RLRecommender: A Representation-Learning-Based Recommendation Method for Business Process Modeling

Huaqing Wang, Lijie Wen(✉), Li Lin, and Jianmin Wang

School of Software, Tsinghua University, Beijing, China
whq16@mails.tsinghua.edu.cn, veralin1994@gmail.com, {wenlj,jimwang}@tsinghua.edu.cn

Abstract. Most traditional business process recommendation methods cannot deal with complex structures such as interacting loops, and they cannot handle large complex datasets with a great quantity of processes and activities. To address these issues, RLRecommender, a method based on representation learning, is proposed. RLRecommender extracts three kinds of relation sets from the models, both activities and relations between them are projected into a continuous low-dimensional space, and proper activity nodes are recommended by comparing the distances in the space. The experimental results show that our method not only outperforms other baselines on small dataset, but also performs effectively on large dataset.

Keywords: Business process modeling · Ordering relations Representation learning · Recommendation

1 Introduction

Business process models are constructed to describe the prescribed behavior of business processes. One method to improve process modeling is process mining. Nonetheless, most process mining algorithms rely on completely purified event logs and real-world event logs are often noisy in most cases. Hence, process modeling is usually conducted by analysts manually. Nevertheless, modeling a business process from scratch is also highly complex, error-prone and time-consuming. Thus, an accurate and efficient business process recommendation method is required.

Business process recommendation has already been used in both academic research and industry applications, and there are several existing works [1,2]. These traditional methods recommend the next nodes of an uncompleted model by iterating all parts of all processes, and they have two main limitations:

1. These methods cannot deal with complex structures such as interacting loops in Fig. 1 (i.e., B and D, and C and D).

2. Existing methods cannot deal with large complex datasets with abundant processes, whose activities complicatedly relate to each other.

To address these problems, business process repositories should be considered as a whole. We find that business process repositories have several properties. First, quite a few activities appear frequently through most processes. Second, the flows in process models relate activities with each other. Thus, regarding activities and relations between them as nodes and edges, process repositories can be transformed into multi-relational data.

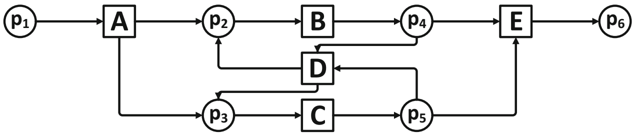

Fig. 1. An example for interacting loops

Recently, machine learning became a hot topic and there are a variety of representation learning models to handle multi-relational data [3]. In representation learning models, multi-relational data is a directed graph whose nodes are entities and edges are relations, and connections are represented as triples (*head entity, relation, tail entity*). Existing representation learning models embed both entities and relations into a continuous low-dimensional vector space [3], and have been proven simple and effective in tasks such as entity prediction.

Motivated by this idea, RLRecommender, a representation-learning-based recommendation method for process modeling, is proposed. To the best of our knowledge, this is the first work that resolves process recommendation problem using representation learning.

The remainder is organized as follows. In Sect. 2, we introduce RLRecommender in detail. Experimental results are given in Sect. 3. Section 4 concludes the paper and sketches future work.

2 The Representation-Learning-Based Method

Our method RLRecommender is composed of three phases: preprocessing, training and recommending as shown in Fig. 2. In this section, these three phases will be presented one by one.

2.1 Preprocessing

Our first phase focuses on extracting relations between activities from Petri nets. The definition of Petri nets can be found in [4]. Since the generated relation sets can be saved at back-end, this part is conducted only once.

