Similarity in Two-Dimensional Strings

Ricardo A. Baeza-Yates

Depto. de Ciencias de la Computación
Universidad de Chile
Casilla 2777, Santiago, Chile
E-mail: rbaeza@dcc.uchile.cl *

Abstract

In this paper we discuss how to compute the edit distance (or similarity) between two images. We present new similarity measures and how to compute them. They can be used to perform more general two-dimensional approximate pattern matching. Previous work on two-dimensional approximate string matching either work with only substitutions or a restricted edit distance that allows only some type of errors.

1 Introduction

A number of important problems related to string processing lead to algorithms for approximate string matching: text searching, pattern recognition, computational biology, audio processing, etc. Two-dimensional pattern matching with errors has applications on computer vision.

The *edit distance* between two strings a and b, $ed(a, b)$, is defined as the minimum number of *edit operations* that must be carried out to make them equal. The allowed operations are insertion, deletion and substitution of characters in a or b. The problem of *approximate string matching* is defined as follows: given a *text* of length n, and a *pattern* of length m, both being sequences over an alphabet Σ of size σ, find all segments (or "occurrences") in *text* whose edit distance to *pattern* is at most k, where $0 < k < m$. The classical solution is $O(mn)$ time and involves dynamic programming [14].

Krithivasan and Sitalakshmi (KS) [11] proposed the following extension of edit distance for two dimensions. Given two images of the same shape, the edit distance is the sum of the edit distance of the corresponding row images. This definition is justified when the images are transmitted row by row and there are not too many communication errors. However, for many other problems, this distance does not reflect well simple cases of approximate matching in different settings. For example, we could have a match that only has the middle row of the pattern missing. In the definition above, the edit distance would be $O(m^2)$ if all pattern rows are different. Intuitively, the right answer should be at most $2m$, because only m characters were deleted in the pattern

*This work was supported by Fondecyt Grant 95-0622 and was published in COOCON'98, Springer-Verlag LNCS 1449, pages 319-328, August 1998, Taipei, Taiwan.

and m characters are inserted at the bottom. In this paper we extend the edit distance to two dimensions lifting the problem just mentioned and also extending the edit distance to images of different shapes.

This paper is organized as follows. First we discuss previous work on two-dimensional pattern matching with errors and image similarity. Next, we introduce new notions of similarity between two-dimensional strings or images. As for the one-dimensional counterpart, we consider first the comparison of two images and we also discuss how to compute efficiently these new similarity measures.

Then, we look at the problem of finding approximate matches with at most k errors of a rectangular pattern image of size $m \times m$ in a larger rectangular image (the text) of size $n \times n$. We present a $O(m^3n^2)$ worst case algorithm, and a $O(n^2 k \log_\sigma m \; / m^2)$ for k up to $O(m^2/\log m)$ where σ denotes the size of the (finite) alphabet, for two of the new measures that we define. We end by discussing possible extensions and open problems. We denote by σ the size of the (finite) alphabet.

2 Previous Work

Two-dimensional approximate string matching usually considers only substitutions for rectangular patterns, which is much simpler than the general case with insertions and deletions (because in this case, rows and/or columns of the pattern can match pieces of the text of different length). For substitutions, the pattern shape matches the same shape in the text.

If we consider matching the pattern with at most k substitutions, one of the best results on the worst case is due to Amir and Landau [2] achieving $O((k + \log \sigma)n^2)$ time but using $O(n^2)$ space. A similar algorithm is presented in Crochemore and Rytter [5]. Ranka and Heywood [13], on the other hand, solve the problem in $O((k + m)n^2)$ time and $O(kn)$ space. Amir and Landau also present a different algorithm running in $O(n^2 \log n \log \log n \log m)$ time. On average, the best algorithm is due to Karkkäinen and Ukkonen [9], with its analysis and space usage improved by Park [12]. The expected time is $O((n^2 k \log_\sigma m)/m^2)$ for

$$k \leq \left\lfloor \frac{m}{\lceil \log_\sigma (m^2) \rceil} \right\rfloor \frac{m}{2} - 1 \;\approx\; \frac{m^2}{4 \log_\sigma m}$$

using $O(m^2)$ space ($O(k)$ space on average). This time result is optimal for the expected case.

