Sequential and Indexed Two-Dimensional Pattern Matching Allowing Rotations

Kimmo Fredriksson* Gonzalo Navarro† Esko Ukkonen ‡

Abstract

We present new and faster algorithms to search for a 2-dimensional pattern in a 2-dimensional text allowing any rotation of the pattern. This has applications such as image databases and computational biology. We consider the cases of exact and approximate matching under several matching models, improving all previous results. We focus on sequential algorithms, where only the pattern can be preprocessed, as well as on indexed algorithms, where the text is preprocessed and an index built on it. On sequential searching we derive average-case lower bounds and then obtain optimal average-case algorithms for all the matching models. At the same time, these algorithms are worst-case optimal. On indexed searching we obtain search time polylogarithmic on the text size, as well as sublinear time in general for approximate searching.

Keywords: Template matching, combinatorial algorithms, image processing, string matching.

1 Introduction

We consider the problem of finding the exact and approximate occurrences of a two-dimensional *pattern* of size $m \times m$ cells in a two-dimensional *text* of size $n \times n$ cells, when all possible rotations of the pattern are allowed. This problem is often called *rotation invariant template matching* in the signal processing literature. Template matching has numerous important applications from science to multimedia, for example in image processing, content based information retrieval from image databases, fingerprint processing, optical character recognition, geographic information systems, processing of aerial and astronomical images, and searching for known substructures (such as proteins) from three dimensional models of biological viruses, to name a few.

In many applications, "close enough" matches of the pattern are also accepted. To this end, the user may specify a parameter k, such that matches that have at most k differences with the pattern should be accepted.

The traditional approach to the problem [9] is to compute the cross correlation between each text location and each rotation of the pattern template. This can be done reasonably efficiently using the Fast Fourier Transform (FFT), requiring time $O(Kn^2 \log n)$ where K is the number of rotations sampled. Typically K is $O(m)$ in the 2-dimensional (2D) case, and $O(m^3)$ in the 3D case, which makes the FFT approach very slow in practice. Other approaches to the problem are reviewed in [33].

*Department of Computer Science, University of Joensuu, Finland. kfredrik@cs.joensuu.fi. Part of this work developed while the author was at University of Helsinki, supported by ComBi and the Academy of Finland.

†Center for Web Research, Department of Computer Science, University of Chile, Chile. gnavarro@dcc.uchile.cl. Funded by Millenium Nucleus Center for Web Research, Grant P01-029-F, Mideplan, Chile.

‡Department of Computer Science, University of Helsinki, Finland. ukkonen@cs.helsinki.fi. Work supported by the Academy of Finland.

Recently, a different approach to template matching, called *combinatorial template matching*, has emerged. The idea is to generalize well-studied string matching techniques. String matching deals with the problem of searching for a one-dimensional pattern on a one-dimensional text, and the aim is to generalize the problem to two and more dimensions.

Examples of combinatorial template matching algorithms are [21, 8, 2, 26, 7, 31, 32, 25]. They address different search problems, from exact pattern matching to handling insertion and deletion errors. However, they do not permit searching for the pattern in rotated form. This was stated as a major open problem already in 1992 [5].

Rotation invariant template matching was first considered from a combinatorial point of view in [17, 19]. The first problem was to define what was to be considered a *match*. If we consider the pattern and text as regular grids, then defining the notion of matching becomes nontrivial when we rotate the pattern: Since every pattern cell intersects several text cells and vice versa, it is not clear what should match what. They proposed a simple model such that (1) the geometric center of the pattern has to align with the center of a text cell (this is called the center-to-center assumption); (2) the text cells involved in the match are those whose geometric centers are covered by the pattern; (3) each text cell involved in an occurrence should match the value of the pattern cell that covers its center. This matching model is called *Exact* in this paper. They presented an $O(n^2)$ average time algorithm for this model. An $O(m^3n^2)$ worst-case time algorithm for a rather similar model was presented in [3]. They show that this algorithm is worst-case optimal.

An extension of the exact matching model more suitable for gray level images states that the value of each text cell involved in a match must be between the minimum and maximum value of the 9 neighboring cells surrounding the corresponding pattern cell [19]. This model is called *MinMax*. We are not aware of any previous algorithm for this model.

The exact model (in fact a 3D version) was extended in [20] such that there may be a limited number k of mismatches between the pattern and its occurrence. Under this *Mismatches* model an $O(k^4n^3)$ average time algorithm was obtained, as well as an $O(k^2n^3)$ average time algorithm for computing a lower bound on the distance. We show here that a 2D version of the same idea yields $O(k^{3/2}n^2)$ average time for any $0 \leq k < m^2$. For very small $k < \pi r^2 e^{-\pi r^2/\sigma}$ ($r = O(m)$), an $O(k^{1/2}n^2)$ average time algorithm was given in [12].

Finally, a more refined model [12, 20] suitable for gray level images adds up the absolute values of the differences in the gray levels of the pattern and text cells supposed to match, and puts an upper limit k on this sum. Under this *Accumulated* model average time $O((k/\sigma)^{3/2}n^2)$ was achieved, assuming that the cell values are uniformly distributed among σ gray levels.

In this paper we address all the above defined matching models. We (1) find tight lower bounds for the average complexity of the search problem, for all the matching models (worst-case complexity is already known); (2) show a technique to make all the algorithms that follow optimal in the worst case, without affecting their average complexity; and (3) consider each of the models in turn and develop optimal average-case search algorithm for them. Therefore, we solve the search problem for all the matching models considered in optimal worst- and average-case time simultaneously.

Our main technique is to extract linear strips from the pattern at every possible rotation, and search for those strips simultaneously along some text rows, permitting or not mismatches. The use of optimal exact and approximate one-dimensional multipattern search algorithms is crucial to obtain optimality in our problem. Our space requirement and preprocessing time is always

polynomial in the pattern size.

All the algorithms considered up to now are *sequential*. This means that they can preprocess the pattern but not the text. An alternative scenario is that the text (that is, the large image) is known in advance and can be preprocessed to speed up searches later. This is called *indexed* searching.

In this work we give the first algorithms for indexed searching. The data structure we use is based on tries. Suffix trees for two-dimensional texts have been considered, for example, in [22, 23, 24]. The idea of searching for a rotated pattern using a "suffix" array of spiral-like strings is mentioned in [24], but only for rotations of multiples of 90 degrees. The problem is much more complex if we want to allow any rotation.

Our search times are polylogarithmic for exact searching and sublinear[1] on average when some conditions on the mismatch threshold are met. In most the cases the index needs $O(n^2)$ (that is, linear) space and it can be constructed in average time $O(n^2 \log_\sigma n)$.

Model	Search time	Type	Comments/conditions
Exact $\Omega(n^2 \log_\sigma(m)/m^2)$	n^2	Seq	Previous result
	$n^2 \log_\sigma(m)/m^2$	Seq	Optimal
	$(\log_\sigma n)^{5/2}$	Ind	$\pi m^2/4 \geq 2\log_\sigma n$
MinMax $\Omega(n^2 \log(m)/m^2)$	$O(n^2 \log(m)/m^2)$	Seq	Optimal
	$(\log_\sigma n)^{3/2} n^{2(1-\log_\sigma(5/4))}$	Ind	$\pi m^2/4 \geq 2\log_\sigma n$
Mismatches $\Omega(n^2(k+\log_\sigma m)/m^2)$	$n^2 k^{3/2}$	Seq	Previous result, adapted by us
	$n^2 k^{1/2}$	Seq	Previous result $k < \pi r^2 e^{-\pi r^2/\sigma}$, $r = O(m)$
	$n^2(k+\log_\sigma m)/m^2$	Seq	Optimal, $\alpha < 1/2(1-O(1/\sigma))$
	$(2\log_\sigma n)^{k+3/2}\sigma^k$	Ind	$\pi m^2/4 \geq 2\log_\sigma n > k_H$
	$n^{2(\alpha+H^H_\sigma(\alpha))} m^3 k$	Ind	$m^2 \geq \max(k_H, 2\log_\sigma n)$
Accumulated $\Omega(n^2(k/\sigma+\log_\sigma m)/m^2)$	$n^2(k/\sigma)^{3/2}$	Seq	Previous result, adapted by us
	$n^2(k/\sigma+\log m)/m^2$	Seq	$\alpha < \sigma/(4e)$. Optimal except $\log_\sigma m \leq k \leq (\sigma/e)\log m$
	$(k+2\log_\sigma n)^{k+3/2} n^{2\log_\sigma 2}$	Ind	$\pi m^2/4 \geq 2\log_\sigma n > k_A$
	$n^{2(\log_\sigma 2+H^A_\sigma(\alpha))} m^3 k$	Ind	$m^2 \geq \max(k_A, 2\log_\sigma n)$

Table 1: Simplified average sequential and indexed time complexities achieved under different models. We include the average-time sequential lower bounds we have proved. All our sequential algorithms are worst-case optimal.

Table 1 shows our main achievements. All the results are on the average, using a probabilistic model where σ is the alphabet size and the cell values are uniformly and independently distributed over those σ values. We have several different results for each model. In particular, some algorithms are sequential and others are indexed. In the mismatches and accumulated models we call $\alpha = k/m^2$ (note that $\alpha < 1$ for mismatches and $\alpha < \sigma$ for accumulated) and k_H and k_A denote the maximum

[1] Throughout this paper we speak of "sublinearity" to mean less than n^2, which is the input size.

k values up to where some techniques work: $k_H = k/(1-e/\sigma)$ and $k_A = k/(\sigma/(2e)-1)$. Moreover, $H^H_\sigma(\alpha) = -\alpha\log_\sigma(\alpha) - (1-\alpha)\log_\sigma(1-\alpha)$ and $H^A_\sigma(\alpha) = -\alpha\log_\sigma(\alpha) + (1+\alpha)\log_\sigma(1+\alpha)$.

The algorithms are easily generalized for handling large databases of images. That is, we may store any number of images in the index, and search the query pattern simultaneously in all the images. The time complexities remain the same, if we now consider that n^2 denotes the size of the whole image library.

We have also considered alternative models where pattern centers are used instead of text centers to define what should match what, where no center-to-center assumption holds, and where there are more dimensions. In most cases we obtain basically the same algorithms, but there are a few interesting exceptions.

Partial preliminary versions of this work appeared in [14, 16, 15].

The organization of the paper follows. We start by proving giving some preliminary concepts that are used throughout the paper, in Section 2. In Section 3, a lower bound on the average sequential complexity of the search problem, both for exact and for approximate searching, is given. In Section 4 we give optimal worst-case algorithms for all the matching models in one shot. Sections 5 to 8 are devoted to optimal/efficient average-case algorithms for the exact, minmax, mismatches and accumulated models, respectively. Section 9 deals with some alternative formulations of the problem and briefly shows how the results could be extended to more dimensions. We give our conclusions and future work directions in Section 10.

2 Preliminaries

2.1 Matching Functions

Let $T = T[1..n, 1..n]$ and $P = P[1..m, 1..m]$ be arrays of unit squares, called *cells*, in the (x, y)-plane. Each cell has a value in ordered finite alphabet Σ, which we sometimes call "colors". The size of the alphabet is denoted by $\sigma = |\Sigma|$. The corners of the cell for $T[i, j]$ are $(i-1, j-1)$, $(i, j-1)$, $(i-1, j)$ and (i, j). The center of the cell for $T[i, j]$ is $(i - \frac{1}{2}, j - \frac{1}{2})$. The array of cells for pattern P is defined similarly. The center of the whole pattern P is the center of the cell in the middle of P. We assume for simplicity that m is odd, hence the center of P is the center of cell $P[\frac{m+1}{2}, \frac{m+1}{2}]$.

Assume now that P has been moved on top of T using a rigid motion (translation and rotation), such that the center of P coincides exactly with the center of some cell of T (this is called the *center-to-center assumption*). The location of P with respect to T can be uniquely given as $((i, j), \theta)$ where (i, j) is the cell of T that matches the center of P, and θ is the angle between the x-axis of T and the x-axis of P. The (approximate) occurrence between T and P at some location is defined by comparing the values of the cells of T and P that overlap. We will use the centers of the cells of T for selecting the comparison points. That is, for the pattern at location $((i, j), \theta)$, we look which cells of the pattern cover the centers of the cells of the text, and compare those cell values. See the leftmost plot of Figure 1.

More precisely, assume that P is at location $((i, j), \theta)$. For each cell $[r, s]$ of T whose center belongs to the area covered by P, let $[r', s']$ be the cell of P whose area covers the center of $[r, s]$. Then $M_{(i,j),\theta}([r, s]) = [r', s']$. Our algorithms compare the cell $T[r, s]$ against the cell $P[M_{(i,j),\theta}([r, s])]$.

Hence the *matching function* $M_{(i,j),\theta}$ is a function from the cells of T to the cells of P. Now consider what happens to $M_{(i,j),\theta}$ when (i, j) is fixed and angle θ grows continuously, starting from

$\theta = 0$. Function M changes only at the values of θ such that some cell center of T hits some cell boundary of P. It was shown in [17] that this happens $O(m^3)$ times, when P rotates full 360 degrees. This result was shown to be also a lower bound in [3]. Hence there are $\Theta(m^3)$ relevant orientations of P to be checked. The set of angles for $0 \leq \theta \leq \pi/2$ is

$$A = \{\beta, \pi/2 - \beta \mid \beta = \arcsin \frac{h + \frac{1}{2}}{\sqrt{i^2 + j^2}} - \arcsin \frac{j}{\sqrt{i^2 + j^2}};$$
$$i = 1, 2, \ldots, \lfloor m/2 \rfloor; j = 0, 1, \ldots, \lfloor m/2 \rfloor; h = 0, 1, \ldots, \lfloor \sqrt{i^2 + j^2} \rfloor\}.$$

By symmetry, the set of possible angles θ, $0 \leq \theta < 2\pi$, is

$$\mathcal{A} = A \ \cup \ A + \pi/2 \ \cup \ A + \pi \ \cup \ A + 3\pi/2.$$

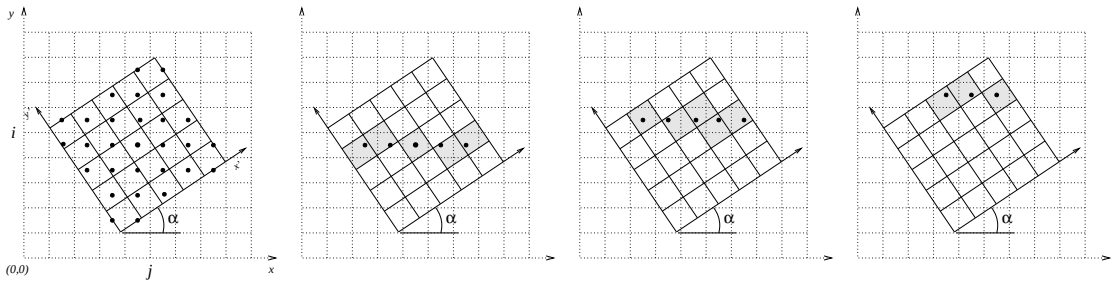


Figure 1: Each text cell is matched against the pattern cell that covers the center of the text cell. For each angle θ, a set of features is read from P.

