DOI: http://dx.doi.org/10.14236/ewic/FDIA2015.9

Two-dimensional point set pattern matching with horizontal scaling

Antti Laaksonen
Department of Computer Science
University of Helsinki
ahslaaks@cs.helsinki.fi

This paper focuses on two-dimensional point set pattern matching with horizontal scaling. Given a dataset S of n points and a pattern P of m points, the task is to find occurrences of P in S. The pattern may be horizontally scaled using a constant value. This problem is relevant in symbolic music information retrieval when each point is interpreted as a musical note and the pattern is a melody that is searched for. The best known general algorithm for the problem works in $O(n^2 m)$ time. In this paper we show that the 3SUM problem can be reduced to this problem, and present a new algorithm that works in $O(n^2)$ time on typical inputs.

Keywords: Music information retrieval, Geometric algorithm, Point set pattern matching, Horizontal scaling, 3SUM problem

1. INTRODUCTION

A natural problem in symbolic music information retrieval is to search for occurrences of a melody in a musical score. This problem has a geometric interpretation where each musical note is a point in the plane. Using this interpretation, the x coordinate of a point corresponds to the onset time of a note and the y coordinate corresponds to the pitch. In addition, horizontal scaling of the pattern should be allowed to adapt to tempo differences between the melody in the query and in the occurrences.

Let us now define the problem more formally. The *dataset* is an array of n points $S[1], S[2], \ldots, S[n]$, and the *pattern* is an array of m points $P[1], P[2], \ldots, P[m]$. Each point is a pair (x, y) where x and y are real number coordinates. We use a dot syntax for accessing the values. For example, $P[3].x$ is the x coordinate of the third point.

Given two points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, we define their order lexicographically, i.e., $p_1 \leq p_2$ if either $x_1 < x_2$ or $x_1 = x_2$ and $y_1 \leq y_2$. Throughout the paper, we assume that S and P are already sorted in the lexicographic order. Furthermore, we use the notations $p_1 + p_2 = (x_1 + x_2, y_1 + y_2)$ and $p_1 - p_2 = (x_1 - x_2, y_1 - y_2)$.

Our goal is to find the indices in the point set from which a *occurrence* of the pattern begins. An occurrence consists of m indices $1 \leq \boldsymbol{i}_1 < \boldsymbol{i}_2 < \ldots < \boldsymbol{i}_m \leq n$ and fulfils the following conditions. First, there is a *scaling factor* $\alpha > 0$ such that $S[\boldsymbol{i}_{k+1}].x - S[\boldsymbol{i}_k].x = \alpha(P[k+1].x - P[k].x)$ for each $k = 1, \ldots, m-1$. Second, we require that $S[\boldsymbol{i}_{k+1}].y - S[\boldsymbol{i}_k].y = P[k+1].y - P[k].y$ for each $k = 1, \ldots, m-1$.

There are two major difficulties in the problem. First, there may be gaps between occurrence points in the dataset. For this reason, standard string matching algorithms cannot be used for solving the problem. Second, partial pattern occurrences with different scaling factors cannot be combined, which makes it difficult to design any dynamic programming style algorithm for the problem.

The problem was first discussed by Romming and Selfridge-Field (2007) in the context of partial pattern matching using geometric hashing. Lemström (2010) defined the problem similar to this paper and gave an $O(n^2 m \log n)$ time algorithm. A more efficient algorithm with time complexity $O(n^2 m)$ was proposed by Laaksonen (2013). A related problem to the problem is one-dimensional point set pattern matching with scaling, where the difference is that the y coordinate of each point is constant. This problem can also be solved in $O(n^2 m)$ time, as shown by Rezende and Lee (1995).

The structure of the rest of the paper is as follows: In Section 2, we show how the 3SUM problem can be reduced to this problem. In Section 3, we present a new algorithm that solves the problem in $O(n^2)$ time under certain assumptions. Finally, in Section 4, we present our conclusions.

2. 3SUM REDUCTION

The 3SUM problem can be defined as follows: Given an array A of integers, is it possible to choose three integers $a, b, c \in A$ such that $a+b+c = 0$? Gajentaan and Overmars (1995) noticed that many problems in computational geometry are at least as difficult as 3SUM. Next we show that two-dimensional point set pattern matching with horizontal scaling also belongs to this group of problems.

The 3SUM problem can be easily solved in $O(n^2)$ time, and it was conjectured for a long time that no $o(n^2)$ time solution exists. However, a recent paper by Grønlund and Pettie (2014) refutes the conjecture by presenting an algorithm that solves 3SUM slightly faster, in $O(n^2(\log\log n)^{5/3}/(\log n)^{2/3})$ time.

We can solve 3SUM in $O(n \log n + \boldsymbol{f}(n))$ time assuming that we can use a subroutine X that solves two-dimensional point set pattern matching with horizontal scaling in $\boldsymbol{f}(n)$ time. First we sort the numbers in A in $O(n \log n)$ time. After this, we check if there is a solution where $a = b$, $a = c$ or $b = c$. For every number $k \in A$, we check if A also contains $-2k$. This can be done in $O(n \log n)$ time using binary search. Another special case is the trivial solution $a = b = c = 0$ when A contains 0.

