Average complexity of exact and approximate multiple string matching☆

Gonzalo Navarro[a,*], Kimmo Fredriksson[b,1]

[a] *Department of Computer Science, University of Chile, Santiago, Chile*
[b] *Department of Computer Science, University of Joensuu, Finland*

Abstract

We show that the average number of characters examined to search for r random patterns of length m in a text of length n over a uniformly distributed alphabet of size σ cannot be less than $\Omega(n \log_\sigma(rm)/m)$. When we permit up to k insertions, deletions, and/or substitutions of characters in the occurrences of the patterns, the lower bound becomes $\Omega(n(k+\log_\sigma(rm))/m)$. This generalizes previous single-pattern lower bounds of Yao (for exact matching) and of Chang and Marr (for approximate matching), and proves the optimality of several existing multipattern search algorithms.

Keywords: Lower bounds; Yao's bound; String matching allowing differences; Multipattern matching

1. Introduction

String matching is one of the main problems in computer science, with applications in virtually every area. Given a pattern $P = p_1 \ldots p_m$ and a text $T = t_1 \ldots t_n$, both sequences over a finite alphabet Σ of size σ, the problem is to determine all the positions where P occurs in T, that is, $\{|x|, T = xPy\}$. The worst-case complexity of the problem,

☆ Partially supported by Fondecyt grant 1-020831.

[*] Corresponding author. Depto. de Ciencias de la Computacion, Universidad de Chile, Blanco Encalada 2120, Santiago, Chile.

E-mail addresses: gnavarro@dcc.uchile.cl (G. Navarro), kfredrik@cs.joensuu.fi (K. Fredriksson).

[1] Work developed while the author was working in the Department of Computer Science, University of Helsinki. Supported by the Academy of Finland.

measured in number of accesses to T, is clearly $\Omega(n)$, and it was achieved by the famous KMP algorithm [8]. If we assume that the text and the pattern characters are uniformly and independently chosen from Σ, then the classical paper of Yao [9] shows that the average complexity of the problem is $\Omega(n \log_\sigma(m)/m)$. This bound is tight, for example it is achieved by BDM algorithm [3]. Other algorithms, like TurboBDM and TurboRF [3], are at the same time worst-case optimal.

Approximate string matching is a variant of the problem where we do not require exact coincidence between P and its occurrences in T. Rather, a distance function d between strings is defined, and an error threshold k is given as an additional problem parameter. Then, the problem becomes finding all the approximate occurrences of P in T, that is, the substrings of T that are at distance at most k from P, $\{|x|, T = xP'y, d(P, P') \leqslant k\}$.

Some of the most commonly used distance functions are Hamming distance (number of character substitutions necessary to convert one string into the other), indel distance (number of character insertions and deletions necessary), and Levenshtein distance (number of character insertions, deletions, and substitutions). The worst-case complexity of this problem is $\Omega(n)$, and it can be achieved by reducing the problem to automata search, although preprocessing is exponential in k. The worst-case complexity when the space has to be polynomial in k is unknown, the best known upper bound being $O(kn)$ [7]. Interestingly, the average case complexity is known. Chang and Marr [2] proved that the average lower bound for this problem is $\Omega(n(k + \log_\sigma(m))/m)$ accesses to T. They focused on Levenshtein distance, although their proof is valid for the other two distances as well. Their bound is also tight, as they gave an algorithm with such average complexity in the same paper [2] (for $k/m < \frac{1}{3}$).

Multiple string matching is the problem of, given r patterns $P^1 \dots P^r$, and T, report all the occurrences of all the patterns in T. This problem arises naturally in many applications, and several algorithms exist to solve it. Again, the worst-case complexity is $\Omega(n)$ and it was achieved by the algorithm of Aho and Corasick [1]. If the minimum length of the patterns is m, then the best average complexity achieved by multipattern search algorithms is $O(n \log_\sigma(rm)/m)$, for example by Dawg–Match [4] and MultiBDM [5] algorithms.

