Real Scaled Matching

Amihood Amir[*†] Ayelet Butman[*] Moshe Lewenstein[* ‡]

Bar-Ilan University and Georgia Tech   Bar-Ilan University   Bar-Ilan University

Abstract

Scaled Matching refers to the problem of finding all locations in the text where the pattern, proportionally enlarged according to an arbitrary integral scale, appears. Scaled matching is an important problem that was originally inspired by problems in Vision.

However, in real life, a more natural model of scaled matching is the *Real Scaled Matching* model. Real scaled matching is an extended version of the scaled matching problem allowing *arbitrary real-sized* scales, approximated by some function, e.g. truncation.

It has been shown that the scaled matching problem can be solved in linear time. However, even though there has been follow-up work on the problem, it remained an open question whether real scaled matching could be solved faster than the simple solution of $O(nm)$ time, where n is the text size and m is the pattern size.

Using a new approach we show how to solve the real scaled matching problem in linear time.

1 Introduction

The original classical string matching problem [BM77, KMP77] was motivated by text searching. Indeed practically every text editor uses some variant of the Boyer-Moore algorithm [BM77].

[*]Department of Mathematics and Computer Science, Bar-Ilan University, 52900 Ramat-Gan, Israel, (972-3)531-8407; {amir, ayelet, moshe}@cs.biu.ac.il

[†]Tel. (972-3)531-8770; Partially supported by NSF grant CCR-96-10170 and BSF grant 96-00509.

[‡]Supported by the Ministry of Science, Israel, Eshkol Fellowship 061-1-97

Wide advances in technology, e.g. computer vision, multimedia libraries, and web searches in heterogeneous data, point to a glaring lack of a theory of multidimensional matching [Pen92]. The last decade has seen some progress in this direction.

Landau and Vishkin [LV94] examine issues relating to the digitization process itself. 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* - The pattern image appearing in the text in a different angle.

Some early attempts to handle local errors were made in [KS87]. These results were improved in [AL91]. The algorithms in [AL91] heavily depend on the fact that the pattern is a rectangle. In reality this is hardly ever the case. In [AF95], Amir and Farach show how to deal with local errors in non-rectangular patterns.

The rotation problem has proven quite difficult. There is currently no known asymptotically efficient algorithm for finding all rotated occurrences of a pattern in an image. In a recent paper [FU98], Fredriksson and Ukkonen give a reasonable definition of rotation in discrete images and introduce a filter for seeking a rotated pattern.

More progress has been made with scaling. In [ALV92] it was shown that all occurrences of a given rectangular pattern in a text can be found in *all discrete scales* in linear time. By discrete scales we mean natural numbers, i.e. the pattern scaled to sizes $1, 2, 3, \ldots$. The algorithm was linear for fixed bounded alphabets. It was not linear for unbounded alphabets.

This result was improved in [AC96]. Amir and Calinescu presented a new idea for solving the scaled matching problem by defining a size invariant "signature" of the pattern. This led to a simpler, alphabet-independent linear-time algorithm for the two dimensional scaled matching problem with discrete scales. In addition this technique enabled solving the *dictionary* scaled matching problem. In this model, a dictionary of patterns is preprocessed. Subsequently, for a given text, all occurrences of dictionary patterns are found, in any discrete scale, in time linear in the text size.

All above papers dealt with *discrete scales* only. There is some justification for dealing with discrete scales in a combinatorial sense, since it is not clear what is a fraction of a pixel.

However, in reality an object may appear in non-discrete scales. It is necessary to, both, define the combinatorial meaning of such scaling, and present efficient algorithms for the problem's solution.

This paper is a first step in this direction. We limit ourselves to string matching with non-discrete scales. Here the problem definition, at least, is natural. The formal definition is presented in section 2. Various one-dimensional problems on numerical strings where matching is defined by certain arithmetic operations were discussed by Eilam-Tsoreff and Vishkin [ETV88]. Our problem has a similar flavor, but deals with real numbers, rather than natural numbers. This adds an extra complication.

