EXPERIMENTAL COMPARISO
STRING MATCHING ALGORIT

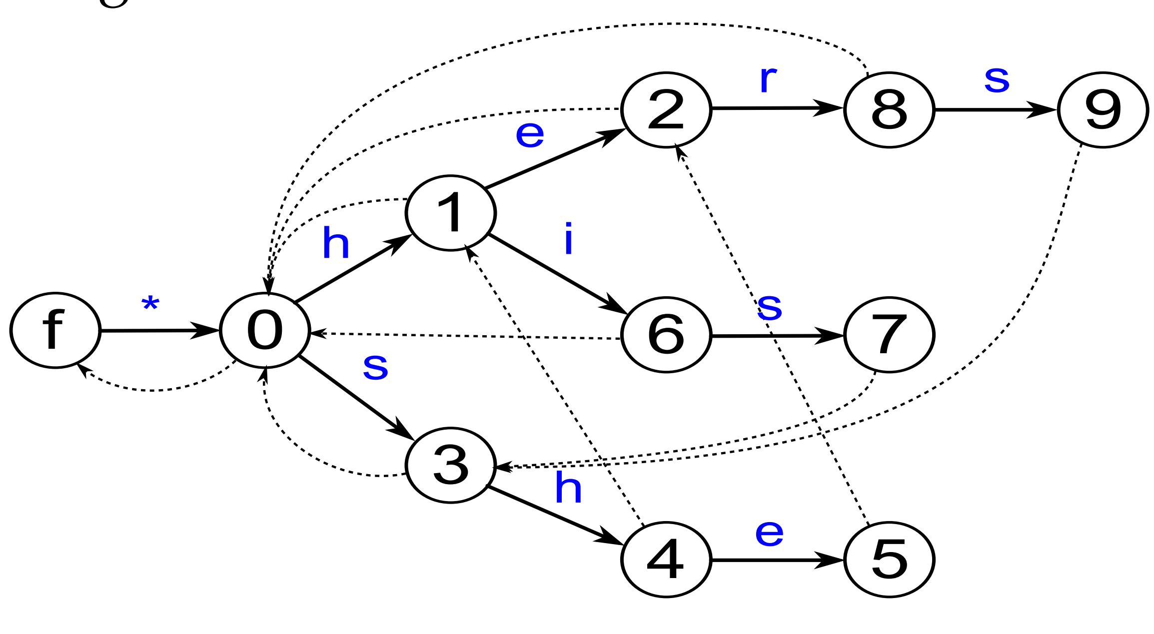
The *multiple exact string matching* problem asks us to search a text T o
search the text for each pattern in $\mathcal{P}$ using some exact string matching
an overview of three such algorithms: Aho-Corasick, Shift-And, and Kar
compare the theoretical performance of these algorithms to experimenta
a constant alphabet size of σ and equal-length patterns, $|P_j| = l$ for eve

THE ALGORITHMS

AHO-CORASICK

The algorithm. We can construct a trie from the patterns $\mathcal{P}$, then add a link to the node that represents the longest proper suffix of the node and make note of pattern occurrences for each node. The automaton can be then simulated with the text T as input to find occurrences.

Aho-Corasick automaton. For the patterns $\mathcal{P} = \{\text{he}, \text{she}, \text{his}, \text{hers}\}$ we can produce the following automaton:



SHIFT-AND

The algorithm. Shift-An
taining a bitvector D w
longest prefixes of patter

```
# preprocessing
for (i = 0; i < m;
  B[P[i/l][i%l]] +=
for (i = 0; i < m;
  pBegin += 2^i
for (i = l - 1; i
  pEnd += 2^i
# search
for (i = 0; i < n;
  D = ((D << 1) | p
  if D & pEnd ≠ 0:
```

Bitparallelism. If w is t
word, the bitvectors D, p
represented in $\lceil m/w \rceil$ m
stored in $\sigma \lceil m/w \rceil$ word

