A Simple Streaming Bit-parallel Algorithm for Swap Pattern Matching

Václav Blažej
(joint work with Ondřej Suchý and Tomáš Valla)

Faculty of Information Technology
Czech Technical University in Prague

November 15, 2017

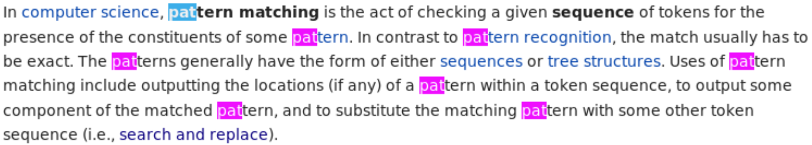
Exact pattern matching

In computer science, **pattern matching** is the act of checking a given **sequence** of tokens for the presence of the constituents of some pattern. In contrast to pattern recognition, the match usually has to be exact. The patterns generally have the form of either sequences or tree structures. Uses of pattern matching include outputting the locations (if any) of a pattern within a token sequence, to output some component of the matched pattern, and to substitute the matching pattern with some other token sequence (i.e., search and replace).

Exact pattern matching

In computer science, **pattern matching** is the act of checking a given **sequence** of tokens for the presence of the constituents of some pattern. In contrast to pattern recognition, the match usually has to be exact. The patterns generally have the form of either sequences or tree structures. Uses of pattern matching include outputting the locations (if any) of a pattern within a token sequence, to output some component of the matched pattern, and to substitute the matching pattern with some other token sequence (i.e., search and replace).

Exact pattern matching

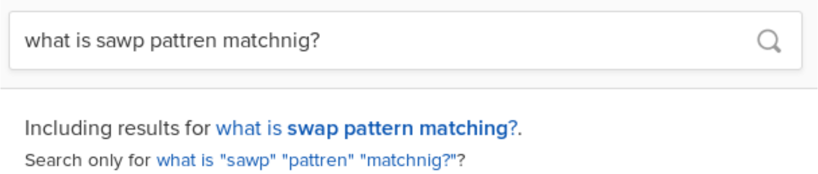


Exact pattern matching

In computer science, pattern **matching** is the act of checking a given **sequence** of tokens for the presence of the constituents of some pattern. In contrast to pattern recognition, the match usually has to be exact. The patterns generally have the form of either sequences or tree structures. Uses of pattern matching include outputting the locations (if any) of a pattern within a token sequence, to output some component of the matched pattern, and to substitute the matching pattern with some other token sequence (i.e., search and replace).

Swap pattern matching (Swap Matching)

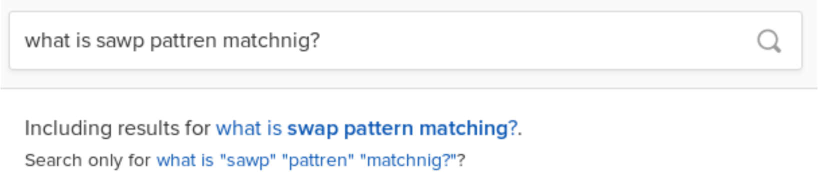
What is sawp pattren matchnig?
Did you mean: “What is swap pattern matching?” ?



- we are allowed to swap adjacent symbols

Swap pattern matching (Swap Matching)

What is sawp pattren matchnig?
Did you mean: “What is swap pattern matching?” ?



- we are allowed to swap adjacent symbols

Definition of Swap Matching problem

We search for occurrences of patterns in the text while allowing pattern to swap adjacent symbols.

We define swaps $\pi : \{1 \dots n\} \to \{1 \dots n\}$ in the pattern S such that:

1. when $\pi(i) = j$ then $\pi(j) = i$ (symbols S_i, S_j are swapped),
2. for all $i, \pi(i) \in \{i-1, i, i+1\}$ (swap only adjacent symbols),
3. when $\pi(i) \neq i$ then $S_{\pi(i)} \neq S_i$ (cannot swap same symbols).

