Similar Sports Play Retrieval with Deep Reinforcement Learning

Zheng Wang, Cheng Long, Gao Cong

Abstract—With the proliferation of commercial tracking systems, sports data is being generated at an unprecedented speed and the interest in sports play retrieval has grown dramatically as well, where a play corresponds to a fragment of a game. Existing solutions for similar play retrieval usually assume that a database of plays are materialized, which, however, is not well aligned with the practice that data is stored in units of game. In this paper, we propose to search for similar plays directly from a database of games. We tackle three challenges of the task, namely (1) how to measure the similarity between two plays, (2) how to efficiently find a similar play to a query play within a game, and (3) how to efficiently find a similar play within a database of many games. For the first challenge, we propose a deep learning approach called play2vec to learn the representations of sports plays. play2vec is robust against noise and runs in linear time. For the second challenge, we develop a suite of algorithms including two based on reinforcement learning, which use learned policies for deciding where to split a game to generate candidate plays. For the third challenge, we develop a method called ScoreSearch based on deep metric learning, which is able to prune games from being searched for better efficiency. We conduct experiments on real-world soccer match data to evaluate the techniques developed in this paper.

Index Terms—similar play retrieval; deep representation learning; deep reinforcement learning; play2vec; trajectory and sequence similarity; mining sub-trajectories

1 INTRODUCTION

Nowadays, it becomes a common practice to track the moving agents in a sports game (e.g., the players and the ball in a soccer game) using cameras and/or GPS devices. For example, the SportVU system by STATS LLC, which is an optical tracking system based on cameras, has been used by professional sports leagues such as France's Ligue de Football Professionnel (LFP) and American National Basketball Association (NBA). The data collected by these tracking systems can be represented as the trajectories of the players and the ball in a game, i.e., it embeds both spatial and temporal features of a game. Hence, it is usually termed as spatiotemporal sports data and has been used in many sports analytics tasks such as team formation detection [1], [2], movement pattern mining [3], [4], tactics discovery [3], score prediction [5], [6], and similar play retrieval [7].

A *play* corresponds to a fragment of a game and has its duration varying from seconds to minutes. Similar play retrieval is a process of finding those plays from a database, which are similar to a query play. It is widely used in some emerging sports applications such as ESPN and Team Stream to recommend similar plays to sports fans. Besides, it could help sports club managers and coaches to improve team tactics when preparing for an upcoming match [3].

Existing studies on similar play retrieval usually assume that some candidate plays called *data plays* are stored in a database for query processing [7], [8]. For example, in the proposal [7], all possible fractions of a game are materialized as data plays, whose duration fall in a range, e.g., from 1s to 5s, and are stored in a database for query processing. While this strategy is simple, it is not an ideal solution. First, with a small range used for materializing data plays, it restricts the space of possible plays for query processing, i.e., only those with the length in the small range are considered. When the length of a query play is outside the range, those plays that are similar to the query play may be missed (since they should have similar lengths as the query play but are not materialized). Second, with a very large range, more data plays will be stored in the database, and the cost of both materializing all possible data plays and searching for similar data plays would be increased accordingly.

In this paper, we propose a new strategy, namely we do not materialize data plays from games, but search games directly for plays that are similar to a given query play. Specifically, given a database of games and a query play, the problem is to find those fragments of the games (i.e., plays), which are the most similar to the query play. Different from existing studies [7], [8], which assume a database of plays, *each as a whole being considered*, this problem takes as input a database of games, *each with its fragments being considered*. This problem setting has two advantages over existing ones. First, the materialization of plays is no longer necessary and thus the specification of a duration range is avoided. Second, this is more aligned with real application scenarios, since the raw spatiotemporal sports data is collected in the units of games, but not plays.

We call this problem of searching for plays from a database of games, which are similar to a query play, *similar sub-game retrieval* or simply *similar play retrieval* (SimPlay). There are three challenges in solving the SimPlay problem: (1) *how to measure the similarity between two plays* (given that a play has a complex structure), (2) *how to search for sub-games (plays) from a specific game, which are similar to a query*

• *Z. Wang, C. Long and G. Cong are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. E-mail: {wang_zheng, c.long, gaocong}@ntu.edu.sg*
Corresponding author: Cheng Long

play (given that a game involves many possible plays), and (3) *how to search over a databases of many games for plays, which are similar to a query play (given that there are usually many games in the database)*. In this paper, we develop techniques to solve these challenges, as explained in Sections 1.1, 1.2, and 1.3, respectively.

1.1 Measure Similarity Between Two Plays

In our prior study [8], we propose to learn representations of plays in a low-dimensional space using deep models, which we call *play2vec*, such that the (Euclidean) distances in the space capture the similarities among the plays well. The core idea of our approach is to treat a play as a sequence of play segments with uniform durations and then design a denoising sequential encoder-decoder (DSED) model for extracting a feature vector from the sequence. play2vec has obvious advantages over existing ones [7] in the aforementioned three aspects. Regarding the effectiveness, our method is based on the popular encoder-decoder deep model which is widely known to perform well in extracting features from sequential data. Regarding the efficiency of computing the similarity of two plays, our method runs in $O(n+d)$ time while existing ones have time complexities at least $O(n^2)$, where n is the length of the longest trajectory in a play and d is the size of a learned feature vector which is small. Regarding the robustness, our method involves two mechanisms to deal with sampling errors and measurement errors. First, in the step of mapping play segments to tokens, a grid is used such that it is not sensitive to errors. Second, in the encoder-decoder model, the data is first injected with some noises and then denoised for training which would help mitigate the problems caused by errors.

1.2 Search Similar Play Within a Game

A simple method is to enumerate all possible plays of a game, compute the similarity between each play and the query play, and return the play with the greatest similarity. For better efficiency, we borrow the ideas of existing techniques that have been proposed for the sub-trajectory similarity search problem in our prior study [9] for searching similar plays within a game. First, we design an *incremental* strategy for computing the similarities between different plays of a game and the query play. The resulting algorithm is still an exact one and we call it ExactS. Second, we design an approximate algorithm, which only considers those plays that have similar length as the query play. We call this algorithm SizeS. Compared with ExactS, SizeS explores a smaller search space and thus has better efficiency. Third, to further improve the efficiency, we propose two types of algorithms, which share the idea of splitting a game into plays and returning the most similar play to the query play. The first type uses heuristics for deciding where to split a game and the second type models the process of splitting a game as one of *Markov decision process* (MDP) and employs reinforcement learning (RL) for finding policies for decision making. These splitting-based algorithms consider an even smaller space of plays to explore and thus have the best efficiency. Among these splitting-based algorithms, the RL based algorithms provide better effectiveness than those based on heuristics since the the policies learned via RL are more intelligent than the heuristics that are human-crafted.

1.3 Search Similar Play Within a Database of Games

An intuitive solution to solve the SimPlay problem is to scan the games in the database, and for each one, compute its most similar play (using the one of the methods presented in Section 4) and update the most similar play found so far. Nevertheless, there are usually many games in a database in practice, and thus this method would not meet the efficiency requirement for real applications. In this paper, we develop a deep metric learning based technique for determining the order of the games in a database for searching the most similar play within a game and ignoring those that are behind others for better efficiency. Specifically, we define a score for each game given a query play to be the maximum similarity between a play of the game and the query play. The rationale of the score is that the games in the databases would be taken in a descending order of their score for searching for the similar plays to a query play. To infer the score of a game efficiently, we develop a deep metric learning method based on triplet network [10] to learn embeddings of games such that the order based on their scores is preserved. When performing the similar play retrieval, we first only consider a fraction r of the games with the top scores for searching for the most similar play, where $r \in (0, 1]$ is a user parameter for controlling the trade-off between effectiveness and efficiency.

1.4 Main Contributions and Organization.

This paper extends our prior work [8], which proposes the play2vec measurement (presented in Section 3), with the following new contributions.

1) We propose a new problem of searching similar plays for a given query play from a database of games. This is different from existing studies, which assume a database of materialized plays, and is more aligned with real applications. (Section 2)
2) We develop a suite of algorithms for searching the play of a game, which is the most similar to a query play. Among them, two use polices learned via reinforcement learning for forming candidate plays. They correspond to adaptions of the techniques in [9], which search for similar sub-trajectories, for searching similar plays within a game. (Section 4)
3) We develop a deep metric learning based method for deciding the order of searching the games in a database and searching only a fraction r of the games that are before others, where $r \in (0, 1]$ is a user parameter for controlling the trade-off of effectiveness and efficiency. (Section 5)
4) We perform extensive experiments on real-world soccer data. The results show that (1) our play2vec measurement consistently outperforms the state-of-the-art in terms of effectiveness and runs faster than existing methods by over one order of magnitude; (2) the splitting-based algorithms achieve the best efficiency and the two RL based algorithms are more effective than other approximate algorithms and run fast; and (3) the deep learning based method improves the efficiency of similar sports play retrieval significantly with some slight effectiveness degrades. (Section 6)

2 PROBLEM DEFINITION

We model a sports *game* by the movements of the objects involved in the game (e.g., in a soccer play, the objects include 22 players from two teams and also a ball). The movement of an object is usually captured by sampling its locations at a certain frequency with tracking technologies such as those based on GPS devices. As a result, the movement of an object corresponds to a sequence of time-stamped locations, which is called a *trajectory*. A trajectory has its form of (x_1, y_1, t_1), (x_2, y_2, t_2), ..., where (x_i, y_i) is the i^{th} location and t_i is the time stamp of the i^{th} location. Therefore, a game corresponds to a set of multiple trajectories.

Given a game $\mathcal{P} =< p_1, p_2, ..., p_n >$ and $1 \leq i \leq j \leq n$, let $\mathcal{P}[i, j]$ denote the portion of $\mathcal{P}$ that starts from the i^{th} frame and ends at the j^{th} frame, i.e., $\mathcal{P}[i, j] =< p_i, p_{i+1}, ..., p_j >$, where each frame p_i contains multiple sampled locations of the players and the ball at that time stamp. We say that $\mathcal{P}[i, j]$ is a *play* of $\mathcal{P}$ for any $1 \leq i \leq j \leq n$. There are in total $\frac{n(n+1)}{2}$ plays in $\mathcal{P}$. Note that any play of a game $\mathcal{P}$ corresponds to a (shorter) game itself.

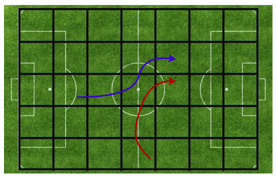
Problem definition. Suppose we have a database of many games. The problem is to search the portion of a game (i.e., a play) in the database, which is the most similar to a query play. We call this problem SimPlay. The SimPlay problem relies on a similarity measurement for two plays, which will be introduced in Section 3.

