Sublinear Algorithms for Parameterized Matching

Leena Salmela and Jorma Tarhio[⋆]

Helsinki University of Technology
{lsalmela, tarhio}@cs.hut.fi

Abstract. Two strings parameterize match if there is a bijection that transforms the first string character by character into the second string. This problem has been studied in both one and two dimensions but the research has been centered on developing algorithms with good worst-case performance. We present algorithms that solve this problem in sublinear time on average for moderately repetitive patterns.

1 Introduction

In the parameterized matching problem a text and a pattern is given and the task is to find all substrings of the text that can be transformed into the pattern by using a bijection on the alphabet. This problem was first considered by Baker [5] with an application to software maintenance. Another application of parameterized matching is plagiarism detection [11].

Later the parameterized matching problem has been investigated in two dimensions by Amir et al. [1] and Hazay et al. [14]. This two-dimensional problem has a fairly obvious application in image searching. Parameterized matching can find an image even if its color map has been changed. Other related work includes parameterized matching of multiple patterns [16], parameterized matching with mismatches [13] and approximate parameterized search [7].

Previous research of parameterized matching has been centered in developing algorithms with good worst-case performance. Some effort to develop an algorithm fast on average was made by Baker [6] who developed an algorithm based on the famous Boyer-Moore algorithm [8] but the average case complexity was not analyzed. In fact the algorithm uses a linear preprocessing with respect to the length of the text and thus loses the good average case complexity of the Boyer-Moore algorithm.

In this paper we introduce new algorithms that are sublinear on average. We present practical solutions for both the one-dimensional and two-dimensional parameterized matching problems. We analyze the time complexities of the algorithms for random texts and moderately repetitive patterns, and present experimental results for certain interesting classes of patterns.

2 Definitions

Let S and S' be equal size strings of characters drawn from the alphabet Σ. S and S' parameterize match (or p-match for short) if there exists a bijection π such

[⋆] Work by Jorma Tarhio was supported by Academy of Finland.

that for each i $S[i] = \pi(S'[i])$. So strings 'abac' and 'bcba' p-match because the bijection $\pi(a) = c, \pi(b) = a, \pi(c) = b$ transforms 'bcba' into 'abac'. On the other hand strings 'aabb' and 'acbb' do not p-match because a bijection cannot map both 'a' and 'c' to 'a' and thus there is no bijection that can transform 'aabb' to 'acbb'.

Let us now define the parameterized matching problem. In the one dimensional case, we are given a text $T[1..n]$ and a pattern $P[1..m]$ in alphabet Σ and the task is to find all substrings of the text that p-match with the pattern. In the two-dimensional case the input is a text T of size $n \times n$ and a pattern P of size $m \times m$. The task then is to find all those $m \times m$ substrings of the text that p-match to P.

Two disjoint alphabets were used in the original definition of the parameterized matching problem by Baker. One of the alphabets was a fixed alphabet like in the standard string matching problem and the other one was a parameterized alphabet like our Σ. Both the pattern and the text could contain characters from both alphabets but characters from the fixed alphabet were required to match exactly. We decided to use only the parameterized alphabet because that is natural for the two dimensional problem of image search and we wished to give a unified treatment to both the one dimensional and two dimensional cases.

Many of the algorithms make use of so called predecessor strings. A string S is transformed into a predecessor string as follows. If a character in position i has occurred earlier in the string in position j the position i in the predecessor string contains $i - j$. Otherwise the predecessor string contains 0. For example the string 'aabac' is transformed into 0-1-0-2-0. Now it can be fairly easily seen that if two strings p-match, their predecessor strings match exactly [5].

Another way to transform the two strings so that the transformed strings will match exactly if the original strings p-matched, is to transform them into restricted growth functions (RGF) [19]. A string is transformed into a RGF by replacing all occurrences of the first occurring character with 1, the second one with 2 and so on. We call the resulting string the RGF string. For example the string 'aabac' is transformed into 1-1-2-1-3. The properties of restricted growth functions have been studied previously. In [19] it is shown that there are b_n different RGFs of length n where b_n is the n:th Bell number. Kreher and Stinson [19] also give an algorithm for ranking RGFs.

