Billiards Sports Analytics: Datasets and Tasks (Supplementary Materials)

Qianru Zhang* , Zheng Wang*, Cheng Long, Siu-Ming Yiu

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 the layouts (i.e., locations) of billiards balls after performing break shots, called break shot layouts, the traces of the balls as a result of strikes (in the form of trajectories), and detailed statistics and performance indicators. We then study and develop techniques for three tasks on the collected dataset, including (1) prediction and (2) generation on the layouts data, and (3) similar billiards layout retrieval on the layouts data, which can serve different users such as coaches, players and fans. We conduct extensive experiments on the collected dataset and the results show that our methods perform effectively and efficiently.

Index Terms—billiards sports analytics; billiards layout prediction; billiards layout generation; billiards layout retrieval

1 INTRODUCTION

Nowadays, it becomes a common practice to collect some data from sports games using tracking devices such as cameras and/or GPS sensors. The collected data facilitates various analytic tasks such as similar game retrieval [1], [2], tactics detection [3] and score prediction [4], [5], which cover sports including basketball, soccer, volleyball and handball. Yet these analytic tasks have not been explored on billiards sports, and this is mainly due to the lack of publicly available datasets of billiards sports. The billiards sport is a two-player game. In each game, there are a certain number of rounds (called *frames*). In each frame, two players take turns to strike a billiards ball (called the cue ball) so as to pot other balls (called object balls) to the pockets of a pool table (See Figure 1 for example). The object balls are usually associated with numbers and/or colors. Billiards sport, which can be played in halls, hotels, or as often in the club houses of other organisations, becomes a popular recreational sport [6], [7].

Billiards Datasets. We propose to collect some data capturing billiards games so that various types of data analyses can be conducted on the data for discovering knowledge about the games and players. Specifically, we collected the dataset from 9-ball games (i.e., one popular type of billiards sports), where a player needs to hit the object ball with the smallest number among those remaining on the table for each strike and wins the frame if he/she pots the object ball with the number 9. The dataset covers games of 94 international professional 9-ball tournaments for the last two decades, which were played by 227 professional players. In summary, the dataset includes 3,019 records for frames, 6,637 records for turns, and 2,082 records for strikes. More detailed summarization could be found in Table 1 and description in the supplementary materials [8].

Fig. 1. A real 9-ball billiards layout.

In this paper, we focus on the billiards *layouts* data involved in the collected dataset. 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. A layout example is shown in Figure 1. 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. We study three tasks on the billiards layouts data, including *prediction*, *generation* and *similarity search*. The first two are covered in our prior study [9] while the third one is newly covered in this paper.

Task 1: Break Shot Layout Prediction. Given a layout of a game, we predict three aspects of the game: (1) clear or not, it predicts whether the player who performs the break shot will pocket all balls; (2) win or not, it predicts whether the player who performs the break shot will win the game; (3) it predicts how many balls are consecutively

- *Equal contribution.
- *Q. Zhang and S.M. Yiu are with Department of Computer Science, The University of Hong Kong, Hong Kong SAR. Z. Wang and C. Long are with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. E-mail: {qrzhang, smyiu}@cs.hku.hk, wang_zheng@e.ntu.edu.sg, c.long@ntu.edu.sg*

potted after the break shot. The prediction task is non-trivial, which is mainly due to the characteristics of the data. In particular, the billiards layout data is unlike other existing datasets including a set of points [10], [11] or a sequence of locations such as trajectory data [12]. In a billiards layout, the billiards balls including one cue (white) ball and several object (colored) balls are all located on a rectangular pool table with six pockets. The data embeds rich correlations, e.g., the spatial correlations between the cue ball and object balls, and the correlations between the cue ball (or object balls) and the six pockets, which is unique and important in billiards sports. We carefully extract those unique features that can reflect the spatial correlation of balls and/or pockets. Then, we use Convolutional Neural Network (CNN) to accomplish this task in a supervised learning manner, since CNN is inherently applicable for perceiving the correlations. The proposed model is called BLCNN.

Task 2: Break Shot Layout Generation. This task is to generate realistic yet high-quality (i.e., easy-to-clear) layouts of break shots. For a player, it is critical to gain knowledge about such layouts that (1) he/she can accomplish them (i.e., the layouts are realistic) and (2) he/she would pot many balls and even clear the table given them (i.e., the layouts are of high quality). We explore a data-driven solution to generate the billiards layouts via Generative Adversarial Networks (GANs) [13], called BLGAN. More specifically, we treat each billiards layout as an ordered sequence, i.e., one cue ball followed by several object balls that remain on the table in ascending order of their numbers. Then, a generator is used to generate a sequence of discrete tokens, each representing a discretized location of the billiards table (e.g., a grid cell). For the discriminator, we collect some real layouts that are predicted to be cleared with high probabilities. These real layouts and the generated layouts together are fed to the discriminator, which tries to discriminate between the real ones and generated ones.

Task 3: Similar Layout Retrieval. This task is to find those billiards layouts from a database that are similar to a billiards layout at hand called *query layout*. This task can be involved 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 Web 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. This application can be implemented as a web-based searching engine, putting all layouts at a web server and providing a search functionality to coaches to look for similar layouts for training. 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) [14] and Hausdorff distance [15] and (2) recently emerging learning-based techniques such as DeepSets [10] and RepSet [11], all assume order invariance and thus they are not suitable for billiards layouts. We note that existing similarity measures on sequences and trajectories such as DTW [16], Frechet [17], LCSS [18], ERP [19], EDR [20], and EDwP [21], 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. For example, for two billiards layouts with totally different object balls such as Figure 5 (c) and (d), it is counter-intuitive to align these different coloured balls to compute the similarity. 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 [16], [17], which has at least a quadratic time complexity in terms of the number of balls and would impose a big challenge when performing similar layout retrieval on a huge database with many layouts and for a big number of queries since the operation of computing the similarity between two layouts needs to be conducted for many pairs of data layouts and query layouts.

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* [22], 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) [23] 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

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

hard negative mining [24] 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 wrt the number of balls in a game. 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 the one in [25].

Summary of Contributions. The contributions of this paper are summarized as follows, with the first two contributions covered by our prior study [9] and the last one newly covered in this extended version.

1) We contribute a billiards sports dataset that includes break shot data (for frames), strike statistics data (for turns) and trajectory data (for strikes). The layouts data is a new data type, which is related to yet different from quite a few existing data types including point sets, trajectories, sequences, etc. Specifically, it consists of a point set, is order-sensitive, and is associated with some contexts (e.g., the pockets). In addition, the dataset has rich contents and labels information to support various machine learning tasks. Our dataset and codes are publicly accessible via the link [2] and provide an opportunity for research communities such as machine learning, data mining, computational geometry, computer vision and sports science, to make a significant impact. (Section 3)
2) We study the prediction and generation tasks on break shot layouts data, which help better understand the sports and serve different users including coaches, players and fans. We develop the BLCNN model and the BLGAN model for the prediction and generation tasks, respectively. We conduct extensive experiments on the collected 9-ball data, which demonstrate that (1) BLCNN achieves superior classification accuracy over baseline methods; (2) BLGAN has good performance on generating the layouts of high quality (i.e., easily to be cleared) and reality. (Section 4, Section 5, Section 7.1, and Section 7.2)
3) We propose a new problem of searching similar billiards layout for a given query layout from a database of layouts. We then develop a deep learning-based similarity measure called BL2Vec for billiards layouts. The similarity not only considers the unique characteristics in a billiards layout but also runs fast in linear time. We conduct experiments on BL2Vec with the collected datasets. The effectiveness experiments show that BL2Vec can outperform the best baseline by at least 27% for similarity search. The efficiency results show that BL2Vec runs up to $420\times$ 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 BL2Vec with expert knowledge. (Section 6 and Section 7.3)

