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

Motivation

Why Tries Matter

- **Purpose:** Fundamental data structures for large string sets
- **Strength:** Efficient prefix-based queries ($O(m)$ time)
- **Challenge:** Memory consumption can be massive
- **Scale:** Real datasets with millions of strings

Real-World Applications

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

Research Objective

Reduce memory footprint while maintaining query efficiency

Example: Trie Representation and String Acceptance

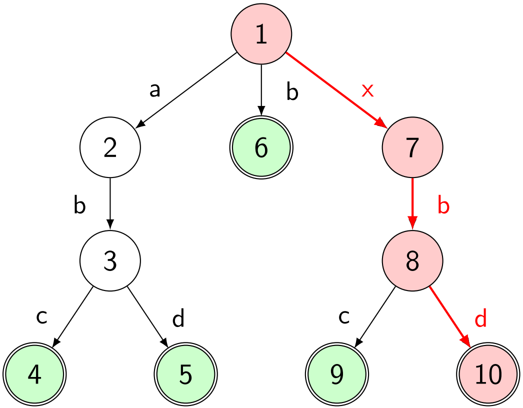


Figure 1: The language is $\{abc, abd, b, xbc, xbd\}$. The path for `xbd` is highlighted in red.

To check if a string like `xbd` is in the language, we traverse the trie from the root:

1. Start at the root (ϵ).
2. Follow the edge labeled `x`.
3. From there, follow the edge labeled `b`.
4. Finally, follow the edge labeled `d`.

Since we end in an accepting state (double circle), the string is in the language.

Indexing and Sub-Path Queries

Indexing Data Structures

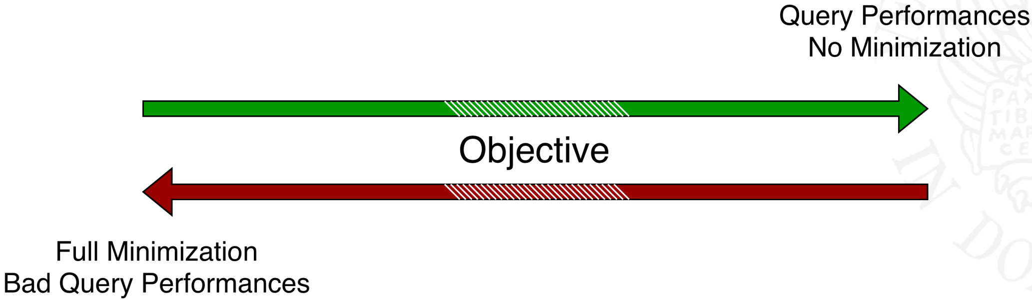
- **Purpose:** Accelerate data retrieval operations
- **Examples:**
  - B-Trees (databases)
  - Hash Tables (key-value stores)
  - Tries (string matching)
- **Applications:** Search engines, databases, file systems

Sub-Path Queries

- **Definition:** Find label sequences starting from *any* node
- **Advantage:** More flexible than prefix queries (root-only)
- **Complexity:** Requires efficient internal node access

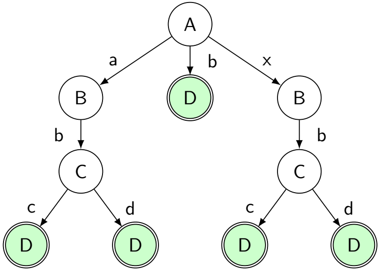
Key Problems

- **Structural Redundancy:** Tries contain **large, identical subtrees**
- **Indexing Challenge:** General DFA indexing is computationally hard [Equi et al. 2023]

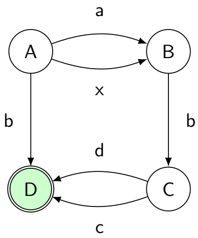


Theoretical Background

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



(a) Trie of 1 with equivalence classes.



(b) The minimized automaton.

Figure 2: An example of automaton minimization for trie of 1.

Wheeler DFA [Gagie et al. 2017]

A **Wheeler DFA (WDFA)** G is a DFA for which there is a total order $\leq$ of the nodes respecting the following **three axioms**.

1. The initial state precedes all other states in the order.

For any two transitions (u, u', a) and (v, v', b):

2. $a < b \implies u \leq v$,
3. $a = b \wedge u' < v' \implies u \leq v$.

