Improved Single and Multiple Approximate String Matching

Kimmo Fredriksson

Department of Computer Science, University of Joensuu, Finland

Gonzalo Navarro

Department of Computer Science, University of Chile

The Problem Setting & Complexity

- Given *text* $T_{1..n}$ and *pattern* $P_{1..m}$ over some finite alphabet Σ of size σ, find the *approximate* occurrences of P from T, allowing at most k differences (edit operations).
- Exact matching (single pattern) lower bound: $\Omega(n \log_\sigma m/m)$ character comparisons (Yao, 79).
- Approximate matcing lower bound: $\Omega(n(k + \log_\sigma m)/m)$ (Chang & Marr, 94).
- We will search simultaneously a *set* $\mathcal{P} = \{P_1, P_2, ..., P_r\}$ of r patterns.
- $\Omega(n(k + \log_\sigma rm)/m)$ lower bound for r patterns (Fredriksson & Navarro, 2003)

Previous work

- Only a few algorithms exist for multipattern approximate searching under the k differences model.
- Naïve approach: search the r patterns separately, using any of the single pattern search algorithms.
- (Muth & Manber, 1996): $O(m(r+n))$ average time algorithm using $\Omega(m^2 r)$ space. The algorithm is based on hashing, and works only for $k = 1$.
- (Baeza-Yates & Navarro, 1997):
  - Partitioning into exact search: $O(n)$ on average ($O(rm)$ preprocessing), but can be improved to $O(k \log_\sigma(rm)n/m)$. Works for $k/m < 1/\log_\sigma(rm)$.
  - Other less interesting ones.

Previous work

- (Fredriksson & Navarro, 2003): The first average-optimal algorithm.
  - average-optimal $O(n(k + \log_\sigma rm)/m)$ up to error level $k/m < 1/3$.
  - linear $O(n)$ on average up to error level $k/m < 1/2$.
- (Hyyrö, Fredriksson & Navarro, 2004): $O(n\lceil r/\lfloor w/m \rfloor\rceil)$ worst case for short patterns, where w is the number of bits in machine word.

This work

- We have improved the (optimal) algorithm of (Fredriksson & Navarro, 2003)
  - Faster in practice, and...
  - ...allows error levels up to $k/m < 1/2$.
- Our algorithm runs in $O(n(k + \log rm)/m)$ average time, which is optimal.
- Preprocessing time is $O(rm\sigma^\ell/w)$, and the algorithm needs $O(r\sigma^\ell)$ space, where $\ell = \Omega(\log_\sigma(rm))$.
- The fastest algorithm in practice for intermediate k/m and small σ.

The method in brief:

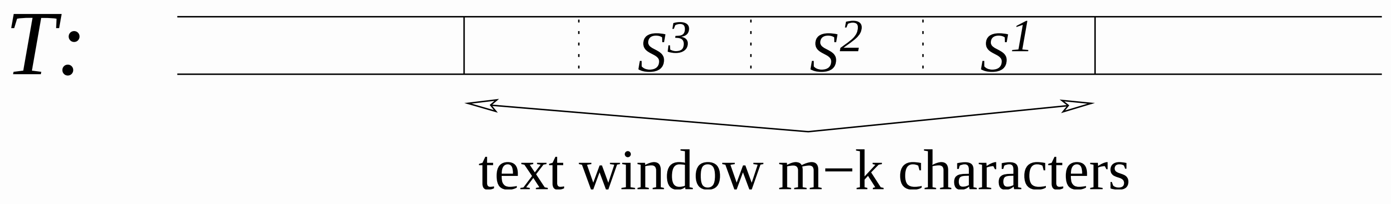
- The algorithm is based on the preprocessing/filtering/verification paradigm.
- The preprocessing phase generates all σ^ℓ strings of lenght ℓ, and computes their minimum distance over the set of patterns.
- The filtering phase searches (approximately) text ℓ-grams from the patterns, using the precomputed distance table, accumulating the differences.
- The verification phase uses dynamic programming algorithm, and is applied to each pattern separately.

Preprocessing

- Build a table D as follows:
  1. Choose a number ℓ in the range $1 < \ell \leq m - k$
  2. For every string S of length ℓ (ℓ–gram), search for S in $\forall P \in \mathcal{P}$
  3. Store in $D[S]$ the smallest number of differences needed to match S inside $\mathcal{P}$ (a number between 0 and ℓ).
- D requires space for σ^ℓ entries and can be computed in $O(rm\sigma^\ell/w)$ time.

