Average-Optimal Multiple Approximate String Matching

Kimmo Fredriksson

Department of Computer Science

University of Joensuu

kfredrik@cs.joensuu.fi

Gonzalo Navarro

Department of Computer Science

University of Chile

gnavarro@dcc.uchile.cl

The Problem Setting & Complexity

- The classic approximate string matching problem:

  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 (insertion, deletion, substitution).

- We will search simultaneously for a *set* $\mathcal{P} = \{P_1, P_2, ..., P_r\}$ of r patterns.

- Applications: virus and intrusion detection, spelling, text retrieval (synonyms and thesauri), computational biology, batched searching, ...

- On average at least $\Omega(n(k + \log_\sigma m)/m)$ character comparisons for *approximate* matching of one pattern (Chang & Marr, 94).

- $\Omega(n(k + \log_\sigma rm)/m)$ for r patterns (this work).

Previous work

- Only a few algorithms exist for multipattern approximate searching under the k differences model.
- Naïve approach: search for 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 low and medium error levels, $k/m < 1/\log_\sigma(rm)$. Best in practice.
  - Other less interesting ones.

Main Results

- We have generalized the (optimal) algorithm of (Chang & Marr, 1994) to multiple patterns.
- We show that the lower bound for the problem is $\Omega(n(k+\log_\sigma(rm))/m)$.
- Our algorithm runs in $O(n(k+\log_\sigma(rm))/m)$ average time, which is optimal. This holds for $k/m < 1/3 - O(1/\sqrt{\sigma})$.
- Another $O(n)$ average time algorithm for $k/m < 1/2 - O(1/\sqrt{\sigma})$.
- Several improvements to the basic algorithms $\Rightarrow$ very efficient in practice too.
- Preprocessing time is $O(r^3m^5)$, and space is $O(r^2m^4)$.
- In theory: best algorithm for low and medium error levels, few or many patterns, filling an important gap.
- In practice: best algorithm for low error levels, few or many patterns, displacing partitioning algorithm to intermediate error levels.

The lower bound

- The lower bound for approximate string matching for one pattern:

  1. (Yao, 79): The text is divided into blocks of length $2m-1$

     $\Rightarrow$

     There are m possible positions for the pattern to occur.
     In order to discard all these positions, at least $\Omega(\log_\sigma m)$ character comparison are needed on average

     $\Rightarrow$

     At least $\Omega(n \log_\sigma(m)/m)$ comparisons to solve the *exact* string matching problem.

  2. (Chang & Marr, 94) generalized this for *approximate* matching: At least $k+1$ charcters has to be examined to skip the block. This leads to lower bound $\Omega(kn/m)$.

  Summing over these two basic facts gives $\Omega(n(k+\log_\sigma(m))/m)$ lower bound (Chang & Marr, 94).

- We adapt this to multipattern matching.
  - The idea is the same as in the Yao's proof.

    We again divide the text in blocks of length $2m-1$, but now there are rm positions to discard.

    This needs at least $\Omega(\log_\sigma(rm))$ character inspections on average.

    $\Rightarrow$

    $\Omega(n \log_\sigma(rm)/m)$ lower bound for exact searching.
  - Summing again leads to $\Omega(n(k+\log_\sigma(rm))/m)$ lower bound for approximate searching.
- Interestingly, there exists several optimal exact multipattern search algorithms (e.g MultiBDM (Crochemore & Rytter, 94)), but the optimality wasn't proved before.

The Basic Algorithm: Preprocessing

- Build a table D as follows:
  1. Choose a number ℓ in the range $1 \leq \ell \leq \lceil (m-k)/2 \rceil$
  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 $O(rm\sigma^\ell)$ time.

The Basic Algorithm: Filtering

- Divide the text into non-overlapping blocks of length

$$b = \lceil (m-k)/2 \rceil.$$

Observation: any approximate occurrence of P_i, which has length of at least $m-k$, contains at least one whole block.

We want to find this block. E.g. for $m = 14$ and $k = 2 \Rightarrow b = 6$

P:

T:

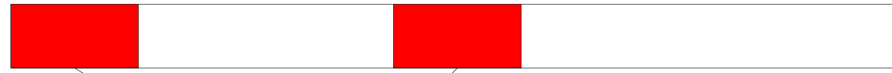
The Basic Algorithm: Filtering

- Each block $T[ib+1 \ldots ib+b]$ is processed as follows.

