Graph pattern matching: what if labels can be strings?

Nicola Cotumaccio

Gran Sasso Science Institute, L'Aquila, Italy
Dalhousie University, Halifax, Canada

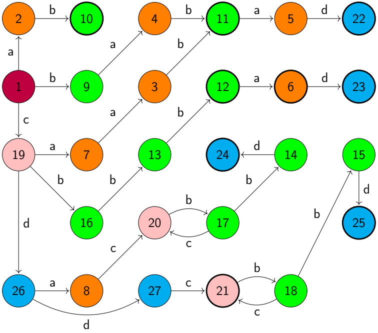
Joint work with Nicola Prezza (Ca' Foscari University, Venice, Italy)

Wheeler graphs

Wheeler graphs generalize the nice properties of De Bruijn graphs.

- A Wheeler graph on the alphabet Σ with n nodes and e edges can be stored using only $2(e+n) + e \log|\Sigma| + |\Sigma| \log e + o(n + e \log|\Sigma|)$ bits.
- This representation allows to decide whether a string α occurs on the graph in only $O(|\alpha| \log|\Sigma|)$ time.

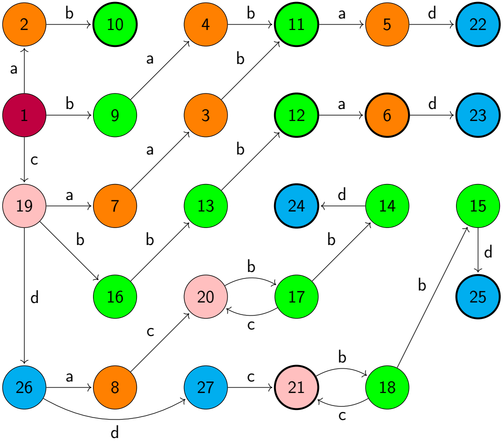
- Wheeler graphs[1] are graphs endowed with a total order $\leq$ on the set of all nodes.
- Here are the properties that the total order $\leq$ must satisfy.



[1] T. Gagie, G. Manzini, J. Sirén, *Wheeler graphs: A Framework for BWT-based Data Structures*, TCS 2017.

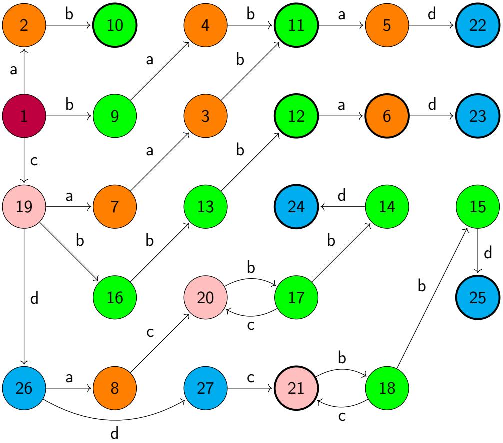
Wheeler graphs

Nodes without incoming edges come first.



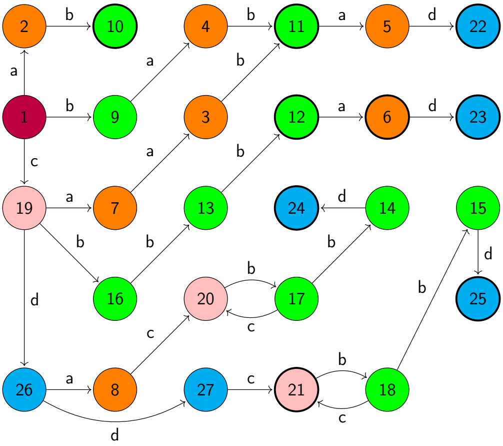
Wheeler graphs

All nodes reached by a come before all nodes reached by b, which come before all nodes reached by c...



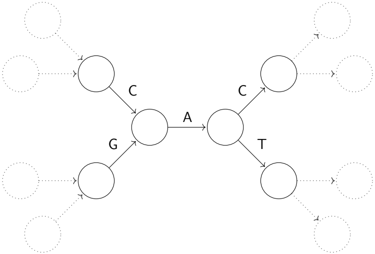
Wheeler graphs

Equally-labeled edges must respect the total order (think of $(7, 3, a)$, $(9, 4, a)$, $(6, 23, d)$, $(15, 25, d)$).



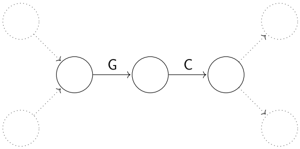
Families of strings

1. We can think of a graph as a data structure storing a family of strings.
2. Solving a pattern matching query is equivalent to deciding whether a pattern is one of such strings.
3. The topology of the graph close to a node yields a local description of the family of strings.
4. Somewhere in the string we may have an occurrence of *CAC* or *CAT* or *GAC* or *GAT*.