We note that a more general query is to find the *top-k* similar plays to a query play, which reduces to the user's query as described above when $k = 1$. In this paper, we stick to the setting of $k = 1$ since extending the techniques for the setting of $k = 1$ to general settings of k is straightforward. Specifically, we can adapt the techniques in this paper to general settings of k by simply maintaining the k^{th} most similar play and updating it when a play that is more similar than it is found.

3 SIMILARITY MEASUREMENT FOR PLAYS

Given a database of plays $\boldsymbol{\mathcal{D}}$ (note that a game can also be regarded a play), we aim to learn a vector representation $\boldsymbol{v} \in \mathbb{R}^d$ for each play $\mathcal{P} \in \boldsymbol{\mathcal{D}}$ in a d-dimensional space such that the dissimilarities among plays are well captured by the Euclidean distances in the d-dimensional vector space, i.e., for any two plays, if they are similar, the distance between their vectors would be small.

In this part, we introduce a deep learning method, play2vec, which was introduced in our prior work [8], for learning vector presentations of plays. The core idea is that we break each play into a sequence of non-overlapping segments of a fixed duration, each called a *play segment*, and then design an encoder-decoder model to extract the features from the sequence as a vector. Specifically, our method first builds a corpus $\boldsymbol{\mathcal{V}}$ based on all play segments (Section 3.1), then adopts the Skip-Gram with Negative Sampling (SGNS) model [11] for learning distributed representations of the tokens in $\boldsymbol{\mathcal{V}}$ (Section 3.2), and eventually glues all distributed representations of the play segments in a play, yielding a vector representation $\boldsymbol{v}$ using the Denoising Sequential Encoder-Decoder (DSED) model that is designed in this paper (Section 3.3).



$$\rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

(a) soccer pitch (b) segment matrix $\mathcal{M}$

Fig. 1. Mapping a play segment to a matrix. The soccer pitch is divided into 5×7 grid map. There are two trajectories with the color red and blue in the soccer pitch [8].

3.1 Building a Sports Corpus

First, we introduce a process of mapping a play segment to a binary matrix with the help of a grid. Specifically, we divide the pitch into a grid with equal cell size γ, for which we would have a corresponding matrix called *segment matrix*, i.e., each cell in the grid has a corresponding entry in the matrix. Given a play segment, which consists of a set of trajectories, we set to 1 all those entries whose corresponding cells are traveled through by the trajectories and 0 the remaining entries. An example is shown in Figure 1 for illustration. Note that in this example, the 5×7 grid map is determined by the cell size, which is set empirically in Section 6.2.4.

Since each segment matrix has binary values only, the number of possible segment matrices is limited. A simple strategy is to create one unique token for each possible segment matrix. Nevertheless, with this strategy, the resulting corpus could be big, which may affect the effectiveness of the representation learning afterwards. Thus, we propose to scan the segment matrices one by one and for each segment matrix, we create a new token only if it is dissimilar from those segment matrices that have been scanned to a certain extent, where we use the Jaccard index for measuring the similarity between two segment matrices $\mathcal{M}$ and $\mathcal{M}'$, which is defined as follows.

$$\mathcal{J}(\mathcal{M}, \mathcal{M}') := \frac{m_{11}}{m_{01} + m_{10} + m_{11}}, \tag{1}$$

where m_{11} means the total number of attributes where $\mathcal{M}$ and $\mathcal{M}'$ both have a value of 1, m_{01} (or m_{10}) means the total number of attributes where $\mathcal{M}$ is 0 (or 1) and $\mathcal{M}'$ is 1 (or 0).

The process of building sports corpus is as follows. It first initializes two variables: $\boldsymbol{\mathcal{V}}$ which is the target sports corpus and id which is the index of tokens. It then has a for loop of scanning the play segments. Within the loop, it first computes the segment matrix for the play segment currently being processed. Then, it inserts the first segment matrix and its corresponding token into $\boldsymbol{\mathcal{V}}$. It then finds the pair $(\mathcal{M}', \mathcal{T}')$ such that $\mathcal{M}'$ is the most similar to $\mathcal{M}$ among those maintained in $\boldsymbol{\mathcal{V}}$ under the Jaccard index. If the similarity is above a threshold ϕ, $\mathcal{M}$ is assigned with the same segment token as $\mathcal{M}'$; otherwise, $\mathcal{M}$ is assigned with a new segment token.

3.2 Learning Distributed Representations

In this part, we introduce a method for embedding the play segments (or their corresponding tokens) as d'-dimensional real-valued vectors. Inspired by the success of word2vec techniques in natural language processing, we adopt the Skip-Gram with Negative Sampling (SGNS) model for this task. The effectiveness of a SGNS model depends on how good the context of a token is modeled. The segment tokens occurring in the same context tend to have similar sports scenes. Hence, we use the consecutive segment tokens after and before a token as the forward-looking and backward-looking context of the token, respectively.

In this part, we abuse the notation $\boldsymbol{\mathcal{V}}$ to denote the set containing all segment tokens of the play segments involved in a database of plays. Consider a play $\mathcal{P}$ which involves L play segments. Correspondingly, there is a sequence of L segment tokens which we assume are $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_L$. Then, the m-size window context of a segment token $\mathcal{T}_t$ $(m+1 \leq t \leq L-m)$, denoted by $c^m(\mathcal{T}_t)$, corresponds to $< \mathcal{T}_{t-m}, \ldots, \mathcal{T}_{t-1}, \mathcal{T}_{t+1}, \ldots, \mathcal{T}_{t+m} >$, where we say $\mathcal{T}_t$ is a target segment token and each token in $c^m(\mathcal{T}_t)$ a context segment token. Note that a segment token could be a target one and also a context one (with others as the target ones), i.e., a segment token has two types of roles, namely a target segment token and a context segment token. Given a target token $\mathcal{T}$ and a context token $\mathcal{C}$, we say the token-context pair $(\mathcal{T}, \mathcal{C})$ is positive if $\mathcal{C} \in c^m(\mathcal{T})$ and negative otherwise. Considering that a segment token has two types of role, we define for each segment token a vector $\boldsymbol{v}_{\mathcal{T}} \in \mathbb{R}^{d'}$ for cases it corresponds to a target segment token and a vector $\boldsymbol{v}_{\mathcal{C}} \in \mathbb{R}^{d'}$ for cases it corresponds to a context segment token, where d' is the embedding dimension. We aim to learn these vectors such that we could infer those context segment tokens from a target one with the maximum probability.

We explain the training data and also the loss next. The training data consists of a set of training samples. Each training sample consists of one positive token-context pair $(\mathcal{T}, \mathcal{C})$ (i.e., $\mathcal{C} \in c^m(\mathcal{T})$) and k negative pairs $(\mathcal{T}^i, \mathcal{C})$, where $\mathcal{T}^i$ for $i = 1, 2, ..., k$ is drawn from a segment token distribution $\mathcal{Q}(\mathcal{T})$. Specifically, $\mathcal{Q}(\mathcal{T})$ is a α-smoothed unigram distribution,

$$\mathcal{Q}(\mathcal{T}) := \frac{f(\mathcal{T})^\alpha}{\sum_{\mathcal{T}' \in \boldsymbol{\mathcal{V}}} f(\mathcal{T}')^\alpha}, \tag{2}$$

where $\alpha \in [0, 1]$ and $f(\mathcal{T})$ is the frequency of the segment token $\mathcal{T}$ appearing in the corpus.

The loss is explained as follows. For a token-context pair $(\mathcal{T}, \mathcal{C})$, we model the probability that the pair is positive as $\sigma\Big((\boldsymbol{v}_{\mathcal{T}})^T \cdot \boldsymbol{v}_{\mathcal{C}}\Big)$ (i.e., a sigmoid function is used with its input as the dot product between the vectors of the segment tokens) and the pair is negative as $1 - \sigma\Big((\boldsymbol{v}_{\mathcal{T}})^T \cdot \boldsymbol{v}_{\mathcal{C}}\Big)$. We then define the loss over one training sample using negative log probabilities as follows.

$$\mathcal{L}(\mathcal{T}, \mathcal{C}, \mathcal{T}^i) := -\log \sigma\Big((\boldsymbol{v}_{\mathcal{T}})^T \cdot \boldsymbol{v}_{\mathcal{C}}\Big) - \sum_{i=1}^{k} \log \sigma\Big(-(\boldsymbol{v}_{\mathcal{T}^i})^T \cdot \boldsymbol{v}_{\mathcal{C}}\Big), \tag{3}$$

where $(\mathcal{T}, \mathcal{C})$ is the positive token-context pair and $(\mathcal{T}^i, \mathcal{C})$, for $1 \leq i \leq k$, are the k negative pairs in the training sample. The overall loss function $\mathcal{L}$ is defined by aggregating the losses on all training samples.

The SGNS of the segmentation tokens yields a distributed representation $\boldsymbol{v}_{\mathcal{T}}$ for each $\mathcal{T} \in \boldsymbol{\mathcal{V}}$. The learning process starts from a random vector as the initialized embedding, and it will be continuously updated by the stochastic gradient descent to push the target segment token close to the context in the positive pairs and away from the context in the negative pairs.

3.3 Bottom-up Gluing

In this section, we aim to glue the distributed representations of the segment tokens up together to achieve a comprehensive representation of the play. We propose a new algorithm framework called the Denoising Sequential Encoder-Decoder (DSED). The intuition is that we try to maximize the probability of recovering the most likely real (or clean) tokens from the corrupted initial inputs. For a given play segment and its corresponding token $\mathcal{T}$, we generate a corrupted version of the token, denoted by $\tilde{\mathcal{T}}$, as follows. We scan the locations of the trajectories contained in the play segment time stamp by time stamp, and at each time stamp, we keep the locations with a pre-set probability (and drop the locations with one minus the probability and continue to the next time stamp) and in the case we keep the locations, we sample for each location a noise following a normal distribution $\mathcal{N}(0, 1)$ and add the noise into the location. We then get a new set of tokens based on the corrupted trajectories.

The architecture of the model is presented in Figure 2. We define the encoding hidden representation $\boldsymbol{h_t^{enc}}$ at each time step t, i.e. $\boldsymbol{h_t^{enc}} := LSTM_\theta^{enc}(\boldsymbol{v}_{\tilde{\mathcal{T}}_t}, \boldsymbol{h_{t-1}^{enc}})$. The encoding hidden vector at the last time step $\boldsymbol{h_{last}^{enc}}$ denotes the target representation $\boldsymbol{v}$ and is used to be the hidden vector of the decoder at the first step, i.e. $\boldsymbol{h_0^{dec}} := \boldsymbol{h_{last}^{enc}}$. And EOS is the special token that signals the first step input of the decoding. Also, the decoding hidden representation $\boldsymbol{h_t^{dec}}$ is computed based on the distributed representation of the clean token $\boldsymbol{v}_{\mathcal{T}_{t-1}}$ and the hidden vector $\boldsymbol{h_{t-1}^{dec}}$ from the previous time step, i.e., $\boldsymbol{h_t^{dec}} := LSTM_{\theta'}^{dec}(\boldsymbol{v}_{\mathcal{T}_{t-1}}, \boldsymbol{h_{t-1}^{dec}})$. Note that LSTM is chosen as the computational unit in our model since some existing studies show that LSTM outperforms GRU in tasks requiring modeling long-distance relations [12].

