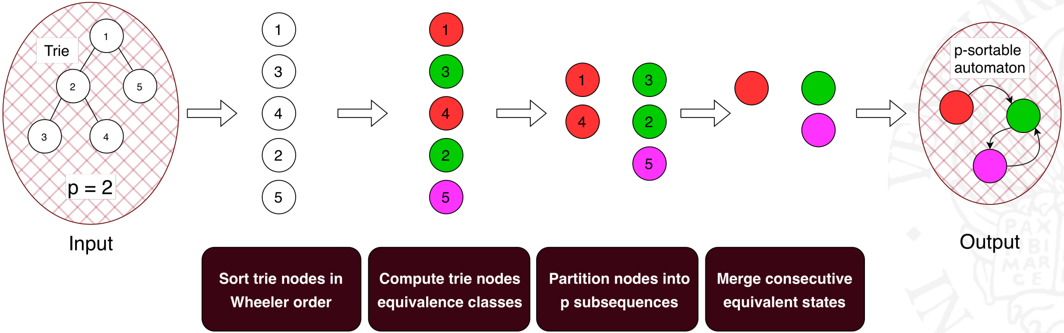
p–sortable automata

Optimal balance: compression + query efficiency

Key Benefits

- **Space:** Significant memory reduction
- **Time:** Efficient query processing
- **Flexibility:** Adjustable compression level

Compression Pipeline



Run

Maximal contiguous subsequence of identical characters within s.

$$s = \texttt{ABDCCDDDDB}$$

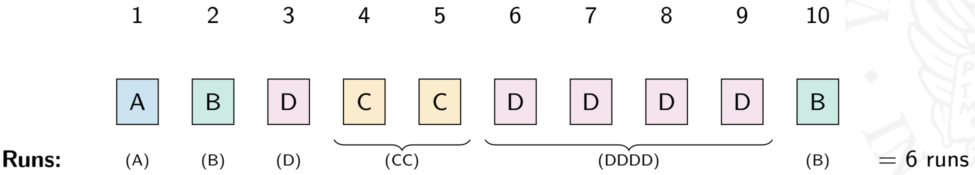


Figure 6: Initial runs of the string induced by the equivalence class of the sorted trie nodes in 2.

String Partitioning Problem

Partition s into p subsequences **minimizing** number of **runs**.

- Reducible to: **Minimum Weight Perfect Bipartite Matching (MWPBM)**

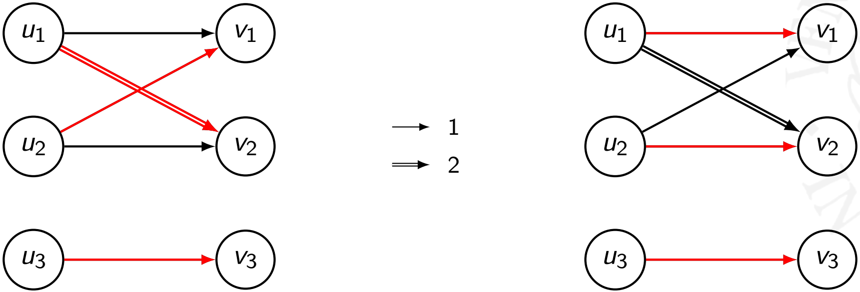
(a) Non-min. weight perfect matching (tot 4)

(b) Min. weight perfect matching (tot 3)

String Partitioning Problem

Partition s into p subsequences **minimizing** number of **runs**.

- Reducible to: **Minimum Weight Perfect Bipartite Matching (MWPBM)**



(a) Non-min. weight perfect matching (tot 4)

(b) Min. weight perfect matching (tot 3)

Reduction to Bipartite Graph Matching

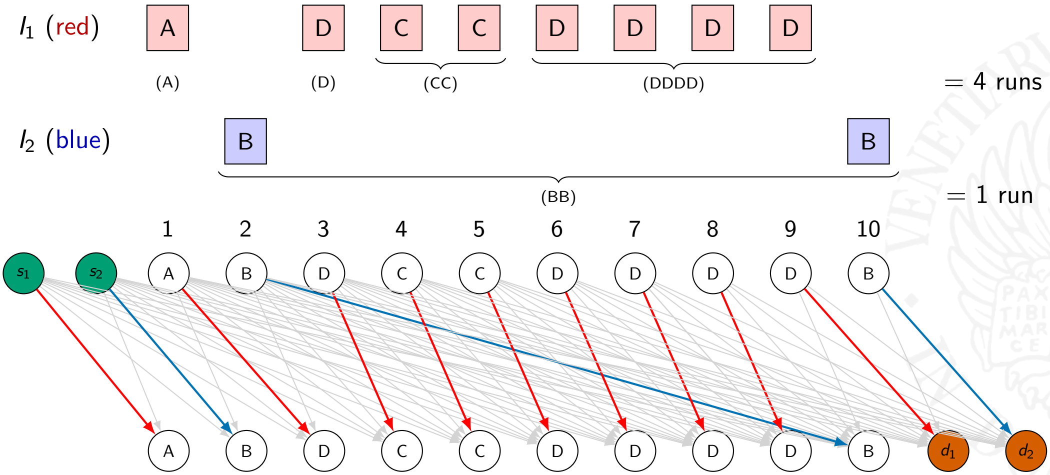


Figure 8: Reduction for string in 6. The minimum weight perfect matching is highlighted.