Finally, we can define multiple approximate string matching by giving a single threshold k for all the r patterns. The best existing average complexity for this problem is $O(n(k + \log_\sigma(rm))/m)$, for $k/m < \frac{1}{3}$ [6].[2] In a recent work [10], the limit was improved to $k/m < \frac{1}{2}$ for single and multiple approximate pattern matching.

Somewhat surprisingly, despite the existence of several efficient algorithms, no average-case lower bounds have been given for multiple string matching, either exact or approximate. Hence it is not known whether the multipattern algorithms mentioned above are average-optimal.

In this paper, we answer this question affirmatively. We show that the average complexity of exact multipattern string matching is $\Omega(n \log_\sigma(rm)/m)$ accesses to T, for $r \leqslant \sigma^m/m$, and that of approximate multipattern string matching is $\Omega(n(k+\log_\sigma(rm))/m)$. For larger r, the lower bound becomes $\Omega(n)$, the same as for the worst case. For

[2] An incomplete optimality proof was sketched in that paper.

$r = \mathrm{O}(\mathrm{poly}(m))$, the bounds are equal to those for a single pattern. These are not surprising, as there exist algorithms that achieve the single pattern complexity for multipattern searching when r is that small. We assume for simplicity that all the patterns are of length m.

2. A lower bound for exact multipattern matching

We extend the classical proof of Yao [9] to the case of searching for several patterns. Let us assume that we search for r random patterns of length m in a random text of length n. More precisely, each character of each pattern, as well as each text character, is drawn with uniform probability from set Σ, independently of all the other characters. Note that the r patterns are not necessarily different from each other. We prove that, under this model, the average lower bound of the problem is $\Omega(n \log_\sigma(rm)/m)$ accesses to T.

Following [9], let us divide the text into $\lfloor n/(2m-1) \rfloor$ contiguous and non-overlapping blocks B_i, $i \in 1 \ldots \lfloor n/(2m-1) \rfloor$, such that $B_i = T_{(2m-1)(i-1)+1 \ldots (2m-1)i}$. Furthermore, assume that we only need to search for the presence of the patterns *inside* each block B_i. This is an optimistic assumption, since we are disregarding any pattern occurrence spanning two blocks. We also optimistically disregard the few last text characters that do not complete a full block. Hence, any lower bound derived for this simpler problem is a lower bound for the original search problem. We establish this fact in the following lemma.

Lemma 1. *The average lower bound to the problem of finding any occurrence of patterns $P^1 \ldots P^r$ inside blocks $B_i = T_{(2m-1)(i-1)+1 \ldots (2m-1)i}$, $i \in 1 \ldots \lfloor n/(2m-1) \rfloor$, is also a lower bound to the problem of finding any occurrence of patterns $P^1 \ldots P^r$ inside $T_{1 \ldots n}$.*

Proof. Any solution to the problem of searching T in time $\Theta(t)$ can be converted into a solution to the problem of searching the B_i's in time $\Theta(t)$, by removing from the result any occurrence $T_{j \ldots j+m-1}$ that spans more than one block, that is, such that $(j-1) \bmod (2m-1) \geqslant m$. The extra discarding work is constant per occurrence, and in time t no more than t occurrences can be reported, so the extra work is $\mathrm{O}(t)$. □

From now on, due to Lemma 1, we will focus on a lower bound to the problem of finding all the occurrences that lie inside some block B_i, $i \in 1 \ldots \lfloor n/(2m-1) \rfloor$. Our next goal is to show that we can actually focus on any single block and multiply the result by $\Omega(n/m)$.

Note that the blocks do not overlap each other. Since each text character is independent of all others, no characters read inside one block can be used to gather information on the others. Therefore, we can regard the search problem inside each block B_i in isolation. Let t_i be the search cost inside block B_i, without any extra information. We remark that random variables t_i are independent of each other, although we do not actually need this for the proof. The following lemma easily follows.

