Finding Similar Movements in Positional Data Streams

Jens Haase[†] and Ulf Brefeld[‡]

Knowledge Mining & Assessment Group
Technische Universität Darmstadt, Germany

[†] je.haase@gmail.com
[‡] brefeld@kma.informatik.tu-darmstadt.de

Abstract. In this paper, we study the problem of efficiently finding similar movements in positional data streams, given a query trajectory. Our approach is based on a translation-, rotation-, and scale-invariant representation of movements. Near-neighbours given a query trajectory are then efficiently computed using dynamic time warping and locality sensitive hashing. Empirically, we show the efficiency and accuracy of our approach on positional data streams recorded from a real soccer game.

1 Introduction

Team sports has become a major business in many parts of the world. Clubs spend a great deal of money on players, training facilities and other ways to further improve their play. For instance, the German Bundesliga records all games with special cameras, capturing bird's eye views of the pitch, to better analyse player movements and tactics. The recordings capture positions of the players and the ball for every fraction of a second. While simple analyses, such as the overall distance a player covered, heat maps of player positions, etc., can be computed (semi-)automatically, more complex analyses involving tactics and counter-strategies rely on human experts. However, the sheer existence of such data paves the way for automatic analyses using intelligent mining techniques. In this paper, we study efficient techniques for detecting similar movements in positional data streams to provide a basis for the analyses of frequent movements and tactical patterns.

For a soccer game taking about 90 minutes, the recorded data translate into a positional data stream. Standard recording rates of 25 frames per second lead to a representation by about 135,000 snapshots. Every snapshot consists of the 23 positions of the players of the two teams and the ball. In sum, the game is described by more than three million coordinates. As player movements are sequences of such coordinates, it is clear that there are a great deal of comparisons necessary to account for different lengths of such sequences across

players. Thus, there is a real need for efficient techniques to further process and analyse the data.

In this paper, we study the problem of finding similar movements of players in such positional data streams. The problem is challenging for two reasons. Firstly, it is not clear how to define an appropriate similarity measure on player trajectories and secondly, the sheer number of coordinates render exact approaches infeasible as we will show in the experiments. We first propose a translation-, rotation-, and scale-invariant representation of movements using Angle/Arc-Lengths [11]. Second, we investigate efficient near-neighbour routines based on dynamic time warping [9] and locality sensitive hashing [2]. Our empirical results on positional data recorded from a real soccer game show the efficiency and accuracy of our approach compared to exact baseline methods.

The remainder is structured as follows. Section 2 briefly reviews related work. Our main contribution is presented in Section 3 and we report on empirical results in Section 4. Section 5 concludes.

2 Related Work

Prior work on mining positional data streams mostly focuses on the performance of individual players. Kang et al. [4] present an approach that uses positional data to assess player positions in particular areas of the pitch, such as catchable, safe or competing zones. Grunz et al. [3] analyse groups of players and their behaviour using self organising maps on positional data. Every neuron of the network represents a certain area of the pitch. Thus, whenever a player moves into such an area, the respective neuron is activated. Perše at al. [8] use positional data of basketball games to compare movements of players with respect to tactical patterns, e.g., a player blocks space for his teammate. The presented approach however does not detect novel movements that deviate from the already known patterns. By contrast, we study a purely data-driven approach to find similar movements in positional data for a given query trajectory without making any assumptions on zones, tasks, or movements.

3 Contribution

3.1 Preliminaries

For each player, we are given a positional data stream $\mathcal{P} = \langle \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots \rangle$ where $\boldsymbol{x}_t = (x_1, x_2)^\top$ denotes the coordinates of the players position on the pitch at time t. A trajectory or movement of the player is a subset $\boldsymbol{p} \subset \mathcal{P}$ of the stream, e.g., $\boldsymbol{p} = \langle \boldsymbol{x}_t, \boldsymbol{x}_{t+1}, \ldots, \boldsymbol{x}_{t+m} \rangle$, where m is the length of the trajectory. A game

$\mathcal{D}$ is thus given by the union of all trajectories of length m of the two teams. For simplicity, we omit the time index t and simply index elements of a trajectory by their offset $1, \ldots, m$ in the remainder. The goal of this paper is to accurately and efficiently compute similarities between trajectories in $\mathcal{D}$. That is, given a query trajectory $\boldsymbol{q}$, we aim at finding the N most similar trajectories in $\mathcal{D}$.

