Provchastic: Understanding and predicting game events using provenance[⋆]

Troy C. Kohwalter[1][0000−0002−5183−473X], Leonardo G. P. Murta[1][0000−0002−5173−1247], and Esteban W. G. Clua[1][0000−0001−5650−1718]

Fluminense Federal University, Niterói RJ 24210-346, BR {troy,leomurta, esteban}@ic.uff.br

Abstract. Game analytics became a very popular and strategic tool for business intelligence in the game industry. One of the many aspects of game analytics is predictive analytics, which generates predictive models using statistics derived from game sessions to predict future events. The generation of predictive models is of great interest in the context of game analytics with numerous applications for games, such as predicting player behavior, the sequence of future events, and win probabilities. Recently, a novel approach emerged for capturing and storing data from game sessions using provenance, which encodes cause and effect relationships together with the telemetry data. In this work, we propose a stochastic approach for game analytics based on that novel game provenance information. This approach unifies all gathered provenance data from different game sessions to create a probabilistic graph that determines the sequence of possible events using the commonly known stochastic model of Markov chains and also allows for understanding the reasons to reach a specific state. We integrated our solution with an existing open-source provenance visualization tool and provided a case study using real data for validation. We could observe that it is possible to create probabilistic models using provenance graphs for short and long predictions and to understand how to reach a specific state.

Keywords: Provenance Graph, Predictive analytics, Markov Chains, Stochastic model

1 Introduction

Game analytics became an emerging field that is very popular and important for business intelligence in the game industry. It provides a wealth of information for game designers, including feedback about design and gameplay mechanics, player experience, production performance, and even market reaction. Thus, the main goal of game analytics is to support the decision-making process at the operational, tactical, and strategic levels for game development. Moreover, it is the main source of business intelligence for the game industry [5].

[⋆] Supported by CAPES, CNPq, and Faperj.

One of the many aspects of game analytics is *predictive analytics* [6], which generates predictive models using statistics derived from datasets to generate statistical scores to predict future events [2]. Predictive analytics has a lot of usages in the game domain. It can be used to predict player behavior, sequence of future events based on the current game state, win probabilities, strategies on competitive games, or even be used in monetization decisions and increase the game's revenue [1].

Thus, the generation of predictive models is of great interest in the context of game analytics, and is not a new field [7,12,13,15–17]. For example, Dereszynski et al. [4] presented a probabilistic framework for RTS games based on hidden Markov models by counting the number of units and buildings to predict build order. Yang et al. [19] proposed a data-driven approach to discover combat strategies of winning teams in MOBA games. Schubert et al. [14] presented a predicting model for *Dota 2* to predict match results based on the encounters within the game session. Cleghern, Zach, et al. [3] introduced an approach to forecast changes in the hero's health in the MOBA game *Dota 2* by observing past game session data. However, none of these works take into consideration contextual information that might impact in the outcome and use game metrics over the course of the game session for predictions.

Recently, Kohwalter et al. [10] proposed a novel approach named PinGU[1] [10] for capturing and storing provenance data from a game session based on the Provenance in Games conceptual framework [9]. The wealth of provenance data collected during a game session is fundamental for understanding the mistakes made as well as reproducing the same results at a later moment. However, up to this point, Kohwalter et al. only focused only on analyzing a single provenance graph at a time, not comparing different game sessions to understand why some paths taken by players lead to failure while others succeeded in reaching their goals. Thus, this led to an opportunity to further explore the applications of this promising approach for game provenance data due to its wealth of information, resulting in the following research question:

RQ:*Does the use of provenance obtained from multiple game sessions support predictions and understanding of events for future game sessions?*

Therefore, in this work, we propose a stochastic approach for game analytics based on the tracked provenance data. This approach merges the tracked provenance data from multiple game sessions into a stochastic graph. This stochastic graph, represented as Markov chains [8], determines the sequence of possible events, taking advantage of the graph nature of the provenance data to navigate this stochastic model for both prediction and understanding causes for the events.

Our solution is compatible with an existing open source provenance visualization tool and provide a case study using real game provenance data. We believe that our stochastic graph supports a variety of usages for game analytics since it is based on an well-known and vastly used model of Markov Chains.

[1] https://github.com/gems-uff/ping

Applications can include: AI behavior, human assistant, a decision process for interactive storytelling, among other possibilities.

This paper is organized as follows: Section 2 presents the existing work in the literature. Section 3 presents our proposed approach for generating a stochastic graph based on provenance data. Section 4 presents our case of study. Finally, Section 5 concludes this work.

