# Detecting long-range cause-and-effect relationships in game provenance graphs with graph-based representation learning

Sidney Araujo Melo
*Instituto de Computação*
*Universidade Federal Fluminense*
Niterói, Brazil
sidneymelo@id.uff.br

Esteban Walter Gonzalez Clua
*Instituto de Computação*
*Universidade Federal Fluminense*
Niterói, Brazil
esteban@ic.uff.br

Aline Paes
*Instituto de Computação*
*Universidade Federal Fluminense*
Niterói, Brazil
alinepaes@ic.uff.br

***Abstract*—Success in Game Analytics tasks demands developers to identify what is happening in a game (an effect) and track its causes. Thus, game provenance graph tools have been proposed to capture cause-and-effect relationships. However, as a downside gathering provenance still may demand huge coding efforts for long-range relationships. In this dissertation, we contributed with a framework named PingUMiL for automatic edge detection using graph representation learning, aiming at alleviating the coding effort. We evaluated the generalization capacity of PingUMiL when learning from similar games and compared its performance to classical machine learning methods. The experiments conducted on two racing games show that PingUMiL (1) outperforms classical machine learning, (2) can be used for inference in unobserved data and (3) enhances detection cross games.**

***Index Terms*—provenance graph, representation learning, machine learning, graph embeddings**

## I. Introduction

A game metric is a quantitative measure of attributes of one or more objects that operate in the context of games. The main idea behind the use of game metrics is to form a basis for analysis using variables and features to support decision-making during both the game design and game development [1].

However, to obtain such metrics, it is necessary to track and remotely gather data [1], an essential task for understanding the player's behavior [2]. By using a structured, relational representation one can naturally handle entities, their properties and their relationships in a game. With that in mind, provenance graphs have been proposed and successfully adapted to record game session history while still denoting the elements of the game and the causal relationships between them [3]. However, there are still few initiatives to discover information and patterns from game provenance, which could bring insights into the process of game design and game development.

Pattern recognition and information discovery are examples of tasks realized by several Machine Learning (ML) techniques. However, ML methods are dependent on data representation on which they are applied [4]. For this reason, learning representations of the data that make it easier to extract useful information in machine learning tasks, has become a field in itself in the machine learning community [4]. Recently, representation learning for structured data has become popular and several techniques have been proposed to generate data representations more suitable for downstream machine learning tasks.

In the context of game provenance, all cause-and-effect relationships in game provenance graphs are materialized as edges, which represent the influence between game elements. Existing game provenance solutions [3] improve the extraction and structured representation of game sessions. Direct influence edges are used to represent cause-and-effect relationships of sequential events or states. Still, there is also a need for domain-specific provenance tracking functions implementation, since different games might have different mechanics. One of the needs for domain-specific functions is *indirect influences*, *i.e.*, a causal relation between non-consecutive events, which represents a long-range cause-and-effect relationship. However, functions that track indirect influences might grow in complexity if developers need to identify the influence between nodes instantiated along distant timestamps and/or with large path distances. Other difficulties may arise when influences are defined by large sets of rules or formulas.

In this work, we propose a novel framework named PingUMiL that uses a recent graph-based representation learning technique called GraphSAGE [5] in order to detect indirect influence edges and improve game provenance data extraction. The main objective of this work is to develop a framework for detecting long-range cause-and-effect relationships by leveraging methods of representation learning in game provenance graphs. The secondary objectives are (1) to propose a conceptual framework for Machine Learning tasks in game provenance graphs based on graph representation learning, (2) evaluate if graph representation learning improves the detection of influence edges compared to classical machine learning techniques with raw features, and (3) evaluate if

The authors would like to thank the Brazilian Research agencies CAPES, CNPQ, and FAPERJ.

the framework achieves generalization capability that allows inferring long-range relationships in unobserved but similar provenance graphs.