3.2 Representation

We aim to exploiting the symmetries of the pitch and use Angle/Arc-Length (AAL) [11] transformations to guarantee *translation*, *rotation*, and *scale* invariant representations of trajectories. The main idea of AAL is to represent a movement $\boldsymbol{p} = \langle \boldsymbol{x}_1, \ldots, \boldsymbol{x}_m \rangle$ in terms of distances and angles

$$\boldsymbol{p} \mapsto \bar{\boldsymbol{p}} = \langle (\alpha_1, \|\boldsymbol{v}_1\|), \ldots, (\alpha_m, \|\boldsymbol{v}_m\|) \rangle, \tag{1}$$

where $\boldsymbol{v}_i = \boldsymbol{x}_i - \boldsymbol{x}_{i-1}$. The difference $\boldsymbol{v}_i$ is called the *movement vector* at time i and the corresponding angle with respect to a reference vector $\boldsymbol{v}_{ref} = (1, 0)^\top$ is defined as

$$\alpha_i = sign(\boldsymbol{v}_i, \boldsymbol{v}_{ref}) \left[cos^{-1} \left(\frac{\boldsymbol{v}_i^\top \boldsymbol{v}_{ref}}{\|\boldsymbol{v}_i\| \, \|\boldsymbol{v}_{ref}\|} \right) \right],$$

where the $sign$ function computes the direction (clockwise ore counterclockwise) of the movement with respect to the reference. In the remainder, we discard the norms in Equation (1) and represent trajectories by their sequences of angles, $\boldsymbol{p} \mapsto \tilde{\boldsymbol{p}} = \langle \alpha_1, \ldots, \alpha_m \rangle$.

3.3 Dynamic Time Warping

In this section, we propose a distance measure for trajectories. The proposed representation of the previous section fulfils the required invariance in terms of translation, rotation and scaling [11]. However, some movements may start slow and end fast, while others start fast and then slow down at the end. Thus, we additionally need to compensate for phase shifts of trajectories. A remedy comes from the area of speech recognition and is called dynamic time warping (DTW) [9]. Given two sequences $\boldsymbol{s} = \langle s_1, \ldots, s_m \rangle$ and $\boldsymbol{q} = \langle q_1, \ldots, q_m \rangle$ and an element-wise distance function $dist : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ (e.g., Euclidean distance), we define the DTW function g recursively as follows

$$\begin{aligned} g(\emptyset, \emptyset) &= 0 \\ g(\boldsymbol{s}, \emptyset) &= dist(\emptyset, \boldsymbol{q}) = \infty \\ g(\boldsymbol{s}, \boldsymbol{q}) &= dist(s_1, q_1) + min \left\{ \begin{array}{l} g(\boldsymbol{s}, \langle q_2, \ldots, q_m \rangle) \\ g(\langle s_2, \ldots, s_m \rangle, \boldsymbol{q}) \\ g(\langle s_2, \ldots, s_m \rangle, \langle q_2, \ldots, q_m \rangle) \end{array} \right\} \end{aligned}$$

The time complexity of DTW is $\mathcal{O}(|\boldsymbol{s}||\boldsymbol{q}|)$ which is clearly intractable for computing similarities of thousands of trajectories. However, recall that we aim at finding the N best matches for a given query. This allows for pruning some DTW computations using lower bounds f, i.e., $f(\boldsymbol{s}, \boldsymbol{q}) \leq g(\boldsymbol{s}, \boldsymbol{q})$, with an appropriate function f that can be more efficiently computed than g [10]. We use two different lower bound functions, f_{kim} [6] and f_{keogh} [5], that are defined as follows: f_{kim} focuses on the first, last, greatest, and smallest values of two sequences [6]