To classify the repetitiveness of a pattern we introduce the concept of q-repetitiveness. A pattern is q-repetitive if for all substrings of length q there is a character that occurs at least twice in the substring. Thus the pattern "aaaa" is 2-repetitive while the pattern "aabb" is 3-repetitive but not 2-repetitive because the substring "ab" contains no repetition. Similarly a two-dimensional pattern is q-repetitive if for all substrings of size $q \times q$ there is a character that occurs at least twice in the substring.

3 Earlier Solutions

3.1 One-Dimensional Algorithms

In her original paper Baker [5] gave a suffix tree based algorithm for finding parameterized matches. The algorithm first preprocesses both the text and the pattern by transforming them into predecessor strings. After this preprocessing the problem can almost be solved by conventional exact string matching algorithms. The only remaining problem is that if we are considering a window on the text, the predecessor pointers might point to positions outside the window. Baker proposed modifications to the suffix tree construction algorithm that take care of this problem. The construction of the suffix tree was further improved by Kosaraju [18] and Cole and Hariharan [9].

In addition Baker [6] has proposed a Boyer-Moore based algorithm. Also this algorithm preprocesses both the text and the pattern into predecessor strings. Baker then uses a modification of the TurboBM algorithm [10] for finding the p-matches. The algorithm has a good worst case performance but because of the preprocessing the sublinearity of the Boyer-Moore type algorithms is lost unless several searches are made with the same text.

Amir et al. [2] have proposed an algorithm for the p-matching problem based on the Knuth-Morris-Pratt algorithm [17] for standard string matching. They also prove that their algorithm is optimal if the alphabet is unbounded.

3.2 Two-Dimensional Algorithms

The two-dimensional parameterized matching problem was first considered by Amir et al. [1] in the context of function matching. They give an algorithm that preprocesses the text into a predecessor representation suitable for two-dimensional strings and then apply a conventional two-dimensional algorithm. Hazay et al. [14] give another algorithm for two-dimensional parameterized matching that is based on the "duel-and-sweep" paradigm. Both of these algorithms are quite complicated and neither one of them has been implemented as far as we know.

4 Our Algorithms

In this section we develop Boyer-Moore type algorithms that do not preprocess the text and thus the preprocessing does not prevent average case sublinearity. Our algorithms use q-grams to achieve longer shifts. The use of q-grams is a well known technique to improve the efficiency of the exact Boyer-Moore-Horspool (BMH) algorithm [15] in case of small alphabets, see e.g. [3].

In this section we first describe the one-dimensional algorithm with several variations and then we discuss the two-dimensional algorithm.

4.1 Three One-Dimensional Algorithms

Our one-dimensional algorithms are derived from the Horspool variant of the Boyer-Moore algorithm. In the BMH algorithm the text is processed in windows of size m. The last character of the window is read first. If it does not match the last character of the pattern the window is shifted based on it. Otherwise the window is checked for a match after which a shift is made. In the parameterized matching problem the last character alone never tells that there cannot be a match and even the last two characters usually do not indicate that the window cannot match the pattern. Therefore we form a q-gram of the last q characters of the window and make the shift based on it.

The preprocessing phase of the BMH algorithm constructs the shift table which is consulted in the matching phase to find out the length of the shift based on the last character of the text window. The shift is calculated so that after the shift the last character of the previous window will be aligned with the last occurrence of that character in the pattern.

In the parameterized matching problem the shifts are made based on the last q-gram of the window and we wish to make a shift that aligns it with the last q-gram of the pattern that p-matches it. As described in Section 2 two strings p-match if their predecessor strings match or equally if their RGF strings match. Thus we wish to index the table with the predecessor or RGF strings. An obvious solution is using the rank of the RGF strings as indexes. We call this algorithm Parameterized Boyer-Moore-Horspool with RGF or PBMH-RGF for short.

The problem with this approach is that calculating the rank of an RGF takes quite a lot of time and this needs to be done for each inspected window. Another alternative for calculating the indexes is to transform the q-grams into predecessor strings and then reserving enough bits for each character of the predecessor strings in the index. The first character of the predecessor string is always 0 so we need not reserve any space for it. The second character is either 0 or 1 because the character in the original string is either the same as the first or not. The third character is 0, 1 or 2 with similar reasoning. This means that the last bit of the index is reserved for the second character of the predecessor strings, the next two bits are reserved for the third character, and so on. We call this algorithm Fast Parameterized Boyer-Moore-Horspool or FPBMH for short. This approach wastes some space but the indexes are much faster to calculate. The RGF approach needs a table of size b_q where b_q is the q:th Bell number while the FPBMH algorithms needs a table of size 2^s where $s = \sum_{i=2}^{q} \lceil \log_2 i \rceil$. Table 1 shows the number of entries in the shift table for both approaches for different values of q.