We show a linear time algorithm that finds all occurrences of a pattern string in a text string, in all *real* scales.

2 Definitions

Denote the string $aa\cdots a$, where a is repeated r times, by a^r. Let $S = a_1^{r_1}a_2^{r_2}...a_j^{r_j}$ be a string for which $a_i \neq a_{i+1}$. Let $k \in \mathcal{N}$, we say that S' is a *k-scaling* of S if $S' = a_1^{r_1k}a_2^{r_2k}...a_j^{r_jk}$. In the *scaled matching problem* we are given a pattern and text and are interested in finding all locations in the text where a k-scaled occurrence of the pattern appears, for any $k \in \mathcal{N}$.

For the previously described string S we define a *real k-scaling*, for $k \in \mathcal{R}$, to be $a_1^{\lfloor r_1k\rfloor}a_2^{\lfloor r_2k\rfloor}...a_j^{\lfloor r_jk\rfloor}$. We now give the formal definition of our problem.

Definition 1 *(Real scaled matching)*

Input: *A pattern P, of length m, and a text T of length n.*

Output: *All locations in the text where there appears a real k-scaling of P, for some $k \in \mathcal{R}$, $k \geq 1$.*

The reason for the $k \geq 1$ assumption is that we are interested in scaling upwards only. Although from a definition standpoint one may define scaling by factors less than 1, but we choose to avoid the conceptual problems that may arise from such a definition. The problems are loss of resolution. We will be losing pixels by shrinking the image too much. In fact, once $k < \frac{1}{m}$ our pattern virtually "disappears" and then there is a scaled match at every text location. Consequently, we assume that the pattern is given at the smallest resolution allowed, and seek its occurence in all scales greater than 1.

Note that we defined the approximation function to be truncation but the results in this paper hold for many other functions as well, e.g. rounding.

The above definition does not precisely capture our motivation. The following alternate definition seems better suited to fit our motivation.

Definition 2 *(Alternate definition)*

Input: *A pattern* $P = a_1^{r_1} a_2^{r_2} \ldots a_j^{r_j}$*, of length* m*, and a text* T *of length* n*.*

Output: *All locations in the text where the substring* $a_1^{c_1} a_2^{\lfloor r_2 k \rfloor} \ldots a_{j-1}^{\lfloor r_{j-1} k \rfloor} a_j^{c_j}$ *appears, where* $c_1 \geq \lfloor r_1 k \rfloor$ *and* $c_j \geq \lfloor r_j k \rfloor$*.*

However, for the rest of this paper we choose to follow definition 1 for the sake of simplicity. This choice of definition does not cause loss of generality for the following reason.

Lemma 1 *Assume that the real scaled matching problem as defined by definition 1 can be solved in time* $O(f(n))$*. Then the real scaled matching problem as defined by definition 2 can be solved in time* $O(f(n))$*.*

Proof: Let $P = a_1^{r_1} a_2^{r_2} \ldots a_j^{r_j}$. Consider $P' = a_2^{r_2} \ldots a_{j-1}^{r_{j-1}}$. Find all real scaled occurrences of P' in T by definition 1. This is done in time $O(f(n))$.

For each occurrence of P' check if it is preceeded by $a_1^{c_1}$ and followed by $a_j^{c_j}$, where $c_1 \geq \lfloor r_1 k \rfloor$ and $c_j \geq \lfloor r_j k \rfloor$. If such a verification can be done in constant time, then an overall $O(n)$ work will enable a decision about which of the occurrences of P' is also an occurrence of P by definition 2. Thus the total time for finding all real scaled occurrences of P by definition 2 is $O(f(n) + n)$. It is clear that $f(n)$ is $\Omega(n)$, making the total time $O(f(n))$.