$$f_{kim}(\boldsymbol{s}, \boldsymbol{q}) = \max\left\{|s_1 - q_1|, |s_m - q_m|, |\max(\boldsymbol{s}) - \max(\boldsymbol{q})|, |\min(\boldsymbol{s}) - \min(\boldsymbol{q})|\right\}$$

and can be computed in $\mathcal{O}(m)$. However, the greatest (or smallest) entry in the transformed paths is always close or identical to π (or $-\pi$) and can thus be ignored. Consequentially, the time complexity reduces to $\mathcal{O}(1)$ [10]. The second lower bound f_{keogh} [5] uses minimum ℓ_i and an maximum values u_i for subsequences of the query $\boldsymbol{q}$ given by

$$\ell_i = \min(q_{i-r}, \ldots, q_{i+r}) \quad \text{and} \quad u_i = \max(q_{i-r}, q_{i+r}),$$

where r is a user defined threshold. Trivially, $u_i \geq q_i \geq \ell_i$ holds for all i and the lower bound f_{keogh} is given by

$$f_{keogh}(\boldsymbol{q}, \boldsymbol{s}) = \sqrt{\sum_{i=1}^{m} c_i} \quad \text{with} \quad c_i = \begin{cases} (s_i - u_i)^2 : if\ s_i > u_i \\ (s_i - \ell_i)^2 : if\ s_i < \ell_i \\ 0 \qquad\quad : otherwise \end{cases}$$

which can also be computed in $\mathcal{O}(m)$ (see [7] for details).

Algorithm 1 extends [10] to compute the N most similar trajectories for a given query $\boldsymbol{q}$. Lines 2–9 compute the DTW distances of the first N entries in the database and store the entry with the highest distance to $\boldsymbol{q}$. Lines 10–21 loop over all subsequent trajectories in $\mathcal{D}$ by first applying the lower bound functions f_{kim} and f_{keogh} to efficiently filter irrelevant movements before using the exact DTW distance for the remaining candidates. Every trajectory, realising a smaller DTW distance than the current maximum, replaces its peer and the variables $maxdist$ and $maxind$ are updated accordingly. Note that the complexity of Algorithm 1 is linear in the number of trajectories in $\mathcal{D}$. In the worst case, the sequences are sorted in descending order by the DTW distance, which requires to compute all DTW distances. In practice we however observe much lower run-times.

An important factor is the tightness of the lower bound functions. The better the approximation of the DTW the better the pruning. The parameter N plays also a crucial part in the effectiveness of the algorithm. If we set $N = 1$ the

Algorithm 1 TOP_N(N,$\boldsymbol{q}$,$\mathcal{D}$)

Input: number of near-neighbour movements N, query trajectory $\boldsymbol{q}$, game $\mathcal{D}$
Output: The N most similar trajectories to $\boldsymbol{q}$ in $\mathcal{D}$

$output = \varnothing \ \wedge \ maxdist = 0 \ \wedge \ maxind = -1$
for $i = 1, \ldots, N$ **do**
  $dist = g(\boldsymbol{q}, \mathcal{D}[i])$
  $output[i] = \mathcal{D}[i]$
  if $dist > maxdist$ **then**
    $maxdist = dist$
    $maxind = i$
  end if
end for
for $i = N + 1, \ldots, |\mathcal{D}|$ **do**
  if $f_{kim}(\boldsymbol{q}, \mathcal{D}[i]) < maxdist$ **then**
    if $f_{keogh}(\boldsymbol{q}, \mathcal{D}[i]) < maxdist$ **then**
      $dist = g(\boldsymbol{q}, \mathcal{D}[i])$
      if $dist < maxdist$ **then**
        $output[maxind] = \mathcal{D}[i]$
        $maxdist = \max\{g(\boldsymbol{q}, output[j]) \ : \ 1 \leq j \leq N\}$
        $maxind = \underset{1 \leq j \leq N}{\arg\max}\ g(\boldsymbol{q}, output[j])$
      end if
    end if
  end if
end for

maximum value will drop faster towards the lowest value in the dataset. By contrast, setting $N = |\mathcal{D}|$ requires to compute the DTW distances for all entries in the database. Hence, in most cases, $N \ll |\mathcal{D}|$ is an appropriate choice to reduce the overall computation time.