2 Related Work

There are many studies related to predicting outcome or strategies in digital games. Most of these studies are focused on competitive games, such as RTS or MOBAs. For example, Erickson and Buro [7] proposed a model for predicting the winning player in *StarCraft* using logistic regression from replay data. Stanescu and Čertický [15] proposed a prediction system based on the Answer Set Programming paradigm to predict the number and type of units a StarCraft or WarCraft III player trains in a given amount of time. Rioult et al. [13] proposed an approach to predict wins and losses based on topological information using player locations. Synnaeve and Bessière [17] employed Bayesian model to predict the likely strategies Starcraft players would employ in the beginning of a game. Summerville et al. [16] used machine learning techniques to predict the selection of heroes in *Dota 2*. Below we cite in more details some of the many related works.

Cleghern, Zach, et al. [3] introduced an approach to forecast changes in the hero's health in the MOBA game *Dota 2* by observing past game session data (i.e., replay logs) and splitting the data related to health values into two time series: one for small changes in health and another for large changes. They used this splitting approach to predict both types (small and large) of changes using statistical models. The authors used an auto-regressive moving-average model for small changes and a combination of statistical models for large changes. They combined both methods to create a forecasting system to forecast changes in heath in a game session. However, their approach considers only health data and not other information of the game session (e.g., events). Furthermore, their prediction model is constrained by the match duration, lacks contextual information, and is only focused on MOBAs.

Yang et al. [19] proposed a data-driven approach to discover combat strategies of winning teams in MOBA games. The authors modelled each combat as a sequence of graphs, where each vertex represent a player associated to a role in the game, resulting in 10 vertices (five for each team). They also created another special vertex to represent the death state. The edges in the graph represent the interactions between players, which can be either doing damage on the adversary or healing a partner, or the death of a player (which connects to the death vertex). The authors use these combat graphs to train a decision tree based on the best features using five graph metrics (in-degree, out-degree, closeness, betweenness, and eigenvector centrality). This tree is then used to mine patterns that are predictive for winning the game. However, their approach generates

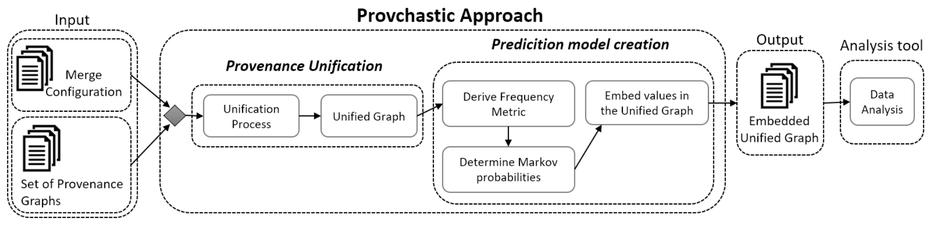


Fig. 1. The *Provchastic* approach overview.

generic high-level rules that does not consider important contextual information such as heroes, level differences, equipment, abilities used, and positioning. As such, it cannot reveal the dynamics of each combat, only high level factors that tends to determine the outcome of the game.

Dereszynski et al. [4] presented a probabilistic framework for RTS games based on hidden Markov models. Their approach is focused on predicting the base building order by observing the timing of the current state to predict future states based on probabilistic inference. The authors limit the states to the number of units and buildings. Each state is measured at every 30 seconds of the game. Their strategy, which is similar to our own, allows to capture the likelihood of different choices and the probability of producing particular units in each state based on the observed events. However, their model is not sufficient to capture other aspects of the game *StarCraft*, such as tactical decisions, and is limited to only the first seven minutes of the game because players tend to execute their first minutes in isolation and not reactive to the other player's tactics and unit composition.

3 Provchastic

Our proposed approach, named **Provchastic** (*Prov*enance for generating sto*chastic* models), is a probabilistic approach for game analytics that takes advantage of a recent approach for tracking and storing game provenance data developed by Kohwalter et al. [10]. Our stochastic model was inspired by the work of Lins et al. [11], where they proposed the *Timed Word Tree* visualization, which is a variation of *Word Tree* displays [18].

The idea is to explore the structured nature of provenance graphs to generate a stochastic model using the commonly known Markov Chains [8]. The stochastic model is derived from a set of provenance graphs from captured game sessions, resulting in a unified provenance graph that contains the probabilities to change states. Our proposed *Provchastic* approach is divided in two major phases: (1) **provenance unification** and (2) **stochastic model creation**. Figure 1 gives an overview of these phases and how they are related, which we describe in more details in the following sections.