*acbab*cabbab

‖ ‖ ‖ ‖ ‖

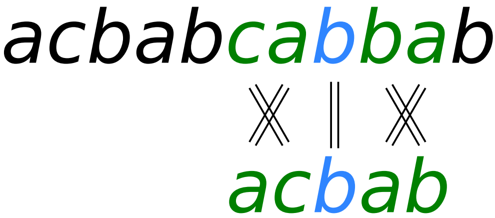
acbab

Definition of Swap Matching problem

We search for occurrences of patterns in the text while allowing pattern to swap adjacent symbols.

We define swaps $\pi : \{1 \dots n\} \to \{1 \dots n\}$ in the pattern S such that:

1. when $\pi(i) = j$ then $\pi(j) = i$ (symbols S_i, S_j are swapped),
2. for all $i, \pi(i) \in \{i-1, i, i+1\}$ (swap only adjacent symbols),
3. when $\pi(i) \neq i$ then $S_{\pi(i)} \neq S_i$ (cannot swap same symbols).



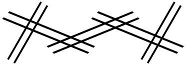
Definition of Swap Matching problem

We search for occurrences of patterns in the text while allowing pattern to swap adjacent symbols.

We define swaps $\pi : \{1 \ldots n\} \to \{1 \ldots n\}$ in the pattern S such that:

1. when $\pi(i) = j$ then $\pi(j) = i$ (symbols S_i, S_j are swapped),
2. for all $i, \pi(i) \in \{i-1, i, i+1\}$ (swap only adjacent symbols),
3. when $\pi(i) \neq i$ then $S_{\pi(i)} \neq S_i$ (cannot swap same symbols).

acbabcabbab



acbab

Swap Matching example

Search for abba in text abbabaaababbbaaabbabaaabbbbaab:

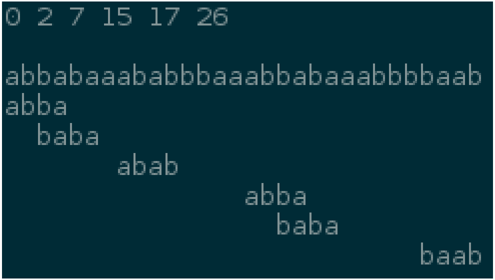


Figure: found occurrences of abba in the text

History

- 1995 - Swap Matching problem was announced as open problem [Muthukrishnan, CPM 95]
- 1997 - first solution using FFT, $O(nm^{\frac{1}{2}} \log m)$ [Amir et al., J. Algorithms]
- 2008 - first non-FFT algorithm, using bit-parallelism [Iliopoulos and Rahman, SOFSEM 2008] $O((n+m)\log m)$
- 2009 - Cross Sampling algorithm which solves the problem in $O(n)$ for short patterns [Cantone and Faro, SOFSEM 2009]
- 2013 - new model using reactive automata and solution with $O(n)$ complexity for short patterns [Faro, PSC 2013]
- 2014 - Smalgo-I algorithm, uses bit-parallelism [Ahmed et al., Theor. Comput. Sci.] $O(\frac{m}{w}n)$

History

- 1995 - Swap Matching problem was announced as open problem [Muthukrishnan, CPM 95]
- 1997 - first solution using FFT, $O(nm^{\frac{1}{2}} \log m)$ [Amir et al., J. Algorithms]
- 2008 - first non-FFT algorithm, using bit-parallelism [Iliopoulos and Rahman, SOFSEM 2008] FATAL ERROR
- 2009 - Cross Sampling algorithm which solves the problem in $O(n)$ for short patterns [Cantone and Faro, SOFSEM 2009]
- 2013 - new model using reactive automata and solution with $O(n)$ complexity for short patterns [Faro, PSC 2013]
- 2014 - Smalgo-I algorithm, uses bit-parallelism [Ahmed et al., Theor. Comput. Sci.] FATAL ERROR

The Model [Iliopoulos and Rahman, SOFSEM 2008]

- Graph represents all patterns which are feasible.

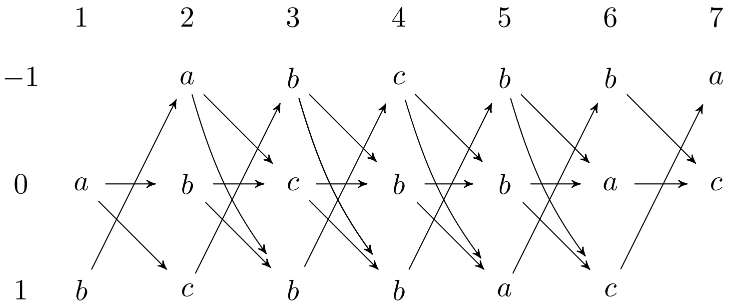


Figure: Graph for $P = abcbbac$

The Model [Iliopoulos and Rahman, SOFSEM 2008]

- Graph represents all patterns which are feasible.
- Each path from first to last column is one such pattern.