From this point on, if a solution a, b, c exists, the numbers are distinct and nonzero. There are two subproblems: either two numbers are positive and one is negative, or two numbers are negative and one is positive. We solve the subproblems separately using our subroutine X. For the first subproblem we construct set S_1, and for the second subproblem we construct set S_2. Pattern P always consists of three points: (1, 1), (2, 2) and (3, 3).

Then for each number $k \in A$ we add one or two points to the datasets. If $k > 0$, we add points $(k, 1)$ and $(k, 3)$ to S_1 and point $(k/2, 2)$ to S_2. If $k < 0$, we add points $(-k, 1)$ and $(-k, 3)$ to S_2 and point $(-k/2, 2)$ to S_1. Now, P can be found in S_1 or S_2 exactly when there are three distinct numbers $a, b, c \in A$ such that $a + b + c = 0$.

For example, suppose that $A = [3, 5, -7, 4, -2]$. Now S_1 consists of points (1, 2), (3, 1), (3, 3), (3.5, 2), (4, 1), (4, 3), (5, 1) and (5, 3). Figure 1 shows the points in S_1 and an occurrence of P that consists of points (3, 1), (3.5, 2) and (4, 3) with scaling factor 0.5. This occurrence corresponds to the solution $a = 3, b = -7, c = 4$ for the 3SUM problem.

Next we prove the construction in the general case where a, b and c are distinct nonzero numbers.

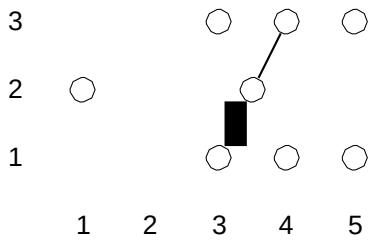

Figure 1: *An instance of the construction used in the 3SUM reduction. Each circle is a dataset point, and a pattern occurrence is shown.*

First, assume that $a + b + c = 0$, a and b are positive, $a < b$, and c is negative. Now set S_1 contains points $(a, 1)$, $(b, 3)$ and $(-c/2, 2) = ((a + b)/2, 2)$, so pattern P appears in S_1 with scaling factor $(b - a)/2$. Correspondingly, if a and b are negative and c is positive, set S_2 contains points $(-a, 1)$, $(-b, 3)$ and $(c/2, 2) = ((-a - b)/2, 2)$.

Then, assume that S_1 contains an occurrence of P. The occurrence consists of points $(z + \alpha, 1)$, $(z + 2\alpha, 2)$ and $(z + 3\alpha, 3)$. Thus, A contains values $z+\alpha$, $z+3\alpha$ and $-2(z+2\alpha)$, and the sum of the values is 0. Correspondingly, if S_2 contains an occurrence of P, A contains values $-(z + \alpha)$, $-(z + 3\alpha)$ and $2(z + 2\alpha)$, again with sum 0.

3. ALGORITHM

In this section we present an algorithm that solves two-dimensional point set pattern matching with horizontal scaling in $O(n^2)$ time under two assumptions. First, we assume that all coordinates of the points are integers. Second, we assume that there is a contant c such that the horizontal distance between any two consecutive points in the dataset is at most c.

The idea in the algorithm is to limit the number of possible scaling factors that need to be checked. For a fixed scaling factor, all pattern occurrences can be found in $O(nm)$ time using a technique presented by Ukkonen (2003). We show that under the above assumptions, there are only $O(n/m)$ possible scaling factors, and using this fact we can construct an algorithm that solves the problem in $O(n^2)$ time.

Listing 1 shows the structure of the algorithm. Note that for simplicity in the presentation, we assume that each pattern point has a distinct x coordinate. However, the algorithm could be modified to handle general patterns by grouping pattern points that have equal x coordinates.

Algorithm 1

```
 1: e ← ∞
 2: for k ← 2, . . . , m do
 3:     e ← min(e, P[k].x − P[k − 1].x)
 4: end for
 5: for z ← 1, . . . , S[n].x − S[1].x do
 6:     α ← z/e
 7:     if α(P[m].x − P[1].x) ≤ S[n].x − S[1].x then
 8:         P′ ← P
 9:         for k ← 2, . . . , m do
10:             P′[k].x ← P′[k − 1].x + α(P[k].x − P[k − 1].x)
11:             Q[k] ← 1
12:         end for
13:         for i ← 1, . . . , n do
14:             c ← 1
15:             for k ← 2, . . . , m do
16:                 while Q[k] < n and
17:                         S[Q[k]] − S[i] < P′[k] − P′[1]
    do
18:                     Q[k] ← Q[k] + 1
19:                 end while
20:                 if S[Q[k]] − S[i] = P′[k] − P′[1] then c ← c + 1
21:             end for
22:             if c = m then print(i)
23:         end for
24:     end if
25: end for
```

3.1. Analysis

First, on lines 1–4, the algorithm calculates the minimum horizontal distance between two consecutive points in the dataset. Then, on lines 5–25, the algorithm goes through all possible scaling factors. After selecting the scaling factor, on lines 8–23, the algorithm uses a technique similar to Ukkonen (2003) for searching for the pattern occurrences. Next we will focus on the condition on line 7 that limits the number of scaling factors to be checked.