3.1 Provenance unification

The first major phase is responsible for the creation of the unified graph, which is used to generate the stochastic graph. The process of creating a unified graph requires four procedures, as illustrated by Figure 2: (1) a **matching heuristic** to match vertices from different graphs, (2) the definition of **vertex similarity**, (3) **vertex merge**, and (4) a **graph merge**. The merge process occurs by merging two graphs at a time and, consequently, the matching heuristic uses only two graphs at a time as well.

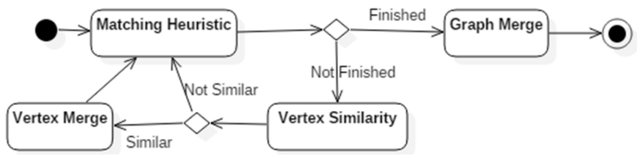


Fig. 2. Provenance unification process.

A *matching heuristic* for vertex selection is used to restrict the search space for vertex matching and avoid making a Cartesian product between vertices from both graphs. One solution for graph matching is graph isomorphism. However, this is a NP-Complete problem and thus we look for heuristics to make the problem resolvable in a reasonable time. Furthermore, the heuristic decides how the comparison between vertices from different graphs is made. It always chooses two vertices (one from each graph) to pass to the Vertex Similarity algorithm for comparison.

Currently, we use a heuristic that employs temporal information to define the order to compare vertices from provenance graphs generated from the same game (e.g., different players playing the game or multiple game sessions from the same player). This matching heuristic takes advantage of the chronological nature of provenance graphs to prune the search space. Once a corresponding vertex from the second graph is found for the matching, then both vertices that were matched are never again revisited. Figure 3 provides an overview of the matching process used by our Matching Heuristic.

The *Vertex Similarity algorithm*, also known as the *distance metric function*, always compares two vertices (e.g., v_x^1 and v_y^2) for the (similarity) evaluation to establish the similarity between them. The similarity value between two vertices ranges from 0 to 1, where 0 represents total mismatch (0%) and 1 represents a total match (100%). Consider $G^1 = (V^1, E^1)$ as a directed graph where $V^1 = v_1^1, v_2^1, \ldots, v_n^1$ and consider $G^2 = (V^2, E^2)$ as a directed graph where $V^2 = v_1^2, v_2^2, \ldots, v_m^2$. The comparison algorithm always compares vertices v_x^1 and v_y^2 at a time, where $v_x^1 \in V^1$ and $v_y^2 \in V^2$ and $V^1 \neq V^2$.

This *Vertex Similarity* process has three steps: (1) **vertex type verification**, (2) **attribute evaluation**, and (3) **similarity evaluation**. Furthermore,

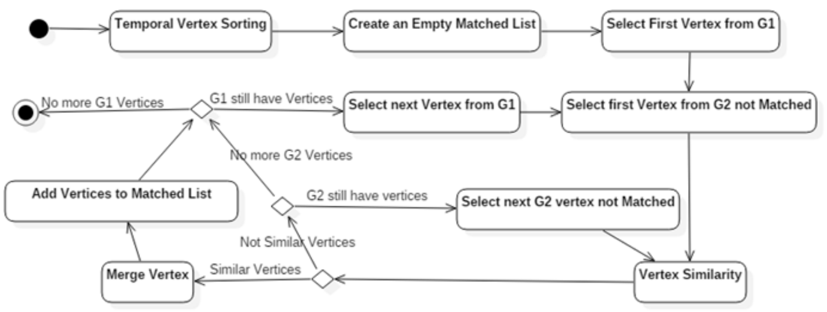

Temporal Vertex Sorting
Create an Empty Matched List
Select First Vertex from G1
No more G1 Vertices
G1 still have Vertices
Select next Vertex from G1
Select first Vertex from G2 not Matched
No more G2 Vertices
Add Vertices to Matched List
G2 still have vertices
Select next G2 vertex not Matched
Not Similar Vertices
Merge Vertex
Similar Vertices
Vertex Similarity

Fig. 3. Matching Heuristic process.

the comparison algorithm uses user-defined parameters for the merge process that are associated with each attribute. The user needs to inform the value that represents the acceptable *error margin* for each specific attribute and the weight of that attribute for the similarity calculation.