Filtering

- Any occurrence is at least $m - k$ characters long $\Rightarrow$ use a sliding window of $m - k$ characters over T
- Invariant: all occurreces starting before the window are already reported.
- Read ℓ-grams $S^1, S^2, ..., S^u$ from the text window, from right to left:

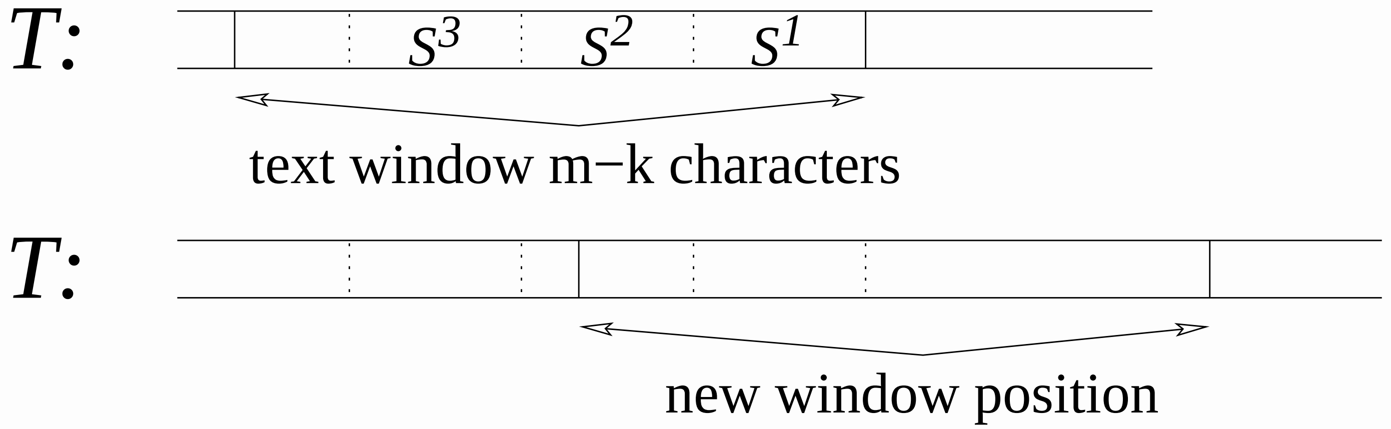


- Any occurrence starting at the beginning of the window must contain all the ℓ-grams read.

Filtering

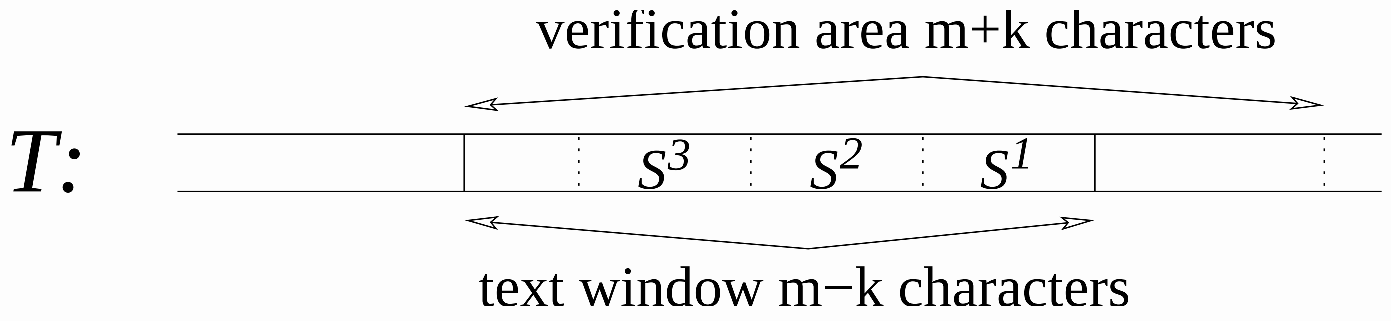
- Accumulate a sum of necessary differences: $M_u = \sum_{1 \le t \le u} D[S^t]$.
- If M_u becomes $> k$ for some (i.e. the smallest) u, then no occurrence can contain the ℓ-grams S^u : ... : S^2 : S^1 $\Rightarrow$ slide the window past the first character of S^u.

E.g. $D[S^1] + D[S^2] > k$:



Verification

- If $M_u \leq k$, then the window might contain an occurrence
  $\Rightarrow$
  the occurrence can be $m + k$ characters long, so verify the area $T_{i..i+m+k-1}$, where i is the starting position of the window



- The verification is done for each of the r patterns, using standard dynamic programming algorithm.

