Future Generation Computer Systems 84 (2018) 108–122

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Filtering irrelevant sequential data out of game session telemetry though similarity collapses

Troy Kohwalter *, Leonardo Murta, Esteban Clua

Instituto de Computação, Universidade Federal Fluminense, Niteroi – RJ, Brazil

HIGHLIGHTS

- Game techniques uses density and hierarchy clustering, ignoring temporal data.
- New algorithms to collapse sequential information through similarity.
- Algorithms preserve temporal sequence of the data.
- Quickly identify relevant information or state transitions.
- Two experimental studies: one automatic and another with human judges.

ARTICLE INFO

Article history:
Received 22 September 2017
Received in revised form 20 December 2017
Accepted 3 March 2018
Available online 10 March 2018

Keywords:
Game analysis
Provenance
Graph
Game flux
Clustering
Collapse

ABSTRACT

A multitude of game sessions is started every day, generating a huge amount of data that may be useful in many different situations. For this reason, game telemetry is an important trend and feature in modern games, especially for tuning games, quality assurance, testing, and finding game aspects that need to be calibrated. The wealth of tracked information is fundamental for analysis and understanding of events, mistakes, and fluxes of a concrete game session. However, due to game dynamics, the resulting telemetry data may be overwhelming in size, making it difficult to identify sections of interest in the game session. The existing telemetry approaches normally use density clustering techniques for visual analysis, losing the temporal relationship in the process. In order to solve this problem, in this paper we propose three different similarity collapse algorithms based on the classic DBSCAN algorithm, collapsing sequential information of the graph that has similar values or represents the same states. These algorithms allow the game designer to quickly identify relevant information or state transitions without compromising the temporal sequence of events. We implemented this solution over tracked provenance data, which is graph-based, and provide two experimental studies of these algorithms using automatic experimentation and human judges to evaluate each of the proposed algorithms. These experiments show that one of the proposed algorithms is superior to DBSCAN when applied to graphs by better preserving the data semantics when collapsing provenance data. We believe that this kind of approaches will become a trend in the future process of game development.

1. Introduction

Methods for automatic telemetry of game session data have become an important component of game design and production in the last few years [1]. The collected data can be used for different purposes, such as behavior understanding and analysis [2], detecting bugs [3,4], balancing the game experience [5], classifying users [6], understanding common behaviors [7], and even improving the monetization process [1]. In this sense, using computing techniques within these collected data may increase not only the game value but also the ARM (Acquisition, Retention, and Monetization), a valuable aspect of the nowadays digital entertainment business model.

However, depending on the usage context, the amount of tracked game data can reach huge sizes that affect negatively the capability of analyzing game data. Furthermore, telemetry data analysis often requires processing the collected game session data and generating a visual representation of the data in order to be used for analysis. Depending on the game style, a single game session might take several hours or days to be completed. This makes the quantity of displayed data overwhelming, making it difficult for the developer to analyze the game's data due to the large volume of tracked information.

* Corresponding author.
E-mail address: tkohwalter@ic.uff.br (T. Kohwalter).

https://doi.org/10.1016/j.future.2018.03.004
0167-739X/

The existing approaches for game telemetry use information clustering as a reducing technique to deal with visualizations of large quantities of tracked game data. Common clustering techniques involve density clustering [8], which create clusters based on the information distribution in a region, and hierarchy clustering [9], which uses the notion of proximity through a distance metric and a distance function to create clusters. However, both methods consider spatial information for clustering, ignoring temporal relationships when summarizing data. Thus, the sequence of events is lost in the process, favoring the perspective of information distribution in the game scene.

Therefore, in this paper we propose three collapse algorithms based on DBSCAN [10], a popular density clustering algorithm, to summarize tracked telemetry data that considers the sequence of events instead of their spatial information. This helps to manage the volume of data and let the users focus on events that may be more important than others. These collapses reduce the volume of displayed information by hiding part of the data that were not significant enough to produce a meaningful impact on the game and did not offer any useful information for analysis. As the proposed collapses preserve the sequence of events and consider temporal information instead of using spatial information to group nearby vertices, designers can still track the player's progress in the game and easily identify noteworthy regions that had a meaningful impact in the game. While our approach focuses on game analysis, we believe that our solution may be useful to many other systems that collect big data and relationships among them.

