Similar Billiards Layout Retrieval with Deep Metric Learning [Technical Report]

Qianru Zhang[1,*], Zheng Wang[2,*], Cheng Long[2], Siu-Ming Yiu[1]
[1]Department of Computer Science, The University of Hong Kong, Hong Kong SAR
[2]School of Computer Science and Engineering, Nanyang Technological University, Singapore
{qrzhang,smyiu}@cs.hku.hk,{wang_zheng,c.long}@ntu.edu.sg

ABSTRACT

Nowadays, it becomes a common practice to capture some data of sports games with devices such as GPS sensors and cameras and then use the data to perform various analyses on sports games, including tactics discovery, similar game retrieval, performance study, etc. While this practice has been conducted to many sports such as basketball and soccer, it remains largely unexplored on the billiards sports, which is mainly due to the lack of publicly available datasets. Motivated by this, we collect a dataset of billiards sports, which includes layouts (i.e., locations) of billiards balls after the break shot and some remark information (e.g., whether all balls are cleared by the player who makes the break shot) for 3,019 international professional billiards sports matches. We then study the *similar billiards layout retrieval* problem, which is to find from a database of many billiards layouts those that are similar to a user-specified one called *query layout*. This problem can be used for many different purposes including game recommendation, prediction and clustering analysis, etc. This problem relies on a similarity measurement on billiards layouts, which is non-trivial. In this paper, we propose a novel solution equipped with deep metric learning to learn the billiards' representations, called BL2Vec, which can effectively capture the unique characteristics in a billiards layout and takes linear time wrt the number of balls. We conduct extensive experiments on the collected dataset and the results show that our BL2Vec performs effectively and efficiently.

PVLDB Reference Format:
Qianru Zhang[1,*], Zheng Wang[2,*], Cheng Long[2], Siu-Ming Yiu[1]. Similar Billiards Layout Retrieval with Deep Metric Learning [Technical Report]. PVLDB, 14(1): XXX-XXX, 2022.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://www.dropbox.com/sh/644aceaolfv4rky/AAAKhpY1yzrbq9-uo4Df3N0Oa?dl=0.

1 INTRODUCTION

Recently, many studies of mining sports data are emerging, including similar play retrieval [21, 23], tactics discovery [6], sports video detection [15], team formation detection [4, 16], sports pattern mining [6, 17], score prediction [2, 25], etc. These studies cover various sports domains, including basketball, soccer, volleyball and swimming. In this paper, we expand this line of research to billiards sports. Billiards is a popular sports, where players strike the cue (white) ball to an object (colored) ball, with the intent to drive the object ball into a pocket. Popular billiards games include 9-ball, 8-ball and Snooker.

Specifically, we study the *similar billiards layout retrieval* (or simply similar layout retrieval) problem, which is to find those billiards layouts from a database that are similar to a billiards layout at hand called *query layout*. Here, a billiards layout consists of the locations and identities of the balls (including the cue ball and the object balls) after a strike during a game. Intuitively, a billiards layout embeds rich information of a game that could help to inform many aspects of the game, e.g., what's the winning probability of a player, what's the strategy to strike the balls, and is there any historical layout that can be used as a reference point, etc.

The similar layout retrieval problem can be used in many scenarios. One scenario is to provide a real-time prediction of the winning result of a game given the latest layout, for which one intuitive solution is to make similar predictions for games with similar layouts. This prediction functionality is highly desirable by broadcasting platforms of the billiards games such as ChalkySticks [1]. Another scenario is that a billiards coach would like to analyze how well a player performs given different layouts after the first strike (called the break shot). In this scenario, the coach can retrieve those historical layouts that are similar to a given one and their performance results (e.g., winning or not) and collect some statistics based on the retrieved layouts. A third scenario could be that a billiards fan is watching a billiards game during which, he/she finds some layouts interesting and would like to find if there are similar layouts and/or games in the past - note that a game corresponds to a sequence of layouts and if so, how the players play given those layouts.

The core challenge of similar billiards layout retrieval is that of measuring the similarity between two billiards layouts, which is non-trivial due to some unique characteristics of billiards layouts. First, billiards layouts are order sensitive, e.g., in a billiards 9-ball game, the object balls are numbered from 1 to 9 and players should legally strike the object ball numbered the lowest in the layout first. Therefore, existing similarity measures on sets (or pointsets), which include (1) pairwise point-matching such as Earth Mover's Distance (EMD) [20] and Hausdorff distance [11] and (2) recently emerging learning-based techniques such as DeepSets [26] and RepSet [22], all assume order invariance and thus they are not suitable for billiards layouts. We note that existing similarity measures on sequences

*Equal contribution.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

[1]https://www.chalkysticks.com/tv

such as DTW [24] and Frechet [1], though order sensitive, are not suitable for billiards layouts either. This is because they would align the balls from two layouts solely based on their locations and totally ignore other information such as the ball numbers. Second, billiards layout data is with some domain-specific features, e.g., the types and locations of pockets, the factors deciding the difficulty of striking an object ball to a pocket such as the angle formed by the cue ball, a target object ball and a pocket, etc. These features are important for measuring similarities among billiards layouts and should be systematically captured in the similarities.

Another challenge is efficiency. A typical application scenario of similar billiards layout retrieval is to compute the similarity between a query billiards layout and many billiards layouts in the database. However, the existing matching-based methods all involve the procedure of finding an optimal alignment [1, 24], which has at least a quadratic time complexity in terms of the number of balls and would impose a big challenge when performing the similar layout retrieval on a huge database in terms of efficiency.

In this paper, we propose a novel solution called BL2Vec, which overcomes the aforementioned challenges of similar billiards layout retrieval. Regarding the effectiveness, we leverage a deep metric learning model, namely *triplet network* [13], to learn a ranking-based metric for measuring the similarities among billiards layouts. More specifically, to capture the ordered nature of the billiards layout data, we treat billiards layouts as ordered sequences, i.e., one cue ball followed by several object balls in the ascending order of their numbers and apply padding when there are missing balls (since some may have been pocketed). To capture those unique characteristics in billiards layouts, we propose to extract features from the balls and pockets and further embed them. Then, we use Convolutional Neural Network (CNN) [14] to generate a vector from the embedded features as the representation of the layout since CNN can inherently capture the local relationships between the balls. In addition, we propose a method to generate training data with hard negative mining [9] so that BL2Vec is robust to a small shift in a billiards layout and learns the similarity in a self-supervised manner. Regarding the efficiency, our method based on BL2Vec computes the similarity between two layouts as the Euclidean distance between their representations (i.e., vectors), which runs in linear time. When performing a similar layout retrieval on a database, we first learn the representations of billiards layouts once offline. Then, for a given query layout, we learn its representation and perform a *nearest neighbour* query in the representation space (i.e., the vector space), which could be efficiently accomplished with existing methods such as locality-sensitive hashing [12].

In summary, the paper makes the following contributions:

(1) We collect a billiards layout dataset. The data covers international professional billiard matches in the most recent two decades. It also has rich remarks and labels information to support various billiards-related analytics tasks. To the best of our knowledge, it would be the first publicly available dataset of billiards layouts.
(2) We propose the similar billiards layout retrieval problem and a deep learning-based similarity measure called BL2Vec for billiards layouts. To the best of our knowledge, BL2Vec is the first deep learning-based solution for learning similarity between billiards layouts. The similarity not only considers the unique characteristics in a billiards layout but also runs fast in linear time.
(3) We conduct extensive experiments on the collected 9-ball data, which demonstrate that our method consistently outperforms the baseline methods for measuring billiards layout similarity. The efficiency experiments show that our method runs up to 420× faster than the existing methods that involve pairwise point-matching for measuring the similarity between two layouts. We further conduct a user study, which validates our method with expert knowledge.