### A. Contributions

To the best of our knowledge, this is the first attempt to combine graph representation learning, provenance graphs and digital games, proposed as the PingUMiL framework. We conduct extensive experiments to evaluate the performance of the proposed framework and its resulting models. Also, our proposal of the PingUMiL framework for edge detection has been published in Elsevier's Entertainment Computing [6].

The PingUMiL framework hereby presented was the first step towards a more generic framework for ML assisted Game Analytics and Game Development. One of the intended tasks is player behavior profiling, which can be approached as a clustering task. Our recent work on player behavior profiling using PingUMiL has been accepted as a full paper at Foundations of Digital Games 2020 (FDG) [7].

## II. Background

This chapter presents two fundamental topics for understanding PingUMiL: Provenance in games and Learning graphs embeddings.

Provenance is well understood in the context of art and digital libraries, where it respectively refers to the documented history of an art object, or the documentation of processes in a digital object's life cycle [8]. Provenance of objects is represented by a directed acyclic graph enriched with annotation capturing further information pertaining to execution [9].

The adoption of data provenance in the context of games was first proposed by Kohwalter et al. in the PinG (Provenance in Games) framework [8]. The authors define a mapping between game elements and each type of node of a provenance graph. Summarizing the proposed mapping, one can say that players, enemies and Non-Playable Characters (NPCs) are mapped as *Agent* nodes; items, weapons, potions, static obstacles or any other object used in the game are mapped as *Entity* nodes; and actions and events such as attacking, jumping or interacting with an item are mapped as *Activity* nodes. Causal relationships between game elements are mapped into edges connecting their respective nodes, resulting in a game data provenance graph.

The PinG framework was later implemented as generic framework for the Unity game engine called Provenance in Games for Unity (PinGU) [3]. The PinGU plugin is a domain-independent and low-coupling solution, written in UnityScript that provides easier provenance extraction, requiring minimal coding in the game's existing components [10] to obtain game provenance graphs.

Regarding graphs, it is known that such structures often represent real-world data containing relationships between entities. Thus, many graph representation learning methods have been proposed to learn a mapping that embeds nodes as points in a low-dimensional space. The goal is to optimize this mapping so that geometric relationships in this learned space reflect the structure of the original graph [11].

Earliest methods generate vector representations for each node independently not considering node attributes during the encoding. This is a major drawback since node attributes can be highly relevant to a node's representation. Thus, convolutional approaches have been proposed to solve this problem. In general, these approaches generate a node embedding iteratively. At the first step, the node embedding is initialized with the node's features. At each iteration, the nodes aggregate their neighbors' embeddings, generating new embeddings. These *neighborhood aggregation methods* determine the node embedding according to its surrounding neighborhood attributes.

The GraphSAGE (Graph Sample and Aggregate) framework is a convolutional approach that uses graph-based loss functions to fine-tune weight matrices and aggregation functions' parameters. The loss function enforce similarities on representations of nearby nodes. We opted for GraphSAGE due to the following reasons:

- As an inductive approach, it facilitates generalization across graphs.
- It comprises an unsupervised setting, which emulates situations where node features are provided to downstream machine learning applications.
- It has multiple aggregator architectures, i.e., functions defined for aggregating node embeddings (Mean, LSTM, Pooling, GCN).

## III. PingUMiL: A framework for completing game provenance based on graph-based representation learning

The general procedure of the framework proposed in this work, named as PingUMil[1] is to induce a latent representation of the edges in a provenance graph so that they become the examples in a classification task, aiming at inducing a model that discriminates whether an edge is of a specific type or not. An overview of PingUMil is shown in Fig. 1.

A set of provenance graphs is the input to the whole framework. These graphs must be preprocessed due to node heterogeneity and the definition of balanced sets of positive and negative example edges for the downstream classification tasks. After the preprocessing, graphs are fed to an embedding generation technique which outputs node embeddings. Then, edges from positive and negative example edge sets are encoded using their connecting nodes' embeddings. Encoded edges are finally fed to a classifier training algorithm. The resulting classifier should be able to detect edges similar to the examples edge sets and generalize this detection capability to semantically analogous edges in graphs not seen in the training phase.