2. https://drive.google.com/drive/folders/1NBqonYLr_cParMMn4xSeE0KTJNhjeYuG?usp=sharing

2 RELATED WORK

(1) Sports Data Analytics. Billiards corresponds to one type of popular sports playing with a cue stick on a pool table. The traditional research in this area addresses the task of training the robotic players such as Deep Green [26] to execute good shots on a physical table and thus win a computer billiards game [27], [28]. Recently, Pan et al. [29] study the importance of a break shot in predicting the 9-ball game outcomes. Nowadays, it becomes a common practice to deploy some devices such as GPS sensors and cameras to capture some data of sports games, and the collected data proliferates various sports analytic tasks. Specifically, Wang et al. [1], [30] study the similar soccer game retrieval, which can recommend similar games to sports fans in some sports recommender systems such as ESPN. Decroos et al. [3] collect event-stream data from professional soccer matches, and explore the problem of tactics detection, which helps players improve their tactics when they prepare for an upcoming match. Aoki et al. [4] collect four different sports data including basketball, soccer, volleyball and handball, and analyze the difficulty of predicting the outcome of sports events. Overall, these tasks involve various sports such as basketball, soccer, volleyball and handball. In our prior study [9], we study the billiards layout prediction and generation tasks, and in this paper, we extend this line of research by studying one additional task of similar billiards layout retrieval.

(2) Sequence Data Prediction. The collected billiards data contains the layouts of break shots in real-world 9-ball games, and it can be treated as data of sequences of one cue ball followed by several object balls in the ascending order of their numbers. We review the existing works of sequence data prediction as follows. The common sequence data includes time series data and trajectory data. Time series prediction [31], [32] is a related but different task. It aims to train models to fit historical data and use them to predict future observations of a sequence. Existing methods for time series prediction can be grouped into two categories: classical models such as SVM [33] and recent deep learning-based models such as RNN variants [34] or Transformer variants [35]. On the other hand, trajectory data corresponds to a sequence of positions to capture the traces of moving objects. The problem of trajectory prediction can be viewed as a sequence generation task of predicting the future trajectories of moving objects based on their past positions, and it is widely used to avoid obstacles for pedestrians [36] and vehicles [37]. Our task differs from these studies mainly in two aspects. First, our billiards data carries the unique characteristics in billiards sports, e.g., it captures the locations of balls on a pool table. Second, our task is to predict some associated information with the layout instead of forecasting a future observation value in a sequence.

(3) Sequence Generative Models. We review existing generative models [38], [39], [40], [41], [42] related to the task of break shot layout generation as follows. The majority of deep generative models are used to generate text. For example, SeqGAN [38] is a typical adversarial text generation model, which builds an unbiased estimator based on the REINFORCE algorithm [43] for the generator, and applies the roll-out policy to obtain the reward from the

discriminator. In addition, to deal with the differentiation difficulty when applying GAN to generate sequences of discrete elements, GSGAN [44] is proposed to use Gumbel-softmax output distributions to train GAN on discrete sequences. Guo et al. [45] propose the LeakGAN, which introduces a hierarchical reinforcement learning framework for the generator, and improves the performance of long text generation. Overall, all of these works consider textual data, e.g., generating some sentences. Our work differs from them mainly in the data, i.e., billiards layout data corresponds to a set of billiards balls that are located on a pool table. Besides, the data is associated with several unique characteristics in billiards sports, e.g., the spatial correlations between the cue (white) ball and object (colored) balls.

(4) 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 [14], [15]. Earth Mover's Distance (EMD) [14] and Hausdorff [15] 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 [10], [11], [46], [47], [48]. 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 [46] and DeepSets [10] 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++ [47] is proposed to capture the local structures of a set by applying PointNet recursively and achieves a more efficient and robust set representation. RepSet [11] 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 [48] 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.

(5) Measuring Sequences or Structural Similarity We can treat billiards layouts as ordered sequences, i.e., one cue ball followed by several object balls in the ascending order of their numbers. In this case, existing similarity measurements for sequences/trajectories can be used for measuring the similarity between two billiards layouts. Some examples of similarity measurements for sequences/trajectories include DTW [16], Frechet [17], LCSS [18], ERP [19], EDR [20], and EDwP [21]. A more detailed survey on trajectory similarity could be found in [49], [50]. As explained in Section 1, these measurements are not suitable for billiards layouts since they would align the balls from two layouts solely based on their locations and totally ignore other information such as the ball numbers. In addition, these measurements usually incur high time costs, e.g., each of them has at least a quadratic time complexity in terms of the number of balls. In addition, some studies take layouts as images for learning the similarity. For example, Patil et al. [51] measures the structural similarity between layouts such as floorplans, and Manandhar et al. [52] measures the similarity between layouts such as Web UI designs. Both of them have specific designs for their data types, which are largely different from our billiards layouts, where we take layouts as a set of locations.

(6) 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. [53] 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 [22]. 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 Billiards Background and Dataset

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.

Our real-world billiards data is extracted from the videos of professional billiards games published on YouTube in the most recent two decades using the software Kinovea (https://www.kinovea.org/). The data collection process by the software Kinovea consists of four steps (i.e., uploading images/videos, adding grids and markers, exporting coordinates and collecting information), which are illustrated in detail in the supplementary materials [8]. The dataset covers 227 players and 94 international professional 9-ball

TABLE 1
Summary of the collected billiards dataset (cover 94 tournaments and 227 players).

Data of Frames (3,019)		Data of Turns (6,637)		Data of Strikes (2,082)			
1	break shot layout	1	player name	1	trajectory	6	stick top position
2	three performance indicators: clear label, win label and # of potted balls label	2	# of strikes	2	camera angle		
		3	order of potted balls	3	cushion	7	direction on hitting
		4	type of foul	4	intersection point		
		5	data of strikes	5	cue ball position		

tournaments. We collect the billiard dataset for frames, turns and strikes. Table 1 summarizes the collected dataset (with details included in [8]).

4 Layout Prediction with BLCNN

In this section, we introduce how to embed each ball in billiards layouts into a real vector. Then, the vectors are concatenated in an ascending order of ball numbers that should be potted as an embedding of features, which is fed into a classifier based on the architecture of CNN to predict the results. The effectiveness of prediction depends on the quality of extracted features. The proposed model for billiards layout prediction is called BLCNN.

Feature Extraction. We capture the correlations in a billiard layout and consider the following three kinds of features: the location information of billiards balls on the table, called Ball-Self (BS); the correlation between balls and pockets, called Ball-Pocket (BP); the correlation between balls, called Ball-Ball (BB). Figure 2 illustrates these features from a 9-ball billiards layout.

(1) Ball-Self (BS). We map the play field of the billiards table into a 200× 100 coordinate system and set the bottom left corner as the origin. Thus, the range of the x-axis of the coordinate system is [0,200], the range of the y-axis of the coordinate system is [0,100]. For each ball b_i in the billiards layout, it is related to (x, y) in the coordinate system, which is regarded as its position feature on the pool table. For example, in Figure 2, the location of ball 2 (blue) is $(88.06, 76.02)$.