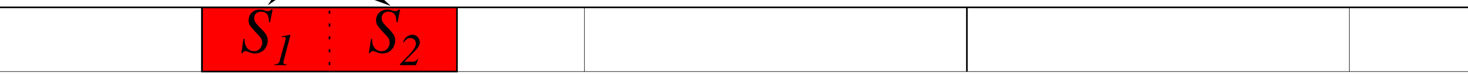
  1. Take the first ℓ–gram of the block, $S_1 = T[ib+1 \ldots ib+\ell]$, and obtain $D[S_1]$.
  2. Take the next ℓ–gram, $S_2 = T[ib+\ell+1 \ldots ib+2\ell]$, and obtain $D[S_2]$, and so on.
  3. If, before reaching the end of the block, the sum $\sum_{1 \leq j \leq t} D[S_j] > k$, the block can be skipped:
     – No occurrence of P_i can contain the block, as merely matching those t ℓ–grams *anywhere* inside P_i requires more than k differences.
  4. If at the end of the block, the sum $\sum_{1 \leq j \leq t} D[S_j] \leq k$, the block must be verified.

$\ell = 2$, $m = 14$ and $k = 2 \Rightarrow b = 6$,

P:



T:



$D[S_1] = 2$

$D[S_2] = 1$

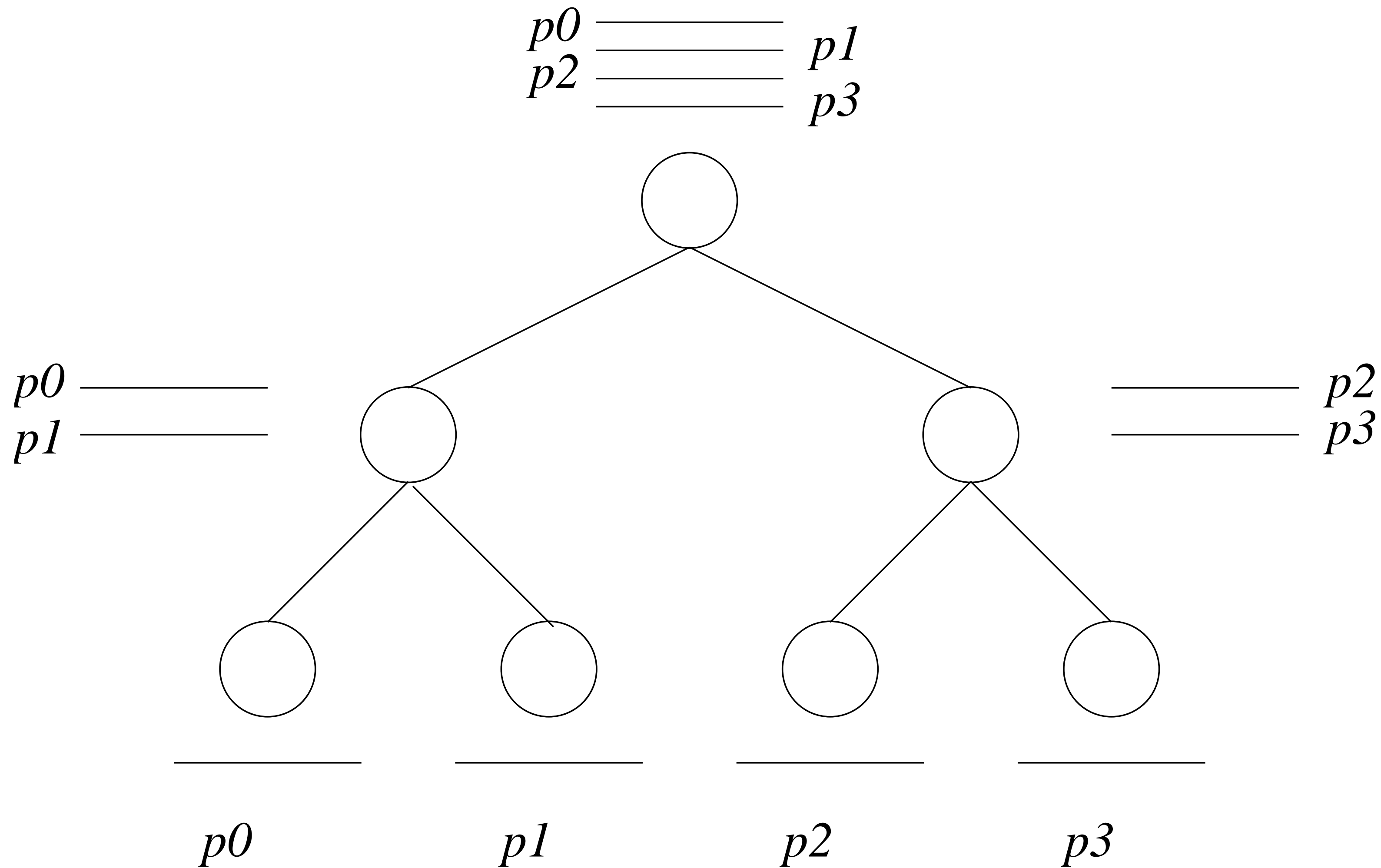
$\Sigma\, D[S_i] > k$ ⟶ skip the block

The Basic Algorithm: Verification

- For each block $T[ib+1 \ldots ib+b]$ that has to be verified run the classical dynamic programming algorithm over $T[ib+1-m-k+b \ldots ib+m+k]$ for $\forall P_i \in \mathcal{P}$.
- This requires worst case time $O(m^2)$ for each pattern.
- Can be improved to $O(m^2/w)$ average time per pattern by using bit-parallelism (Myers, 99).

Extensions: (binary) pattern hierarchy

- Build a binary tree for the patterns.
  - The root node contains all the patterns.
  - Each child node contains only the half of the patterns of its parent.
  - Only one pattern in each leaf.
- Build the table D for each node, taking only the corresponding patterns into account.
- Filter recursively (starting from the root node):
  1. Filter the current block with the table D for the current node.
  2. If the filter triggers "verification", recursively filter with D for the children.
  3. If the current node is leaf, then verify with dynamic programming.



- In practice much faster except for very small k/m.
- But it needs more space.

Bit-parallel counters & higher arity pattern hierarchy

- If we allow at most k errors, then $B = \lceil \log_2(k + \ell + 1) \rceil = O(\log_2 k)$ bits is enough for the counter $\sum_{1 \leq j \leq t} D[S_j]$.