The overall goal of this research is to answer the following research question in relation to the three proposed algorithms and the existing DBSCAN algorithm: *RQ: Which type of similarity summarization is an effective method for reducing the information to be analyzed?*

We implemented our approach in the open-source tool Prov Viewer [11], which displays game telemetry data collected with the PinG [12] framework. Prov Viewer shows the collected telemetry data as a provenance graph [13], where vertices represent the events and edges represent the chronological order in which these events were executed. We evaluated our algorithms within two experiments: (1) an automatic experiment to obtain quantitative results and (2) a manual experiment involving human judges to obtain qualitative results. The rest of the paper is organized as follows: Second section presents a brief overview of the provenance concept and an overview of the Provenance in Games concept. The third section presents our approaches for hiding irrelevant information through collapses and the evaluation of our approaches are presented in the fourth section. The fifth section presents the related work in the area of game session analysis, along with their clustering techniques. Finally, the last section concludes this work and points out future works.

2. Provenance

Provenance is well understood in the context of art or 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 [14]. At the *International Provenance and Annotation Workshop* (IPAW) [15], the participants were interested in the issues of data provenance, documentation, derivation, and annotation. As a result, the *Open Provenance Model* (OPM) [16] was created and later improved during the *Provenance Challenge* [17], which is a collocated event of IPAW. More recently, another provenance model was developed, named PROV [13], which can be viewed as the successor of OPM and tries to deal with some of the OPM's shortcomings. Both models intend to apply provenance concepts to digital data.

Both provenance models assume that provenance of objects is represented by an annotated causality graph, which is a directed acyclic graph enriched with annotations. These annotations capture further information related to its execution. According to Moreau et al. [16], a provenance graph is a record of a past or current execution, and not a description of something that could happen in the future. The provenance graph captures causal dependencies between elements and can be summarized by means of transitive rules. Because of this, sets of completion rules and inferences can be used in the graph in order to summarize the information.

2.1. Provenance in games

A typical digital game architecture is mainly composed of game objects and the game loop. All objects present in a game, from environment objects to characters, are inherently defined as game objects. Game objects by themselves do not add characteristics to the game. Instead, they are containers that hold components that implement actual functionality, such as scripts (i.e., artificial intelligence, player controller, etc.), meshes (the object structure or "body"), physics, textures, animations, and audio. Meanwhile, the game loop is responsible for the sequence of events that occur in a game, allowing the game to keep running regardless of the user's input. The game loop keeps the game alive, updating its scene graph states and executing their actions and behaviors. Each script in a game object has a function *update*, which is called by the game loop in order to execute the specific game object functionalities. Every time the game loop is ticked, it executes the *update* function of the scripts that belong to the game objects present in the scene.

In a simulation or serious game, some facts might not be clear or transparent enough for the player to understand why something went wrong. While in a traditional game this can be solved with a new game session, in a serious games or simulation it is important to give the opportunity to the player to find what caused this situation in an analytical way. Thus, in a previous work [12], we proposed a novel usage for provenance in the game field. In order to adopt provenance for the context of games, we mapped each type of vertices of a provenance graph into elements typically found in games.

The PROV provenance model assumes that provenance of objects is represented by an annotated causality graph, which is a directed acyclic graph enriched with annotations. These annotations capture further information belonging to the system execution. Using the PROV notations, an *entity* was mapped to static game objects present in a game, such as weapons, equipment, and furniture. *Agents* were mapped to dynamic game objects, such as characters, event controllers, and plot triggers. Lastly, *activities* were mapped to actions or events executed throughout the game, such as interactions with other *agents* and *entities*. The causal relations, which are the edges of a provenance graph, were mapped to influences occurred during the game. Fig. 1 illustrates this mapping of provenance concepts into the game context, outlining important information of each element type to be collected during game execution for provenance analysis.

The information collected during the game session is used for the generation of the provenance graph. Thus, all relevant data should be registered, preferentially at a fine grain. The way of measuring relevance varies from game to game, but ideally, it is any information deemed relevant by the game designer that can be used to aid the analysis process. However, this fine grain data gathering results in an exponential increase of the graph, containing multiple events that can act as noise to certain types of analysis and could be hidden or summarized.