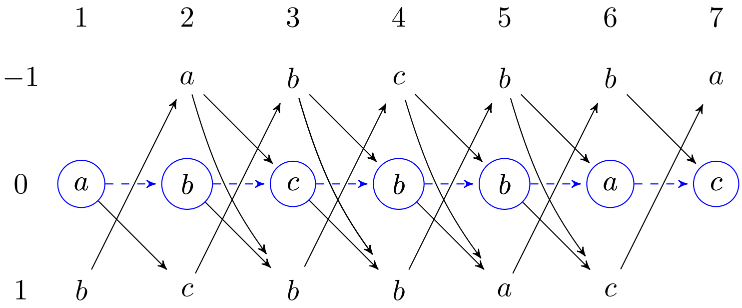


Figure: Graph for $P = abcbbac$

The Model [Iliopoulos and Rahman, SOFSEM 2008]

- Graph represents all patterns which are feasible.
- Each path from first to last column is one such pattern.

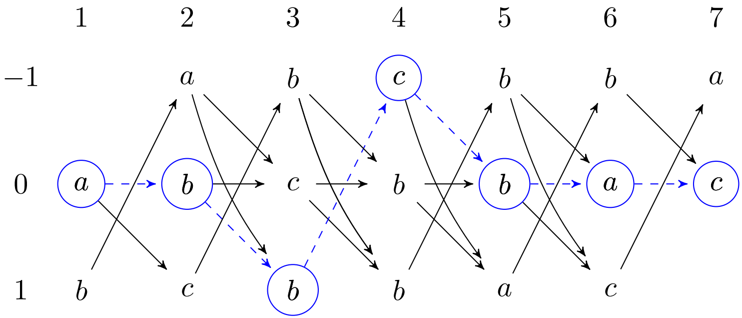


Figure: Graph for $P = abcbbac$

The Model [Iliopoulos and Rahman, SOFSEM 2008]

- Graph represents all patterns which are feasible.
- Each path from first to last column is one such pattern.

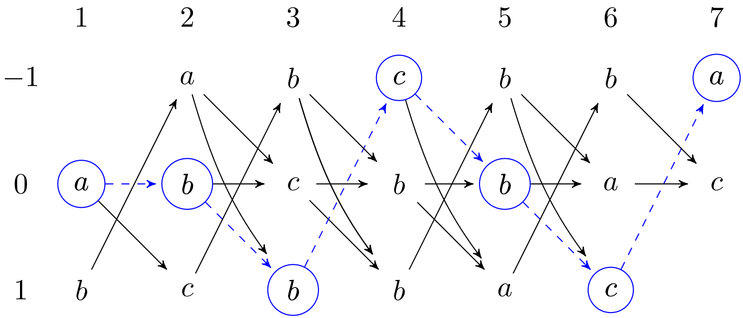


Figure: Graph for $P = abcbbac$

The Model [Iliopoulos and Rahman, SOFSEM 2008]

- Graph represents all patterns which are feasible.
- Each path from first to last column is one such pattern.
- The *signal* is information that a path partially matches.

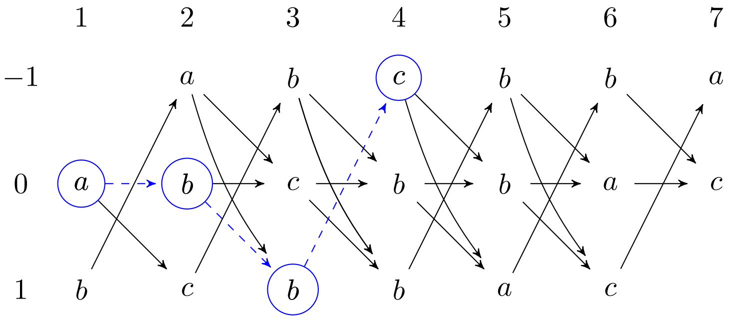


Figure: Graph for $P = abcbbac$

Our algorithm

We designed a new algorithm for the swap matching problem

- uses the graph theoretical model,
- takes input as stream of symbols,
- bitwise parallelism of machine instructions,
- can be implemented using only $7 + |\Sigma|$ memory cells.