In a random text the distribution of the predecessor strings is very steep. The most common predecessor string of length q, 0^q, has a high probability if the alphabet is reasonably large while the least common predecessor string, 01^{q-1}, has a probability close to 0. This means that we might need to use quite large q-grams which is a problem for FPBMH. On the other hand hashing the q-grams cleverly might let us use even larger q-grams than the PBMH-RGF algorithm can

Table 1. The number of entries in the shift table for PBMH-RGF, FPBMH and PBMH-Hash for various values of q

Algorithm	$q=2$	$q=3$	$q=4$	$q=5$	$q=6$	$q=7$	$q=8$	$q=9$	$q=10$
PBMH-RGF	2	5	15	52	203	877	4,140	21,147	115,975
FPBMH	2	8	32	256	2,048	16,384	131,072	2,097,152	33,554,432
PBMH-Hash	2	4	7	11	16	22	29	37	46

handle. We tried hashing the q-grams by transforming them first to predecessor strings and then adding up all the positions of the predecessor string. With this hashing scheme the most common q-gram is the only one hashed to 0 and thus the hashing might even out the distribution of the q-grams. This modification of the algorithm called PBMH-Hash needs a table of size $q(q-1)/2+1$. Table 1 includes the space requirement for this approach also.

4.2 A Two-Dimensional Algorithm

The two-dimensional algorithm is based on the two-dimensional string matching algorithm by Tarhio [20] which is an extension of the BMH algorithm. In the algorithm by Tarhio the text is divided into $\lceil (n-m)/m \rceil + 1$ strips each of which has m columns. Each strip is then searched for an occurrence with a BMH like algorithm and each potential match is verified with the trivial algorithm.

In each position the character at the lower righthand corner is investigated. If this character occurs in the lowest row of the pattern, there is a potential match which has to be verified. These are found with the help of two tables, M and N. $M[x]$ is the column where the character x occurs first in the lowest row of the pattern and N links the occurrences of x in the lowest row of the pattern. The pattern is shifted down the strip with another table D. $D[x]$ is the occurrence of x that is closest to the lowest row of the pattern but not in the last row. If x does not appear in the pattern, $D[x]$ is m.

The algorithm can be modified to read several characters and calculate the shifts based on all these characters. If we read $q \times q$ characters (a two-dimensional q-gram), the text will then be divided into $\lceil (n-m)/(m-q+1) \rceil + 1$ strips each containing $m-q+1$ columns.

This algorithm which uses q-grams can fairly easily be extended to parameterized matching in a similar fashion as the BMH algorithm was extended for one-dimensional parameterized matching. The resulting algorithm proceeds exactly like the algorithm by Tarhio but the read q-grams are transformed into predecessor strings and these are then used to index the tables. As with the one-dimensional case, there are several ways to transform the predecessor strings into indexes. We implemented the transformation the same way as in the FPBMH algorithm.

5 Analysis

We first analyze the worst and average case complexity of the one-dimensional algorithms and then turn to the two-dimensional case. When analyzing the average case complexity we assume the standard random string model where each character of the text is chosen independently and uniformly.

5.1 The One-Dimensional Algorithms

The preprocessing phase of the algorithms consists of initializing the shift table which takes time proportional to the number of entries in the table. In addition to preprocess the pattern we need to keep track of where the different symbols of the alphabet occurred previously and thus the preprocessing of the q-grams of the pattern takes $O(\sigma + mq)$ time where σ is the size of the alphabet. As stated earlier the number of entries in the shift table is b_q for PBMH-RGF, 2^s for FPBMH and $q(q-1)/2+1$ for PBMH-Hash where b_q is the q:th Bell number and $s = \sum_{i=2}^{q} \lceil \log_2 i \rceil$. Therefore the preprocessing phases of PBMH-RGF, FPBMH and PBMH-Hash have time complexities $O(b_q + \sigma + mq)$, $O(q^{q-1} + \sigma + mq)$, and $O(q^2 + \sigma + mq)$ respectively.

The matching phases of PBMH-RGF and FPBMH algorithms have the same time complexities. The only difference in the algorithms is in handling of the q-grams but both algorithms do this in $O(q)$ time and thus the resulting complexities will be the same. The hashing in the PBMH-Hash algorithm slightly changes the time complexity of the algorithm but the difference is negligible.

