DEVELOPMENT OF ALGORITHM LIBRARY FOR STRING MATCHING SUPPORTED BY VISUALIZATION AND PERFORMANCE COMPARISON USING BENCHMARKS FOR BIOLOGICAL SEQUENCE DATASETS

ALEXANDROS KONSTANTINOS KOKOZIDIS
2019.06.21

UNIVERSITY OF MACEDONIA
DEPARTMENT OF APPLIED INFORMATICS

DEVELOPMENT OF ALGORITHM LIBRARY FOR STRING MATCHING SUPPORTED BY VISUALIZATION AND PERFORMANCE COMPARISON USING BENCHMARKS FOR BIOLOGICAL SEQUENCE DATASETS

Introduction
Implementation
Benchmark
Visualization
Conclusion

Introduction: String searching and applications

Definition.

Pattern E X A

Text S T R I N G S E A R C H I N G E X A M P L E

Well-known algorithms.

Brute Force, Knuth-Morris-Pratt, Boyer-Moore, Karp-Rabin, etc.

Applications.

Text editors (search help, spell check),
Plagiarism detection,
Web search engines,
System intrusion detection,
DNA/ Biological sequence matching,
…

Introduction: Purpose of the thesis

Observations.

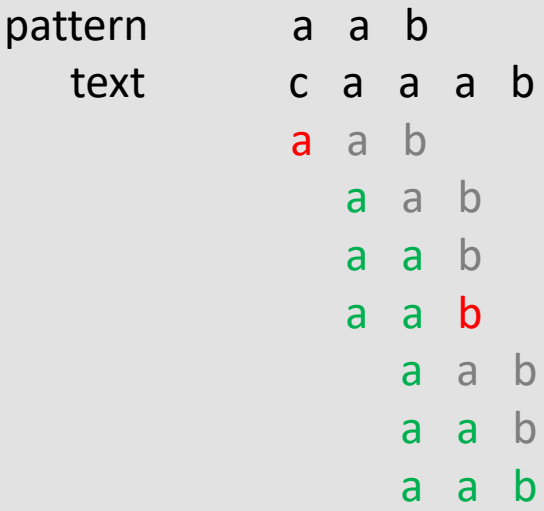
- Over 80 algorithms since 1970.
- Best algorithm in all cases does not exist; each thrives in different scenarios
- Many algorithms are too complicated and hard to understand.
- Theoretical analysis focuses on upper limits; need for practical performance testing.
- Data-heavy applications call for efficient algorithms; need for collective benchmarks!

Purpose.

- Implement algorithms in OO paradigm.
- Visualize their functionality.
- Offer a collective benchmark focused on biological data.

Introduction: About the algorithms

- 35 string searching algorithms presented in C (Charras and Lecroq).
- Each has a preprocessing and a searching phase.
- All receive first the pattern (preprocessing) and then the text (searching).
- All utilize the ‘sliding window’ approach.

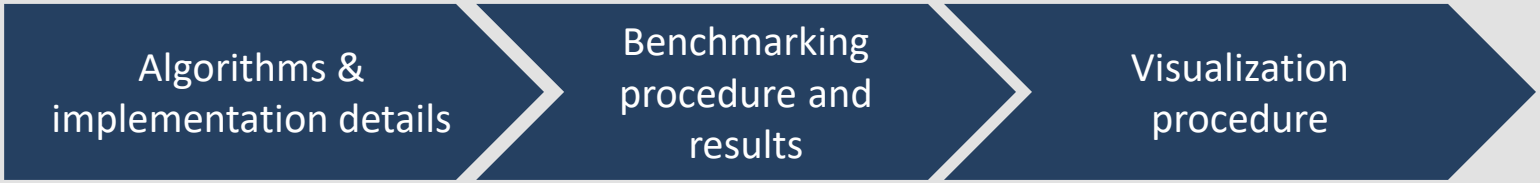


- Categorized in 4 groups: left to right, right to left, specific order, any order.

Introduction: Line of work Vs. Presentation order

Line of work.