Under the KS definition, Krithivasan [10] presents an $O(m(k + \log m)n^2)$ algorithm that uses $O(mn)$ space. This was improved (for $k < m$) by Amir and Landau [2] to $O(k^2n^2)$ worst case time using $O(n^2)$ space. Amir and Farach [1] also considered non-rectangular patterns achieving $O(k(k + \sqrt{m \log m}\sqrt{k \log k})n^2)$ time. This algorithm is very complicated and non-practical because it uses numerical convolutions.

Very recently, Baeza-Yates and Navarro [4] obtained the first fast algorithm on average for the KS model. They use a filter algorithm based in multiple approximate string matching, achieving $O(n^2 k \log_\sigma m \; / m^2)$ average-case behavior for $k < m(m+1)/(5 \log_\sigma m)$, and using $O(m^2)$ space. This time matches the best known result for the same problem allowing just substitutions and is optimal [9], being the upper bound on k only a bit smaller. For higher error levels, they present an algorithm with time complexity $O(n^2 k/(w\sqrt{\sigma}))$ (where w is the size in bits of the computer word), which works for $k < m(m+1)(1 - e/\sqrt{\sigma})$.

Another related problem is geometric matching, where we have to match a geometric figure or a set of points. In this case, the problem is in a continuous space rather than a discrete space and usually the Hausdorff measure [3] is used.

There are other approaches to matching images, which are very different to our approach (which belongs to what is called combinatorial pattern matching). Among them we can mention techniques used in pattern matching related to artificial intelligence (for example image processing and neural networks [15]) and techniques used in databases (extracting features of the image, like color histograms [6, 7]).

3 Extending the Edit Distance

Let a and b be two images of size $nr \times nc$ and $mr \times mc$ respectively. In the sequel we use $row_i(a)$ to denote the i-th row of a and $col_i(a)$ to denote the i-th column of a. For example, the KS distance is given by

$$KS(a,b) = \sum_{i=1}^{nr} ed(row_i(a), row_i(b))$$

with the restriction that $nr = mr$. We also use the L-shape idea of Giancarlo [8] used for extending suffix trees to two dimensions. We denote by $LS_{i,j}(a)$ the L-shaped string consisting of the first (left) j elements of the i-th row and the first (top) $i-1$ elements of the j-th column.

Because our main motivation is approximate matching, we assume that the pattern and a text subimage are compared from top to bottom and from left to right. That is, a sub-image can be enlarged by extending it the bottom or/and the right side. It can be argued that a pattern can match better fixing a different corner, but this does not make any difference, because that only changes the text position where the match will be reported, and still only one match is found. Another convention is that the text occurrence must have the same shape of the pattern. Otherwise, we may have occurrences that have at most k errors that basically do not count unmatched characters on the boundaries, which is not fair. Hence, although our similarity measures work for two images of different size, they will be used later for subimages in the text that have the same shape as the pattern.

First, we solve the limitation of the KS model to handle deletions or insertions of whole rows. The solution is simple, we just treat each row as a single string which is compared to other rows using the normal edit distance (that is, only one dimension). If $R_{i,j}$ is the distance between rows 1 and i of image a and rows 1 and j of image b, we have that

$$R_{i,j} = \min(R_{i-1,j} + nc, R_{i,j-1} + mc, R_{i-1,j-1} + ed(row_i(a), row_j(b)))$$

where the boundary conditions are $R_{i,0} = i \cdot nc$ and $R_{0,j} = j \cdot mc$, and the distance between the two images is given by $R(a,b) = R_{nr,mr}$.

In the example given in the introduction, the distance is reduced to less or equal than $2m$ instead of being $O(m^2)$ as in the KS model. Similarly, we could use columns instead of rows, obtaining $C(a,b)$. This model is much more fair than the KS model. Although we use rectangular images, this measure also works for any images where rows are connected and continuous.

