Efficient One Dimensional Real Scaled Matching

Amihood Amir* Ayelet Butman† Moshe Lewenstein‡ Ely Porat§
Dekel Tsur¶

Abstract

Real Scaled Matching is the problem of finding all locations in the text where the pattern, proportionally enlarged according to an *arbitrary real-sized* scale, appears. Real scaled matching is an important problem that was originally inspired by Computer Vision.

In this paper, we present a new, more precise and realistic, definition for one dimensional real scaled matching, and an efficient algorithm for solving this problem. For a text of length n and a pattern of length m, the algorithm runs in time $O(n \log m + \sqrt{n} m^{3/2} \sqrt{\log m})$.

1 Introduction

The original classical string matching problem [10, 14] was motivated by text searching. Wide advances in technology, e.g. computer vision, multimedia libraries, and web searches in heterogeneous data, have given rise to much study in the field of pattern matching.

Landau and Vishkin [16] examined issues arising from the digitization process. Once the image is digitized, one wants to search it for various data. A whole body of literature examines the problem of seeking an object in an image.

In reality one seldom expects to find an exact match of the object being sought, henceforth referred to as the *pattern*. Rather, it is interesting to find all text locations that "approximately" match the pattern. The types of differences that make up these "approximations" are:

1. *Local Errors* — introduced by differences in the digitization process, noise, and occlusion (the pattern partly obscured by another object).

2. *Scale* — size difference between the image in the pattern and the text.

3. *Rotation* — angle differences between the pattern and text images.

*Bar-Ilan University and Georgia Tech.; email: amir@cs.biu.ac.il; Partially supported by ISF grant 282/01. Part of this work was done when the author was at Georgia Tech, College of Computing and supported by NSF grant CCR–01–04494.

†Holon Academic Institute of Technology, email: butmosh@zahav.net.il.

‡Bar-Ilan University; email: moshe@cs.biu.ac.il.

§Bar-Ilan University; email: porately@cs.biu.ac.il.

¶University of Haifa; email: dekelts@cs.haifa.ac.il

Some early attempts to handle local errors were made in [15]. These results were improved in [8]. The algorithms in [8] heavily depend on the fact that the pattern is a rectangle. In reality this is hardly ever the case. In [6], Amir and Farach show how to deal with local errors in non-rectangular patterns.

The rotation problem is to find all rotated occurrences of a pattern in an image. Fredriksson and Ukkonen [12], made the first step by giving a reasonable definition of rotation in discrete images and introduce a filter for seeking a rotated pattern. Amir et al. [2] presented an $O(n^2m^3)$ time algorithm. This was improved to $O(n^2m^2)$ in [7].

For scaling it was shown [5, 9] that all occurrences of a given rectangular pattern in a text can be found in all integer scales in linear time. The first result handling real scales was given in [3]. In that paper, a linear time algorithm was given for one-dimensional real scaled matching. In [4], the problem of two-dimensional real scaled matching was defined, and an efficient algorithm was presented for the problem.

The definition of one-dimensional scaling in [3] has the following drawback: For a pattern P of length m and a scale r, the pattern P scaled by r can have a length which is far from mr. In this paper, we give a more natural definition for scaling, which has the property that the length of P scaled by r is mr rounded to the nearest integer. This definition is derived from the definition of two-dimensional scaling which was given in [4]. We give an efficient algorithm for the scaled matching problem under the new definition of scaling: For a text T of length n and a pattern P of length m, the algorithm finds in T all occurrences of P scaled to any real value in time $O(n \log m + \sqrt{n}m^{3/2}\sqrt{\log m})$. In the design of the algorithm we use some ideas from [4], together with some new ones.

2 Scaled Matching Definition

Let T and P be two strings over some finite alphabet Σ. Let n and m be the lengths of T and P, respectively.

Definition 1. A *pixel* is an interval $(i-1, i]$ on the real line $\mathbb{R}$, where i is an integer. The *center* of a pixel $(i-1, i]$ is its geometric center point, namely the point $i - 0.5$.