Eventually, we predict $\boldsymbol{v}_{\mathcal{T}_t}^{pred}$ by the softmax function from the decoding hidden representation $\boldsymbol{h_t^{dec}}$ at each time step t,

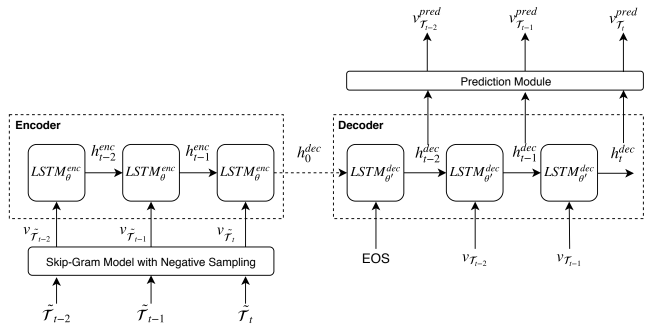
$$\boldsymbol{v}_{\mathcal{T}_t}^{pred} := \frac{exp(\boldsymbol{W}^T \cdot \boldsymbol{h_t^{dec}} + \boldsymbol{b})}{\sum_{\boldsymbol{v} \in \boldsymbol{\mathcal{V}}} exp(\boldsymbol{W}_{\boldsymbol{v}}^T \cdot \boldsymbol{h_t^{dec}} + b_{\boldsymbol{v}})}, \tag{4}$$

where $\boldsymbol{W} \in \mathbb{R}^{|h_t^{dec}| \times |\boldsymbol{\mathcal{V}}|}$, $\boldsymbol{b} \in \mathbb{R}^{|\boldsymbol{\mathcal{V}}|}$, $\boldsymbol{W_v} \in \mathbb{R}^{|h_t^{dec}|}$ and $b_{\boldsymbol{v}} \in \mathbb{R}$ are the weights and bias represented by η, and softmax is the activation function. We define the loss function $\mathcal{L}(\boldsymbol{v}_{\mathcal{T}}, \boldsymbol{v}_{\mathcal{T}}^{pred})$ as the average sequence cross-entropy,

$$\mathcal{L}(\boldsymbol{v}_{\mathcal{T}}, \boldsymbol{v}_{\mathcal{T}}^{pred}) := \frac{1}{L} \cdot \sum_{i=1}^{L} \mathcal{H}\Big(\boldsymbol{v}_{\mathcal{T}_i}, \boldsymbol{v}_{\mathcal{T}_i}^{pred}\Big), \tag{5}$$

where $\mathcal{H}$ is the cross-entropy operator. Parameter θ of the encoding function $LSTM_\theta^{enc}$, θ' of the decoding function $LSTM_{\theta'}^{dec}$ and η are trained to minimize loss $\mathcal{L}(\boldsymbol{v}_{\mathcal{T}}, \boldsymbol{v}_{\mathcal{T}}^{pred})$ over a training set with the Adam stochastic gradient descent method.

The DSED model works as follows. It first builds the sports corpus $\boldsymbol{\mathcal{V}}$, which contains segment tokens $\mathcal{T}$. During the iterative training process, it first maps the plays $\mathcal{P}$ to a sequence of segment tokens $< \mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_L >$ of length L. Then, it constructs for each token $\mathcal{T}_i$ a corrupted version $\tilde{\mathcal{T}}_i$ and learns the distributed representations $\boldsymbol{v}_{\mathcal{T}_i}$ and $\boldsymbol{v}_{\tilde{\mathcal{T}}_i}$ for $\mathcal{T}_i$ and $\tilde{\mathcal{T}}_i$, respectively. Next, it gets the reconstruction pairs $(\boldsymbol{v}_{\mathcal{T}}^{pred}, \boldsymbol{v}_{\mathcal{T}})$, which are computed by the encoder and



$v^{pred}_{\mathcal{T}_{t-2}}$
$v^{pred}_{\mathcal{T}_{t-1}}$
$v^{pred}_{\mathcal{T}_{t}}$
Prediction Module
Encoder
Decoder
$LSTM^{enc}_{\theta}$
h^{enc}_{t-2}
h^{enc}_{t-1}
h^{dec}_{0}
$LSTM^{dec}_{\theta'}$
h^{dec}_{t-2}
h^{dec}_{t-1}
h^{dec}_{t}
$v_{\tilde{\mathcal{T}}_{t-2}}$
$v_{\tilde{\mathcal{T}}_{t-1}}$
$v_{\tilde{\mathcal{T}}_{t}}$
Skip-Gram Model with Negative Sampling
EOS
$v_{\mathcal{T}_{t-2}}$
$v_{\mathcal{T}_{t-1}}$
$\tilde{\mathcal{T}}_{t-2}$
$\tilde{\mathcal{T}}_{t-1}$
$\tilde{\mathcal{T}}_{t}$

Fig. 2. Denoising Sequential Encoder-Decoder Model. Take the sequence of the corrupted tokens $< \tilde{\mathcal{T}_{t-2}}, \tilde{\mathcal{T}_{t-1}}, \tilde{\mathcal{T}_t} >$ as an example, where EOS is a special token indicating the end of the input [8].

decoder components of the learning model and uses an optimizer such as Adam stochastic gradient descent to optimize the parameters.

3.4 Complexity Analysis

The time complexity for computing the similarity between two plays consists of two parts, namely one for learning the representations of the plays and the other for computing the distance between the learned representations in the form of vectors in the d-dimensional space. The former costs $O(n)$ time, where n is the length of the longest trajectory involved in the plays, and we explain as follows: (1) it takes $O(n)$ to convert a query play to a sequence of L segment matrices; (2) it takes $O(|\boldsymbol{\mathcal{V}}|L)$ time to map the segment matrices to their corresponding segment tokens; (3) it takes $O(L)$ to map the segment tokens to their corresponding vectors; and (4) it takes roughly $O(n)$ time to fed the vectors to the DSED model and obtains the target vector representation of the play. Since $|\boldsymbol{\mathcal{V}}|$ could be regarded as a constant and L is bounded by n, we know this process takes $O(n)$ time. And the latter costs $O(d)$ which is obvious. Hence, the overall time complexity is $O(n + d)$.

4 Similar Play Search Within a Game

In this section, we introduce a suite of algorithms for the process of searching for the play of a game, which is the most similar to a query play. They include (1) one exact algorithm called ExactS (Section 4.1), (2) one approximate algorithm, called SizeS, with a tunable parameter to control the trade-off between effectiveness and efficiency (Section 4.2), (3) five splitting-based algorithms including three based on heuristics (Section 4.3) and two on reinforcement learning (Section 4.3). These algorithms were originally proposed for searching similar sub-trajectories in a single trajectory [9]. We adapt these algorithms for searching similar plays, which comprise multiple trajectories, where play2vec [8] is used for measuring the similarity between plays. The summary of the complexities of these algorithms is presented in Table 1.

4.1 The ExactS Algorithm

A straightforward solution is to enumerate all possible plays $\mathcal{P}[i, j]$ $(1 \leq i \leq j \leq n)$ of a game $\mathcal{P}$ and compute the similarity between each $\mathcal{P}[i, j]$ and $\mathcal{P}_q$, and then return the play

TABLE 1
Time complexities of SimPlay within a game.

Algorithms	Time complexity
ExactS	$O(n^2)$
SizeS	$O((m + \xi) \cdot n)$
PSS, POS, POS-D (Splitting-based Algorithms)	$O(n)$
RLS, RLS-Skip (Learning-based Algorithms)	$O(n)$

with the greatest similarity. We further adopt an incremental strategy for computing the similarities, i.e., it initializes the similarity between $\mathcal{P}[i, i]$ and $\mathcal{P}_q$ and then computes $(\mathcal{P}[i, i+1], \mathcal{P}_q), ..., (\mathcal{P}[i, n], \mathcal{P}_q)$ sequentially and incrementally (note that the latent representation of $\mathcal{P}[i, j+1]$ can be computed in $O(1)$ time based on that of $\mathcal{P}[i, j]$ by going through an LSTM block once). This corresponds to an exact solution because it would traverse all possible candidate plays (i.e., $\frac{n(n+1)}{2}$ ones) and runs in $O(n^2)$ time.

4.2 The SizeS Algorithm

Given that the sports data is usually uniformly sampled, the enumeration of all possible plays as ExactS does is too conservative. Therefore, we further design an algorithm called SizeS, which only considers those plays with the length similar to that of the query play. Specifically, we introduce a parameter ξ and consider only those plays with the length within the range $[m - \xi, m + \xi]$, where m denotes the length of the query play and $\xi \in [0, n - m]$. The time complexity of SizeS is $O((\xi + m) \cdot n)$ when the similarities are computed incrementally as it is done by ExactS.

4.3 Splitting-based Algorithms

An intuitive idea to push the efficiency further up is to explore fewer plays than SizeS does. We design a series of algorithms, which share the idea of splitting a game into several plays and returning the one that is the most similar to the query play. These algorithms use different heuristics for deciding where to split the game and are presented as follows.

(1) Prefix-Suffix Search (PSS). The PSS algorithm adopts a greedy heuristic. Specifically, it scans the frames of a game $\mathcal{P}$ in the order of $p_1, p_2, ..., p_n$. When it scans p_i, it computes the similarities between the two plays that would be formed if it splits $\mathcal{P}$ at p_i, i.e., $\mathcal{P}[h, i]$ (i.e., the prefix) and $\mathcal{P}[i, n]$ (i.e., the suffix), and the query $\mathcal{P}_q$, where p_h denotes the frame, at which the last split happened if any and the first frame p_1 otherwise. In particular, for the similarity between the suffix and the query play, it is approximated as that between their reversed versions, where a reversed version of a play $< p_1, p_2, ..., p_n >$ corresponds to $< p_n, ..., p_2, p_1 >$. With this strategy, the similarities of both the prefixes and the suffixes from the query play can be computed incrementally in $O(1)$ time: (1) the similarity of the prefix ending at p_i can be computed incrementally based on that of the prefix ending at p_{i-1} and (2) the similarity of the reversed version of the suffix starting at p_j can be computed incrementally based on that of the reversed version of the suffix starting at p_{j+1}. PSS performs a split operation at p_i if any of these two similarities are larger than the best-known similarity and updates it if so; otherwise, it continues to scan the next frame. Finally, it returns the play with the best-known similarity. The time complexity of PSS is $O(n)$ since at each frame, it takes $O(1)$ to compute the similarities of the prefix and the suffix (reversed version).

