Journal of Universal Computer Science, vol. 16, no. 14 (2010), 1902-1911
submitted: 24/4/09, accepted: 4/3/10, appeared: 28/7/10 © *J.UCS*

Geometric Point Pattern Matching in the Knuth–Morris–Pratt Way [1]

Esko Ukkonen
(Department of Computer Science and HIIT, University of Helsinki, Finland
Esko.Ukkonen@cs.Helsinki.Fi)

Abstract: Given finite sets P and T of points in the Euclidean space $\mathbf{R}^d$, the point pattern matching problem studied in this paper is to find all translations $f \in \mathbf{R}^d$ such that $P + f \subseteq T$. A fast search algorithm with some variants is presented for point patterns P that have regular grid–like geometric shape. The algorithm is analogous to the Knuth–Morris–Pratt algorithm of string matching. The time requirement of the search is $O(r|T|)$ where r is the grid dimension of P. Pattern P has grid dimension $r = 1$ if it consists of evenly spaced points on a line. In general, a pattern P is an r–dimensional grid if it has for some $p \in P$ and $e_1, \ldots, e_r \in \mathbf{R}^d$ and positive integers $m_1, \ldots, m_r$ a representation $P = \{p + i_1 e_1 + \cdots + i_r e_r \,|\, 0 \le i_j \le m_j\}$ where the i_j's are integers. Both P and T are given to the search algorithm in the lexicographic order.
Key Words: pattern matching, point sets, translation, Knuth–Morris–Pratt algorithm
Category: F.2, I.3, I.3, I.7

1 Introduction

Finding the translates of a point set within another point set is a basic geometric pattern matching problem. Given point sets P (the pattern) and T (the background) in an Euclidean space, one has to find all translation vectors f such that $P + f \subseteq T$, where $P + f = \{p + f \mid p \in P\}$ is the pattern P translated by f. Typically the pattern is (much) smaller than the background. Other variants of the point pattern matching problem allow transformations other than the translation and may allow partial and/or approximate matches; see, e.g., [Alt and Guibas, 1999, Bishnu, Das, Nandy and Bhattacharya, 2006], [Alt, Mehlhorn, Wagener and Welzl, 1988, Atkinson, 1997, Brass, 2002], [Efrat and Itai, 1996, Chew and Kedem, 1992, de Rezende and Lee, 1995].

The point pattern matching problem under translations is a relevant model for example in music information retrieval. The standard writing of music as notes can be represented as point sets in two-dimensional space. In this representation, a piece of music is given as points (x, y) where x gives the on-set time and y the pitch of a note. Given two such pieces of music, a larger one ('music database') and a shorter one ('melody'), searching the melody from the databasae actually asks for point pattern matching under translation.

[1] A research supported by the Academy of Finland grant 7523004 (Algorithmic Data Analysis).

As for the solution algorithms of the problem of point pattern matching under translations, review [4] only mentions the trivial solution (systematic trial and error, called the *alignment method* in [Huttenlocher and Ullman, 1990]). Its running time is $O(mn \log n)$ where $m = |P|$, $n = |T|$, and $m \leq n$. This contrasts with the classical string pattern matching problem in which P and T are strings of symbols in some finite alphabet. The string pattern matching (which in fact can be seen as a special case of the geometric point pattern matching under translations) can be solved in $O(m + n)$ time using the well–known Knuth–Morris–Pratt algorithm [Knuth, Morris and Pratt, 1977].

In this paper we concentrate on an 'on–line' version of the point pattern matching problem. We only assume that T is given in the lexicographically sorted order but do not apply any other preprocessing on T. On P, on the other hand, we could apply any preprocessing before the search. In our case, ordering of P into the lexicographic order will suffice for the algorithms to be presented. We will utilize string–matching style technique, based on the lexicographic order, to organize and speed–up the search of matches of point patterns. In fact, the lexicographic order will have here the same role as the linear left–to–right order (as implicitly given by the adjacency of symbols) has in string matching.