Consider a graph $G = (V, E, T_v)$. A node $v$ is defined as $v \in V = (x_v, t_v)$ where $x_v \in \mathbb{R}^{n(\tau(v))}$ is the node feature vector, $\tau$ is a mapping function that maps node into a type $t \in T_v$ and $n$ is a function that maps a type of node $t$

[1] *https://github.com/sidneyaraujomelo/PingUMiL*

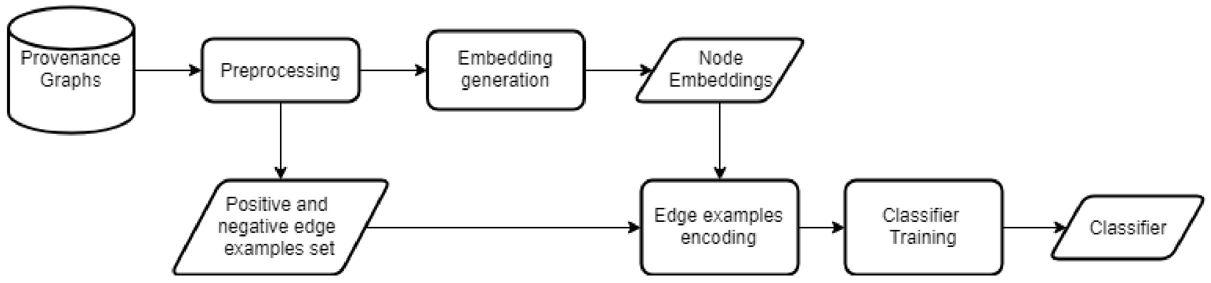

Fig. 1. Overview of the proposed framework.

into an integer that represents the dimension of the type $t$. $T_v = \{t_i \in \mathbb{R}^{d_i}, i = 1, ..., |T_v|\}$ is the set of node types, where $d_i$ is the number of dimensions of the type $t_i$. Every dimension of $t_i$ represents a label $l \in L$ for node attributes. Every type $t$ of a node defines a set of labels so that each label corresponds to a value in the node feature vector. Using this notation, an activity node with attributes $HP = 10$ and $Speed = 5$ and provenance label $Running$ could have, for example, a type $t_x = (provenance\_type, provenance\_label, hp, speed)$ and $x_v = (activity, running, 10, 5)$. Edges are defined as $(v, v') \in V \times V$. Let $L_v$ be the set of node's provenance label values, we define edge labels as $L_e = l_i \rightarrow l_j$ where $l_i, l_j \in L_v$.

Embedding generation is realized through GraphSAGE, which takes as input graphs with homogeneous nodes, i.e., nodes with the same set of features. Provenance graphs with heterogeneous nodes must, therefore, be mapped into homogeneous nodes. That can be achieved by creating a new type $t' = \{l \mid l \in t_i, i = 1, ..., |T_v|\}$. A homogeneous node is defined as $v' = (x_{v'}, t')$, where $x_{v'} \in \mathbb{R}^{|t'|}$ is composed of the original values of node feature vector $x_v$, respecting labeling order and default values for previously unaddressed features. Once all the nodes are homogeneous, any non-numeric attribute must be mapped into one-hot-vector representations. Using the unsupervised setting of GraphSAGE with any of the previously mentioned architectures, we obtain the node embeddings that are going to pose as features to a machine learning classifier. Since generated node embeddings are vectors all with the same dimension, we generate edge embeddings using a simple function $embed\_edge(v_1, v_2) = f(x_{v_1}, x_{v_2})$ where $v_1$ and $v_2$ are the connecting nodes of edge $e$ and $f$ is an operation such as concatenation, element-wise sum or cross product. Finally, a classifier is trained with edge embeddings and their classes. The resulting model can then be used to detect the targeted edges.