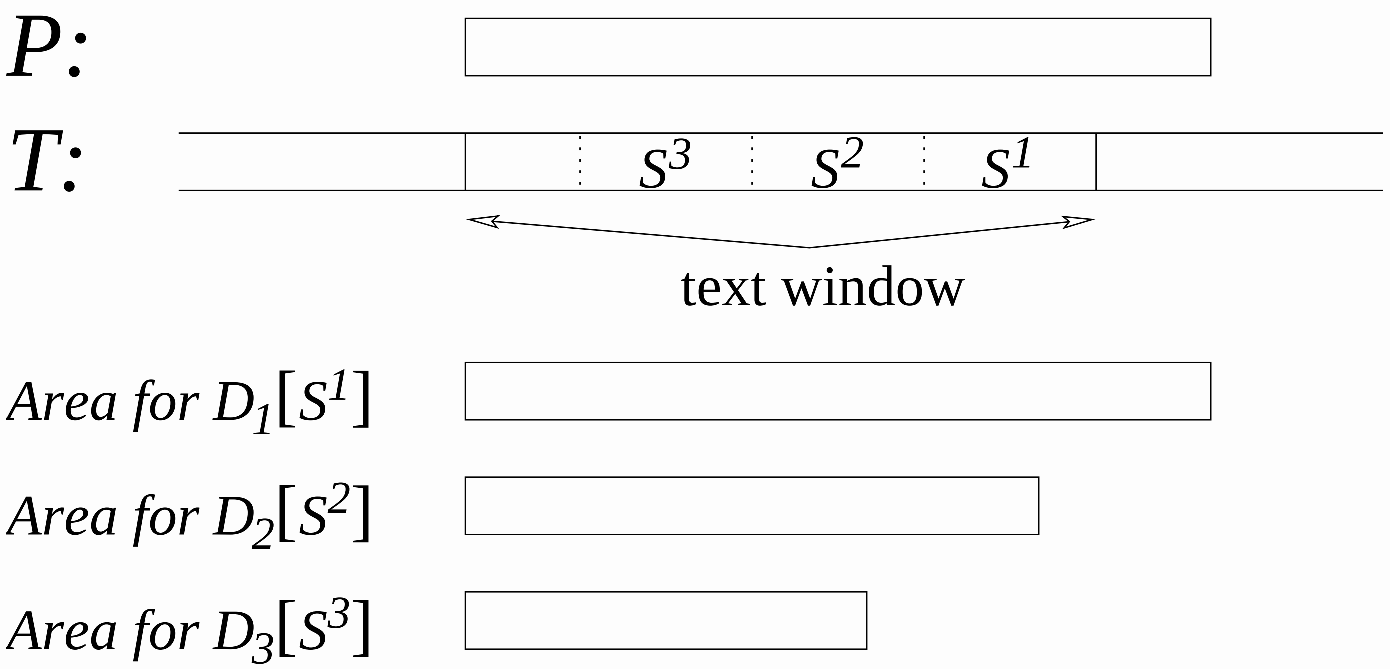
Stricter matching condition

- Our basic algorithm: text ℓ-grams can match anywhere inside the patterns.
  $\Rightarrow$
  If $M_u > k$, then we know that no occurrence can contain the ℓ-grams $S^u : ... : S^1$ in any position.
- The matching area can be made smaller without losing this property.

Stricter matching condition

- Consider an approximate occurrence of $S^2 : S^1$ inside the pattern.
  - S^2 cannot be closer than ℓ positions from the end of the pattern.
    $\Rightarrow$
    For S^2 precompute a table D_2, which considers its best match in the area $P_{1 \ldots m-\ell}$ rather than $P_{1 \ldots m}$.
  - In general, for S^t preprocess a table D_t, using the area $P_{1 \ldots m-(t-1)\ell}$
  - Compute M_u as $\sum_{t=1}^{u} D_t[S^t]$

Stricter matching condition



Stricter matching condition

- $D_t[S] \geq D[S]$ for any t and S
  $\Rightarrow$
  the smallest u that permits shifting the window is never smaller than for the basic method.
  $\Rightarrow$
  this variant never examines more ℓ-grams, verifies more windows, nor shifts less.
- Drawback: needs more space and preprocessing effort
  $\Rightarrow$
  Can be slower in practice.
- The matching condition can be made even stricter
  - Work less per window...
  - ...but the shift can be smaller.

Analysis

- It can be shown that the basic algorithm has optimal average case complexity $O(n(k + \log_\sigma rm)/m)$.

  This holds for $k/m < 1/2 - O(1/\sqrt{\sigma})$.
- The worst case complexity can be made $O(n + rkn)$ (filtering $+$ verification).
- The preprocessing cost is $O(m^5 r^3 \sigma^{O(1)})$, and it requires $O(m^4 r^2 \sigma^{O(1)})$ space.
- Since the algorithm with the stricter matching condition is never worse than the basic version, it is also optimal.