We still need to show that the verification of the start and end of the pattern can be done in constant time. This can be easily done following a linear time preprocessing of the text that counts the consecutive runs of every symbol. In fact, we will see in the following sections that our solution to real scaled matching using definition 1 will do this preprocessing in any event. $\square$

¿From this point on, we will forget about the alternate definition. When we say *real scaled matching* we will be referring definition 1.

3 A simple solution

A natural observation, that we have already hinted at in the last section, is that we can view a real scaled match as a two-fold match. The first part is that the pattern structure matches the text structure at the appropriate text-location, i.e. the a_1's appear and then the a_2's etc. The second part is that the pattern is indeed properly scaled. Formally, for string

$S = a_1^{r_1} a_2^{r_2} ... a_j^{r_j}$ we denote its *run-length* string $S^L = r_1 ... r_j$ and its *form* string $S^F = a_1 ... a_j$. Finally, denote the *k-scaled run length* string $kS^L = \lfloor r_1 k \rfloor \lfloor r_2 k \rfloor ... \lfloor r_j k \rfloor$. The following lemma is immediate.

Lemma 2 *Let P be a pattern and T be a text, such that $T^L = r_1 ... r_t$. Let $\Sigma_{j=1}^{h} r_j \leq i < \Sigma_{j=1}^{h+1} r_j$. There is a real k-scaling appearance of P at text-location i iff P^F appears at location h of T^F and kP^L appears at location h of T^L.*

This lemma yields a simple solution to the real scaled matching problem. The locations where P^F appear in T^F can be found using any linear time pattern matching algorithm, e.g. [KMP77]. Therefore it is sufficient to check each location where P^F indeed appeared for the existence of a k, such that kP^L appears in T^L.

Let l and r be natural numbers. The k's that satisfy the equation $l = \lfloor kr \rfloor$ are those that belong to $[\frac{l}{r}, \frac{l+1}{r})$. Therefore, to check for the existence of k such that kP^L *matches* in a given location of T^L it is enough to see that the intersection of each of the $|P^L|$ intervals is non-empty. This takes overall $O(|P^L||T^L|)$ time. Since $|P^L| \leq |P|$ and $|T^L| \leq |T|$, this can be bounded by $O(nm)$, where $n = |T|$ and $m = |P|$.

Example: Let $T = aaaabbccccaaaaaaabbbbcccccddd$ and $P = aabccaaabb$.

It can be easily checked that there is a real $\frac{7}{3}$-scaling of P in the first text location. Yet, even though there is a real scaling there is no exact scaling of P into T in the first text location.

To see the algorithm's steps, note that $T^F = abcabcd$ and $P^F = abcab$. $T^L = 4\ 2\ 4\ 7\ 4\ 5\ 3$ and $P^L = 2\ 1\ 2\ 3\ 2$. It is easy to see that P^F matches T^F in the first location. Now we can line up P^L under the first location of T^L and see the appropriate intersections.

$$\begin{array}{rccccccc} T^L = & 4 & 2 & 4 & 7 & 4 & 5 & 3 \\ P^L = & 2 & 1 & 2 & 3 & 2 & & \\ & [2, \frac{5}{2}) & [2, 3) & [2, \frac{5}{2}) & [\frac{7}{3}, \frac{8}{3}) & [2, \frac{5}{2}) & & \end{array}$$

Since, $[2, \frac{5}{2}) \cap [2, 3) \cap [2, \frac{5}{2}) \cap [\frac{7}{3}, \frac{8}{3}) \cap [2, \frac{5}{2}) = [\frac{7}{3}, \frac{5}{2})$, the intersection is not empty and any $k \in [\frac{7}{3}, \frac{5}{2})$ is a legal k-scale for a match in the first location.

¿From the above discussion it is apparent that the "bottleneck" of the algorithm is finding the matches of the run-length pattern in the run-length text, since occurrences of the form part can be easily found in linear time. Therefore, from this point on, we concentrate on finding the real scaled matches of P^L in T^L.