Our algorithm

We use the bitwise representation and operations to simulate signal propagation through the model.

- Represent each row with bit array.
- Use *shift* and *or* to move signal.
- Use *and* to filter our signal.

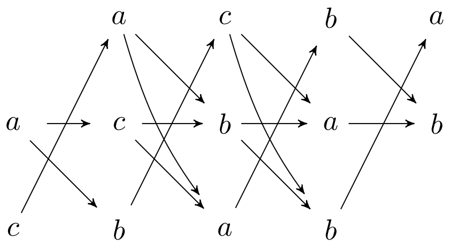


Figure: Example run for $P = acbab$ and $T = acabab$

Our algorithm

We use the bitwise representation and operations to simulate signal propagation through the model.

- Represent each row with bit array.
- Use *shift* and *or* to move signal.
- Use *and* to filter our signal.

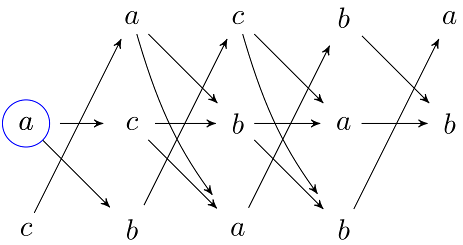


Figure: Example run for $P = acbab$ and $T = \underline{a}cabab$

Our algorithm

We use the bitwise representation and operations to simulate signal propagation through the model.

- Represent each row with bit array.
- Use *shift* and *or* to move signal.
- Use *and* to filter our signal.

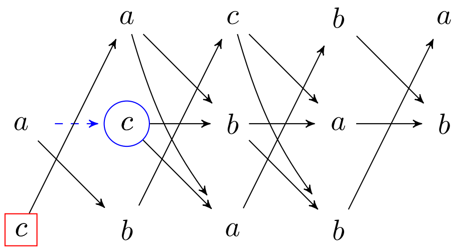


Figure: Example run for $P = acbab$ and $T = \underline{ac}abab$

Our algorithm

We use the bitwise representation and operations to simulate signal propagation through the model.

- Represent each row with bit array.
- Use *shift* and *or* to move signal.
- Use *and* to filter our signal.

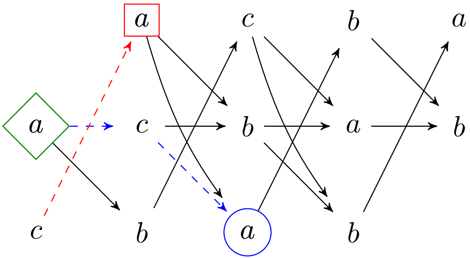


Figure: Example run for $P = acbab$ and $T = \underline{aca}bab$

Our algorithm

We use the bitwise representation and operations to simulate signal propagation through the model.

- Represent each row with bit array.
- Use *shift* and *or* to move signal.
- Use *and* to filter our signal.

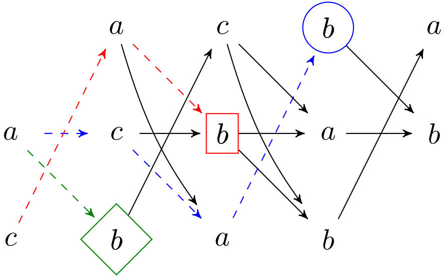


Figure: Example run for $P = acbab$ and $T = \underline{acab}ab$

Our algorithm

We use the bitwise representation and operations to simulate signal propagation through the model.

- Represent each row with bit array.
- Use *shift* and *or* to move signal.
- Use *and* to filter our signal.