The **first step** from the *Vertex Similarity algorithm* is the **vertex type verification**. This step receives two vertices (v_x^1 and v_y^2) from the matching heuristic and checks the type of v_x^1 and v_y^2 to verify if they match. If they belong to the same vertex type (i.e., Agent, Activity, or Entity) then it proceeds to the second step (**attribute evaluation**), which evaluates the attributes from both vertices. In the case where the types are mismatched, then the vertices are not considered similar and the comparison is halted, setting a similarity factor of 0%, skipping the second step (attribute evaluation), and going directly to the third step (similarity evaluation).

The **second step** of the algorithm, which is the **attribute evaluation**, tries to match each attribute from one vertex (v_x^1) with the same attribute from the other vertex (v_y^2), comparing their values. This comparison also searches the parameters inside the *merge configuration file* to verify the acceptable *error margin* for the attribute when dealing with numeric values. Thus, if the difference between the numeric values is lower than the accepted *error margin*, then the attribute values are considered similar.

The **third step**, which is the **similarity evaluation**, determines if v_x^1 and v_y^2 can be considered similar and thus suitable for combining into a single vertex in the unified graph. The *similarity factor*, which is used for the evaluation, is calculated from the number of attributes that were considered similar in both vertices during the second step. The *similarity factor* is then compared with the *similarity threshold*. If the *similarity factor* is below the accepted *similarity threshold*, then v_x^1 and v_y^2 are not considered similar. However, if the *similarity factor* is equal or greater than to the *similarity threshold*, then both vertices can be considered similar vertices. Note that these two vertices are the ones received from a *matching heuristic*. If two vertices are similar, then the algorithm merge them into a single vertex with all attributes from both vertices and their respective values plus a new attribute named *GraphFile* that shows the origins of this new merged vertex, which is the name of the graphs that were used during the merge process.

The *Graph Merge* process is the last process to create a unified graph. This occurs only after the **matching heuristic** finishes matching vertices from both graphs. All the resulting merged vertices and vertices that were not matched are added in the unified graph. Then, after adding all vertices in the unified graph, we add all edges.

Finally, an important step is the **distance metric configuration**. This step uses the *merge configuration file*, which contains parameters that are used during the merge process, such as *error margin*, *similarity threshold*, and attribute weights for computing the overall similarity of the vertex.

3.2 Stochastic model creation

The proposed stochastic approach is based on the stochastic model of Markov Chains, which describes a sequence of possible states with the probability of occurring being dependable of the previous state. We use the unified provenance graph, which was described in the previous section, to derive statistical information about the states that occurred in all of the provenance data and to calculate the probability of jumping from one state to a neighboring state. The generation of our Markov stochastic model is composed of two steps.

The **first** step is to calculate the *frequency* metric for each vertex and edge of the unified graph. We define the *frequency* metric as the number of graphs inside the *GraphFile* attribute divided by the number of total graphs used in all merges that composed the unified graph. This *frequency* metric is then used to determine the *Markov* probability for an state to happen based on the previous state.

The **second** step is to calculate the *Markov* probability for each edge based on the *frequency* metric. We calculate two types of *Markov* probabilities for each edge: one for navigating to the **future state** (*future probability*) and another when navigating to the **past state** (*past probability*). The probability of reaching a future state is used for predicting the next state. The past probability is useful for determining common states that happened before the current state, allowing the analyst to determine the path with the highest probability to reach a desired state, understanding how this state was reached in the game.

Its important to remember that the edge orientation in provenance graphs always points from the present to the past, by definition. Thus, if we want to navigate in the graph to predict *future states*, then we need to traverse the graph in the direction of the source of the incoming edge. If we want to analyse the *past states*, then we need to traverse the graph in the direction of the target of the outgoing edge. Therefore, if we want to determine the probability of the next state, we need to: (1) Look at how many incoming edges the vertex has; (2) Check each incoming edge *frequency*; (3) Divide each edge's *frequency* by the vertex *frequency*.

It is also important to remember that the sum of all incoming edge's *frequency* will always be equal to the vertex's *frequency*. If there is only one incoming edge,

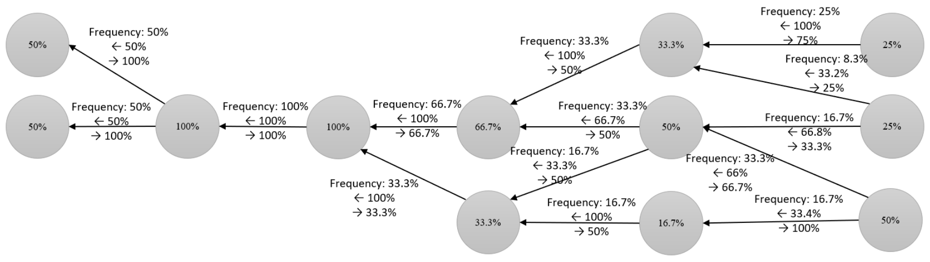


Fig. 4. Abstract example of a provenance graph embedded with *Markov probabilities* for *short predictions*. The vertex frequency is represented inside the vertex.

then its *frequency* is equal to the vertex's *frequency*. If there are more than one incoming edge, then their frequency acts as a weight value when distributing the probabilities for taking each path.