The trivial algorithm needs in this framework $O(m)$ traversals of T, and hence its running time becomes $O(nm)$, improving the bound $O(mn \log n)$ mentioned above. However, for patterns whose geometric shape is periodic the number of traversals can be much smaller leading to time bounds of the form $O(kn)$ where k is a parameter quantifying the periodicity of P. We will introduce some simple classes of periodic patterns and give for them search algorithms that utilize the periodicity.

In more detail, our results are as follows. Let $P = \{p_1, p_2, \ldots, p_m\} \subset \mathbf{R}^d$ and $T = \{t_1, t_2, \ldots, t_n\} \subset \mathbf{R}^d$ where the elements of P and T are listed in increasing lexicographic order, and d is fixed.

In Section 2, we recall from [Ukkonen, Lemström and Mäkinen, 2003] the search algorithm (here called the Basic Algorithm) that runs in time $O(mn)$. The algorithm is analogous to the trivial $O(mn)$ string matching algorithm that for two strings (keyword and text) attempts to find the occurrences of the keyword starting successively from each location of the text and comparing the elements of the keyword with the elements of the text from this location on until a mismatch is encountered or an occurrence is found. We use the Basic Algorithm as a building block of our new algorithm for point patterns with small linear dimension.

In Section 3, a pattern P is called a *linear grid* if $m > 1$ and $p_{i+1} - p_i = p_2 - p_1$ for all $i = 1, \ldots, m-1$. We give a simple algorithm that finds the translated occurrences of a linear grid in T in time $O(m + n)$. The algorithm is based on bookkeeping of the occurrences of the initial segments of P in T and expanding them whenever $t+(p_2-p_1)$ is in T for some $t = t_1, t_2, \ldots, t_n$. This is analogous to the Knuth–Morris–Pratt algorithm. A pattern P is called an *r–dimensional grid* if it has for some $p \in P$ and $e_1, \ldots, e_r \in \mathbf{R}^d$ and positive integers $m_1, \ldots, m_r$ a representation $P = \{p+i_1e_1+\cdots+i_re_r \,|\, 0 \leq i_j \leq m_j\}$ where the i_j's are integers. When $r = 1$ we get the linear grids. Generalizing our algorithm for linear grids, we will obtain an algorithm that finds the occurrences of an r–dimensional grid in time $O(rn)$. Finally, a pattern P is said to have *linear dimension* k_L if k_L is the smallest number such that P can be represented as a union of k_L linear grids. We describe an algorithm that finds the occurrences of such patterns in time $O(k_L n)$.

Section 4 concludes the paper by making a note on how the above results generalize to 'colored' P and T that occur in some applications. In this case not only the geometric shape but also the colors associated with the points should match.

Our results utilize the internal periodicity of the pattern to speed up geometric point pattern matching. However, if the points of P are in a general position then there is no periodicity. This suggests our conjecture that the time bound $O(mn)$ could be the best possible in the general case of our problem.

2 The problem and the basic $O(mn)$ solution

We let $P = \{p_1, \ldots, p_m\}$ denote a *pattern* set of m points and $T = \{t_1, \ldots, t_n\}$ a *background* set of n points in the Euclidean space $\mathbf{R}^d$ for some fixed d. The *point pattern matching problem under translations* is to find all $f \in \mathbf{R}^d$ such that $P + f \subseteq T$ where $P + f = \{p + f \,|\, p \in P\}$ is the pattern P translated by f. We are in particular interested in the case $m \ll n$ as it then makes sense to use relatively slow algorithms to analyze P, if this leads to a faster search phase.

We assume that P and T are given in the lexicographic order: $p_1 < p_2 < \ldots < p_m$ and $t_1 < t_2 < \ldots < t_n$. Recall that this order $<$ of points $a = (a_1, \ldots, a_d)$ and $b = (b_1, \ldots, b_d)$ is defined as $a < b$ if and only if there is j such that $a_i = b_i$ for $i < j$ and $a_j < b_j$. To sort P and T, a preprocessing time of $O(dm \log m + dn \log n)$ is needed.