(2) Ball-Pocket (BP). To report features between each ball and each pocket, we set the pocket at the bottom left corner as the start pocket and mark the pockets clockwise from 1 to 6, which is shown in Figure 2. For each ball b_i, we use four features including cushion angle, distance to pocket, an indicator of occlusion and pocket index to describe it wrt each pocket $p_j (1 \leq j \leq 6)$ on the table. Thus, there are in total $4 \times 6 = 24$ features for each ball b_i. (i) Cushion angle. Cushion angle is used to describe the minimum angle between the line from a ball b_i to a pocket p_j (i.e., denoted by $\overline{b_i p_j}$) and its two cushion edges. (ii) Distance to pocket. Distance to pocket is the distance from a ball b_i to a pocket p_j (i.e., $\|b_i - p_j\|_2$). (iii) Indicator of occlusion. An indicator of occlusion is used to report whether there is a ball on the line from a ball b_i to a pocket p_j. If so, the indicator of occlusion is 1; 0 otherwise. (iv) Pocket index. The pocket index is used to distinguish different called pockets, which is from 1 to 6. Take ball 2 as an example in Figure 2, the cushion angle is $40.80°$ between the line $\overline{b_2 p_1}$ and cushion 1-6; the distance from ball 2 to pocket 1 is 116.33; the indicator of occlusion is 1 since cue ball appears in the path from ball 2 to pocket 1; the pocket index is 1 for the above three features. In brief, the feature (i) and (ii) capture the feasibility to sink the ball 2 into a called pocket; the feature (iii) captures whether there will be collisions or bounces on the path of ball 2 to a called pocket; the feature (iv) is to distinguish different called pockets varying from 1 to 6.

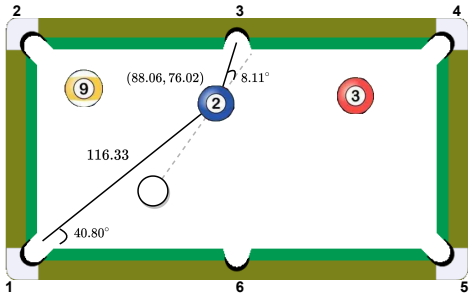


Fig. 2. Illustration of the BS, BP and BB features of ball 2 (blue) in a layout.

(3) Ball-Ball (BB). We consider two features to capture the correlation of balls, including shot angle and pocket index. (i) shot angle. Shot angle describes the minimum angle between the line of two balls with consecutive numbers on the table (i.e., denoted by $\overline{b_i b_{i+1}}$) and the line from ball b_{i+1} to each pocket p_j. In Figure 2, the shot angle is $8.11°$ since the minimum angle corresponds to the angle of a line from the cue ball to ball 2 (i.e., $\overline{b_1 b_2}$) and the line from ball 2 to pocket 3. (ii) Pocket index. The pocket index is selected to distinguish the index of the most likely called pocket when performing a strike. In the case of ball 2 in Figure 2, the pocket index is 3 since pocket 3 is more likely to be called with the minimum shot angle $8.11°$.

CNN-based Classifier. BLCNN for the prediction task is a CNN-based architecture, which is illustrated in Figure 3. We consider three labels in the task including clear or not, win or nor and the number of potted balls after the break shot based on a given billiards layout.

In particular, for each billiards ball, we transform its extracted features (i.e., BS, BB and BP) into real vectors via an embedding layer, then the vectors are concatenated into a long vector as the embedding of the ball. To achieve the embedding, we granulate these features into the tokens. Next, the ball embeddings are further concatenated in the ascending order of their numbers and we apply padding when there are missing balls on the table. Thus, we get an embedding for the features of billiards balls in a matrix form, called embedded features. Note that we embed these features via an embedding layer instead of using the feature values directly because the feature values restrict the embeddings in a low dimensional space, which makes it difficult for the model to be further optimized. Further, the embedded features are fed to a convolutional layer with multiple filters of varying sizes to obtain feature

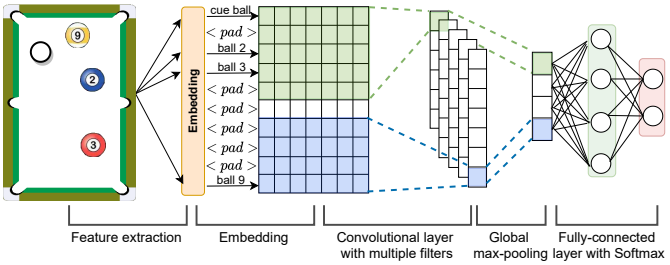


Fig. 3. The model architecture of BLCNN. The colours (i.e., green and blue) denote the convolutional filters with different sizes to obtain feature maps.

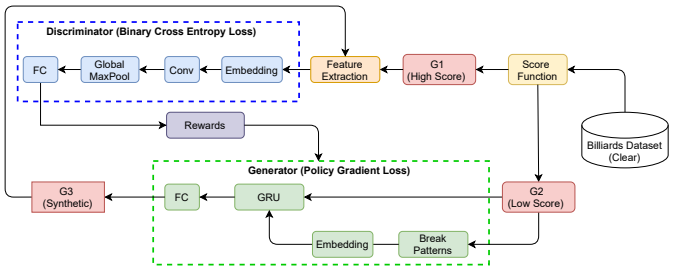


Fig. 4. The model architecture of BLGAN. The score function is to identify two groups of layouts with high and low scores (i.e., G_1 and G_2) from the dataset. In the discriminator, it follows the architecture in BLCNN, and provides the rewards outputted via a fully-connected (FC) layer for training the generator. In the generator, the break patterns are extracted from G_2, each pattern corresponds to a token, which is fed into an embedding layer to obtain a latent vector as the initial hidden vector, to generate the G_3 via GRU followed by a FC layer.

maps, and a global max-pooling layer is then employed to capture the most important feature for each feature map, and those important features are further aggregated into a vector, which corresponds to a representation of the billiards layout. The layout representation is then fed to a standard neural network architecture, i.e., the fully-connected layer, with softmax nonlinearity to produce the output. We train the BLCNN in a supervised learning manner, with the cross-entropy loss for the prediction of three types of labels (i.e., clear, win and potted balls).

5 Layout Generation with BLGAN

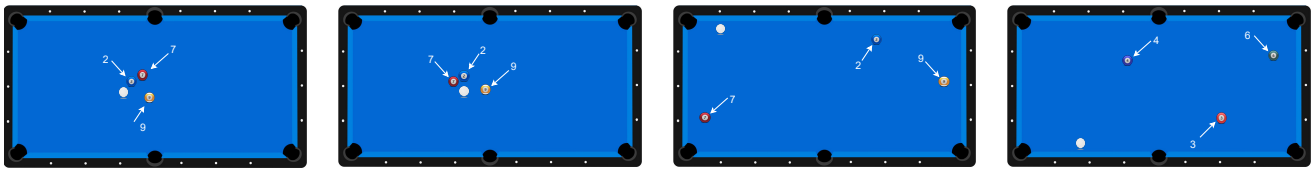
We first explain that the layout generation task is useful by referring to some existing literature of billiards sports [27], [28], which aim to find high-quality break shots so as to achieve desirable layouts. For example, in [28], it models a layout (i.e., the positions of balls in a billiards table) as a state and how to execute a shot as an action. Each action is controlled by five continuous parameters including the direction of hitting, etc. It aims to search for a shot that would generate a desirable layout. It is also worth mentioning that our dataset includes sufficient information on the the strike level (including the break shots), such as the hitting position, stick top position, direction on hitting, which we believe will facilitate more in-depth research along this line.