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI
MATEMAATTIS-LUONNONTIETEELLINEN TIEDEKUNTA
MATEMATISK-NATURVETENSKAPLIGA FAKULTETEN
FACULTY OF SCIENCE

ON OF MULTIPLE EXACT THMS

Paul Saikko

of length n for a set of patterns $\mathcal{P}$ with total length $\|\mathcal{P}\| = m$. We could algorithm, but much better algorithms exist for this task. Here we give p-Rabin. Each provides a different approach to solving the problem. We l results and identify some of their strenghts and weaknesses, assuming ry $P_j \in \mathcal{P}$.

d scans the text, it main-hich keeps track of the rns it has encountered.

```
 i++):
=  2^i
 i += l):

< m; i += l):


 i++):
pBegin) & B[T[i]]
 yield i
```

the length of a machine pBegin, and pEnd can be nachine words. B can be s and D can be updated

KARP-RABIN

Karp-Rabin hash function. For some fixed positive integers r and q, the karp-rabin hash of a string $S = s_0 s_1 \ldots s_{m-1}$ is

$$H(S) = \sum_{i=0}^{m-1} (s_i r^{m-1-i}) \mod q.$$

This is an example of a rolling hash function – if we know the hash $H(T_{[i \ldots i+m]})$, we can compute $H(T_{[i+1 \ldots i+1+m]})$ in constant time.

Multiple string matching. We can precompute the hash value for every pattern and store them in a data structure that supports constant-time lookups. Potential occurrences in the text can be found by computing $H(T_{[i \ldots i+l]})$ for each $i \in [0 \ldots n-l)$, and comparing them to the precomputed pattern hashes. Every potential occurrence must be checked, which leads to poor worst-case

Time complexity.