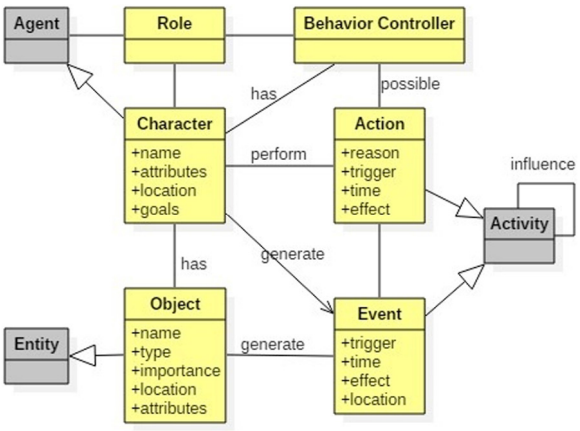

Fig. 1. Mapping of provenance and game domains. Gray classes belong to the provenance domain. Yellow classes belong to the game domain.

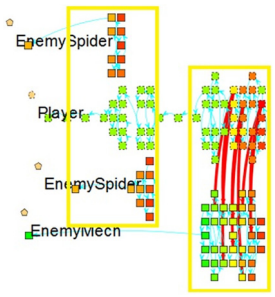

Fig. 2. Two combat examples, each marked by a yellow box. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

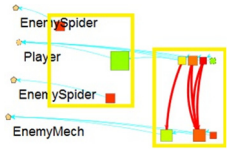

Fig. 3. Graph from Fig. 2 after the similarity collapse.

3. Similarity collapse

As previously discussed, game sessions may generate large quantities of data, making it difficult for designers to visually explore their telemetry results. However, as multiple consecutive telemetry data actually represent subtle variations of the game state, they can be collapsed, keeping only the most relevant data. These remaining telemetry data actually represent relevant variations in the game state according to specified game attribute.

Our approach intends to reduce the volume of displayed information by summarizing the tracked data. In order to accomplish the summarization and to respect the sequence of events, the raw data must be structured as a graph, with vertices representing the events and edges representing the chronological order in which these events were executed.

Fig. 2 illustrates an example where the player experienced two distinct battles against different enemies, with each battle being represented by a yellow box. The blue edges represent the chronological order of events, while the red edges represent moments when an enemy hit the player. Vertex positioning also represents the timeframe (from left to right) and vertices within the same column mean that they occurred at the same time slice (*e.g.*, within the same one second period). Each vertex in the graph represents the execution of an action. The vertex color is proportional to health value, which ranges from green (high health) to red (low health). In the first battle, there are zero red edges connecting the player's vertices. Therefore, the player managed to dispatch the enemies without taking any hit. However, in the second battle, the player struggled to overcome his enemy and lost a large amount of health, which is reflected by the vertices colors.

If we are interested in analyzing the player's challenges, then we should focus on sections where the player struggled to overcome or had significant changes in the game state. As such, all instances that the player's hit point did not change or barely fluctuated are not relevant to the analysis, which happens to be the case of the player's first battle. Therefore, we can omit all vertices in the graph that have similar values with their neighbors by doing a collapse based on similarity. In the example from Fig. 2, it would mean collapsing the first fight entirely and grouping some of the vertices from the second fight that shares the same color, resulting in a graph similar to the one illustrated by Fig. 3.

In order to achieve the graph from Fig. 3, it is necessary to compare each vertex with its neighbor, which is connected by an edge, to omit similar states. If the vertices' values are similar for the specific game attribute being analyzed, they can be collapsed into a single vertex. Since the objective is to omit all similar states, it is necessary to go beyond the vertex first neighbor. If any of the vertices in the *collapse group* have an edge to a vertex outside of the *collapse group* and that vertex has a similar attribute value to those in the group, then it is added to the *collapse group*. Thus, the *collapse group* will keep growing until a significant change of state is detected.

We initially considered using Dimension Reduction algorithms [18,19] to achieve this type of information reduction. However, Dimension Reduction works by removing the data and can lead to information loss if not handled with care, which results in a loss of flexibility when manipulating and examining the game data during visual exploratory analyses. Thereby, we decided to use density clustering algorithms to achieve this information reduction while at the same time maintaining the flexibility to manipulate and handle the data since the information is only clustered and not removed. Thus, the analyst can easily expand a cluster to explore the collapsed data. Furthermore, clustering algorithms can be used to create different levels of detail through successive clustering of the game data, enabling different overviews of the tracked data.

Similarly, we also considered using spatio-temporal clustering algorithms as a basis of our heuristics. However, there is no significant difference between density-clustering algorithms to spatio-temporal clustering algorithms since the spatial and temporal variables can be easily inserted in the distance function. This is basically what the ST-DBSCAN [20] and other similar spatio-temporal algorithms do.