Calculating the probability to go to a previous state is analogous. The difference is using the outgoing edges instead of incoming edges. The procedure to calculate those *Markov probabilities* for each path can be summarized as follows:

```
for each edge in graph do
    sourceVertex ← edge.source
    targetVertex ← edge.target
    edge.markovFutureProbability ← edge.frequency / targetVertex.frequency
    edge.markovPastProbability ← edge.frequency / sourceVertex.frequency
```

This procedure embeds the *Markov probability* information to navigate both ways in the provenance graph in each edge of the graph. Thus, given a game state, we know the probability to transit to another neighboring state by checking the edge that connects the given state. Figure 4 illustrates an example of an abstract provenance graph with the *Markov probabilities* for *short predictions*, considering both ways. Navigation probability to the *future* is represented by right arrow (→) and for the *past* is represented by left arrow (←). This Markov embedding is related only to *short predictions*, which is the immediate vicinity of the current state. Calculating *long predictions* (multiple states ahead) is achieved by simply determining a path in the graph that connects the current state with the desired state and multiplying all the *Markov probabilities* of the edges that composes that path. Figure 5 illustrates an example of the same abstract provenance graph with the *Markov probabilities* for *long predictions* originating from the vertex marked as source.

Another information resulted from the *long predictions* calculations is the prediction of a particular outcome or event happening at least once in the near future. The event type information is embedded in the provenance vertex and is used in parallel when calculating the probability to reach a specific vertex. For example, Figure 6 illustrates another (different) graph with embedded *long predictions* showing the probability to reach at least one of each of the existing different events. This type of prediction is achieved by calculating the probability

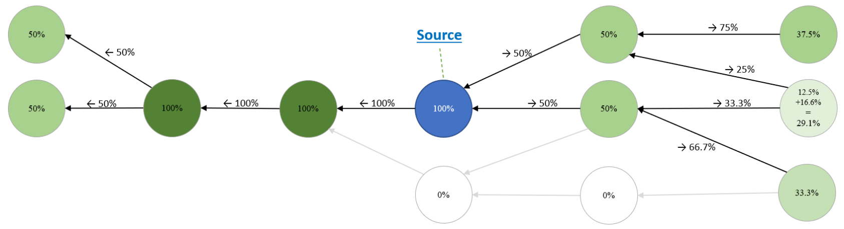
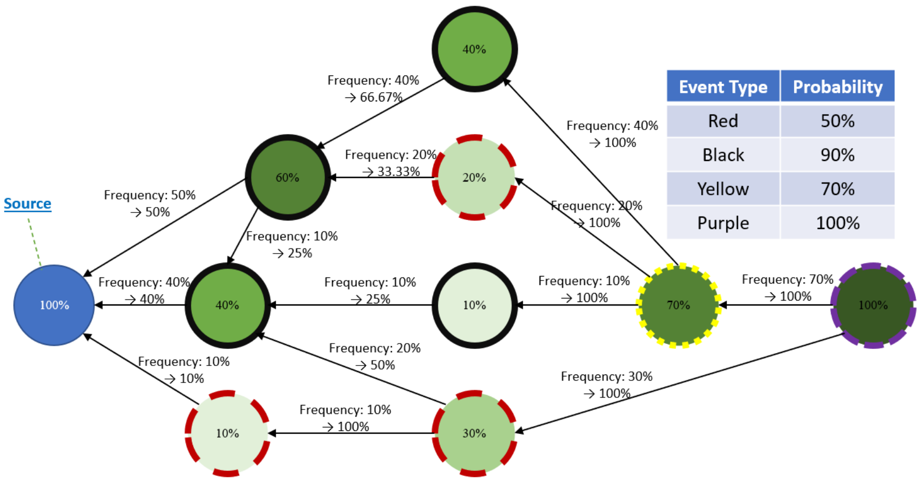


Fig. 5. Same provenance graph from Figure 5 embedded with *Markov probabilities* for long predictions originated from the vertex marked as *source*. The interpretation for the vertices at the right side of the *source* (i.e., the future) refers to the probability of reaching each vertex. On the left side of *source* (i.e., the past), the interpretation is related to common states that led to the *source*.