(2) Prefix-Only Search (POS). In PSS, when it scans a frame p_i, it considers two plays, i.e., a prefix $\mathcal{P}[h,i]$ and a suffix $\mathcal{P}[i,n]$. An alternative is to consider the prefix $\mathcal{P}[h,i]$ only - one argument is that the suffix $\mathcal{P}[i,n]$ might be destroyed when further splits are conducted. A consequent benefit is that the time cost of computing the similarity of the suffix would be saved. We call this algorithm the POS algorithm. POS has the same time complexity as PSS though the former runs faster in practice.

(3) Prefix-Only Search with Delay (POS-D). In POS, it performs a split operation when a better play is found. This looks a bit rush and may prevent a better play from being formed by extending it with a few more frames. Thus, we design a variant of POS, called POS-D. Whenever a prefix is found to be more similar to the query than the best play known so far, POS-D continues to scan D more frames and splits at one of these $D+1$ frames such that the corresponding prefix is the most similar to the query play with this delay mechanism, the time complexity of the algorithm does not change though in practice, it would be slightly higher.

4.4 Learning-based Algorithms

The effectiveness of splitting-based algorithms relies on the quality of their heuristics. In order to find a play with high similarity, it needs to perform split operations at appropriate frames such that the plays that are formed are similar to the query play. We observe that the process of splitting a game into plays can be regarded as a typical sequential decision making process, i.e., it sequentially scans the frames in a play and for each frame, it makes a decision on whether or not to perform a split operation at the frame. We thus model this process as a Markov decision process (MDP) [13], adopt a deep-Q-network (DQN) reinforcement learning method [14] to learn an optimal policy for the MDP, and then develop two algorithms based on the learned policy for splitting a game into plays, namely RLS and RLS-Skip. We note that reinforcement learning (RL) has been utilized to solve other algorithm problems, such as classic NP-hard problems [15], bipartite graph matching [16], similar subtrajectory search [9], etc.

We first introduce the MDP as follows. A MDP consists of four components, namely states, actions, transitions, and rewards.

• **States.** We denote a state by s, which captures the environment that is taken into account for decision making by an agent. Specifically, when it scans the frame p_t, we define the current state as a pair $(\Theta_{best}, \Theta_{pre})$, where Θ_{best} is the largest similarity of a play found so far, Θ_{pre} is the similarity of the prefix $\mathcal{P}[h,t]$, where p_h denotes the last frame where a split operation happened if any and p_1 otherwise. As could be noticed, the state captures the information about the query play, the game, the frame where the last split happened, and the frame that is being scanned, etc. Note that we ignore the similarity of the suffix in a state based on empirical findings.

• **Actions.** We denote an action by a, which is a possible decision that can be made by the agent. We define two actions, namely $a=1$ meaning to perform a split operation at the frame that is being scanned, and $a=0$ meaning to move on to scan the next frame.

• **Transitions.** A transition means that the state changes from one to another once an action is taken. Specifically, in the process of splitting a game, given a current state s and an action a to be taken, we would observe the next state s' by maintaining the current best Θ_{best} and computing the similarity of the prefix Θ_{pre} incrementally.

• **Rewards.** We denote a reward by r, which is associated with a transition and corresponds to some feedback indicating the quality of the action that causes the transition. We define the reward associated with the transition from state s to state s' after action a is taken as $(s'.\Theta_{best} - s.\Theta_{best})$. We note that this reward definition is consistent with the goal of finding the play that is the most similar to the query play. To see this, let $r_1, r_2, ..., r_{N-1}$ denote the rewards received at the states $s_1, s_2, ..., s_{N-1}$, respectively. Then, when the future rewards are not discounted, we have

$$\Sigma_t r_t = \Sigma_t (s_t.\Theta_{best} - s_{t-1}.\Theta_{best}) = s_N.\Theta_{best} - s_1.\Theta_{best} \quad (6)$$

where $s_N.\Theta_{best}$ corresponds to the similarity of the play returned and $s_1.\Theta_{best}$ corresponds to the best known similarity at the beginning, i.e., 0. That is, maximizing the accumulative rewards is equivalent to maximizing the similarity between the play to be found.

The core problem of a MDP is to find an optimal policy for the agent, which corresponds to a function π that specifies the action that the agent should choose when at a specific state so as to maximize the accumulative rewards. In our MDP, the state space is a three dimensional continuous one, and thus we adopt a Deep-Q-Network method [14]. Specifically, we use the deep Q learning with replay memory [14] for learning the policy.

We next describe two learning-based algorithms RLS and RLS-Skip as follows.

(1) Reinforcement Learning based Search Algorithm (RLS). RLS is same as POS except that it uses a policy learned via DQN instead of human-crafted heuristics for making decisions on where to perform split operations. It can be verified that the time complexity of RLS is $O(n)$. To see this, when scanning each frame, it takes $O(1)$ time to construct the state, which consists of two similarities that can be maintained/computed incrementally, $O(1)$ time to find an action by going through a network with the state, and $O(1)$ time to perform the action (either splitting the game or continuing to scan the next frame).

(2) Reinforcement Learning based Search with Skipping (RLS-Skip). In the RLS, each frame is considered as a candidate location to perform a split operation. In order to push the efficiency further up, we propose to *skip* some frames from being considered as candidate locations for split operations. The benefit would be immediate, i.e., the cost of constructing states and taking actions at these frames is saved. Motivated by this, we augment the MDP definitions by introducing k more actions (apart from two existing ones: performing a split operation $a=1$ and scanning the next frame $a=0$), namely skipping 1 frame, skipping 2 frames, ..., skipping k frames, where k is a hyperparameter and we will evaluate it in the experiments. By skipping j frames ($j = 1, 2, ..., k$), it means to skip frames $p_{i+1}, p_{i+2}, ..., p_{i+j}$ and scan frame p_{i+j+1} next, where p_i is the frame that is being scanned. All other components of the MDP are kept the same as those for RLS. We call this algorithm Reinforcement Learning based Search with Skipping (RLS-Skip). Note that RLS-Skip reduces to RLS when $k = 0$. In addition, RLS-Skip has the same time complexity as RLS and runs faster in practice.

We illustrate the RLS-Skip algorithm in Figure 3. Suppose that it has learned a policy with the DQN and the hyperparameter k is set to 1, which implies that there are three possible actions 0 (no split), 1 (split), and 2 (no split and skip the next 1 frame). For simplicity, we write the $\Theta_{pre=\mathcal{P}[i,j]}$ as the similarity between the $\mathcal{P}[i, j]$ and $\mathcal{P}_q$, and omit the similarity values which are outputted by play2vec. At the beginning, it initializes h, $\mathcal{P}_{best}$ and Θ_{best}. It then scans frame p_1, observes the first state $s_1 = (\Theta_{best}, \Theta_{pre=\mathcal{P}[1,1]})$ and finds the action which is to perform a split operation at p_1. It then updates h, Θ_{best}, and $\mathcal{P}_{best}$. It continues to scan point p_2, observes the second state s_2 and finds the action which is to skip the next 1 frame, i.e., p_3. It keeps h unchanged (since no splits are done) but updates Θ_{best} and $\mathcal{P}_{best}$, respectively. As a result of the skip action, it scans frame p_4 (note that the efforts for constructing the state and taking the action at p_3 are saved) and proceeds similarly for the following frames. Finally, It terminates after scanning frame p_8 and returns $\mathcal{P}[5, 7]$.

5 Similar Play Search Within a Database of Games

An straightforward method for the SimPlay problem is to search the most similar play from *each* game to the query play (using a method introduced in Section 4) and then return the most similar play among the found plays. However, this method would be costly and does not meet the efficiency requirement in practice since there are usually many games in the database. For better efficiency, we present a method called *score-based search* (ScoreSearch) for the SimPlay problem. ScoreSearch computes a score for each game, which corresponds to an estimate of the maximum similarity between a play of the game and the query play, and then searches for the similar play from only a fraction r of the games with the highest scores, where r is a user parameter for controlling the trade-off between effectiveness and efficiency.

Overview of ScoreSearch. We propose to define a score for each game to reflect the *maximum* similarity between a play in the game and the query play. The intuition is that based on the scores, we can focus on a fraction of games with the highest scores only for the SimPlay problem since the most similar play is likely to be from one of these games and other games can be ignored. One possible way to compute the score of a game is to search for the most similar play of the game to the query play (by using one of the method in Section 4) and then return the corresponding similarity as the score of the game. However, this solves the problem more than necessary and would reduce to the straightforward method of searching a similar play within *each* game. A better idea is to estimate the score of a game with some light method only so that if the score is low, the game would then be pruned for searching the similar play. We propose a triplet network [10] based model, which embeds games into vectors and then uses the Euclidean distance between the vector of a game and that of the query play as the score of the game for the query play. To train the network, we use the maximum similarity between a play of a game and a query play for constructing positive and negative labels so that the learned scores capture well the maximum similarities between plays of games and query plays. Once the network is trained, given a query play, we first compute the scores of all games in the database for the query play by feeding them into the network. We can then focus on only a fraction r of games with the highest scores for searching for the most similar play to a query play.