Presentation order.



Implementation: Points of interest

Implemented in Java; Object-Oriented approach (Sedgewick).

Common interface.

- Common public methods: **constructor** + ***search*** + ***searchAll***
- Clean separation of preprocessing and searching, consistency.
- Efficient searching of a pattern in different texts

Optimization. Not the point of this thesis. Consistent port of C source code.

The '\0' issue. Termination character in C strings. Reason of most modifications.

Benchmarking: Decisions

Past work. Measured on # of character comparisons; only a handful of algorithms where compared.

Algorithms? All.

Data? Biological. Escherichia coli genome sequence ({A, C, G, T}) of 4.5m chars. https://www.ezbiocloud.net/genome/explore?puid=172783

Measure of performance? Execution times

Functions? ***search*** and ***searchAll***

Pattern size? 2 groups:

- small {2, 3, 4, 5, 6, 7, 8, 9, 10}
- large {10, 20, 40, 80, 160, 320, 640}

Results. Displayed in 3 sections:

- Individual
- Collective
- Collective (grouped)

Benchmarking: Decisions

Past work. Measured on # of character comparisons; only a handful of algorithms where compared.

Algorithms? All.

Data? Biological. Escherichia coli genome sequence ({A, C, G, T}) of 4.5m chars. https://www.ezbiocloud.net/genome/explore?puid=172783

Measure of performance? Execution times

Functions? ***search*** and ***searchAll***

Pattern size? 2 groups:

- small {2, 3, 4, 5, 6, 7, 8, 9, 10}
- large {10, 20, 40, 80, 160, 320, 640}

4 scenarios:

- search for small
- search for large
- searchAll for small
- searchAll for large

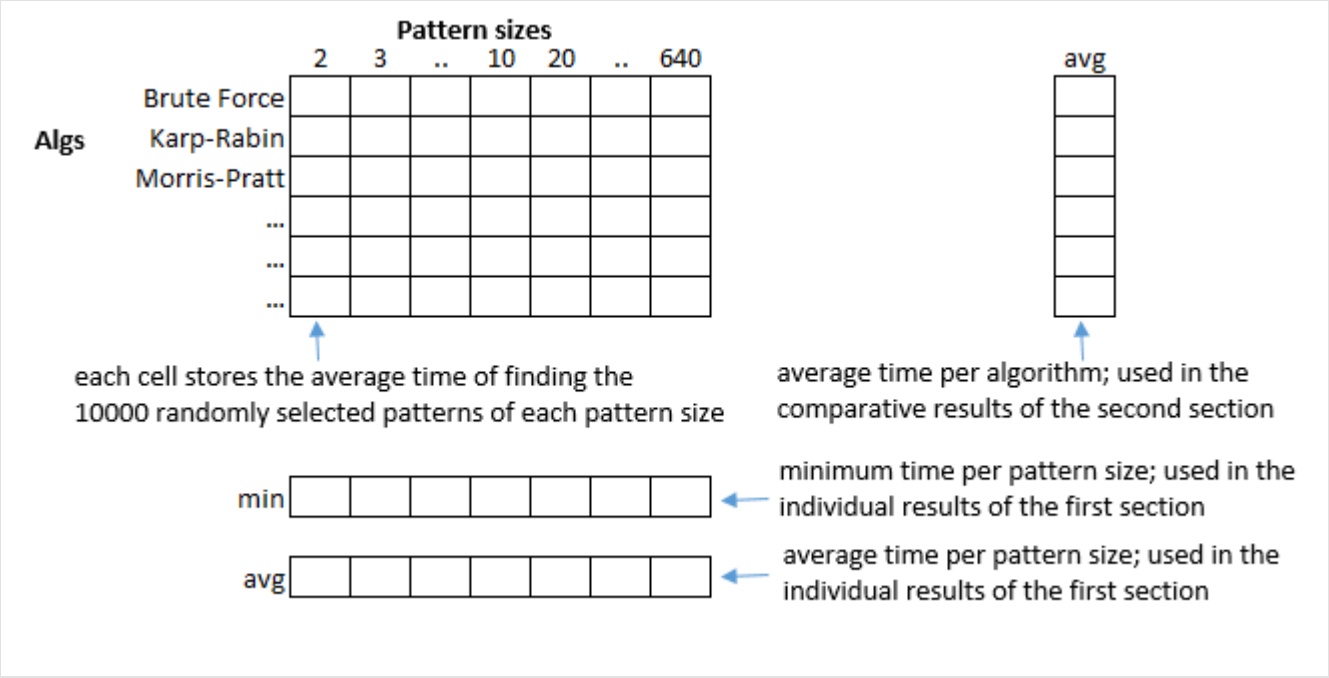
Results. Displayed in 3 sections:

- Individual
- Collective
- Collective (grouped)

Benchmarking: Benchmark Suite

Implementation. Java. Exploits the Reflection API to automatically create and run the tests and auto-validate the correctness of the algorithms.

Scheme. Each algorithm is measured (average) in each of the pattern sizes by selecting randomly 10000 patterns; done for both ***search*** and ***searchAll.***



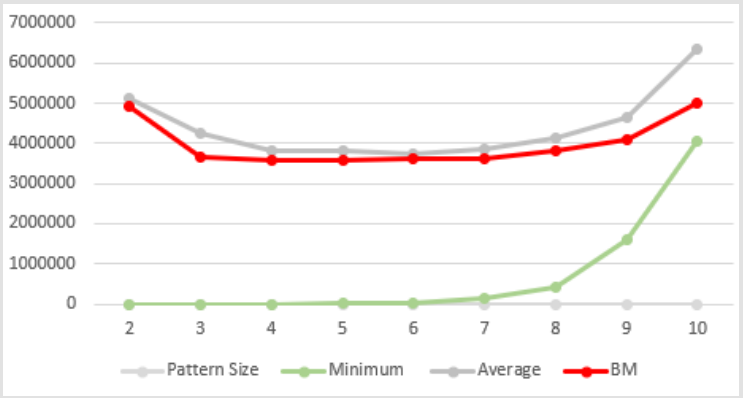
Benchmarking: Individual results

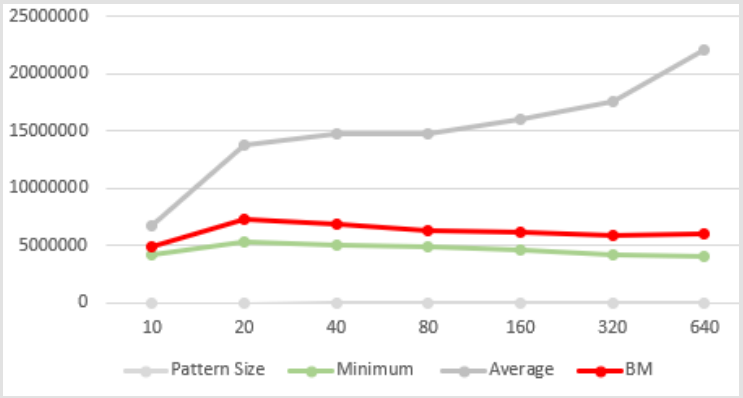
Results. For each algorithm four line charts; one for each scenario. Highlights the performance against the minimum and average times from all the algorithms.