- Preprocessing: $O(m)$
- Search: $O(n)$

EXPERIMENTAL RESULTS

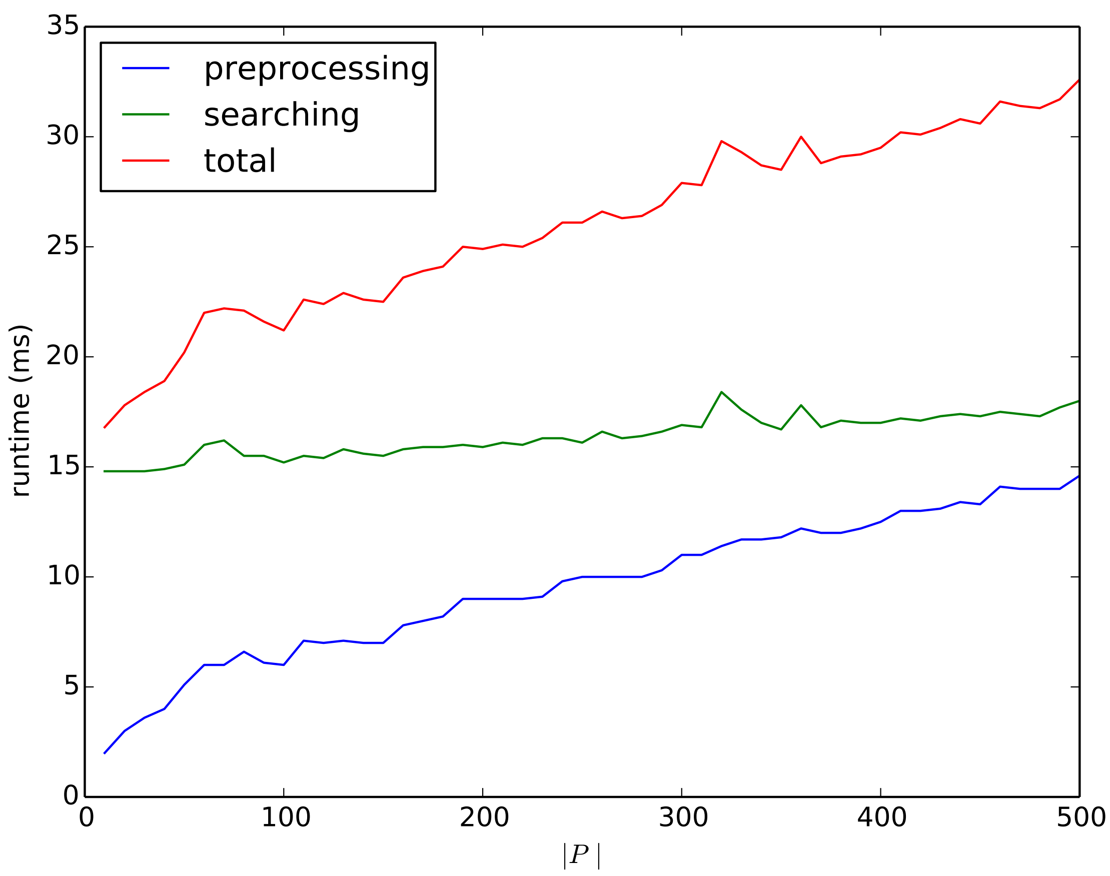
METHODOLOGY

Implementations. The algorithms shown above were implemented with java version 1.7. The source code for the implementations is available on github at `https://github.com/psaikko/string-algorithms-project`.

Experiments. The experiment data was gathered using python scripts to automate running the algorithms. Every plotted data point is an average of 10 runs of the algorithm.

COMPARISON WITH THEORETICAL PERFORMANCE

With constant text length. We plot the preprocessing and search times for each algorithm as the number of patterns to search for is increased.

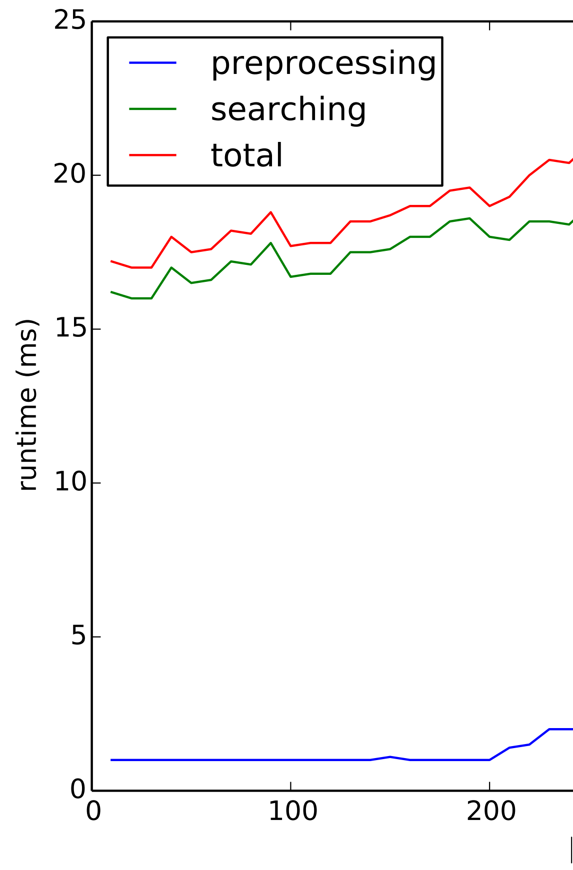


Aho-Corasick. We see search time staying roughly constant as expected, while preprocessing time increases linearly as we add more patterns.