3.4 Locality Sensitive Hashing

To further improve the efficiency of our algorithm, we will use locality sensitive hashing (LSH) [2] to remove a great deal of trajectories before processing them with Algorithm 1. The idea of LSH is to hash similar objects to the same bucket, so that all objects of a bucket are considered candidates for being near-neighbours. An interesting equivalence class of LSH functions are distance based hashes (DBH) [1] that can be applied together with arbitrary (e.g., non-metric) distance measures.

To define a hash family for our purposes, we first need to define a function $h : \mathcal{D} \rightarrow \mathbb{R}$ that maps a trajectory $\boldsymbol{s} \in \mathcal{D}$ to the set of real numbers. Choosing

two randomly drawn members $\boldsymbol{s}_1, \boldsymbol{s}_2 \in \mathcal{D}$ we define the function h as follows:

$$h_{\boldsymbol{s}_1,\boldsymbol{s}_2}(\boldsymbol{s}) = \frac{dist(\boldsymbol{s},\boldsymbol{s}_1)^2 + dist(\boldsymbol{s}_1,\boldsymbol{s}_2)^2 - dist(\boldsymbol{s},\boldsymbol{s}_2)^2}{2\,dist(\boldsymbol{s}_1,\boldsymbol{s}_2)}.$$

In the remainder, we will use the identity $dist(\boldsymbol{s}_1,\boldsymbol{s}_2) = f_{kim}(\boldsymbol{s}_1,\boldsymbol{s}_2)$ for simplicity. To compute a discrete hash value for $\boldsymbol{s}$ we verify whether $h(\boldsymbol{s})$ lies in a certain interval $[t_1, t_2]$,

$$h^{[t_1,t_2]}_{\boldsymbol{s}_1,\boldsymbol{s}_2}(\boldsymbol{s}) = \begin{cases} 1 : h_{\boldsymbol{s}_1,\boldsymbol{s}_2}(\boldsymbol{s}) \in [t_1,t_2] \\ 0 : \quad otherwise \end{cases}$$

Optimally, the interval boundaries t_1 and t_2 are chosen so that the probability that a randomly drawn $\boldsymbol{s} \in \mathcal{X}$ lies with 50% chance within and with 50% chance outside of the interval. The set $\mathcal{T}$ defines the set of admissible intervals,

$$\mathcal{T}(\boldsymbol{s}_1,\boldsymbol{s}_2) = \left\{ [t_1,t_2] \,:\, Pr_{\mathcal{D}}(h^{[t_1,t_2]}_{\boldsymbol{s}_1,\boldsymbol{s}_2}(\boldsymbol{s})) = 0) = Pr_{\mathcal{D}}(h^{[t_1,t_2]}_{\boldsymbol{s}_1,\boldsymbol{s}_2}(\boldsymbol{s})) = 1) \right\}.$$

Given h and $\mathcal{T}$ we can now define the DBH hash family that can be directly integrated in standard LSH algorithms:

$$\mathcal{H}_{DBH} = \left\{ h^{[t_1,t_2]}_{\boldsymbol{s}_1,\boldsymbol{s}_2} \,:\, \boldsymbol{s}_1,\boldsymbol{s}_2 \in \mathbb{R} \wedge [t_1,t_2] \in \mathcal{T}(\boldsymbol{s}_1,\boldsymbol{s}_2) \right\}$$

Using random draws from $\mathcal{H}_{DBH}$, we construct several hash functions by AND- and OR-concatenation [2]. Given a query trajectory $\boldsymbol{q} \in \mathcal{D}$, the retrieval process first identities candidate objects that are hashed to the same bucket for at least one of the hash functions and computes the exact distances of the remaining candidates using Algorithm 1.