Small patterns

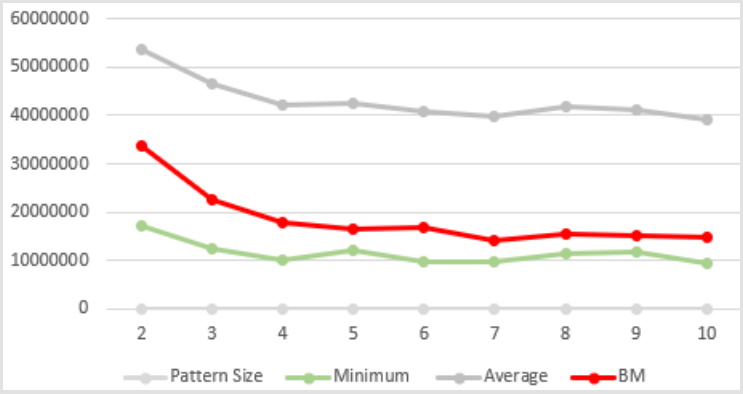
Large patterns

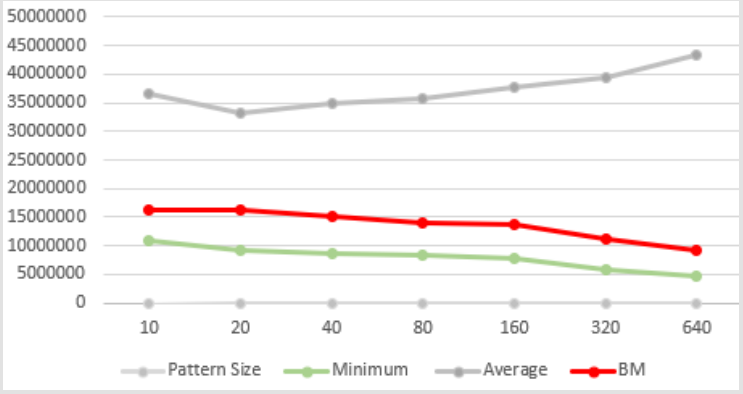
search



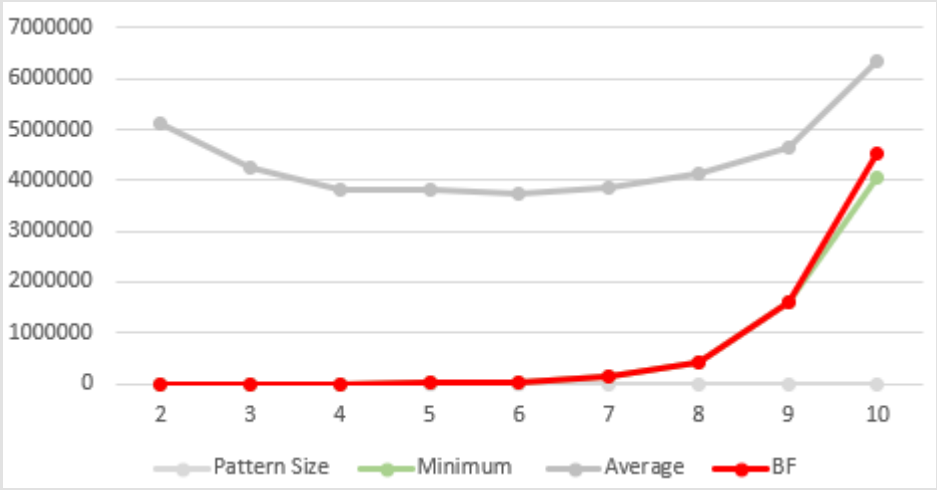


searchAll





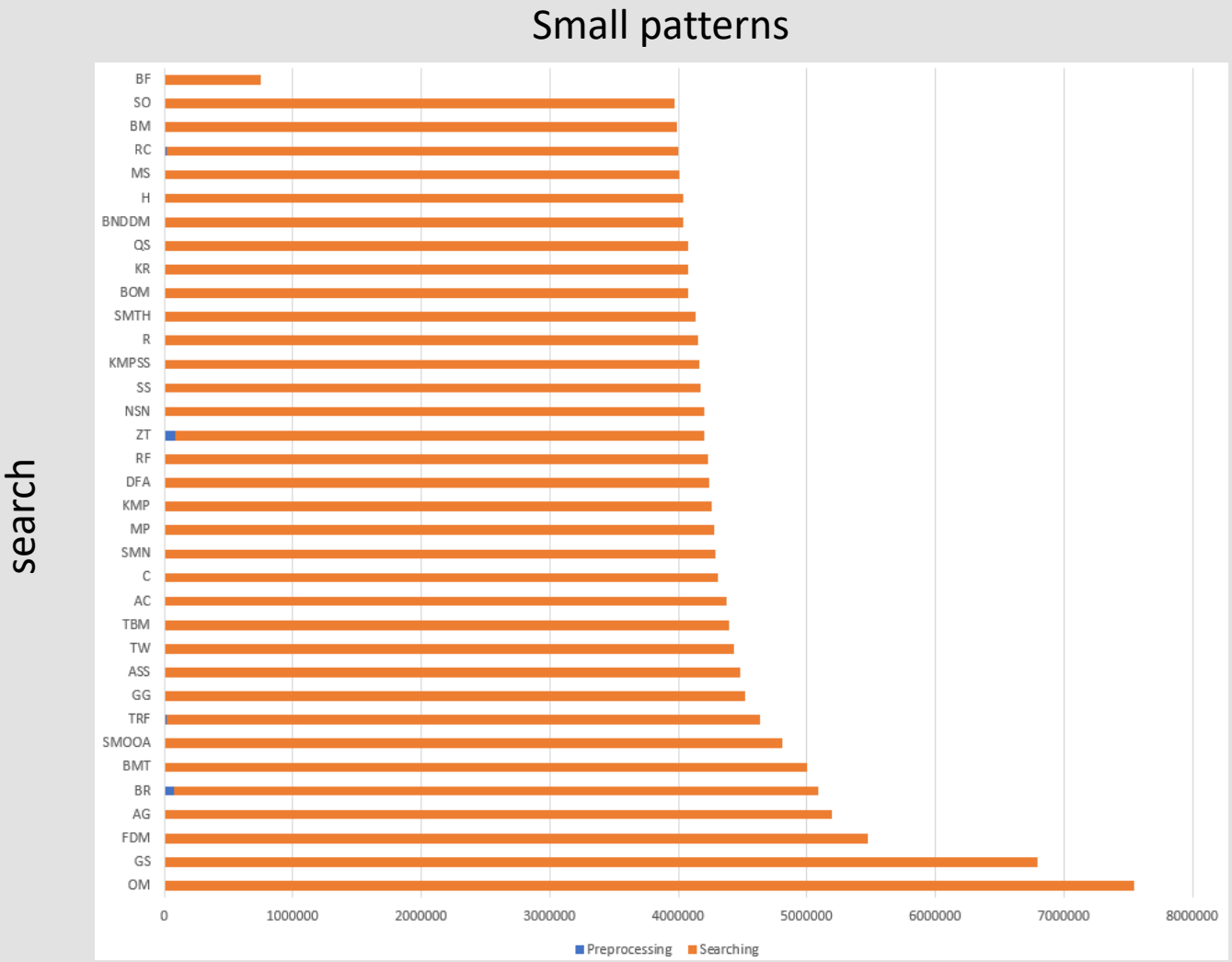
Benchmarking: Individual results, a surprising result!



Brute Force. For small alphabet and small patterns, lack of preprocessing phase thrives over shift tables and other preprocessing techniques.