Challenges. Generative Adversarial Network (GAN) [13] is a promising framework to generate new data based on existing data. A natural idea is to treat a billiards layout as an ordered sequence, i.e., one cue ball followed by object balls that remain on the table in ascending order of their numbers, and apply GAN to generate sequences of balls (and their locations). Nevertheless, it has the following challenges. First, when the physical space of the billiards balls (i.e., the billiards table) is discretized (e.g., as a 15 $\times$ 15 grid), applying GAN to generate the balls and their locations (as discrete tokens) would suffer from a classical issue of indifferentiability [44], since the samples from a distribution of discrete tokens are not differentiable with respect to the distribution parameters. Second, in a break shot layout, some balls may have been potted already, and in this case, the player who performs the break shot can keep his turn and thus have the chance to clear the table given the layout. Yet a straightforward GAN does not provide a mechanism of deciding some ball/balls to be omitted from the generated layouts, i,e., meaning some of the balls are potted already.

Overview. To serve the target of generating the layouts that are more likely to be cleared, we collect all layouts with clear labels from the dataset. We employ a *score function* (more details would be discussed later) to score the quality of a layout, sort the layouts in descending order of their scores and divide the billiards layouts equally into two groups without overlapping (i.e., one group (G_1) represents layouts with high scores (high-score), the other group (G_2) represents layouts with low scores (low-score)). G_2 is fed to the *generator* in BLGAN to guide the new layout generation denoted by G_3. Also, we train a *discriminator* to provide guidance for improving the quality of billiards layouts the generator generates by feeding the examples of G_1 (i.e., the real layouts with high scores) and the examples of G_3 (i.e., the generated layouts). The GAN-based architecture is illustrated in Figure 4.

Score Function. We use the BLCNN model to compute the score of a layout since BLCNN model can predict whether a layout will be cleared or not, which indicates the quality of the layout. BLCNN involves a fully-connected layer with the softmax function and thus it computes scores from 0 to 1.

Generator. To overcome the issue of indifferentiability, we follow an existing strategy [38], [54], [55] to model the sequence generation procedure as a sequential decision making process, which is optimized by the REINFORCE algorithm via Policy Gradient [43] and thus it naturally bypasses the differentiation difficulty for discrete tokens. In particular, we train a generative model G to generate a sequence $C_{1:T} = (c_1, c_2, ...c_t, ...c_T)$, $c_t \in \gamma$, where c_t denotes the location token of a ball, and γ corresponds to the vocabulary of all location tokens on the billiards table. At time step t, the state s corresponds to the sequence of generated tokens so far, i.e., $(c_1, c_2, ...c_{t-1})$; and the action a is to select the next location token (i.e., c_t) based on the state s. To obtain the reward r, the existing study [38] adopts the roll-out policy by sampling the unknown tokens in a sequence to obtain an immediate reward from the discriminator. In this paper, we adopt a simpler strategy, i.e., we collect a reward only after an entire layout is generated and fed into the discriminator - recall the discriminator would return a score as the reward signal, and it is shared to all steps when the entire layout has not been generated.



(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)

Fig. 5. An example of illustrating the billiards layout similarity, where $sim(\mathcal{B}_q, \mathcal{B}_1) > sim(\mathcal{B}_q, \mathcal{B}_2) > sim(\mathcal{B}_q, \mathcal{B}_3)$.

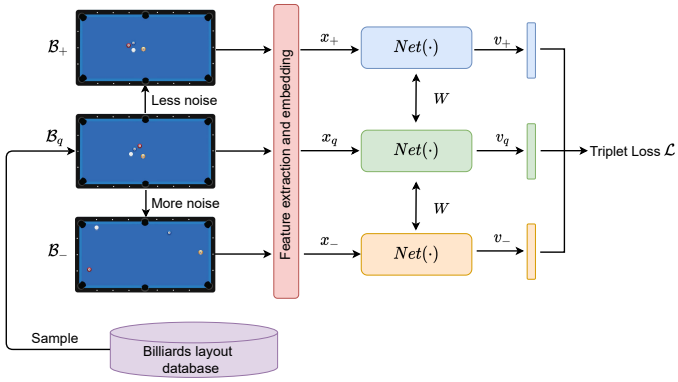


Fig. 6. The model architecture of BL2Vec.

Besides, we collect a set of patterns, each corresponding to a set of balls of a layout, which appear in the real dataset. Each such pattern is called a *break pattern* and is encoded using one-hot vectors. For example, consider a break shot layout in a 9-ball game with the cue ball and ball 1, 2, 3, 5, 7, 8, 9 on the pool table. The break pattern of this layout is denoted by $(4, 6)$ since ball 4 and 6 are potted already and thus missing from the layout. When generating a layout, we randomly sample a known break pattern and feed its vector to the generator as an initial hidden vector to guide the generation task. This is to handle the second challenge.

Discriminator. By empirical findings, the CNN-based model has shown good performance to capture the correlations for the billiards layout data, e.g., it achieves the accuracy around 90% on the previous prediction tasks. Thus, we adopt the CNN-based architecture in the discriminator, whose main function is to discriminate the generated layouts (i.e., G_3) and high-score real layouts (i.e., G_1), and provide a reward signal to guide the learning of the generator, i.e. it encourages the generator to generate the layouts of high-score (i.e., G_3) from those low-score real layouts (i.e., G_2). The optimization of the discriminator is to minimize Binary Cross-Entropy (BCE) between G_1 and G_3.

6 Layout Retrieval with BL2Vec

6.1 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 5 that given a query layout and two candidate layouts, it is much easier to distinguish which one is more similar to the query layout. Specifically, in Figure 5, $\mathcal{B}_q$ 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., $sim(\mathcal{B}_q, \mathcal{B}_1) > sim(\mathcal{B}_q, \mathcal{B}_2) > sim(\mathcal{B}_q, \mathcal{B}_3)$). Here, $sim(,)$ defines the similarity between two layouts.

Motivated by this, we propose a deep metric learning model based on *triplet network* [56] to learn a similarity measure for billiards layouts. We call the model *BL2Vec* and show its overview in Figure 6. 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:

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

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.

6.2 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 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 $\mathcal{B}_-$.

6.3 Billiards Layout Feature Extraction and Embedding

We extract the features Ball-Self (BS), Ball-Pocket (BP) and Ball-Ball (BB) from a billiards layout as presented in Section 4, and then map the continuous BS, BP and BB features into discrete tokens with some predefined granularities (e.g., the granularity of cell size is set to 15×15 for BS, $15°$ and 10 are used to partition the angle and distance ranges for BP and BB, respectively). 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.

6.4 Model Architecture of $Net(\cdot)$

We adopt a convolutional neural network (CNN) to instantiate the $Net(\cdot)$, 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 $x_{1:n} = x_1 \oplus x_2 \oplus ... \oplus x_n$, 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 [57], 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, and r denotes each dimension in an embedded feature x_i ($1 \leq r \leq K'$). 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

$$\hat{c} = \max_{1 \leq i \leq n-h+1} \{c_i\}$$

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.