For the rest of the paper, it is important to understand how the $O(m^3)$ bound is obtained. Consider a given pattern cell, at Euclidean distance ℓ from the pattern center. As the pattern rotates 360 degrees around its center, the pattern cell describes a circle of circumference $2\pi\ell$, and hence its cell borders touch $O(\ell)$ different text centers. Each such hit defines a border between two angle ranges that must be taken as different, because there is at least one text cell that will differ in its matching function M. If we add up the $O(\ell)$ different angles defined for all the m^2 pattern cells, at distances 1 to $O(m)$ from the center, we get the $O(m^3)$ different angles. Much more difficult is to prove that a sufficient number of these angles are indeed different and hence there are $\Theta(m^3)$ different rotations, see [3].

In general, note that if we consider a set of c pattern cells which are up to distance ℓ from the center, then there will be only $\Theta(c\ell)$ relevant angles to match them. Note that the set of angles $\mathcal{B}$ defined by the c cells chosen is a subset of $\mathcal{A}$. The exact size of $\mathcal{B}$ depends on how the cells are chosen. If a match of such cell set is found and we want to extend it to an occurrence of P, then there are $O(|\mathcal{A}|/|\mathcal{B}|) = O(m^3/(c\ell))$ possible orientations where the matching function M for the whole P can change as long as it does not change for those c cells. Hence we may have to check $O(m^3/(c\ell))$ possible orientations. The reason is that the $O(m^3)$ angles distribute uniformly enough over the 360 degrees.

More precisely, assume that $\mathcal{B} = (\gamma_1, \ldots, \gamma_K)$, and that $\gamma_i < \gamma_{i+1}$. Therefore, the text cells corresponding to the chosen c cells can be read for angle γ_i using any θ such that $\gamma_i \leq \theta < \gamma_{i+1}$.

On the other hand, there are $O(|\mathcal{A}|/|\mathcal{B})|$ angles $\beta \in \mathcal{A}$ such that $\gamma_i \leq \beta < \gamma_{i+1}$. If there is an angle γ_i such that the c cells of P match the text, then P may occur with any such angle β.

2.2 Matching Models

We consider four matching models in this paper. These can be defined on top of the matching function just defined. Assume P is at location $((i,j),\theta)$ and let us call $M = M_{(i,j),\theta}$. Hence the following models are defined:

Exact: defines condition

$$T[r,s] \quad = \quad P(M[r,s]), \quad \forall [r,s] \in [1 \ldots n, 1 \ldots n] \text{ such that } M[r,s] \in [1 \ldots m, 1 \ldots m]$$

MinMax: defines condition

$$\min\{P(M[r,s]+[\delta,\delta']),\ -1 \leq \delta,\delta' \leq 1\} \leq T[r,s] \leq \max\{P(M[r,s]+[\delta,\delta']),\ -1 \leq \delta,\delta' \leq 1\},$$

$$\forall [r,s] \in [1 \ldots n, 1 \ldots n] \text{ such that } M[r,s] \in [1 \ldots m, 1 \ldots m]$$

where we have defined $+$ as the pairwise summation over cells.

Mismatches: defines number

$$\sum\{\text{if } T[r,s] = P(M[r,s]) \text{ then 0 else 1}, \quad [r,s] \in [1 \ldots n, 1 \ldots n], \quad M[r,s] \in [1 \ldots m, 1 \ldots m]\}$$

Accumulated: defines number

$$\sum\{|T[r,s] - P(M[r,s])|, \quad [r,s] \in [1 \ldots n, 1 \ldots n], \quad M[r,s] \in [1 \ldots m, 1 \ldots m]\}$$

Given the models, the exact and minmax search problems are to report all the triples (i,j,θ) such that their matching condition is met, while the mismatches and accumulated search problems are to report all the triples (i,j,θ) such that their defined value at that point does not exceed a given threshold k.

2.3 Features

As shown in [17], any match of a pattern P in a text T allowing arbitrary rotations must contain a so-called "feature", that is, a one-dimensional string obtained by reading a line of the pattern in some angle and crossing the center. These features are used to build a filter for finding the position and orientation of P in T.

We now define a *set* of linear features (strings) for P (see Figure 1). The length of a particular feature is denoted by u, and the feature for angle θ and row q is denoted by $F^q(\theta)$. Assume for simplicity that u is odd. To read a feature $F^q(\theta)$ from P, let P be on top of T, on location $((i,j),\theta)$. Consider cells $T[i - \frac{m+1}{2} + q, j - \frac{u-1}{2}], \ldots, T[i - \frac{m+1}{2} + q, j + \frac{u-1}{2}]$. Denote them as $t_1^q, t_2^q, \ldots, t_u^q$. Let c_i^q be the value of the cell of P that covers the center of t_i^q. The (horizontal) feature of P with angle θ and row q is now the sequence $F^q(\theta) = c_1^q c_2^q \cdots c_u^q$. Note that this value depends only on q, θ and P, not on T.

The sets of angles for the features are obtained the same way as the set of angles for the whole pattern P. As explained in Section 2.1, the set of angles $\mathcal{B}^q$ for the feature set F^q is subset of $\mathcal{A}$. The size of $\mathcal{B}$ varies from $\Theta(u^2)$ (for features crossing the center of P) to $\Theta(um)$ (for features at distance $\Theta(m)$ from the center of P). Therefore, if a match of some feature $F^q(\theta)$ is found, and the feature is at distance r from the center of P, then there are $O(m^3/(ur))$ possible orientations to be verified for an occurrence of P. For those orientations, M changes but $F^q(\theta)$ does not change.

2.4 Spiral Reads and Sistrings

Each cell of the text defines a string which is obtained by reading text positions at increasing Euclidean 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 in some predefined order, the only important thing is to read the cells in order of increasing distances from the central cell. If such a string hits the border of the text it is considered finished there. We will call *sistrings* (for "semi-infinite strings") [24] the strings obtained in this way. Figure 2 shows a possible reading order.

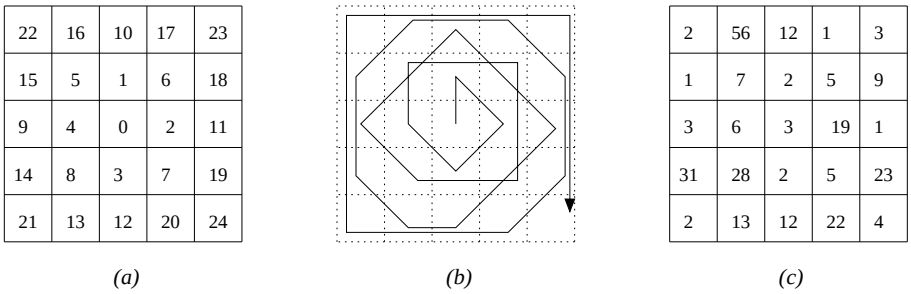


Figure 2: A spiral reading order for the sistring that starts in the middle of a text of size 5×5. Figure *(a)* shows the reading order by enumerating the cells, and figure *(b)* shows the enumeration graphically. Figure *(c)* shows the color values of the cells of the image, so for that image the sistring corresponding to the reading order is $\langle 3, 2, 19, 2, 6, 7, 5, 5, 28, 3, 12, 1, 12, 13, 31, 1, 56, 1, 9, 23, 22, 2, 2, 3, 4\rangle$.

Note that, by reading cells in increasing distance from the center, we are making good use of the $O(m^3)$ bound on the number of angles. Say that we have read ℓ cells in spiral order from pattern and text and want to determine whether there is a match or not. Since we are at distance $O(\sqrt{\ell})$ from the centers, there are only $O(\ell^{3/2})$ relevant rotations to consider, according to Section 2.1.

2.5 Sistring Trie and Tree

Our index data structure is a trie [6], a well-known tree structure for storing strings. Each trie node denotes a prefix of some string. Each tree edge is labeled by a character. The root node represents the empty string, and if node v descends from node u by an edge labeled a, and if u represents prefix s, then v represents prefix sa. Nodes that represent a complete string in the set are marked with a pointer to that string.

It should be clear that it is possible to search for string s in time $O(s)$ over a trie that stores any set of strings of any size. It is just a matter of stepping down from the root node following the characters of s. If at any point there is no proper edge to follow, then s is not in the set. If we traverse the whole s, then the whole subtree of the current node contains the strings with prefix s. In particular, if the current node is marked then we have found string s. At this point it should be clear that a trie can be built in time proportional to its size, by inserting the strings one by one.

To save space, we assume that the strings in the set are stored separately. Hence, as soon as some trie node represents the prefix of a unique string, we remove all the unary path that descends from it and make the node a leaf in the trie, with a pointer to the proper string. The search algorithm must be slightly modified so as to continue comparing s directly against the string once a leaf node has been reached. As a result, internal trie nodes represent prefixes shared by at least two strings, and leaf nodes represent the whole unique strings.

As explained in Section 2.4, each text cell defines a sistring of length $O(n^2)$. A trie built on those strings will be called the *sistring trie*. This time, pointers to strings are replaced by text coordinates, which represent the starting point of the spiral that reads the corresponding sistring.

It is well known that a trie built on n^2 random strings has on average $O(n^2)$ nodes (instead of the worst case $O(n^4)$) and $O(\log_\sigma(n^2))$ depth. In fact this also happens when strings are not independent, under rather general conditions [29, 30]. It is not hard to see that such a trie can be built in $O(n^2 \log_\sigma n)$ time by successive insertions.

Alternatively, the unary paths of such a trie can be compressed so as to form a *sistring tree*, just like when compressing suffix tries to suffix trees [6]. Each edge is labeled now by a string, which is represented in constant space by a text center and a range of values in a spiral read from that center. This will yield the characters of the string represented. Since the resulting tree is at least binary and has $O(n^2)$ leaves, it follows that it has $O(n^2)$ nodes overall, in the worst case. For simplicity, we describe our algorithms on a sistring trie, although they run with the same complexity over sistring trees.

We finish with folklore property of the sistring trie that is important for some analyses in this paper. We show that, under a uniform model, the number of sistring trie nodes at depth ℓ is on average $\Theta(\min(\sigma^\ell, n^2))$. Roughly, this is to say that in levels $\ell \leq h$, for $h = \log_\sigma(n^2) = 2\log_\sigma n$, all the different strings of length ℓ exist, while from that level on the $\Theta(n^2)$ sistrings are already different. In particular, this means that nodes deeper than h have $O(1)$ children because there exists only one sistring in the text with that prefix of length h (note that a sistring prefix is graphically seen as a spiral inside the text, around the corresponding text cell).

To prove this property we consider that there are n^2 sistrings uniformly distributed across σ^ℓ different prefixes of length ℓ, for any ℓ. The probability of a given prefix not being "hit" after n^2 "attempts" (sistrings) is $(1 - 1/\sigma^\ell)^{n^2}$, so the average number of different prefixes hit (i.e., existing sistring trie nodes) is

$$\sigma^\ell(1 - (1 - 1/\sigma^\ell)^{n^2}) \quad = \quad \sigma^\ell(1 - e^{-\Theta(n^2/\sigma^\ell)}) \quad = \quad \sigma^\ell(1 - e^{-x})$$

for $x = \Theta(n^2/\sigma^\ell)$. Now, if $n^2 = o(\sigma^\ell)$ then $x = o(1)$ and $1 - e^{-x} = 1 - (1 - x + O(x^2)) = \Theta(x) = \Theta(n^2/\sigma^\ell)$, which gives the result $\Theta(n^2)$. On the other hand, if $n^2 = \Omega(\sigma^\ell)$ then $x = \Omega(1)$ and the result is $\Theta(\sigma^\ell)$. Hence the number of sistring trie nodes at depth ℓ is on average $\Theta(\min(\sigma^\ell, n^2))$, which is the same as in the worst case. Indeed, in the worst case the constant is 1, that is, the number of different strings is at most $\min(\sigma^\ell, n)$, while on average the constant is smaller.

2.6 Optimal Multipattern Exact String Matching

In [11] a one-dimensional multipattern exact search algorithm using a *suffix automaton* was proposed. A suffix automaton built on a set of strings is the smallest deterministic automaton able of recognizing any suffix of any string in the set. The automaton is not complete, that is, edges that cannot lead to acceptance are not present. When traversing it, we will have an edge to follow as long as we have read a substring of some of the patterns.

The algorithm works basically as follows. Assume all the patterns are of the same length m. We build the suffix automaton over the set of *reversed* patterns (that is, read backwards). Then we slide a window of length m over the text. Each window is read from right to left, and the characters are input to the suffix automaton. This process can finish in two possible ways. First, it may happen that we have no edge to follow in the automaton. This means that the suffix of the window read is not a substring of any pattern in the set, so no window covering the suffix read can contain a match. In this case we can safely shift the window so it starts right after the last character read. Second, we can reach the beginning of the window, which means that we have read a string of length m which is a substring of some pattern of length m, that is, we have recognized a pattern in the set and we report it. We shift the window by one position.

The above scheme is a bit simplified. In fact the algorithm uses the suffix automaton also to remember the last position in the suffix read where the automaton recognized a suffix of some reversed pattern. Suffixes of reverse patterns are reverse prefixes of patterns. It is not hard to see that, no matter how the processing of the current window finished, we can shift it so that it is aligned to the last prefix recognized.