Event Type	Probability
Red	50%
Black	90%
Yellow	70%
Purple	100%

Fig. 6. Example for predicting possible outcomes for the *source* vertex. Outcome types are distinguished by the vertex border color: Red, black, yellow, and purple.

to reach the first event of the desired type for each existing path that leads to that event type and then sum these probabilities of each path for each event type.

3.3 Implementation

We implemented an open source module that allows for merging multiple provenance graphs to generate the unified graph. Our module also computes the *Markov probabilities* and embeds them in the unified graph as described in the previous section. The resulting unified graph is compatible with the open source provenance graph visualization tool *Prov Viewer*[2], allowing the user to visually analyze and explore our stochastic graph.

Our module also generates *short predictions* when building the unified graph. *Long predictions* requires the desired source vertex as an input, which is used to compute the probabilities to reach all other vertices present in the graph and to generate a new unified graph with this information embedded in each vertex. This method allows us to load the generated graph into *Prov Viewer* in order to generate a visual representation of the predictions.

4 Case Study

In this section, we validate our proposed approach for provenance graphs through a case study analysis. The following research question guided our study:
RQ:*Does the use of provenance obtained from multiple game sessions support predictions and understanding of events for future game sessions?*

Our case study is based on the adaptation of the Car Tutorial from Unity asset store [3]. This prototype has only one racetrack and focuses on the arcade style racing game. In addition, there are no opponent cars, only the player's car to simulate a practice run. The prototype gathers provenance data related to key events and actions executed during the game session, along with their effects on other events, to compose the provenance graph (e.g., crashing the car, pressing the car's brake, losing car control, accelerating, coasting, etc.).

We generated a provenance stochastic model for this study using our approach from 75 provenance graphs of the game that were captured from 75 gaming sessions using PinGU [10]. We used a *similarity threshold* of 95% and *error margins* for each attribute of approximately 20% (some attributes have slight different *error margins* due to the observed domain values). Thus, considering the total number of attributes (eight), a vertex is only considered similar to another vertex if all their attributes are considered similar.

This unified graph is composed of 2,302 vertices and 8,201 edges extracted from all game sessions, where each graph represents one complete lap in the track. In contrast, the sum of all vertices and edges from the original 75 graphs is 7,840 and 16,494, respectively. Thus, the unified graph had a 70.63% reduction in the number of vertices and 50.27% reduction for the edges. We then used this generated unified provenance graph for the analysis described in the following paragraphs.

In Figure 7 we have a different example that shows two different states that are almost in the same coordinates in the track, but with a difference: the source's speed of the left graph is around 35km/h (*Slower Vertex*), while the right graph is only at 200km/h (*Faster Vertex*). We can see that this small difference causes different possible outcomes, as show in Table 1. Thus, we can observe that, at this point of the track, being slower increase the probability to crash the car by

[2] https://github.com/gems-uff/prov-viewer
[3] https://assetstore.unity.com/packages/templates/tutorials/car-tutorial-unity-3-x-only-10

13% due to miss-calculating the necessary turn from the lack of speed, resulting in tighter turn and crashing at the side-way of the road, as illustrated at the zoomed section of the figure marked with a red circle. In contrast, going faster increases the chances of the car losing contact with the ground by 9% due to a slight decline in that section of the track.

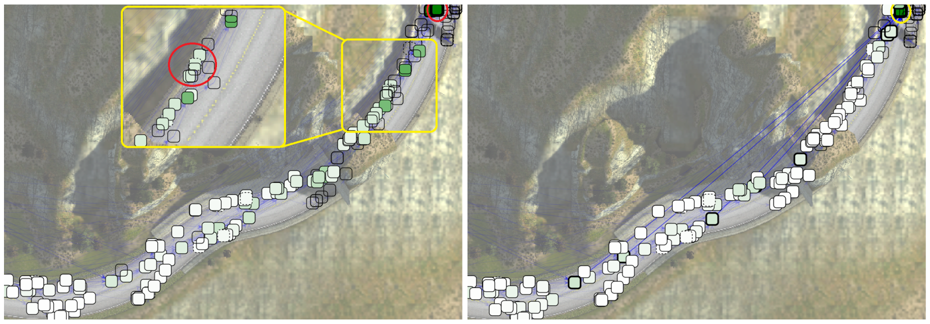


Fig. 7. Contrasting the different probabilities from two states with similar coordinates in the game. Vertex color is based on the probability of reaching it.