Definition 2. Let $r \in \mathbb{R}$, $r > 1$. The *r-ary pixel array* for P consists of m intervals of length r, which are called *r-intervals*. The i-th r-interval is $((i-1)r, ir]$. Each interval is identified with the value from Σ: The i-th interval is identified with the i-th letter of P. For each pixel center that is inside some r-interval, we assign the letter that corresponds to that interval. The string obtained by concatenating all the letters assign to the pixel centers from left to right is denoted by P^r, and called *P scaled to r*.

The *scaled matching problem* is to find all the locations in the text T in which there is an occurrence of P scaled to some $r \geq 1$.

Example 1. Let $P = aabccc$ then $P^{\frac{3}{2}} = aaabbcccc$, and let $T = aabdaaaabbcccccb$. There is an occurrence of P scaled to $\frac{3}{2}$ at text location 6.

Let $x \in \mathbb{R}$. $\|x\|$ denotes the *rounding* of x, i.e.

$$\|x\| = \begin{cases} \lfloor x \rfloor & \text{if the fraction part of } x \text{ is less than .5} \\ \lceil x \rceil & \text{otherwise.} \end{cases}$$

We need the following technical claim.

Claim 1. *Let k, k', and l be some positive integers.*

1. $\{r|\ \|rk\| = l\} = [\frac{l-0.5}{k}, \frac{l+0.5}{k})$.
2. $\{r|\ \|r(k+k')\| - \|rk'\| = l\} \subseteq (\frac{l-1}{k}, \frac{l+1}{k})$.

Proof. The first part follows immediately from the definition of rounding. To prove the second part, note that $x - 0.5 < \|x\| \leq x + 0.5$, so

$$\|r(k+k')\| - \|rk'\| < r(k+k') + 0.5 - (rk' - 0.5) = rk + 1$$

and

$$\|r(k+k')\| - \|rk'\| > r(k+k') - 0.5 - (rk' + 0.5) = rk - 1.$$

Hence, the r's that satisfy $\|r(k+k')\| - \|rk'\| = l$ are those that satisfy $rk-1 < l < rk+1$, which implies $\frac{l-1}{k} < r < \frac{l+1}{k}$. ■

3 A Local Verification Algorithm

One possible straightforward approach to solving the scaled matching problem is to verify for each location of the text whether the scaled occurrence definition holds at that location. However, even this verification needs some clarification. To simplify, we split the input strings into two parts, the symbol part and the run length part (defined below), and then handle each part separately. An additional benefit is that this representation also compresses the text and pattern and hence will lead to faster algorithms.

Definition 3. Let $S = \sigma_1\sigma_2\cdots\sigma_n$ be a string over some alphabet Σ. The *run-length representation* of string S is the string $S' = \sigma_1'^{r_1}\sigma_2'^{r_2}\cdots\sigma_k'^{r_k}$ such that: (1) $\sigma_i' \neq \sigma_{i+1}'$ for $1 \leq i < k$; and (2) S can be described as concatenation of the symbol σ_1' repeated r_1 times, the symbol σ_2' repeated r_2 times, ..., and the symbol σ_k' repeated r_k times. We denote by $S^\Sigma = \sigma_1'\sigma_2'\cdots\sigma_k'$, the *symbol part* of S', and by $c(S) = r_1 r_2 \cdots r_k$, the *run-length part* of S' ($c(S)$ is a string over the alphabet of natural numbers).

The locator function between S and S' is $\mathrm{loc}_S(i) = j$, where j is the index for which $\sum_{l=1}^{j-1} r_l < i \leq \sum_{l=1}^{j} r_l$.

The *center* of S, denoted C_S, is the substring of S that contains all the letters of S except the first r_1 letters and the last r_k letters.

Example 2. Let $S = aaaaaabbcccaabbbddddd$ then $S' = a^6b^2c^3a^2b^3d^5$, $S^\Sigma = abcabd$ and $c(S) = 623235$. The locator function is $\mathrm{loc}_S(1) = 1, \mathrm{loc}_S(2) = 1, \ldots, \mathrm{loc}_S(6) = 1, \mathrm{loc}_S(7) = 2, \ldots, \mathrm{loc}_S(21) = 6$. The center of S is *bbcccaabbb*.