If $P + f \subseteq T$, the translation f is said to give an *occurrence* $P + f$ of P in T. Every point of an occurrence $P + f$ must match some point of T. Therefore $p_1 + f = t_j$ for some $t_j \in T$. This means that to find all occurrences of P in T it suffices to test only the n translations f_j defined for $1 \leq j \leq n$ as

$$f_j = t_j - p_1. \tag{1}$$

For each such f_j, we can check in time $O(m \log n)$ using binary search on the sorted T, that also the remaining points $p_2 + f_j, \ldots, p_m + f_j$ of $P + f_j$ match T. This is the trivial algorithm for finding all translated occurrences of P which is analogous to the brute–force string matching algorithm and has running time $O(mn \log n)$.

In [Ukkonen, Lemström and Mäkinen, 2003], an improved version of the trivial algorithm was given. We briefly describe this algorithm, which we will here call the Basic Algorithm, as we need it in the sequel. The Basic Algorithm is given below in Fig. 1. Function $next(t_j)$ gives t_{j+1}.

for $i \leftarrow 1, \ldots, m$ **do** $s_i \leftarrow -\infty$
$s_{m+1} \leftarrow \infty$
for $j \leftarrow 1, \ldots, n-m$ **do**
    $f \leftarrow t_j - p_1$
    $i \leftarrow 1$
    do
        $i \leftarrow i + 1$
        $s_i \leftarrow \max(s_i, t_j)$
        while $s_i < p_i + f$ **do** $s_i \leftarrow next(s_i)$
    until $s_i > p_i + f$
    if $i = m + 1$ **then** $output(f)$
end for.

Figure 1: Basic Algorithm [Ukkonen, Lemström and Mäkinen, 2003].

Basic Algorithm traverses T and matches p_1 against t_1, t_2, and so on until t_n. The traversal is implemented using a pointer s_1. When $s_1 = t_j$ during the traversal, the algorithm has to check whether or not the translation $s_1 - p_1 = t_j - p_1$, i.e., the translation f_j of (1), would give an occurrence of P. To this end the algorithm maintains for each element p_i of P a pointer s_i that also traverses T, in cascade with s_1. When s_i points to t_h, it in effect represents the translation $t_h - p_i$. The algorithm compares this translation to the current f_j which is represented as $s_1 - p_1$. If it is smaller than f_j, s_i is updated to t_{h+1}. Now, after the update, s_i represents the translation $t_{h+1} - p_i$ which is larger than $t_h - p_i$ as $t_{h+1} > t_h$ (but $t_{h+1} - p_i < t_s - p_i$ for any $s > h + 1$). If a translation equal to f_j is found after some updates of s_i in this way through T, the algorithm actually has found a match for $p_i + f_j$. The algorithm then continues with testing and updating the next pointer s_{i+1}, and so on. If finally s_m hits an element t of T such that $t_h - p_m = f_j$, the algorithm has verified the entire occurrence $P + f_j$.

The other possibility is that the translation represented by some s_i grows larger than f_j without matching with it. Then the search fails for the present s_1. In both cases s_1 is next updated to t_{j+1} and the search is repeated to verify $P + f_{j+1}$, and so on. The lexicographic traversal order insures that no backtracking of any pointer s_i is needed during the search as the target translation f_j is monotonically increasing when the search goes on. Hence Basic Algorithm actually performs m one–way traversals of T, giving running time of $O(mn)$. The algorithm can be much faster as its best case running time is $O(n)$. This is the case for points in 'general position' meaning that the intersection $\{p_{i+1} - p_i|i\} \cap \{t_j - t_i|j > i\}$ of relevant pairwise distances within P and T is small and hence Basic Algorithm runs fast.

3 Grid–like patterns

3.1 Linear grids

We start by describing the general principle of our search algorithm for linear grids. The idea is to consider *simultaneously* all potential occurrences of P, that may contain t_j, and to check which of these occurrences can be successfully expanded to the next element of P.

Let us denote by $P_i = \{p_1, \ldots, p_i\}$ the ith *initial segment* of P. Let $\mathit{Alive}(t_j) = \{i \mid P_i + (t_j - p_i) \subseteq T\}$ denote all initial segments that are 'alive' at t_j in the sense that the translation $t_j - p_i$ that makes p_i match t_j also makes the rest of the initial segment P_i match T. If set $\mathit{Alive}(t_j)$ is available at every t_j, then the occurrences of P can be found: there is an occurrence ending at t_j if and only if $m \in \mathit{Alive}(t_j)$.