4 Evaluation

In this section, we evaluate our approach in terms of run-time, pruning efficiency, and precision. For our experiments, we use positional data published by the DEBS Grand Challenge in 2013[1]. There are eight players in each team, where every player is equipped with two sensors, one for each shoe. We average these two values to obtain only a single measurement for every player at a time. Discarding additional data that is not useful in our context leaves us with a stream of

$$(\text{sensor/player id}, \text{timestamp}, \text{player coordinates})$$

[1] http://www.orgs.ttu.edu/debs2013/index.php?goto=cfchallengedetails

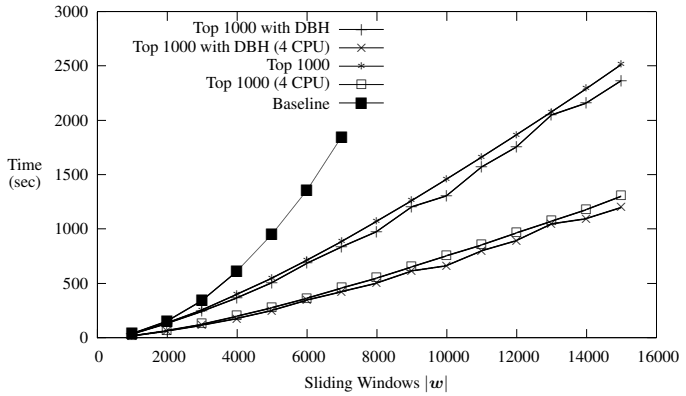

Fig. 1. Run-time.

triplets. Additionally, we discard all data points that have been recorded before or after the game as well as data points that occurred outside of the pitch. We also remove the effects of changing sides after half time by appropriate transformations. Finally, we average the positional information of each player over 100ms to reduce the overall amount of data and use trajectories of size 10.

In our first experiment, we focus on 15,000 consecutive positions of one player, so that we are still able to compare performances to the exact baseline using the DTW distance from Section 3.3. We compute the N-most similar trajectories using Algorithm 1, where $N = 1,000$ and study run-times of the different approaches. Figure 1 (left) shows the results. The computation time of the baseline grows exponentially in the size of the data $\mathcal{D}$. Algorithm 1 performs slightly super-linear and clearly outperforms the baseline. Pre-filtering trajectories using DBH results in only a small speed-up. Adding more CPUs further significantly improves the run-time of the algorithms and indicates that parallelisation of the approach allows for computing near-neighbours for large data sets in only a couple of minutes.

The observed improvements in run-time are the result of a highly efficient pruning strategy. Table 4 shows the amount of trajectories that are pruned for different amounts of data. Note that the DBH pruning depends on the data and not on the ratio $\frac{N}{|\mathcal{D}|}$. The effectiveness of pruning using f_{kim} and f_{keogh} increases with increasing amounts of data for constant N.

We now investigate the accuracy of the proposed approach. We compute the 1000 most similar trajectories for all 35,248 player movements and measure the effect of DBH in terms of the *precision@N*. For every query $\boldsymbol{q}$ we computed the performance for $N \in \{100, 200, \ldots 1000\}$ and averaged the results that are shown in Figure 2. For completeness we also included the worst cases. The

Table 1. Pruning efficiency

trajectories	f_{kim}	f_{keogh}	**DBH**	**Total**
1000	0%	0%	11.42%	11.42%
5000	0.28%	34.00%	16.33%	50.61%
10000	9.79%	41.51%	17.80%	60.10%
15000	17.50%	46.25%	11.82%	75.57%