- We manage several counters in a single integer and update them all in one shot.
- This increases tree arity, reduces verification time and space.

Analysis

- The filtering and verification take only

$$O\left(\frac{n(k+\log_\sigma(rm))}{m}\right)$$

  total time on average, which is optimal. This holds for

$$k/m < 1/3 - O(1/\sqrt{\sigma}).$$

- The preprocessing cost is $O(rm\sigma^\ell/w)$, which is $O(m^5r^3\sigma^{O(1)}/w)$ for the optimal

$$\ell = (4\log_\sigma m + 2\log_\sigma r)/d$$

  (where $0 < d < 1$ is a constant, if we take k/m as a constant).

  The space requirement is $\sigma^\ell = m^4r^2\sigma^{O(1)}$.

Linear time algorithm for higher error levels

- The algorithm cannot cope with difference ratios k/m beyond 1/3.
- This is partly due to the use of fixed text blocks of length $(m-k)/2$.

  $\Rightarrow$ Use a *sliding window* of *overlapping* blocks of length $m-k$

  $\Rightarrow$ Difference ratios up to $k/m < 1/2 - O(1/\sqrt{\sigma})$
- A sliding window of t ℓ-grams, where $t = \lfloor (m-k+1)/\ell \rfloor - 1$.

  $\Rightarrow$ Blocks overlap with each other by $t-1$ ℓ-grams.
- Consider text blocks for the form $T[i\ell+1 \ldots i\ell+t\ell]$.

  $\Rightarrow$ Every occurrence (whose min. length is $m-k$) contains a complete block.

  $\Rightarrow$ If the ℓ-grams inside the window add up more than k differences, we can move to the next block.
- The result is an algorithm that takes $O(n)$ time for $k/m < 1/2 - O(1/\sqrt{\sigma})$.

Experimental results

- Implementation in C, compiled using `gcc 3.2.1` with full optimizations, run in 2GHz Pentium 4, with 512MB RAM, with Linux 2.4.
- Experiments for alphabet sizes $\sigma = 4$ (DNA), $\sigma = 20$ (protein) and $\sigma = 256$ (ASCII text).
- The test data for DNA and protein alphabets was randomly generated.
  - 64MB characters for DNA,
  - 16MB characters for protein and ASCII,
- The patterns were 64 characters long.
- The texts were stored used only 2 (DNA) and 5 bits (protein) per character, which allowed $O(1)$ time access to the ℓ-grams.