We show now that this algorithm takes average time $O(n \log_\sigma(rm)/m)$ to search a text of length n for r patterns of length m. Let us say that we have read ℓ characters from the suffix of a text window with the suffix automaton. Let us call s the string just read. There are r different patterns where s could match. Inside each of these, there are $m - \ell + 1$ positions where s could match. Hence the probability of s matching any substring of any search pattern is at most rm/σ^ℓ. This is smaller than $1/m^2$ provided $\ell \geq \log_\sigma(rm^3)$, so for now on let this be the value of ℓ. Let us pessimistically assume that ℓ characters are always read, that if at that point s matches any substring of any pattern (with probability $1/m^2$) then we read the whole window (at cost $O(m)$) and shift the window by 1 position, and that otherwise we shift the window by $m - \ell$ positions.

On a text of length n the number of window verifications is on average $O(n/m^2)$, and each costs $O(m)$, so this part adds a negligible $O(n/m)$ cost. The scanning time dominates. The average number of windows processed is $O(n/(m - \ell))$ and each costs $O(\ell)$ character inspections, yielding an overall average search time of $O(n \log_\sigma(rm^3)/(m - \log_\sigma(rm^3)))$, which is $O(n \log_\sigma(rm)/m)$. We show that this complexity is indeed optimal in [13].

An additional improvement in [11] is to combine the backward scanning with a Aho-Corasick-like forward scanning [1], so as to ensure linear worst case time for the algorithm.

The space requirement of the suffix automaton is $O(rm)$, one state per each suffix of each pattern.

2.7 Optimal Multipattern Approximate String Matching

In [13], two algorithms for one-dimensional multiple approximate string matching were presented. They permit searching for r patterns of length m allowing up to k insertions, deletions and substitutions in a text of length n. The first has optimal average search time $O(n(k+\log_\sigma(rm))/m)$ for $k/m < 1/3 - O(1/\sqrt{\sigma})$. The second has average search time $O(n)$ for $k/m < 1/2 - O(1/\sqrt{\sigma})$.

The algorithms precompute, for every possible string of length ℓ, the minimum number of character insertions, deletions and substitutions needed to match them somewhere inside some pattern in the set, that is, the minimum Levenshtein distance against any substring of any pattern.

Hence, an optimal algorithm works as follows. It divides the text into blocks of length $(m-k)/2$, so that every possible match must contain a complete block. Then, each block is processed as follows: successive non-overlapping ℓ-grams (that is, substrings of length ℓ) are read from the block, and their precomputed minimum distances are accumulated. If, before reaching the end of the block, the accumulated differences exceed k, then the block can be safely abandoned since there is no way to match those ℓ-grams inside any particular pattern with few enough differences. Otherwise, we reach the end of the block without totalizing $k+1$ differences and must verify the block with a classical algorithm.

For the purposes of this paper, the analysis can be rephrased as follows. Let $p(\ell, c)$ be the probability of two random strings of length ℓ matching under a given matching model (Levenshtein in [13]) with at most $c\ell$ differences. Pessimistically, assume that we will verify the block as soon as we find one ℓ-gram whose precomputed minimum distance is less than $c\ell$, for some constant c to fix later. Otherwise we will abandon the block after testing $1 + \lceil k/(\ell c)\rceil$ ℓ-grams. For this to be correct it must hold $\ell\ k/(\ell c) \le (m-k)/2$, or $2k/(m-k) < c$, otherwise we process all the blocks anyway without any sublinearity. Processing each ℓ-gram takes time $O(\ell)$. The probability of verifying a block is $O(p(\ell,c)mrk/(\ell c))$ because each of the $k/(\ell c)$ ℓ-grams can match any pattern at any position, and the cost of verification is $O(m^2 r)$. Hence the search cost has two components: $O(n(\ell + k/c)/(m-k))$ to scan the ℓ-grams of each window, and $O(nm^2 r(p(\ell,c)mrk/(\ell c))/(m-k))$ for the verifications.

In order to make verifications negligible compared to scanning, it is sufficient that $p(\ell,c) = O(1/(m^3 r^2))$. Under the Levenshtein model, it holds $p(\ell,c) = \gamma^\ell/\ell$ where $\gamma = 1/(\sigma^{1-c}c^{2c}(1-c)^{2(1-c)})$. Hence it is necessary that $c < 1 - e/\sqrt{\sigma}$ to make $\gamma < 1$, as otherwise verification cost is very high. For such c, choosing $\ell = \log_{1/\gamma}(m^3 r^2) = \Theta(\log_\sigma(mr)/(1-c))$ gives an overall search cost of $O(n(\log_\sigma(mr)/(1-c) + k/c)/(m-k))$, which must be optimized for c in the range $2k/(m-k) < c < 1 - e/\sqrt{\sigma}$. For the range of c values to be nonempty we need $k/m < (1 - e/\sqrt{\sigma})/(3 - e/\sqrt{\sigma}) = 1/3 - O(1/\sqrt{\sigma})$. In fact, any constant c in the range will yield the optimal complexity, and it will be within bounds as long as k/m is bounded away from $1/3$. Hence the algorithm is $O(n(k+\log_\sigma(rm))/m)$ for $k/m < 1/3 - O(1/\sqrt{\sigma})$.

A second algorithm consists of scanning all the ℓ-grams of the text one by one, and keep the accumulated sum over a *sliding* text window of length $m-k$, which will be verified whenever the accumulated sum does not exceed k. The search time is clearly $O(n)$ and the analysis of the permissible k/m values yields $k/m < 1/2 - O(1/\sqrt{\sigma})$, because the window is of length $m-k$ instead of $(m-k)/2$.

The space requirement of the algorithm is $O(m^3 r^2 \sigma^{O(1)}) = O(m^3 r^2)$.

3 Problem Complexity

In this section we prove that the lower bound for the average time of the rotation invariant search problem in d dimensions is $\Omega(dn^d \log_\sigma m/m^d)$ for the exact model, $\Omega(dn^d \log m/m^d)$ for the minmax model, $\Omega(n^d(k + d\log_\sigma m)/m^d)$ for the mismatches model, and $\Omega(n^d(k/\sigma + d\log_\sigma m)/m^d)$ for the accumulated model. Worst case complexity is known to be $O(m^3n^2)$ for all these models, in two dimensions [3].

For the average case results of this proof and the rest of the paper, we state our probabilistic model. We assumed that the cells of pattern and text are independent random variables chosen uniformly from an alphabet of size σ. Several of our results can be extended to nonuniform probabilities, as long as we replace σ by $1/p$, where $p < 1$ is the probability that two random cells chosen from the pattern and the text match.

The exact number of relevant rotations in d dimensions is unknown. Particular cases are known for $d = 2$ ($O(m^3)$ [17]) and $d = 3$ ($O(m^{11})$ [20]). However, it seems clear that it is $\Omega(m^{d-1})$ because each cell of P can be forced, by rotations, to match any of the $O(m^{d-1})$ text cells that are at the same distance to the center. On the other hand, there cannot be more than $m^{O(d^3)}$ relevant rotations. The reason is that there are $\binom{d}{2} = O(d^2)$ rotation planes. For each such plane we can assume that, once all the other rotations at other planes are fixed, each cell of P can be made to match $O(m)$ different cells of P, for a total of $O(m^{d+1})$ rotations regarding that rotation plane. We finally can assume that each choice of rotation for each rotation plane produces a different combination, so we have overall $O((m^{d+1})^{d^2}) = O(m^{d^3})$ possibilities. Therefore, although we know very little about the number of rotations in d dimensions, we have that they are $\Theta(m^{\rho(d)})$, where $\rho(d) = \text{poly}(m)$ (in particular, $d - 1 \leq \rho(d) \leq (d - 1)d(d + 1)/2$. This will be enough for us.

3.1 Exact and MinMax Models

Under our probabilistic model, there exists a general lower bound for d-dimensional exact pattern matching. In [34] Yao showed that the one-dimensional string matching problem requires at least $\Omega(n \log_\sigma m/m)$ comparisons on average, where n and m are the lengths of the text string and the pattern respectively. In [25] this result was generalized for the d-dimensional case, for which the lower bound (without rotations) is $\Omega(n^d \log_\sigma(m^d)/m^d) = \Omega(dn^d \log_\sigma(m)/m^d)$.

The above lower bound also holds for the case with rotations allowed, as exact pattern matching reduces (as a special case) to matching with rotations. To search for the pattern exactly, we first search for it allowing rotations, and once we find an occurrence we verify whether or not the rotation angle is zero. Since in 2D there are $O(m^3)$ rotations [17], on average there are $O(n^2m^3/\sigma^{m^2})$ occurrences. Each rotated occurrence can be verified in $O(m^2)$ time (indeed, $O(1)$ average time using the results of the present paper, but this is not relevant). Hence the total exact search time (et) is that of searching with rotations (rt) plus $O(n^2m^5/\sigma^{m^2}) = o(n^2 \log_\sigma(m)/m^2)$ for verifications. Because of the bound in [25], $et = \Omega(n^2 \log_\sigma(m)/m^2) = rt + o(n^2 \log_\sigma(m)/m^2)$, and so $rt = \Omega(n^2 \log_\sigma(m)/m^2)$ as well. This argument can be easily generalized to d dimensions because there are $O(n^d m^{\rho(d)}/\sigma^{m^d})$ matches to verify at $O(m^d)$ cost each, and hence $n^d m^{d+\rho(d)}/\sigma^{m^d} = o(n^d \log_\sigma(m^d)/m^d)$. Hence we get a general bound of $\Omega(n^d \log_\sigma(m^d)/m^d) = \Omega(dn^d \log_\sigma(m)/m^d)$ comparisons.

This result is easily generalizable to the minmax model if we consider that each text cell matches a range of values. The size of the range is the average difference between the maximum and

minimum of 9 values (that is, its neighboring cells) uniformly distributed over σ. It is easy to see that the maximum over t uniformly distributed discrete random variables in the interval $1 \ldots \sigma$ is on average $\leq \sigma t/(t+1)$, and the minimum is $\geq \sigma/(t+1)$, hence the average difference is bounded above by $\sigma(t-1)/(t+1)$. Since in our case $t = 9$, we have that the average size of our range is $(4/5)\sigma$ (this pessimistic bound is tight, as the relative error is $O(1/\sigma)$). Hence the matching probability is $4/5$ instead of $1/\sigma$ and $\log_\sigma m$ becomes $O(\log m)$, yielding a lower bound of $\Omega(n^2 \log(m)/m^2)$. In d dimensions $t = 3^d$, so the $4/5$ becomes $(3^d-1)/(3^d+1)$ and $\log_{5/4} m$ becomes $\log(m)/(\log(3^d+1) - \log(3^d-1)) = \log(m)/\Theta(1/3^d)$. Therefore the lower bound for the minmax model in d dimensions is $\Omega(d3^d n^d \log(m)/m^d)$.

3.2 Mismatches and Accumulated Models

A lower bound for the k differences problem (approximate string matching with $\leq k$ mismatches, insertions or deletions of characters) was given in [10] for the one dimensional case. This bound is $\Omega(n(k + \log_\sigma m)/m)$, where n is the length of the text string and m is the length of the pattern. This bound is tight; an algorithm achieving it was also given in [10].

This lower bound can be generalized to the d-dimensional case also. By [25], exact d-dimensional searching needs $\Omega(n^d \log_\sigma(m^d)/m^d)$ comparisons, and this is a special case of approximate matching. Following [10], we divide the text into n^d/m^d disjoint "windows", that is, hypercubes of size m^d. Hence, at least $k+1$ symbols of each window have to be examined to guarantee that the window cannot match the pattern. So a second lower bound is $\Omega(kn^d/m^d)$. The lower bound $\Omega(n^d(k + \log_\sigma m^d)/m^d) = \Omega(n^d(k + d\log_\sigma m)/m^d)$ follows, without considering rotations.

We can apply the same reduction as before to show that the bound is also valid when rotations are permitted. The only delicate part of the reduction is to ensure that $n^d m^{d+\rho(d)} p = o(n^2 \log_\sigma(m^d)/m^d)$, where p is the probability that the pattern matches some text position at some given rotation. According to Appendix A, $p \leq \binom{m^d}{k}/\sigma^{m^d-k}$. Substituting we get that our bound is valid for $k/m^d < 1/(1 + d\log_\sigma m)(1 + o(1))$. However, for larger k the lower bound is also valid, simply because the number of occurrences to report by a rotation invariant search is on average $n^d m^{\rho(d)} p = \Omega(n^2 \log_\sigma(m^d)/m^d)$, and we cannot work less than the size of the output.

To generalize this result for the accumulated model, we have to take into account that $(k+1)/\sigma$ cell inspections may be sufficient to discard a window, so the lower bound becomes just $\Omega(n^d(k/\sigma + d\log_\sigma m)/m^d)$. Considering the results of this paper, however, we conjecture that this bound is not tight, and that it could be improved to $\Omega(n^d(k/\sigma + d\log m)/m^d)$. If we consider that the alphabet size is a constant, however, both bounds are indistinguishable.

3.3 Worst Case Complexity

With respect to worst case complexity, it has been shown in [3] to be $O(m^3 n^2)$, for the exact and any other of our models. They give an optimal worst-case algorithm for exact searching. In this paper we show that this can be obtained simultaneously with average-case optimality.

In d dimensions, this worst-case lower bound generalizes to $O(m^{\rho(d)} n^d)$, although unfortunately we know little about the exact value of $\rho(d)$, except that it is between $d-1$ and $(d-1)d(d+1)/2$.

4 A Worst-Case Optimal Algorithm

In [3] it was shown that for the problem of two dimensional pattern matching allowing rotations the worst case lower bound is $\Omega(n^2m^3)$. They give an algorithm with such complexity. In this section we show a simple algorithm with the same complexity that works for any of our matching models.

In [20], a 3D algorithm is presented for the mismatches model which involves an idea that can be exploited much further. We reuse that idea and convert it into an optimal worst-case algorithm.

Assume that, when computing the set of angles $\mathcal{A} = (\beta_1, \beta_2, \ldots)$ to match the pattern, we also sort those angles so that $\beta_i < \beta_{i+1}$, and associate with each angle β_i the set $\mathcal{C}_i$ containing the corresponding text cell centers that must hit a pattern cell boundary at β_i. Hence we can evaluate the number of mismatches for successive rotations of P incrementally. That is, assume that the number of mismatches has been evaluated for β_i, then to evaluate the number of mismatches for rotation β_{i+1}, it suffices to re-evaluate the cells restricted to the set $\mathcal{C}_{i+1}$. This is repeated for each $\beta \in \mathcal{A}$. Therefore, the total time for evaluating the number of mismatches for P centered at some position in T, for all possible angles, is $O(\sum_i |\mathcal{C}_i|)$. This is $O(m^3)$ because each fixed cell center of T, covered by P, can belong to $O(m)$ different $\mathcal{C}_i$ sets. To see this, note that when P is rotated the whole 360 degrees, any cell center of T is touched by $O(m)$ cell boundaries of P (see Section 2.1).