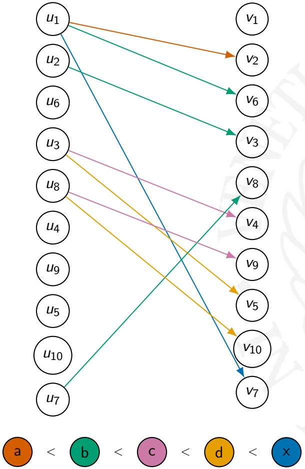


Figure 3: Bipartite representation for trie of 1.

p–sortable Automata: A Generalization of Wheeler Graphs

The Wheeler Limitation

- **Problem:** Not all automata are Wheeler
- **Issue:** Many useful automata cannot be totally ordered
- **Solution:** Relax the ordering constraint
- **Goal:** Extend Wheeler-based indexing applicability

p–sortable Automaton [Cotumaccio et al. 2021]

An automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ is **p–sortable** if its states can be **partitioned** into p subsets $\{Q_1, \ldots, Q_p\}$, each admitting its own **total order** that satisfies the Wheeler axioms within the subset.

Important Result:

- Even a modest increase in p can yield exponential compression [Manzini et al. 2024]

Example: p–sortable Automata

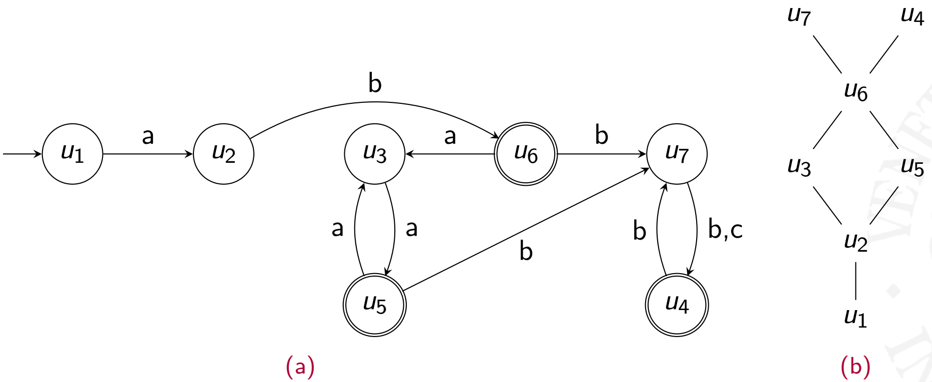


Figure 4: An example of (a) a 2–sortable DFA and (b) the corresponding Hasse diagram of its partial order.

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

Main Result

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

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

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

Proposed Compression Scheme

Our Approach

Partial Minimization Strategy

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

Mathematical Result

Partial Minimization

$\Downarrow$

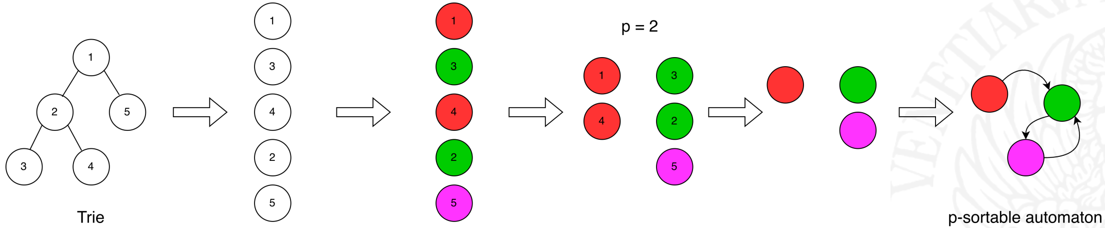
p–sortable automata

Optimal balance: compression + query efficiency

Key Benefits

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

Compression Pipeline



Sort trie nodes

Compute equivalency classes for trie nodes

Partition nodes into p subsequences

Merge consecutive equivalent states

String Partitioning Problem

Run

Maximal contiguous subsequence of identical characters within s.

$$s = \texttt{ABDCCDDDDB}$$

	1	2	3	4	5	6	7	8	9	10	
	A	B	D	C	C	D	D	D	D	B	
Runs:	(A)	(B)	(D)	(CC)		(DDDD)				(B)	= 6 runs

Figure 5: Initial runs of the string induced by the equivalence class of the sorted trie nodes in 1.

String Partitioning Problem

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

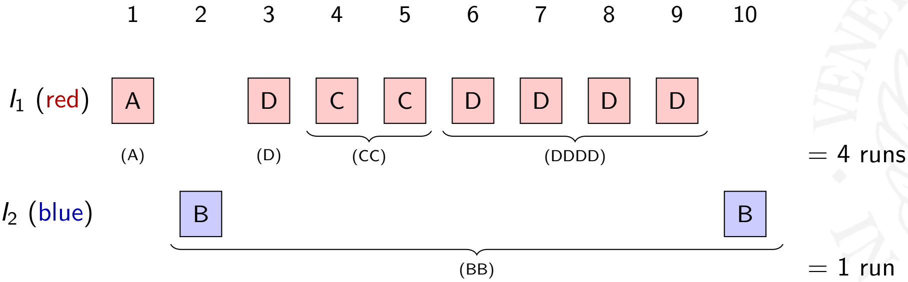


Figure 6: Partition of string in 5 into $p = 2$ subsequences minimizing number of runs. Runs are reduced from 6 to 5.