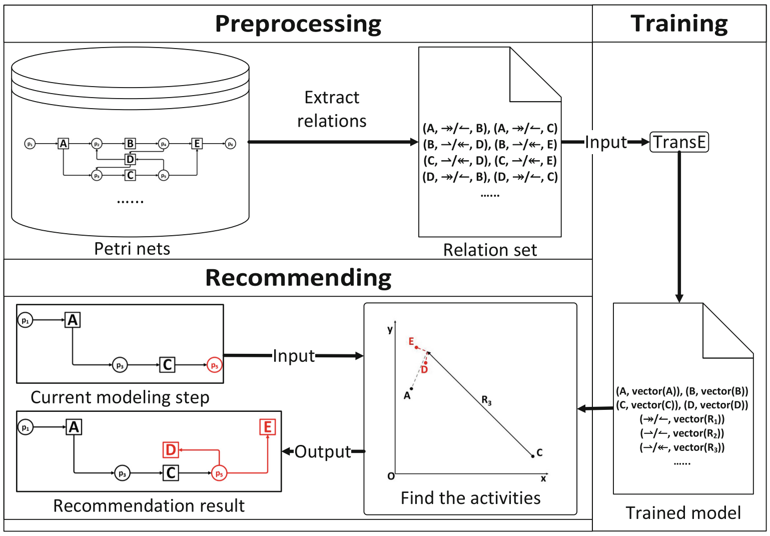

Fig. 2. Overview of our method RLRecommender

"Direct After" Relations. For relation extraction, our first strategy (i.e. RLafter) is to use only "direct after" relations, which is defined as follow.

Definition 1 *(Direct after). In a Petri net $N = (P, T, F)$, two transitions $x, y \in T$ are in "direct after" relation (denoted as $x \rightarrow y$) $\iff$ $\exists p \in x\bullet$, s.t. $y \in p\bullet$.*

The "direct after" relations can correctly depict the connections of transitions. However, it cannot tell the differences between AND-splits and XOR-splits.

"Direct Causal" Relations. A combination of "direct causal" and "direct inverse causal" is used as the relations as the second strategy (i.e. RLcausal). We first define the two "direct causal" relations.

Definition 2 *(Always/sometimes direct causal). In a Petri net $N = (P, T, F)$:*

- *two transitions $x, y \in T$ are in "always direct causal" relation (denoted as $x \twoheadrightarrow y$) $\iff$ $\exists p \in x\bullet$, s.t. $p\bullet = \{y\}$,*
- *two transitions $x, y \in T$ are in "sometimes direct causal" relation (denoted as $x \rightharpoonup y$) $\iff$ $x \rightarrow y \wedge x \not\twoheadrightarrow y$.*

Two "direct inverse causal" relations are defined in a similar way (i.e. "always inverse direct causal" relation (denoted as $x \twoheadleftarrow y$) and "sometimes inverse direct causal" relation (denoted as $x \leftharpoonup y$)).

The two "direct causal" relations describe the possible orders of transitions, but they fail to distinguish transition groups $\{A, B\}$ and $\{C, D\}$ of the model in Fig. 3.

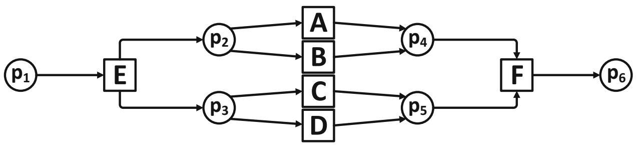

Fig. 3. An example that two "direct causal" relations cannot handle

"Direct Concurrent" Relations. We introduce "direct concurrent" relation in the following. A combination of two "direct causal" relations and "direct concurrent" relation is used as the third strategy (i.e. RLconcurrent).

Definition 3 *(Always/never/sometimes direct concurrent). In a Petri net $N = (P, T, F)$:*

- *two transitions $x, y \in T$ are in "never direct concurrent" relation (denoted as $x \nparallel y$) $\iff \exists p \in P$, s.t. $x, y \in p\bullet$,*
- *two transitions $x, y \in T$ are in "sometimes direct concurrent" relation (denoted as $x \Vdash y$) $\iff \exists p_1, p_2 \in t\bullet$, $t \in T, p_1 \neq p_2$, s.t. $(|p_1\bullet| > 1 \vee |p_2\bullet| > 1) \wedge x \in p_1\bullet \wedge y \in p_2\bullet \wedge \neg(x \nparallel y)$,*
- *two transitions $x, y \in T$ are in "always direct concurrent" relation (denoted as $x \parallel y$) $\iff \exists p_1, p_2 \in t\bullet$, $t \in T, p_1 \neq p_2$, s.t. $p_1\bullet = \{x\} \wedge p_2\bullet = \{y\} \wedge \neg(x \nparallel y) \wedge \neg(x \Vdash y)$.*

When generating relation set, each "direct concurrent" relation would only be recorded once between any pair of activities in the specific order such as dictionary order (e.g. $B \parallel C$ is recorded while $C \parallel B$ not).