Scaled matching requires finding all scaled occurrences of P in T. To achieve this goal we will use P' and T'. There are two requirements for a scaled occurrence of P at a given location of T. The first is that P^Σ matches a substring of T^Σ beginning at location $\mathrm{loc}_T(i)$ of T. This can be verified in linear time with any classical pattern matching algorithm, e.g. [14]. The second requirement is that there is a scale r for which P scales properly

to match at the appropriate location in T. For this we will use $c(P)$ and $c(T)$. Since the first requirement is easy to verify, from here on we will focus on the second requirement.

Denote $m' = |c(P)|$ and $n' = |c(T)|$. We also denote $c(P) = p_1, \ldots, p_{m'}$ and $c(T) = t_1, \ldots, t_{n'}$. We assume that $m' \geq 3$ and $n' \geq 3$ otherwise the problem is easily solvable in linear time. When a scaled match occurs at location i of T the location in the compressed text that corresponds to it is $j = \mathrm{loc}_T(i)$. However, only part of the full length of t_j may need to match the scaled pattern. More precisely, a length $\hat{t}_i = \sum_{l=1}^{j} t_l + 1 - i$ piece of t_j needs to match the scaled pattern. The full set of desired scaling requirements follows.

Scaling requirements at text location i, where $\mathrm{loc}_T(i) = j$

$$\begin{aligned}
\|p_1 \cdot r\| &= \hat{t}_i \\
\|(p_1 + p_2) \cdot r\| &= \hat{t}_i + t_{j+1} \\
&\vdots \\
\|(p_1 + \cdots + p_k) \cdot r\| &= \hat{t}_i + t_{j+1} + \cdots + t_{j+k-1} \\
&\vdots \\
\|(p_1 + \cdots + p_{m'-1}) \cdot r\| &= \hat{t}_i + t_{j+1} + \cdots + t_{j+m'-2} \\
\|m \cdot r\| &\leq \hat{t}_i + t_{j+1} + \cdots + t_{j+m'-1}
\end{aligned}$$

The following claim follows from the discussion above and its correctness follows directly from the definition.

Claim 2. *Let P be a pattern and T a text. There is an occurrence of P^r at location i of T iff P^Σ matches at location $\mathrm{loc}_T(i)$ of T^Σ and the scaling requirements for location i are satisfied.*

Claim 3. *Let i be a location of T. The scaling requirements for i can be verified in $O(m')$ time.*

Proof. The scaling requirements are verified by finding the set of all scales r that satisfy requirements. By Claim 1, for each of the first $m' - 1$ scaling requirements, the set of all r's that satisfy the requirement is an interval of length 1. Moreover, the last scaling requirement demands that $r \in [1, (\hat{t}_i + t_{j+1} + \cdots + t_{j+m'-1} + 0.5)/m)$. The intersection of these intervals is set of all r's for which P^r appears in location i. This intersection can be found in $O(m')$ time. ∎

The following straightforward algorithm can now be devised: The algorithm checks the scaling requirements for every location i of T.

Running time: There are $n - m + 1$ locations in T. For each location the existence of a scaled match can be checked in $O(m')$ time by Claim 3. So the overall time of the algorithm is $O(nm')$.

4 A Dictionary Based Solution

A different approach for solving the scaled matching problem is to create a dictionary containing the run length part of P scaled at all possible scales. Substrings of the compressed text can then be checked for existence in the dictionary. The problem with this solution is that there may be many scales and hence many different strings in the dictionary. In fact, the dictionary to be created could be as large as, or larger than, the running time of the naive algorithm. To circumvent this problem we will keep in the dictionary only scaled instances of the pattern with a scale at most α, where the value of α will be determined later. Checking for occurrences of the pattern with a scale larger than α will be performed using the algorithm from the previous section.

To bound the number of strings in the dictionary, we use the following lemma, which is a special case of Lemma 1 in [4].

Lemma 4. *Let pattern P be scaled to size $l \geq m$. Then there are $k \leq m'$ intervals, $[a_1, a_2), [a_2, a_3), \ldots, [a_k, a_{k+1})$, where $a_1 < a_2 < \cdots < a_{k+1}$ for which the following hold:*

1. *$P^{r_1} = P^{r_2}$ if r_1 and r_2 are in the same interval.*
2. *$P^{r_1} \neq P^{r_2}$ if r_1 and r_2 are in different intervals.*
3. *P^r has length l if and only if $r \in [a_1, a_{k+1})$.*