4 Using parameterized matching

Definition 3 *(Parameterized-matching). Let Σ be an alphabet, $T = t_1 \cdots t_n$ the text and $P = p_1 \cdots p_m$ the pattern, $t_i, p_j \in \Sigma$, $i = 1, ..., n$, $j = 1, ..., m$. We say that P mapped-matches or simply m-matches T in location i if and only if there exists a one-one function $\pi : \Sigma \rightarrow \Sigma$ such that $\pi(P)$ $(= \pi(p_1)\pi(p_2)...\pi(p_m)) = t_i t_{i+1} \cdots t_{i+m-1}$.*

The *parameterized-matching* problem is to determine all m-matching of P in T. Amir, Farach and Muthukrishnan [AFM94] have shown an algorithm that takes time $O(nlog\sigma)$, where $\sigma = min(m, |\Sigma|)$, independent of the alphabet. For alphabets that can be sorted in linear time their algorithm runs in $O(n)$ time.

We utilize parameterized pattern matching for our problem in the following way.

Lemma 3 *Let P be a pattern and T a text. There exists a k such that kP^L matches T^L in location i only if P^L m-matches T^L in location i.*

Proof: Say there exists k such that kP^L matches T^L. If two characters (numbers) of P^L are the same, then clearly they are the same in kP^L. Conversely, any two different numbers in P^L must be different in kP^L. This is true for the following reason. Let $a, b \in P^L$ be two different numbers. Without loss of generality we may assume that $b \geq a + 1$. Thus $\lfloor kb \rfloor \geq \lfloor k(a+1) \rfloor = \lfloor ka + k \rfloor$. But since $k \geq 1$, we have $\lfloor ka + k \rfloor > \lfloor ka \rfloor$. □

In our case T^L and P^L are composed of numbers not exceeding n. Therefore we can parameterize-match P^L in T^L in $O(n)$ time. Once we know that for a given location there is a parameterized match of P^L in T^L then for any two characters (numbers) in P^L that are the same, we need only check one for its intersection with its corresponding text number. Therefore, we need only check one representative of each different character in P^L. This leads to the following algorithm.

Algorithm:

1. Split P into P^F and P^L and T into T^F and T^L.
2. Let $p_{i_1}, ..., p_{i_\ell}$ be representatives of the different characters of P^L.
3. Find all occurrences of P^F in T^F.
4. Find all parameterized-matches of P^L in T^L.
5. for each location i of T^F in which P^F appears and for which P^L matches T^L do

    Check the intersection of the intervals between $p_{i_1}, ..., p_{i_\ell}$ and their corresponding characters in T^L.

```
        Intersection not empty? There's a match.
        Otherwise - No match.
    end
```

end Algorithm

The correctness of this algorithm follows from the discussion beforehand. Since the "sum" of the characters (numbers) of P^L is the length of P, i.e. m, there can be at most $\sqrt{m}$ different numbers in P^L. Therefore, the time of the algorithm is $O(n\sqrt{m})$.

5 Tighter Analysis of the Algorithm

We now utilize the fact that P^L and T^L are actually compressed strings of P and T. In the following lemma we show that the more m-matches there are, the more compressed P^L and T^L are. This gives a limit on the number of m-matches possible.

Lemma 4 *Let P be a pattern of length m and T a text of length n. If P^L contains ℓ different characters then there are at most $\frac{n}{2\ell}$ m-matches of P^L in T^L.*

Proof: We prove the claim by a counting argument. Let $p_{i_1}, ..., p_{i_\ell}$ be representatives of the different characters in P^L. Say P^L m-matches at location i of T^L. Since $p_{i_1}, ..., p_{i_\ell}$ are different characters, the characters in T^L that match them in the m-matching must be different from one another. The sum of these characters is at least $\Sigma_{j=1}^{\ell} j > \frac{\ell^2}{2}$.