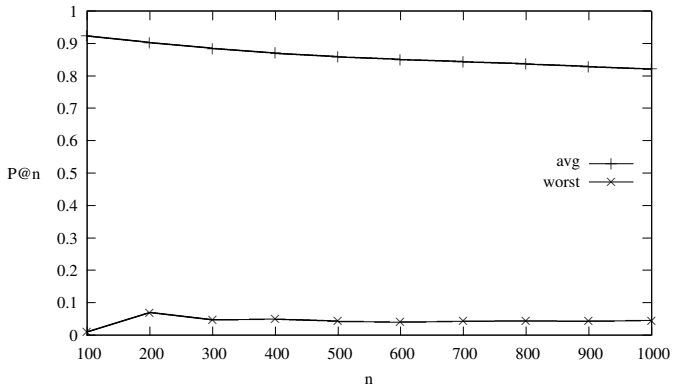

Fig. 2. Accuracy of DBH.

quality of the candidates covers a broad range and the worst cases are clearly inappropriate for accurate computations of similarity. Nevertheless, on average DBH performs well and only slightly decreases in the size of N. Figure 3 shows an exemplary query trajectory (top, left) as well as five similar trajectories found by DBH, where the axes denote the coordinates on the pitch of the respective movement. The retrieved near-duplicates are very close to the query and well suited for further processing.

5 Conclusion

In this paper, we presented an approach to efficiently compute similar movements in positional data streams. Our solution is based on dynamic time warping and distance based hashing. Empirically, we showed the efficiency and accuracy of our approaches. Future work will deal with detecting frequent movements across players.

References

1. V. Athitsos, M. Potamias, P. Papapetrou, and G. Kollios. Nearest neighbor retrieval using distance-based hashing. In *Proceedings of the 2008 IEEE 24th International Conference on*

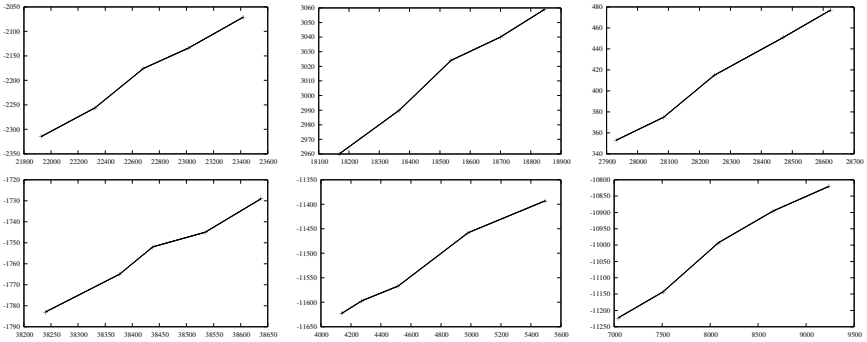

Fig. 3. Exemplary results: The query trajectory is shown in the top-left figure. The other figures show five similar trajectories found by our approach.

Data Engineering, pages 327–336, 2008.

2. A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529, 1999.
3. Andreas Grunz, Daniel Memmert, and Jrgen Perl. Tactical pattern recognition in soccer games by means of special self-organizing maps. *Human Movement Science*, 31(2):334 – 343, 2012. Special issue on Network approaches in complex environments.
4. C.-H. Kang, J.-R. Hwang, and K.-J. Li. Trajectory analysis for soccer players. In *Proceedings of the Sixth IEEE International Conference on Data Mining - Workshops*, pages 377–381, 2006.
5. Eamonn Keogh. Exact indexing of dynamic time warping. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 406–417, 2002.
6. S.-W. Kim, S. Park, and W. W. Chu. An index-based approach for similarity search supporting time warping in large sequence databases. In *Proceedings of the 17th International Conference on Data Engineering*, pages 607–614, 2001.
7. D. Lemire. Faster retrieval with a two-pass dynamic-time-warping lower bound. *Pattern Recogn.*, 42(9):2169–2180, September 2009.
8. M. Perše, M. Kristan, S. Kovačič, G. Vučkovič, and J. Perš. A trajectory-based analysis of coordinated team activity in a basketball game. *Computer Vision and Image Understanding*, 113(5):612 – 621, 2009.
9. Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of speech recognition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.
10. T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, pages 262–270, 2012.
11. Michail Vlachos, D. Gunopulos, and Gautam Das. Rotation invariant distance measures for trajectories. In *Proceedings of tInternational Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 707–712, 2004.