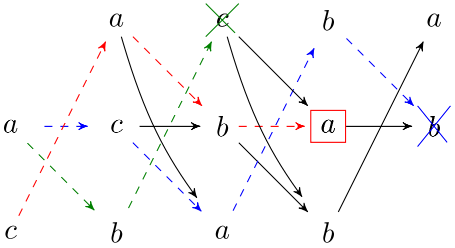


Figure: Example run for $P = acbab$ and $T = \underline{acabab}$

Our algorithm

We use the bitwise representation and operations to simulate signal propagation through the model.

- Represent each row with bit array.
- Use *shift* and *or* to move signal.
- Use *and* to filter our signal.

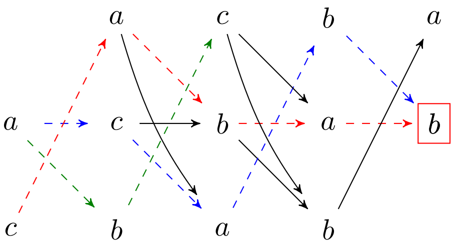
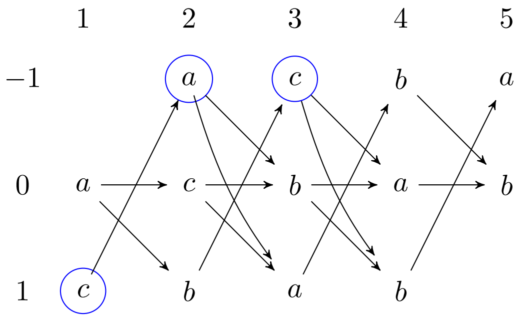


Figure: Example run for $P = acbab$ and $T = \underline{acabab}$

Our algorithm

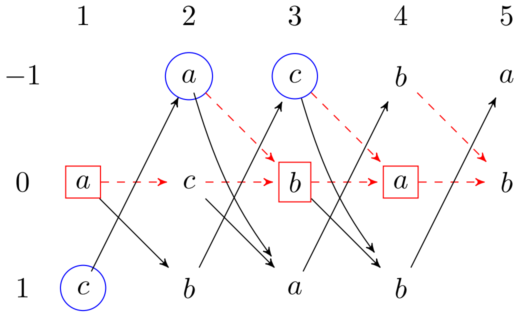
Step 1 – signal propagation – using shift and or operation



Our algorithm

Step 1 – signal propagation – using shift and or operation

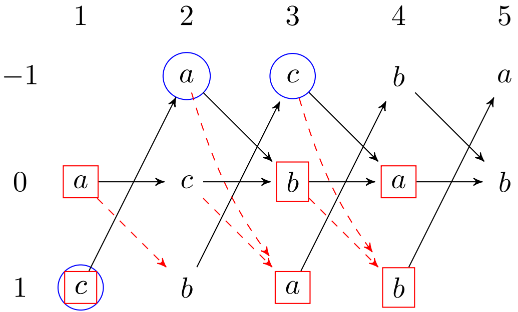
- for row 0 shift and make *or* of rows -1 and 0



Our algorithm

Step 1 – signal propagation – using shift and or operation

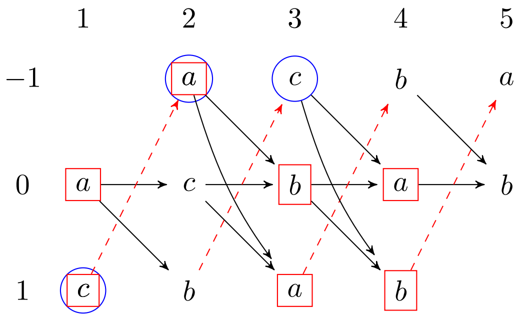
- for row 0 shift and make *or* of rows -1 and 0
- for row 1 shift and make *or* of rows -1 and 0



Our algorithm

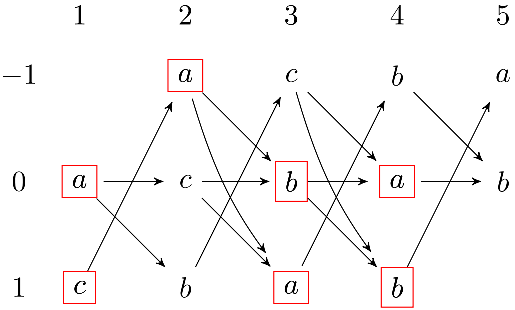
Step 1 – signal propagation – using shift and or operation