The rest of the paper is organized as follows. We review the literature in Section 2 and give the problem definition in Section 3. We present our BL2Vec model in Section 4 and report our experimental results in Section 5. We finally conclude the paper and discuss some future work in section 6.

2 RELATED WORK

(1) Sports Data Analytics. Sports data analytics is becoming a popular practice nowadays [7]. For example, similar play retrieval [21, 23] is widely used in some sports recommender systems such as ESPN and Team Stream to recommend similar soccer and basketball plays to users. Tactics discovery [6] can effectively help sports club managers and coaches to improve their team tactics when preparing for an upcoming match. Sports video detection [15] fundamentally improves users' experience of watching a sports game, and sports prediction [2, 25] aims to understand the dynamics of competitive sports and provide more entertainments and profits for sports fans. In addition, the majority of sports data that is publicly available is sourced from basketball and soccer [7]. In this paper, we focus on the similar billiards layout retrieval problem, which has not been studied before, and contribute the first billiards dataset, which consists of 3,019 break shot layouts of real-world 9-ball games with rich contextual information and semantic labels. The dataset is of independent interest and provides opportunities to conduct various analytics tasks on billiards games, including but not limited to score prediction, player performance analysis, striking tactics and strategies, etc.

(2) Measuring Pointsets Similarity. A billiards layout involves a set containing the locations of the balls, i.e., a pointset (or simply a set). Therefore, it is possible to adopt an existing pointsets similarity for billiards layouts. Existing studies of pointset similarity fall in two categories. The first category focuses on finding an optimal alignment of the points between two pointsets [11, 20]. Earth Mover's Distance (EMD) [20] and Hausdorff [11] are two typical measures in this category. EMD measures the least amount of work needed to transform one set to another. Hausdorff computes the largest distance from a point in one set to the nearest point in the other set. In general, the time complexity of computing an alignment between two sets is quadratic wrt the number of their points.

The second category aims at designing neural networks for pointsets, which receives growing research attention in recent years [3, 18, 19, 22, 26]. The idea is to learn representations of pointsets in the form of vectors and measure the similarities among pointsets based on the vectors (e.g., the Euclidean distances among

the vectors). PointNet [18] and DeepSets [26] are two classic methods in this category. They transform a pointset into a vector, which guarantees some permutation invariance of points in the set (i.e., a set has unordered nature and therefore it is invariant to permutations of its points). Specifically, PointNet uses max-pooling layers to aggregate information across the vectors of the points, while DeepSets adds up the vectors. Then, the representation of the set is fed to a standard neural network architecture (e.g., some fully-connected layers with nonlinearities) to produce the output. Further, PointNet++ [19] is proposed to capture the local structures of a set by applying PointNet recursively and achieves a more efficient and robust set representation. RepSet [22] handles the set representation by computing the correspondences between an input set and some generated hidden sets using a bipartite matching algorithm. More recently, WSSET [3] employs deep metric learning based on EMD to learn set representations and measure the similarity between two pointsets as the Euclidean distance between their corresponding vectors and achieve the state-of-the-art results for measuring pointsets similarity in a self-supervised setting. Nevertheless, these studies target general pointsets, which are not order sensitive and cannot capture unique characteristics of billiards layouts.

(3) Deep Metric Learning. Deep metric learning is proposed to learn a distance function for measuring the similarity between two objects via neural networks. Bromley et al. [5] pioneer the classic Siamese network in deep metric learning and employ it for signature verification. Then, the triplet network architecture is proposed for image retrieval, which learns a ranking-based metric and shows superior performance over Siamese network [13]. In this paper, we propose to learn representations of billiards layouts via a deep metric learning method called BL2Vec, which is quite different from those in previous studies. Specifically, our BL2Vec is designed for the spatial layout of billiards based on a triplet architecture of CNNs. The reason why we use the CNN is that it has the inherent ability to capture local spatial correlations of balls in a billiards layout.

3 PROBLEM DEFINITION

A billiards game refers to a game playing with a cue stick on a rectangular billiards table, which has six pockets at the four corners and in the middle of each long side. The game involves a cue ball (white) and some object balls (colored), where different billiards games have different numbers of object balls such as 9 object balls in 9-ball game. The game begins with one player's break shot. If a ball is legally pocketed after the break shot, the player keeps his turn; otherwise, the opponent gets a chance to shoot. There are usually some rules to follow when striking a ball, e.g, in a 9-ball game, a player must use the cue ball (white) to strike the object balls (colored) in an ascending order of their numbers.

A billiards layout $\mathcal{B}$ as shown in Figure 1 corresponds to a rectangular pool table with six pockets and a set of balls O, where each ball $o \in O$ is associated with a spatial location (x, y) on the table.

Given two billiards layouts $\mathcal{B}_1$ and $\mathcal{B}_2$, we denote their similarity as $sim(\mathcal{B}_1, \mathcal{B}_2)$. Intuitively, the similarity function $sim(\cdot)$ should capture the locations of balls on the table. Figure 1 illustrates an example of designing the similarity. $\mathcal{B}_q$ denotes a query layout, which contains a cue ball and three object balls 2,7,9. There are three different candidate layouts $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$. Among these candidate layouts, $\mathcal{B}_3$ is the most dissimilar to $\mathcal{B}_q$ since $\mathcal{B}_3$ contains totally different object balls (i.e., ball 3,4,6). Both $\mathcal{B}_1$ and $\mathcal{B}_2$ contain the same balls as $\mathcal{B}_q$; however, it is clear that the locations of the balls in $\mathcal{B}_1$ are more similar to those in $\mathcal{B}_q$ since these balls are located near to each other as they are in $\mathcal{B}_q$, while the balls in $\mathcal{B}_2$ are scattered. Therefore, while we do not know the exact values of these similarities, we tend to know a ranking of them (i.e., $sim(\mathcal{B}_q, \mathcal{B}_1) > sim(\mathcal{B}_q, \mathcal{B}_2) > sim(\mathcal{B}_q, \mathcal{B}_3)$).

Problem statement. Given a database of billiards layouts $\mathcal{D}$, we aim to map each billiards layout $\mathcal{B} \in \mathcal{D}$ to a low-dimensional space to capture similarities among billiards layouts. More specifically, for any two given billiards layouts $\mathcal{B}_1$ and $\mathcal{B}_2$, we map them to two K-dimensional vectors v_1 and v_2 via deep metric learning. The learned vectors should be similarity preserving, i.e., if two billiards layouts are similar, the Euclidean distance between their vectors would be small.