In the worst case the one-dimensional algorithms find a match in each position. This means that for each window the whole window is read and compared to the pattern so the worst case complexity of the algorithms is $O(nm)$.

Let us then analyze the average case complexity. In order to do that we need to consider the probability distribution of the different predecessor strings corresponding to random q-grams. Let σ denote the size of the alphabet and let z be the number of zeros in the given predecessor string. Each of the zeros presents a different character in the original string and each non-zero element of the predecessor string is defined by the zeros. Then the probability that the given predecessor string matches a random string is:

$$P(z, q) = \frac{\sigma!}{\sigma^q \cdot (\sigma - z)!}$$

The probability of a window to be checked is the probability that the last q-gram of the window p-matches the last (or $m - q$:th) q-gram of the pattern. Thus the expected number of checked windows is

$$C = (n - m + 1) \cdot P(z_{m-q}, q)$$

where z_{m-q} is the number of zeros in the last q-gram of the pattern. Now if we can choose q so that $z_{m-q} < q$ the probability $P(z_{m-q}, q)$ is low enough and there are only a few checked windows so the scanning time will dominate.

Let us now turn to analyzing the scanning time. We estimate the expected length of shift in the algorithm with

$$S \geq (m - q + 1) \cdot (1 - P(z_{\max}, q))^{m-q} + \sum_{i=1}^{m-q} i \cdot P(z_{m-q-i}, q) \cdot (1 - P(z_{\max}, q))^{i-1}$$

where $z_{\max}$ is the maximum number of zeros in the q-grams of the pattern. Note that this estimate for S is not quite accurate because the consecutive overlapping q-grams of the pattern are not independent. However the difference from the accurate value is insignificant.

If we now choose q to be the smallest q such that the pattern is q-repetitive, the probability $P(z_{\max}, q)$ will be low enough. Then the expected length of shift approaches the value $m - q + 1$ so on average $O((qn)/(m - q + 1))$ characters are read. Furthermore if $q < (m + 1)/2$ the algorithm is sublinear on average.

Note that all patterns are not q-repetitive for any $q < (m+1)/2$ and in these cases we cannot guarantee the sublinearity of the algorithm. However parameterized matching is most often applied to searching for repetitive patterns so in most practical cases the sublinearity can be guaranteed.

The above analysis holds also if we have both a fixed and a parameterized alphabet. In fact the fixed alphabet makes the problem easier. In this case a sufficient condition for sublinearity is that each q-gram of the pattern contains repetition or at least one character from the fixed alphabet.

5.2 The Two-Dimensional Algorithm

Let us first consider the complexity of the preprocessing phase. The two-dimensional algorithm uses the strategy of the FPBMH algorithm when calculating the indexes of the shift table. Thus the number of entries in the shift table is 2^s where $s = \sum_{i=2}^{q^2} \lceil \log_2 i \rceil$. As with the one-dimensional algorithms we also need to keep track of the previous occurrences of the alphabet symbols and thus a table of size σ is needed for that. The time complexity of the preprocessing phase of the two-dimensional algorithms is thus $O((q^2)^{q^2-1} + \sigma + m^2q^2)$.

The worst case for the two-dimensional algorithm occurs when all positions of the text match. The worst case time complexity is then clearly $O(n^2m^2)$.

For the average case complexity we will need to estimate the number of checked windows. There are a total of $(n - m + 1)^2$ windows so on average

$$C = (n - m + 1)^2 \cdot P(z_{m-q,m-q}, q^2)$$

of them are checked where $z_{m-q,m-q}$ is the number of zeros in the q-gram of the pattern that starts at position $(m-q, m-q)$. If $z_{m-q,m-q} < q^2$, $P(z_{m-q,m-q}, q^2)$ is low enough and there will only be a few checked windows. Therefore the scanning time will dominate.

