An Index for Two Dimensional String Matching Allowing Rotations

Kimmo Fredriksson * Gonzalo Navarro † Esko Ukkonen *

November 17, 1999

Abstract

We present an index to search a two-dimensional pattern of size $m \times m$ in a two-dimensional text of size $n \times n$, even when the pattern appears rotated in the text. The index is based on suffix trees. By using $O(n^2)$ space the index can search the pattern in $O(m^5)$ worst case time and in $O(\log_\sigma(n)^{5/2})$ on average, where σ is the alphabet size. A larger index of size $O(n^2 \log_\sigma(n)^{3/2})$ yields an average search time of $O(\log_\sigma n)$. We also discuss alternative matching models.

1 Introduction

The problem of searching a two-dimensional pattern of size $m \times m$ in a two-dimensional text of size $n \times n$ when the pattern can appear in the text in rotated form was firstly addressed from a combinatorial point of view in [1]. They present an online algorithm for searching a pattern allowing rotations.

In this work we are interested in offline searching, that is, in building an index over the text that allows fast querying. The data structure we use is based on suffix trees. Suffix trees for two-dimensional texts has already been considered, e.g. in [6, 4, 5]. The idea of searching a rotated pattern using a suffix array is mentioned in [6], but they allow only rotations of multiples of 90 degrees. The problem is much more complex if we want to allow any rotation.

The matching model we use in this paper is that the pattern center must match a text center [1]. Moreover, for the pattern to be considered to appear centered at some text position, each text center involved must match the value of the pattern cell where the text center lies. We also consider another model (called "grays" here) where the value of the text center must be between the minimum and maximum among the 9 neighbors surrounding the corresponding pattern cell [2].

Our results can be summarized in Table 1. We call σ the alphabet size and our average case results assume uniform distribution over those values. The complexities assume the use of suffix trees, while for suffix arrays the time has to be multiplied by $O(\log n)$.

*Dept. of Computer Science, University of Helsinki.

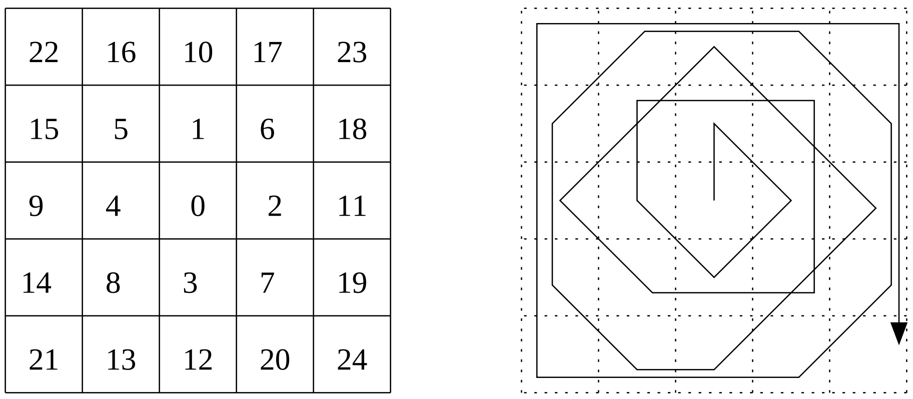
†Dept. of Computer Science, University of Chile. Work developed while the author was in a postdoctoral stay at the Dept. of Computer Science, Univ. of Helsinki. Partially supported by Fundación Andes.

Model	Time	Space
Exact	$\log_\sigma(n)^{5/2}$	n^2
Exact	$\log_\sigma n$	$n^2 \log_\sigma(n)^{3/2}$
Grays	$n^{2-\frac{2}{\log_{1.25}\sigma}} \log_\sigma(n)^{3/2}$	n^2

Table 1: Our results.

2 The Data Structure

We propose to use a suffix tree or array of the text, defined as follows. Each cell of the text defines a string which is obtained by reading text positions at increasing distances from the center of the cell. The first character is that of the cell, then come the 4 closest centers (from the cells above, below, left and right of the central cell), then the other 4 neighbors, and so on. The cells at the same distance are read at some predefined order. If such a string hits the border of the text it is considered finished there. We will call "sistrings" those strings obtained. Figure 1 shows a possible reading order.



22	16	10	17	23
15	5	1	6	18
9	4	0	2	11
14	8	3	7	19
21	13	12	20	24

Figure 1: A possible reading order for the sistring that starts in the middle of a text of 5×5.

Therefore each text cell defines a string of length $O(n^2)$. A suffix trie on those cells can be built, which has average size $O(n^2)$ and average depth $O(\log_\sigma(n^2))$. Alternatively, the unary paths of such trie can be compressed in order to obtain a suffix tree of $O(n^2)$ worst case size. Finally, a suffix array can be obtained by collecting the leaves of the suffix tree. The suffix array is also $O(n^2)$ space but much smaller in practice. The suffix trie can be constructed in $O(n^2 \log_\sigma n)$ average time. The suffix array can be built in $O(n^2 \log n)$ string comparisons, which has to be multiplied by $O(\log_\sigma n^2)$ to obtain average character comparisons.