Therefore, it suffices to apply the above algorithm for each of the n^2 possible text positions in order to solve the exact, minmax[2], and mismatches problem in $O(n^2m^3)$ worst case time. The accumulated problem is easily solved in the same time by (re)computing absolute differences instead of number of mismatches.

In fact, the algorithm is very flexible, and can be adapted to many other matching models as well. Its space requirement is $O(m^3)$.

All the algorithms that follow from now on are filters that discard most of the text positions and orientations, and check a small subset of the candidates. Although we will use a different verification algorithm that is faster on average, their same average complexity can be achieved if we replace their verification algorithm with the one we have just described. Hence, without affecting their good average complexity (optimal in most cases), we have that all these algorithms will be also optimal in the worst case, since the worst that can happen is that they have to verify all the text positions and orientations, just as we have assumed here. We only have to make sure that we do not verify the same text position more than once.

5 The Exact Model

In [17] only a set of features of length m crossing the center of P is extracted from P, that is, $q = \frac{m+1}{2}$ and $u = m$ (see Section 2.3). The text is then scanned row-wise for the occurrence of some feature, and upon such an occurrence the whole pattern is checked at the appropriate angles.

According to Section 2.1, they have to check $O(m)$ angles per occurrence, and since each verification takes $O(1)$ time on average, their average verification cost is $O(m)$. Overall, their algorithm is $O(n^2)$, dominated by multipattern text search for the features using an Aho-Corasick machine [1].

[2] In this case recomputing a cell means considering a new 9-cell set. This is constant but depends on the dimension. A truly constant-time solution is to precompute, for each cell of P, the range of values permitted given its neighbors.

In [19] the possibility of using features of length $u \leq m$ is considered, since it reduces the space usage and number of rotations.

We show now how to improve both search and verification time. In which follows we assume that the features are of length $u \leq m$, and later find the optimal u.

5.1 Faster Search

In [8] a 2-dimensional search algorithm (not allowing rotations) is proposed which works by searching for all the pattern rows in the image. Only every m-th row of the image needs to be considered because one of them must contain *some* pattern row in any occurrence.

We take a similar approach. Instead of taking the $O(u^2)$ features that cross the center of the pattern, we also take some not crossing the center. More specifically, we take features for q in the range $\frac{m-r}{2}+1 \ldots \frac{m+r}{2}$, where r is an odd integer for simplicity. For each such q, we read the features at all the relevant rotations. This is illustrated in Figure 1.

This allows us to search only one out of r image rows, but there are $O(ru\max(r,u))$ features now (since the farthest feature cell is at distance $\max(r,u)$ from the pattern center). Figure 1 also shows that the features may become shorter than m when they are far away from the center and the pattern is rotated. On the other hand, there is no point in taking features farther away than $m/2$ from the center, since in the case of unrotated patterns this is the farthest we can go. Therefore we have the limit $r \leq m$. If we take features from $r = m$ rows then the shortest ones (for the pattern rotated at 45 degrees) are of length $(\sqrt{2}-1)m = \Theta(m)$. Note that some features do not cross the pattern center, but they are still fixed if the pattern center matches a text center.

The search time per character is independent on the number of features if an Aho-Corasick machine [1] is used. Alternatively, we can use a suffix automaton (DAWG-MATCH algorithm) [11] to get optimal average search time. The worst case time for the suffix automaton is the same as for the Aho-Corasick automaton.

5.2 Faster Verification

We show how verifications can be performed faster, in $O(1)$ average time instead of $O(m)$. Note that, between to consecutive angles of this $O(m)$-size set, there are many text cells that actually match the same pattern cell, so by checking each rotation separately we are unnecessarily recomparing a lot of cells. Our idea is to recompare cells only when necessary.

Imagine that a feature taken at angle θ has been found in the text. Since the feature has length u and its farthest cell can be at distance at least u from the center, there at most $O(m^3/u^2)$ different angles to check, whose limits we call γ_1 to γ_K, and we have $\gamma_i \leq \theta < \gamma_{i+1}$.

We will show first an overkill technique that is $O(1)$, and then will turn into practical efficiency considerations. Let us assume that we want to check for an occurrence centered at a given text position, for *any* possible orientation.

The idea is that, instead of checking each orientation separately, we interleave checking and orientation refining. We perform a spiral read of the text around the position of interest, as in Section 2.4. If the text center does not match the pattern center, we finish immediately. Otherwise, at any moment, we have a set of ranges of orientations in which the pattern has matched the text

up to now. In the beginning, after we see that the centers match, our range set contains a single range of 360 degrees.

Each time we read a new text cell, we consider all the active orientation ranges. For each such range, the text cell center may be covered by one or more different pattern cells. If there is more than one choice of pattern cells, then we split the orientation range under consideration into several smaller ranges and replace them in the set. After this operation, each range gives us a single choice of which pattern cell should match the current text cell. Now, we remove from the active ranges all those where the text cell does not match the pattern cell. We repeat the process until either there are no more active ranges or we have checked all the pattern cells for a given range. In the latter case we report the occurrence of P.

Let us analyze this algorithm. Say that we are reading the ℓ-th text cell in a spiral read. This means that we are at distance $O(\sqrt{\ell})$ from the center, and therefore there are $O(\ell^{3/2})$ relevant orientations up to now (Section 2.1). For each such orientation, we will read the new text cell only if all the previous $\ell - 1$ cells have matched in this fixed orientation, and if the text and pattern centers have matched initially. Hence the total cost of the algorithm is the sum of the text cells read for each orientation for every ℓ, which is

$$\sum_{\ell=1}^{O(m^2)} O(\ell^{3/2})/\sigma^{\ell} \quad = \quad O(1)$$

This shows that even the least orientation-restricted verification can be done in constant average time if we cleverly interleave checking and orientation restriction. Note that the algorithm can use backtracking instead of actually keeping a set of active orientations.

It is interesting that, using this smart verification technique, we would obtain $O(n^2)$ average search time, just by checking every text cell.

If we use the technique for verification of matching features, then the initial angle range will be much smaller than the 360 degrees we start with. Indeed, all the pattern cells closer to the center that the farthest feature cell are checked with a single orientation, and the incremental process described above is started only in order to check cells that are farther away.

Note that this result holds even if the cell values are not uniformly distributed in the range $1 \ldots \sigma$. It is enough that there is an independent probability $p < 1$ of match between a random pattern cell and a random text cell, in which case σ is replaced by $1/p$.

5.3 Analysis

The average search time of the suffix automaton for s features of length u is $O(n^2 \log_\sigma(su)/(r(u - \log_\sigma(su))))$: There are su feature suffixes to match, so we examine $O(\log_\sigma(su))$ characters on average before abandoning each text window, and shift the window by $O(u - \log_\sigma(su))$ positions. We are searching for $s = O(ru \max(r, u))$ features (counting all their rotations). Finally, we scan only every r-th row of the text.

The verification time per feature that matches is $O(1)$ as explained, and there are $O(ru \max(r, u)/\sigma^u)$ features matching each text position on average. This results in a total search cost of

$$O\left(\frac{n^2}{r}\left(\frac{\log_\sigma(ru^2 \max(r, u))}{u - \log_\sigma(ru^2 \max(r, u))} + \frac{ru \max(r, u)}{\sigma^u}\right)\right)$$

The optimum is at $r = u = \Theta(m)$, which leads to total average time

$$O(n^2(\log_\sigma(m)/m^2 + m^2/\sigma^m)) = O(n^2 \log_\sigma(m)/m^2)$$

which is average-optimal. So the exact matching problem can be solved in optimal average time $O(n^2 \log_\sigma(m)/m^2)$. The space requirement of the suffix automaton is $O(m^4)$.

Again, this analysis is valid for non-uniformly distributed cell values, by replacing $1/\sigma$ by p, the probability of a match between random pattern and text cells.

If we want to make this algorithm worst-case optimal at the same time, we must resort to the $O(m^3)$ verification technique of Section 4 instead of our constant-time verification. In this case the average search cost with $r = u = \Theta(m)$ stays optimal:

$$O(n^2(\log_\sigma(m)/m^2 + m^3\ m^2/\sigma^m)) = O(n^2 \log_\sigma(m)/m^2)$$

5.4 Indexed Searching

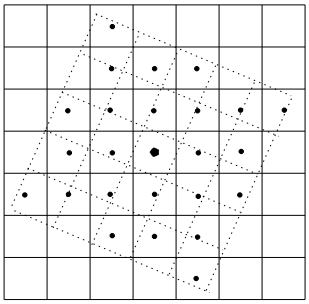
Let us now consider that we have built a sistring trie over T (Section 2.5). A simple search approach is to consider all the $O(m^3)$ pattern orientations in turn and search each one in the sistring trie. To check the pattern at a given orientation we have to see in which order the pattern cells have to be read so that they match the reading order of the sistring trie construction.

Figure 3 shows the reading order induced in the pattern by a rotated occurrence, using the spiral reading order given in Figure 2. 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 sistring trie. Note in particular that some pattern cells may be read twice and others may not be considered at all. Observe that in our example the cells numbered 30, 32, 34, and 36 are outside the maximum circle contained in the pattern, and are therefore ignored in the sistring trie search. This is because those values cannot be used unless some trie levels are skipped in the search, which would mean entering into all the branches after reading cell number 20. Rather, we prefer to check the surviving candidates directly in the text. Finally, text cells 21-29, 31, 33, 35, and 37- all fall outside the pattern.

As in Section 5.2, we can do better. We do not need to consider all the $O(m^3)$ orientations from the beginning, because of two reasons: (1) when we are not reading far away from the pattern center, many of those rotations are indistinguishable; (2) we may have found no string in the text equal to the pattern long before reading all the pattern cells.

The main idea is to use the sistring trie to verify all the text positions simultaneously. Just as in Section 5.2, we start by considering the pattern center and a single orientation range of 360 degrees. We refine the relevant angles as we get farther away from the center, so that there is always a single pattern cell to match. However, instead of directly comparing a particular text cell against our pattern cell, we try to descend in the trie using the current pattern cell value. In this way, we search all the possible text positions at the same time.

Our algorithm reads deeper and deeper cells in the sistring trie, and considers finer rotations (hence entering more than one trie branch at times) as it gets farther away from the pattern center. When the pattern cells are exhausted (that is, we have reached the end of the pattern in our spiral read), the rest of the pattern cells are directly verified in the text individually, for each text position that has survived up to now, using the algorithm of Section 5.2.



(a)

34	10	17	18	*36*
16	5	1	6	11
15	4	0	2	19
9	8	3	7	20
32	14	13	12	*30*

(b)

2	19	7	1	3
1	7	2	5	9
3	6	3	4	1
9	12	2	5	3
2	7	6	5	4

(c)

Figure 3: Reading order induced in the pattern by a rotated occurrence. *(a)* shows the pattern superimposed on the image, *(b)* shows the enumeration of the induced reading order, and *(c)* shows the color values for the pattern cells. The corresponding string is $\langle 3, 2, 4, 2, 6, 7, 5, 5, 12, 9, 19, 9, 5, 6, 7, 3, 1, 7, 1, 1, 3\rangle$. Cells numbered 30, 32, 34, and 36 are ignored in the trie search.

Let us analyze this algorithm. The number of rotations grows as we get farther from the center, and they are tried only on the existing branches of the sistring trie. That is, when we reach depth ℓ in the sistring trie we are considering a pattern cell which is at distance $O(\sqrt{\ell})$ from the center, and hence we need to consider $O(\ell^{3/2})$ different rotations (Section 2.1).

The fact that on average every different string up to length $h = O(\log_\sigma n^2)$ exists in the sistring trie (Section 2.5) means that we always enter until depth h. The number of sistring trie nodes considered up to depth h is thus

$$\sum_{\ell=1}^{h} \ell^{3/2} \quad = \quad O(h^{5/2})$$

At this point we have $O(h^{3/2})$ candidates that are searched deeper in the sistring trie. Now, each such candidate corresponds to a node of the sistring trie at depth h, which has $O(1)$ children because there exist $O(1)$ text sistrings that share this prefix with the pattern (Section 2.5, a "prefix" here means a spiral around the pattern center).

A simple way to see this process is to consider that following a unary path is equivalent to directly checking in the text each candidate that arrived at depth h, instead of using the sistring trie. Indeed, this is what actually happens because the trie does not store the final unary path, but stores a pointer to the text coordinate as soon as the prefix becomes unique. Hence, we may resort to direct text verification long before reaching a border of the pattern. There are $O(h^{3/2})$ candidates and each one can be checked in $O(1)$ time using the result of Section 5.2. Hence this part of the search is dominated by the $O(h^{5/2})$ time spent in the top part of the trie.

Therefore we have a total average search cost of $O((\log_\sigma n)^{5/2})$. This assumes that the pattern is large enough, that is, that we can read h characters from the pattern center in spiral form without hitting the border of the pattern. This is equivalent to the condition

$$m^2 \geq \frac{4}{\pi} \log_\sigma n^2$$

which is a precondition for our analysis. Smaller patterns leave us without the help of the sistring

trie long before we have eliminated enough candidates, so their text verification is expensive ($O(m^3n^2/\sigma^{m^2\pi/4})$, that is, on average $O(\sigma^{m^2(1-\pi/4)})$ time per result delivered). Note, incidentally, that this would be optimal if we searched for circular instead of square patterns, because even if the pattern were very small, once the pattern cells were all used we would have all the results in the subtrees of our current trie node. Hence we would work $O(1)$ time per delivered result, which is optimal.

6 The MinMax Model

In the MinMax model a match requires that the color of each text cell must be between the minimum and maximum pattern colors in the 9-cell neighborhood of the corresponding pattern cell. As it was shown in Section 3.1, the probability of matching between a random pattern and text cell in this model is $p = 4/5$. In practice p is much smaller than $4/5$, as images do not usually have many color changes between neighboring cells [19].

6.1 Sequential Searching

In principle, we can apply our exact search algorithm as is, just taking care of precomputing the range of color values that match each pattern cell in $O(m^2)$ time before the search.