Proof. By Claim 1, if the length of P^r is l, then r belongs to the interval $I = [\frac{l-0.5}{m}, \frac{l+0.5}{m})$. Consider the r-ary pixel array for P when the value of r goes from $r = \frac{l-0.5}{m}$ to $r = \frac{l+0.5}{m}$. Consider some value of r in which a new scaled pattern is reached, namely $P^r \neq P^{r-\epsilon}$ for every $\epsilon > 0$. By definition, this happens when some pixel center changes its r-interval, or in other words, when a pixel center coincides with the right endpoint of some r-interval. The right endpoint of the rightmost r-interval moves a distance of exactly 1 when r goes over I, and each other endpoint moves a distance smaller than 1. Thus, each endpoint coincides with a pixel center at most one time. ■

4.1 Building the dictionary

Let $\mathcal{P}$ be a set containing $c(C_{P^r})$ for every scaled pattern P^r with $r \leq \alpha$. By Lemma 4, the number strings in $\mathcal{P}$ is $O(\alpha m m')$. For each $S \in \mathcal{P}$, let R_S be the set of all values of r such that $c(C_S) = S$. Each set R_S is a union of intervals, and let $|R_S|$ denote the number the intervals in R_S.

Example 3. Let $P = abcd$ and $\alpha = 2$. Then,

$$c(P^r) = \begin{cases} 1111 & \text{for } r \in [1, 1.125) \\ 1112 & \text{for } r \in [1.125, 7/6) \\ 1121 & \text{for } r \in [7/6, 1.25) \\ 1211 & \text{for } r \in [1.25, 1.375) \\ 1212 & \text{for } r \in [1.375, 1.5) \\ 2121 & \text{for } r \in [1.5, 1.625) \\ 2122 & \text{for } r \in [1.625, 1.75) \\ 2212 & \text{for } r \in [1.75, 11/6) \\ 2221 & \text{for } r \in [11/6, 1.875) \\ 2222 & \text{for } r \in [1.875, 2) \end{cases}$$

Thus, $\mathcal{P} = \{11, 12, 21, 22\}$, $R_{11} = [1, 7/6)$, $R_{12} = [7/6, 1.25) \cup [1.5, 1.75)$, $R_{21} = [1.25, 1.5) \cup [1.75, 11/6)$, and $R_{22} = [11/6, 2]$.

Lemma 5. *For every $S \in \mathcal{P}$, $|R_S| = O(m)$.*

Proof. Let l be the sum of the characters of S, and let a be the number of characters in C_P, namely $a = p_2 + \cdots + p_{m'-1}$. If $r \in R_S$, then $\|rp_1\| = \hat{t}_i$ and $\|r(p_1 + a)\| = \hat{t}_i + l$, so $\|r(p_1 + a)\| - \|rp_1\| = l$. By Claim 1 we obtain that $r \in (\frac{l-1}{a}, \frac{l+1}{a})$. For $r \in (\frac{l-1}{a}, \frac{l+1}{a})$, the length of P^r is in the interval $[\ \|\frac{l-1}{a}m\|, \|\frac{l+1}{a}m\|\ ]$. From Lemma 4, it follows that $|R_S| = O(m' \cdot m/a)$. We have assume that $m' \geq 3$, so $a \geq m' - 2 \geq \frac{1}{3}m'$. Hence, $|R_S| = O(m)$. ∎

We now describe how to build the dictionary. Instead of storing in the dictionary the actual strings of $\mathcal{P}$, we will assign a unique name for every string in $\mathcal{P}$ using the naming technique of Karp et al. [13] (see also [1]), and we will store only these names.

The first step is generating all the scaled patterns P^r for $r \leq \alpha$ in an increasing order of r. This is done by finding all the different values of r in which some pixel center changes its r-interval. These values can be easily found for each pixel center, and then sorted using bin sorting. Denote the sorted list by L. We assume that for each value r in L, only a single pixel center changes its r-interval (if k centers simultaneously change their r-intervals for some value of r, we put k copies of this value in L).