We describe the algorithms in the suffix trie for simplicity. For suffix arrays one needs to multiply the results by $O(\log n)$ as well.

3 The Search Algorithm

A brute force search approach is to check the pattern in its $O(m^3)$ orientations and search each one in the suffix trie. To check the pattern in a given orientation we have to see in which order have the pattern cells to be read so that they match the same reading order of the suffix trie construction. This gives immediately an algorithm which is $O(m^5)$ time.

Figure 2 shows the reading order induced in the pattern by a rotated occurrence. For each possible rotation we compute the induced reading order, build the string obtained by reading the pattern in that order from its center, and search that string in the suffix trie. Note in particular that some pattern cells may be read more than once and others may not be considered at all.

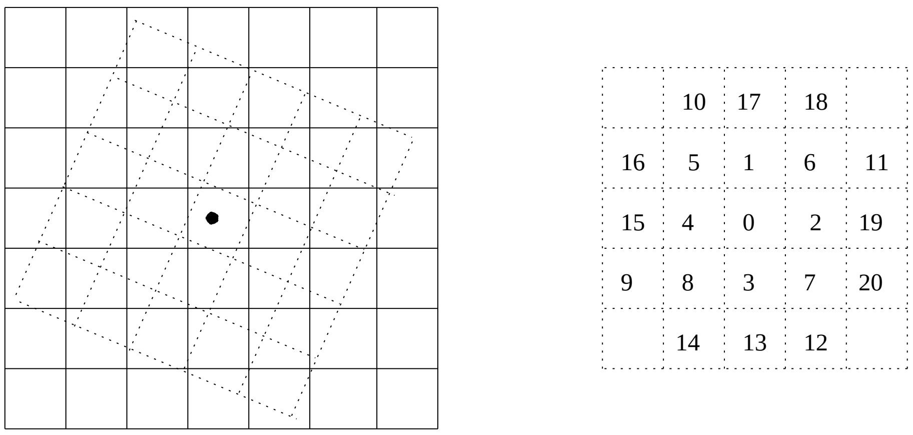


Figure 2: Reading order induced in the pattern by a rotated occurrence.

Note that the number of rotations to try is incremental: until we do not consider the 6th text cell, there are only 4 relevant orientations. The number of rotations grows as we get farther from the center and they are tried only on the surviving branches of the suffix trie. In the worst case, however, we still check $O(m^5)$ cells (because the number grows polynomially instead of exponentially).

However, if n is not too large, not all these paths exist on average. We use the simple model where it is assumed that all the different strings of length up to $\log_\sigma n^2 = 2\log_\sigma n$ exist, and after that there are n^2 different strings. This model is pessimistic but has the same order of the average case if the text is random.

This means that we enter always until depth $h = 2\log_\sigma n$ in the suffix trie. The radius of the pattern considered at that depth is $O(\sqrt{h})$, and therefore the number of different rotations is $O(h^{3/2})$. Since each rotation involved getting into the trie up to depth h, the total amount of work is $O(h^{5/2})$. After that point we have $O(h^{3/2})$ candidates that are searched deeper in the suffix tree. However, there are on average n^2/σ^ℓ sistrings of length ℓ matching a given string, so the total work done when traversing the deeper levels of the suffix tree until they get eliminated is

$$\sum_{\ell \geq 1+2\log_\sigma n} \frac{n^2}{\sigma^\ell}\ell^{3/2} \quad = \quad \sum_{\ell\geq 1} \frac{(\ell + 2\log_\sigma n)^{3/2}}{\sigma^\ell} \quad = \quad O(\log_\sigma n)^{3/2}$$

Therefore we have a total search cost of $O(\log_\sigma n)^{5/2}$.

4 A Larger but Faster Index

Since the main cost of the search lies in the first part, we consider now indexing also the rotated versions of the text sistrings, so that the rotations of the pattern need not be considered. Imagine that we index all the rotations of the text up to depth H. This means that there will be $O(n^2H^{3/2})$ sistrings, and the sizes of the suffix tree and array will grow accordingly.

The benefit comes at search time: in the first part of the search we do not need to consider rotations of the pattern, since all the rotated ways to read the text are already indexed. Since we index $O(n^2H^{3/2})$ strings now, all the different sistrings will exist until depth $h' = \log_\sigma(n^2H^{3/2}) = 2\log_\sigma n + 3/2\log_\sigma H$. We first assume that $H \geq h'$. This means that until depth H we pay $O(H) = O(\log_\sigma n)$. After that depth all the strings are searched, which since $H \geq h'$ are all different and the summation is as before and yields $O(n^2H^{3/2}/\sigma^H)$. Therefore the total search time is

$$O\left(H + \frac{n^2H^{3/2}}{\sigma^H}\right)$$

which is optimized for $H = 2\log_\sigma n + (1/2)\log_\sigma H$. Since this is smaller than h' we take the minimal $H = h'$. For instance $H = x\log_\sigma n$ works for any $x > 2$.