Let us next consider the expected length of shift, S. The estimate is very similar to the one-dimensional case:

$$\begin{aligned} S \geq\ & (m - q + 1) \cdot (1 - P(z_{\max}, q^2))^{(m-q)\cdot(m-q+1)} \\ & + \sum_{i=1}^{m-q} i \cdot P(\min_{1\leq x\leq m-q+1} z_{m-q-i,x}, q^2) \cdot (1 - P(z_{\max}, q^2))^{(i-1)\cdot(m-q+1)} \end{aligned}$$

where $z_{\max}$ is the maximum number of zeroes in the predecessor strings corresponding to any of the q-grams of the pattern. As with the one-dimensional case, if we now choose q to be the smallest value such that the pattern is q-repetitive, $z_{\max} < q^2$, $P(z_{\max}, q^2)$ is low enough and S approaches $m - q + 1$. So on average $O((n-m)/(m-q+1)\cdot q^2n/(m-q+1)) = O(q^2n^2/(m-q)^2)$ characters are read. Therefore if the pattern is q-repetitive for a suitable q then the algorithm will be sublinear on average. Again some patterns are not q-repetitive for a suitable q and in these cases the sublinearity of the algorithm cannot be guaranteed.

6 Experimental Results

The analysis predicts that the value of q should be chosen to be the smallest q such that the pattern is q-repetitive. To validate this we ran our algorithms with several patterns and a randomly generated text with alphabet size 256. Figures 1, 2 and 3 show the proportion of read characters and the runtime for some patterns. The proportion of read characters is calculated as lookups divided by the length of the text so for a sublinear algorithm this value is less than one. All these tests were run on a computer with a 1.0 GHz AMD Athlon processor, 512 MB of memory and 256 kB on-chip cache. The computer was running Linux 2.4.22. The algorithms were written in C and compiled with gcc 3.2.2.

Figure 1 shows that choosing a larger q with a highly repetitive pattern does not make the algorithms perform faster. Using 2-grams already guarantees long enough shifts and thus assembling larger q-grams just wastes time. Figure 2 presents a completely different scenario. Here the pattern is not q-repetitive for any q and as can be seen we cannot choose large enough q to guarantee the sublinearity of the algorithms. In Figure 3 the situation is something in between. The pattern is 3-repetitive but not 2-repetitive. As can be seen the value $q = 3$ is optimal in this situation and using larger q-grams only makes the algorithms do more work.

Table 2 shows a runtime comparison of our one-dimensional algorithms and a Boyer-Moore-Horspool algorithm (PBMH) which we use as a reference method. The PBMH algorithm preprocesses the text into a predecessor string and then matches the pattern against the text. The preprocessing of the text is included in the figures but the preprocessing of the pattern is not. As can be seen our algorithms are faster when the pattern contains a substantial amount of repetition. However when the pattern contains no repetition the algorithm that preprocesses the text is faster.

To demonstrate the performance of our algorithms in a more realistic scenario we ran some tests with DNA data. The text was a chromosome from the fruit fly genome (20 MB) and the patterns were chosen randomly from the text. Our algorithms were fastest when using 6-grams. Figure 4 shows the averages over 50 runs. As can be seen our algorithms have characteristics typical to Boyer-Moore based algorithms. With longer patterns the shifts get longer and thus the algorithms are faster.

We ran also some tests with the two-dimensional algorithm. We used two different texts. One was a randomly generated text where the characters were drawn from an alphabet of 256 characters. The other one was a picture of a map from the photo archive of Gimp-Savvy.com [12]. We examined the proportion of read characters for three different patterns of size 8×8. The first one contained repetitions of one character. The second pattern contained no repetitions and the third contained a map symbol which contains some repetition. Table 3 shows the results of the tests run with the two-dimensional algorithm using 3-grams. As can be seen the algorithm performs well when the text or the pattern contains repetitions.

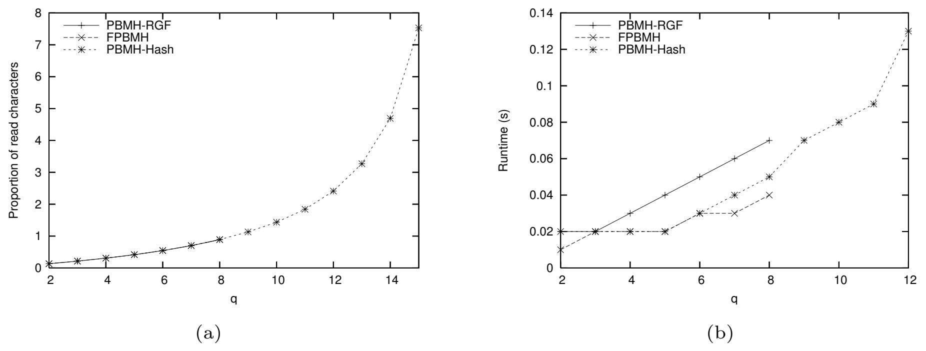


Fig. 1. Proportion of read characters (a) and runtime (b) for the pattern "aaaaaaaaaaaaaaaa" in a random text