The simultaneous search for multiple pattern features, however, is rather different now. Each pattern cell has a range of values rather than a single value. Standard string matching algorithms do not work in this scenario. The most similar problem is called "string matching with classes", where each pattern position matches a set of alphabet values. Although there are some solutions for single-pattern searching, no good multipattern search algorithms exist [28].

We opt for expanding the pattern set into the full set of all simple patterns that may result from the given ranges. That is, if a pattern of length 2 has the ranges $\langle [1 \ldots 3], [2 \ldots 3] \rangle$ then we expand it into the 6 patterns $\langle 1,2 \rangle$, $\langle 1,3 \rangle$, $\langle 2,2 \rangle$, $\langle 2,3 \rangle$, $\langle 3,2 \rangle$, $\langle 3,3 \rangle$.

If we call Δ, where $1 \leq \Delta \leq \sigma$, the random variable telling the range of values of each feature cell, then this expansion yields $O(\Delta^u)$ simple features for each feature of length u. Hence, returning to the analysis in Section 5.3, we are now searching for $O(\Delta^u ru \max(r,u))$ simple features of length u. It is not convenient, however, to analyze the search time of a a suffix automaton for this number of patterns, because the set is not a general set of patterns, but a rather particular one.

Let us redo the analysis of Section 2.6 for our particular case. Assume that we have read ℓ characters from the suffix of a text window with the suffix automaton. Let us call s the string just read. There are $ru \max(r,u)$ different patterns where s could match. Inside each of these, there are $u - \ell + 1$ positions where s could match. Finally, for each such position, there are Δ^ℓ possible values for s that match that position. Overall, the probability of s matching any substring of any search pattern is $ru^2 \max(r,u)\Delta^\ell/\sigma^\ell$. This is smaller than $1/u^2$ provided $\ell \geq \log_{\sigma/\Delta} ru^4 \max(r,u)$, so for now on let this be the value of ℓ. Now pessimistically assume that ℓ characters are always read, that if at that point s matches any substring of any pattern (with probability $1/u^3$) then we check the window (at cost $O(u)$ with an Aho-Corasick machine [1]) and shift the window by 1 position, and that otherwise we shift the window by $u - \ell$ positions. On a text of length n the number of window verifications is on average $O(n/u^2)$, and each costs $O(u)$, so this part adds

a negligible $O(n/u)$ cost. The other case dominates, yielding an overall average search time of $O(n \log_{\sigma/\Delta}(ru^4 \max(r,u))/(u - \log_{\sigma/\Delta}(ru^4 \max(r,u)))$.

To continue, consider that the expectation of Δ is $(4/5)\sigma$, and therefore $\log_{\sigma/\Delta} x = \log_{5/4} x$. Substituting, we have that the search cost for all the features is

$$O\left(\frac{n^2}{r}\left(\frac{\log_{5/4}(ru^4 \max(r,u))}{u - \log_{5/4}(ru^4 \max(r,u))}\right)\right)$$

where the optimum is clearly $r = u = \Theta(m)$, so as to obtain $O(n^2 \log(m)/m^2)$ search time.

Verifications still cost $O(1)$ time using the technique of Section 5.2, but their number has raised to $O(n^2 \Delta^u ru \max(r,u)/\sigma^u) = O(n^2 ru \max(r,u)/(5/4)^u)$. This is smaller than the search cost provided we choose $u > 5 \log_{5/4} m$. This is no problem since $u = \Theta(m)$.

A remaining problem is that the size of the automaton is exponential in m, because we store $\Delta^u = \Delta^{O(m)}$ features. A way to circumvent this is as follows. We store every ℓ-sized substring of every pattern. Instead of running the real DAWG-MATCH algorithm, we run something much closer to its pessimistic (but still accurate) analysis: We read at most ℓ characters in the window, and if the string read still appears in some pattern, then we verify the window with Aho-Corasick and shift it by 1 position. As shown with our analysis, this still has the optimal complexity. However, now the space usage is $O(\Delta^\ell ru^2 \max(r,u)) = O(m^{7 \log_{5/4}(\sigma)-3})$, which is polynomial in m.

If we wish to retain the optimal worst-case complexity, then we would pay $O(m^3)$ for verifications. We need to choose $u > 8 \log_{5/4} m$ and spend a bit more space for the automaton (but still polynomial in m).

6.2 Indexed Searching

Indexed searching can be adapted as well. In this case, we do not enter into a single branch of the sistring trie. Rather, for each pattern cell we follow all the branches whose color is between the minimum and maximum neighbor of that pattern cell. The number of branches qualifying for the next pattern cell is again a random variable Δ.

Since there are now $O(\Delta^\ell)$ possible strings that match the pattern prefix of length ℓ, and $O(\ell^{3/2})$ rotations for each, we touch

$$\sum_{\ell=1}^{h} \ell^{3/2} \Delta^\ell \quad = \quad O(h^{3/2} \Delta^h)$$

sistring trie nodes up to depth h, because all those qualifying strings exist up to that depth. From that depth on, we verify the remaining $O(h^{3/2}\Delta^h)$ candidates one by one using the $O(1)$ verification technique. Therefore, the total complexity is

$$O(h^{3/2}\Delta^h) \quad = \quad O\left((\log_\sigma n)^{3/2} n^{2 \log_\sigma \Delta}\right) \quad = \quad O\left((\log_\sigma n)^{3/2} n^{2(1-\log_\sigma 5/4)}\right).$$

where the last step is based on that $\Delta = (4/5)\sigma$ on average.

7 The Mismatches Model

This model is an extension of the exact matching model. An additional parameter k is provided to the search algorithm, such that a mismatch occurs only when more than k characters have not

matched. In this section we use $\alpha = k/m^2$. We require $0 \leq \alpha < 1$, as otherwise the pattern would match everywhere.

7.1 A Robust Algorithm

We first present a 2D version of the incremental algorithm of [20], which runs in $O(k^{3/2}n^2)$ average time to search for a pattern in a text allowing rotations and at most k mismatches. This algorithm works for any k and converges smoothly to the worst case $O(m^3n^2)$ when k tends to m^2.

Recall the verification technique proposed in Section 4. Then consider the k mismatches problem. As proven in Appendix A, the matching probability with k mismatches becomes exponentially decreasing as soon as we compare more than $k_H = \Theta(k)$ characters.

This suggests the following algorithm for the k-mismatches case. Check, for every text position, whether or not it matches the pattern with k mismatches. However, do not use the whole pattern P, but rather the smallest subpattern P' of P, with the same center cell, of size $m' \times m' > k_H$. Then use the verification algorithm of Section 4 at each text position. This has a worst case cost of $O(m'^3n^2)$.

Now, for those text positions where P' has been found, continue the verification until the whole P has been matched or discarded. For this stage, use the verification algorithm of Section 5.2. Since at this point we have examined more than $m'^2 > k_H$ cells, the probability of continuing the verification is exponentially decreasing at any moment, and therefore completing the verification adds only $O(1)$ cost per text cell on average.

Overall, we have an average search time of $O(m'^3n^2)$, which is $O(k^{3/2}n^2)$. Note that this algorithm assumes nothing on how we compare the cell values, so any other distance measure than counting the mismatches can be also used.

We show now how to improve this time complexity. The results we present next completely overshadow our previous results for this model [16].

7.2 Approximate Features

If there are k mismatches or less in an occurrence of P and we choose r features as for exact searching, then at least one of the features must appear in the occurrence with at most $\lfloor k/r \rfloor$ mismatches. Otherwise, each of the r features appears with at least $\lfloor k/r \rfloor + 1$ mismatches, for a total of $r(\lfloor k/r \rfloor + 1) > r(k/r) = k$ mismatches.

This folklore property can be used as follows. Instead of scanning one text row out of r as in Section 5.1, we scan one text row out of $\lfloor r/\ell \rfloor$, for some $1 \leq \ell \leq r$. This guarantees that at least ℓ features of every potential occurrence are scanned. Hence the above argument implies that we can search for all the r features allowing $\lfloor k/\ell \rfloor$ mismatches in their occurrences. When we find such a feature, we check the complete pattern.

It turns out that it is trivial to simplify the one-dimensional multipattern approximate search algorithms presented in Section 2.7 so that they permit only mismatches (that is, Hamming distance). Just the preprocessing has to be changed to precompute Hamming instead of Levenshtein distances.

Let us now adapt the analysis. For the optimal algorithm, the windows can be of length $m/2$ now, instead of $(m-k)/2$. The reason is that an approximate occurrence has length m (instead

of a minimum length of $m-k$ under Levenshtein model). Hence the first requirement on c is $2k/m < c$. Also, we show in Appendix A that, under the mismatches model, $p(\ell, c) = \gamma^\ell/\sqrt{\ell}$, where $\gamma = 1/(\sigma^{1-c}c^c(1-c)^{1-c})$ and $\gamma < 1$ for $c < 1 - e/\sigma$. Therefore we get again $\ell = \log_{1/\gamma}(m^3r^2) = \Theta(\log_\sigma(m^3r^2)/(1-c))$ and the analysis follows the same. The change only affects the condition of applicability, to $2k/m < c < 1 - e/\sigma$. We can obtain an optimal algorithm by choosing a constant c in the interval, which is not empty as long as $k/m < (1-e/\sigma)/(2-e/\sigma) = 1/2 - e/(4\sigma - 2e) = 1/2 - O(1/\sigma)$. The linear time algorithm follows the same path, the only condition on k/m being now $k/m < 1 - e/\sigma$.

7.2.1 Using an Optimal Algorithm

We search one out of r/ℓ text rows. Our r features are of length u, and the number of relevant rotations for each is $O(u\max(r,u))$. Hence the total number of length-u strings searched for is $O(ru\max(r,u))$. The number of mismatches permitted is k/ℓ. So the search cost is

$$O\left(\frac{n^2}{r/\ell}\frac{(k/\ell + \log_\sigma(ru^2\max(r,u)))}{u}\right) = O\left(\frac{n^2}{ru}\left(k + \ell\log_\sigma(ru^2\max(r,u))\right)\right)$$

It is clear that ℓ should be as small as possible. Yet, we have to fulfill the condition that the number of mismatches divided by pattern length should be away from 1/2. This means $(k/\ell)/u \leq 1/2 - O(1/\sigma)$, and therefore $\ell \geq 2k/u(1+O(1/\sigma))$. Since we want the smallest possible ℓ, let us set $\ell = \max(1, 2k/u(1+O(1/\sigma)))$.

Let us first assume $u > 2k(1+O(1/\sigma))$ and hence $\ell = 1$. The search cost becomes

$$O\left(\frac{n^2}{ru}\left(k + \log_\sigma(ru^2\max(r,u))\right)\right)$$

where it is clear that we prefer the maximum possible $u = r = \Theta(m)$, which yields the optimal search time

$$O\left(\frac{n^2}{m^2}(k + \log_\sigma m)\right)$$

under the condition $u > 2k(1+O(1/\sigma))$. The maximum possible value for u such that $r = \Theta(m)$ and still r features of length u can be extracted from the pattern rotated 45 degrees is $u = m\sqrt{2-\epsilon^2}$ and $r = \epsilon m$. Hence the condition under which we can reach optimal search time is $\alpha < 1/(\sqrt{2}m)(1+O(1/\sigma))$.

Let us now assume $u \leq 2k(1+O(1/\sigma))$ and $\ell = 2k/u(1+O(1/\sigma))$. Hence we have that the search cost is

$$O\left(\frac{n^2}{ru}\left(k + \frac{k\log_\sigma(ru^2\max(r,u))}{u}\right)\right)$$

where again the best choice is to have the largest possible u and r, so we set $r = \Theta(m)$ and $u = \min(2k(1+O(1/\sigma)), \sqrt{2}m)$. First assume $2k(1+O(1/\sigma)) \leq \sqrt{2}m$ and hence $u = 2k(1+O(1/\sigma))$. Then the search cost is

$$O\left(\frac{n^2}{mk}(k + \log_\sigma m)\right)$$

which is valid only for $2k(1+O(1/\sigma)) \leq \sqrt{2}m$, that is, $\alpha \leq 1/(\sqrt{2}m)(1+O(1/\sigma))$. This branch is not interesting, as it is not optimal and is valid in the same k range of the optimal result. Hence assume $\sqrt{2}m \leq 2k(1+O(1/\sigma))$ and then $u = \Theta(m)$. The search cost becomes

$$O\left(\frac{n^2}{m^2}\left(k + \frac{k}{m}\log_\sigma m\right)\right) = O\left(\frac{n^2 k}{m^2}\right)$$

and this is valid for $\alpha \geq 1/(\sqrt{2}m)(1+O(1/\sigma))$. There is another condition of applicability, however, namely $\ell \leq r$. This means $2k/u(1+O(1/\sigma)) \leq r$, that is, $2k(1+O(1/\sigma)) \leq ru$. The maximum ru that works for all rotations (in particular, 45 degrees) is reached for $r = u = m/\sqrt{2}$, and therefore the applicability condition is $\alpha \leq 1/4(1+O(1/\sigma))$.

To summarize, we have optimal search time $O(n^2(k+\log_\sigma m)/m^2)$ for $\alpha < 1/(\sqrt{2}m)(1+O(1/\sigma))$ (choosing $u = m\sqrt{2-\epsilon^2}$, $r = \epsilon m$, $\ell = 1$) and search time $O(n^2k/m^2)$ for $1/(\sqrt{2}m)(1+O(1/\sigma)) \leq \alpha \leq 1/4(1+O(1/\sigma))$ (choosing $u = r = m/\sqrt{2}$, $\ell = 2k/u(1+O(1/\sigma))$).

7.2.2 An Algorithm for Higher Error Levels

The second algorithm in [13] is $O(n)$ time for an error level of at most $1 - O(1/\sigma)$. If we use it for searching we get $O(n^2\ell/r)$ search time. Again, we want to minimize ℓ and therefore $\ell = \max(1, k/u(1+O(1/\sigma)))$. The same options as before appear. If $k/u(1+O(1/\sigma)) < 1$ then $\ell = 1$ and we get a search time of $O(n^2/m)$ that works for $\alpha < \sqrt{2}/m(1+O(1/\sigma))$. If $k/u(1+O(1/\sigma)) \geq 1$ then $\ell = k/u(1+O(1/\sigma))$ and we have $u = \min(k(1+O(1/\sigma)), \sqrt{2}m)$. If $k(1+O(1/\sigma)) \leq \sqrt{2}m$ we get again $O(n^2/m)$ time, so this branch is not interesting. If $k(1+O(1/\sigma)) > \sqrt{2}m$ then we get $O(n^2k/m^2)$ time. The other applicability condition is $\ell \leq r$, which translates into $\alpha \leq 1/2(1+O(1/\sigma))$.