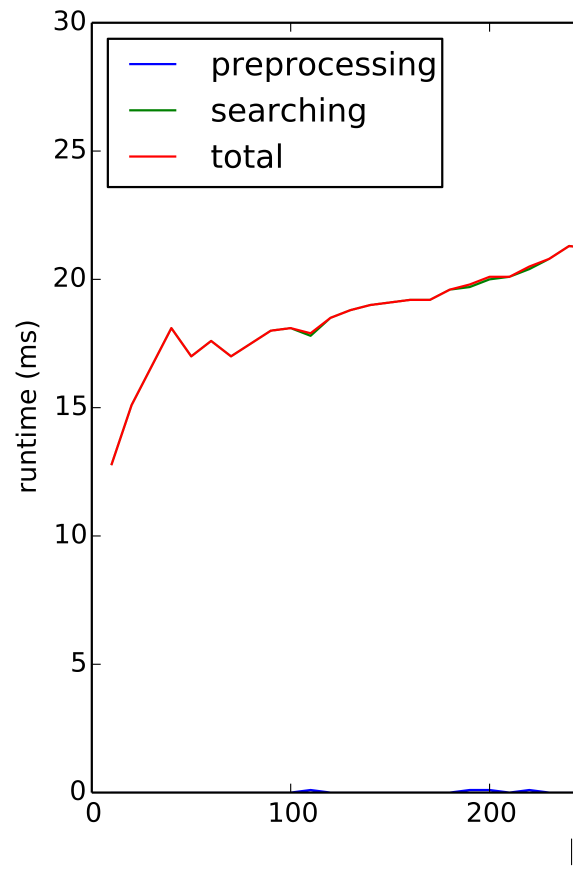
in $O(\lceil m/w \rceil)$ bitwise op

Time complexity.

- Preprocessing: $O(m$
- Search: $O(n\lceil m/w \rceil)$



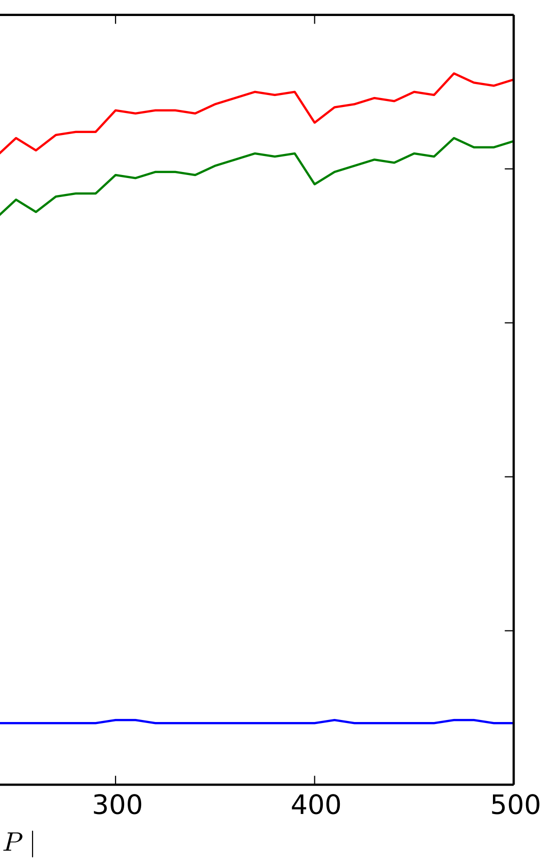
Karp-Rabin. The algori
well despite poor worst-
preprocessing is quite fa



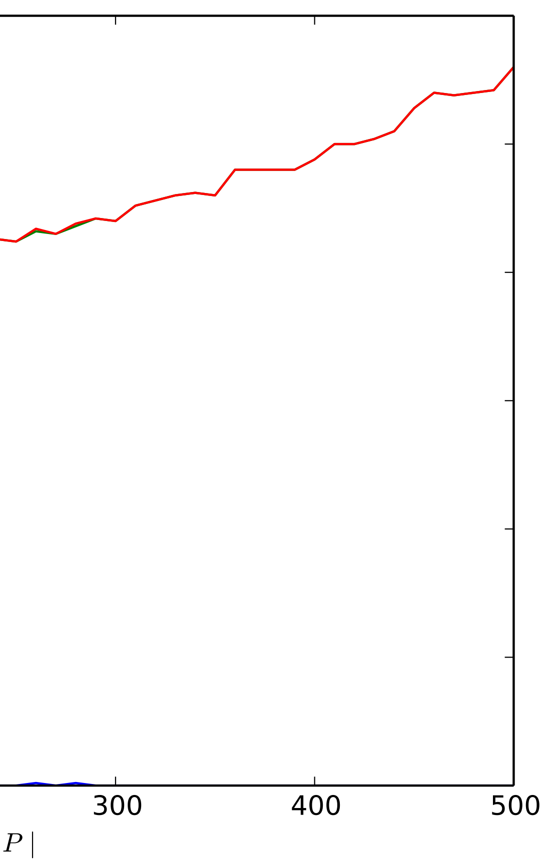
Shift-And. The preproce
is much shorter than th
spite the same asymptot