Figure 8 shows an example of different possibilities for outcome due to small differences in the game state. The left graph's source vertex is in the process of decelerating (*Decelerating Vertex*), while the right graph is maintaining speed (*Accelerating Vertex*). Table 1 shows the differences in probabilities, which resulted in a 20% chance to crash while decelerating vs 3% while accelerating at that moment due to increased chances to lose car control to instability while braking. This can be observed in the figure by analysing the differences in the marked region inside the yellow rectangle.

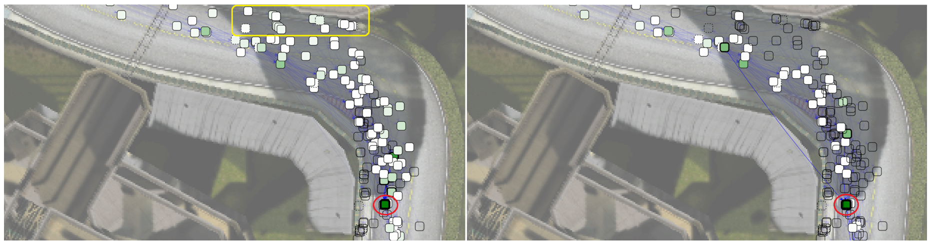


Fig. 8. Contrasting the different probabilities from two states with similar coordinates in the game.

Figure 9 shows an example of our stochastic model being used to understand the common paths that led to a particular state, or how to reach it. The vertex

Table 1. Contrasting long term predictions

Figure 7: Event Probability	Faster Vertex	Slower Vertex
Lost Contact w/ Ground	86.2%	78.5%
Crash	33.4%	46.3%
Lost Control	44.7%	35.7%

Figure 8: Event Probability	Decelerating Vertex	Accelerating Vertex
Lost Contact w/ Ground	1.2%	17.5%
Crash	20.1%	3.3%
Lost Control	9.4%	3.5%
Scrapped	17.1%	11.8%

circled in red is the source vertex and represents a crash. Looking at that state's past in our stochastic graph we can see the most common pathways that lead to that outcome. The yellow and blue circles denotes the region with past events with the most probable cause due to the vertex greenish coloration. The red path denotes the general path taken that led to that crash. Looking at these vertices, and comparing to the others nearby, we could see two things that influenced the crash: the high speed, which in turns decreases the maximum turn rate of the car, and their positioning that, when maintaining a high speed does not allow to make that sharp curve without crashing at the side-rails.

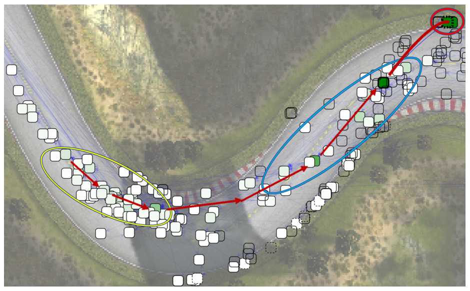


Fig. 9. Analysing the most probable causes of a crash.

Our stochastic model, in the form of unified provenance graph, allows to determine possible outcomes and probable causes of a particular game state. Thus, answering our research question:

RQ: Does the use of provenance obtained from multiple game sessions support predictions and understanding of events for future game sessions?
Answer: Provenance graphs indeed can be used to create stochastic models based on Markov chain, for example, to predict short and long-term out-

comes by navigating the graph in the future and to understand how to reach a specific outcome by navigating to the past.

5 Conclusion

In this paper we presented **Provchastic**, a novel approach for game analytics that creates stochastic models using provenance data from previous game sessions. Our approach is the first work to unify multiple game provenance data from multiple game sessions for multi-session analysis and to create a stochastic provenance graph that determines the sequence of probable events using the commonly known stochastic model of Markov chains. This stochastic model allows the analyst to find out common outcomes for different game states, how they were reached, and to explore multiple game session provenance data at the same time.

Provchastic is compatible with the existing provenance capture framework *PinGU* and the open source visualization tool named *Prov Viewer*. We demonstrated its usage in conjunction with Prov Viewer by generating the stochastic model from 75 game sessions that tracked provenance data. We could observe that it is possible to create stochastic models using provenance graphs for short and long predictions and to understand probable causes for certain events. A limitation of the approach is related to unseen traces. Each query find a similar state in the existing stochastic model through the similarity algorithm. This can degrade the results if the closest state that was matched is too different from the current state. Moreover, the predictions are only available for previously known traces, which is a limitation on learned systems.