This algorithm gives us two interesting results. First, for the range $1/(\sqrt{2}m)(1+O(1/\sigma)) \leq \alpha \leq \sqrt{2}/m(1+O(1/\sigma))$, we have $O(n^2/m)$ time (choosing $u = m\sqrt{2-\epsilon^2}$, $r = \epsilon m$, $\ell = 1$). Second, for the range $\sqrt{2}/m(1+O(1/\sigma)) \leq \alpha \leq 1/2(1+O(1/\sigma))$, we have $O(n^2k/m^2)$ time (choosing $u = r = m/\sqrt{2}$, $\ell = k/u(1+O(1/\sigma))$). Both are better or equal than the $O(n^2k/m^2)$ result obtained with the optimal algorithm, and hence using the optimal algorithm remains interesting only for $\alpha \leq 1/(\sqrt{2}m)(1+O(1/\sigma))$.

Finally, to simplify the presentation of the results, notice that all the three relevant complexities are indeed the optimal $O(n^2(k+\log_\sigma m)/m^2)$ in the range of k values where they are applicable. So, overall, we have an optimal algorithm for $k/m^2 \leq 1/2(1+O(1/\sigma))$. The space requirement of the feature search algorithm is $O(u^5r^4) = O(m^9)$, polynomial in m.

7.3 Verification

We have not considered up to now how the pattern should be verified once a matching feature has been found. This time verification at $O(1)$ cost is not possible because we must let $k+1$ mismatches occur before abandoning the verification.

The technique described in Section 7.1 is useful here. As we have shown, we can check for all the potential occurrences centered any text cell in $O(k^{3/2})$ average time.

The next concern is how many verifications we perform on average. Using the same reasoning of Appendix A, the probability of a piece of length u matching with k/ℓ mismatches is $O(\gamma^u/\sqrt{u})$,

where $\gamma < 1$ as long as $\beta = (k/\ell)/u < 1 - e/\sigma$. On the other hand, we search for $ru\max(r,u)$ such features, so the total verification cost is

$$O(n^2 k^{3/2} ru \max(r,u)\gamma^u/\sqrt{u}) = O(n^2 k^{3/2} m^{5/2}\gamma^{\Theta(m)})$$

where the equality is based on our previous decisions $r = u = \Theta(m)$. This is negligible compared to the search time as long as $\gamma < 1$, that is, $\beta = k/(u\ell) < 1 - e/\sigma$. In our algorithms we have two cases: (i) $\ell = 1$, $u = \sqrt{2}m$ and $k/m^2 \leq \sqrt{2}/m(1 + O(1/\sigma))$, where $\beta = k/(\sqrt{2}m) \leq 1 - e/\sigma$ implies $k/m \leq \sqrt{2}(1 - e/\sigma)$, which matches our previous conditions; and (ii) $\ell = k/u(1 + O(1/\sigma))$, $u = m/\sqrt{2}$ and $k/m^2 \leq 1/2(1 + O(1/\sigma))$, where $\beta = k/(k(1 + O(1/\sigma))) = 1 + O(1/\sigma) < 1 - e/\sigma$, also matching our previous conditions. Hence verifications do not introduce any new constraints.

In order to achieve $O(m^3 n^2)$ worst case time simultaneously, we should use the algorithm of Section 4 for verifications. This does not change the conditions under which verifications are negligible (that is, $\gamma < 1$), so the average case stays the same.

7.4 Indexed Searching

The idea of traversing all the relevant branches of the sistring trie gives several complications now. Even for a fixed rotation, the number of sistring trie nodes to consider grows exponentially with k. To see this, note that at least k characters have to be read in all the sistrings, which gives a minimum of $\Omega(\sigma^k)$ nodes to process. This means in particular that if $k \geq h$ then we will consider all the n^2 sistrings and the index will be of no use, so we assume $k < h$; still stricter conditions will appear later. We first present a standard technique and then a pattern partitioning technique.

As in Section 6.2, we enter into the trie and at the same time perform the rotations incrementally. This time, however, we do not follow only the branch whose label coincides with the next character of the pattern. Rather, we enter into *all* branches and keep a count of the number of mismatches found up to now. Only when this counter exceeds k can we abandon a branch.

As explained, up to depth k we enter into all the branches of the trie. Since we assume $h > k$, we have to analyze which branches we enter at depths $k < \ell \leq h$. Since all those strings exist in the sistring trie, this is the same as to ask how many different strings of length ℓ match a pattern prefix of length ℓ with at most k mismatches. Resorting again to Appendix A, we have that $\binom{\ell}{k}\sigma^k$ is a tight upper bound. To this we have to add the fact that we are searching for $O(\ell^{3/2})$ different strings at depth ℓ. Hence, the total number of trie nodes touched up to level h is

$$\sum_{\ell=1}^{k} \ell^{3/2}\sigma^\ell \;+\; \sum_{\ell=k+1}^{h} \ell^{3/2}\binom{\ell}{k}\sigma^k \;=\; O\left(h^{3/2}\sigma^k\binom{h}{k}\right)$$

After level h we are left with a number of candidate text positions to verify. There are two cases to distinguish here, according to Appendix A. First, if $h > k_H = k/(1 - e/\sigma)$, then when we start the verification we have already seen enough characters so that the matching probability is exponentially decreasing. Hence we can apply an incremental verification as in Section 5.2 and it will cost $O(1)$ time per verification. Second, if $h \leq k_H$, then it holds that $\binom{h}{k}/\sigma^{h-k} = \Omega(h^{-1/2})$, and therefore just the trie search costs $h^{3/2}\sigma^k\binom{h}{k} = \Omega(hn^2)$, which is not sublinear.

Therefore, the condition for a sublinear search time is $k_H < h < m^2$. This in particular implies that $\alpha < 1 - e/\sigma$. In this case the search cost is

$$O\left(h^{3/2}\sigma^k\binom{h}{k}\right) \quad = \quad O\left((2\log_\sigma n)^{k+3/2}\sigma^k\right)$$

7.5 Partitioning the Pattern into Pieces

The above search time is polylogarithmic in n, but exponential in k. We present now a *pattern partitioning* technique that obtains a cost of the form $O(n^{2\lambda})$ for $\lambda < 1$. The idea is to split the pattern into j^2 pieces (j divisions across each coordinate). If there are at most k mismatches in an occurrence, then at least one of the pieces must have at most $\lfloor k/j^2 \rfloor$ mismatches. Otherwise, each piece would need at least $\lfloor k/j^2 \rfloor + 1$ mismatches, and the total number of mismatches would be $j^2(\lfloor k/j^2 \rfloor + 1) > j^2(k/j^2) = k$.

So the technique is to search for each of the j^2 pieces (of size $(m/j) \times (m/j)$) separately allowing $\lfloor k/j^2 \rfloor$ mismatches, and for each (rotated) match of a piece in the text, go to the text directly and check if the match can be extended to a complete occurrence with k mismatches. Note that the α value for the pieces stays the same as for the whole pattern, k/m^2.

The center-to-center assumption does not hold when searching for the pieces. However, for each possible rotation of the whole pattern with the center-to-center assumption, it is possible to fix some position of the center of each piece inside its text cell. The techniques developed to read the text in rotated form can be easily adapted to introduce such a fixed offset at the center of the matching subpattern.

Let us consider a piece of size $(m/j) \times (m/j)$ which is at distance $\Theta(m)$ from the center of the pattern. The number of relevant rotations for that piece is $O((m/j)^2 m) = O(m^3/j^2)$ (Section 2.1). Since it is problematic to consider a spiral search with incremental rotations for a piece that is not centered, let us devise a more brute-force approach. We search for each of the $O(m^3/j^2)$ rotations of each of the j^2 pieces separately, for a total of $O(m^3)$ independent searches in the trie.

Hence the search cost for this technique becomes $O(m^3)$ times the cost to search for a piece (with a fixed rotation and center offset) in the sistring trie plus the cost to check for a complete occurrence if the piece is found.

If we consider that $(m/j)^2 \leq h$, then all the strings corresponding to pattern pieces exist in the trie. Therefore the cost to traverse the sistring trie for a piece at a fixed rotation is equivalent to the number of strings that can be obtained with k/j^2 mismatches from it. According to Appendix A, this is

$$U = \binom{(m/j)^2}{k/j^2}\sigma^{k/j^2}$$

On average there are $n^2/\sigma^{(m/j)^2}$ text sistrings matching each string in U. Each of these strings has to be checked in the text for a complete occurrence. Since the rotation is fixed, the cost to check an occurrence is $O(k)$, which is the average time needed to find k mismatches when comparing two strings (Appendix A). So the total amount of verification work per piece is $Ukn^2/\sigma^{(m/j)^2}$ and the overall cost is

$$m^3\left(U \;+\; Uk\frac{n^2}{\sigma^{(m/j)^2}}\right)$$

According to Appendix A, $U/\sigma^{(m/j)^2}$ is of the form $\gamma^{(m/j)^2}/(m/j)$, where $\gamma < 1$ if $\alpha < 1 - e/\sigma$. Since in case $\alpha \geq 1 - e/\sigma$ the second term is not sublinear, let us assume $\gamma < 1$. It can also be seen in Appendix A that $\gamma \geq 1/\sigma$. Hence, the first term of the formula, U, is of the form $\gamma^{(m/j)^2}\sigma^{(m/j)^2}/(m/j) = (\gamma\sigma)^{(m/j)^2}/(m/j)$, which decreases with j. The second term of the formula is of the form $kn^2\gamma^{(m/j)^2}/(m/j)$ and increases with j.

The optimum is therefore found when both terms meet, that is, $j = m/\sqrt{2\log_\sigma n}(1 + o(1))$, which incidentally is away by a lower order term from our limiting condition $(m/j)^2 \leq h$. (The fact that it is away only means that the analysis is pessimistic, because in the last levels not all the different sistrings exist in the trie.)

Expanding the value of γ, we can write

$$U \quad = \quad \left(\frac{\sigma^\alpha}{\alpha^\alpha(1-\alpha)^{1-\alpha}}\right)^{(m/j)^2} / \ (m/j) \quad = \quad (kn^2)^{\alpha + H_\sigma^H(\alpha)}/\sqrt{\log_\sigma n}$$

where the equality holds because the optimal j satisfies $(m/j)^2 = \log_\sigma(kn^2)$. For this sake we have defined

$$H_\sigma^H(\alpha) = -\alpha\log_\sigma(\alpha) - (1-\alpha)\log_\sigma(1-\alpha).$$

Hence the overall search time is

$$O\left(\frac{m^3 k}{\sqrt{\log_\sigma n}}\ n^{2(\alpha + H_\sigma^H(\alpha))}\right)$$

This bound is sublinear as long as $\alpha < 1 - e/\sigma$. On the other hand, we can consider to use a larger j, violating the assumed condition $(m/j)^2 \leq h$ so as to reduce the verification time. However, the search time will not be reduced and therefore the time bound will not decrease.

Note that this time there is no need that the trie be deep enough, as we indeed expect that the search for the pieces will stop before reaching the end of the trie. This time the sublinearity is obtained by making the pieces large enough to ensure that they will not appear in the text on average. All we require is $k_H \leq m^2$ and $h \leq m^2$ (so $j \geq 1$).

8 The Accumulated Model

Even more powerful is the accumulated model, which provides a mismatches-like search capability for gray-level images. Here, the sum of the absolute differences between text and pattern colors must not exceed k. In this section we call $\alpha = k/m^2$ as before, but this time $0 \leq \alpha \leq \sigma$.

8.1 Sequential Searching

A first relevant choice is to apply the algorithm of Section 7.1 to this model. According to Appendix B, once we have tried more than $k_A = O(k/\sigma)$ cells, the probability of matching with threshold k becomes exponentially decreasing. With the same method of Section 7.1 we get an $O((k/\sigma)^{3/2}n^2)$ average time algorithm for this model.

We can improve the search by using again the reduction to one-dimensional multipattern approximate searching, as in Section 7.2. Again, it is trivial to adapt the one-dimensional algorithms

described in Section 2.7 so that they use a distance defined as the sum of the absolute differences between pattern and text characters. Just the preprocessing has to be changed to precompute these new distances instead of Levenshtein distances.

The analysis follows the same lines of Section 7.2, although this time $0 < c < \sigma$. As in Section 7.2, windows are of length $m/2$ and m instead of $(m-k)/2$ and $m-k$. The probability $p(\ell, c)$ is obtained in Appendix B; we have $p(\ell, c) = \gamma^\ell/\sqrt{\ell}$, where $\gamma = 2(1+c)^{1+c}/(\sigma c^c)$, which is smaller than 1 provided $c < \sigma/(2e) - 1$. Hence we get again $\ell = \log_{1/\gamma}(m^3 r^2)$, but now we have

$$\begin{aligned} \log(1/\gamma) &= \log\sigma + c\log c - (c+1)\log(c+1) - \log 2 \quad = \quad \log\sigma - \log c - \log 2 + O(1/c) \\ &= \log(\sigma/(2c)) + O(1/c) \end{aligned}$$

and the search cost of the optimal algorithm becomes

$$O\left(\frac{n}{m}\left(\log_{\sigma/(2c)}(rm) + \frac{k}{c}\right)\right)$$

which is not optimal in the range $\log_\sigma mr < k < \sigma\log(rm)$. In fact, it is simpler to assume $c = \Theta(\sigma)$ to obtain

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

which is not optimal in the same range, and only by an $O(\log\sigma)$ factor[3]. The range of applicability of this solution is given by $2k/m < \sigma/(2e) - 1$, that is $k/m < \sigma/(4e) - O(1)$. For the linear-time algorithm we get a limit of applicability of the form $k/m < \sigma/(2e) - O(1)$.