The relation sets extracted from the model in Fig. 1 are shown in Table 1.

Table 1. The relation sets (direct after|direct causal/direct inverse causal|direct concurrent) extracted from the process model in Fig. 1

	A	B	C	D	E
A		$\rightarrow \mid \twoheadrightarrow / \leftharpoonup \mid$	$\rightarrow \mid \twoheadrightarrow / \leftharpoonup \mid$		
B			$\mid \quad \mid \quad \parallel$	$\rightarrow \mid \rightharpoonup / \twoheadleftarrow \mid$	$\rightarrow \mid \rightharpoonup / \twoheadleftarrow \mid$
C				$\rightarrow \mid \rightharpoonup / \twoheadleftarrow \mid$	$\rightarrow \mid \rightharpoonup / \twoheadleftarrow \mid$
D		$\rightarrow \mid \twoheadrightarrow / \leftharpoonup \mid$	$\rightarrow \mid \twoheadrightarrow / \leftharpoonup \mid$		$\mid \quad \mid \quad \nparallel$
E					

2.2 Training a Representation Learning Model

The second phase of our method is to learn a representation learning model through generated relation sets. This part can be done at back-end and the trained model can be saved and used for recommendation. The training model we use is based on TransE [3].

For the training relation set T_r, each triple (*precursor activity, relation, successor activity*) (denoted as (p, r, s)) is composed of two activities p, $s \in \mathscr{A}$ (the set of activities) and one relation $r \in \mathscr{R}$ (the set of relations). The embeddings take values in $\mathbb{R}^k$ ($\mathbb{R}^k$ means continuous k-dimensional vector space where k is a model hyperparameter) and are denoted with the same letters, in boldface characters (i.e. $\boldsymbol{p}$, $\boldsymbol{r}$, $\boldsymbol{s} \in \mathbb{R}^k$). We want $\boldsymbol{p} + \boldsymbol{r} \approx \boldsymbol{s}$ when (p, r, s) holds and $\boldsymbol{p} + \boldsymbol{r}$ is far away from $\boldsymbol{s}$ otherwise. We use dissimilarity measure function d as the distance measurement, and in this paper d stands for L1-norm (i.e. Manhattan distance) while there are other choices such as L2-norm (i.e. Euclidean metric).

We minimize the margin-based loss function as the objective for training:

$$L = \sum_{(p,r,s) \in T_r} \sum_{(p^{'},r,s^{'}) \in T_r^{'}} max(\gamma + d(\boldsymbol{p} + \boldsymbol{r}, \boldsymbol{s}) - d(\boldsymbol{p}^{'} + \boldsymbol{r}, \boldsymbol{s}^{'}), 0) \tag{1}$$

$$T_r^{'} = \{(p^{'}, r, s) | p^{'} \in \mathscr{A}\} \cup \{(p, r, s^{'}) | s^{'} \in \mathscr{A}\} - T_r \tag{2}$$

where γ is a margin hyperparameter, and $T_r^{'}$ is a negative sampling set of training set T_r. The triples in $T_r^{'}$ are generated by replacing p or s in a triple in T_r by another activity. Note that a triple already in T_r will not be generated in $T_r^{'}$. The loss function L can reach minimum value only when $d(\boldsymbol{p} + \boldsymbol{r}, \boldsymbol{s}) < d(\boldsymbol{p}^{'} + \boldsymbol{r}, \boldsymbol{s}^{'})$ for most (p, r, s).

2.3 Recommending

Our final phase is to recommend the activities using the trained model. After training, the embedded vectors for activities and relations can be used for recommendation by predicting the successor activity in a triple. During process modeling, the most suitable activities are recommended by traversing all activity vectors and calculating the distance with the sum of the vector of the previous activity and the vector of the proper relation. Notice that the proper relation can be either specified by user or chosen by the recommendation system.

More specifically, during the construction of one process, suppose the previous activity is $p_i \in \mathscr{A}$. For each relation set, we first check if each $r_i \in \mathscr{R}$ is valid after current position. Then we calculate the vector distance $d(\boldsymbol{p}_i + \boldsymbol{r}_i, \boldsymbol{s}_i)$ for each $s_i \in \mathscr{A}$. We then recommend K activities whose $d(\boldsymbol{p}_i + \boldsymbol{r}_i, \boldsymbol{s}_i)$ are the smallest. Here K is a parameter determined by user.