Preprocessing times in seconds:

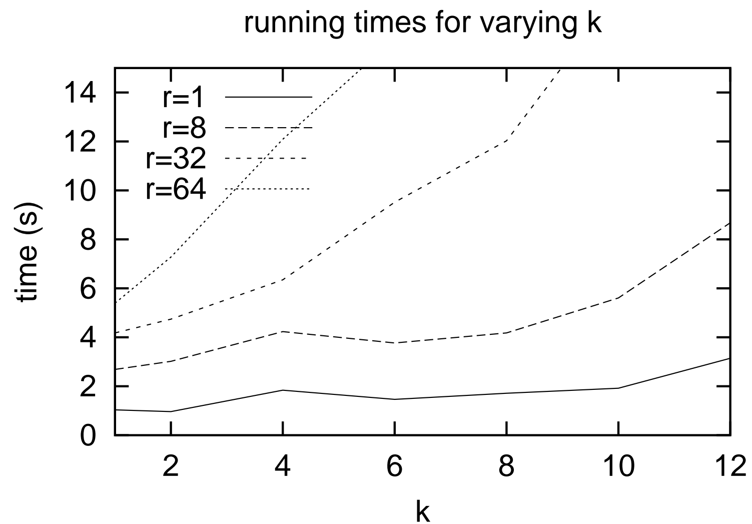
DNA	$r = 1$	$r = 8$	$r = 32$	$r = 64$
$\ell = 4$	0.00	0.00	0.00	0.00
$\ell = 6$	0.01	0.01	0.04	0.08
$\ell = 8$	0.02	0.15	0.58	1.17
$\ell = 10$	0.38	3.02	12.00	24.19

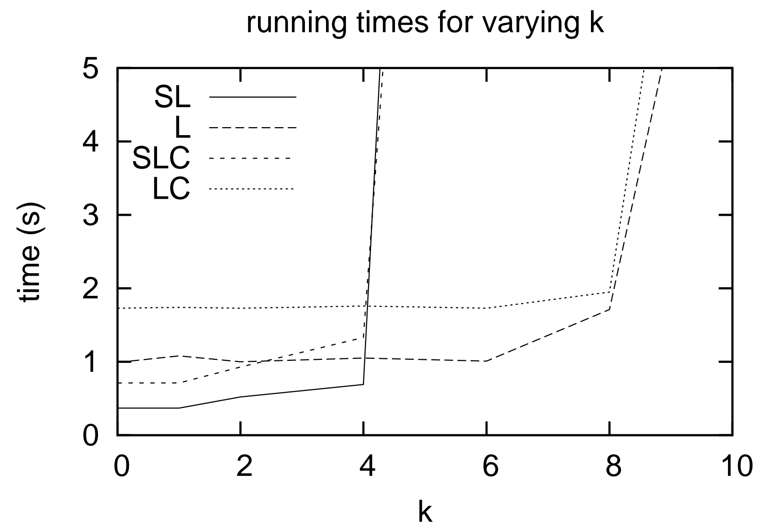
protein	$r = 1$	$r = 64$	$r = 256$	$r = 1024$
$\ell = 1$	0.00	0.01	0.01	0.02
$\ell = 2$	0.00	0.01	0.03	0.07
$\ell = 3$	0.01	0.09	0.40	1.55
$\ell = 4$	0.04	6.09		

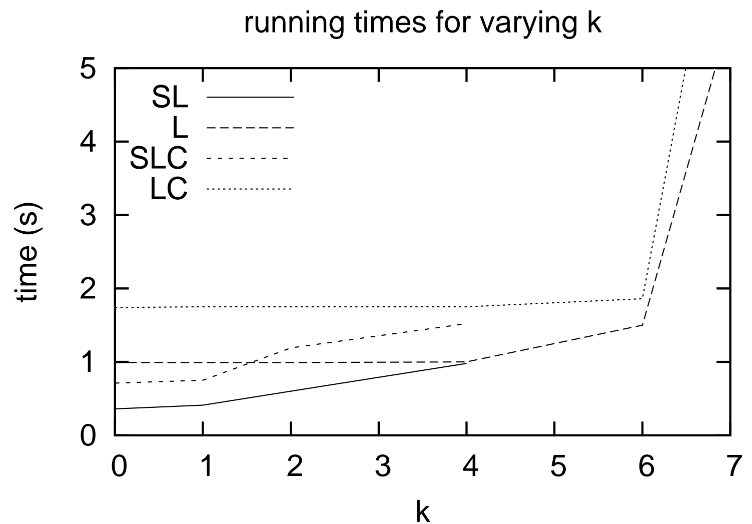
ASCII	$r = 1$	$r = 64$	$r = 256$	$r = 1024$
$\ell = 1$	0.00	0.01	0.01	0.06
$\ell = 2$	0.01	0.59	2.60	10.65
$\ell = 3$	4.26			