Algorithm 1 The ScoreSearch Algorithm

```
Input: Embedded vectors $V$ of all games with the size $N$ via
    preprocessing and a query play $\mathcal{P}_q$
Output: The most similar play
1: embed the query play, whose embedding is denoted by $v_{\mathcal{P}_q}$;
2: Retrieve the $r \cdot N$ nearest vectors of games to $\mathcal{P}_q$ via a $k$NN
   algorithm;
3: for all retrieved games do
4:     find the play in the game, which is the most similar to
       the query play (e.g., with RLS-Skip);
5: end for
6: return the most similar play to the query play among all
   found plays;
```

Model Architecture. Figure 4 illustrates the model architecture. To train the triplet network, we randomly sample triplets (i.e., an anchor sample as the query, a positive sample, and a negative sample) from the dataset. For each triplet, we use the shortest sample in the triplet as the anchor and denote it by $\mathcal{P}_q$. Among the two other samples, we choose the one which has a larger maximum similarity between one of its plays and $\mathcal{P}_q$ as the positive sample $\mathcal{P}_+$, and the other one as the negative sample $\mathcal{P}_-$. We then embed the samples to the play2vec model, and feed the corresponding embeddings to a triplet network. The triplet network is with three shared feedforward neural networks, denoted as $Net(\cdot)$, which computes two Euclidean distances. That is,

$$d_+ = ||Net(\mathcal{P}_q) - Net(\mathcal{P}_+)||_2 \tag{7}$$

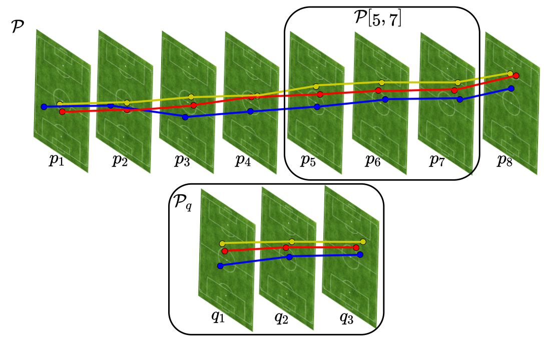
$$d_- = ||Net(\mathcal{P}_q) - Net(\mathcal{P}_-)||_2 \tag{8}$$

where $Net(\mathcal{P}_q)$ (resp. $Net(\mathcal{P}_+)$ and $Net(\mathcal{P}_-)$) denotes the embedding vector of $\mathcal{P}_q$ (resp. $\mathcal{P}_+$ and $\mathcal{P}_-$) via the network $Net(\cdot)$. The network is trained via optimizing the triplet loss shown below:

$$\mathcal{L}(\mathcal{P}_q, \mathcal{P}_+, \mathcal{P}_-) = \max\{d_+ - d_- + \delta, 0\} \tag{9}$$

where δ denotes a desired margin between the positive and negative distances in the embedding space, which aims to enlarge the gap between positive sample $\mathcal{P}_+$ and negative sample $\mathcal{P}_-$. Empirically, we found the light architecture is easy to train, and achieves a superior performance.

The procedure of ScoreSearch algorithm is presented in Algorithm 1. Suppose all games have been fed to the network and their vectors have been computed as a preprocessing process. For a given query play, the ScoreSearch



(a) SimPlay problem

Initial	$h = 1, \mathcal{P}_{best} = \emptyset$ and $\Theta_{best} = 0$				
Frame	State	Action	h	Θ_{best}	$\mathcal{P}_{best}$
p_1	$s_1 = (\Theta_{best}, \Theta_{pre=\mathcal{P}[1,1]})$	split p_1	2	$\Theta_{best=\mathcal{P}[1,1]}$	$\mathcal{P}[1,1]$
p_2	$s_2 = (\Theta_{best}, \Theta_{pre=\mathcal{P}[2,2]})$	skip p_3	2	$\Theta_{best=\mathcal{P}[2,2]}$	$\mathcal{P}[2,2]$
p_3 (skipped)	-	-	-	-	-
p_4	$s_3 = (\Theta_{best}, \Theta_{pre=\mathcal{P}[2,4]})$	split p_4	5	$\Theta_{best=\mathcal{P}[2,4]}$	$\mathcal{P}[2,4]$
p_5	$s_4 = (\Theta_{best}, \Theta_{pre=\mathcal{P}[5,5]})$	skip p_6	5	$\Theta_{best=\mathcal{P}[5,5]}$	$\mathcal{P}[5,5]$
p_6 (skipped)	-	-	-	-	-
p_7	$s_5 = (\Theta_{best}, \Theta_{pre=\mathcal{P}[5,7]})$	split p_7	8	$\Theta_{best=\mathcal{P}[5,7]}$	$\mathcal{P}[5,7]$
p_8	$s_6 = (\Theta_{best}, \Theta_{pre=\mathcal{P}[8,8]})$	no-split p_8	8	$\Theta_{best=\mathcal{P}[5,7]}$	$\mathcal{P}[5,7]$
Output	$\mathcal{P}_{best} = \mathcal{P}[5,7]$ with Θ_{best}				

(b) RLS-Skip with play2vec

Fig. 3. Illustration of similar play retrieval problem and RLS-Skip algorithm with play2vec.

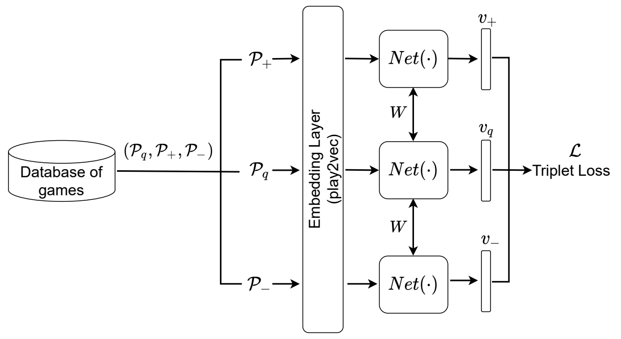


Fig. 4. Overview of model architecture.

algorithm would (1) embed the query play (Line 1), (2) retrieve a fraction r of games using a nearest neighbor algorithm [17] (Line 2), and then (3) search the most similar play from each of the retrieved games using one of the algorithms presented in Section 4 (Lines 3-6).

Time Complexity Analysis. As described above, ScoreSearch involves three parts, whose time complexities are as follows. The part of (1) is $O(m)$, where m is the length of the query play; that of (2) is $\alpha(N)$, where $\alpha(N)$ is the time complexity of a k-nearest neighbor (kNN) search algorithm on a set of N items (e.g., $\alpha(N)$ could be $k \cdot \log(N)$ if the algorithm in [18] is adopted); and that of (3) is $O(r \cdot N \cdot \beta(n))$, where $\beta(n)$ is the time complexity of searching the most similar play of a game of length n to the query play (e.g., if the RLS-Skip algorithm is adopted, $\beta(n)$ is $O(n)$). In conclusion, the time complexity of ScoreSearch is $O(m + \alpha(N) + r \cdot N \cdot \beta(n))$.

6 EXPERIMENTS

6.1 Experimental Setup

Dataset. Our experiments are conducted on real-world soccer player tracking data [1]. The data consist of 7500 sequences and each sequence contains a segment of tracking data corresponding to actual game from a recent professional soccer league, totaling approximately 45 games worth of playing time and over 30 million data points, with redundant and "dead" situations removed. Each segment consists of the tracking data of three parts: 11 defense players, 11 attacking players and a ball. Each part contains (x, y) coordinates obtained at a sampling frequency of 10Hz. More specifically, the coordinates generally belong to the $[-52.5meters, +52.5meters]$ range along the x-axis, and $[-34meters, +34meters]$ range along the y-axis, with the very center of the pitch being $(0, 0)$. Table 2 presents the statistics of the dataset.

1. Data Source: STATS, copyright 2019 (https://www.stats.com/artificial-intelligence)

TABLE 2
Dataset statistics.

Statistics	Frequency
#Sequences	7500
Playing Time	45 games
Data Points	30.4M
X-axis	$[-52.5meters, +52.5meters]$
Y-axis	$[-34meters, +34meters]$
Sampling Rate	10Hz

Algorithms. For the play similarity measurement, we study our proposed play2vec as well as the following four measurements, namely DTW [19], Frechet [20], Chalkboard [7], and EMDT [21]. Detailed description of these measurements can be found in our prior work [8].

For the problem of searching for the most similar play to a query play within a game, since it is a new problem and no existing algorithms can be used, we evaluate the algorithms proposed in this paper, namely ExactS, SizeS, PSS, POS, POS-D, RLS, and RLS-Skip. Note that we did not consider the method proposed in the work [7], which materializes all plays of lengths from 1s to 5s, for two reasons: (1) the range from 1s to 5s is ad-hoc and may not serve well longer query plays and there is no clear clue about how to set the range; and (2) the SizeS algorithm could be regarded as a similar method to this one, which considers all those plays with similar lengths to the query play's, i.e., the range is adaptive.

Parameter Setting. The default size of cells is 3 meters and short duration of segments is 1 second in the experiments. After building a sports corpus via the Jaccard index (the threshold is 0.3), we got 50,465 unique tokens for our dataset. We use a 2-layer LSTM as the computational unit in LSTM-Encoder. The representation dimension of the learned segment tokens and plays are set to 20 and 50 respectively. The context window size in the distributed representation learning is set to 5. The α-smoother is set to 3/4 following the negative sampling in word2vec. Additionally, in the training process, we train our model on 5k generated plays with random noise and dropping rate and adopt Adam stochastic gradient descent with an initial learning rate of 0.01. In order to avoid the gradient vanishing problem, a maximum gradient norm constraint is used and set to 5. For the parameters of baselines, we follow their strategies described in the original papers.

For SizeS and POS-D, we use the setting $\xi = 5$ and $D = 5$ by default, respectively. We will study the effects of these two parameters in Section 6.3. For training the RL-based models (i.e., RLS and RLS-Skip), we use a feedforward neural network with 2 layers. In the first layer, we use the tanh function with 25 neurons, and in the second layer, we use the softmax function with $2+k$ neurons as the output, corresponding to different actions, where for RLS, we use the setting $k=0$ and for RLS-Skip, we use the setting $k=3$ by default. We will study the effect of the skipping parameter k in Section 6.3. In the training process, the size of replay memory for the DQN method is set at 2000. We train our model on 25k pairs of plays chosen randomly, using the Adam stochastic gradient descent with an initial learning rate of 0.001. The parameter ϵ is set at 0.1 with decay 0.99 for the ϵ-greedy strategy in the DQN method, and the reward discount rate γ is set as 0.99.

Evaluation Platform. All the methods are implemented in Python 3.6. The implementation of play2vec and RLS are based on tensorflow 1.8.0. The experiments are conducted on a machine with Intel(R) Xeon(R) CPU E5-1620 v2 @3.70GHz 16.0GB RAM and one Nvidia GeForce GTX 1070 GPU. The codes can be downloaded via the link [2] to reproduce the results.

6.2 Evaluation of the play2vec Measurement

We first study the effectiveness of play2vec. The lack of ground-truth makes it a challenging problem to evaluate the accuracy. To overcome it, we follow three recent studies [22], [23], [24] which propose to quantify the accuracy of trajectory similarity with Self-similarity, Cross-similarity and KNN-similarity comparisons, respectively. There are two frequently used parameters: noise rate (radius is set to 1 meter.) and dropping rate with varying values from 0.2 to 0.6. The two parameters are to measure the probabilities of adding noise or dropping sampling points of each trajectory in a play, respectively.