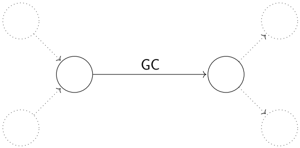
Families of strings

1. However, with this formalism we are not able to capture some properties.
2. For example, we are not able to capture a family of strings such that every occurrence of G must be followed by C.
3. The graph also captures strings ending with G.



String-labeled graphs

1. The solution is to allow string-labeled graphs.
2. Now, when we read a string on the graph, we are required to read the whole string labeling the last edge.
3. The idea of compressing unary paths (for examples, in compressed tries) is an old one, but here it is not only a matter of efficiency.
4. By allowing labels to be strings, we increase the expressive power of Wheeler graphs.



String-labeled graphs

1. By extending Wheeler axioms to string-labeled graphs, we can capture more families of strings while allowing efficient pattern matching queries.
2. We expect pattern matching queries to be solvable in $O(k^2|\alpha|\log|\Sigma|)$ time, where k is the maximum length of a string labeling an edge.

String-labeled automata

1. A more formal way to understand which families of string we are now able to capture is obtained by switching to automata.
2. A regular language is Wheeler if it is recognized by an automaton whose underlying graph is Wheeler.
3. Not all regular languages are Wheeler, if we can only consider classical automata (where edges are characters, not strings).

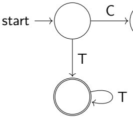
String-labeled automata

For standard automata, we have[2]:

Theorem

A regular language is Wheeler if and only if its Nerode classes can be split into a finite number of (co-lexicographically) convex sets.

1. The minimum DFA in the figure is not Wheeler.
2. However, we now show that the language recognized by the minimum DFA is Wheeler.



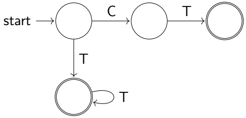
[2] J. Alanko, G. D'Agostino, A. Policriti, N. Prezza, *Regular Languages Meet Prefix Sorting*, SODA 2020.

String-labeled automata

Theorem

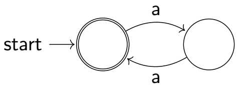
A regular language is Wheeler if and only if its Nerode classes can be split into a finite number of (co-lexicographically) convex sets.

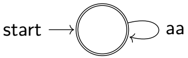
1. Nerode classes can be visualized by considering the minimum DFA and considering the words reaching each state from the initial state.
2. Nerode classes are $\{\epsilon\}$, $\{C\}$, $\{CT\}$, $\{T, TT, TTT, \dots\}$.
3. They can be split into $\{\epsilon\}$, $\{C\}$, $\{T\}$, $\{CT\}$, $\{TT, TTT, \dots\}$, which is a finite partition.



String-labeled automata

1. On string-labeleled automata, we generalize the Nerode relation to subsets of $Pref(\mathcal{L})$.
2. A subset $\mathcal{W}$ of $Pref(\mathcal{L})$ is *admissible* if for some string-labeled automaton recognizing $\mathcal{L}$ we have that $\mathcal{W}$ is equal to set of words reaching some state from the initial state.
3. In the figure, $\mathcal{L} = (aa)^*$.
4. On standard automata, the unique admissible subset is $Pref(\mathcal{L})$ itself, but if we allow generalized automata more subsets can be admissible ($\mathcal{W} = \{\epsilon, aa, aaaa, aaaaaa, \dots\}$).





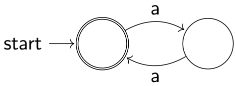
String-labeled automata

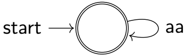
One can show:

Theorem

A regular language is (generalized) Wheeler if and only if the Nerode classes of some admissible set for the language can be split into a finite number of (co-lexicographically) convex sets.

The language $\mathcal{L} = (aa)^*$ is not Wheeler, but it is generalized Wheeler (consider $Pref(\mathcal{L}) = \{\epsilon, a, aa, aaa, \dots\}$ and $\mathcal{W} = \{\epsilon, aa, aaaa, aaaaaa, \dots\}$).





Open problems (work in progress)

1. Are all regular languages generalized Wheeler?
2. How to characterize admissible sets?
3. How to merge this idea with previous techniques (p-sortable languages...)?

Graph pattern matching: what if labels can be strings?

Nicola Cotumaccio

Gran Sasso Science Institute, L'Aquila, Italy
Dalhousie University, Halifax, Canada

Joint work with Nicola Prezza (Ca' Foscari University, Venice, Italy)