3 Experimental Evaluation

In this section, we first introduce the datasets and experiment settings. Then the results on two different datasets are presented. The code, datasets and results of our experiment are publicly accessible from GitHub[1].

[1] https://github.com/THUBPM/RLRecommender/.

3.1 Datasets and Experiment Settings

Datasets. In this paper, two datasets are used to evaluate our method, including large real dataset (LRD) [5] and small real dataset (SRD) [2]. The statistics of them are listed in Table 2.

Notice that for LRD, we only use BPMN 1.1, BPMN 2.0 [6] and Petri net models. All models are lowercased and only letters are retained, and non-English models and the ones without activity names are deleted from the dataset. BPMN models are transformed to Petri nets using the method in [7].

Table 2. Statistics of the two datasets used in the experiments

	LRD	SRD
#Processes	23576	221
#Distinct activities	36801	52
#Activities per process	2-380	5-15
AND/XOR-split/join	True	True
Cycle	True	True
Other complex structures	True	False

Benchmarks. We use *HitRate* and *F1 score* to estimate the performance of each method. For each position in a model, a list of K activity nodes are recommended, denoted by A_r. The set of activities that the real process actually has at the current position is denoted by A_t.

HitRate is the percentage of hits, which is defined as follow:

$$HitRate = \#hit / \#rec \tag{3}$$

where $\#hit$ is the number of times when $A_r \cap A_t \neq \varnothing$, and $\#rec$ is the number of recommendations.

F1 score is the harmonic mean of *precision* and *recall*, where *precision* and *recall* are the percentage of correctly recommended activities in recommended activity set and ground-truth activity set respectively. *F1 score* is defined as follow:

$$F1 = (2 \times Precision \times Recall) \,/\, (Precision + Recall) \tag{4}$$

$$Precision = (\Sigma|A_r \cap A_t|) \,/\, (\Sigma|A_r|) \tag{5}$$

$$Recall = (\Sigma|A_r \cap A_t|) \,/\, (\Sigma|A_t|) \tag{6}$$

Experiment Settings. Since the small dataset, benchmarks and experiment protocol we use are the same as those in [2], we use the settings determined in this paper for other state-of-the-art approaches.

For our method, we set the embedding dimension k to 200, the margin value γ in the loss function to 1.0, and the learning rate to 0.001. For the small and large datasets, we train our model for 5000 and 1000 rounds, respectively.

The user could set K to a proper value due to the size of the dataset. We conduct our experiment when recommendation number K is 1 to 5, we also show the result for $K = 10$ on LRD.

Experiments are conducted on an Intel Xeon E5-2620 v4@2.10 GHz CPU computer with 128 G RAM, and three GeForce GTX 1080Ti GPU are used.

Experiment Protocol. We perform all experiments on SRD using five-fold cross-validation. For LRD, we partition the 23576 models into 3041 models for the testing set, 1759 models for the validation set and the rest for the training set. For each model in the testing set, we conduct the experiment step by step for each position.

For SRD, our method takes 2–3 ms per recommendation. For LRD, our method takes 0.1–0.3 s per recommendation.

3.2 Evaluation Based on Large Complex Real-Life Dataset

The other recommendation system failed to deal with LRD while our RLRecommender is naturally capable of this situation. So we only conduct our methods on this dataset. Figure 4 shows the experimental results. We can conclude the following:

1. All three methods achieve good accuracy performance. Since this dataset has 36801 distinct activities, we also test the situation when $K = 10$, and the *HitRate* is up to 89.2%, which is highly acceptable in real life.
2. Our methods cannot handle the situation when $K = 1$, because the large dataset contains too many activities and all models are complicated. Nevertheless, our methods perform well when $K \geq 2$.

3.3 Evaluation Based on Small Real-Life Dataset

We compared with three methods MCSD, xGED and xSED in [2] on SRD. The results are shown in Fig. 5, from which we can observe that:

1. The RLRecommender based on "causal" and "concurrent" relations outperforms other methods on *HitRate* as recommendation number K increases.
2. For *F1 score*, our methods outperform the others most of the time. Especially, when $K = 5$, the improvement of *F1 score* is up to 14.4%.
3. The methods based on "causal" and "concurrent" outperform "after" on *HitRate*, while the latter performs better on *F1 score* when K increases.