- The String Partitioning Problem can be reduced to:

Minimum Weight Perfect Bipartite Matching (MWPBM)

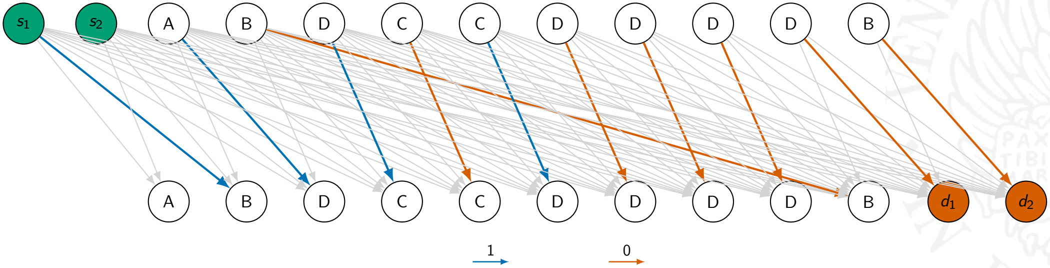


Figure 7: Reduction for string in 5. The minimum weight perfect matching is highlighted.

Reduction to Bipartite Graph Matching

Graph Construction

- **Nodes:** Correspond to string characters
- **Edges:** Encode run boundary costs
  - Weight represents transition cost
- **Complexity:** $\mathcal{O}(n^2)$ edges in current version

Optimization Objective

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

Key Benefits

- **Optimality:** Guaranteed minimum cost solution
- **Generality:** Works for any string partition problem
- **Efficiency:** Polynomial-time solvable

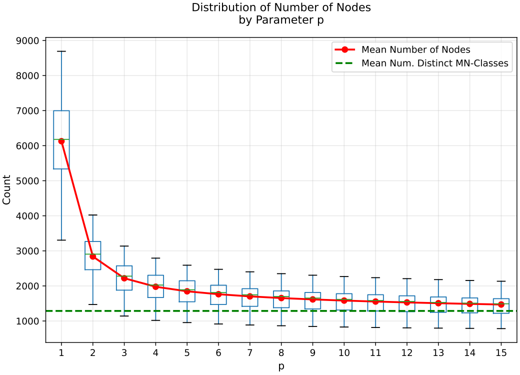
Implementation and Experiments

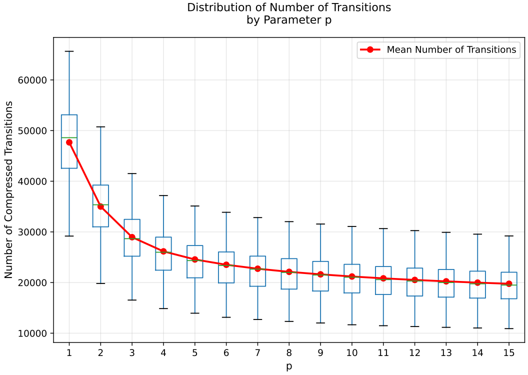
Experimental Setup

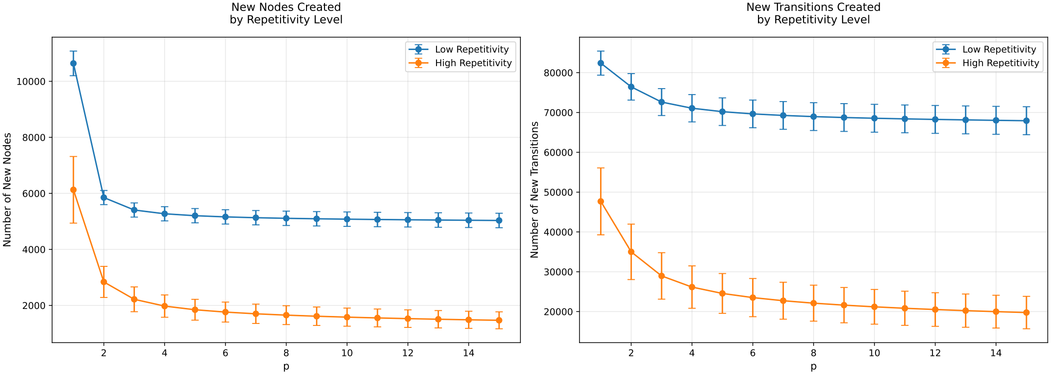
- **Implementation**: C++ for high performance and efficiency
- **Hardware**: Apple M4 Pro with 24 GB RAM
- **Dataset**: Synthetic tries with controlled repetitiveness
  - Each trie contains approximately **100,000 nodes**
  - Variable alphabet sizes and string patterns
  - Designed to test scalability and compression effectiveness