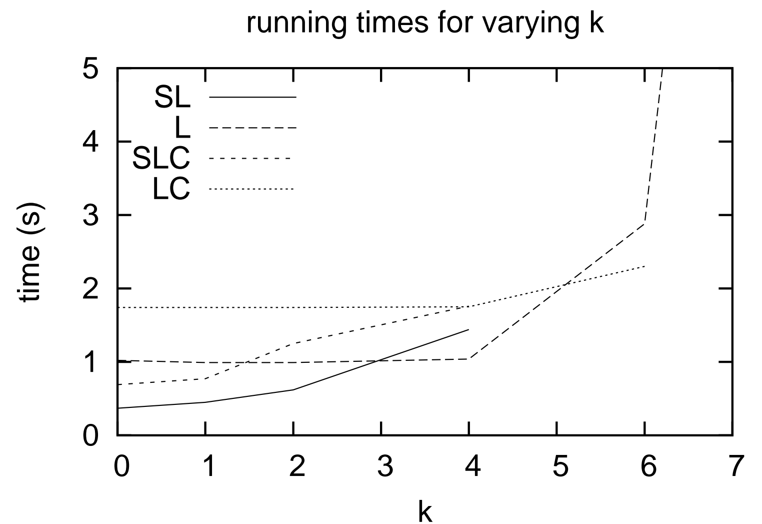
We cannot use the optimal ℓ values! 18 (DNA), 10 (proteins), and 6 (ASCII).

Search times (s) for DNA, for EP and for $\ell = 8$ with $r = 8$, $r = 32$, and $r = 64$.