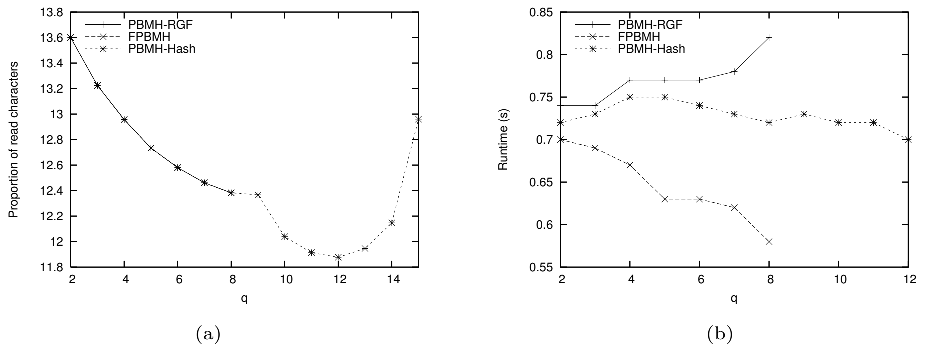


Fig. 2. Proportion of read characters (a) and runtime (b) for the pattern "qwertyuiopsadfgh" in a random text

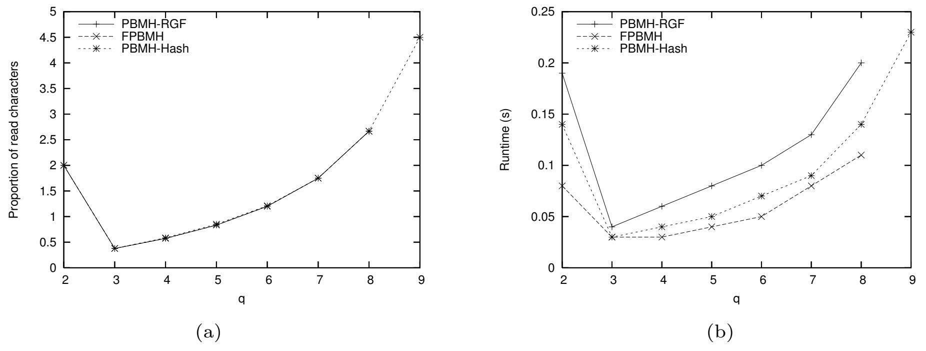
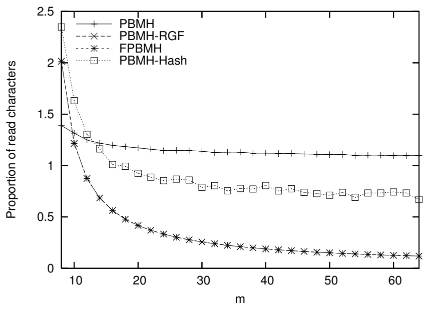


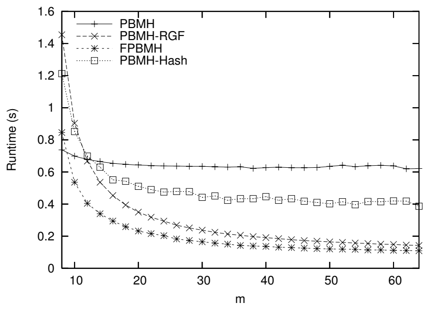
Fig. 3. Proportion of read characters (a) and runtime (b) for the pattern "aassddssaa" in a random text

Table 2. Runtime comparison of the one-dimensional algorithms in a random text

Algorithm	P=aaaaaaaaaaaaaaaaaa	P=qwertyuiopasdfgh	P=aassddssaa
PBMH	0.08 s	0.29 s	0.08 s
PBMH-RGF	0.02 s	0.74 s	0.04 s
FPBMH	0.01 s	0.58 s	0.03 s
PBMH-Hash	0.02 s	0.70 s	0.03 s



(a)



(b)

Fig. 4. Proportion of read characters (a) and runtime (b) for a text of DNA data and patterns of varying length. Our algorithms used 6-grams in these tests.

Table 3. Proportion of read characters for two different texts and several different patterns. All the patterns are of size 8×8.