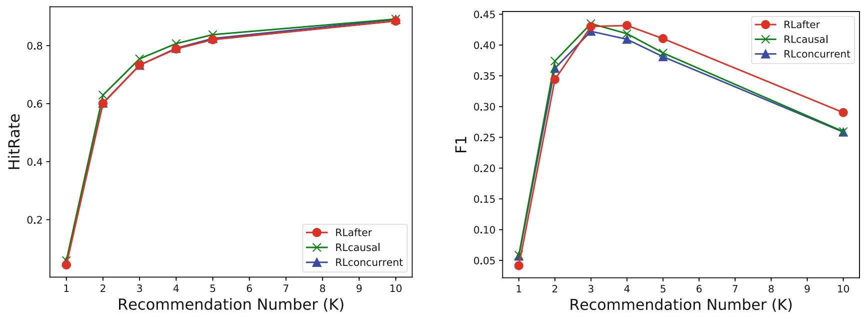

Fig. 4. Experimental results on LRD

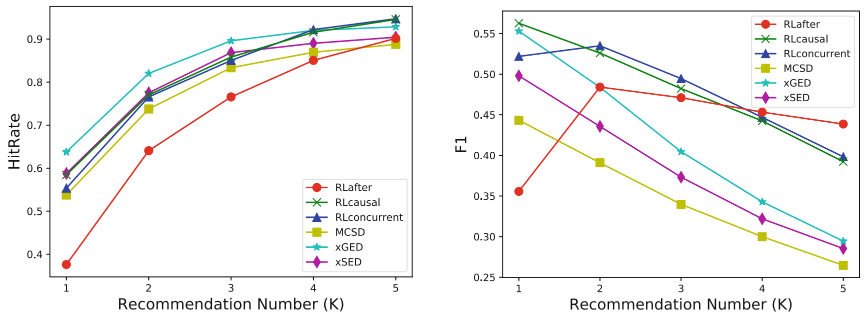

Fig. 5. Experimental results on SRD

4. When K is small, such as $K = 1$, traditional methods have higher *HitRate* since they iterate all structures of the models and this dataset is small and do not contain complex structures. Nevertheless, our methods still achieve better performance on *F1 score*.

4 Conclusion and Future Work

A representation-learning-based recommendation method named RLRecommender is proposed and it is the first recommendation method for process modeling using representation learning. It first extracts the relations between activities with three strategies. Relation sets are then used to train the representation learning model. The trained model is used in recommendation by finding the nearest vector for each modeling step. Experiments show that RLRecommender not only outperforms the state-of-the-art recommendation algorithms on small real dataset, but also performs uniquely well on large real dataset.

Our future work would mainly focus on improving the accuracy by modifying the current representation learning model. The current model only considers one previous activity during recommendation, while several activities can be used during one recommendation. We can also generate other kinds of relation set which may have positive effect on performance. Preliminary clustering can also be conducted on activity labels in order to make recommendations based on activities that relate to one another.

Acknowledgements. The work was supported by the National Key Research and Development Program of China (No. 2016YFB1001101), the National Nature Science Foundation of China (No. 61472207, No. 71690231), and BNRist.

References

1. Zhang, J., Liu, Q., Xu, K.: FlowRecommender: a workflow recommendation technique for process provenance. In: Proceedings of the Eighth Australasian Data Mining Conference-Volume 101, pp. 55–61. Australian Computer Society, Inc. (2009)
2. Deng, S., et al.: A recommendation system to facilitate business process modeling. IEEE Trans. Cybern. **47**(6), 1380–1394 (2017)
3. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Advances in Neural Information Processing Systems, pp. 2787–2795 (2013)
4. Murata, T.: Petri nets: properties, analysis and applications. Proceed. IEEE **77**(4), 541–580 (1989)
5. BPM Academic Initiative: BPMAI process model collection. http://bpmai.org/
6. OMG: Business Process Modeling Notation (BPMN) Version 2.0. (2011)
7. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Inform. Softw. Technol. **50**(12), 1281–1294 (2008)