Generalizing this idea to insertions and deletions at the same time in rows and/or columns is not as simple. Suppose that we have two subimages that we want to compare. One alternative is to decompose the border of a subimage in rows or columns. Then we can use the following decompositions:

1. removing one row or one column from one of the subimages or

2. removing one row or one column in the same side of each subimage and computing the edit distance between them.

We can apply dynamic programming to find the best possible decomposition. That is, if $RC_{i,j,k,\ell}$ is the distance between the left-top corner of a bounded by row i and column j and the left-top corner of b bounded by row k and column ℓ, we have that $RC_{i,j,k,\ell}$ is the minimum of the following values:

- $RC_{i-1,j,k,\ell}+j$, $RC_{i,j-1,k,\ell}+i$, $RC_{i,j,k-1,\ell}+\ell$, and $RC_{i,j,k,\ell-1}+k$, which corresponds to deleting one row or column in one sub-image; and

- $RC_{i-1,j,k-1,\ell} + ed(pref_j(row_i(a)), pref_\ell(row_k(b))))$ and $RC_{i,j-1,k,\ell-1} + ed(pref_i(col_j(a)), pref_k(col_\ell(b))))$ where $pref_i(s)$ denotes the first i characters of string s (a prefix) and which corresponds to comparing two rows at the bottom or two columns at the right.

The boundary conditions are $RC_{0,0,i,j} = RC_{i,j,0,0} = i \cdot j$. The distance $RC(a,b)$ is given by $R_{nr,nc,mr,mc}$. Figure 1 shows all these cases. This distance can also be applied to any convex image, for example circles or other regular polygons.

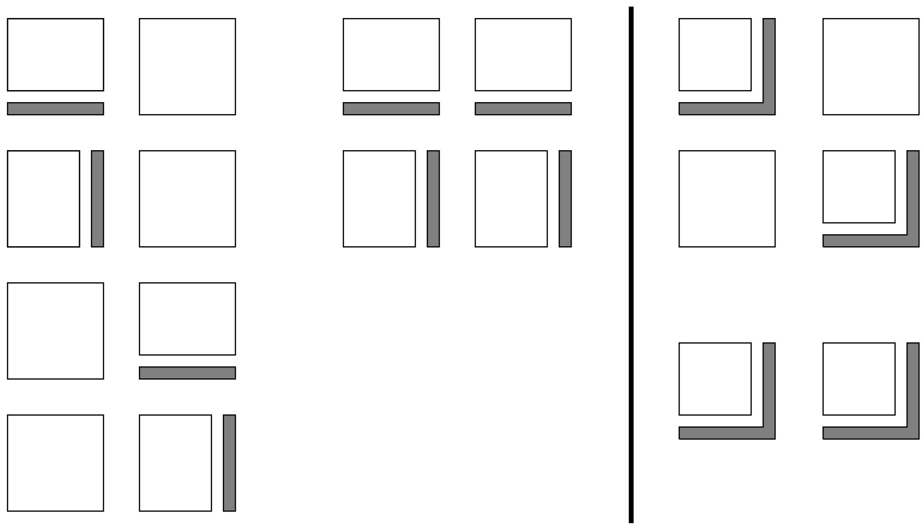

Figure 1: Decomposition used in RC (left, 6 cases) and L (right, 3 cases).

Nevertheless, this distance does not handle cases where we want to change at the same time a row and a column (for example, a motivation could be scaling). For that we use the L-shape mentioned earlier. So, we can also decompose the border of a subimage using L-shapes and we can have the same extensions as for rows or columns. To compare two L-shapes we see them as two one-dimensional strings. Then we have the following cases to find the minimal decomposed distance:

- $L_{i-1,j-1,k,\ell} + i + j - 1$ and $L_{i,j,k-1,\ell-1} + k + \ell - 1$ which corresponds to removing an L-shape in a subimage; and
- $L_{i-1,j-1,k-1,\ell-1} + ed(LS_{i,j}(a), LS_{k,\ell}(b)))$ which corresponds to comparing two L-shapes.