Algorithm 1 Training of the BL2Vec Model

Require: $\mathcal{D}$: the database of billiards layouts need to be embedded; K (resp. K'): the embedding dimension of each billiards layout (resp. ball)
Ensure: The trained BL2Vec model
1: **repeat**
2: sample an anchor $\mathcal{B}_q$ from $\mathcal{D}$, and generate $\mathcal{B}_+$ and $\mathcal{B}_-$ by adding noises;
3: extract features and obtain tokens $(\mathcal{T}_q, \mathcal{T}_+, \mathcal{T}_-)$;
4: embed the tokens and obtain the embedded features (x_q, x_+, x_-);
5: get (v_q, v_+, v_-) from $Net(\cdot)$;
6: optimize the triplet loss $\mathcal{L}$ according to Equation 1;
7: **until** No improvement on validation set

6.5 Model Training

The detailed procedure of training the BL2Vec is presented in Algorithm 1. During the iterative training process (lines 1-7). It first samples a billiards layout $\mathcal{B}_q$ from the database $\mathcal{D}$ as an anchor and generates its positive version $\mathcal{B}_+$ and negative version $\mathcal{B}_-$ by adding noises (line 2). It then extracts the BS, BP and BB features from the layouts, and map them into discrete tokens $(\mathcal{T}_q, \mathcal{T}_+, \mathcal{T}_-)$ (line 3). These tokens are further fed into the embedding layer of the model to get the embedded features (x_q, x_+, x_-) (line 4). Next, it obtains the embedding vectors (v_q, v_+, v_-) from $Net(\cdot)$ (line 5). Finally, it computes the triplet loss $\mathcal{L}$ according to Equation 1 and optimizes the loss via some optimizers such as Adam stochastic gradient descent (line 6).

Once the model has been trained, we can embed each billiard layout in a database to a vector as a pre-processing step. When a query layout comes, we embed the query layout to a vector, perform a k *nearest neighbor* (kNN) [25] query to search k data vectors that have the smallest Euclidean distances to the query vector, and then return the billiards layouts corresponding to the found data vectors.

6.6 Time Complexity Analysis

We analyze the time complexity of computing the similarity between two billiards layouts based on BL2Vec. Specifically, it 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 [58]. For distance computation, it costs $O(K)$ which is obvious. Hence, the overall time complexity is $O(n + K)$. We note that existing matching-based methods [16], [17] have the time complexity at least $O(n^2)$ and the operator of computing similarity is usually conducted highly frequently. As a result, the similar billiard layout retrieval based on BL2Vec is significantly faster than that based on existing methods (e.g., up to 420x speed-up in our experiments).

Finally, we analyze the time complexity of the similar billiards layout retrieval for a query layout based on BL2Vec. Suppose there are N billiards layouts in a database. The time complexity of computing the vectors of the N billiards layouts is $O(N \cdot (n + K))$, where n is the number of balls in a billiards game and K is the number of dimensions of a vector. Note that this cost is one-time cost and shared by all queries. The time complexity of answering each similar layout retrieval query is dominated by that of conducting a kNN query, for which we can leverage a rich literature. For example, the time complexity is $O(k \cdot \log(N))$ if the algorithm in [25] is adopted for the kNN query.

7 EXPERIMENTS

7.1 Evaluation on the Billiards Layout Prediction Task

We evaluate the BLCNN for the prediction task on the collected dataset, including (1) an effectiveness study to evaluate the accuracy of the classification on three tasks (i.e., clear, win and potted ball), (2) a parameter study to evaluate the effect of cell size, and (3) an ablation study to evaluate the importance of each component of extracted features in BLCNN. It demonstrates the superior performance of BLCNN (e.g., with the accuracy of 89.69%, 86.56% and 80.94% for the clear, win and potted ball tasks, respectively). Detailed results and explanations can be found in [9].

7.2 Evaluation on the Billiards Layout Generation Task

We further evaluate the BLGAN for the generation task. We first evaluate (1) the quality and reality of the generated layouts by BLGAN. Overall, we observe BLGAN has good performance on generating the layouts of high quality (e.g., the scores are all-around 0.9 for four frequent break patterns), and the generated layouts are in general quite similar to real ones (e.g., it exceeds the similarity of more than half of the real layouts). We also conduct (2) a user study to show the quality of the generated layouts. We observe BLGAN generates the layouts that can be easily cleared with 82.5% votes from users. We present the detailed results and explanations for BLGAN in [9].

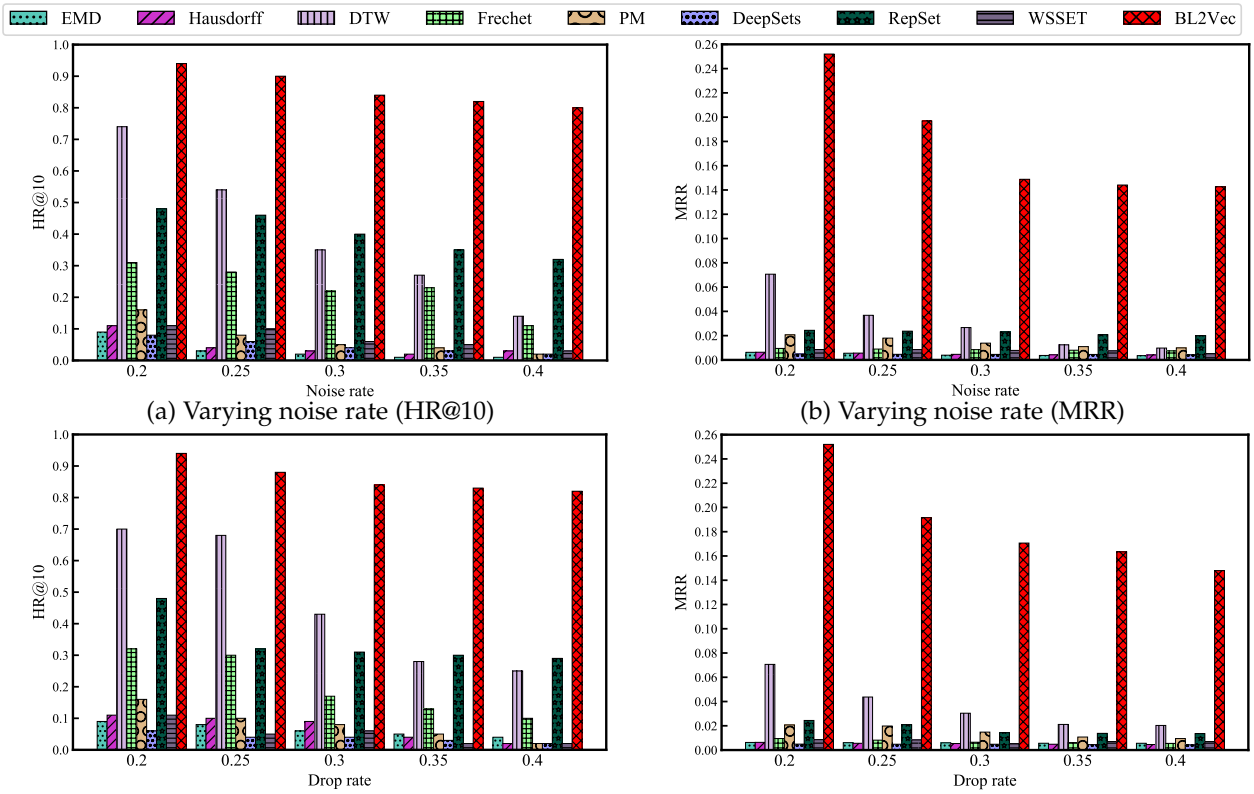
7.3 Evaluation on the Billiards Layout Retrieval Task

7.3.1 Experimental Setup

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

- EMD [14]. 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 [15]. 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 [16] and Frechet [17]. DTW and Frechet are two widely used similarity measurements for sequence 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 [10]. 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 [11]. 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 [48]. 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. 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. The parameter is set to 15 because it provides better performance in general via empirical studies (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 vary the parameter of angle (resp. distance) from 5° to 20° (resp. 5 to 20), and since the results are similar, we use the setting of 15° and 10 for partitioning the angle range and distance range, respectively. 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



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

Fig. 7. Effectiveness evaluation (similarity search).