Analysis

- For a single pattern our complexity is the same as the algorithm of Chang & Marr, i.e. $O(n(k + \log_\sigma m)/m)$...
- ...but our filter works up to $k/m < 1/2 - O(1/\sqrt{\sigma})$, whereas the filter of Chang & Marr works only up to $k/m < 1/3 - O(1/\sqrt{\sigma})$.

Experimental results

- Implementation in C, compiled using `icc 7.1` with full optimizations, run in a 2GHz Pentium 4, with 512MB RAM, running Linux 2.4.18.
- Experiments for alphabet sizes $\sigma = 4$ (DNA) and $\sigma = 20$ (proteins), both random and real texts.
- Text lengths were 64Mb, and patterns 64 characters.
- In the implementation we used several practical improvements described in (Fredriksson & Navarro, 2003)
  - Bit-parallel counters
  - Hierarchical / bit-parallel verification

Experimental results

- We used $\ell = 8$ for DNA, and $\ell = 3$ for proteins.
  - the maximum values we can use in practice, otherwise the preprocessing cost becomes too high.
- Analytical results:
  - $\ell = 12 \ldots 20$ for DNA, and $\ell = 6 \ldots 10$ for proteins (depending on r).
    $\Rightarrow$
    Altought our algorithms are fast, in practice they cannot cope with as high difference ratios as predicted by the analysis.

Experimental results

- Comparison against:

CM: Our previous optimal filtering algorithm

LT: Our previous linear time filter

EXP: Partitioning into exact search

MM: Muth & Manber algorithm, works only for $k = 1$

ABNDM: Approximate BNDM algorithm, a single pattern approximate search algorithm extending classical BDM.

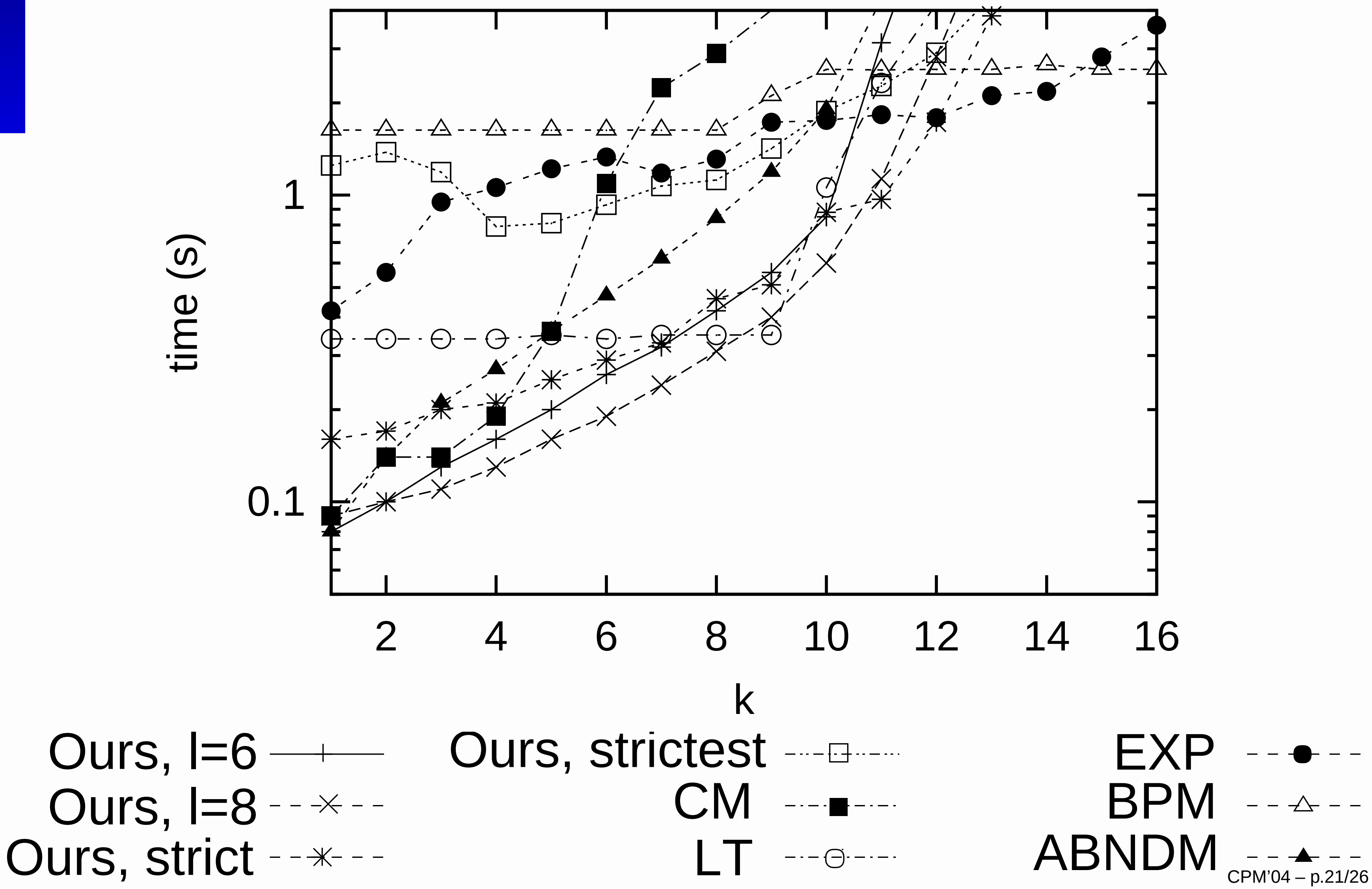
BPM: Bit-parallel Myers, currently the best non-filtering algorithm for single patterns.