The boundary conditions are the same as the RC measure and the final distance is similarly given by $L(a,b) = L_{nr,nc,mr,mc}$. Figure 1 shows the decompositions associated to L. We will see later that this definition can be simplified by using the fact that one row and one column are considered at the same time when using L-shapes.

Finally, we can have a general distance $All(a,b)$ that uses both decompositions at the same time (RC and L) computing the minimal value of all possible cases. It is easy to show that $KS(a,b) \geq R(a,b) \geq RC(a,b) \geq All(a,b)$ and that $L(a,b) \geq All(a,b)$ because each case is a subset of the next. On the other hand, there are cases where $RC(a,b)$ will be less than $L(a,b)$ and vice versa. In fact, Figure 2 this is shown together with other examples, where each color is a different symbol. The last example shows that combining RC and L can actually lead to a distance less than each separate case.

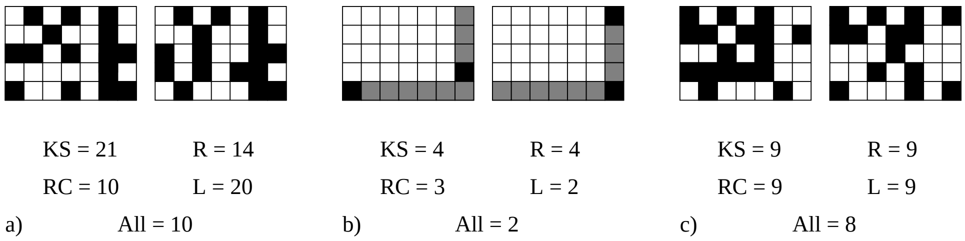

Figure 2: Three examples for our new measures.

4 Computing the Distances

For sake of simplicity and without loss of generality assume that $nr = nc = mr = mc = m$. A direct implementation for R would take $O(m^4)$ time and $O(m^2)$ space, while for RC and L would require $O(m^6)$ time and $O(m^4)$ space. The later, in particular, is prohibitive even for small images. However, this can be done better. The space is easily reduced to $O(m)$ for R and $O(m^3)$ to RC and L by noticing that we only need to store the boundary of the matrices of the dynamic programming computation as they are computed incrementally.

However, the computation of L can be simplified further by noticing that to compute the best decomposition $i - j$ and $k - \ell$ are always constant (in fact, is equal to $nr - nc$, which for squares images is 0). This means that only a quadratic number of entries must be computed, which implies a matrix boundary of size $O(m)$ and a running time of $O(m^4)$.

We can also improve the computation of RC by noticing that the edit distances of prefixes involved can also be computed incrementally. That is, we store the distances between each two last prefixes computed and we use them when we extend the distance between two prefixes by one or two

Measure	Time	Space
KS	m^3	m
R, C	m^4	m
L	m^4	m
RC	m^4	m^3
All	m^4	m^3

Table 1: Time and space complexity to compute the distance.

characters. This needs additional space which matches the improved space bound mentioned before. Therefore, each edit distance needed in the dynamic programming computation can be computed in constant time, reducing the total time to $O(m^4)$ for *RC*, and hence for *All* considering that L requires the same time. These optimizations allows to handle patterns of reasonable size (say up to 50×50).

Table 1 summarizes the space and time complexity obtained for all the measures, including KS. We can see that our measures need only one order of magnitude more time with respect to KS, using the same space, except for *RC* and *All*.

5 Two Dimensional Approximate Pattern Matching

Now we discuss approximate two-dimensional pattern matching. In this section, image a is the text, and image b is the pattern. We use k to denote the maximum number of errors allowed. For simplicity we use $nc = nr = n$ and $mc = mr = m$.