Lemma 2. *The total search time inside blocks B_i, $i \in 1 \ldots \lfloor n/(2m-1) \rfloor$, is on average equal to $\lfloor n/(2m-1) \rfloor$ times the average search time inside any such block.*

Proof. The total search time inside the blocks is by definition $t_1 + t_2 + \ldots t_{\lfloor n/(2m-1) \rfloor}$. Hence, the average of this search cost is

$$E\left(\sum_{i=1}^{\lfloor n/(2m-1) \rfloor} t_i \right) = \sum_{i=1}^{\lfloor n/(2m-1) \rfloor} E(t_i),$$

because the expectation commutes with the sum (for independent as well as dependent random variables). Since all the blocks B_i are identically distributed, independently of each other and uniformly over strings in the set Σ^{2m-1}, it turns out that $E(t_i) = E(t_j)$ for any $1 \leqslant i,\ j \leqslant \lfloor n/(2m-1) \rfloor$. Let us call $E(t)$ this value, which corresponds to the average search time inside a block B_i, for any i. Therefore, the average search cost inside all the blocks is

$$\sum_{i=1}^{\lfloor n/(2m-1) \rfloor} E(t_i) = \lfloor n/(2m-1) \rfloor E(t) = \Omega((n/m)E(t)). \qquad \square$$

Up to this point we have established that the average search cost is $\Omega((n/m)E(t))$, where t is the time necessary to search the occurrences of r patterns inside a text block of length $2m-1$. In the following we establish that $E(t) = \Omega(\log_\sigma(rm))$ and this will complete our proof. Note that this formulation of the problem is similar to the original, for the particular case $n = 2m-1$.

Inside a given block $B = b_1 \ldots b_{2m-1}$, each of the r patterns can match at m different positions (starting at block position 1 to m). Each possible match position of each pattern will be called a *candidate* and identified by the pair (s,i), where $s \in 1 \ldots r$ is the pattern number and $i \in 1 \ldots m$ is the starting position inside the block. Hence there are rm candidates.

Definition 1. A *candidate* is a pair (s,i), where $s \in 1 \ldots r$ is a pattern index and $i \in 1 \ldots m$ is the initial block position where an occurrence of pattern s may start. The pattern matching problem is equivalent to that of determining which of the rm candidates occur in B, that is, of computing the set

$$R = \{(s,i), B_{i \ldots i+m-1} = P^s\}.$$

We have to examine enough characters of B to fully determine set R. We will perform a sequence of *accesses* (character reads) inside the block, at different positions $i_1, i_2, \ldots,\ 1 \leqslant i_j \leqslant 2m-1$, until the information we have gathered is enough to determine R.

Given the definition of R, it is clear that the only way to prove that $(s,i) \notin R$, is to perform an access at a position i_j, $i \leqslant i_j \leqslant i+m-1$, such that $B_{i_j} \neq P^s_{i_j - i+1}$. On the other hand, if after accessing all positions $i_j \in i \ldots i+m-1$ (in any order inside the access sequence), it turns out that $B_{i_j} = P^s_{i_j - i+1}$, then we have that $(s,i) \in R$. Observe that it is not possible to determine $(s,i) \in R$ with less than m accesses.

As our proof refers to the number of accesses to the text, the cost model we use is that we pay $O(t)$ to perform t accesses, $i_1, i_2 \ldots i_t$, so that with those t accesses we gather enough information to determine R. That is, for every candidate (s,i), we determine either $(s,i) \in R$ or $(s,i) \notin R$.

At this point we make another optimistic assumption. If, after $m-1$ accesses, we have not determined R, then the rest is for free. Since with $m-1$ access we cannot prove $(s,i) \in R$ for any (s,i), the only way to determine R with $m-1$ access is to find out that $R = \emptyset$. This assumption permits us focusing in how many accesses are necessary to determine $(s,i) \notin R$ for all $1 \leqslant s \leqslant r$ and $1 \leqslant i \leqslant m$.