- for row 0 shift and make *or* of rows -1 and 0
- for row 1 shift and make *or* of rows -1 and 0
- for row -1 shift and add row 1



Our algorithm

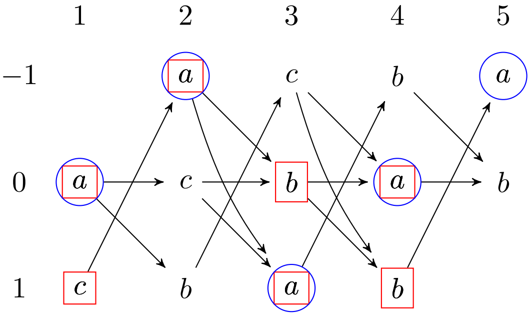
Step 2 – signal filtration – using and operation



Our algorithm

Step 2 – signal filtration – using and operation

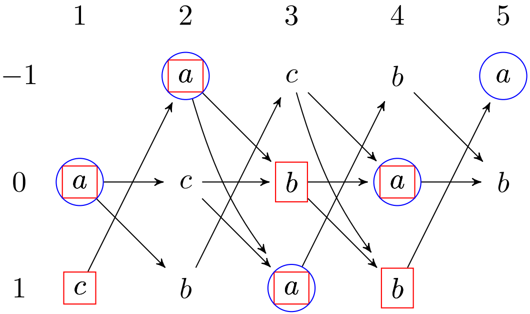
- get mask for currently read symbol (say a)



Our algorithm

Step 2 – signal filtration – using and operation

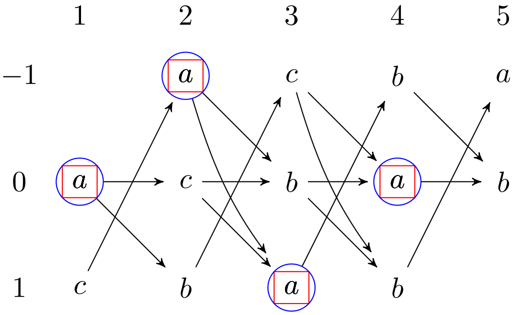
- get mask for currently read symbol (say a)
- make and operation so that invalid signals are filtered out



Our algorithm

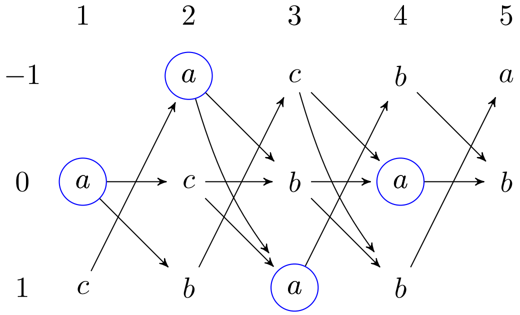
Step 2 – signal filtration – using and operation

- get mask for currently read symbol (say a)
- make and operation so that invalid signals are filtered out



Our algorithm

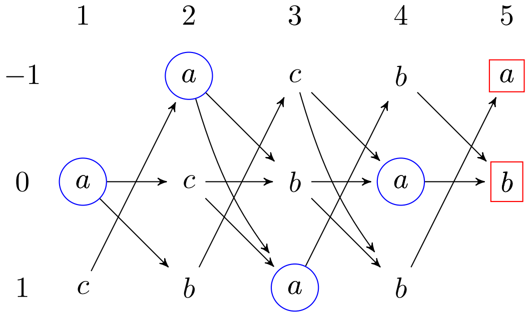
Step 3 – check result



Our algorithm

Step 3 – check result

- check if there is a signal in the last column



Properties of our algorithm

For pattern of length m, text n and word size w (using the word-Ram model) we have

- time complexity
$$O(\lceil \frac{m}{w} \rceil (|\Sigma| + n) + m),$$
- space complexity
$$O(\lceil \frac{m}{w} \rceil |\Sigma|).$$
- If $m \leq w$ we get $O(|\Sigma| + m + n)$ time and $O(|\Sigma|)$ space.

Deterministic Finite Automaton (DFA)

Question: Is Swap Matching problem solvable with DFA?