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 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 [3].

TABLE 2
Effectiveness with classification and clustering.

Tasks	Classification	Clustering	
	Acc (%)	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**

7.3.2 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 [48] 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 6.2, (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

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

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 7 (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 7 (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 and 9.6 times in terms of $HR@10$ and MRR.

(2) Effectiveness evaluation (classification). Table 2 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 2. 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 3, 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 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.

TABLE 3
The effect of the cell size.

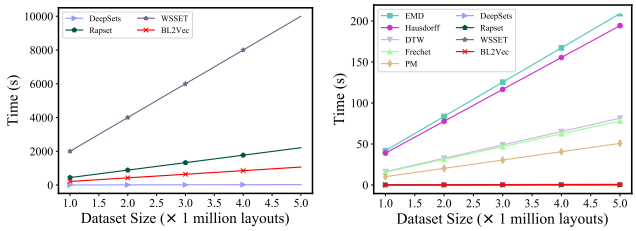
Cell size	#tokens	Similarity		Classification	Clustering	
		HR@10	MRR	Acc (%)	ARI (%)	AMI (%)
10	200	0.92	0.240	66.67	52.25	55.80
15	98	0.94	0.252	**72.62**	**61.94**	**63.26**
20	50	0.95	0.259	67.33	56.96	58.25
25	32	0.96	0.303	70.14	55.21	57.37
30	28	**0.96**	**0.306**	68.72	53.96	56.28

TABLE 4
Effectiveness evaluation with ablation study.

Model	Similarity		Classification	Clustering	
	HR@10	MRR	Acc (%)	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

(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 4 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, 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).



(a) Embedding scalability (b) Search scalability

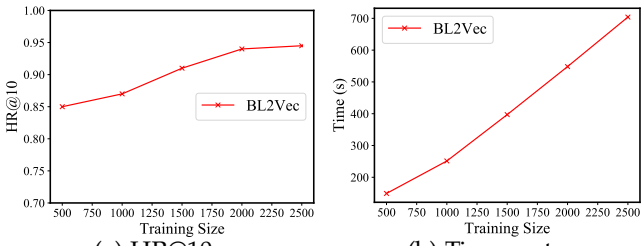
Fig. 8. Scalability test.

(6) Scalability test. To test scalability, we generate more layouts by adding noise on the original dataset. Figure 8(a) reports the running time of embedding the database from 1 million layouts to 5 million layouts for the learning-based methods, i.e., DeepSets, WSSET, Rapset and BL2Vec. Figure 8(b) reports the average running time over 10 queries for computing the similarity between a query billiards layout and the billiards layouts when we vary the size of the target database from 1 million layouts to 5 million layouts.

For embedding time, we notice all learning-based methods have their running times linearly increase with the dataset size, which is consistent with their linear time complexity for embedding layouts. We observe DeepSets model is fastest for the embedding, because the model is light via embedding each point into a vector, and aggregating all embeddings together to an overall embedding of the pointsets, which can be accomplished very quickly. Overall, BL2Vec runs comparably fast for the layout embedding, compared with other learning-based methods.

For searching time, 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) Training time. The training time of BL2Vec is shown in Figure 9. In particular, we study how the number of training samples affects the model performance for similarity search (i.e., $HR@10$) and the training time cost. We randomly 5 training sets from the dataset, including 500, 1,000, 1,500, 2,000, 2,500 billiards layouts, respectively. For each training



(a) HR@10 (b) Time cost

Fig. 9. Training cost.

set, we report its training cost per epoch and the corresponding effectiveness on the testing set with 500 layouts, where we follow the default setup in Section 7.3.1. Based on the results, we observe the effectiveness improves as the training size increases, and it approximately converges when the training size exceeds 2,000. For the training time, it increases almost linearly with the training size as expected. We use the setting of 70% (i.e., 2,000) billiards layouts for training, since it provides a reasonable trade-off between effectiveness and training cost.

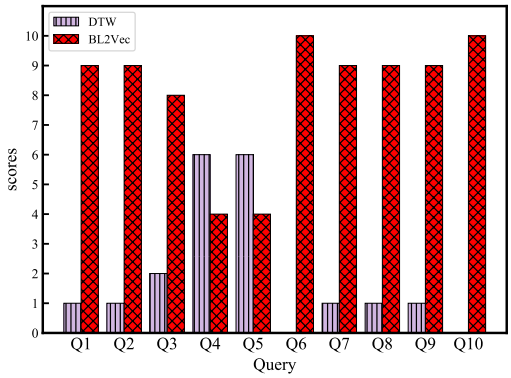


Fig. 10. User study.

(8) User study. Since DTW performs the best among all baselines in the similar billiards layout retrieval task as shown in Figure 7, we conduct a user study to compare the BL2Vec with DTW. We randomly select 10 billiards layouts as the query set as shown in the first row in Figure 11 and Figure 12, 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 on introducing the background to the volunteers so that they understand these images as shown in Figure 11 and Figure 12. 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 10. 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-1 results of BL2Vec and DTW for Q1 to Q5 and Q6 to Q10 in Figure 11 and Figure 12, respectively. From these figures, we observe the

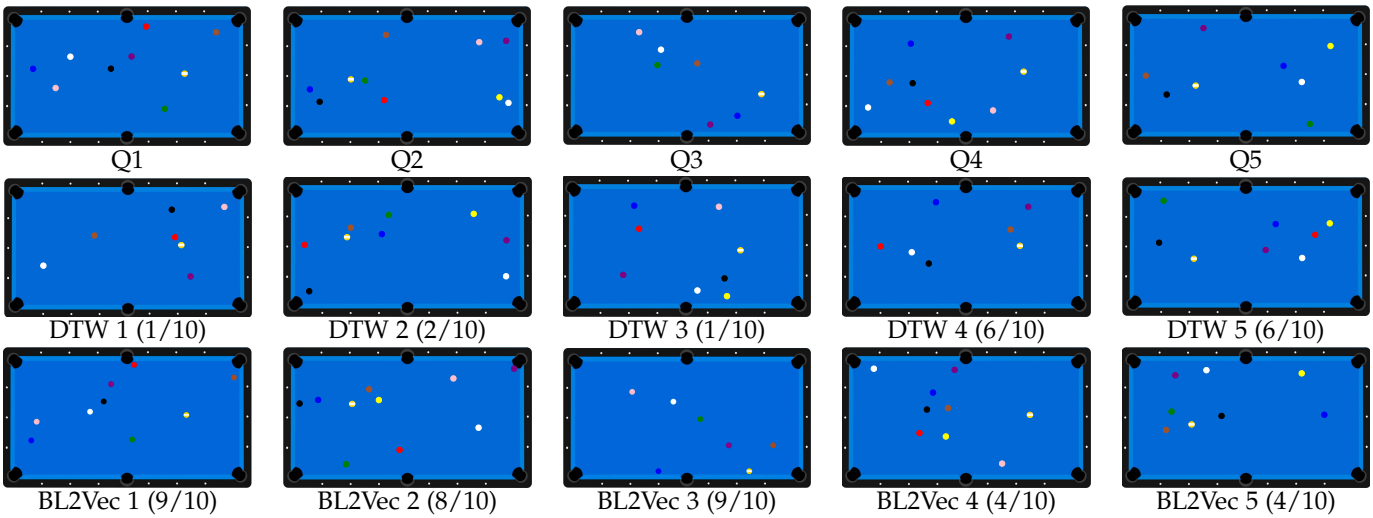


Fig. 11. Top-1 billiards layouts for Q1-Q5 returned by DTW and BL2Vec, (1/10) means the 1 user supports this result among the total of 10 users.