Benchmarking: Collective results

Results. For each algorithm, the average time on all patterns are displayed for each of the four scenarios.



Benchmarking: Collective results - summary

Best performing algorithms on average of all tested pattern sizes

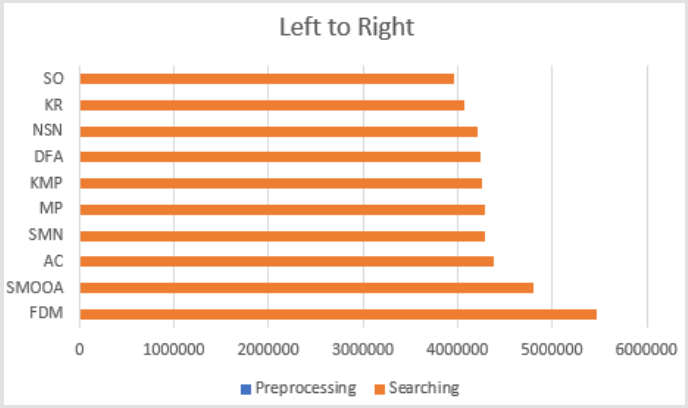
	Small patterns	Large patterns
search	Brute Force	Zhu-Takaoka
searchAll	Shift Or	Zhu-Takaoka