The straight forward technique to search an image in a larger image is by scanning sequentially the larger image (the text) and computing its distance to the pattern. For R, we will scan the text from top to bottom, using m columns each time, from column j to $j + mc - 1$, starting with $j = 1$ and ending with $j = n - m + 1$. For each j, we use the same dynamic programming formulas, changing the boundary conditions to allow the pattern to match in any possible row i. That is, we use $R_{i,0} = 0$ and $R_{0,j} = j \cdot mc$. We report a match whenever we have at most k errors and we have considered all the rows of the pattern. That is, when $R_{i,mr} \leq k$. For each scan, the time complexity is $O(nm^3)$ and we repeat that n times. Therefore, the total worst case time is $O(n^2m^3)$, using $O(m)$ extra space. This algorithm also works for C by exchanging rows and columns.

Now, we consider a fast algorithm on average. For the R (or C) measures we can use the same fast expected time algorithm of Baeza-Yates and Navarro [4]. This algorithm uses a filter that searches all the pattern rows (or columns) with a multiple approximate string matching algorithm to find potential matching areas (see Figure 3). We only change the verification phase (each potential area found by the filter must be verified using the $O(n^2m^3)$ worst case searching algorithm described before) by using R (or C) to compute the distance in a fixed area (see Figure 3). Because this algorithm works up to certain threshold level up to where the verification time is negligible, the fact that computing R (or C) takes $O(m)$ the time of computing KS, will only change the maximal k for which the algorithm performs well. Because the maximal k depends on the inverse of the

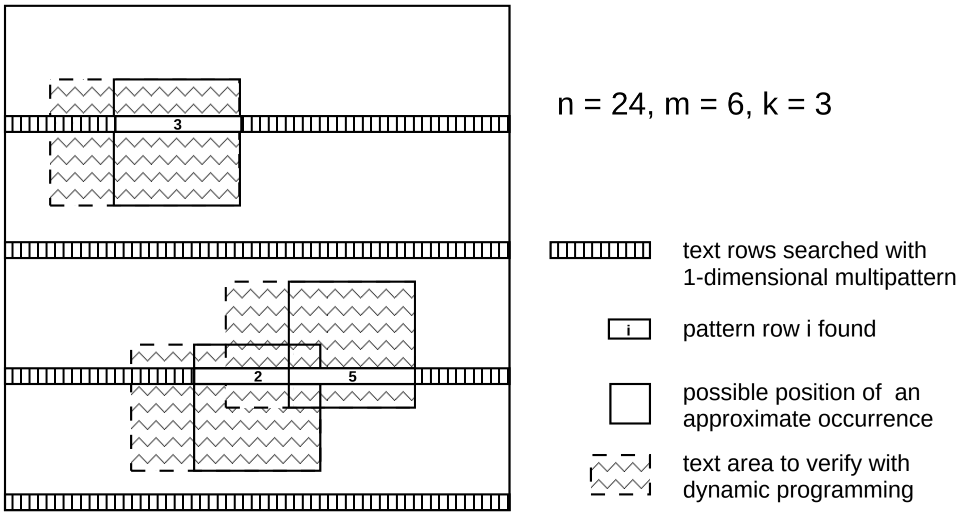

Figure 3: Filtering algorithm to find potential matches.

logarithm of the verification algorithm plus two, the maximal k changes from $m(m+1)/(5\log_\sigma m)$ to $k < m(m+1)/(7\log_\sigma m)$.

Therefore, we obtain the same average-case time bound of $O(n^2 k \log_\sigma m\ /m)$ and almost the same error level bound for which the expected time result is valid. That is, $k < m(m+1)/(7\log_\sigma m)$, using $O(m)$ extra space. This expected time is optimal [9]. How to extend these results to the other measures is still an open problem.

6 Concluding Remarks

Our measures can be easily extended to more dimensions. For d-dimensions we use $(d-1)$-dimensional strings for the decompositions. The only drawback is that the number of cases grows exponentially with the number of dimensions. Then, computing $All(a,b)$ for d-dimensional strings would require $O(2^d n^{2d})$ time and $O(n^d)$ space.

An open problem is to design optimal worst-case time algorithms for approximate searching using the new measures. Also, finding fast filtering algorithms for the other measures is matter of future research.