The sets $\mathit{Alive}(t_j)$ will be constructed incrementally. When the search enters t_j, set $\mathit{Alive}(t_j)$ has already been accumulated, as a side–effect of visiting the earlier elements of T.

At t_j, if $m \in \mathit{Alive}(t_j)$ then we have found an occurrence of P. Moreover, for each $i \in \mathit{Alive}(t_j)$ such that $i < m$, we have to check if the occurrence of the initial segment P_i can be expanded to an occurrence of P_{i+1}. Therefore we check whether or not the point $t = t_j + (p_{i+1} - p_i)$ is in T. If t actually is in T, then t must equal t_{j+h} for some $h > 0$ because $p_{i+1} - p_i > 0$ and hence t must occur after t_j in T. So we scan T beyond t_j and if such a t_{j+h} is found, then $i + 1$ is added to $\mathit{Alive}(t_{j+h})$.

The updating of the sets *Alive* can be made more efficient by utilizing the periodicities in P. If $p_{i+1} - p_i$ is the same for several different $i \in \mathit{Alive}(t_j)$ then $i + 1$ for all these i's should be added to the same set $\mathit{Alive}(t_{j+h})$. We will do such an update in one step for each different $\Delta = p_{i+1} - p_i$. We need for each Δ a pointer that traverses T. If P has k different values Δ, we end up with an $O(kn)$ algorithm. This requires that the updates for each Δ be done in constant time at each t_j. A precomputed transition table is needed, that gives for each possible set *Alive* what are the k updates to be propagated with each different Δ.

We concentrate on the simple special case $k = 1$. We call the corresponding patterns linear grids.

Definition 1. A pattern $P = (p_1 < p_2 < \ldots < p_m)$ is a *linear grid* if $m > 1$ and $p_{i+1} - p_i = p_2 - p_1$ for all $i = 1, 2, \ldots, m - 1$. The difference $p_2 - p_1$ between the adjacent points of a linear grid P is called the *displacement*, point p_1 the *starting point*, and point p_m the *end point* of the linear grid.

Note that the classical string matching problem of searching a key–word from a text deals with patterns and backgrounds whose spatial geometric structure is a linear grid.

Obviously, for a linear grid there is only one Δ and hence $k = 1$. It is also immediate that whenever in this case $i \in Alive(t_j)$ then also all i', $i' < i$, are members of $Alive(t_j)$. But then we can use a simplified constant size representation of *Alive*, namely represent each $Alive(t_j)$ by its largest element. The test for an occurrence at t_j becomes testing whether or not $Alive(t_j) = m$. The search algorithm is as follows.

Algorithm 1. Let P be a linear grid of m elements and let $\Delta = p_2 - p_1$ and let initially $Alive(t_j) = 1$ for all t_j. Function $next(t_j)$ gives t_{j+1}, and we let $t_0 = -\infty$.

$t \leftarrow s \leftarrow -\infty$
while $t < t_n$ **do**
    $t \leftarrow next(t)$
    if $Alive(t) = m$ **then** P occurs ending at t
    while $s < t + \Delta$ **do** $s \leftarrow next(s)$
    if $s = t + \Delta$ **then** $Alive(s) \leftarrow Alive(t) + 1$

Analysis. The correctness of the algorithm should be evident from the above discussion. The running time consists of scanning T with t in $O(n)$ steps and with s in another $O(n)$ steps. That a sorted P is a linear grid can be tested in time $O(m)$. We have obtained:

Theorem 2. *Algorithm 1 finds the translated occurrences of a linear grid P in a lexicographically sorted T in time $O(n)$ where $n = |T|$.*

3.2 Multi–dimensional grids

A pattern P is an *r–dimensional grid* if it has for some $p \in P$ and $e_1, \ldots, e_r \in \mathbf{R}^d$, each $e_i > 0$, and positive integers $m_1, \ldots, m_r$ a representation

$$P = \{p + i_1 e_1 + \cdots + i_r e_r \mid i_1 = 0, 1, \ldots, m_1; \ldots; i_r = 0, 1, \ldots, m_r\}. \quad (2)$$

Note that the grid dimension r is independent of the dimension d of the underlying Euclidean space. Even for $d = 1$, the grid dimension of P can have any integer value.