## IV. Experimental Results

We used two racing game prototypes as case studies, in which a single player drives along a single track. The first game is Car Tutorial Unity (CT)[2], and the second one is Arcade Car Physics (AC)[3]. Car Tutorial Unity (CT) is a free prototype asset for racing games, designed for Unity 3.x, i.e., an older version of the game engine. The game is single player, contains only a single track and uses Unity's native car physics. Arcade Car Physics (AC) is an open source prototype implemented in Unity 2018. Like CT, this prototype is single player and has only one track. However, AC implements several algorithms over or instead native engine physics.

Both games were made available for 4 playtesters, all male, age between 20 and 30 years old, experienced with racing games and considered themselves hardcore gamers. For CT, 10 game session graphs were extracted, which in total contain 9194 nodes and 47497 edges. For AC, 3 game session graphs were extracted, which in total contain 4146 nodes and 21397 edges. Regarding long-range cause-and-effect relationships, we target edges with the following labels: Crash $\rightarrow$ Crash, Crash $\rightarrow$ LostControl, Crash $\rightarrow$ Scraped, Flying $\rightarrow$ Crash, Flying $\rightarrow$ Landing, Flying $\rightarrow$ LostControl, Flying $\rightarrow$ Scraped, HandBrake $\rightarrow$ LostControl, Scraped $\rightarrow$ Crash and Scraped $\rightarrow$ Scraped.

We test the performance of PingUMiL generated classifier models for AC and CT. Every model uses a combination of the following options:

- Aggregation architecture: LSTM, MaxPool, MeanPool, Mean, GCN
- Edge encoding functions: Mult(Elementwise Multiplication) and Cat (Concatenation)
- Classifier method: MLP (multilayer perceptron neural network), SGD (Stochastic Gradient Descent), SVM (Support Vector Machine).

Precision, recall and f1-score are measured for both study cases encoded edges in a stratified k-fold cross-validation setting. In our results, we show the average mean of all folds measured metrics. We compare the generated classifiers from PingUMiL against traditional classification methods using the raw features and the same classifiers as before (SVM, MLP, and SGD). We call raw features the attribute values attached to each node in the original provenance graph, such as speed, acceleration, position, etc.

For CT, the best average performance was achieved by LSTM-based aggregation method, concatenation as edge encoder and an MLP neural network with approximately 67% on all metrics, which implies in a 13% gain over the best baseline (Raw Features with SGD). The best average precision

[2]https://assetstore.unity.com/packages/templates/tutorials/car-tutorial-unity-3-x-only-10
[3]https://github.com/SergeyMakeev/ArcadeCarPhysics

TABLE I
GENERALIZATION RESULTS PER EDGE TYPE

| Edge Type | AC | | CTAC | |
|---|---|---|---|---|
| | Precision (Var) | Recall (Var) | Precision (Var) | Recall (Var) |
| Flying → Landing | 0.7325 (0.007) | 0.575 (0.001) | 0.673 (0.014) | 0.655 (0.012) |
| Flying → Crash | 0.635 (0.004) | 0.645 (0.003) | 0.643 (0.014) | 0.73 (0.037) |
| Scraped → Crash | 0.58 (0.015) | 0.683 (0.023) | 0.605 (0.006) | 0.723 (0.075) |

inside a fold was achieved by PingUMiL.LSTM + Mult + MLP(100,1) with 72,3%, while the best average recall and F1 was achieved by PingUMiL.LSTM + Cat + MLP(100,1) with 72,2% and 71,9% respectively. These top performances suggest that PingUMiL generated models capable of detecting target edges.

For AC, the best average performance was obtained by LSTM-based aggregation method, concatenation as edge encoder and an MLP neural network with approximately 70% on all metrics, which implies in a 10% gain over the best baseline (Raw Features with SGD). The best average precision and F1 inside a fold were achieved by PingUMiL.GCN + Cat + MLP(256,1) with 77,1% and 74,3%, respectively, on fold 4, while the best average recall was achieved by PingUMiL.MeanPool + Cat + MLP(256,1) with 74% on fold 4. These metrics corroborate the edge detection capability of PingUMiL models.