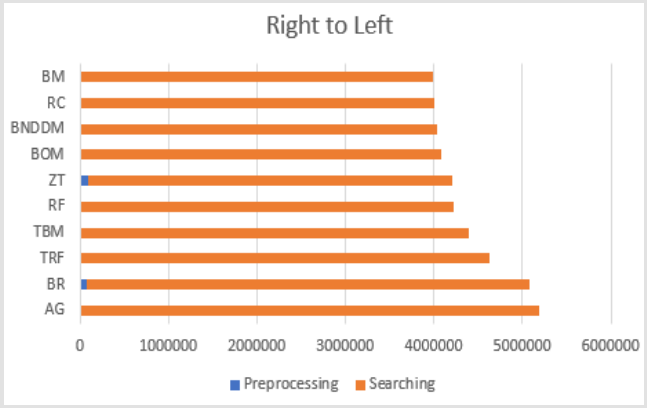
Remarks.

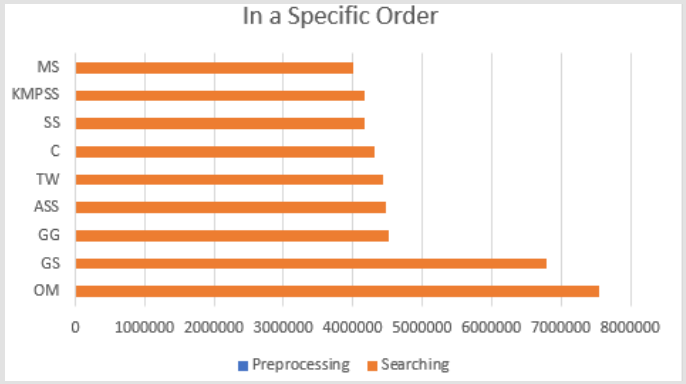
- In larger patterns, prevail algorithms that perform comparisons from right to left;
- Almost all algorithms of the Boyer-Moore family vastly outperform those of Knuth-Morris-Pratt family.

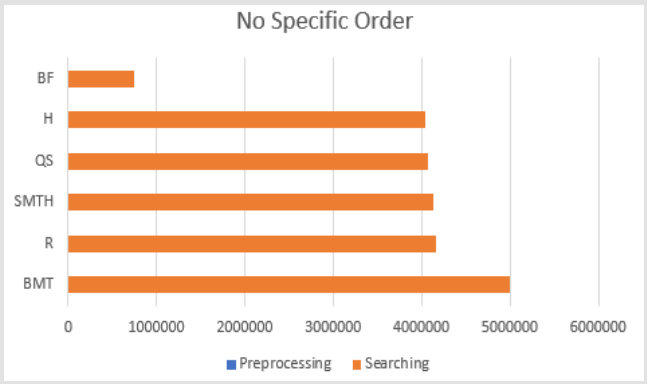
Benchmarking: Collective results (grouped)

Each scenario's collective result was grouped according to the character comparison order of each algorithm.









Search – small patterns

Visualization: Visualization Suite

Goal. Provide an animated glimpse of the functionality of a string search algorithm.

Implemented? HTML, CSS, JavaScript, jQuery

Advantages.