The algorithm is based on the following observation: in any pattern occurrence the scaled horizontal distance between some two consecutive pattern points is at most $cn/(m-1)$. The reason for this is that the horizontal distance between the first and last dataset point is at most cn. Therefore there has to be two points in the pattern whose scaled horizontal distance is at most $cn/(m-1)$, because otherwise the scaled horizontal distance between the first and the last pattern point would be more than cn.

To calculate the number of the possible scaling factors we use the fact that all coordinates are integers. Thus, the minimum scaled horizontal distance between two pattern points is an integer between 1 and $\lfloor cn/(m-1) \rfloor$. The time complexity of the algorithm is $O(n^2)$ because there are $O(n/m)$ possible scaling factors to check and each check works in $O(nm)$ time.

3.2. General case

It would be tempting to try to limit the number of possible scaling factors also in the general case. However, the following construction shows that if there can be arbitrary gaps between dataset points (thus a constant c is not involved), there can be $\Theta(n)$ pattern occurrences with distinct scaling factors. Given integers n and m, the construction produces an instance of the problem with $n^t \geq n$ dataset points, exactly m pattern points, and $\Theta(n^t)$ pattern occurrences with distinct scaling factors.

The database consists of q point groups, each having a total of $2m-1$ points. The groups are numbered $1, \ldots, q$. The value q is selected so that it is the minimum number for which $n^t = q(2m-1) \geq n$. Each note in group x is a pair $(p(x)t(y), x)$ where $y = 1, \ldots, 2m-1$; $p(k)$ is the k'th prime number, $t(1) = 0$ and $t(k) = 2^{k-2}$ if $k > 1$. The pattern consists of pairs $(t(x), 1)$ where $x = 1, \ldots, m$.

For example, suppose that $n = 20$ and $m = 4$. In this case $2m - 1 = 7$, $q = 3$ and $n^t = 21$. The dataset consists of three groups of points:

1. (0, 1), (2, 1), (4, 1), (8, 1), (16, 1), (32, 1), (64, 1)
2. (0, 2), (3, 2), (6, 2), (12, 2), (24, 2), (48, 2), (96, 2)
3. (0, 3), (5, 3), (10, 3), (20, 3), (40, 3), (80, 3), (160, 3)

The pattern is (0, 1), (1, 1), (2, 1), (4, 1) and there are 4 pattern occurrences in each group. In group 1 the scaling factors are 2, 4, 8 and 16, in group 2 they are 3, 6, 12 and 24, and in group 3 they are 5, 10, 20 and 40.

In the general case, in each group there are m pattern occurrences. The total number of pattern occurrences with distinct scaling factors is qm, and $n^t = q(2m-1) < 2qm$, thus the number of scaling factors is $\Theta(n^t)$.

4. CONCLUSIONS

In this paper we showed that the 3SUM problem can be reduced to two-dimensional point set pattern matching with horizontal scaling. This suggests that it is unlikely that the problem could be solved significantly more efficiently than in $O(n^2)$ time.

In addition, we presented a new algorithm for the problem that works in $O(n^2)$ time. The algorithm assumes that all coordinates in the input are integers and there is a constant that limits the distance between consecutive points in the dataset. These assumptions hold in many applications. For example, in music information retrieval, note onset times can

be seen as integers and there cannot be long rests between consecutive notes.

It would be interesting to know if the problem could be solved in $o(n^2 m)$ time in the general case. The one-dimensional point set pattern matching problem could be a good starting point because it captures the special case where all y coordinates have a constant value.

5. ACKNOWLEDGEMENTS

The participation of the author in the ESSIR 2015 summer school and in the FDIA 2015 symposium has been supported by the Helsinki Doctoral Programme in Computer Science and the ELIAS ESF Research Networking Programme.

REFERENCES

Gajentaan, A. and Overmars, M. "On a class of $O(n^2)$ problems in computational geometry," *Computational Geometry*, 5(3), pp. 165–185, 1995.

Grønlund, A. and Pettie, S.: "Threesomes, degenerates, and Love Triangles," available at http://arxiv.org/abs/1404.0799, 2014

Laaksonen, A.: "Efficient and simple algorithms for time-scaled and time-warped music search," *CMMR 2013*

Lemström, K.: "Towards more robust geometric content-based music retrieval," *ISMIR 2010*

Rezende, P. and Lee, D.: "Point set pattern matching in d-dimensions," *Algorithmica*, 13(4) pp. 387–404, 1995

Romming, C. and Selfridge-Field, E.: "Algorithms for polyphonic music retrieval: the Hausdorff metric and geometric hashing," *ISMIR 2007*

Ukkonen, E., Lemström, K. and Mäkinen, V.: "Geometric algorithms for transposition invariant content-based music retrieval," *ISMIR 2003*