4 METHODOLOGY

4.1 Overview of BL2Vec

Given two billiards layouts, it is hard to tell if they are similar or dissimilar. However, it could be noticed from the example in Figure 1 that given a query layout and two candidate layouts, it is much easier to distinguish which one is more similar to the query layout. Motivated by this, we propose a deep metric learning model based on *triplet network* [10] to learn a similarity measure for billiards layouts. We call the model *BL2Vec* and show its overview in Figure 2. Specifically, we randomly sample a billiards layout from the dataset as an anchor layout $\mathcal{B}_q$. We then generate a positive sample $\mathcal{B}_+$ and a negative sample $\mathcal{B}_-$ by injecting less and more noise to $\mathcal{B}_q$, respectively. This is based on the intuition that a billiards layout, when injected with less noise, should be more similar to the original layout. We then extract features x_q (resp. x_+ and x_-) from the generated layouts $\mathcal{B}_q$ (resp. $\mathcal{B}_+$ and $\mathcal{B}_-$) and fed them to a triplet network. The triplet network consists of three instances of a shared feedforward neural network, denoted as $Net(\cdot)$, and outputs two Euclidean distances as follows: $d_+ = ||Net(x_q) - Net(x_+)||_2$ and $d_- = ||Net(x_q) - Net(x_-)||_2$, where $Net(x_q)$ denotes the embedded representation of x_q via the network (i.e., a vector). We adopt the following loss for the triplet network:

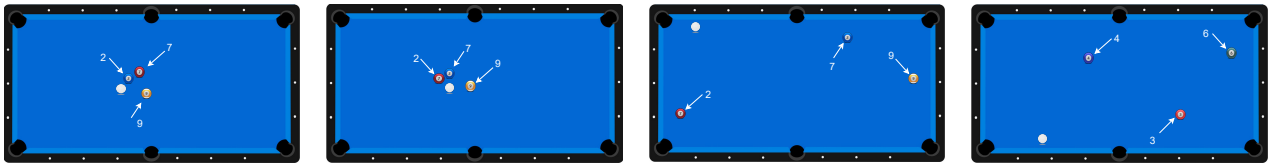
$$\mathcal{L}(x_q, x_+, x_-) = \max\{d_+ - d_- + \delta, 0\}, \tag{1}$$

where δ is a margin between the positive and negative distances. By optimizing this loss, it learns to correctly distinguish $\mathcal{B}_+$ and $\mathcal{B}_-$.

Next, we present the details of BL2Vec and analyze the time complexity of computing the similarity between two layouts with BL2Vec.

4.2 Positive and Negative Layouts Generation

Positive billiards layout $\mathcal{B}_+$. We generate $\mathcal{B}_+$ by conducting two operations on $\mathcal{B}_q$, including (1) randomly shifting the locations of the balls to some extent and (2) randomly removing some balls and adding them back at random locations. We control the generation process with two parameters, namely a noise rate and a drop rate and ensure that no balls overlap. More specifically, setting



(a) $\mathcal{B}_q$ (cue ball and ball 2,7,9) (b) $\mathcal{B}_1$ (cue ball and ball 2,7,9) (c) $\mathcal{B}_2$ (cue ball and ball 2,7,9) (d) $\mathcal{B}_3$ (cue ball and ball 3,4,6)

Figure 1: An example of illustrating the billiards layout similarity, where $sim(\mathcal{B}_q, \mathcal{B}_1) > sim(\mathcal{B}_q, \mathcal{B}_2) > sim(\mathcal{B}_q, \mathcal{B}_3)$.

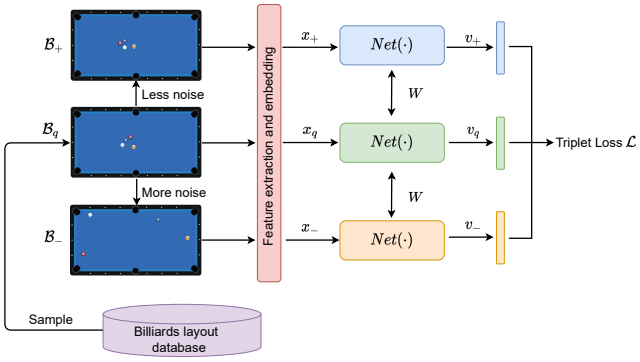


Figure 2: Overview of BL2Vec.

the noise rate as α means moving each ball along the x axis and the y axis towards a random direction and by a random distance, which is bounded by α times the billiards table's length and width, respectively. Setting the drop rate as β means randomly deleting $\beta \cdot n'$ balls rounded with a ceiling function and then adding them at random locations on the billiards table, where n' is the number of balls in the current layout. The two parameters of noise rate and shift rate would be studied in experiments.

Negative billiards layout $\mathcal{B}_-$. One intuitive idea is to randomly sample a billiards layout that is not the same as $\mathcal{B}_q$ to be $\mathcal{B}_-$ from the dataset. However, it suffers from the issue that the randomly sampled $\mathcal{B}_-$ would be generally very dissimilar to $\mathcal{B}_q$, and thus distinguishing $\mathcal{B}_+$ and $\mathcal{B}_-$ becomes trivial. In this case, the model would converge with inferior performance. To address this issue, we propose to generate the $\mathcal{B}_-$ based on $\mathcal{B}_q$ in the same way as we generate $\mathcal{B}_+$, but with more noise (i.e., larger noise rate and drop rate). In summary, both $\mathcal{B}_+$ and $\mathcal{B}_-$ are noisy versions of $\mathcal{B}_q$, where $\mathcal{B}_+$ is more similar to $\mathcal{B}_q$ since it contains less noise than $\mathcal{B}_-$.

4.3 Billiards Layout Feature Extraction

We extract the following types of features from a billiards layout, namely Ball-Self (BS), Ball-Pocket (BP) and Ball-Ball (BB). Figure 3(a)-(b) shows an example of the features extracted from a 9-ball layout.

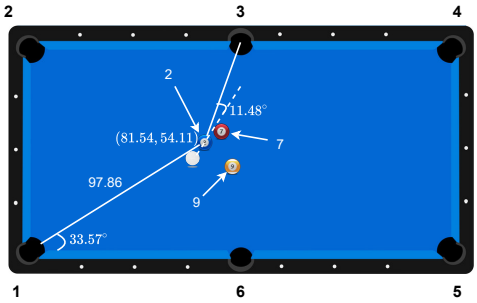
Ball-Self (BS). We standardize the play field of the billiards table into a 200×100 coordinate system, and take the bottom left corner as the origin. Thus, the coordinates belong to the $[0, 200]$ range along the x-axis, and the $[0, 100]$ range along the y-axis. For each ball b_i in the layout, it is associated with a location feature (x, y). For example, the location feature of Ball 2 is $(81.54, 54.11)$ in Figure 3(a).

Ball-Pocket (BP). To extract the features concerning balls and pockets, we take the pocket at the bottom left corner with the location $(0, 0)$ as the first one and mark the pockets clockwise from 1 to 6 as shown in Figure 3(a). For each ball b_i, we extract 4 features, namely *cushion angle*, *pocket distance*, *occlusion indicator* and *pocket index*, wrt each pocket $p_j (1 \le j \le 6)$. Thus, there are in total $4 \times 6 = 24$ features for each ball b_i. We describe the details of these features as follows. (1) *Cushion angle*: it denotes the minimum angle between the straight line linking the ball b_i and pocket p_j, denoted by $\overline{b_i p_j}$, and one of p_j's two neighboring cushions. For example, in Figure 3(a), the cushion angle is 33.57°, which is the angle between $\overline{b_2 p_1}$ and its neighboring cushion linking Pocket 1 and 6. (2) *Pocket distance*: it denotes the distance of the ball b_i and pocket p_j, i.e., $||b_i - p_j||_2$. In Figure 3(a), the pocket distance of Ball 2 and Pocket 1 is 97.86. Intuitively, the features (1) and (2) capture the feasibility to strike an object ball into a pocket. For example, when the cushion angle is bigger and the pocket distance is smaller, the object ball b_i is more likely to be pocketed in the pocket p_j. (3) *Occlusion indicator*: it indicates whether there is a ball located on the line linking the ball b_i and pocket p_j (i.e., $\overline{b_i p_j}$). For example, in Figure 3(a), the indicator is 1 for the object ball b_2 and the pocket p_1 since the cue ball is on $\overline{b_2 p_1}$. The indicator captures whether there are collision cases which may appear in the path from the object ball b_i to the pocket p_j. (4) *Pocket index*: the index of a pocket that is underlying each of above three BP features.