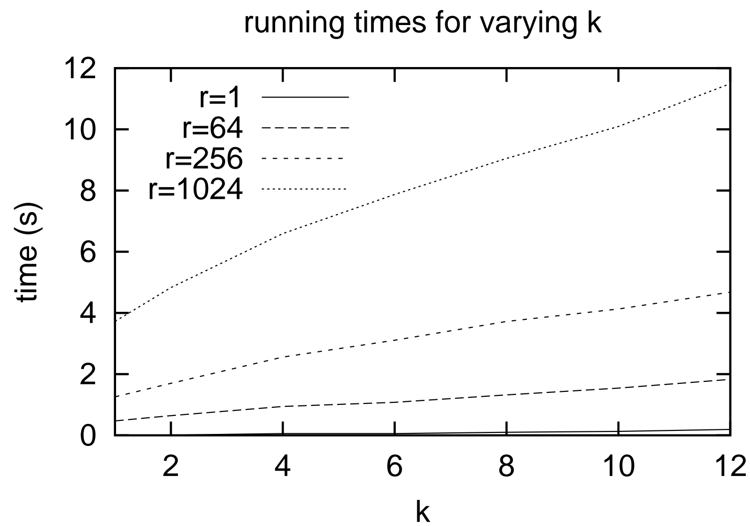


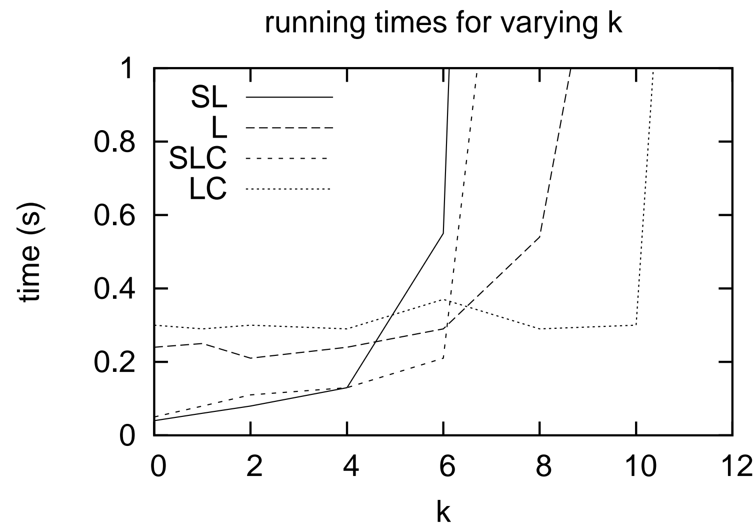


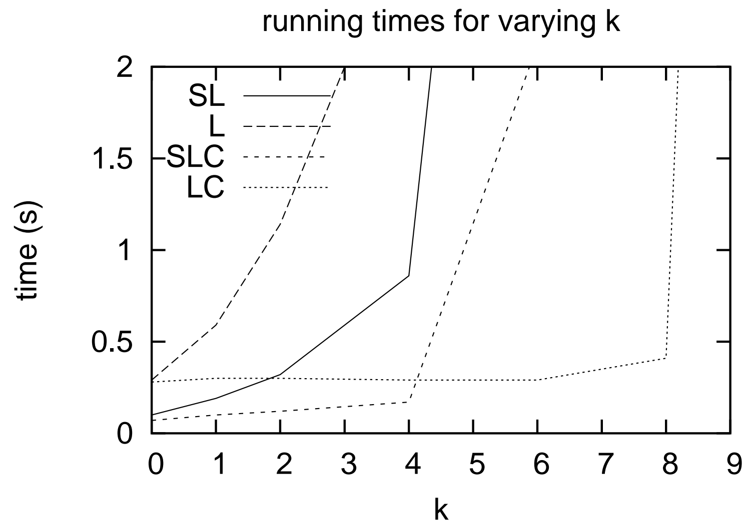


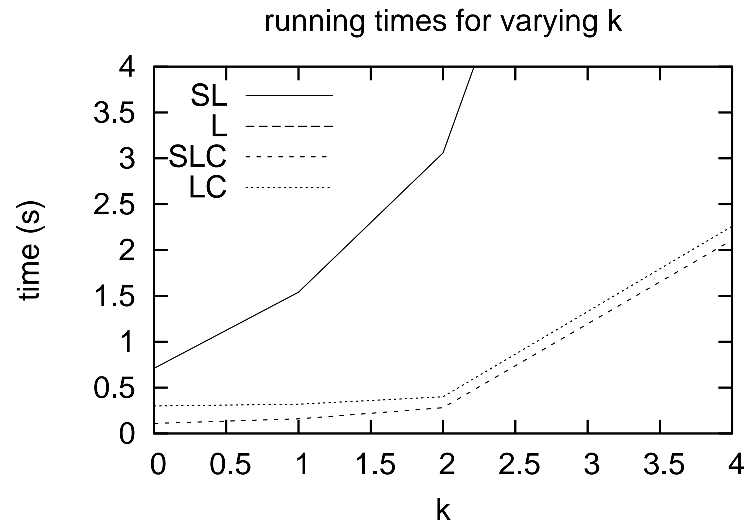


Times for protein, for EP and for $\ell = 3$, with $r = 64$, $r = 256$, and $r = 1024$.