Lemma 3. *Any lower bound to the problem of determining whether $R = \emptyset$, where only the first $m-1$ accesses are counted, is a lower bound to the problem of determining R.*

Proof. Consider any algorithm that determines R. We modify it so that, at the end, we only answer whether $R = \emptyset$ or not. Furthermore, only the first $m-1$ accesses of the modified algorithm are counted. The modified algorithm necessarily costs the same or less than the original algorithm for every possible block, and therefore any lower bound to the simpler problem is a lower bound to the original problem. □

Therefore, we can focus only on the first $m-1$ accesses to the block. More importantly, we can focus only on discarding elements from R. The question is, therefore, how many accesses we need on average to prove $R = \emptyset$. If we need m or more, we assume we need just $m-1$.

Lemma 4. *Given an accesses i_j to block B, the probability to discard a candidate (s,i) with that access is at most $1 - 1/\sigma$. This bound is independent on what has happened with previous accesses.*

Proof. Since we access different block positions as we progress, it turns out that block character i_j has never been examined before. Hence, the event $B_{i_j} = P^s_{i_j - i+1}$ has probability $1/\sigma$ because characters are drawn independently and with uniform probability from the alphabet. Since with probability $1/\sigma$ it might happen that $B_{i_j} = P^s_{i_j - i+1}$, in which case candidate (s,i) cannot be discarded, then the probability of discarding (s,i) cannot exceed $1 - 1/\sigma$. □

Note that the probability of discarding candidate (s,i) with the current access i_j could be less than $1/\sigma$ (actually, it would be zero) because of two reasons: (1) position i_j could be outside the area $i \ldots i+m-1$ covered by candidate (s,i); (2) candidate (s,i) could have already been discarded by a previous access. In particular, reason (2) shows that the probability of discarding candidates with accesses depend on each other access. However, as stated in Lemma 4, the upper bound holds in any case and we can consider accesses in isolation.

Lemma 4 establishes that the probability that a given access does *not* rule out a given candidate is at least $1/\sigma$. Consequently, since we can consider these upper bounds in isolation, the probability of *not* discarding a given candidate after t accesses is at least $1/\sigma^t$.

Definition 2. Let c_t be the probability that there is at least one candidate left after t accesses. In particular, $c_0 = 1 \geqslant c_1 \geqslant c_2 \ldots$.

Then we will perform the first access with probability $c_0 = 1$. We will perform a second access with probability c_1, a third access with probability c_2, and so on. The following lemma establishes the expected number of accesses to determine $R = \emptyset$ with the limit of $m - 1$ accesses.

Lemma 5. *The average number of accesses until we establish* $R = \emptyset$ *or we perform* $m - 1$ *accesses is* $\sum_{t=0}^{m-1} c_t$.

Proof. Let us consider that we, iteratively, perform one access and determine whether we need more accesses to establish $R = \emptyset$. Then the expected number of accesses can be written as

$$\sum_{t=0}^{m-1} 1 \times Pr \text{ (we need to perform more than } t \text{ accesses)}$$

$$= \sum_{t=0}^{m-1} 1 \times Pr \text{ (} t \text{ accesses are not enough to establish } R = \emptyset) = \sum_{t=0}^{m-1} c_t. \qquad \square$$

The next lemma establishes the final result we need.

Lemma 6. *If* $r < \sigma^m/m$, *it holds* $\sum_{t=0}^{m-1} c_t = \Omega(\log_\sigma(rm))$.

Proof. As we have seen, the probability that a particular candidate is not discarded after t accesses is lower bounded by $1/\sigma^t$. The probability that at least one candidate is left after t accesses is thus

$$c_t \geqslant 1 - \left(1 - \frac{1}{\sigma^t}\right)^{rm} \geqslant 1 - \mathrm{e}^{-rm/\sigma^t}.$$

Let us now lower bound c_t as follows. If $t \leqslant t^* = \log_\sigma(rm)$, then $c_t \geqslant c_{t^*} = 1 - \mathrm{e}^{-1}$. For $t > t^*$, we will simply use $c_t \geqslant 0$. (This argument is valid for $t^* < m$, that is, $r < \sigma^m/m$, which is the precondition of the Lemma). Therefore,

$$\sum_{t=0}^{m-1} c_t \geqslant \sum_{t=0}^{t^*} (1 - \mathrm{e}^{-1}) + \sum_{t=t^*+1}^{m-1} 0 = (1 - \mathrm{e}^{-1})(1 + \log_\sigma(rm))$$