Graph Construction

- **Nodes:** Correspond to string characters
- **Edges:** Encode successor relation
  - Weight represents transition cost: 1 if characters differ, 0 otherwise
- **Complexity:** $\mathcal{O}(n^2)$ edges in current version
  - Can be improved to $\mathcal{O}(np)$

Optimization Objective

Solving **MWPBM** yields an **optimal partition** minimizing the **total number of runs** across all subsequences.

Reduction to Bipartite Graph Matching

Graph Construction

- **Nodes:** Correspond to string characters
- **Edges:** Encode successor relation
  - Weight represents transition cost: 1 if characters differ, 0 otherwise
- **Complexity:** $\mathcal{O}(n^2)$ edges in current version
  - Can be improved to $\mathcal{O}(np)$

Optimization Objective

Solving **MWPBM** yields an **optimal partition** minimizing the **total number of runs** across all subsequences.

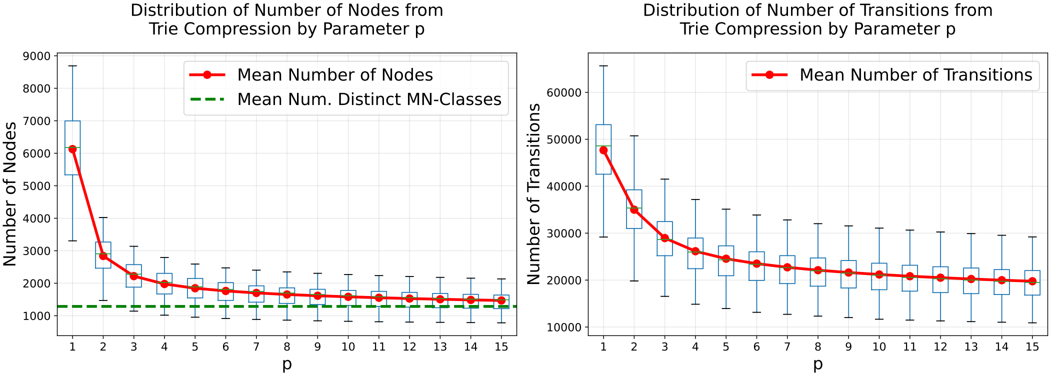
Implementation and Experiments

Experimental Setup

- **Implementation**: C++ for high performance and efficiency
- **Hardware**: Apple M4 Pro with 24 GB RAM
- **Dataset**: Synthetic tries with controlled size and repetitiveness

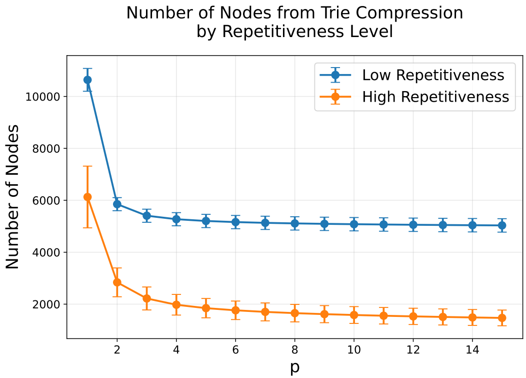
Experimental Results (1)

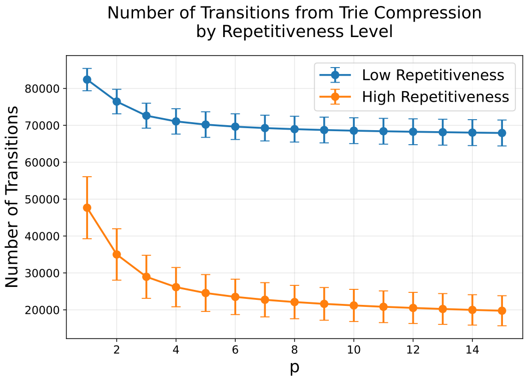
- Each trie in the dataset contains **100,000 nodes**
- Increasing p from 1 to 2 **halves the number of states**
- Compression ratio **approaches the minimal number of states**



Experimental Results (2)

- Each trie in the datasets contains **100,000 nodes**
- **Tries with high repetitiveness compress better** independently of p





Conclusions

Conclusions

Our Contribution

Introduced a new compressed index for tries

Key Achievements:

- **Memory Efficiency**: Significant space reduction on repetitive datasets
- **Query Performance**: Maintains efficient sub-path queries
- **Adaptability**: Tunable compression via parameter p

Conclusions

Our Contribution

Introduced a new compressed index for tries

Key Achievements:

- **Memory Efficiency**: Significant space reduction on repetitive datasets
- **Query Performance**: Maintains efficient sub-path queries
- **Adaptability**: Tunable compression via parameter p

Future Works

- **Scalability**: Develop more efficient solutions for large-scale applications.
- **DFA Construction**: Always get a p–sortable DFA from the compression scheme.
- **DFA Minimality**: Guarantee the minimality of the constructed p–sortable DFA.