Finally, we trained a classifier with CT and AC data and applied it into AC data to observe generalization capabilities in the model, which we named CTAC. Even though the models did not benefit from such configuration on overall performance statistics, a deeper analysis shown that it enhanced the metrics on edges with fewer examples, as observed in Table I.

Edges of type Flying → Landing presented gain on recall metrics (8%) and a decrease in precision metrics (approximately 6%). CT training edges add 84 Flying → Landing examples. These results means that edges from CT enhanced the model's comprehension of what defines a negative example at the cost of complicating the comprehension of what defines a positive example. As we believe a high recall value is more essential for provenance graph enhancement, this result shows that for some type of edges, a model generated using different games might lead to improvements in preferred metrics.

Edges of type Flying → Crash and Scraped → Crash presented gain on both recall (1% and 2.5% respectively) and precision (8.5% and 4% respectively) metrics. For both edge types, the number of CT examples is more than twice the number of AC examples. This result corroborates to a positive answer for secondary objective 3, in which the use of another similar game data improves the capacity to infer long-range influences.

Finally, we conclude from this generalization across games experiment that simply feeding new edges from a similar game does not guarantee overall model's quality enhancement for the racing games used in the experiment. However, PingUMiL models could be enhanced by fine-tuning the model with some types of edges. Further investigations about quality enhancement through generalization using other types of games is suggested as future work.

## V. CONCLUSION

Results from CT and AC show that machine learning techniques are able to detect influence between game components represented as edges in a game provenance graph, given that the models achieved above 77% averaged precision. Also, PingUMiL best performance presents a gain of at least 10% over classical machine learning approaches. Moreover, data from different games of a similar genre can be used to fine-tune the detection of edges with fewer examples.

## REFERENCES

[1] M. S. El-Nasr, A. Drachen, and A. Canossa, *Game analytics*. Springer, 2016.

[2] C. Bauckhage, K. Kersting, R. Sifa, C. Thurau, A. Drachen, and A. Canossa, "How players lose interest in playing a game: An empirical study based on distributions of total playing times," in *Computational Intelligence and Games (CIG), 2012 IEEE conference on*. IEEE, 2012, pp. 139–146.

[3] T. C. Kohwalter, L. G. P. Murta, and E. W. G. Clua, "Capturing game telemetry with provenance," in *2017 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE, 2017, pp. 66–75.

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

[5] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1025–1035.

[6] S. A. Melo, A. Paes, E. W. G. Clua, T. C. Kohwalter, and L. G. P. Murta, "Detecting long-range cause-effect relationships in game provenance graphs with graph-based representation learning," *Entertainment Computing*, vol. 32, p. 100318, 2019.

[7] S. A. Melo, T. C. Kohwalter, E. W. G. Clua, A. Paes, and L. G. P. Murta, "Player behavior profiling through provenance graphs and representation learning," in *To appear in Proceedings of the 15th International Conference on the Foundations of Digital Games*, 2020.

[8] T. Kohwalter, E. Clua, and L. Murta, "Provenance in games," in *Brazilian Symposium on Games and Digital Entertainment (SBGAMES)*, 2012, p. 11.

[9] L. Moreau, B. Clifford, J. Freire, J. Futrelle, Y. Gil, P. Groth, N. Kwasnikowska, S. Miles, P. Missier, J. Myers *et al.*, "The open provenance model core specification (v1. 1)," *Future generation computer systems*, vol. 27, no. 6, pp. 743–756, 2011.

[10] T. C. Kohwalter, F. M. de Azeredo Figueira, E. A. de Lima Serdeiro, J. R. da Silva Junior, L. G. P. Murta, and E. W. G. Clua, "Understanding game sessions through provenance," *Entertainment Computing*, vol. 27, pp. 110–127, 2018.

[11] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Engineering Bulletin*, 2017.