$$= \Omega(\log_\sigma(rm)). \qquad \square$$

So we have shown that $\Omega(\log_\sigma(rm))$ accesses per block are necessary on average to determine $R=\emptyset$, even with the benefit of stopping after $m-1$ accesses. From the previous Lemmas, it follows that the average lower bound of multipattern string matching is $\Omega(n\log_\sigma(rm)/m)$. Note that our argument is valid for $r<\sigma^m/m$. At that point, however, the average case bound meets the worst case bound $\Omega(n)$. On the other hand, the bound is tight because, as explained in the Introduction, there exist algorithms that obtain average search time $\mathrm{O}(n\log_\sigma(rm)/m)$.

3. A lower bound for approximate multipattern matching

We generalize the proof of Chang and Marr [2], which is rather simple. Let us divide the text into consecutive blocks of length m and assume, again, that we are only interested in the occurrences that are totally inside some block. Now, under Hamming and Levenshtein distances, if we examine characters $i_1\dots i_k$ of the block $B=b_1\dots b_m$, even if none of them match a given pattern P, it could be that all the others match, $p_j=b_j$ for $j\in\{1\dots m\}-\{i_1\dots i_k\}$. In this case, P would appear in B with threshold k. Hence, we need to examine at least $k+1$ characters in order to discard block B. In the case of indel distance, since deleting the k characters that do not match costs $2k$, we need to examine at least $\lfloor k/2\rfloor+1$ characters of B before possibly discarding it.

This argument is valid for one or for r patterns, and shows that there is a lower bound of $\Omega(kn/m)$ in approximate multipattern matching. On the other hand, in any approximate search we have to report in particular all the exact occurrences of the patterns, and therefore the lower bound $\Omega(n\log_\sigma(rm)/m)$ applies here too. So we can take the maximum (or equivalently the sum) of both lower bounds to obtain $\Omega(n(k+\log_\sigma(rm))/m)$. Again, this bound is tight because there have appeared algorithms that are $\mathrm{O}(n(k+\log_\sigma(rm))/m)$ on average [6].

References

[1] A.V. Aho, M.J. Corasick, Efficient string matching: an aid to bibliographic search, Commun. ACM 18 (6) (1975) 333–340.

[2] W. Chang, T. Marr, Approximate string matching and local similarity, in: Proceedings of the Fifth Combinatorial Pattern Matching (CPM'94), Lecture Notes in Computer Science, vol. 807, 1994, pp. 259–273.

[3] M. Crochemore, A. Czumaj, L. Gąsieniec, S. Jarominek, T. Lecroq, W. Plandowski, W. Rytter, Speeding up two string matching algorithms, Algorithmica 12 (4/5) (1994) 247–267.

[4] M. Crochemore, A. Czumaj, L. Gąsieniec, T. Lecroq, W. Plandowski, W. Rytter, Fast practical multi-pattern matching, Inform. Process. Lett. 71 (3–4) (1999) 107–113.

[5] M. Crochemore, W. Rytter, Text Algorithms, Oxford University Press, Oxford, 1994.

[6] K. Fredriksson, G. Navarro, Average-optimal multiple approximate string matching, in: Proceedings of the 14th Combinatorial Pattern Matching (CPM'03), Lecture Notes in Computer Science, vol. 2676, 2003, pp. 109–128.

[7] Z. Galil, K. Park, An improved algorithm for approximate string matching, SIAM J. Comput. 19 (6) (1990) 989–999.

[8] D.E. Knuth, J.H. Morris Jr., V.R. Pratt, Fast pattern matching in strings, SIAM J. Comput. 6 (1) (1977) 323–350.

[9] A.C. Yao, The complexity of pattern matching for a random string, SIAM J. Comput. 8 (3) (1979) 368–387.

[10] K. Fredriksson, G. Navarro, Improved Single and Multiple Approximate String Matching, in: Proceedings of the 15th Annual Symposium on Combinatorial Pattern Matching (CPM 2004), LNCS, 2004, to appear.