Text	Single-character pattern	Pattern with no repetitions	Pattern with repetitions
Random	0.25	7.90	0.25
Map	1.14	0.25	0.33

7 Conclusions and Further Work

We have presented practical Boyer-Moore type algorithms for one and two-dimensional parameterized matching. We have showed that these algorithms are sublinear on average for q-repetitive patterns and confirmed this analysis with experiments.

Parallel to our work and independently of us Fredriksson and Mozgovoy [11] have also developed sublinear algorithms for one-dimensional parameterized matching. Their algorithms are based on the shift-or [4] and backward DAWG matching (BDM) [10] algorithms. As further work we need to compare our algorithms also with these algorithms.

The analysis assumes the random string model which might not be applicable especially with two-dimensional texts which are typically images. It is very characteristic of such data that the probability of two nearby characters being the same is very high. We need to further investigate these typical characteristics of texts and analyze our algorithms in this context. We also need to make further tests on real data to confirm the usefulness of our algorithms.

References

1. Amir, A., Aumann, Y., Cole, R., Lewenstein, M., Porat, E.: Function matching: algorithms, applications and a lower bound. In: Proceedings of ICALP. (2003) 929–942
2. Amir, A., Farach, M., Muthukrishnan, S.: Alphabet dependence in parameterized matching. Information Processing Letters **49**(3) (1994) 111–115
3. Baeza-Yates, R.: Improved string searching. Software – Practice and Experience **19**(3) (1989) 257–271
4. Baeza-Yates, R., Gonnet, G.: A new approach to text searching. Communications of ACM **35**(10) (1992) 74–82
5. Baker, B.S.: A theory of parameterized pattern matching: algorithms and applications. In: Proceedings of the 25th ACM Symposium on the Theory of Computation. (1993) 71–80
6. Baker, B.S.: Parameterized pattern matching by Boyer-Moore-type algorithms. In: Proceedings of the 6th Annual ACM Symposium on Theory of Computing. (1995) 541–550
7. Baker, B.S.: Parameterized diff. In: Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms. (1999) 854–855
8. Boyer, R., Moore, J.: A fast string searching algorithm. Communications of the ACM **20**(10) (1977) 762–772
9. Cole, R., Hariharan, R.: Faster suffix tree construction with missing suffix links. In: Proceedings of the 32nd ACM Symposium on the Theory of Computation (STOC). (2000) 407–415
10. Crochemore, M., Lecroq, T., Czumaj, A., Gąsieniec, L., Jarominek, S., Plandowski, W.: Speeding up two string-matching algorithms. In: 9th Annual Symposium on Theoretical Aspects of Computer Science (STACS'92). Volume 577 of LNCS. (1992) 589–600
11. Fredriksson, K., Mozgovoy, M.: Sublinear parameterized single and multiple string matching. Technical Report A-2006-2, Department of Computer Science, University of Joensuu (2006)
12. Gimp-Savvy.com: Copyright-free photo archive: Public domain photos and images. http://gimp-savvy.com/PHOTO-ARCHIVE/ (2000)
13. Hazay, C., Lewenstein, M., Sokol, D.: Approximate parameterized matching. In: Proceedings of the 12th European Symposium on Algorithms (ESA). (2004) 414–425
14. Hazay, C., Lewenstein, M., Tsur, D.: Two dimensional parameterized matching. In: Proceedings of the 16th Annual Symposium on Combinatorial Pattern Matching (CPM'05). Volume 3537 of LNCS. (2005) 266–279
15. Horspool, N.: Practical fast searching in strings. Software – Practise and Experience **10** (1980) 501–506
16. Idury, R.M., Schäffer, A.A.: Multiple matching of parameterized patterns. Theorethical Computer Science **154**(2) (1996) 203–224
17. Knuth, D.E., Morris, J.H., Pratt, V.R.: Fast pattern matching in strings. SIAM Journal of Computing **6** (1977) 323–350
18. Kosaraju, S.R.: Faster algorithms for the construction of parameterized suffix trees. In: Proceedings of the 36th Symposium on Foundation of Computer Science (FOCS). (1995) 631–637
19. Kreher, D.L., Stinson, D.R.: Combinatorial Algorithms: Generation, Enumeration and Search. CRC Press (1999)
20. Tarhio, J.: A sublinear algorithm for two dimensional string matching. Pattern Recognition Letters **17** (1996) 833–838