We can repeat step by step the development of Section 7.2, replacing variable k there by $k' = k/\sigma$, to make formulas as similar as possible. The only changes are (1) search time of the sublinear algorithm is $O(n(\log(rm) + k')/m)$, (2) applicability of the sublinear algorithm is $k'/m < 1/(4e)(1 + O(1/\sigma))$, (3) applicability of the linear algorithm is $k'/m < 1/(2e)(1 + O(1/\sigma))$. By following all the details one arrives at $O(n^2(k/\sigma + \log m)/m^2)$ time for $\alpha < \sigma/(4e)$. The space remains $O(m^9)$. Incidentally, the same would be obtained if we applied the technique of coarsening gray levels presented in [14], and combined it with the algorithms for the mismatches model presented in this paper.

As usual, we can use the verification algorithm of Section 4 to achieve $O(m^3 n^2)$ worst case time simultaneously, without major complications.

8.2 Indexed Searching

As in Section 7.4, we have to enter, for each relevant rotation, into all the branches of the sistring trie until we obtain an accumulated difference larger than k. We present first a standard approach and then a pattern partitioning technique.

We enter into all the branches of the sistring trie until we can report a match or the sum of the differences exceeds k. As we show in Appendix B, the number of strings matching a given string

[3] In fact, this can be regarded as optimal if we consider that the alphabet size is a constant. We have not done that in this paper for time complexities.

of length ℓ under this model is at most $2^\ell \binom{k+\ell}{k}$. Since up to length h all them exist, we traverse

$$\sum_{\ell=1}^{h} \ell^{3/2} 2^\ell \binom{k+\ell}{k} \quad = \quad O\left(h^{3/2} 2^h \binom{k+h}{k}\right)$$

nodes in the trie. As in Section 7.4, we have that the candidate-wise verification that follows is $O(1)$ per candidate provided $h > k_A$ (where $k_A = k(\sigma/(2e)-1)$ is the limit defined in Appendix B), and it is not sublinear for $h \leq k_A$.

Hence for $k > k_A$ the total search cost is

$$O\left(h^{3/2} 2^h \binom{k+h}{k}\right) \quad = \quad O\left((k+2\log_\sigma n)^k (\log_\sigma n)^{3/2} n^{2\log_\sigma 2}\right)$$

which is sublinear in n^2 for $\sigma > 2$. On the other hand, $\sigma = 2$ means that the image is bilevel, and in that case the mismatches model is the adequate choice. Hence we obtain sublinear complexity (albeit exponential in k) for $k_A < 2\log_\sigma n$.

8.3 Partitioning the Pattern into Pieces

As in Section 7.5, we can partition the pattern into j^2 subpatterns that are searched exhaustively in the sistring trie. Again considering $(m/j)^2 \leq h$ we have a total search cost of

$$m^3 \left(U \ + \ Uk \frac{n^2}{\sigma^{(m/j)^2}}\right)$$

where this time

$$U = 2^{(m/j)^2} \binom{(m/j)^2 + k/j^2}{k/j^2}$$

according to Appendix B. Again, looking at the detailed formula, we have that the first term decreases and the second term increases as a function of j. The optimum is found when both terms meet, which is again $j = m/\sqrt{2\log_\sigma n}(1+o(1))$, consistent with our condition $(m/j)^2 \leq h$. In fact, the second term is decreasing only for $\alpha < \sigma/(2e) - 1$ (or $k_A < m^2$), otherwise the optimum is $j = 1$, i.e., no pattern partitioning.

For this optimal j, the overall complexity is

$$O\left(\frac{m^3 k}{\sqrt{\log_\sigma n}} \ n^{2(\log_\sigma 2 + H_\sigma^A(\alpha))}\right)$$

where we have defined

$$H_\sigma^A(\alpha) = -\alpha \log_\sigma(\alpha) + (1+\alpha)\log_\sigma(1+\alpha).$$

This complexity is sublinear as long as $\alpha < \sigma/(2e) - 1$. Again, we can consider to use a larger j value but the complexity does not improve.

9 Alternative Matching Models

In this section we consider the implications of changing some of the assumptions we have been working on. We reconsider the decisions of (1) defining matching in terms of text cell centers rather than pattern cell centers; (2) assuming the pattern center has to match a text center; and (3) sticking to the bidimensional case.

9.1 Considering Pattern Centers

We have considered up to now that text centers must match the value of the pattern cells they lie in. This has been done for technical convenience, although an equally reasonable alternative model is that the pattern cells must match the text color where their centers lie in the text. In applications it is most likely that one can choose either way.

With respect to sequential algorithms, the main problem is related to extracting pattern features to search for. There may be more than one pattern center lying at the same text cell, and even no pattern center at all. This means that, when searching for features in the text, it might be that our search pattern has some positions that do not have to be matched against the text, or several positions that have to be matched against the same text character. The resulting string matching problem can hardly be solved in optimal or even linear time, so all the average-optimal complexities would be lost.

Verification, on the other hand, will not be affected by this model change, because each pattern cell read will have to match against some text cell. We will still be able to check each text position in $O(1)$ time for the exact and minmax model, $O(k^{3/2})$ time on the mismatches model and $O((k/\sigma)^{3/2})$ time on the accumulated model. Probably the only reasonable choice to deal with this situation is to use these verification algorithms to check every text position, with a complexity equal to n^2 times the verification cost per cell.

Indexed searching presents similar complications, but some surprising opportunities too. Using pattern centers means that in some branches of the sistring trie we may have more than one condition to meet (which may be incompatible and then the branch can be abandoned under some models) or there may be no condition at all, in which case we have to follow all the branches at that level of the trie.

On average, however, we still have $\Theta(\ell)$ conditions when entering in the sistring trie with a pattern string of length ℓ, and therefore all the time bounds remain the same. The reason behind this is the same that permits retaining the complexity of verification.

We present two cases where it turns out to be possible to develop new algorithms. The first is for the exact matching model, where we can have a larger index that searches faster than with the standard model. The second is for the minmax model.

9.1.1 The Exact Model

In the normal index the text sistrings are indexed once at a fixed rotation (zero). When a given pattern rotation is tried, the pattern is read in rotated form, in an order driven by the text centers. This becomes problematic when using pattern centers, because some text cells must be skipped and others read more than once. However, the dual approach becomes feasible. Imagine that the text

sistrings are read in all the rotated forms. The way to do this is to assume that a rotated pattern is superimposed onto it and to read the text cells where the pattern cells, read in order, fall. This effectively corresponds to the model we are considering in this section, and would be problematic under the text-centers model.

We consider then indexing the rotated versions of the text sistrings, instead of considering the rotated versions of the pattern at search time. Hence, the pattern is just searched with no rotations. 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 size of the sistring trie 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 $O(H)$. After that depth all the surviving positions and rotations are individually considered. Since $H \geq h'$, there are $O(1)$ candidates, which are checked in constant time each. Hence the overall cost is $O(H)$, and the best choice is to take $H = h'$. Therefore h' must satisfy $h' = 2\log_\sigma n + 3/2\log_\sigma h'$, so $h' = x\log_\sigma n$, for any $x > 2$, will do.

This makes the total search time $O(\log_\sigma n)$ on average. The space complexity becomes now $O(n^2(\log_\sigma n)^{3/2})$. The same technique can be used for the mismatches and accumulated model, but in that case the complexity does not improve significantly.

9.1.2 The MinMax Model

In a model where pattern centers are used, it might be more natural to require that the color of each pattern cell is between minimum and maximum colors of the text cells neighboring the one where the center of the pattern cell lies. However, this time the indexed algorithm cannot be used. This is because the condition to enter into a branch of the sistring trie does not depend only on its value but also on its neighbors.

A possible solution to this problem is to generate a new text from the current one, where the value of each cell is replaced by a range of values. This range goes from the minimum to the maximum over its 9 neighbors. After this transformation we can search for the pattern, in principle, as in the exact model, where equality is redefined as that the pattern value must belong to the text range.

However, some problems arise. First, the alphabet of the text is of size $O(\sigma^2)$ to account for all the possible ranges. This changes h to $h = \log_{\sigma^2}(n^2) = \log_\sigma n$. Second, the search has to enter into all the branches whose value (a range) contains the value of the current pattern cell. Let us call Δ the number of ranges matching a random pattern cell. Then, using the same techniques as before, the search cost of this backtracking is

$$O\left((\log_\sigma n)^{3/2} n^{\log_\sigma \Delta}\right) \quad = \quad O\left((\log_\sigma n)^{3/2} n^{2-\log_\sigma(5/4)}\right)$$

where for the last step we have considered that $\Delta \leq (4/5)\sigma^2$ on average. This is only a bit worse than the complexity under the standard model. Some considerations for efficient implementation of this type of trie can be found in [19].

9.2 Center to Center Assumption

It is possible to extend the model by removing the center-to-center assumption [18]. In this case the number of matching functions goes as high as $O(m^7)$ instead of $O(m^3)$. Despite the algorithmic complications, sequential complexities are not affected: The exponent of m is either inside logarithms (so it is translated just into worse constant factors) or is divided by functions exponential in m (and hence any polynomial in m is the same). The only exceptions are the robust algorithms for mismatches and accumulated models, which now become $O(k^{7/2}n^2)$ and $O((k/\sigma)^{7/2}n^2)$, respectively.

Indexing techniques do change their complexity. Since there are $O(\ell^{7/2})$ sistrings to search for at depth ℓ, the search time for the exact model 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 $O(n^2(\log_\sigma n)^{7/2})$.

An interesting technique that can be used as a pre-filter based on the center-to-center assumption, is as follows. Assume that P is at some location $((u,v),\theta)$ on top of T, such that $(u,v) \in T[i,j]$ is not a center-to-center translation, and that the number of mismatches is k for that position of P. Then assume that P is translated to $((i,j),\theta)$, that is, center-to-center becomes true while the rotation angle stays the same. As a consequence, some cell centers of T may have moved to the cell of P that is one of its eight neighbors. Now compute the number of mismatches such that $T[r,s]$ is compared against $M(T[r,s])$ and its eight neighbors as well. If any of those 9 cells match $T[r,s]$, then we count a match, otherwise we count a mismatch. Let the number of mismatches obtained this way be k'.

This means that $k' \leq k$, because all matches that contribute to $m^2 - k$ must be present in $m^2 - k'$ too. Hence this gives a lower bound on the number of mismatches that occur in a match.

Hence we use the algorithm with the center-to-center assumption, but count a mismatch only when the text cells differs from all the 9 pattern cells that surround the one it matches with. The net result in efficiency is that the alphabet size becomes $\sigma' = 1/(1-(1-1/\sigma)^9)$, meaning that cells match with probability $1/\sigma'$.

This is useful both for the exact and mismatches models, and is easy to adapt to the minmax and accumulated models.

9.3 Three and More Dimensions

Rotation invariant template matching in three dimensions has important applications in microbiology, when searching some known substructures (e.g., proteins) from three dimensional models of biological viruses. Matching models in three dimensions have been defined in [20].

The techniques we have developed can be extended to d dimensions in a straightforward manner. As seen in Section 3, in d dimensions, a pattern of size m^d can be rotated in $O(m^{\rho(d)})$ ways, where $d-1 \leq \rho(d) = \Theta(\text{poly}(m)) \leq (d-1)d(d+1)/2$.

For exact sequential searching, for example, we can extract $r^{d-1} = O(m^{d-1})$ features of length $u = O(m)$, and search for all them together in the text. It is enough to select one out of r text rows along each dimension, since this still ensures that a cube of size m^d matching the pattern will touch one of the selected rows [7]. Therefore, we have to traverse n^d/m^{d-1} text cells, with a multipattern

search for r^{d-1} features, rotated in $O(m^{\rho(d)})$ ways each. The search cost is

$$O\left(\frac{n^d}{m^{d-1}}\frac{\log_\sigma\left(m^{d-1}m^{\rho(d)}\right)}{m}\right) = O\left(\frac{\mathrm{poly}(d)n^d\log_\sigma m}{m^d}\right)$$

which, at least for constant d, is optimal (Section 3.1). It is easy to see that all the other optimal complexities can be achieved as well.

Similarly, it is possible to adapt the robust algorithms (Section 7.1) for the mismatches model. After comparing $O(k)$ cells, we are at distance $O(k^{1/d})$ from the center, and therefore we have considered $O(k^{\rho(d)/d})$ rotations. Hence the average search time is $O(k^{\rho(d)/d}n^d)$. For the accumulated model this is $O((k/\sigma)^{\rho(d)/d}n^d)$.

Finally, it is possible to apply the techniques of Section 4 to limit the worst case to $O(m^{\rho(d)}n^d)$, which we conjecture is worst-case optimal.

With respect to indexing, the index needs $O(n^d)$ space and is able of exact searching in $O((d\log_\sigma n)^{1+\rho(d)/d})$ time. For the minmax model there will be 3^d neighboring cells, and the 5/4 becomes $(3^d+1)/(3^d-1)$. Hence the search time will be $O((d\log_\sigma n)^{\rho(d)/d}n^{d(1-\log_\sigma((3^d+1)/(3^d-1)))}) = O((d\log_\sigma n)^{\rho(d)/d}n^{d(1-1/\Theta(3^d\log\sigma))})$. For the Hamming model we get $O((k+d\log_\sigma n)^{k+\rho(d)/d}\sigma^k)$ or $O(m^{\rho(d)}kn^{d(\alpha+H^H_\sigma(\alpha))}/(d\log_\sigma n)^{1/d}$. The formulas for the accumulated model are similar.

10 Conclusions and Future Work

We have addressed the problem of searching for a two-dimensional pattern in a large two-dimensional image (text), so that the pattern can appear with any rotation in the image. The problem has applications in image processing, image databases, geographic information systems, and computational biology, to name a few areas.

We have considered not only the exact matching problem but also several matching models that permit a few differences between the pattern and its occurrences.

We have derived average-case lower bounds for the problem, and designed search algorithms that are optimal on average and in the worst case simultaneously, for all the matching models under consideration. We have also considered text preprocessing techniques, which have resulted in search times which are sublinear in the text size. These indexing techniques can be used to search all the images of a library in one shot.

The techniques we describe here can be implemented, and have been successfully used to solve some computational biology problems [20]. These combinatorial techniques are much faster than those based on the FFT. However, despite that our matching models permit relaxing the matching conditions, even more flexible matching models should be addressed in order to closing the gap towards more ambitious applications in image retrieval.

- In real applications, images may suffer from deformations because of several factors. Some models permit handling displacement errors such as insertions/deletions along rows and columns and in any dimension [7], but these do not address rotations. Combining both would make up a stronger model.