6.2.1 Self-similarity Comparison

In this experiment, we randomly choose 50 plays to form the query set (denoted as Q) and 1000 plays as the target database (denoted as D) from the dataset. For each play $P \in Q$, we create two plays by randomly sampling 20% points for each trajectory in the play, denoted as P_a and P_b, and we use them to construct two datasets $Q_a=\{P_a\}$ and $Q_b=\{P_b\}$. Similarly, we get D_a and D_b from the target database D. Then for each query $P_a \in Q_a$, we compute the rank of P_b in the database $Q_b \cup D_b$ using different methods. Ideally, P_b should be ranked at the top since P_a and P_b are generated from the same play P. To evaluate the robustness of different approaches to noise, we consider introducing two types of noises. First, we corrupt each trajectory of each play in both Q_a and $Q_b \cup D_b$ as follows: We randomly sample a fraction of the points (denoted by noise rate r_1) and for each sample point we distort the coordinate values by adding Gaussian noises with a standard normal distribution. We vary r_1 from 0.2 to 0.6 and report the mean rank of the queries. Note that the mean rank in self-similarity evaluation is a rank-based metric defined as $\frac{1}{|Q_a|}\sum_{P_a} rank(P_b)$, where $rank(P_b)$ denotes the rank of P_b in $Q_b \cup D_b$ for a query $P_a \in Q_a$. Second, we randomly drop a fraction of points from each trajectory of each play in both Q_a and $Q_b \cup D_b$. We vary dropping rate r_2 from 0.2 to 0.6 and report the mean rank of the queries. The mean rank results can be found in our prior work [8]. We observe that play2vec consistently outperforms the other methods by a large margin as we vary the two types of noise. We also observe that most of the methods are not very sensitive to the noise rate except that DTW and EMDT degrade quickly when we increase the noise rate to 0.5. This is because the matching cost of DTW is determined by the pairwise point-matching and errors will be accumulated with noises. However, Frechet maintains an infimum of the matching cost that is robust to noise changes. With regard to Chalkboard, it splits trajectories into overlapping segments, which can alleviate the noise interruption to some degree.

2. https://github.com/zhengwang125/SimPlay

6.2.2 Cross-similarity Comparison

A good similarity measure should preserve the distance between two plays regardless of the sampling rate or noise interference. We use a metric from the literatures [23], [24] to evaluate the effectiveness for Cross-similarity comparison, namely Cross Distance Deviation (CDD) as defined below.

$$CDD(P_a, P_b) = \frac{|S(P_{a'}(r), P_{b'}(r)) - S(P_a, P_b)|}{S(P_a, P_b)}, \quad (10)$$

where $S(\cdot,\cdot)$ is a similarity measure such as DTW or Frechet; P_a and P_b are two original plays; $P_{a'}(r)$ and $P_{b'}(r)$ are their variants that are obtained by randomly dropping points (or adding noise) with rate r. A small CDD value indicates that an algorithm is robust and is able to preserve the original distance well. In this experiment, we randomly select 1,000 play pairs (P_a, P_b) from the dataset. The average CDD results can be found in our prior work [8]. We observe that play2vec outperforms other baselines consistently for different noise and dropping rates. Note that play2vec is very close to the ground truth over various dropping rates because a cell of the grid maps is considered occupied only if one sample point falls in the cell. Therefore, play2vec can effectively handle the low sampling issue of data.

6.2.3 KNN-similarity Comparison

In this experiment, we study the accuracy and robustness of play2vec and the other baselines for KNN-similarity search on plays when we vary the dropping rate or noise rate. To circumvent the issue of lack of ground-truth, we follow the experimental methodology that is proposed by existing studies [22], [24]: We first randomly select 20 plays as the query set and 500 plays as the target database, and for each query we employ each method to find its Top-K plays as the ground-truth of each method; Then we corrupt each play in the target database by randomly dropping points or adding noise, and retrieve the Top-K plays from the corrupted database; Finally, we compare the retrieved Top-K plays against the ground-truth to compute the precision, i.e., the proportion of true Top-K plays among the retrieved Top-K plays. We vary the value of K by 20, 30, 40, and vary the dropping rate or noise rate from 0.2 to 0.6. The average precision results can be found in our prior work [8]. We observe that play2vec performs the best consistently.

6.2.4 Parameter Study

We next evaluate the effect of the cell size on the effectiveness of play2vec. Intuitively, a small cell size provides a higher resolution of the sports scene, but it also generates more tokens, which lead to higher training complexity and reduce robustness. Detailed results can be found in [8]. We observe that the performance becomes better as the cell size grows from 1m to 3m, and drops for Cross-similarity and KNN similarity when the cell size becomes 4m. With the smallest cell size (1m) play2vec performs the worst. This is probably because the high model complexity makes it difficult to train and this is in line with our intuition. Therefore, we set the cell size at 3 meters for the other experiments because it offers a better robustness.

6.2.5 Efficiency Evaluation

This set of experiments is to evaluate the efficiency of different methods for the sports play retrieval. Detailed results can be found in [8]. We observe that play2vec performs the best among all methods and is over an order of magnitude faster than the most efficient baseline Chalkboard. This is because play2vec takes linear time to retrieve similar plays as discussed in Section 3.4. We also notice that play2vec scales linearly with the database size and the disparity between them increases as the size of the target database grows.

6.2.6 User Study

Since Chalkboard performs the best among all the baselines, and is dedicated for similar play retrieval, we compare play2vec with Chalkboard for play retrieval via a user study. We randomly select 10 plays as the query set and for each query we employ play2vec and Chalkboard to retrieve Top-1 play from the target database, respectively. We recruited seven volunteers with strong soccer background to annotate the relevance of retrieved plays. We first spent 10 minutes to get everyone understand the questions. Then, for each of the 10 queries each volunteer specifies the most relevant result between the Top-1 result retrieved by play2vec and the Top-1 result retrieved by Chalkboard. Note that volunteers do not know which result is from which method. Results can be found in [8]. We observe that play2vec performs much better than Chalkboard for 8 out of the 10 queries. Overall, play2vec gets 82.86% votes (over 70 votes) while Chalkboard only gets 7.14% votes.

6.3 Evaluation of Algorithms of Searching Similar Plays Within a Game

We randomly sample 10,000 pairs of sports games/plays from the dataset, and for each pair, we use the short one as the query play to search the most similar play from the other one. We report the average results under each setting.

6.3.1 Evaluation Metrics

To evaluate the effectiveness of an approximate algorithm for the SimPlay problem, we adopt three metrics in the recent study [9]. (1) Approximate Ratio (AR): It is defined as the ratio between the dissimilarity of the play returned by an approximate algorithm and that of the play returned by an exact algorithm. (2) Mean Rank (MR): We sort all the plays of a game in an ascending order of their dissimilarities wrt a query. MR is defined as the rank of the solution returned by an approximate algorithm. (3) Relative Rank (RR): RR is a normalized version of MR by the total number plays of a game. Note that for all measurements, a smaller value indicates a better algorithm.

6.3.2 Effectiveness Evaluation

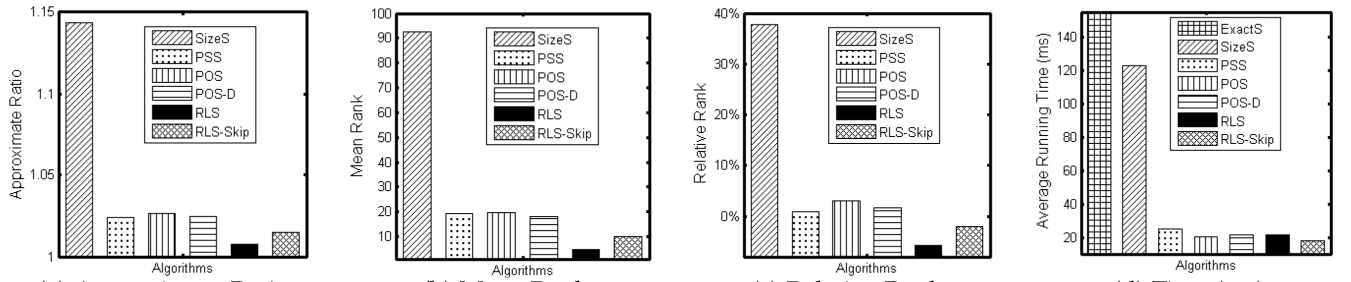
Figure 5 (a)-(c) show the effectiveness results. We observe that learning-based algorithms (i.e., RLS and RLS-Skip) consistently outperform all other non-learning based approximate algorithms in terms of all three metrics. For example, RLS outperforms PSS, the best non-learning based algorithm, by 75% (resp. 60%) in terms of Mean Rank (resp. Relative Rank). As expected, POS-D slightly outperforms POS because it checks more plays for choosing split points during the search process. Additionally, we observe that the effectiveness of SizeS is much worse, probably because the length of the most similar play may not have similar length as the query play. RLS-Skip is a bit less effective than RLS, but still better than those non-learning based algorithms owing to the benefit of its learned policy for decision making.

Furthermore, we partition the query plays into four groups, namely G1, G2, G3, and G4, each with 10,000 plays, such that the lengths of the plays in a group are as follows: G1 = [100, 150), G2 = [150, 200), G3 = [200, 250) and G4 = [250, 300). Then, we randomly select 10,000 games and use them as a database for each group. We report the average results for each group in terms of RR in Figure 6 (a).

The results of AR and MR provide similar clues and thus are omitted due to the page limit. Overall, these approximate algorithms remain stable except for SizeS. We observe that SizeS gets worse as the query length increases. This is because for a longer query play, a larger range may be required to retain the effectiveness.

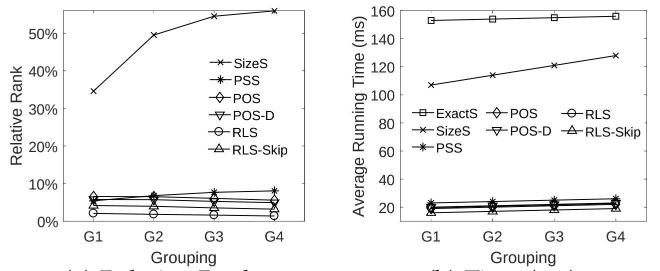
6.3.3 Efficiency Evaluation

The results of running time are shown in Figure 5 (d). We observe that RLS-Skip runs the fastest because it skips some frames and the cost of maintaining the states and making decisions for these frames is saved. ExactS has the largest running time. For example, it is usually around 7-9 times slower than PSS, POS, POS-D, RLS and RLS-Skip. SizeS runs faster than ExactS since it explores fewer plays, and PSS is slightly slower than POS, POS-D, and RLS. This is because PSS needs to compute the similarities of suffixes to make the splitting decision while the other three only compute the similarities of prefixes. We show the results of running time when varying the length of query plays in Figure 6 (b). We notice that the average running time of all the algorithms except SizeS is almost not affected by the query length. This is because the time complexity of computing a similarity is constant once the vector of the query is learned (we did not count the time for learning the representation of the query play since it is shared by all algorithms). For SizeS, the time grows as the query length increases because the candidate plays that are explored are longer and it costs more time learn their representations for measuring their similarities. Overall, the efficiency results are consistent with their time complexities as shown in Table 1.