Such a P consists of linear grids of length m_r with displacement $\Delta = e_r$. The starting points of these grids can be grouped into linear grids of length m_{r-1} with $\Delta = e_{r-1}$. The starting points of these grids can again be grouped into linear grids, and so on, until the last phase will give one linear grid of length m_1 with displacement $\Delta = e_1$.

This suggests that the occurrences of P can be found by repeatedly applying Algorithm 1: First find and mark in T with Algorithm 1 all starting points of the occurrences of the linear grid with $\Delta = e_r$ and length m_r. This takes time $O(n)$. Then apply Algorithm 1 on these points (they constitute a lexicographically ordered subset of T) to find all starting points of the occurrences of the linear grid with $\Delta = e_{r-1}$ and length m_{r-1}, and so on. Finally the starting points of the occurrences of the linear grid with $\Delta = e_1$ and length m_1 will give the occurrences of entire P. If t is any such point in T, the translation $t - p$ where p is the smallest point of P, gives the occurrence: $P + (t - p) \subseteq T$. We call this method Algorithm 2.

Theorem 3. *Algorithm 2 finds the translated occurrences of an r–dimensional grid P in a lexicographically ordered T in time $O(rn)$ where $n = |T|$.*

That P is an r–dimensional grid for some r can be tested in polynomial time as follows. We start with the following characterization of r–dimensional grids; the proof by induction is almost immediate.

Lemma 4. *Pattern P is an r–dimensional grid if and only if P is a union of equally long disjoint linear grids that have the same displacement and the set of the starting points of these linear grids is either a singleton set (in which case P is a linear grid) or an $(r-1)$–dimensional grid.*

Lemma 1 gives a test whether or not P is an r–dimensional grid for some r: we need to find out whether or not P is a union of equally long linear grids with the same displacement, and if it is, then continue testing recursively on the set of the starting points of the linear grids. To this end, we find all possible displacements

$$D = \{p_i - p_j \mid p_i, p_j \in P, i > j\}$$

in time $O(m^2)$. Then we test for each $u \in D$ whether u covers P in the sense that each $p_i \in P$ has a $p_j \in P$ such that $p_i - p_j = u$ or $p_j - p_i = u$. This test can be conveniently done in time $O(m^2)$ with the Basic Algorithm. If t covers P, then P obviously is a union of disjoint linear grids with displacement u. Testing that the linear grids are of equal length and finding their starting points still takes time $O(m)$. As there are $O(m^2)$ different possible displacements u to test, the total time becomes $O(m^4)$.

If there are more than one starting point, the test is then recursively continued on the set of starting points. Pattern P is an r–dimensional grid if the test finds on the rth recursive round that the starting point set given to that round is a linear grid. The displacements from different rounds are the base vectors e_i of the grid representation (2) of P, the lexicographically smallest point of P being the base point p of the representation. As the number of the starting points given to the next recursive call is at most half of the current number, the total number of recursive calls is $O(\log m)$ and $r \leq \log_2 m$. Hence the running time of our algorithm is $O(m^4 \log m)$.

3.3 Patterns with small linear dimension

The *linear dimension* of a pattern P is the smallest number k_L such that P can be represented as a union of k_L linear grids.

The occurrences of P in T can be found by using first Algorithm 1 k_L times to find in T all starting points of the occurrences of the k_L linear grids and using then Basic Algorithm to find where in T the starting points form the same subpattern of size k_L as in P. The resulting algorithm has running time $O(k_L n)$.

Finding k_L and the corresponding decomposition of P seems hard. However, we can easily find all linear grids in P. These form a collection of subsets of P whose union equals P. The size of the smallest subcollection that covers P is $= k_L$. Hence we may use the greedy set cover approximation algorithm to find in polynomial time a covering subcollection, giving a suboptimal linear dimension $\leq k_L \log m$.