erations at each step.

)



thm generally performs
case behavior for search,
st.



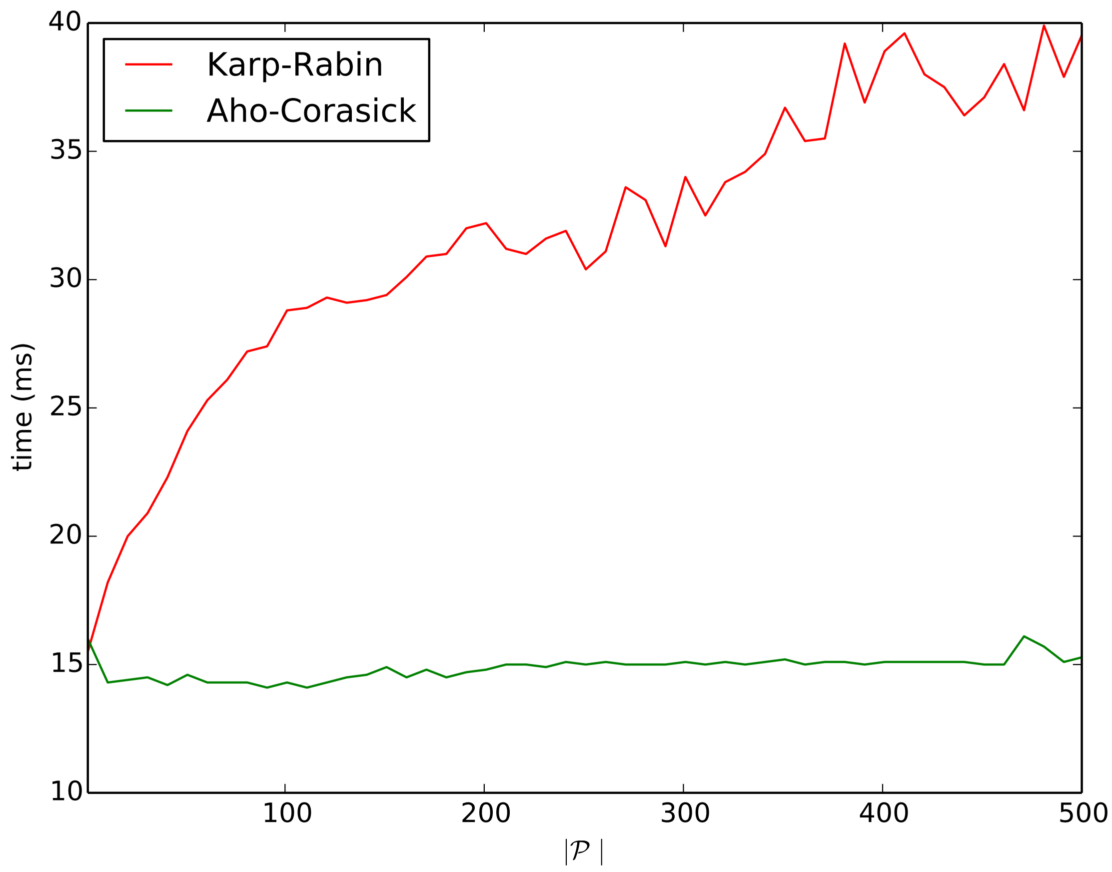
essing time for Shift-And
e other algorithms, de-
ic time complexity.

behavior.