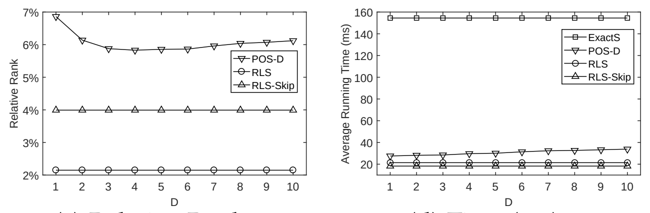
(a) Approximate Ratio (b) Mean Rank (c) Relative Rank (d) Time (ms)

Fig. 5. Results of AR, MR and RR (a)-(c) and running time (d).



(a) Relative Rank (b) Time (ms)

Fig. 6. Results of RR and running time for varying query play lengths.

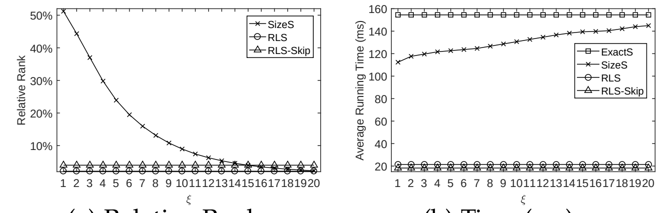


(c) Relative Rank (d) Time (ms)

Fig. 8. The effect of delay D for POS-D.

TABLE 3
The effect of skipping steps k for RLS-Skip.

Metrics	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
AR	1.007	1.010	1.012	1.015	1.027	1.034
MR	4.794	6.554	9.299	10.163	23.159	33.959
RR	2.1%	2.9%	3.5%	4.0%	6.9%	10.4%
Time (ms)	21.407	20.236	19.364	18.252	15.181	12.377
Skipped Frs	0%	0.5%	1.0%	1.4%	6.5%	9.7%



(a) Relative Rank (b) Time (ms)

Fig. 7. The effect of soft margin ξ for SizeS.

6.3.4 Parameter Study

We evaluate the effect of the soft margin ξ for SizeS. Figure 7 shows the results of RR and runing time. The results of AR and MR provide similar clues and thus are omitted due to the page limit. As expected, the effectiveness of SizeS becomes better (i.e., it approaches and exceeds RLS and RLS-Skip gradually) when ξ becomes larger. However, its running time increases and approaches that of the ExactS. This is because a larger setting of ξ indicates a larger search space, which approaches the search space of the ExactS as ξ grows.

In Figure 8, we study the effects of the parameter D for POS-D. We vary the D from 1 to 10. Overall, the D would not greatly affect the POS-D on both effectiveness and efficiency. As D increases, the effectiveness improves at the beginning because it checks more frames to perform a split operation; however, the effectiveness would drop slightly with a larger setting of D since some prefixes that would have been formed and considered if D is 0 or small would be destroyed. The efficiency drops slightly as D increases since it checks more plays during the search. We thus adopt the setting $D = 5$ as the default one since it provides a good trade-off between effectiveness and efficiency.

We further study the effects of skipping steps k for RLS-Skip in Table 3. As expected, the general trend is that the effectiveness drops but its efficiency improves as k increases since with a larger k, it tends to skip more frames, which may exclude more plays from being considered and save some computation costs. In addition, we report the statistics of the portion of skipped frames. Note that RLS-Skip with $k = 0$ corresponds to RLS. We adopt the setting $k = 3$ as the default one since it provides a reasonable trade-off between effectiveness and efficiency.

6.3.5 Case Study

We illustrate a case study to show the most similar play returned by RLS and RLS-Skip for two random query plays in Figure 9, where the blue, red and yellow lines denote the trajectories of defense players, attacking players and ball in the play, and the small "x" is the end point of the movement. In general, we observe the returned plays by RLS and RLS-Skip match the query plays very well, which is due to their data-driven nature to find a similar play.

6.4 Evaluation of Algorithms of Searching Similar Plays Within a Database

6.4.1 Compared Methods

We compare the ScoreSearch with two baseline methods. The ScoreSearch algorithm uses the RLS-Skip algorithm for finding the most similar play of a game to a query play.

(1) Full Scan. It finds the most similar play of each game in the database to a query play using RLS-Skip and then returns the one with the greatest similarity to the query play among the found plays.

(2) Random Sampling. It randomly samples a fraction of $r \cdot N$ games from the database, finds the most similar play of each sampled game to a query play using RLS-Skip, and then returns the one with the greatest similarity to the query play among the found plays.

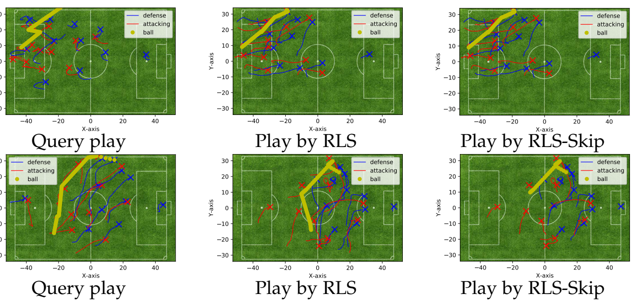


Fig. 9. Case study on similar play retrieval.

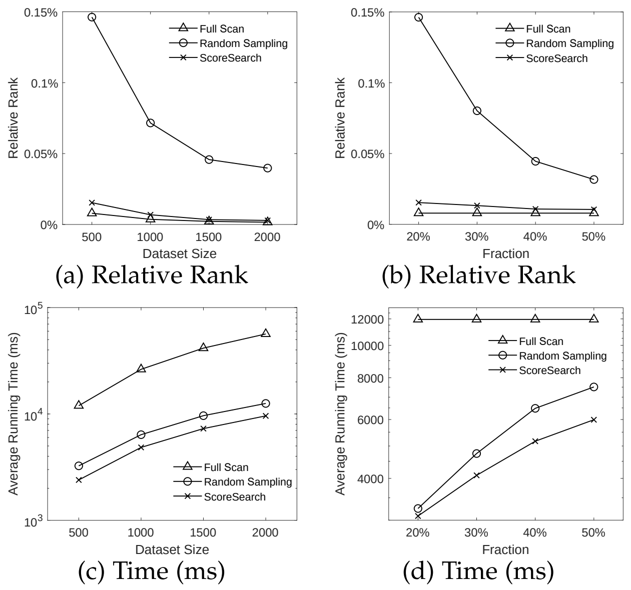


Fig. 10. Results of RR (a)-(b) and running time (c)-(d) for varying dataset size and fraction.

6.4.2 Parameter Settings and Evaluation Metrics

For training the triplet network in ScoreSearch, we randomly sample 30k triplets from the dataset and set $\delta = 1.0$ in the triplet loss based on empirical findings. To evaluate the effectiveness of ScoreSearch, we reuse the evaluation metrics of AR, MR and RR, which are calculated based on the results returned by Full Scan.

6.4.3 Effectiveness Evaluation

We study the ScoreSearch algorithm for searching similar plays in a database. We vary two parameters, i.e., dataset size and the fraction parameter r. We report the average effectiveness results in Figure 10 (a)-(b). We observe the effectiveness improves as the fraction parameter r increases. This is because with a larger r, more games that contain similar plays would be considered for searching for the similar play. In addition, the effectiveness improves slightly as the dataset size increases. This is because with a larger N, more games would be considered for searching for the similar play. As expected, ScoreSearch is much better than the Random Sampling since it filters games guided by a learning model instead of a simple random process. The results provided by AR and MR show similar trends and thus they are omitted.

6.4.4 Efficiency Evaluation

The running times of Full Scan, Random Sampling and ScoreSearch are presented in Figure 10 (c)-(d). As expected, the running time increases with a larger fraction parameter r since more games are selected from the database for conducting the search. In addition, we observe that ScoreSearch is faster than the Random Sampling. This could be explained by the fact that ScoreSearch selects shorter games compared with Random Sampling and the process of searching for a similar play from a shorter game takes less time than from a longer game.

7 RELATED WORK

Sports Data Analytics. The conventional methods for sports play retrieval are based on "keywords", which however requires the data to be annotated with keywords and users to have necessary prior knowledge on keywords. Most germane to our work is the work by Sha et al. [7], [25], which measures the similarity between two plays by first aligning trajectories from two plays (based on the extracted roles of trajectories) and then aggregating the similarities between aligned trajectories as one between the two plays. Probst et al. [26] focus on queries such as region queries based on spatiotemporal sports data. Di et al. [27] propose to extract features from sports plays by using CNNs on the visual representations of trajectories and use the extracted features together with some other features to learn a rankSVM model for serving users with specific preferences (conveyed with click-through data). Other types of sports analytics include those of detecting team formations [1], [2], identifying spatial patterns of movement [3], [4], analyzing sports videos [28] and sports prediction [5], [6]. The work [29] gives a more detailed survey on spatiotemporal sports data analytics.

Measuring Trajectory Similarity. The problem of measuring the similarity between trajectories (time series in general) has been studied extensively. DTW [19] is the first attempt towards solving the local time shift issue for computing trajectory similarity. Frechet distance [20] is a classical similarity measure that treats each trajectory as a spatial curve and takes into account the location and order of the sampling points. ERP [30] and EDR [31] are proposed to further improve the ability to capture spatial semantics in trajectories. Nevertheless, these methods are mainly based on alignment of matching sample points, and thus they are inherently sensitive to noise and varying sampling rates which exist commonly in trajectory data. To address this issue, Su et al. [23] propose an anchor-based calibration system that aligns trajectories to a set of fixed locations. Ranu et al. [22] formulate a robust distance function called EDwP to match trajectories under inconsistent and variable sampling rates. These similarity measures are usually based on the dynamic programming technique to identify the optimal alignment which leads to $O(n^2)$ computation complexity, where n is the length of the trajectories. More recently, Li et al. [24] propose to learn representations of trajectories in the form of vectors and then measure the similarity between two trajectories as the Euclidean distance between their corresponding vectors and Yao et al. [32] employ deep metric learning to accelerate trajectory similarity computation. Another related study is one studying trajectory set similarity on road networks by He et al. [21], in which the idea of the Earth Mover's Distance (EMD) is leveraged to capture both spatial and temporal characteristics of trajectories.