Theorem 5. *Given a pattern P of linear dimension k_L with its decomposition into the k_L linear grids, there is an algorithm that finds the translated occurrences of P in a lexicographically ordered T in time $O(k_L n)$ where $n = |T|$. A suboptimal linear dimension within a logarithmic factor from the optimum can be found in polynomial time; the resulting search algorithm runs in total time $O(poly(m) + k_L n \log m)$.*

4 Concluding remark: colored P and T

In some applications it is not enough that the points match spatially but there can be additional requirements for acceptable matches. In the case of music retrieval, for example, the notes have duration in addition to the on-set time and pitch. To model this situation we use 'colored' P and T. Then the elements of P and T are of the form (c, u) such that $u \in \mathbf{R}^d$ and $c \in A$ where A is a finite set of the color values. Colored pattern $P = ((c_1, p_1), \ldots, (c_m, p_m))$ has an occurrence in a colored T if $\{(c_1, p_1 + t), \ldots, (c_m, p_m + t)\} \subseteq T$ for some $t \in \mathbf{R}^d$. For the colored case we mention here some basic results but leave the details to the interested reader.

It is not difficult to see that all our algorithms generalize to the colored case, with the same time bounds. The only point needing some care is the generalization of Algorithm 1 such that the $O(n)$ time bound is retained. The Knuth–Morris–Pratt algorithm for string pattern $c_1 c_2 \cdots c_m$ can be used on top of Algorithm 1 to check in linear time that also the color components match whenever Algorithm 1 has found an occurrence of the geometric component. For r–dimensional grids (Algorithm 2) the Aho–Corasick multipattern string matching machine [Aho and Corasick, 1975] can be used for the same purpose, analogously to the Baker–Bird algorithm [Baker, 1978, Bird, 1977] for two–dimensional string matching.

References

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

[Alt and Guibas, 1999] Alt H., Guibas L.: Discrete geometric shapes: Matching, interpolation, and approximation. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 121-153. Elsevier Science Publishers B.V. North-Holland 1999

[Alt, Mehlhorn, Wagener and Welzl, 1988] Alt H., Mehlhorn K., Wagener H., Welzl E.: Congruence, similarity and symmetries of geometric objects. *Discrete Comput. Geom.*, 3:237-256, 1988.

[Atkinson, 1997] Atkinson M.D.: An optimal algorithm for geometric congruence. *J. Algorithms*, 8:159–172, 1997.

[Baker, 1978] Baker T.P.: A technique for extending rapid exact–match string matching to arrays of more than one dimension. *SIAM J. Comput.*, 7 (1978), 533–541.

[Bird, 1977] Bird R.S.: Two–dimensional pattern matching *Inf. Proc. Lett.*, 6 (1977), 168–170.

[Bishnu, Das, Nandy and Bhattacharya, 2006] Bishnu A., Das S., Nandy S.C., Bhattacharya B.B.: Simple algorithms for partial point set pattern matching under rigid motion. *Pattern Recognition* 39, 9 (2006), 1662-1671.

[Brass, 2002] Brass P.: Combinatorial geometry problems in pattern recognition. *Discrete Comput. Geom.* 28 (2002), 495–510.

[Chew and Kedem, 1992] Chew L.P., Kedem, K.: Improvements on geometric pattern matching problems. In *Proceedings of the Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 318-325, 1992.

[Efrat and Itai, 1996] Efrat A., Itai A.: Improvements on bottleneck matching and related problems using geometry. In *Proc. of the Twelfth Ann. Symp. on Computational Geometry*, pages 301-310. ACM Press, 1996.

[Huttenlocher and Ullman, 1990] Huttenlocher D., Ullman S.: Recognizing solid objects by alignment with an image. *Intern. J. Computer Vision.* 5:195–212, 1990.

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

[de Rezende and Lee, 1995] de Rezende P.J., Lee D.T.: Point Set Pattern Matching in d-Dimensions. *Algorithmica* 13 (1995), 387-404.

[Ukkonen, Lemström and Mäkinen, 2003] Ukkonen E., Lemström K., Mäkinen V.: Sweepline the Music! In: *Computer Science in Perspective, Essays dedicated to Th. Ottmann*, Springer Lect. Notes in Computer Science, vol. 2598, pp. 330–342, 2003.