- Flexibility
- Adding a string search algorithm is simple.
- Easily and dynamically embeddable in web sites

Visualization: Visualization Suite

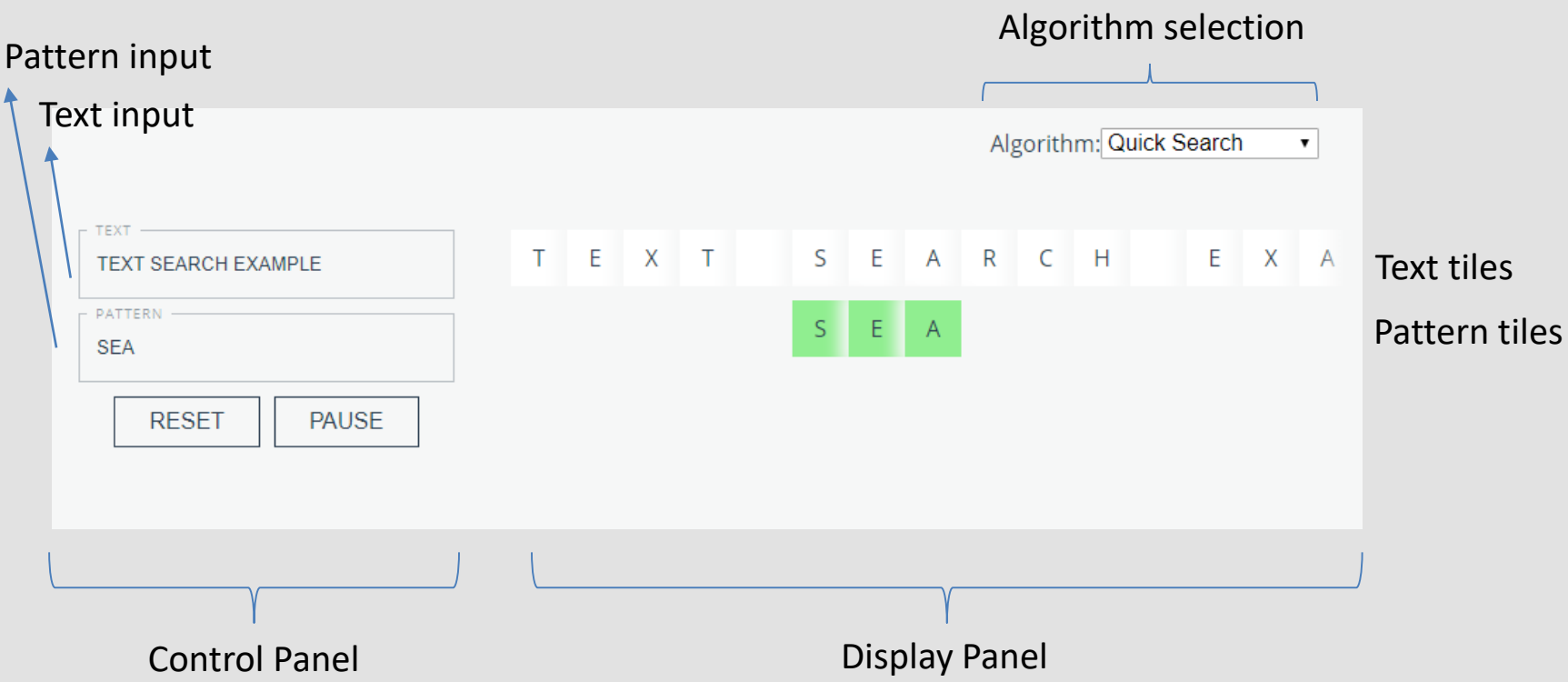
Basic Idea.

- All algorithms are ported to JavaScript.
- They can produce predefined queries describing the actions to be animated
- A JavaScript module acts as a hub that receives those queries; interprets them with respective animations.

Remarks.

- A resemblance of a controller-view model was implemented.
- A script adds dynamically functionality in all marked HTML elements.

Visualization: Visualization Suite Preview



Published with ***Surge*** tool as a static web site at esmaj.surge.sh

Conclusion

- 35 string search algorithms were ported/implemented in Java; the OO nature attempted to make the algorithms simpler to understand and reuse.
- We ranked their performance running benchmarks on huge biological sequence data
- Implemented a simple visualization suite to aid the visual understanding of their functionality.

Conclusion: Future Extensions

- Focus on optimizing some of the algorithms
- Use a bigger sample on the benchmarks (run all the possible substrings of a huge text-or several huge texts)
- Add more animations in the visualization suite

Thank you!

Any questions?