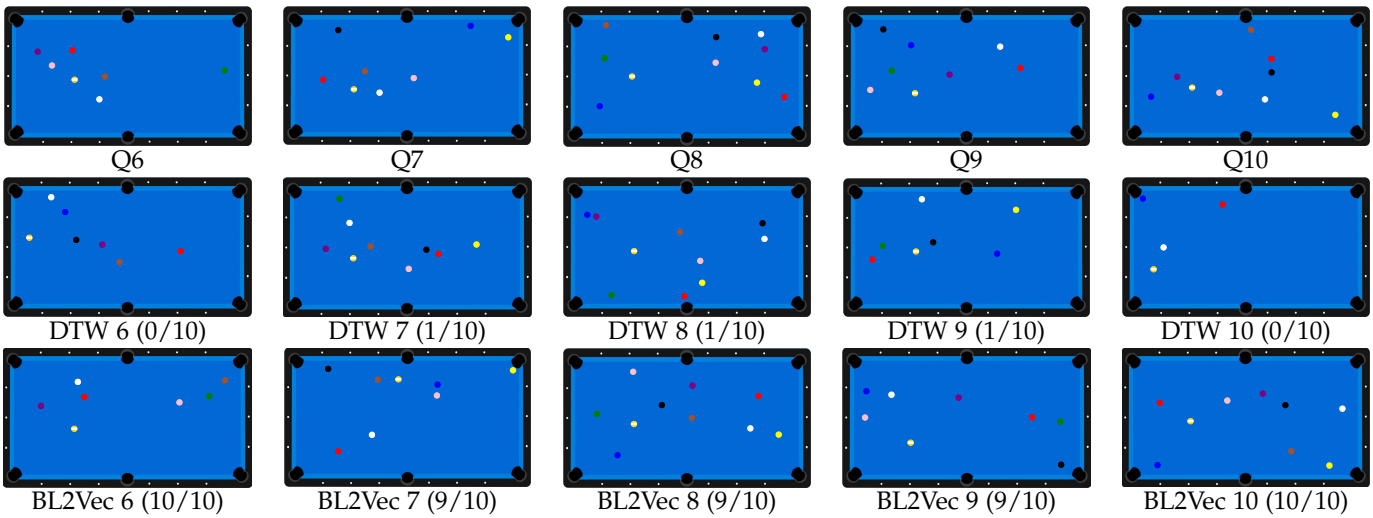


Fig. 12. Top-1 billiards layouts for Q6-Q10 returned by DTW and BL2Vec.

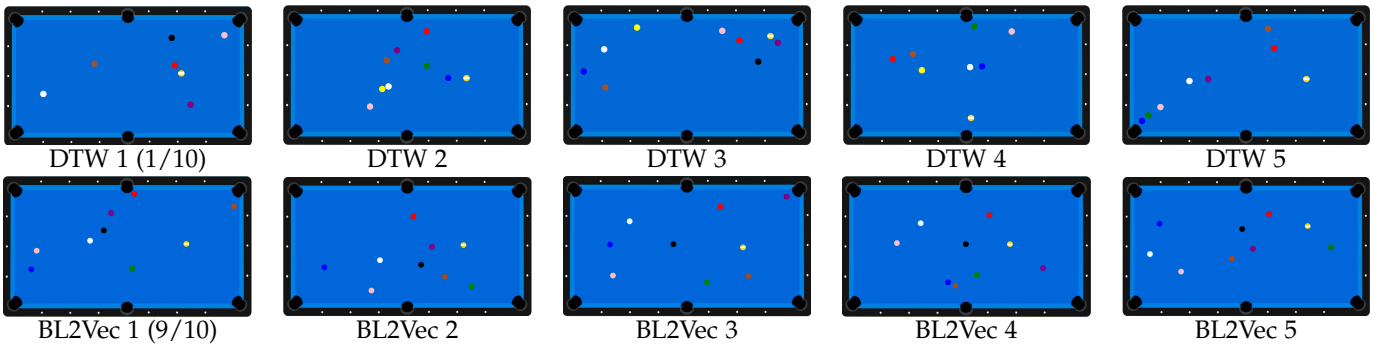


Fig. 13. Top-5 billiards layouts for Q1 returned by the DTW and BL2Vec (from left to right).

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. DTW is slightly better than BL2Vec for Q4 and Q5 (i.e., 2% votes). This is because the overall layouts returned by DTW are more similar to the queries; however, DTW falsely matches the balls, e.g., 9 balls in Q4, but the layout returned by DTW only has 7 balls and they are matched with different colors. It indicates that the pairwise point-matching solution is not suitable for the billiards layout retrieval task. In addition, we visualize the Top 5 results for Q1 returned by the DTW and BL2Vec in Figure 13, we observe BL2Vec maintains a high consistency between the retrieved Top 2-5 results and the query.

8 CONCLUSION

In this paper, we contribute a publicly available billiards layout dataset, which includes break shot layout data, trajectory data of strikes, details of strikes, etc. On the dataset, we investigate a few tasks including (1) prediction and (2) generation based on the break shot layout data, and (3) similar billiards layout retrieval. Extensive experiments are conducted on the collected dataset, which verify the usefulness of the dataset and also the proposed methods for the tasks on the dataset. In the future, we plan to explore more methods for billiards related analytics tasks, e.g., player performance analysis, tactics discovery, etc.

REFERENCES

[1] Z. Wang, C. Long, G. Cong, and C. Ju, "Effective and efficient sports play retrieval with deep representation learning," in *SIGKDD*, 2019, pp. 499–509.

[2] L. Sha, P. Lucey, Y. Yue, P. Carr, C. Rohlf, and I. Matthews, "Chalkboarding: A new spatiotemporal query paradigm for sports play retrieval," in *IUI*. ACM, 2016, pp. 336–347.

[3] T. Decroos, J. Van Haaren, and J. Davis, "Automatic discovery of tactics in spatio-temporal soccer match data," in *SIGKDD*. ACM, 2018, pp. 223–232.

[4] R. Aoki, R. M. Assuncao, and P. O. Vaz de Melo, "Luck is hard to beat: The difficulty of sports prediction," in *SIGKDD*. ACM, 2017, pp. 1367–1376.

[5] Y. Yue, P. Lucey, P. Carr, A. Bialkowski, and I. Matthews, "Learning fine-grained spatial models for dynamic sports play prediction," in *ICDM*. IEEE, 2014, pp. 670–679.

[6] M. Smith, "Pickpocket: A computer billiards shark," *Artificial Intelligence*, vol. 171, no. 16-17, pp. 1069–1091, 2007.

[7] S. Mathavan, M. Jackson, and R. M. Parkin, "A theoretical analysis of billiard ball dynamics under cushion impacts," *JMES*, vol. 224, no. 9, pp. 1863–1873, 2010.

[8] Q. Zhang, Z. Wang, C. Long, and S. M. Yiu, "On predicting and generating a good break shot in billiards sports (supplementary materials)," https://zhengwang125.github.io/paper/SDM_Supplementary.pdf, 2022.