This makes the total search time $O(\log_\sigma n)$ on average. The space complexity bocomes now $O(n^2(\log_\sigma n)^{3/2})$. Trying to use $H < h'$ gives the same complexity of the previous section.

5 A Matching Model for Gray Levels

A different matching model is that the text center must be between the minimum and maximum pattern color surrounding the pattern cell where the text center lies. In this case, we do not enter into a single branch of the suffix tree, but for each pattern cell we must enter into all the text colors which are between the minimum and maximum neighbor of that pattern cell. Say that this number is Δ.

Then we enter into all the Δ^h nodes up to depth h (since all exist). This is $\Delta^{2\log_\sigma n} = n^{2\log_\sigma \Delta}$. After this we have n^2 different strings, we select Δ of them and go to their subtrees. Each selected subtree has average size n^2/σ since we have selected the substrings that are continued with some letter. The recurrence and its solution is

$$T(n^2) = \Delta T(n^2/\sigma) = O(n^{2\log_\sigma \Delta})$$

Therefore the search of a single string (no rotations allowed yet) takes $O(n^\alpha)$, for $\alpha = 2\log_\sigma \Delta$.

If we allow rotations, then all them exist up to depth h. So we have that the amount of work up to there is

$$\sum_{\ell=1}^{h-1} \Delta^\ell \ell^{3/2} \quad = \quad O\left(n^\alpha(\log_\sigma n)^{3/2}\right)$$

and the same happens to the second part of the search. Indexing the rotations is not a good idea now, because the cost to traverse the dense part of the tree is very high. If we index all the rotations up to depth $H = x\log_\sigma n$ for $x > 2$ then the first part of the traversal will cost $O(\Delta^H = n^{x\log_\sigma n}) = \omega(n^\alpha(\log_\sigma n)^{3/2})$.

If we consider that the distribution is uniform then, by taking the maximum (minimum) over 9 samples of a uniform distribution we get at 9/10 of the real maximum (minimum). Hence $\Delta/\sigma = 4/5$ and $\alpha = 2(1 - 1/\log_{5/4}\sigma)$.

6 Conclusions and Future Work

We have proposed a suffix tree index to search two dimensional patterns in two dimensional texts allowing rotations. The proposed method works best for circular and without-holes patterns. Otherwise we should take the biggest circle inside the pattern with that property, search it using the suffix tree, and check the rest directly in the text occurrences.

On average, it is enough that a circle containing more than $2\log_\sigma n$ characters can be found inside the pattern to make the total cost of this verification negligible. This means a circle of radius at least $\sqrt{2/\pi \log_\sigma n}$.

It is possible to extend the model to not needing to match pattern centers against text centers. In this case the number of patterns grows as $O(m^7)$ and therefore there are $O(\ell^{7/2})$ sistrings to search at depth ℓ. The search time becomes $O(\log_\sigma n)^{9/2}$. By indexing all the rotations and center displacements we get $O(\log_\sigma n)$ time again, but at a space cost of $O(n^2(\log_\sigma n)^{7/2})$.

It is also possible to extend the method to 3 dimensions [3]. With center to center assumption we have $O(m^11)$ rotations. This means $O(\ell^{11/3})$ sistrings at depth ℓ. Therefore, at $O(n^3)$ space the total search time becomes $O(\log_\sigma n)^{14/3}$, and if we index all the rotations up to $H = x\log_\sigma n$ with $x > 3$ we will have a space requirement of $O(n^3(\log_\sigma n)^{11/3})$ and a search cost of $O(\log_\sigma n)$. For gray levels we have again $O(n^\alpha)$ (for $\alpha = 2(1 - 1/\log_{14/13}\sigma)$) and indexing more rotations helps little (now $\alpha = 3\log_\sigma \Delta$).

Finally, it is interesting to extend this method to allow errors in the matches. Standard techniques of one-dimensional string matching can be applied [8], but the cost will have a part exponential on the number of errors allowed [7].

References

[1] K. Fredriksson and E. Ukkonen. A rotation invariant filter for two-dimensional string matching. In *Proc. CPM'98*, LNCS 1448, pages 118–125, 1998.

[2] K. Fredriksson and E. Ukkonen. Combinatorial methods for approximate image matching under translations and rotations. year?

[3] K. Fredriksson and E. Ukkonen. Combinatorial methods for approximate pattern matching under rotations and translations in 3D arrays. year?

[4] R. Giancarlo. A generalization of suffix trees to square matrices, with applications. *SIAM J. on Computing*, 24:520–562, 1995.

[5] R. Giancarlo and R. Grossi. On the construction of classes of suffix trees for square matrices: Algorithms and applications. *Information and Computation*, 130:151–182, 1996.

[6] G. Gonnet, R. Baeza-Yates, and T. Snider. *New indices for text: PAT trees and PAT arrays*, pages 66–82. Addison-Wesley, 1992.

[7] G. Navarro and R. Baeza-Yates. A new indexing method for approximate string matching. In *Proc. CPM'99*, LNCS 1645, pages 163–185, 1999.

[8] E. Ukkonen. Approximate string matching over suffix trees. In *Proc. CPM'93*, pages 228–242, 1993.