Afterward, build an array $A[1..m' - 2]$ that contains the run length part of C_P. For simplicity, assume that $m' - 2$ is a power of 2 (otherwise, we can append zeros to the end of A until the size of A is a power of 2). Now, compute a name for A by giving a name for every sub-array of A of the form $A[j2^i + 1..(j + 1)2^i]$. The name given to a sub-array $A[j..j]$ is equal to its content. The name given to a sub-array $A' = A[j2^i + 1..(j + 1)2^i]$ depends on the names given to the two sub-arrays $A[2j2^{i-1} + 1..(2j + 1)2^{i-1}]$ and $A[(2j + 1)2^{i-1} + 1..(2j + 2)2^{i-1}]$. If the names of these sub-arrays are a and b, respectively, then check whether the pair of names (a, b) was encountered before. If it was, then the name of A' is the name that was assigned to the pair (a, b). Otherwise, assign a new name to the pair (a, b) and also assign this name to A'.

After naming A, traverse L, and for each value r in L, update A so it will contain the run length part of C_{P^r}. After each time a value in A is changed, update the names

of the $\log(m'-2)+1$ sub-arrays of A that contain the position of A in which the change occurred.

During the computation of the names we also compute the sets R_S for all $S \in \mathcal{P}$.

Running time: Each value in L causes at most two changes in the array A, and thus creates at most $2(\log(m'-2)+1)$ new names. Thus, the total number of distinct names is $O(m' + |L| \log m')$. From Lemma 4 we have that $|L| = O(\alpha m m')$, so the number of distinct names is at most $s = O(\alpha m m' \log m')$. We store the pairs of names in an $s \times s$ table. Thus, the initial naming of A takes $O(m')$ time, and each update takes $O(\log m')$ time. Therefore, the time for computing all the names is $O(\alpha m m' \log m')$. Using the approach of [11], the space complexity can be reduced to $O(s)$.

4.2 Scanning the text

The first step is naming every substring of $c(T)$ of length 2^i for $i = 0, \ldots, \log(m'-2)$. The name of a substring of length 1 is equal to its content. The name of a substring T' of length 2^i $(i > 0)$ is computed from the names of the two substrings of length 2^{i-1} whose concatenation forms T'. The naming is done using the same $L \times L$ array that was used for the naming of A, so the names in this stage are consistent with the names in the dictionary building stage. In other words, a substring of T that is equal to a string S from $\mathcal{P}$, will get the same name as S.

Now, for every location i of T, compute the range of scales r that satisfy the first and last two scaling requirements. If this range is empty, proceed to the next i. Otherwise, suppose that this interval is $[r_1, r_2)$. If $r_2 > \alpha$, check all the other scaling requirements. If $r_2 \leq \alpha$, check whether the name of the substring of $c(T)$ of length $m'-2$ that begins at $\mathrm{loc}_T(i)+1$ is equal to the name of a string $S \in \mathcal{P}$. If there is such a string S, compute $[r_1, r_2) \cap R_S$, and report a match if the intersection is not empty.

Running time: Computing the names takes $O(n' \log m')$ time. By storing each set R_S using a balanced binary search tree, we can compute the intersection $[r_1, r_2) \cap R_S$ in time $O(\log |R_S|) = O(\log m)$ (the equality follows from Lemma 5). Therefore, the time complexity of this stage is $O(n \log m + l m')$, where l is the number of locations in which all the scaling requirements are checked. Let X_j be the set of all such locations i with $\mathrm{loc}_T(i) = j$, and let X be the set of all indices j for which X_j is not empty. Clearly, $l = \sum_{j \in X} |X_j|$. The following lemmas give an upper bound on l.

Lemma 6. *For every j, $|X_j| = O(1 + p_1/m')$.*

Proof. Fix some j. Let $a = p_2 + \cdots + p_{m-1}$ and $l = t_{j+1} + \cdots + t_{j+m'-2}$. Suppose that $i \in X_j$. As the first and second last scaling requirements are satisfied, we have that $\|r(p_1+a)\| - \|rp_1\| = (\hat{t}_i + l) - \hat{t}_i = l$. By Claim 1, $r \in (\frac{l-1}{a}, \frac{l+1}{a})$, so $rp_1 \in (\frac{l-1}{a}p_1, \frac{l+1}{a}p_1)$. Since $i = t_1 + \cdots + t_j + 1 - \hat{t}_i = t_1 + \cdots + t_j + 1 - \|rp_1\|$ and this is true for every $i \in X_j$, it follows that

$$|X_j| \leq \left| \left\{ \|x\| \;\middle|\; x \in \left(\frac{l-1}{a} p_1, \frac{l+1}{a} p_1 \right) \right\} \right| \leq 2 + \frac{2p_1}{a} \leq 2 + \frac{6p_1}{m'},$$

where the last inequality follows from the fact that $m' \geq 3$. ∎

Lemma 7. $|X| = O(\frac{n}{\alpha p_1})$.