- It would be interesting to generalize the method for scaling invariance also, so that the pattern and the text do not have to be exactly of the same size. Some work on scaling invariance has been carried out [4], but rotations have not been addressed.
- Another problem that should be addressed is due to different lighting conditions. The pattern and the image might have been obtained under different conditions, and hence one image may be brighter than the other. The pixel values in this case may differ considerably, even if the images are taken from the very same object. In this respect, some recent work on transposition invariant string matching, aimed at music retrieval, can be adapted to search under lighting invariance [27].
- One general problem for this kind of pattern matching algorithms is that the pattern template may contain different background than the occurrence of the pattern in the text, or some parts of the pattern or its counterpart in the text may be occluded by some other objects. One possible way to solve this to specify which parts of the pattern are actually relevant and which are background. In pattern partitioning techniques, we could require that only some pieces match.

On the other hand, our basic ideas (approximate feature search, spiral reads) are rather general and can be applied to address several other matching models. For example, our approximate feature search algorithm can be easily adapted to a variant of the accumulated model where the squares of the color differences (rather than their absolute values) are accumulated, with the same time complexity. This model is closer to the traditional cross-correlation approach usually addressed via FFT. The cross-correlation comes from the model $(a-b)^2 = a^2 + b^2 - 2ab$, and the basic traditional cross-correlation uses the term ab (normalized, sometimes), since ab corresponds to the convolutions that can be efficiently computed using FFT. Using our algorithms would yield a much more efficient technique.

References

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

[2] A. Amir, G. Benson, and M. Farach. An alphabet independent approach to two-dimensional pattern matching. *SIAM Journal on Computing*, 23(2):313–323, 1994.

[3] A. Amir, A. Butman, M. Crochemore, G. Landau, and M. Schaps. Two-dimensional pattern matching with rotations. In *Proc. 14th Annual Symposium on Combinatorial Pattern Matching (CPM 2003)*, LNCS, 2003. To appear.

[4] A. Amir and G. Călinescu. Alphabet independent and dictionary scaled matching. In *Proc. 7th Annual Symposium on Combinatorial Pattern Matching (CPM'96)*, LNCS v. 1075, pages 320–334, 1996.

[5] Amihood Amir. Multidimensional pattern matching: A survey. Technical Report GIT-CC-92/29, Georgia Institute of Technology, College of Computing, 1992.

[6] A. Apostolico. The myriad virtues of suffix trees. In A. Apostolico and Z. Galil, editors, *Combinatorial Algorithms on Words*, volume 12 of *NATO Advanced Science Institutes, Series F*, pages 85–96. Springer-Verlag, Berlin, 1985.

[7] R. Baeza-Yates and G. Navarro. New models and algorithms for multidimensional approximate pattern matching. *Journal of Discrete Algorithms*, 1(1):21–49, 2000. Special issue on Matching Patterns.

[8] R. Baeza-Yates and M. Régnier. Fast two-dimensional pattern matching. *Information Processing Letters*, 45(1):51–57, 1993.

[9] L. Brown. A survey of image registration techniques. *ACM Computing Surveys*, 24(4):325–376, 1992.

[10] W. Chang and T. Marr. Approximate string matching with local similarity. In *Proc. 5th Annual Symposium on Combinatorial Pattern Matching (CPM'94)*, LNCS 807, pages 259–273, 1994.

[11] M. Crochemore, A. Czumaj, L. Gąsieniec, T. Lecroq, W. Plandowski, and W. Rytter. Fast practical multi-pattern matching. *Information Processing Letters*, 71(3–4):107–113, 1999.

[12] K. Fredriksson. Rotation invariant histogram filters for similarity and distance measures between digital images. In *Proc. 7th International Symposium on String Processing and Information Retrieval (SPIRE 2000)*, pages 105–115. IEEE CS Press, 2000.

[13] K. Fredriksson and G. Navarro. Average-optimal multiple approximate string matching. In *Proceedings of the 14th Annual Symposium on Combinatorial Pattern Matching (CPM 2003)*, LNCS, 2003. To appear.

[14] K. Fredriksson, G. Navarro, and E. Ukkonen. An index for two dimensional string matching allowing rotations. In *Proc. 1st IFIP International Conference on Theoretical Computer Science (IFIP TCS 2000)*, LNCS 1872, pages 59–75, 2000.

[15] K. Fredriksson, G. Navarro, and E. Ukkonen. *Faster than FFT: Rotation Invariant Combinatorial Template Matching*, volume II, pages 75–112. Transworld Research Network, 2002.

[16] K. Fredriksson, G. Navarro, and E. Ukkonen. Optimal exact and fast approximate two dimensional pattern matching allowing rotations. In *Proc. 13th Annual Symposium on Combinatorial Pattern Matching (CPM 2002)*, LNCS 2373, pages 235–248, 2002.

[17] K. Fredriksson and E. Ukkonen. A rotation invariant filter for two-dimensional string matching. In *Proc. 9th Annual Symposium on Combinatorial Pattern Matching (CPM'98)*, LNCS 1448, pages 118–125, 1998.

[18] K. Fredriksson and E. Ukkonen. Algorithms for 2D Hamming distance under rotations. University of Helsinki. Manuscript, 1999.

[19] K. Fredriksson and E. Ukkonen. Combinatorial methods for approximate image matching under translations and rotations. *Pattern Recognition Letters*, 20(11–13):1249–1258, 1999.

[20] K. Fredriksson and E. Ukkonen. Combinatorial methods for approximate pattern matching under rotations and translations in 3D arrays. In *Proc. 7th International Symposium on String Processing and Information Retrieval (SPIRE 2000)*, pages 96–104. IEEE CS Press, 2000.

[21] Z. Galil and K. Park. Truly alphabet-independent two-dimensional pattern matching. In *Proc. 33rd Annual Symposium on Foundations of Computer Science (FOCS'92)*, pages 247–257. IEEE CS Press, 1992.

[22] R. Giancarlo. A generalization of the suffix tree to square matrices, with applications. *SIAM Journal on Computing*, 24(3):520–562, 1995.

[23] R. Giancarlo and R. Grossi. On the construction of classes of suffix trees for square matrices: Algorithms and applications. *Information and Computation*, 130(2):151–182, 1996.

[24] G. H. Gonnet. Efficient searching of text and pictures. Report OED-88-02, University of Waterloo, 1988.

[25] J. Kärkkäinen and E. Ukkonen. Two- and higher-dimensional pattern matching in optimal expected time. *SIAM Journal on Computing*, 29(2):571–589, 2000.

[26] G. M. Landau and U. Vishkin. Pattern matching in digitalized image. *Algorithmica*, 12(4/5):375–408, 1994.

[27] V. Mäkinen, G. Navarro, and E. Ukkonen. Algorithms for transposition invariant string matching. In *Proc. 20th International Symposium on Theoretical Aspects of Computer Science (STACS 2003)*, LNCS 2607, pages 191–202, 2003.

[28] G. Navarro and M. Raffinot. *Flexible Pattern Matching in Strings – Practical on-line search algorithms for texts and biological sequences.* Cambridge University Press, 2002. ISBN 0-521-81307-7.

[29] W. Szpankowski. On the height of digital trees and related problems. *Algorithmica*, 6:256–277, 1991.

[30] W. Szpankowski. Analysis of generalized suffix trees. In *Proc. 3rd Annual Symposium on Combinatorial Pattern Matching (CPM'92)*, LNCS v. 664, pages 1–14, 1992.

[31] Tadao Takaoka. Approximate pattern matching with grey scale values. In M. Houle and P. Eades, editors, *Proc. Conference on Computing: The Australian Theory Symposium*, pages 196–203. Australian Computer Science Communications, 1996.

[32] J. Tarhio. A sublinear algorithm for two-dimensional string matching. *Pattern Recognition Letters*, 17:833–838, 1996.

[33] J. Wood. Invariant pattern recognition: a review. *Pattern Recognition*, 29(1):1–17, 1996.

[34] A. C. Yao. The complexity of pattern matching for a random string. *SIAM Journal on Computing*, 8(3):368–387, 1979.

A Probability of Matching under the Mismatches Model

We are interested in the probability of two strings of length ℓ matching with at most k mismatches. For this to hold, at least $\ell - k$ characters must match. Hence, the probability of matching is upper bounded by

$$\frac{\binom{\ell}{k}\sigma^k}{\sigma^\ell} \quad = \quad \frac{1}{\sigma^{\ell-k}}\binom{\ell}{k}$$

where the combinatorial counts all the possible locations for the mismatching characters and σ^k all the ways to choose their value. Note that we are permitting to replace a character by itself, and hence we are including the probability of matching with less than k mismatches as well. The formula is an upper bound because, when less than k mismatches exist, we are counting the same strings more than once.

In the analysis that follows, we call $\beta = k/\ell$ and take it as a constant (which is our case of interest, as seen later). We will prove that, after some length ℓ, the matching probability is $O(\gamma(\beta)^\ell)$, for some $\gamma(\beta) < 1$. By using Stirling's approximation $x! = (x/e)^x\sqrt{2\pi x}(1+O(1/x))$ over the matching probability we have

$$\frac{1}{\sigma^{\ell-k}}\left(\frac{\ell^\ell\sqrt{2\pi\ell}}{k^k(\ell-k)^{\ell-k}\sqrt{2\pi k}\sqrt{2\pi(\ell-k)}}\right)\left(1+O\left(\frac{1}{\ell}\right)\right)$$

which is

$$\left(\frac{1}{\sigma^{1-\beta}\beta^\beta(1-\beta)^{1-\beta}}\right)^\ell \ \ell^{-1/2} \ \left(\frac{1}{\sqrt{2\pi\beta(1-\beta)}}+O\left(\frac{1}{\ell}\right)\right)$$

This formula is of the form $\gamma(\beta)^\ell \ \Theta(1/\sqrt{\ell})$, where we define

$$\gamma(x) \quad = \quad \frac{1}{\sigma^{1-x}x^x(1-x)^{1-x}}$$

Therefore the probability is exponentially decreasing with ℓ if and only if $\gamma(\beta) < 1$, that is,

$$\sigma \quad > \quad \left(\frac{1}{\beta^\beta(1-\beta)^{1-\beta}}\right)^{\frac{1}{1-\beta}} \quad = \quad \frac{1}{\beta^{\frac{\beta}{1-\beta}}(1-\beta)}$$

It is easy to show analytically that $e^{-1} \leq \beta^{\frac{\beta}{1-\beta}} \leq 1$ if $0 \leq \beta \leq 1$, so it suffices that $\sigma > e/(1-\beta)$, or equivalently, $\beta < 1 - e/\sigma$ is a sufficient condition for the probability to be exponentially decreasing with ℓ.

Hence, the result is that the matching probability is very high (actually, $\Theta(1/\sqrt{\ell})$) for $\beta = k/\ell \geq 1 - e/\sigma$, and otherwise it is exponentially decreasing, $O(\gamma(\beta)^\ell/\sqrt{\ell})$, where $\gamma(\beta) < 1$.

Seen another way, we have that the probability of matching is large as long as

$$\ell \quad \leq \quad k_H \quad = \quad \frac{k}{1-e/\sigma}$$

but from then on it is exponentially decreasing.

B Probability of Matching under the Accumulated Model

We are interested in the probability of two random strings of length ℓ matching with threshold k. Our model is as follows: we consider the sequence of ℓ absolute differences between both strings $\delta_1 \ldots \delta_\ell$. The matching condition states that $\sum_{i=1}^{\ell} \delta_i \leq k$.

The number of different sequences of differences satisfying this is $\binom{k+\ell}{\ell}$, what can be seen as the number of ways to insert ℓ divisions into a sequence of k elements. The ℓ divisions divide the sequence into $\ell + 1$ zones. The sizes of the first ℓ zones are the δ_i values and the last allows the sum to be $\leq k$ instead of exactly k. Note that we are pessimistically forgetting about the fact that indeed $\delta_i \leq \sigma$.

Finally, each difference δ_i can be obtained in two ways: $c_i + \delta_i$ and $c_i - \delta_i$, where c_i is the i-th character of the other string (we pessimistically count twice the case $\delta_i = 0$). Therefore, the total matching probability is upper bounded by

$$\frac{2^\ell}{\sigma^\ell} \binom{\ell + k}{k}$$

In the analysis that follows, we call $\beta = k/\ell$ and take it as a constant (which is our case of interest, as seen later). We will prove that, after some length ℓ, the matching probability is $O(\gamma(\beta)^\ell)$, for some $\gamma(\beta) < 1$. By using Stirling's approximation $x! = (x/e)^x \sqrt{2\pi x}(1 + O(1/x))$ over the matching probability we have

$$\frac{2^\ell}{\sigma^\ell} \left(\frac{(k+\ell)^{k+\ell} \sqrt{2\pi(k+\ell)}}{k^k \ell^\ell \sqrt{2\pi k} \sqrt{2\pi \ell}} \right) \left(1 + O\left(\frac{1}{\ell}\right) \right)$$

which is

$$\left(\frac{2(1+\beta)^{1+\beta}}{\sigma \beta^\beta} \right)^\ell \ \ell^{-1/2} \ \left(\sqrt{\frac{1+\beta}{2\pi\beta}} + O\left(\frac{1}{\ell}\right) \right)$$

This formula is of the form $\gamma(\beta)^\ell \ \Theta(1/\sqrt{\ell})$, where we define

$$\gamma(x) \quad = \quad \frac{2(1+x)^{1+x}}{\sigma x^x}$$

Therefore the probability is exponentially decreasing with ℓ if and only if $\gamma(\beta) < 1$, that is,

$$\frac{2(1+\beta)}{\sigma} \left(1 + \frac{1}{\beta} \right)^\beta \quad < \quad 1$$

It can be easily seen analytically that $(1+1/\beta)^\beta \leq e$, so $\beta \ < \ \sigma/(2e) - 1$ is a sufficient condition for the probability to be exponentially decreasing with ℓ.

Hence, the result is that the matching probability is very high $(\Theta(1/\sqrt{\ell}))$ for $\beta = k/\ell \geq \sigma/(2e) - 1$, and otherwise it is $O(\gamma(\beta)^\ell/\sqrt{\ell})$, where $\gamma(\beta) < 1$.

Seen another way, we have that the probability is exponentially decreasing for

$$\ell \ > \ k_A \ = \ \frac{k}{\sigma/(2e) - 1}$$