Querying and Mining Subtrajectories. There have been existing studies which take fragments of trajectories as units for analytics, including subtrajectory clustering [33], [34], [35], subtrajectory similarity search [9], [36] and subtrajectory join [37]. Specifically, Lee et al [35] propose a partition-and-group framework for clustering, which partitions a trajectory into many subtrajectories rather than grouping similar trajectories as a whole, and Buchin et al. [34] prove the subtrajectory clustering problem is NP-Hard and propose several approximation algorithms. Agarwal et al. [33] further apply the trajectory simplification technique to develop approximation algorithms for subtrajectory clustering based on discrete Frechet distance. Recently, Tampakis et al. [37], [38] propose a distributed solution for subtrajectory clustering and join based on the MapReduce programming model. Additionally, in our prior work [9] and the work [36] by Koide et al., the problem of searching sub-trajectories that are similar to a given trajectory is studied, where the former focuses on trajectories in free space and the latter on road networks. In [9], we study the similar sub-trajectory search problem and develop both exact and approximate algorithms. In this work, we study a counterpart problem on plays, which comprise multiple trajectories, and adapt the algorithms for the problem of splitting a *trajectory* in [9] to that of splitting a *game* in this paper.

Representation Learning. Inspired by the success of word2vec, the idea of representation learning [39] is widely used for many tasks such as natural language processing [40] and graph embedding [41]. The Skip-Gram with Negative Sampling (SGNS) model [11] is one of common methods of word2vec which is based on the assumption in linguistics that words frequently occurring in a sentence tend to share more statistical information. Seq2Seq based learning model has achieved good performance on spatiotemporal data [24], [42]. The ability to capture the local spatial correlation makes it inherently applicable to various downstream analysis tasks. Our proposed model is inspired by the Seq2Seq model and the SGNS architecture. Our play2vec model is different from those targeted in previous studies. To accelerate the training of play2vec, we design a method to generate training data with hard negative sampling. We also use a grid structure that is robust to noise and varying sampling rates.

8 CONCLUSIONS

In this paper, we study the similar play retrieval problem. This problem is to find a fragment of a game in a database, which is the most similar to a query play. We make the following technical contributions. First, we design the first deep learning based method called play2vec for computing the similarity between two sports plays. play2vec is robust to the non-uniform sampling rates and noises and runs in linear time. Second, we develop a suite of algorithms for the problem of searching for the most similar play to a query play within a game. Third, we develop a novel algorithm based on deep metric learning, ScoreSearch, for searching for the most similar play to a query play within a database. One interesting future research direction could be to conduct other analytic tasks such as play clustering based on the proposed play2vec similarity measurement.

Acknowledgments: This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG-PhD/2021-08-024[T]), by the Nanyang Technological University Start-Up Grant from the College of Engineering under Grant M4082302, by the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (RG20/19 (S)), and by a MOE Tier-2 grant MOE2019-T2-2-181.

REFERENCES

[1] A. Bialkowski, P. Lucey, P. Carr, Y. Yue, S. Sridharan, and I. Matthews, "Large-scale analysis of soccer matches using spatiotemporal tracking data," in *Data Mining (ICDM), 2014 IEEE International Conference on*. IEEE, 2014, pp. 725–730.
[2] P. Lucey, A. Bialkowski, P. Carr, S. Morgan, I. Matthews, and Y. Sheikh, "Representing and discovering adversarial team behaviors using player roles," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2706–2713.
[3] T. Decroos, J. Van Haaren, and J. Davis, "Automatic discovery of tactics in spatio-temporal soccer match data," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 223–232.
[4] A. Miller, L. Bornn, R. Adams, and K. Goldsberry, "Factorized point process intensities: A spatial analysis of professional basketball," in *International Conference on Machine Learning*, 2014, pp. 235–243.
[5] R. Aoki, R. M. Assuncao, and P. O. Vaz de Melo, "Luck is hard to beat: The difficulty of sports prediction," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 1367–1376.
[6] Y. Yue, P. Lucey, P. Carr, A. Bialkowski, and I. Matthews, "Learning fine-grained spatial models for dynamic sports play prediction," in *Data Mining (ICDM), 2014 IEEE International Conference on*. IEEE, 2014, pp. 670–679.
[7] L. Sha, P. Lucey, Y. Yue, P. Carr, C. Rohlf, and I. Matthews, "Chalkboarding: A new spatiotemporal query paradigm for sports play retrieval," in *Proceedings of the 21st International Conference on Intelligent User Interfaces*. ACM, 2016, pp. 336–347.
[8] Z. Wang, C. Long, G. Cong, and C. Ju, "Effective and efficient sports play retrieval with deep representation learning," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 499–509.
[9] Z. Wang, C. Long, G. Cong, and Y. Liu, "Efficient and effective similar subtrajectory search with deep reinforcement learning," *PVLDB*, vol. 13, no. 11, pp. 12–25, 2020.
[10] M. Kaya and H. Ş. Bilge, "Deep metric learning: A survey," *Symmetry*, vol. 11, no. 9, p. 1066, 2019.
[11] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.
[12] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.
[13] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.
[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
[15] W. Kong, C. Liaw, A. Mehta, and D. Sivakumar, "A new dog learns old tricks: Rl finds classic optimization algorithms," in *International Conference on Learning Representations*, 2018.
[16] Y. Wang, Y. Tong, C. Long, P. Xu, K. Xu, and W. Lv, "Adaptive dynamic bipartite graph matching: A reinforcement learning approach," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1478–1489.
[17] N. S. Altman, "An introduction to kernel and nearest-neighbor nonparametric regression," *The American Statistician*, vol. 46, no. 3, pp. 175–185, 1992.

[18] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[19] B.-K. Yi, H. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," in *Data Engineering, 1998. Proceedings., 14th International Conference on*. IEEE, 1998, pp. 201–208.

[20] H. Alt and M. Godau, "Computing the fréchet distance between two polygonal curves," *International Journal of Computational Geometry & Applications*, vol. 5, no. 01n02, pp. 75–91, 1995.

[21] D. He, B. Ruan, B. Zheng, and X. Zhou, *Trajectory Set Similarity Measure: An EMD-Based Approach*, 05 2018, pp. 28–40.

[22] S. Ranu, P. Deepak, A. D. Telang, P. Deshpande, S. Raghavan *et al.*, "Indexing and matching trajectories under inconsistent sampling rates," in *2015 IEEE 31st International Conference on Data Engineering (ICDE)*. IEEE, 2015, pp. 999–1010.

[23] H. Su, K. Zheng, H. Wang, J. Huang, and X. Zhou, "Calibrating trajectory data for similarity-based analysis," in *Proceedings of the 2013 ACM SIGMOD international conference on management of data*. ACM, 2013, pp. 833–844.

[24] X. Li, K. Zhao, G. Cong, C. S. Jensen, and W. Wei, "Deep representation learning for trajectory similarity computation," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 2018, pp. 617–628.

[25] L. Sha, P. Lucey, Y. Yue, X. Wei, J. Hobbs, C. Rohlf, and S. Sridharan, "Interactive sports analytics: An intelligent interface for utilizing trajectories for interactive sports play retrieval and analytics," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 25, no. 2, p. 13, 2018.

[26] L. Probst, I. Al Kabary, R. Lobo, F. Rauschenbach, H. Schuldt, P. Seidenschwarz, and M. Rumo, "Sportsense: User interface for sketch-based spatio-temporal team sports video scene retrieval," in *Proceedings of the first workshop on User Interface for Spatial and Temporal Data Analysis (UISTDA'18). CEUR-WS*, 2018.

[27] M. Di, D. Klabjan, L. Sha, and P. Lucey, "Large-scale adversarial sports play retrieval with learning to rank," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 12, no. 6, p. 69, 2018.

[28] T.-Y. Liu, W.-Y. Ma, and H.-J. Zhang, "Effective feature extraction for play detection in american football video," in *Multimedia Modelling Conference, 2005. MMM 2005. Proceedings of the 11th International*. IEEE, 2005, pp. 164–171.

[29] J. Gudmundsson and M. Horton, "Spatio-temporal analysis of team sports," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, p. 22, 2017.

[30] L. Chen and R. Ng, "On the marriage of lp-norms and edit distance," in *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. VLDB Endowment, 2004, pp. 792–803.

[31] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. ACM, 2005, pp. 491–502.

[32] D. Yao, G. Cong, C. Zhang, and J. Bi, "Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019.

[33] P. K. Agarwal, K. Fox, K. Munagala, A. Nath, J. Pan, and E. Taylor, "Subtrajectory clustering: Models and algorithms," in *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 2018, pp. 75–87.

[34] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo, "Detecting commuting patterns by clustering subtrajectories," *International Journal of Computational Geometry & Applications*, vol. 21, no. 03, pp. 253–282, 2011.

[35] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering: a partition-and-group framework," in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, 2007, pp. 593–604.

[36] S. Koide, C. Xiao, and Y. Ishikawa, "Fast subtrajectory similarity search in road networks under weighted edit distance constraints," *arXiv preprint arXiv:2006.05564*, 2020.

[37] P. Tampakis, C. Doulkeridis, N. Pelekis, and Y. Theodoridis, "Distributed subtrajectory join on massive datasets," *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, vol. 6, no. 2, pp. 1–29, 2020.

[38] P. Tampakis, N. Pelekis, C. Doulkeridis, and Y. Theodoridis, "Scalable distributed subtrajectory clustering," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, 2019, pp. 950–959.

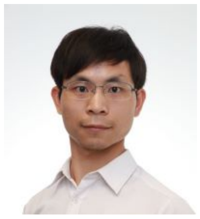
[39] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[40] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International Conference on Machine Learning*, 2014, pp. 1188–1196.

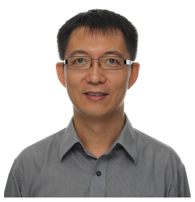
[41] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2014, pp. 701–710.

[42] Z. Wang, C. Ju, G. Cong, and C. Long, "Representation learning for spatial graphs," *arXiv preprint arXiv:1812.06668*, 2018.

Zheng Wang is currently working towards the Ph.D. degree at the School of Computer Science and Engineering (SCSE), Nanyang Technological University (NTU) under the supervision of Cheng Long and Gao Cong. His research interests are broadly in data management, data mining and machine learning. He received the Bachelor and Master degrees from Shandong University and the University of Hong Kong in 2016 and 2018, respectively.



Cheng Long is currently an Assistant Professor at the School of Computer Science and Engineering (SCSE), Nanyang Technological University (NTU). From 2016 to 2018, he worked at Queen's University Belfast, UK. He got the PhD degree from the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology (HKUST) in 2015. His research interests include data management, data mining and big data analytics.



Gao Cong is a professor at the School of Computer Science and Engineering (SCSE), Nanyang Technological University (NTU). He received his PhD degree from the National University of Singapore in 2004. He worked at Aalborg University, Microsoft Research Asia, and the University of Edinburgh. His current research interests include data management and data mining.