**Slide 1: Title Slide**

"Good morning, everyone. My name is Davide Tonetto, and today I am presenting my Master's thesis defense for the program in Computer Science and Information Technology.

The title of my thesis is 'A New Compression Technique for Repetitive Tries’.”

**Slide 2: Introduction and Motivation**

"I will begin with an introduction to the problem and the motivation behind this research."

**Slide 3: Motivation**

"So, why are tries important? Tries are fundamental data structures used to store large sets of strings. Their primary strength is their ability to perform very efficient prefix-based queries, which run in linear time with respect to the length of the query.

However, they face a significant challenge: memory consumption, which can become massive, especially when dealing with real-world datasets that contain millions of strings.

These structures are not just theoretical; they have many real-world applications, such as spell-checking systems, pattern matching for DNA or proteins in bioinformatics, and string indexing in databases.

**Slide 4: Example: Trie Representation**

“Here is a simple trie for five strings. To check if 'xbd' is in the set, we follow the path from the root. Since we land on an accepting state (the double circle), the string is in the language."

**Slide 5: Indexing and Sub-Path Queries**

"Tries are a powerful type of indexing data structure, which are fundamentally designed to accelerate data retrieval operations.

In this thesis, we focus on a specific, more flexible query type called 'Sub-Path Queries'. These are defined as finding label sequences that start from *any* node in the trie, not just the root."

**Slide 6: Challenges**

"This brings us to the key challenges we must address. The first is structural redundancy. In many real-world datasets, tries contain large, identical subtrees.

The second is the indexing challenge. We know from existing research that indexing general DFAs is computationally hard.

This creates a difficult trade-off, as shown in the diagram. If we use 'No Minimization', we have excellent query performance but a massive memory footprint. If we use 'Full Minimization', we get a very small structure but suffer from bad query performance. Our goal is to find an optimal balance in the middle."

**Slide 7: Theoretical Background**

"To find this balance, we first need to cover some important theoretical background."

**Slide 8: Tries, DFAs and Minimization**

"A trie can be viewed as an acyclic Deterministic Finite Automaton, or DFA. A standard way to compress a DFA is to merge equivalent states, as shown here. For acyclic DFAs, this minimization is very efficient."

**Slide 9: Wheeler Graphs**

"A key concept for us is the Wheeler DFA. This is a special type of DFA where the nodes have a total order that respects the following three axioms…. This special ordering is precisely what allows for highly efficient indexing and compression.

In the right figure, we see a bipartite representation of our running example that allows us to easily check that the following total order for the states of the trie respects the Wheeler axioms."

**Slide 10: p-sortable Automata**

“The problem is, not all automata are Wheeler. A solution is the 'p-sortable automaton'. Here, we partition the states into 'p' subsets. Each subset then has its own internal Wheeler order. This is a crucial idea: even a small 'p' can lead to an exponential compression ratio in the equivalent minimal DFA."

**Slide 11: Example: p-sortable Automata**

"Here is a visual example of a 2-sortable DFA in figure (a), and on the right in figure (b), you can see the corresponding Hasse diagram of its partial order."

**Slide 12: p-sortable Automata Index**

"The main takeaway is that a compressed index exists for p-sortable automata. It supports efficient subpath queries, and its performance and space scale with the parameter 'p'."

**Slide 13: Proposed Compression Scheme**

"With this background, I will now introduce our proposed compression scheme."

**Slide 14: Our Approach**

"Our approach is a 'Partial Minimization Strategy'. The goal is to produce a smaller, equivalent automaton that remains indexable. We achieve this by allowing a *controlled* partial merging of equivalent states.

The central result of our work is that this partial minimization strategy yields p-sortable automata.

This allows us to achieve the optimal balance between compression and query efficiency. The key benefits are significant memory reduction, efficient query processing, and a flexible, adjustable level of compression controlled by the 'p' parameter."