Proof. Suppose that $j \in X$, and let i be some element in X_j. Let $[r_1, r_2)$ be the scales interval computed for location i using the first and last two scaling requirements. By the definition of the algorithm, $r_2 > \alpha$. The interval of r's that satisfy the first scaling requirement is $[\frac{\hat{t}_i - 0.5}{p_1}, \frac{\hat{t}_i + 0.5}{p_1})$, so we obtain that $\frac{\hat{t}_i + 0.5}{p_1} \geq r_2 > \alpha$. Thus, $p_j \geq \hat{t}_i > \alpha p_1 - 0.5$. Since this is true for every $j \in X$, it follows that $|X| \leq \frac{n}{\alpha p_1 - 0.5}$. ■

By Lemmas 6 and 7, $l = O(\frac{n}{\alpha p_1}(1 + \frac{p_1}{m'})) = O(\frac{n}{\alpha})$. Therefore, the total time complexity of the algorithm is $O(\alpha m m' \log m' + n \log m + n m'/\alpha)$. This expression is minimized by choosing $\alpha = \sqrt{n/(m \log m')}$. We obtain the following theorem:

Theorem 8. *The scaled matching problem can be solved in* $O(n \log m + \sqrt{nm} m' \sqrt{\log m'})$ *time.*

References

[1] A. Amir, A. Apostolico, G. M. Landau, and G. Satta. Efficient text fingerprinting via Parikh mapping. *J. of Discrete Algorithms*, 1(5–6):409–421, 2003.

[2] A. Amir, A. Butman, A. Crochemore, G. M. Landau, and M. Schaps. Two-dimensional pattern matching with rotations. In *Proc. 14th Annual Symposium on Combinatorial Pattern Matching (CPM '03)*, pages 17–31, 2003.

[3] A. Amir, A. Butman, and M. Lewenstein. Real scaled matching. *Information Processing Letters*, 70(4):185–190, 1999.

[4] A. Amir, A. Butman, M. Lewenstein, and E. Porat. Real two dimensional scaled matching. In *Proc. 8th Workshop on Algorithms and Data Structures (WADS '03)*, pages 353–364, 2003.

[5] A. Amir and G. Calinescu. Alphabet independent and dictionary scaled matching. In *Proc. 7th Annual Symposium on Combinatorial Pattern Matching (CPM 96)*, LNCS 1075, pages 320–334. Springer-Verlag, 1996.

[6] A. Amir and M. Farach. Efficient 2-dimensional approximate matching of half-rectangular figures. *Information and Computation*, 118(1):1–11, April 1995.

[7] A. Amir, O. Kapah, and D. Tsur. Faster two dimensional pattern matching with rotations. In *Proc. 15th Annual Symposium on Combinatorial Pattern Matching (CPM '04)*, pages 409–419, 2004.

[8] A. Amir and G. Landau. Fast parallel and serial multidimensional approximate array matching. *Theoretical Computer Science*, 81:97–115, 1991.

[9] A. Amir, G. M. Landau, and U. Vishkin. Efficient pattern matching with scaling. *Journal of Algorithms*, 13(1):2–32, 1992.

[10] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Comm. ACM*, 20:762–772, 1977.

[11] G. Didier, T. Schmidt, J. Stoye, and D. Tsur. Character sets of strings. Submitted, 2004.

[12] 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. Springer, 1998.

[13] R. Karp, R. Miller, and A. Rosenberg. Rapid identification of repeated patterns in strings, arrays and trees. *Symposium on the Theory of Computing*, 4:125–136, 1972.

[14] D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Comp.*, 6:323–350, 1977.

[15] K. Krithivansan and R. Sitalakshmi. Efficient two dimensional pattern matching in the presence of errors. *Information Sciences*, 13:169–184, 1987.

[16] G. M. Landau and U. Vishkin. Pattern matching in a digitized image. *Algorithmica*, 12(3/4):375–408, 1994.