We propose the usage of DBSCAN clustering algorithm as a basis to achieve this type of collapse, using the distance function to determine the similarity between vertices. Since provenance graphs are predominantly composed of vertices with two or three neighbors, we had to discard the density parameter in order to use this type of clustering algorithm. In the DBSCAN algorithm, the similarity between values is then defined by the epsilon (ε) parameter, which is used by the distance function (*e.g.*, Euclidian distance). This epsilon must be related to the attribute (or attributes) being used for analysis and be an absolute value. Furthermore, the DBSCAN algorithm decides if neighboring vertices should belong to the same *collapse group* by comparing their distances (*e.g.*, values) with the epsilon. Note that a vertex is only a neighbor of the *collapse group* if it has an edge connecting it to the group. As such, the *collapse group* only expands through the edges that connect the

vertices in the graph and, therefore, the temporal sequence of events is always preserved because a *collapse group* can never be a disconnected sub-graph.

When taking out the density parameter from DBSCAN, the clustering algorithm could face situations where it would collapse the entire graph in a single node when vertices had small and gradual variations in their values (*e.g.*, crescent graphs, monotonic graphs). To mitigate this problem, we propose an adaption for the DBSCAN, named as ***IC* (inter-cluster verification)**, to extend the heuristic that defines the *collapse groups*. Instead of comparing only with the closest neighbor (in the case of graphs, the direct neighbor), the comparison would also be made with the entire group due to small variations that each vertex can have in its value without causing a change of state. Thus, the epsilon must be checked against the group's minimum and maximum values before adding new vertices. If adding new neighbor results in a difference between values greater than the epsilon, then it is marked as a change of state and the vertex is not added to the group, possibly implying in the creation of a new group with this new vertex and the similar ones. This modification limits the cluster growth through a distance restriction, imposing a variance limitation on the range of possible values by always comparing with the farthest member of the cluster instead of only the direct neighbor. Therefore, this heuristic avoids creating a single cluster when the data contains monotonic behavior.

To better formalize this process, consider $G = (V, E)$ as a directed graph where $V = \{v_1, v_2, \ldots, v_n\} \mid v_i, 1 \leq i \leq n$, represents an event related to an element of the game (*e.g.*, player or enemy) and $E = \{e_1, e_2, \ldots, e_m\} \mid e_j = (u, v), 1 \leq j \leq m, u \in V$, and $v \in V$, represents the influence of an event into another, which indirectly indicates the chronological order of events in the game. Furthermore, consider that $A = \{a_1, a_2, \ldots, a_k\} \mid a_l, 1 \leq l \leq k$, represents a numeric attribute used for analysis of each vertex. Moreover, consider S to be a similarity collapse set of connected vertices for a specific attribute and $min\,(S, a)$ and $max\,(S, a)$ as the minimum and maximum values for the attribute $a \in A$ considering all vertices in the *collapse group* S . A new vertex $x \in V$ is added to a *collapse group* S if and only if $abs\,(max\,(S \cup \{x\}, a\,) - min\,(S \cup \{x\}, a)) < \varepsilon$. This change is made only in the distance function inside the neighborhood query, which is responsible for returning all reachable points within the epsilon restriction, according to the *DBSCAN* algorithm. Thus, instead of comparing the distance with only the direct neighbor, the *IC* variant of the *DBSCAN* algorithm considers all the vertices already in the *collapse group*.

For example, consider $G = (V, E)$ where $V = \{v_1, v_2, v_3, v_4, v_5\}$ and $E = \{(v_5, v_4), (v_4, v_3), (v_3, v_2), (v_2, v_1)\}$. All vertices have only one attribute with the same value, but v_3 that has the double of the value of the other vertices. Let us define the epsilon to be one standard deviation of the v_i values. By running the *IC* variant, it would return three disjoint sets: $S_1 = \{v_1, v_2\}$, $S_2 = \{v_3\}$ and $S_3 = \{v_4, v_5\}$. The reasoning for this is: first, it adds v_1 then v_2 to S_1. When trying to add v_3 to S_1, the difference between values is greater than the established threshold. Thus, v_3 is not added to S_1. Since v_2 has no incoming neighbor beside v_3, no other vertex is added to S_1. The same happens when evaluating v_3 against v_4 and thus only v_3 belongs to S_2. When starting with v_4, the algorithm inserts v_4 in S_3 and evaluates v_5. Because the difference between v_5 and v_4 is lesser than the epsilon, v_5 is added to S_3. Since v_5 has no other neighbor, the algorithm ends and returns S_1, S_2, and S_3. By using the proposed modification, nearby vertices in the graph that have similar values are collapsed into a single vertex, reducing the graph's overall size by creating clusters of similar information.