Experimental results

- Comparison against Muth and Manber ($k = 1$):

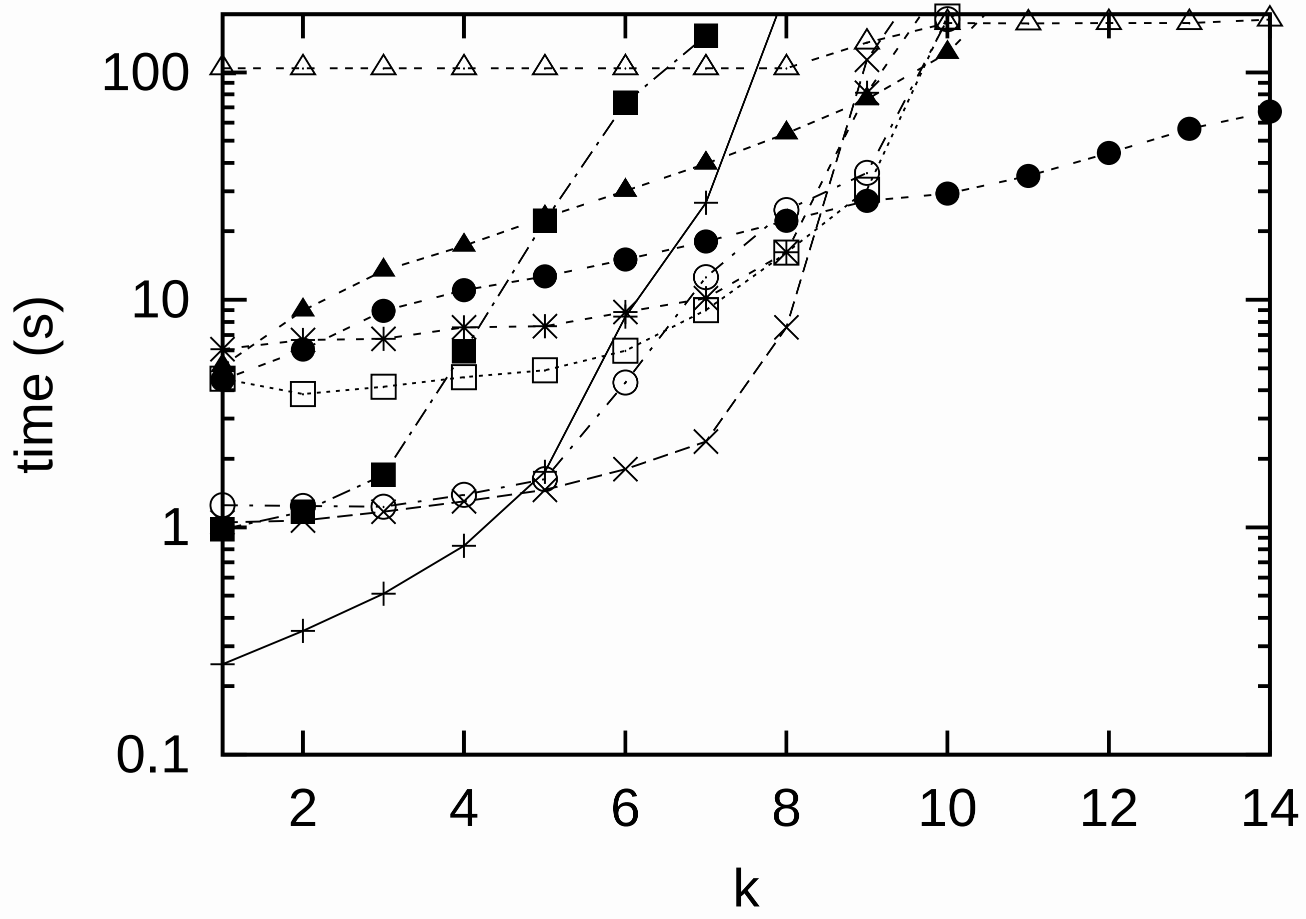
	$r = 1$	$r = 16$	$r = 64$	$r = 256$
Alg.	DNA			
MM	1.30	3.97	12.86	42.52
Ours	0.08	0.12	0.21	0.54
Alg.	proteins			
MM	1.17	1.19	1.26	2.33
Ours	0.08	0.11	0.18	0.59