Neither of the new measures defined can handle scaling transformations nor rotations. A more realistic distance can be defined using the following idea, which tries to define the *largest common image* of two images, which generalizes the concept of longest common subsequence of one-dimensional strings. Given two images, find a set of position pairs that match exactly in both images subject to the following restrictions:

1. The set of positions for the same pattern are disjoint;

2. a suitable order given by the position values is the same for both images (for example, image pixels can be sorted by their $i+j$ value, using the value of i in the case of ties); and

3. the total size of the set of positions is maximized.

For the edit distance, condition 3 has to be changed to:

3. Minimize the number of mismatches, insertions and deletions needed to obtain the set of matching positions.

This model may match a rotated pattern, because no corner is fixed. Figure 4 gives an example. All pieces of the pattern not in the text corresponds to deletions and mismatches and should be counted. In the text, black regions are not counted, because correspond to mismatches. All other pieces are insertions in the pattern. It is not clear that the minimal string editing solution gives the same answer as the largest common set of sub-images. Also, it could be argued that characters inserted/deleted on external borders should not be counted as errors.

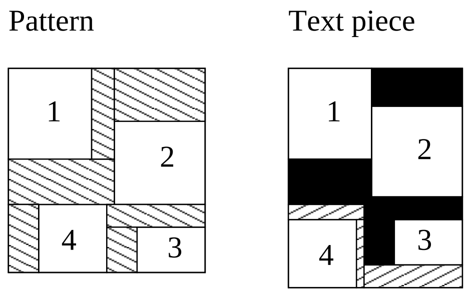

Figure 4: Example of largest common image.

The approximate two-dimensional pattern matching problem can be stated as usual using the above definition as searching for all rectangular subimages of the text that have edit distance at most k with the pattern. An alternative definition would be to find all pieces of the text that have at least $m^2 - k$ matching positions with the pattern.

Acknowledgements

We would like to thank the comments of Gonzalo Navarro, Ladan Rostami, Sven Schuerier (which in particular pointed out the simplification for computing the L measure), and the unknown referees. This work was supported by Fondecyt Grant 95-0622.

References

[1] A. Amir and M. Farach. Efficient 2-dimensional approximate matching of non-rectangular figures. In *Proc. SODA'91*, pages 212–223, San Francisco, CA, Jan 1991.

[2] A. Amir and G. Landau. Fast parallel and serial multidimensional approximate array matching. *Theoretical Computer Science*, 81:97–115, 1991. Also as report CS-TR-2288, Dept. of Computer Science, Univ. of Maryland, 1989.

[3] M. Atallah. A linear time algorithm for the Hausdorff distance between convex polygons, *Information Processing Letters* 17, 207-209, 1983.

[4] R. Baeza-Yates and G. Navarro. Fast two-dimensional approximate string matching. In *LATIN'98*, Campinas, Brazil, April 1998. Springer-Verlag LNCS 1380, edited by C. Lucchesi and A. Moura, 341-351.

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

[6] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *J. of Intelligent Information Systems*, 3:231-262, 1994.

[7] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Petkovic, and W. Equitz.

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

[9] J. Karkkäinen and E. Ukkonen. Two and higher dimensional pattern matching in optimal expected time. In *Proc. SODA'94*, pages 715–723. SIAM, 1994.

[10] K. Krithivasan. Efficient two-dimensional parallel and serial approximate pattern matching. Technical Report CAR-TR-259, University of Maryland, 1987.

[11] K. Krithivasan and R. Sitalakshmi. Efficient two-dimensional pattern matching in the presence of errors. *Information Sciences*, 43:169–184, 1987.

[12] K. Park. Analysis of two dimensional approximate pattern matching algorithms. In *Proc. CPM'96*, LNCS 1075, pages 335–347, 1996.

[13] S. Ranka and T. Heywood. Two-dimensional pattern matching with k mismatches. *Pattern recognition*, 24(1):31–40, 1991.

[14] P. Sellers. The theory and computation of evolutionary distances: pattern recognition. *J. of Algorithms*, 1:359–373, 1980.

[15] P. Suetens, P. Fua, and A. Hanson. Computational Strategies for Object Recognition, *ACM Computing Surveys*, 24:5-62, 1992.