The research and development of these future works will be continued during the author's PhD program.

Future Works

- **Scalability**: Develop more efficient solutions for large-scale applications.
- **DFA Construction**: Always get a p–sortable DFA from the compression scheme.
- **DFA Minimality**: Guarantee the minimality of the constructed p–sortable DFA.

The research and development of these future works will be continued during the author's PhD program.

Future Works

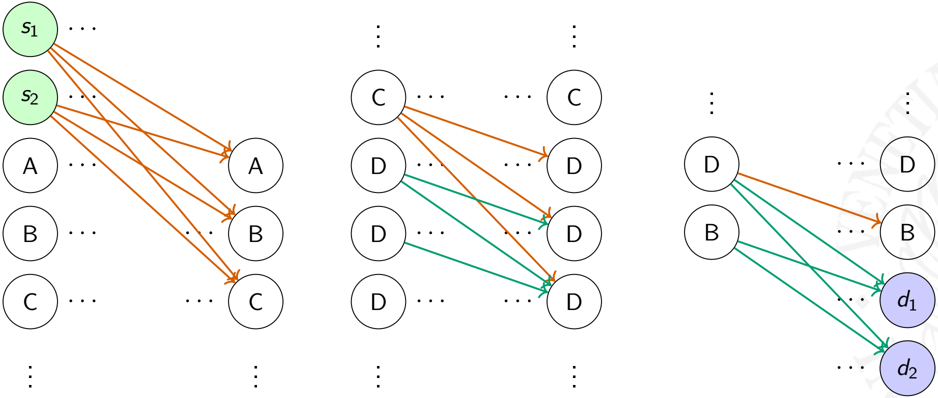
- **Scalability**: Develop more efficient solutions for large-scale applications.
- **DFA Construction**: Always get a p–sortable DFA from the compression scheme.
- **DFA Minimality**: Guarantee the minimality of the constructed p–sortable DFA.

The research and development of these future works will be continued during the author's PhD program.

Thank you for your attention!

Questions?

Example: $\mathcal{O}(n^2)$ MWPBM Reduction



(a) (b) (c)

Figure 9: Small Example for $\mathcal{O}(n^2)$ MWPBM Reduction. Green edges: 0, Red edges: 1.

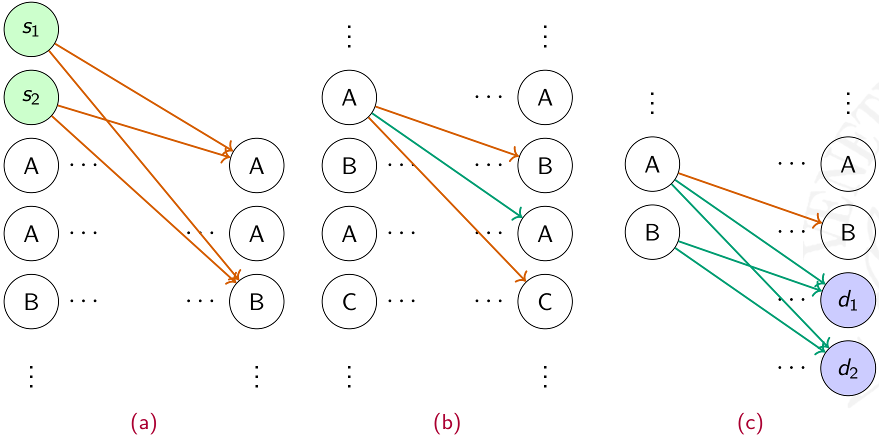


Figure 10: Small Example of $\mathcal{O}(np)$ MWPBM Reduction. Green edges: 0, Red edges: 1..

p–sortable Automata Index [Cotumaccio et al. 2021]

Main Result

Let $\mathcal{A}$ be a **p–sortable automaton**. There exists a **compressed data structure** for $\mathcal{A}$ that supports **subpath queries** on a query word α of length m in

$$\mathbf{O(mp^2 \log\log(p|\Sigma|))} \text{ time}$$

Space Complexity

DFA Case

$\log(|\boldsymbol{\Sigma}|) + \log \mathbf{p} + \mathbf{2}$ bits per edge

NFA Case

$\log(|\boldsymbol{\Sigma}|) + \mathbf{2} \log \mathbf{p} + \mathbf{2}$ bits per edge

Key Properties

- **Efficiency:** Efficient subpath query time
- **Generality:** Works for both DFA and NFA
- **Scalability:** Performance and space depend on p