[9] Q. Zhang, Z. Wang, C. Long, and S.-M. Yiu, "On predicting and generating a good break shot in billiards sports," in *SDM*. SIAM, 2022, pp. 109–117.

[10] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. Salakhutdinov, and A. Smola, "Deep sets," *arXiv*, 2017.

[11] K. Skianis, G. Nikolentzos, S. Limnios, and M. Vazirgiannis, "Rep the set: Neural networks for learning set representations," in *AISTATS*. PMLR, 2020, pp. 1410–1420.

[12] Y. Zheng, X. Xie, W.-Y. Ma *et al.*, "Geolife: A collaborative social networking service among user, location and trajectory." *IEEE DEB.*, vol. 33, no. 2, pp. 32–39, 2010.

[13] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *arXiv*, 2014.

[14] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *IJCV*, vol. 40, no. 2, pp. 99–121, 2000.

[15] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge, "Comparing images using the hausdorff distance," *TPAMI*, vol. 15, no. 9, pp. 850–863, 1993.

[16] B.-K. Yi, H. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," in *ICDE*. IEEE, 1998, pp. 201–208.

[17] H. Alt and M. Godau, "Computing the fréchet distance between two polygonal curves," *IJCGA*, vol. 5, no. 01n02, pp. 75–91, 1995.

[18] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories," in *Proceedings 18th international conference on data engineering (ICDE)*. IEEE, 2002, pp. 673–684.

[19] L. Chen and R. Ng, "On the marriage of lp-norms and edit distance," in *PVLDB*. VLDB Endowment, 2004, pp. 792–803.

[20] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *SIGMOD*. ACM, 2005, pp. 491–502.

[21] S. Ranu, P. Deepak, A. D. Telang, P. Deshpande, S. Raghavan *et al.*, "Indexing and matching trajectories under inconsistent sampling rates," in *ICDE*. IEEE, 2015, pp. 999–1010.

[22] M. Kaya and H. Ş. Bilge, "Deep metric learning: A survey," *Symmetry*, vol. 11, no. 9, p. 1066, 2019.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[24] J. F. Henriques, J. Carreira, R. Caseiro, and J. Batista, "Beyond hard negative mining: Efficient detector learning via block-circulant decomposition," in *ICCV*, 2013, pp. 2760–2767.

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

[26] Z. Lin, J.-S. Yang, and C. Yang, "Grey decision-making for a billiard robot," in *ICSMC*, vol. 6. IEEE, 2004, pp. 5350–5355.

[27] C. Archibald, A. Altman, and Y. Shoham, "Analysis of a winning computational billiards player." in *IJCAI*, vol. 9. Citeseer, 2009, pp. 1377–1382.

[28] M. Smith, "Running the table: An ai for computer billiards," in *AAAI*, vol. 21, no. 1, 2006, p. 994.

[29] J. W. Pan, J. Komar, S. B. K. Sng, and P. W. Kong, "Can a good break shot determine the game outcome in 9-ball?" *Frontiers in psychology*, vol. 12, 2021.

[30] Z. Wang, C. Long, and G. Cong, "Similar sports play retrieval with deep reinforcement learning," *IEEE TKDE*, 2021.

[31] A. S. Weigend, *Time series prediction: forecasting the future and understanding the past*. Routledge, 2018.

[32] R. J. Frank, N. Davey, and S. P. Hunt, "Time series prediction and neural networks," *JIRS*, 2001.

[33] N. I. Sapankevych and R. Sankar, "Time series prediction using support vector machines: a survey," *CIM*, 2009.

[34] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," *ICLR*, 2018.

[35] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond efficient transformer for long sequence time-series forecasting," *arXiv*, 2020.

[36] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces," in *CVPR*, 2016.

[37] C. Ju, Z. Wang, C. Long, X. Zhang, and D. E. Chang, "Interaction-aware kalman neural networks for trajectory prediction," in *IEEE IV*. IEEE, 2020, pp. 1793–1800.

[38] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," in *AAAI*, vol. 31, no. 1, 2017.

[39] T. Che, Y. Li, R. Zhang, R. D. Hjelm, W. Li, Y. Song, and Y. Bengio, "Maximum-likelihood augmented discrete generative adversarial networks," *arXiv*, 2017.

[40] W. Fedus, I. Goodfellow, and A. M. Dai, "Maskgan: better text generation via filling in the_," *arXiv*, 2018.

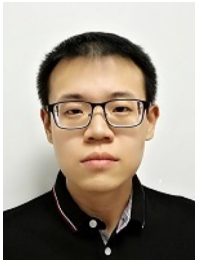
[41] K. Lin, D. Li, X. He, Z. Zhang, and M.-T. Sun, "Adversarial ranking for language generation," *NIPS*, 2017.

[42] W. Nie, N. Narodytska, and A. Patel, "Relgan: Relational generative adversarial networks for text generation," in *ICLR*, 2018.

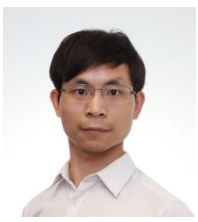
[43] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[44] M. J. Kusner and J. M. Hernández-Lobato, "Gans for sequences of discrete elements with the gumbel-softmax distribution," *arXiv*, 2016.
[45] J. Guo, S. Lu, H. Cai, W. Zhang, Y. Yu, and J. Wang, "Long text generation via adversarial training with leaked information," in *AAAI*, vol. 32, no. 1, 2018.
[46] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *CVPR*, 2017, pp. 652–660.
[47] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *arXiv*, 2017.
[48] P. Arsomngern, C. Long, S. Suwajanakorn, and S. Nutanong, "Self-supervised deep metric learning for pointsets," in *ICDE*, 2021.
[49] H. Su, S. Liu, B. Zheng, X. Zhou, and K. Zheng, "A survey of trajectory distance measures and performance evaluation," *The VLDB Journal*, vol. 29, no. 1, pp. 3–32, 2020.
[50] S. Wang, Z. Bao, J. S. Culpepper, and G. Cong, "A survey on trajectory data management, analytics, and learning," *ACM Computing Surveys (CSUR)*, vol. 54, no. 2, pp. 1–36, 2021.
[51] A. G. Patil, M. Li, M. Fisher, M. Savva, and H. Zhang, "Layoutgmn: Neural graph matching for structural layout similarity," in *CVPR*, 2021, pp. 11 048–11 057.
[52] D. Manandhar, D. Ruta, and J. Collomosse, "Learning structural similarity of user interface layouts using graph networks," in *ECCV*, 2020, pp. 730–746.
[53] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a" siamese" time delay neural network," *NIPS*, vol. 6, pp. 737–744, 1993.
[54] P. Bachman and D. Precup, "Data generation as sequential decision making," *arXiv*, 2015.
[55] D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. Courville, and Y. Bengio, "An actor-critic algorithm for sequence prediction," *arXiv*, 2016.
[56] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," in *SIMBAD*. Springer, 2015, pp. 84–92.
[57] Y. Zhang and B. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," *arXiv*, 2015.
[58] K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *CVPR*, 2015, pp. 5353–5360.

Qianru Zhang is currently pursuing the Ph.D. degree at the department of Computer Science (CS), The University of Hong Kong (HKU) under the supervision of Siu Ming Yiu. Her research interests are data mining and machine learning.



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. His research has been recognized by AISG PhD Fellowship and Google PhD Fellowship.



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.

Siu-Ming Yiu obtained his Ph.D. in Computer Science from the Department of Computer Science, The University of Hong Kong in 1996. He is currently a professor in the same department. His research interests include information security, cryptography, and bioinformatics.