Ball-Ball (BB). We extract two features concerning the relationships among balls. (1) *Shot angle*: it denotes the minimum angle between the *cue orientation* for a ball b_i and the straight line linking the ball b_i and a pocket p_j (i.e., $\overline{b_i p_j}$ where $1 \le j \le 6$). Here, the cue orientation for a ball b_i is (1) the direction from the cue ball to b_i (in the case that b_i is the ball with the smallest number among all balls on the table) or (2) the direction from $b_{i'}$ to b_i with $b_{i'}$ being the ball with the largest number that is smaller than i (in the other case). The intuition for the latter case is that the cue ball will be located near $b_{i'}$'s location once $b_{i'}$ is pocketed and the player will target b_i next. For example, in Figure 3(a), the shot angle of ball 2 is 11.48°, where the cue orientation is $\overline{b_1 b_2}$. (2) *Pocket index*: the index of the pocket corresponding to b_i's shot angle. In the above example, the pocket index for Ball 2 is 3 since the shot angle is computed based on the pocket 3.

4.4 Billiards Layout Feature Embedding

A straightforward method for embedding the extracted features of a layout is to simply use the extracted feature values of BS, BP and BB as shown in Figure 3(b) directly; however, this method would restrict the embeddings in a low dimensional space, which makes it difficult for the loss function to further optimize the embeddings in their parameter space.



(a) A layout $\mathcal{B}_q$ (features of Ball 2)

	BS		BP					BB	
Cue ball	76.01	45.35	30.82	88.51	0	1	⋯	0	0
< PAD >	0	0	0	0	0	0	⋯	0	0
Ball 2	81.54	54.11	33.57	97.86	1	1	⋯	11.48	3
< PAD >	0	0	0	0	0	0	⋯	0	0
< PAD >	0	0	0	0	0	0	⋯	0	0
< PAD >	0	0	0	0	0	0	⋯	0	0
< PAD >	0	0	0	0	0	0	⋯	0	0
Ball 7	89.73	59.11	33.38	107.45	1	1	⋯	11.48	4
< PAD >	0	0	0	0	0	0	⋯	0	0
Ball 9	94.76	42.05	23.93	103.67	0	1	⋯	8.11	6

(b) BS, BP and BB features

	BS	BP				BB	
Cue ball	48	3	9	0	1	⋯	0	0
< PAD >	0	0	0	0	0	⋯	0	0
Ball 2	48	3	10	1	1	⋯	1	3
< PAD >	0	0	0	0	0	⋯	0	0
< PAD >	0	0	0	0	0	⋯	0	0
< PAD >	0	0	0	0	0	⋯	0	0
< PAD >	0	0	0	0	0	⋯	0	0
Ball 7	48	3	11	1	1	⋯	1	4
< PAD >	0	0	0	0	0	⋯	0	0
Ball 9	35	2	11	0	1	⋯	1	6

(c) The token $\mathcal{T}_q$

Figure 3: An example of illustrating the feature extraction from a 9-ball layout $\mathcal{B}_q$.

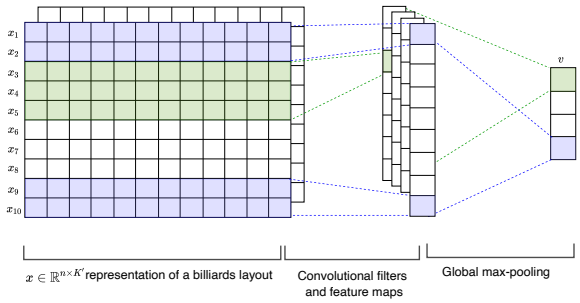


Figure 4: Architecture of $Net(\cdot)$, where the colours (i.e., purple and green) mean the filters with different sizes.

We propose to first map the continuous BS, BP and BB features into discrete tokens, denoted by $\mathcal{T}$. More specifically, (1) for ball location, we partition the billiards table into cells of equal size and treat each cell as a token, e.g., as shown in Figure 3(c), the cell size is set to 15×15 in the BS column of token $\mathcal{T}_q$; (2) for angle (i.e., *cushion angle* and *shot angle*) and distance (i.e., *pocket distance*), we partition the ranges with a predefined granularity, e.g., as shown in Figure 3(c), we use the granularity of $15°$ and 10 to partition the angle and distance ranges, respectively; (3) for *occlusion indicator* and *pocket index*, they are naturally in the form of tokens. We then obtain a vector for each layout by embedding the tokens as one-hot vectors and concatenating the vectors in an ascending order of the ball numbers, denoted by $x \in \mathbb{R}^{n \times K'}$, where K' is the dimension of the embedded space and n denotes the total number of balls in a billiards game to which the layout belongs, such as $n = 10$ for 9-ball layouts.

4.5 Model Architecture of $Net(\cdot)$

We adopt a convolutional neural network (CNN), as shown in Figure 4, for $Net(\cdot)$ since the CNN is inherently appropriate for perceiving the layout, and preserves the spatial correlation of the balls that form the layout.

To feed a layout $\mathcal{B}$ into $Net(\cdot)$, we use its embedded features $x \in \mathbb{R}^{n \times K'}$. Let x_i be a K'-dimensional embedding vector corresponding to the ball with the number i in the layout. Then, a billiards layout could be represented as $x_{1:n} = x_1 \oplus x_1 \oplus ... \oplus x_n$, where $\oplus$ is the concatenation operator. We denote by $x_{i:i+j}$ the concatenation of $x_i, x_{i+1}, ..., x_{i+j}$. A convolution operation [27], which involves a *filter* $\boldsymbol{W} \in \mathbb{R}^{h \times K'}$, is applied to a window of h balls to extract a new feature. Specifically, a feature c_i is extracted from a window of balls $x_{i:i+h-1}$ as follows.

$$c_i = \sum_{l=i}^{i+h-1} \sum_{r=1}^{K'} \boldsymbol{W}_{l-i+1,r} \cdot x_{l,r} + \boldsymbol{b},$$

where $b \in \mathbb{R}$ denotes a bias term. Then, the filter is applied to each possible window of balls in the layout $\{x_{1:h}, x_{2:h+1}, ..., x_{n-h+1:n}\}$ to produce a feature map as $c = [c_1, c_2, ..., c_{n-h+1}]$, where $c \in \mathbb{R}^{n-h+1}$. We then apply a global max-pooling layer over the feature map and take the maximum value $\hat{c} = \max_{1 \le i \le n-h+1} \{c\}$ as the feature corresponding to this particular filter. The intuition is that it can capture the most important feature (i.e., one with the highest value) for each feature map, and deals with variable lengths of different feature maps. Note that each feature $\hat{c}$ is extracted from each convolution filter, and we use multiple filters with varying widths to obtain multiple features. Then, these features are concatenated to be a K-dimensional vector v as the representation of a billiards layout.

4.6 Time Complexity Analysis

We can train the BL2Vec model by repetitively sampling an anchor layout from the database, computing the corresponding loss in Equation (1), and adjusting the parameters of the model via some optimizers such as Adam stochastic gradient descent until some criterion is satisfied (e.g., the model shows no improvement on a validation set). Once the model has been trained, we use it to embed a billiard layout to a vector and compute the similarity between two layouts based on the Euclidean distance between their embedded vectors. Specifically, the time complexity of computing the similarity between two billiards layouts based on BL2Vec includes two parts, namely the embedding part and the distance computation part. For embedding, the time complexity is $O(n)$, where n denotes the total number of balls in a billiards game to which the layout belongs. We explain as follows: (1) It takes $O(n')$ to extract features from a layout and map the features to the corresponding token, where n' denotes the number of balls in the current layout, and it is bounded by n. (2) It takes $O(n)$ to get the embedded features with padding the missing balls. (3) It takes roughly $O(n)$ to map the embedded features to the target K-dimensional vector representation via the classical convolutions [8]. For distance computation, it

costs $O(K)$ which is obvious. Hence, the overall time complexity is $O(n+K)$.