Experimental results
$r = 1$, *random DNA*

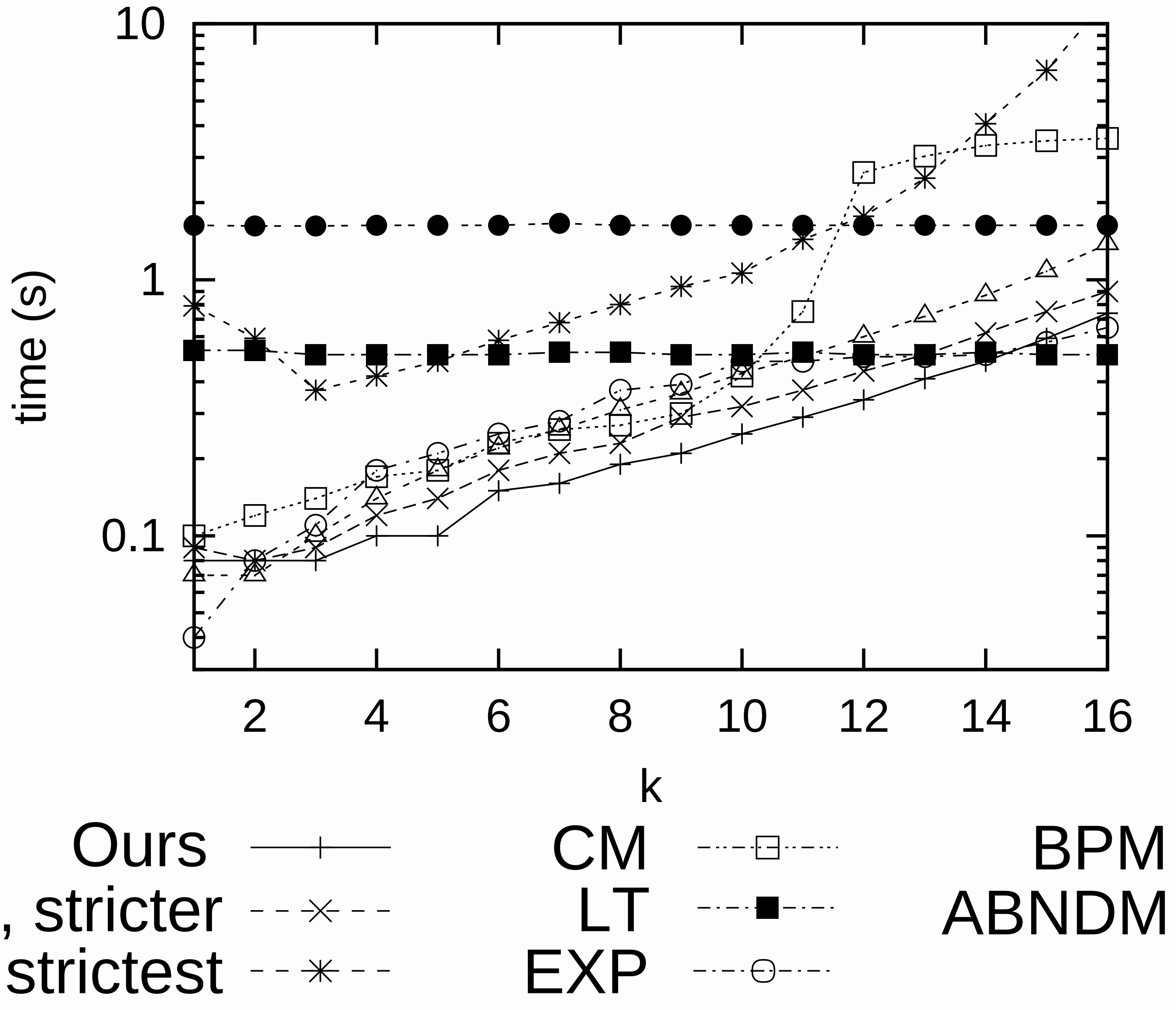


Experimental results

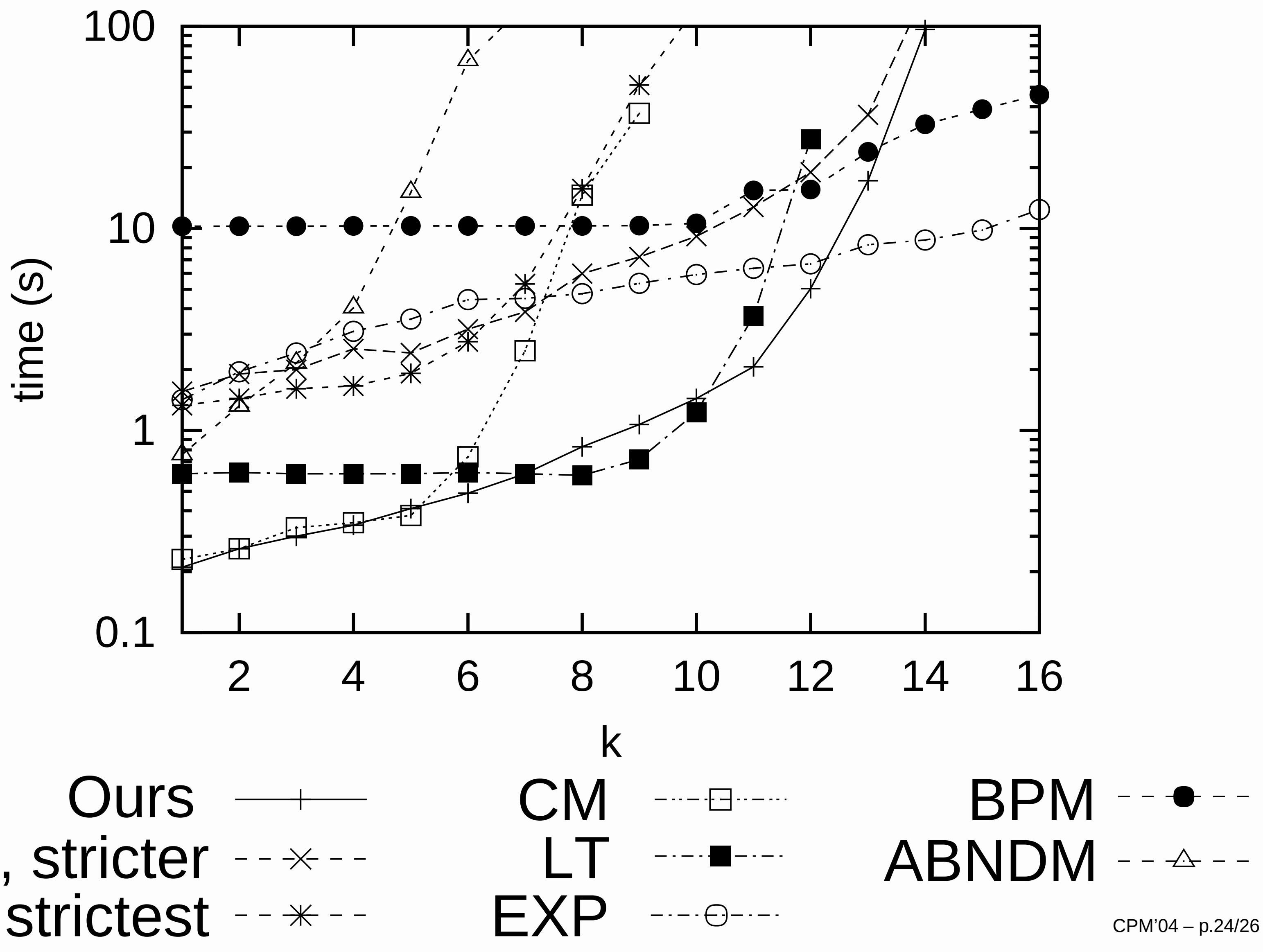
$r = 64$, *random DNA*



Experimental results
$r = 1$, ***random proteins***

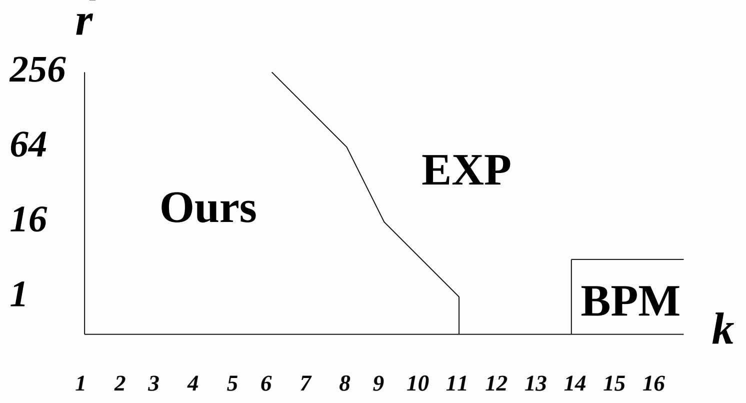


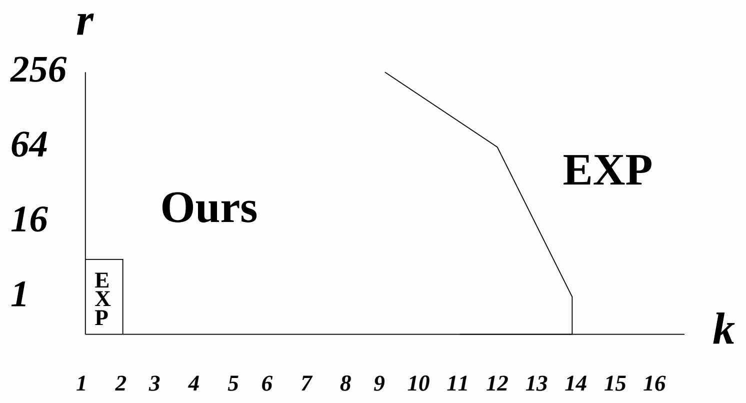
Experimental results
$r = 64$, *random proteins*

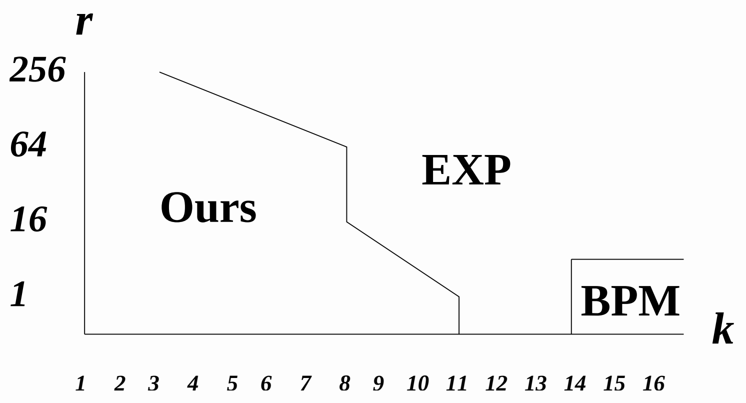