Experimental Results

- Increasing p from 1 to 2 **halves the number of states**
- Compression ratio improves rapidly, approaching the minimal number of states fo larger p






New Nodes Created
by Repetitivity Level
Low Repetitivity
High Repetitivity
Number of New Nodes
10000
8000
6000
4000
2000
2
4
6
8
10
12
14
p
New Transitions Created
by Repetitivity Level
Low Repetitivity
High Repetitivity
Number of New Transitions
80000
70000
60000
50000
40000
30000
20000
2
4
6
8
10
12
14
p

Conclusions

Conclusions

- **Novel Contribution**: Introduced a trie compression method based on p–sortable automata theory
- **Key Achievements**:
  - **Memory Efficiency**: Significant space reduction on repetitive datasets
  - **Query Performance**: Maintains efficient search and traversal operations
  - **Adaptability**: Tunable compression via parameter p
- **Impact**: Opens new research directions in:
  - Compressed automata design
  - String indexing and pattern matching
  - Large-scale text processing applications

Future Works

- **Scalability**: Investigate improvements to the proposed reduction for the String Partitioning problem to develop more scalable and efficient solutions for large-scale applications.
- **DFA Construction**: Explore methods to directly construct a p—sortable deterministic finite automaton from the pipeline, potentially by developing a pruning strategy for the output NFA.
- **DFA Minimality**: Minimize the size of the returned automaton, potentially by providing explicit guarantees of minimality of the returned 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

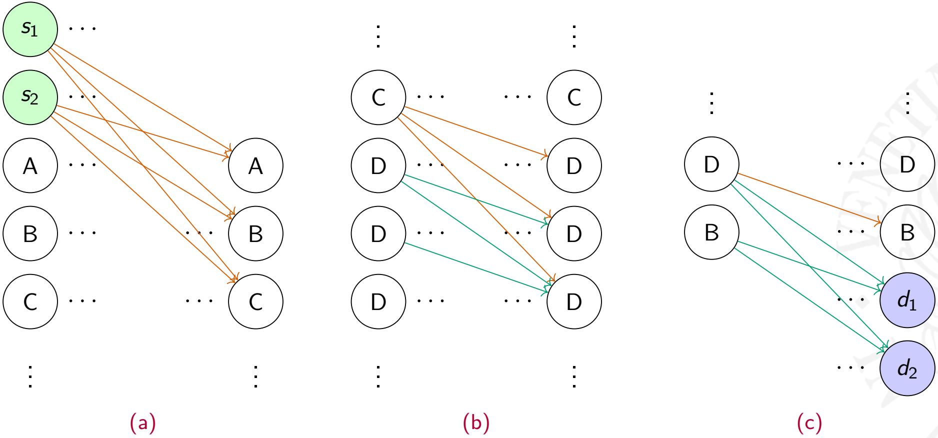


Figure 8: Small Example for $\mathcal{O}(n^2)$ MWPBM Reduction.

Example: $\mathcal{O}(np)$ MWPBM Reduction

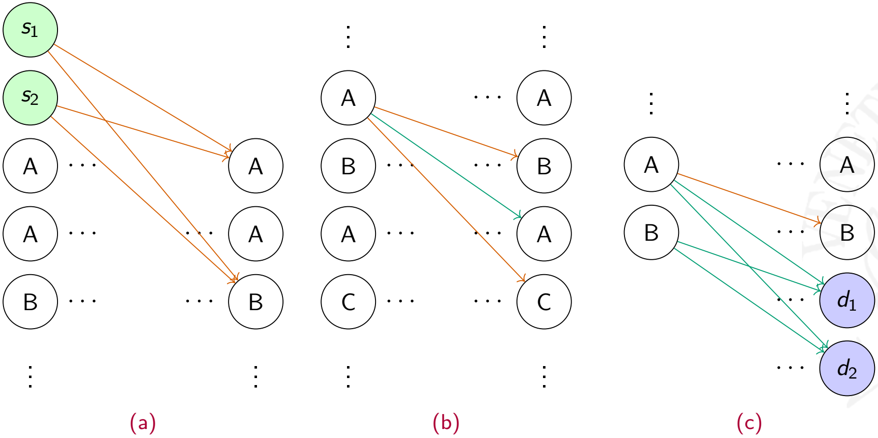


Figure 9: Small Example of $\mathcal{O}(np)$ MWPBM Reduction.