5 EXPERIMENTS

5.1 Experimental Setup

Dataset. We collected a real-world billiards layouts dataset, which consists of 3,019 break shots in 94 international professional 9-ball games played by 227 players. For each layout, we have the positions of balls that remain on the pool table after the break shot, the number of potted balls after the break shot, a binary tag indicating clear or not (clear if the player who performs the break shot pocketed all balls and thus win the game; unclear otherwise), win or not (in a 9-ball game, win if the player potted ball 9 into the pocket, not win otherwise), and remarks. Remarks are used to capture some special situations, such as golden break, which means the ball 9 is potted into a pocket in the break shot; a foul shot, which means the shot is not making the first contact with the lowest numbered ball in the layout, and it is regarded as breaking the rule of competition. In addition, we provide the YouTube video link for each layout. In the dataset, there are 1,216 (resp. 1,295) billiards layouts with the label "clear" (resp. "win").

Baseline. To evaluate the effectiveness and efficiency of our BL2Vec, we compare it with the following baselines:

•EMD [20]. The Earth Mover's Distance (EMD) is based on the idea of a transportation problem that measures the least amount of work needed to transform one set to another.

•Hausdorff [11]. Hausdorff distance is based on the idea of point-matching and computes the largest distance among all the distances from a point in one set to the nearest point in the other set.

•DTW [24] and Frechet [1]. DTW and Frechet are two widely used similarity measurements for sequential data. To apply the measurements to billiards layouts, we model each layout as a sequence of locations, which corresponds to one cue ball followed by several object balls in ascending order of ball numbers.

•Peer Matching (PM). We adopt a straightforward solution for measuring the similarity between two billiards layouts as the average of the distances between their balls matched based on the numbers, e.g., ball 1 in one layout is matched to the ball 1 in another layout, and so on. For those mismatched balls with different numbers, we align them in the ascending order of their numbers and match them accordingly.

•DeepSets [26]. DeepSets is used to generate pointsets representation in an unsupervised fashion. The main idea of DeepSets is to transform vectors of points in a set into a new representation of the set. It aggregates the representations of vectors and then passes them to fully-connected layers to get the representation.

•RepSet [22]. RepSet is also used to generate the representations of pointsets. It adapts a bipartite matching idea by computing the correspondences between an input set and some generated hidden sets to get the set representation. Besides, RepSet captures the unordered nature of a set and can generate the same output for all possible permutations of points in the set.

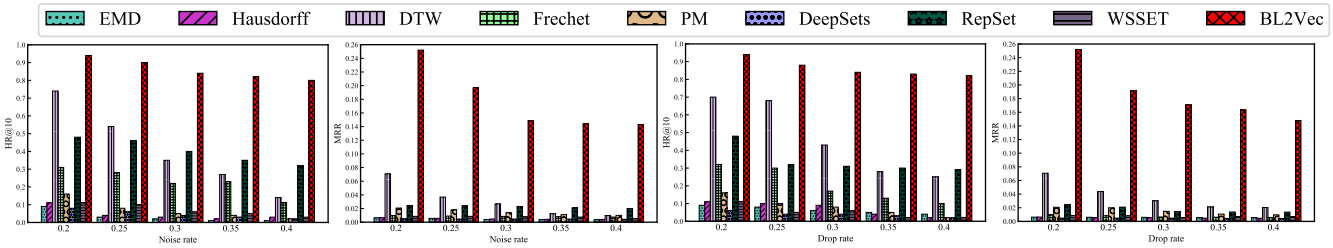
•WSSET [3]. WSSET achieves the state-of-the-art performance on pointsets embedding, which is an approximate version of EMD based on the deep metric learning framework and uses the EMD as a metric for inferring the positive and negative pairs. It generates the representations of pointsets, and these generated representations can be used for similarity search, classification tasks, etc.

Parameter Setting. We set the cell size to be 15 (with the results of its effect shown later on), and thus we have 99 unique tokens for the BS features. For BP and BB features, we use the setting of 15° and 10 for partitioning the angle range and distance range, respectively. We tried different settings, the results are similar and thus omitted. For each feature, we embed it to a 10-dimensional vector, and thus the representation dimension of the ball is $10 * (1 + 4 * 6 + 2) = 270$ (i.e., $K' = 270$). Recall that for each ball, there is one token in the BS feature, four tokens in the BP feature with totally six pockets, and two tokens in the BB feature. We use a convolutional layer as $Net(\cdot)$ with 3 output channels. For each, it has 52 filters with varying sizes from $(1, K')$ to $(7, K')$. Note that larger filters help capture the correlations between the balls. A ReLU function is applied after the convolutional layer and then a global max-pooling layer is employed to generate a 156-dimensional representation of the billiards layout (i.e., $K = 3 \times 52 = 156$). Additionally, in the training process, we randomly sample 70% billiards layouts to generate training tuples and adopt Adam stochastic gradient descent with an initial learning rate of 0.00001, and the remaining layouts are used for testing. To avoid overfitting, we employ a L2 regularization term with $\lambda = 0.001$ for $Net(\cdot)$. The margin in the triplet loss is set to $\delta = 1.0$ based on empirical findings. For generating each $\mathcal{B}_+$ (resp. $\mathcal{B}_-$), we set both the noise rate and the drop rate to 0.2 (resp. 0.3) by default. For parameters of baselines, we follow their settings in the original papers.

Evaluation Metrics. We consider the following three tasks to evaluate the effectiveness. (1) Similarity search, we report the Hitting Ratio for Top-10 (HR@10) and Mean Reciprocal Rank (MRR). In particular, for a query billiards layout $\mathcal{B}_q$, if its positive version $\mathcal{B}_+$ is in the top-10 returned billiards layouts, then HR@10 is defined as $HR@10 = 1$, otherwise $HR@10 = 0$. MRR is defined as $MRR = \frac{1}{rank}$, where $rank$ denotes the rank of $\mathcal{B}_+$ in the database for its query billiards layout $\mathcal{B}_q$. The average results of HR@10 and MRR over all queries are reported in the experiments. A higher HR@10 or MRR indicates a better result. (2) Classification, we report the average classification accuracy for the label of "clear" and "not clear". Classification accuracy is a widely used metric in the classification task. A higher accuracy indicates a better result. (3) Clustering, we report two widely used metrics Adjusted Rand Index (ARI) and Adjusted Mutual Information (AMI) for clustering. They measure the correlation between the predicted result and the ground truth. The ARI and AMI values lie in the range $[-1, 1]$. For ease of understanding, we normalize the values in the range of $[0, 1]$. A higher ARI or AMI indicates a higher correspondence to the ground-truth.

Evaluation Platform. All the methods are implemented in Python 3.8. The implementation of $BL2Vec$ is based on PyTorch 1.8.0. The experiments are conducted on a server with 10-cores of Intel(R) Core(TM) i9-9820X CPU @ 3.30GHz 64.0GB RAM and one Nvidia GeForce RTX 2080 GPU. The datasets and codes can be downloaded via the link [2].

[2] https://www.dropbox.com/sh/644aceaolfv4rky/AAAKhpY1yzrbq9-uo4Df3N0Oa?dl=0



(a) Varying noise rate (HR@10) (b) Varying noise rate (MRR) (c) Varying drop rate (HR@10) (d) Varying drop rate (MRR)

Figure 5: Effectiveness evaluation with similarity search.