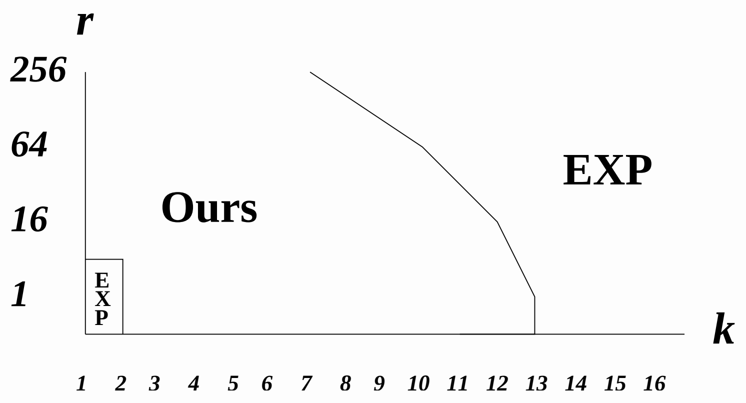
Experimental results

- Areas where each algorithm performs best. From left to right, DNA ($m = 64$), and proteins ($m = 64$). Top row: random data. bottom row: real data.









Conclusions

- Our new algorithm becomes the fastest for low k.
- The larger r, the smaller k values are tolerated.
- When applied to just one pattern, our algorithm becomes the fastest for low difference ratios.
- Our basic algorithm usually beats the extensions.
  - True only if we use the same parameter ℓ for both algorithms.
  - For limited memory we can use the stricter matching condition with smaller ℓ, and beat the basic algorithm
- Our algorithm would be favored on even longer texts (relative preprocessing cost decreases).