Deterministic Finite Automaton (DFA)

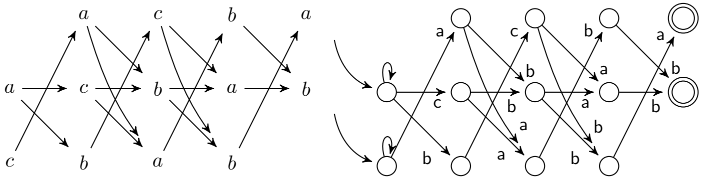
Question: Is Swap Matching problem solvable with DFA?

- Use the model to create non-deterministic automaton.

Deterministic Finite Automaton (DFA)

Question: Is Swap Matching problem solvable with DFA?

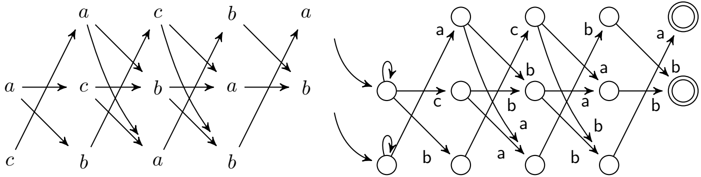
- Use the model to create non-deterministic automaton.



Deterministic Finite Automaton (DFA)

Question: Is Swap Matching problem solvable with DFA?

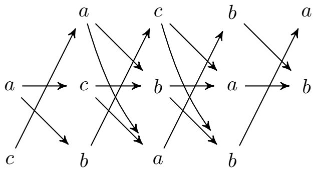
- Use the model to create non-deterministic automaton.
- Determinize the automaton.

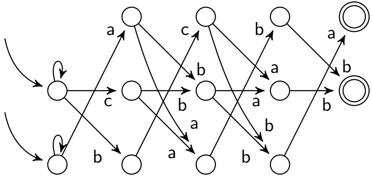


Deterministic Finite Automaton (DFA)

Question: Is Swap Matching problem solvable with DFA?

- Use the model to create non-deterministic automaton.
- Determinize the automaton.
- Any time it reaches final state it reports an occurrence and continues reading input.





Deterministic Finite Automaton (DFA)

Question: Is Swap Matching problem solvable with a *small* DFA?

Theorem

There is an infinite family F of patterns such that any deterministic finite automaton A_P accepting the language $L_S(P) = \{u\pi(P) \mid u \in \Sigma^*, \pi$ is a swap permutation for $P\}$ for $P \in F$ has $2^{\Omega(|P|)}$ states.

Length if these patterns is $|4+5k|$ a $k \in \{1, 2, \ldots\}$.

$P = T_0$	$acccabcccabccc$
T_1	$acccbacccabccc$
T_2	$acccabcccbaccc$
T_3	$acccbacccbaccc$

Table: All strings for $k = 2$.

If the automaton is in the same state after reading strings T_i, T_j such that $T_i \neq T_j$, then there exists such a suffix S such that $T_i.S \in A$ and $T_j.S \notin A$.

Deterministic Finite Automaton (DFA)

Question: Is Swap Matching problem solvable with a *small* DFA?

Theorem

There is an infinite family F of patterns such that any deterministic finite automaton A_P accepting the language $L_S(P) = \{u\pi(P) \mid u \in \Sigma^*, \pi$ is a swap permutation for $P\}$ for $P \in F$ has $2^{\Omega(|P|)}$ states.

Length if these patterns is $|4 + 5k|$ a $k \in \{1, 2, \dots\}$.

$P = T_0$	$acccabcccabccc$
T_1	$acccbacccabccc$
T_2	$acccabcccbaccc$
T_3	$acccbacccbaccc$

Table: All strings for $k = 2$.

If the automaton is in the same state after reading strings T_i, T_j such that $T_i \neq T_j$, then there exists such a suffix S such that $T_i.S \in A$ and $T_j.S \notin A$.

Conclusion

Our results:

- new algorithm for Swap Matching
  - uses the graph theoretical model
  - takes input as stream
  - bitwise parallelism
  - can be implemented in few registers
- found an error in known swap matching algorithm
- proved that Swap Matching is not solvable in poly-time with deterministic finite automata

Open problem: Considering some computational model, is Swap Matching problem solvable in linear time? If not prove there is no effective solution.

Thank you for your attention!