Table 1: Effectiveness evaluation with classification and clustering.

Tasks	Classification	Clustring	
	Accuracy (%)	ARI (%)	AMI (%)
EMD	66.33	52.15	55.81
Hausdorff	59.17	51.68	54.98
DTW	59.33	50.12	50.13
Frechet	59.16	50.16	50.22
PM	54.24	50.02	50.32
DeepSets	54.23	49.50	50.41
RepSet	58.99	50.54	50.72
WSSET	56.62	49.12	49.78
BL2Vec	**72.62**	**61.94**	**63.26**

5.2 Experimental Results

(1) Effectiveness evaluation (similarity search). The lack of ground truth makes it a challenging problem to evaluate the similarity. To overcome this problem, we follow recent studies [3] which propose to use the *HR*@10 and *MRR* to quantify the similarity evaluation via self-similarity comparison. Specifically, we randomly select 100 billiards layouts to form the query set (denoted as Q) and 500 billiards layouts as the target database (denoted as D), For each billiards layout $\mathcal{B}_q \in Q$, we create its positive version denoted as $\mathcal{B}_+$ by introducing two types of noises. As discussed in Section 4.2, (1) we shift each ball along a random direction by a random distance, which is controlled by a noise rate; (2) we randomly delete some balls controlled by a drop rate and then add them at random locations. Then for each $\mathcal{B}_q$, we compute the rank of $\mathcal{B}_+$ in the database $D \cup \mathcal{B}_+$ using BL2Vec and other baselines. Ideally, $\mathcal{B}_+$ should be ranked at the top since $\mathcal{B}_+$ is generated from the $\mathcal{B}_q$.

Figure 5 (a) and (b) show the results of varying the noise rate from 0.2 to 0.4 when the drop rate is fixed to 0.2 in terms of *HR*@10 and *MRR*. We can see that *BL2Vec* is significantly better than the baselines in terms of *HR*@10 and *MRR*. For example, when applying 40% noise, BL2Vec outperforms RepSet and DTW (the two best baselines) by 1.5 times and 4.7 times in terms of *HR*@10, respectively, and by more than 6.2 times and 13.3 times in terms of *MRR*, respectively. EMD and DeepSets perform the worst in most of the cases because EMD is based on the idea of point-matching and affected significantly by the noise; DeepSets is developed for a set that has an unordered nature and therefore it is not suitable for measuring the similarity among billiards layouts, which are order sensitive. Figure 5 (c) and (d) show the results of varying the drop rate from 0.2 to 0.4 when the noise rate is fixed to 0.2 in terms of *HR*@10 and *MRR*. BL2Vec still performs the best in terms of both metrics. For example, it outperforms RepSet, which performs best in baselines when the drop rate is 0.4, by more than 1.8 times (resp. 9.6 times) in terms of *HR*@10 (resp. *MRR*).

Table 2: The effect of the cell size.

Cell size	#tokens	Similarity		Classification	Clustring	
		HR@10	MRR	Accuracy (%)	ARI (%)	AMI (%)
10	200	0.92	0.240	66.67	52.25	55.80
15	98	0.94	0.252	**72.62**	**61.94**	**63.26**
20	50	0.95	0.259	67.33	56.96	58.25
25	32	0.96	0.303	70.14	55.21	57.37
30	28	**0.96**	**0.306**	68.72	53.96	56.28

(2) Effectiveness evaluation (classification). Table 1 shows the results of different methods in the classification task for predicting the labels of "clear" and "not clear". Recall that "clear" means the player who performs the break shot pocketed all balls and thus win the game, and "not clear" means the otherwise case. We train a k-nearest neighbour classifier (kNN) with $k = 10$ for this task and report the average accuracy on 500 billiards layouts. For BL2Vec, we fix the noise rate and drop rate to be 0.2 by default, and use the representations of billiards layouts for computing the distance in the KNN classifier, we do the same for DeepSets, RepSet and WSSET. For EMD, Hausdorff, DTW, Frechet and PM, we use their distances for the KNN classifier directly. We observe BL2Vec outperforms all the baselines. For example, it outperforms EMD (the best baseline) by 9.5%. The result indicates that the representation learned from BL2Vec is more useful for learning the target task.

(3) Effectiveness evaluation (clustering). We further show the clustering task, which is to group billiards layouts belonging to the same family (i.e., "clear" or "not clear") into the same cluster. We use the K-means algorithm with $k = 2$ and report ARI and AMI on the 500 billiards layouts. The results are shown in Table 1. According to the results, BL2Vec also performs best. For example, it outperforms EMD (the best baseline) by 18.8% (resp. 13.3%) in terms of ARI (resp. AMI). In addition, we observe other baselines have similar ARI and AMI, e.g., the values are between 49% and 50%. Similar to the classification task, the results indicate adopting the embeddings of BL2Vec for billiards layout clustering will yield superior performance.

(4) Effectiveness evaluation (parameter study). We next evaluate the effect of the cell size on the effectiveness of similarity search, classification and clustering for BL2Vec. Intuitively with the smaller cell size, it generates more tokens and provides a higher resolution of the pool table. However, with the smaller size, it reduces the robustness against the noise for the similarity search task. In Table 2, we report the performance for the similarity search, classification and clustering. We notice that the performance of the similarity search becomes better as the cell size grows and this is in line with our intuition. In addition, we observe the smallest cell size of 10 leads to the worst performance, especially for classification and

Table 3: Effectiveness evaluation with ablation study.

Model	Similarity		Classification	Clustering	
	HR@10	MRR	Accuracy (%)	ARI (%)	AMI (%)
BL2Vec	**0.94**	**0.252**	**72.62**	**61.94**	**63.26**
w/o BS	0.92	0.207	62.42	53.50	59.25
w/o BP	0.11	0.009	51.08	49.27	50.18
w/o BB	0.91	0.197	69.52	52.45	61.73
w/o $Net(\cdot)$	0.40	0.078	41.76	50.38	50.67

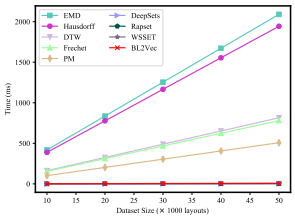


Figure 6: Scalability.

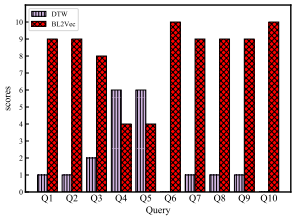


Figure 7: User study.

clustering, because the smallest cell size produces more tokens, which makes the model more difficult to train. In our experiments, we set the cell size to 15 because it provides better performance in general.

(5) Effectiveness evaluation (ablation study). To show the importance of each component of the extracted features in our BL2Vec model, including Ball-self (BS), Ball-Pocket (BP) and Ball-Ball (BB), and the ability of CNN for capturing spatial features in $Net(\cdot)$, we conduct an ablation experiment. We denote our BL2Vec model without BS, BP, BB and $Net(\cdot)$ as w/o BS, w/o BP, w/o BB and w/o $Net(\cdot)$, respectively. Table 3 compares the different versions for similarity search, classification and clustering tasks. Overall, all these components help improve the effectiveness of the BL2Vec model and enable the model to achieve superior performance than baselines. A detailed discussion on the effectiveness of each component is given as follows. (1) BS features can effectively capture the spatial information of each ball; if the features are omitted, the classification performance drops significantly by around 14%. (2) BP features capture the correlations between the ball and each pocket, and these are the unique characteristics in the billiards layout data. In addition, BP features are associated with the most tokens (i.e., $4 \times 6 = 24$) in the total of 27 tokens to represent each ball as discussed in Section 4.3, and therefore the BP features contribute the most for all of the tasks. As expected, we observe that if the BP features are omitted, the performance of similarity search drops by 88% (resp. 96%) in terms of HR@10 (resp. MRR); that of classification drops by 30% in terms of accuracy; that of clustering drops by 20% (resp. 21%) in terms of ARI (resp. AMI). (3) BB features capture the correlation between balls in the layout. The features also affect the overall performance. For example, if the BB features are omitted, the clustering performance drops a lot by around 15% (resp. 2%) in terms of ARI (resp. AMI). (4) The CNN in $Net(\cdot)$ can well capture the correlation of those spatial features (i.e., BS, BP and BB), and simply using the average of feature embeddings cannot lead to the superior performance, e.g., similarity search drops by 57.4% (resp. 69.05%) in terms of HR@10 (resp. MRR).