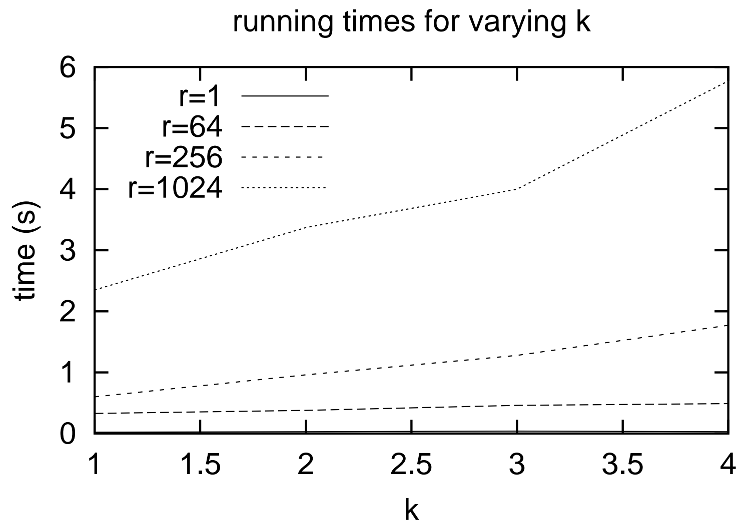


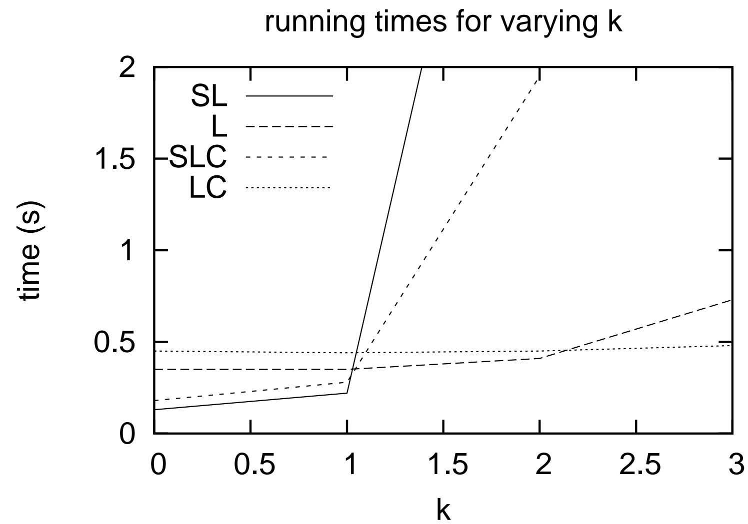


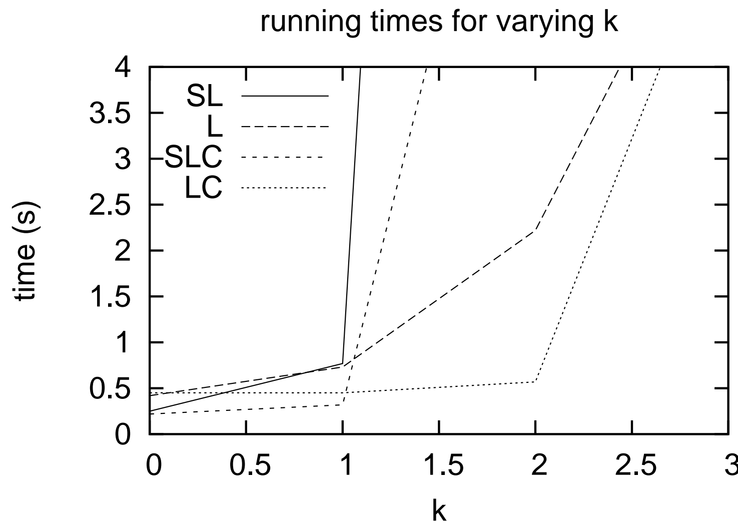


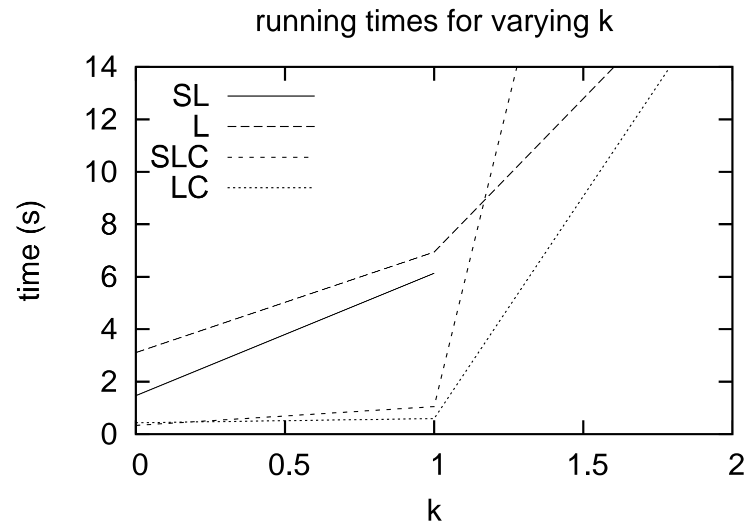


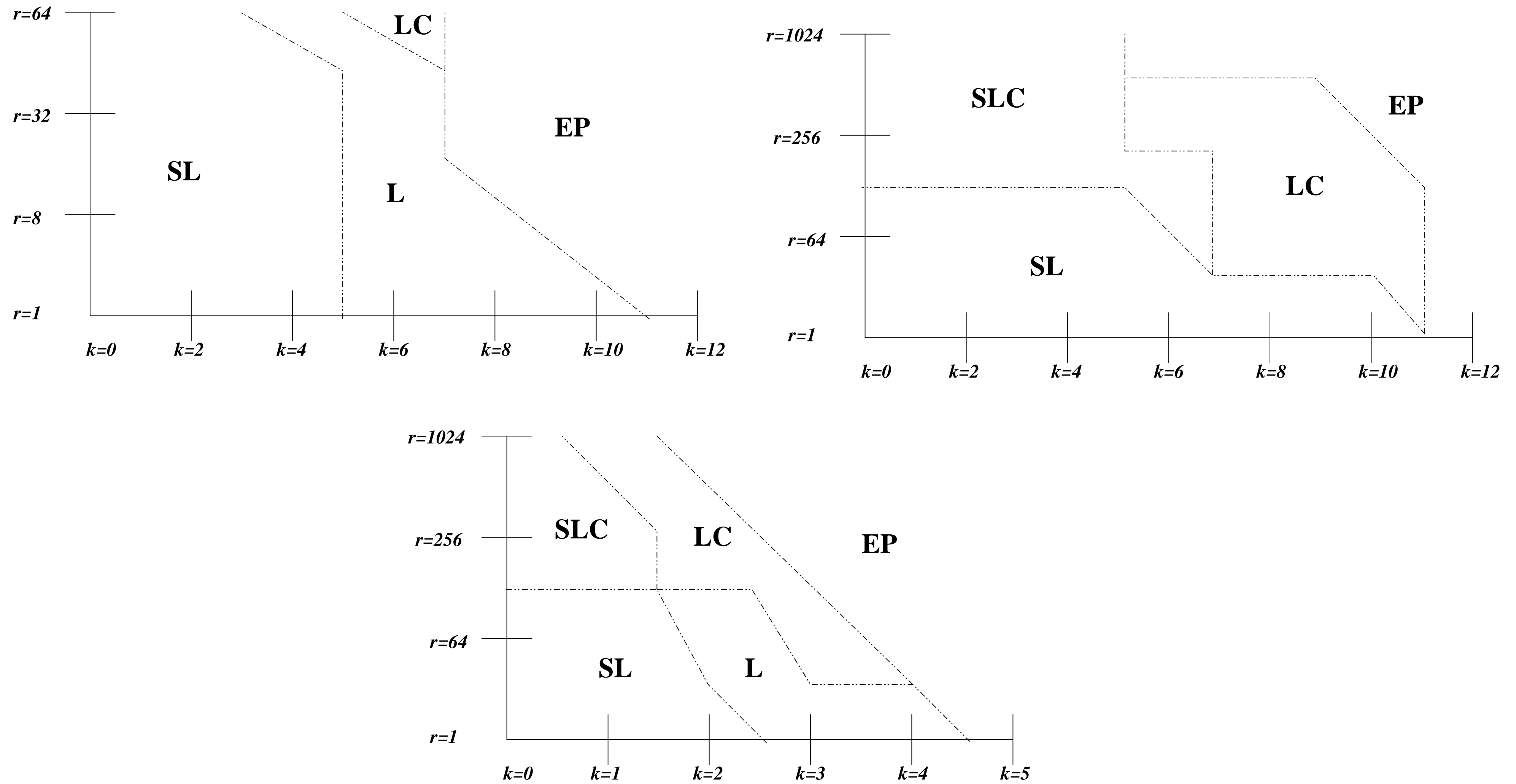
Times for ASCII, for EP and for $\ell = 3$ with $r = 64$, $r = 256$, and $r = 1024$.











(DNA, proteins, and ASCII)

Conclusions...

- We have proved lower bounds for exact and approximate multipattern string matching.
- We have presented a new algorithm for multiple approximate string matching, that is not only optimal on average, but also practical.
- A second algorithm is slower but handles higher difference ratios.
- We have shown that they perform well in handling large numbers of patterns and low difference ratios.
- Currently the best algorithms for low difference ratios (better for low alphabets), for searching 1 or many patterns.

...and future work

- The algorithms do not induce any order on the ℓ-grams
  - they can appear in any order, as long as their total distance is at most k.
  - require that the ℓ-grams from $\mathcal{P}$ must appear in approximately same order in T. Utilized in the LAQ algorithm (Sutinen & Tarhio, 1996)
- Reducing the preprocessing time and space (to use a larger ℓ)
  - Compute D only for those ℓ-grams that appear in a pattern with at most ℓ' differences, and assume that all the others appear with $\ell' + 1$ differences.
    $\Rightarrow$ only $O(rm(\sigma\ell)^{\ell'})$ time and space but more verifications...