Assume that there are x m-matches of P^L in T^L. The sum of all numbers in T^L that correspond to $p_{i_1} \cdots p_{i_\ell}$ in the entire text is at least $x\frac{\ell^2}{2}$. This number incorporates possible overlaps. However, since these are m-matches and there are only ℓ different pattern symbols, each text location in T^L can participate in no more than ℓ m-matches.

Conclude that the sum of all numbers in T^L that correspond to $p_{i_1} \cdots p_{i_\ell}$ *without* overlaps is at least $x\frac{\ell^2}{2}\frac{1}{\ell} = x\frac{\ell}{2}$. We have:

$$x\frac{\ell}{2} \leq n$$

$$x \leq \frac{2n}{\ell}.$$

$\square$

Using this lemma and the previous algorithm we achieve our main result.

Theorem 1 *Let P be a pattern of length m and T a text of length n. Real scaled matching can be done in $O(n)$ time.*

Proof: Steps 1-3 of the algorithm clearly take $O(n)$ time. Step 4, the parameterized matching, takes $O(n)$ time because the algorithm in [AFM94] runs in linear time for alphabets that can be sorted in linear time. This is the case for the alphabets of P^L and T^L since the alphabets are the integers bounded by n.

Since by Lemma 4, if we have ℓ different characters there are at most $O(\frac{n}{l})$ m-matches, step 5 takes overall $O(n)$ time. $\square$

6 Further Work

The next step is an intuitively valid definition of real scaled two-dimensional matching and an efficient algorithm for that problem.

Another important variant of the problem presented here is the real scaled indexing problem. Can we efficiently preprocess the text in a manner that will enable, upon inputing a pattern, finding all real scaled occurrences of the pattern in the text in time $O(m + tocc)$, where m is the pattern size and $tocc$ is the number of real scaled occurrences of the pattern in the text.

References

[AC96] A. Amir and G. Calinescu. Alphabet independent and dictionary scaled matching. *7th annual symposium on Combinatorial Pattern Matching (CPM 96)*, pages 320–334, 1996.

[AF95] A. Amir and M. Farach. Efficient 2-dimensional approximate matching of half-rectangular figures. *Information and Computation*, 118(1):1–11, April 1995.

[AFM94] A. Amir, M. Farach, and S. Muthukrishnan. Alphabet dependence in parameterized matching. *Information Processing Letters*, 49:111–115, 1994.

[AL91] A. Amir and G. Landau. Fast parallel and serial multidimensional approximate array matching. *Theoretical Computer Science*, 81:97–115, 1991.

[ALV92] A. Amir, G.M. Landau, and U. Vishkin. Efficient pattern matching with scaling. *Journal of Algorithms*, 13(1):2–32, 1992.

[BM77] R.S. Boyer and J.S. Moore. A fast string searching algorithm. *Comm. ACM*, 20:762–772, 1977.

[ETV88] T. Eilam-Tsoreff and U. Vishkin. Matching patterns in a string subject to multilinear trasformations. *Procedings of the International Workshop on Sequences, Combinatorics, Compression, Security and Transmission, Salerno, Italy*, June 1988.

[FU98] K. Fredriksson and E. Ukkonen. A rotation invariant filter for two-dimensional string matching. In *Proc. 9th Symposium on Combinatorial Pattern Matching (CPM 98)*, pages 118–125. Springer, LNCS 1448, 1998.

[KMP77] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast pattern matching in strings. *SIAM J. Computing*, 6:323–350, 1977.

[KS87] K. Krithivansan and R. Sitalakshmi. Efficient two dimensional pattern matching in the presence of errors. *Information Sciences*, 13:169–184, 1987.

[LV94] G. M. Landau and U. Vishkin. Pattern matching in a digitized image. *Algorithmica*, 12(3/4):375–408, 1994.

[Pen92] A. Pentland. Invited talk. NSF Institutional Infrastructure Workshop, 1992.