(6) Scalability. To test scalability, we generate more layouts by adding noise on the original dataset. Figure 6 reports the average running time over 100 queries for computing the similarity between a query billiards layout and the billiards layouts when we vary the size of the target database from 10,000 layouts to 50,000 layouts. Clearly, EMD and Hausdorff perform extremely slow, which match their quadratic time complexity. We observe the running time of DTW and Frechet is smaller than that of EMD and Hausdorff though all of them have the same time complexity. This is because DTW and Frechet compute the similarity via dynamic programming to find an optimal pairwise point-matching. PM is faster than DTW and Frechet because it matches the balls peer-to-peer based on their numbers and the complexity is linear.In addition, DeepSets, WSSET, Rapset and BL2Vec have a similar running time. Specifically, they run up to 420 times faster than EMD and Hausdorff, 160 times faster than DTW and Frechet and 100 times faster than PM. Moreover, we notice that DeepSets, WSSET, Rapset and BL2Vec scale linearly with the dataset size and the disparity between them increases as the size of the target database grows. This is because all of the learning-based methods have linear time complexity to compute the similarity by embedding layouts to vectors, and the embedding process can also be done offline, which is consistent with their time complexities.

(7) User study. Since DTW outperforms baselines in the most of cases for the similar billiards layout retrieval task, we conduct a user study to compare the BL2Vec with DTW. We randomly select 10 billiards layouts as the queries as shown in Figure 8, and for each query in the query set, we use BL2Vec and DTW to retrieve Top-1 billiards layouts from the target database. We invite ten volunteers with strong billiards knowledge to annotate the relevance of the retrieved results. More specifically, we firstly spent 15 minutes introducing the background to the volunteers so that they understand these queries. Then for each of 10 queries, the volunteers specify the more similar result from the Top-1 result retrieved by BL2Vec and the Top-1 result retrieved by DTW. Note that the volunteers do not know which result is returned by which method in advance. We report the scores of the ten queries for both methods in Figure 7. We observe that our BL2Vec outperforms DTW with expert knowledge. In particular, BL2Vec outperforms DTW for 8 queries out of the total 10 queries. In addition, BL2Vec gets 81% votes while DTW gets only 19% votes. We illustrate the Top-5 results of BL2Vec and DTW for the query Q1 in Figure 9, and the Top-1 results of BL2Vec and DTW for Q2 to Q10 in Figure 10 and Figure 11, respectively. From these figures, we observe the layouts retrieved by BL2Vec match the queries very well. For example, we find that the overall layout of the TOP-1 result by BL2Vec is very similar to Q1, and the Top 2-5 results also maintain high consistency in the layouts. We also note that DTW is slightly better than BL2Vec for Q4 and Q5 (i.e., by 2% votes). We believe this might be because the layouts returned by DTW have the locations of the balls more similar to those in the query layouts and this is perceived by the users. Nevertheless, with a closer check, the balls at similar locations in the layouts returned by DTW and the query layouts are of different numbers (i.e., colors) and thus they should be regarded be not that similar to each other. This illustrates DTW's drawback that it does not captures other information than the locations of layouts well.

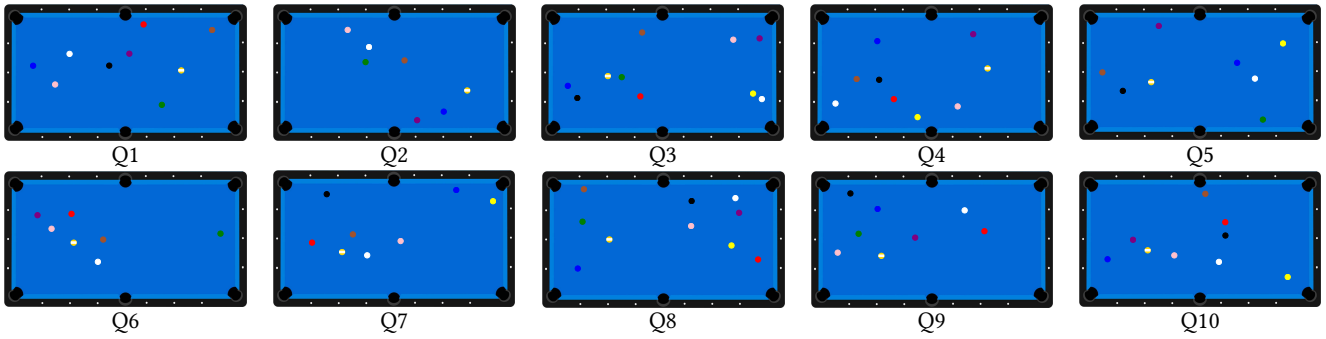


Figure 8: Ten queries that are used for user study.

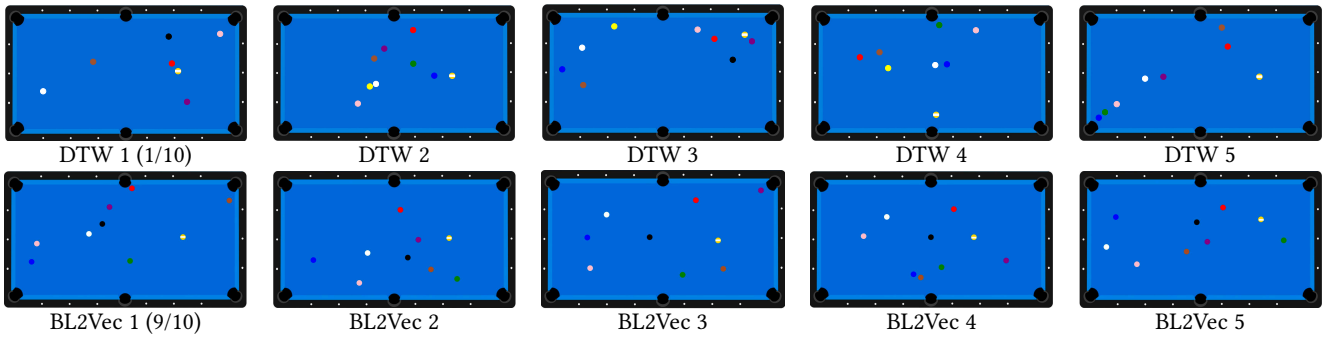


Figure 9: Top-5 billiards layouts for Q1 returned by the DTW and BL2Vec (from left to right), where (1/10) means the 1 user supports this result among the total of 10 users.