**Slide 15: Compression Pipeline**

"Our compression pipeline consists of four main stages:

1. First, we sort the trie nodes.
2. Second, we compute the equivalence classes for these sorted nodes.
3. Third, we partition the nodes into 'p' subsequences.
4. And finally, we merge the consecutive equivalent states within each subsequence."

**Slide 16: String Partitioning Problem**

"The most critical step is the third one, which we modeled as the String Partitioning Problem.

First, we define a 'Run' as a maximal contiguous subsequence of identical characters within a string.

In Figure 5, we see an example string 's', which represents the equivalence classes of our sorted trie nodes. This string 'ABDCCDDDDB' has 6 initial runs: (A), (B), (D), (CC), (DDDD), and (B)."

**Slide 17: String Partitioning Problem (Continued)**

"The String Partitioning Problem is this: partition the string 's' into 'p' subsequences in a way that *minimizes* the total number of runs.

In Figure 6, we see an optimal partition for.

* Subsequence 1 (red) becomes 'ADCCDDDD', which has 4 runs.
* Subsequence 2 (blue) becomes 'BB', which has only 1 run.

The total number of runs is now 4 + 1 = 5. We have successfully reduced the total runs from 6 to 5."

**Slide 18: Reduction to Bipartite Graph Matching**

"We found that this String Partitioning Problem can be reduced to a classic, well-understood problem: Minimum Weight Perfect Bipartite Matching.

Figure 7 shows the bipartite graph reduction for the string we just saw."

**Slide 19: Reduction to Bipartite Graph Matching (Continued)**

"In our graph construction, the nodes correspond to the string characters. The edges are weighted to encode the 'cost' of run boundaries.

By solving the MWPBM on this graph, we get an optimal partition that is guaranteed to minimize the total number of runs."

**Slide 20: Implementation and Experiments**

"I will now briefly present our implementation and experimental results."

**Slide 21: Experimental Setup**

"Our implementation was written in C++ for high performance. The experiments were run on an Apple M4 Pro with 24 GB of RAM.

We used a dataset of synthetic tries where we could control the repetitiveness. Each trie contained 100,000 nodes."

**Slide 22: Experimental Results**

"The results were very positive. We observed that simply increasing 'p' from 1 to 2 was enough to halve the number of states.

This demonstrates that the compression ratio improves rapidly as 'p' increases, allowing us to approach the minimal number of states."

**Slide 23: Experimental Results 2**

"In the following plots, we compared the compression obtained in a low repetitivity dataset and a high repetitivity one. As we can see, we obtain greater compression for the high repetitivity dataset in both states and transitions."

**Slide 24: Conclusions**

"This brings me to the conclusions of our work."

**Slide 25: Conclusions**

"Our novel contribution is the introduction of a new trie compression method that is based on p-sortable automata theory.

Our key achievements are:

* First, achieving significant memory efficiency, especially on repetitive datasets.
* Second, maintaining efficient search and traversal operations.
* And third, providing adaptable, tunable compression using the parameter 'p'.

The impact of this work is that it opens new research directions in compressed automata design, string indexing, and other large-scale text processing applications."

**Slide 26: Future Works**

"Finally, for future work, we have identified three main directions.

1. **Scalability:** We want to improve the proposed reduction for the String Partitioning problem to develop more scalable solutions.
2. **DFA Construction:** We will explore methods to *directly* construct a p-sortable DFA from our pipeline, perhaps by pruning the output NFA.
3. **DFA Minimality:** We aim to minimize the size of the final automaton, and ideally, provide explicit guarantees of minimality for the returned p-sortable DFA.

The research and development of these future works will be continued during my PhD program."

**Slide 27: Thank You**

"Thank you very much for your attention. I want to express my sincere gratitude to my supervisor, Nicola Prezza, my co-supervisor, Alessio Campanelli, to Ruben Becker, and the REGINDEX group for their guidance and support.

I would be happy to answer any questions."