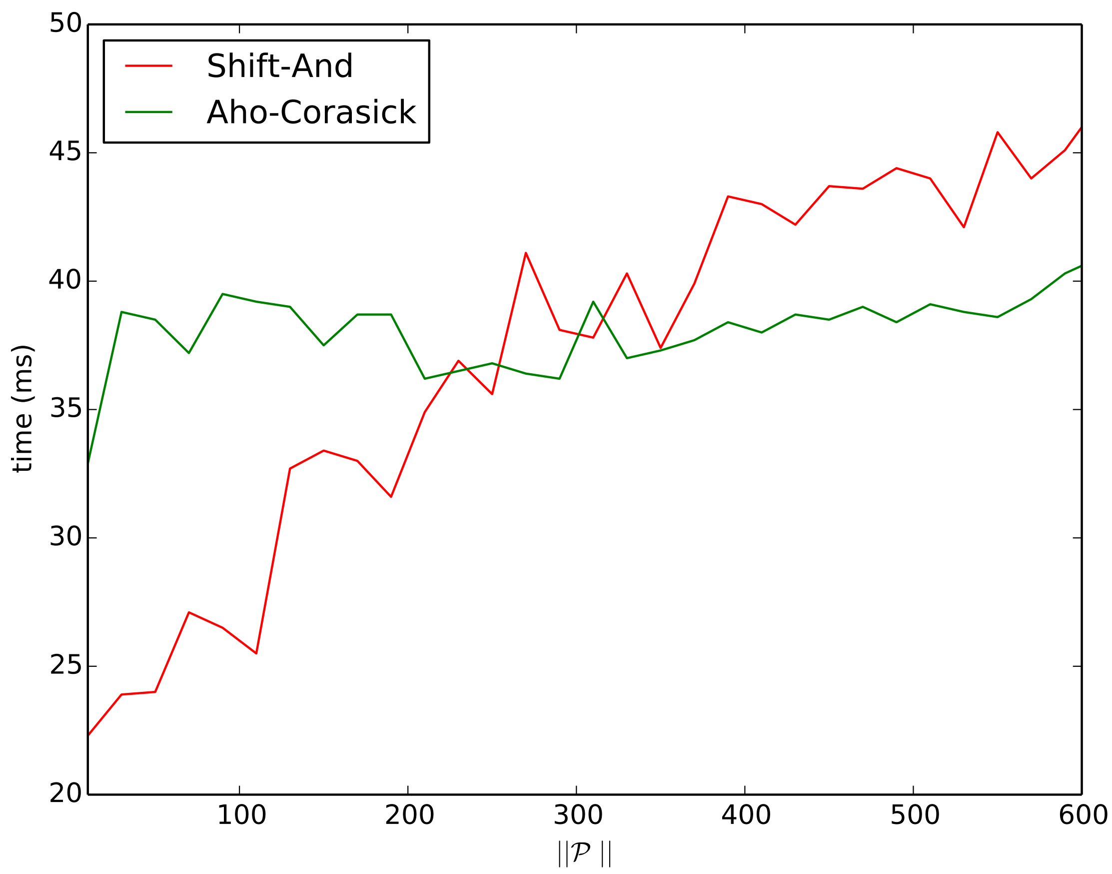
Average time complexity.

- Preprocessing: $O(m)$
- Search: $O(n+m)$

EDGE CASES



Many short patterns. A combination of the Karp-Rabin algorithm's $O(n+m)$ time complexity and hash collisions leads it to perform poorly.



Small total pattern length. Although Shift-And scales poorly with total pattern length, the bit-parallel algorithm is fast when $\|\mathcal{P}\|$ is not large.