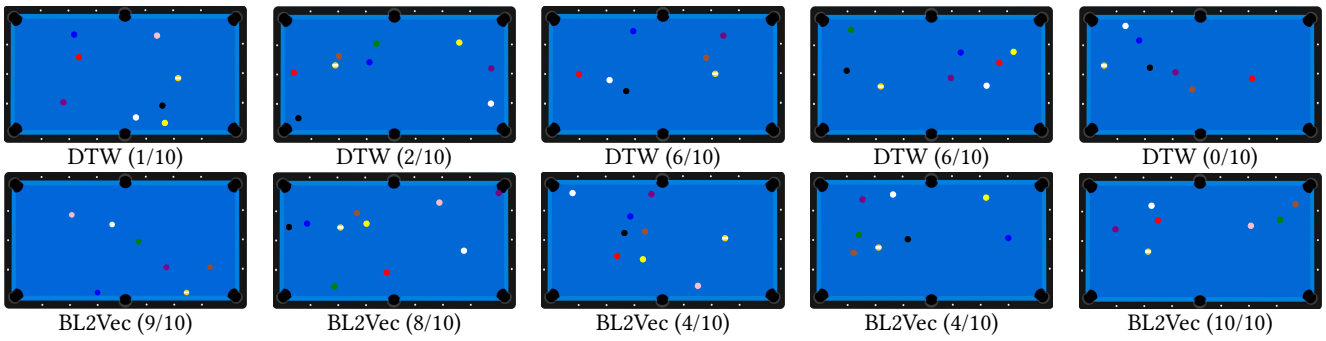


Figure 10: Top-1 billiards layouts for Q2-Q6 returned by DTW and BL2Vec (from left to right).

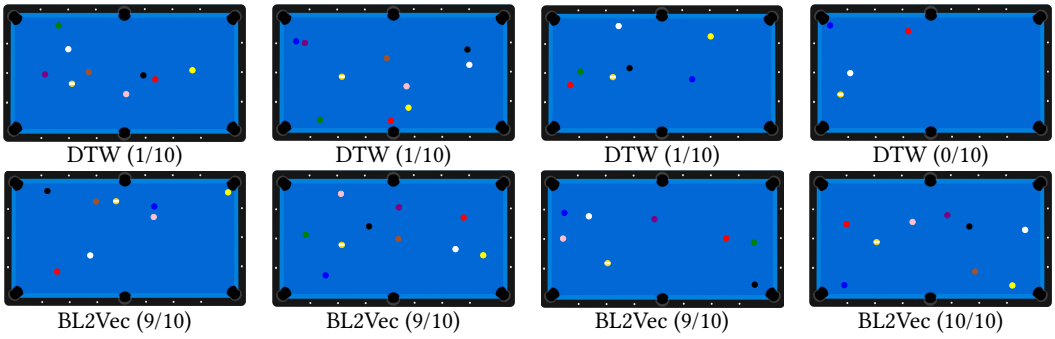


Figure 11: Top-1 billiards layouts for Q7-Q10 returned by DTW and BL2Vec (from left to right).

6 CONCLUSION

In this paper, we contribute the first billiards layout dataset and propose to study the similar billiards layout retrieval problem. We develop the first deep learning-based solution called BL2Vec by learning the representations of billiards layouts for computing their similarity. BL2Vec captures the unique characteristics in a billiards layout and runs very fast, i.e., it runs in linear time with the size of the database when performing the similarity search. We conduct extensive experiments on the collected real-world datasets, and the results verify that our method achieves the best effectiveness and efficiency. In the future, we plan to explore more methods for billiards related analytics tasks, e.g., realistic layouts generation and player performance analysis.

REFERENCES

[1] Helmut Alt and Michael Godau. 1995. Computing the Fréchet distance between two polygonal curves. *IJCGA* 5, 01n02 (1995), 75–91.

[2] Raquel Aoki, Renato M Assuncao, and Pedro OS Vaz de Melo. 2017. Luck is hard to beat: The difficulty of sports prediction. In *SIGKDD*. ACM, 1367–1376.

[3] P. Arsomngern, C. Long, S. Suwajanakorn, and S. Nutanong. 2021. Self-Supervised Deep Metric Learning for Pointsets. In *ICDE*.

[4] Alina Bialkowski, Patrick Lucey, Peter Carr, Yisong Yue, Sridha Sridharan, and Iain Matthews. 2014. Large-scale analysis of soccer matches using spatiotemporal tracking data. In *ICDM*. IEEE, 725–730.

[5] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. 1993. Signature verification using a" siamese" time delay neural network. *NIPS* 6 (1993), 737–744.

[6] Tom Decroos, Jan Van Haaren, and Jesse Davis. 2018. Automatic Discovery of Tactics in Spatio-Temporal Soccer Match Data. In *SIGKDD*. ACM, 223–232.

[7] Joachim Gudmundsson and Michael Horton. 2017. Spatio-temporal analysis of team sports. *CSUR* 50, 2 (2017), 22.

[8] Kaiming He and Jian Sun. 2015. Convolutional neural networks at constrained time cost. In *CVPR*. 5353–5360.

[9] Joao F Henriques, Joao Carreira, Rui Caseiro, and Jorge Batista. 2013. Beyond hard negative mining: Efficient detector learning via block-circulant decomposition. In *ICCV*. 2760–2767.

[10] Elad Hoffer and Nir Ailon. 2015. Deep metric learning using triplet network. In *SIMBAD*. Springer, 84–92.

[11] Daniel P Huttenlocher, Gregory A. Klanderman, and William J Rucklidge. 1993. Comparing images using the Hausdorff distance. *TPAMI* 15, 9 (1993), 850–863.

[12] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*. 604–613.

[13] Mahmut Kaya and Hasan Şakir Bilge. 2019. Deep metric learning: A survey. *Symmetry* 11, 9 (2019), 1066.

[14] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[15] Tie-Yan Liu, Wei-Ying Ma, and Hong-Jiang Zhang. 2005. Effective feature extraction for play detection in american football video. In *MMM*. IEEE, 164–171.

[16] Patrick Lucey, Alina Bialkowski, Peter Carr, Stuart Morgan, Iain Matthews, and Yaser Sheikh. 2013. Representing and discovering adversarial team behaviors using player roles. In *CVPR*. 2706–2713.

[17] Andrew Miller, Luke Bornn, Ryan Adams, and Kirk Goldsberry. 2014. Factorized point process intensities: A spatial analysis of professional basketball. In *ICML*. 235–243.

[18] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*. 652–660.

[19] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413* (2017).

[20] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. 2000. The earth mover's distance as a metric for image retrieval. *IJCV* 40, 2 (2000), 99–121.

[21] Long Sha, Patrick Lucey, Yisong Yue, Peter Carr, Charlie Rohlf, and Iain Matthews. 2016. Chalkboarding: A new spatiotemporal query paradigm for sports play retrieval. In *IUI*. ACM, 336–347.

[22] Konstantinos Skianis, Giannis Nikolentzos, Stratis Limnios, and Michalis Vazirgiannis. 2020. Rep the set: Neural networks for learning set representations. In *AISTATS*. PMLR, 1410–1420.

[23] Zheng Wang, Cheng Long, Gao Cong, and Ce Ju. 2019. Effective and efficient sports play retrieval with deep representation learning. In *SIGKDD*. 499–509.

[24] Byoung-Kee Yi, HV Jagadish, and Christos Faloutsos. 1998. Efficient retrieval of similar time sequences under time warping. In *ICDE*. IEEE, 201–208.

[25] Yisong Yue, Patrick Lucey, Peter Carr, Alina Bialkowski, and Iain Matthews. 2014. Learning fine-grained spatial models for dynamic sports play prediction. In *ICDM*. IEEE, 670–679.

[26] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. 2017. Deep sets. *arXiv preprint arXiv:1703.06114* (2017).

[27] Ye Zhang and Byron Wallace. 2015. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820* (2015).