Future works includes finding good patterns in the graphs that reached desirable outcomes in order to improve the chances of reaching the same goal in future iterations. Similarly, another approach could be related to detecting bad patterns that should be avoided and compare current player performance with previous executions and point out the decisions that improved or degraded her overall performance in the game session. Another future work is to create a real-time prediction "helper" to aid the player in the decision making process by using projections of the outcome for each of her decisions. Finally, our approach depends on the definition the similarity thresholds. A possible future work consists on discretizing the game scene in multiple regions to run **Provchastic** in a less fine-grained graph.This might result in more precise predictions since the evaluation will be in a more coarse grain while also improving visual legibility due to having less vertices in the stochastic graph.

Acknowledgment

The authors would like to thank CAPES, CNPq, and Faperj for the financial support.

References

1. Burelli, P.: Predicting customer lifetime value in free-to-play games. Data Analytics Applications in Gaming and Entertainment p. 79 (2019)
2. Clarke, B.S., Clarke, J.L.: Predictive Statistics: Analysis and Inference Beyond Models, vol. 46. Cambridge University Press (2018)
3. Cleghern, Z., Lahiri, S., Özaltin, O., Roberts, D.L.: Predicting future states in dota 2 using value-split models of time series attribute data. In: Proceedings of the 12th International Conference on the Foundations of Digital Games. p. 5. ACM (2017)
4. Dereszynski, E., Hostetler, J., Fern, A., Dietterich, T., Hoang, T.T., Udarbe, M.: Learning probabilistic behavior models in real-time strategy games. In: Seventh Artificial Intelligence and Interactive Digital Entertainment Conference (2011)
5. Drachen, A., El-Nasr, M.S., Canossa, A.: Game Analytics: Maximizing the Value of Player Data. Springer (2013)
6. Eckerson, W.W.: Predictive analytics. Extending the Value of Your Data Warehousing Investment. TDWI Best Practices Report **1**, 1–36 (2007)
7. Erickson, G.K.S., Buro, M.: Global state evaluation in starcraft. In: Proceedings of the Tenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2014, October 3-7, 2014, North Carolina State University, Raleigh, NC, USA (2014), http://www.aaai.org/ocs/index.php/AIIDE/AIIDE14/paper/view/8996
8. Kemeny, J.G., Snell, J.L.: Markov Chains. Springer-Verlag, New York (1976)
9. Kohwalter, T., Clua, E., Murta, L.: Provenance in games. In: Braz. Symp. Games Digit. Entertain. SBGAMES. pp. 162–171 (2012)
10. Kohwalter, T.C., Murta, L.G.P., Clua, E.W.G.: Capturing game telemetry with provenance. In: 2017 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames). pp. 66–75. IEEE (2017)
11. Lins, L., Heilbrun, M., Freire, J., Silva, C.: Viscaretrails: Visualizing trails in the electronic health record with timed word trees, a pancreas cancer use case. In: Workshop on Visual Analytics in Healthcare (VAHC) (2011)
12. Ravari, Y.N., Bakkes, S., Spronck, P.: Starcraft winner prediction. In: Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference (2016)
13. Rioult, F., Métivier, J.P., Helleu, B., Scelles, N., Durand, C.: Mining tracks of competitive video games. AASRI procedia **8**, 82–87 (2014)
14. Schubert, M., Drachen, A., Mahlmann, T.: Esports analytics through encounter detection. In: Proceedings of the MIT Sloan Sports Analytics Conference. vol. 1, p. 2016 (2016)
15. Stanescu, M., Čertickỳ, M.: Predicting opponent's production in real-time strategy games with answer set programming. IEEE Transactions on Computational Intelligence and AI in Games **8**(1), 89–94 (2014)
16. Summerville, A., Cook, M., Steenhuisen, B.: Draft-analysis of the ancients: predicting draft picks in dota 2 using machine learning. In: Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference (2016)
17. Synnaeve, G., Bessiere, P.: A bayesian model for opening prediction in rts games with application to starcraft. In: 2011 IEEE Conference on Computational Intelligence and Games (CIG'11). pp. 281–288. IEEE (2011)
18. Wattenberg, M., Viégas, F.B.: The word tree, an interactive visual concordance. IEEE transactions on visualization and computer graphics **14**(6), 1221–1228 (2008)
19. Yang, P., Harrison, B.E., Roberts, D.L.: Identifying patterns in combat that are predictive of success in moba games. In: FDG (2014)