We also propose the ***VE* (Variable Epsilon)** adaptation to be incorporated in the DBSCAN: instead of using a single universal epsilon to define the collapse sets, each set defines its own epsilon and then adapt it as the set grows in size. Thus, each set initially uses the universal epsilon until it reaches a certain size. After reaching the size threshold, each set computes its own epsilon based on the current members of the set. This set epsilon is then used for further insertions in its respective set and each set have their own epsilon based on their members. If a new element is inserted in the set, then that set's epsilon is recalculated based in its members, including the newest one. This allows for each set to only have members that orbit around the cluster's defined epsilon and allow each cluster to have its own defining characteristic that can be completely different from another cluster. This might be useful when the data distribution is more erratic but controlled in some sort, containing sections of very close vertices (e.g., 1.01, 1.0095, 1.015, 1.02) and others that are more distant from each other but still orbit around some value (e.g., 20, 24, 19, 22). However, since the epsilon can change as the set grows, then in a way it would be adapting accordingly to its surroundings, providing a controlled variation of its own epsilon. For the first prototype, we based the epsilon on standards deviations of the values from all members of the collapse set, including the new additions, which will provoke slight variations in the epsilon. However, the bigger the set, the lesser impact the new additions will have in the set's epsilon.

These two new variants described above (*IC* and *VE*) can also be incorporated together in the *DBSCAN*, generating three different variants: (1) inter-cluster verification with a fixed epsilon (IC), (2) no inter-cluster verification with a variable epsilon (VE), and the combination of both, resulting in (3) inter-cluster verification with a variable epsilon (ICVE). The ***ICVE* (Inter-Cluster verification with a Variable Epsilon)** provides, even more, cluster growth control since new members need to be within epsilon distance of the farthest member of the cluster. However, it also allows for each cluster to have its own characteristic, creating a more knit group of likeness. Furthermore, we defined the initial epsilon to be based on the standard deviation of the attribute values from the graph being analyzed. Similarly, the algorithms with variable epsilon also updated their epsilon values based on the standard deviation of the cluster.

3.1. Implementation

We implemented the *DBSCAN* algorithm and all its three variations presented in the previous section in Java using the graph data format from *Prov Viewer* [11]. *Prov Viewer*[1] is a provenance visualization tool that uses the JUNG framework for rendering game telemetry data. *Prov Viewer* uses vertices to represent actions and events that occurred during the game along with the actors (*i.e.*, characters) that executed these actions. The edges of the graph represent the causal relationships between vertices, including the chronological order of events and other influences that occurred.

Prov Viewer can execute the similarity collapse when visualizing the graph by calling one of its collapse algorithms with the desired attribute or a distance metric that involves multiple attributes. The selected collapse algorithm returns the collapse sets to *Prov Viewer*, which in turn updates the graph visualization by executing the collapse for each set, creating a cluster vertex that assembles together all vertices of the set.

4. Evaluation

In this section, we assess the proposed clustering algorithms for provenance graphs and determine which is the most appropriate algorithm to meet our goals. The algorithms try to detect similar sequences and summarize them into a single collapse, shrinking the graph to show only vertices that are semantically different from

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

each other. Each adaptation proposed in this paper accomplish this feat in different ways.

We elaborated two distinct experiments in order to answer our research question (RQ: Which type of similarity summarization is an effective method for reducing the information to be analyzed?): (1) an automated experiment and (2) an experiment with human judges. The experiment with human judges uses graphs from the automatic experiment and real provenance data from a single game session. These two experiments assess the usage of the DBSCAN algorithm and its three variations to summarize provenance data. In both experiments, we are evaluating the effectiveness of all three clustering algorithms in terms of graph reduction and semantic preservation using the DBSCAN algorithm as a baseline. We used a synthetic oracle in the first experiment to automatically evaluate each algorithm through precision, recall, and F-measure metrics. A synthetic oracle is an artificially created oracle for generic graph clustering. This allows us to measure the effectiveness of each algorithm in a broader aspect, using thousands of graphs for the evaluation.

However, due to the synthetic nature of the experiment and the very individual nature of the definition of "what is the best summarization", we also devised a second experiment that uses a smaller set of graphs to evaluate the four algorithms. Unlike the first experiment, which uses a synthetic oracle, the second experiment uses human judges to decide which algorithm is the most appropriate for each situation. However, we need to use a smaller set of graphs due to time constraints and to avoid burnout of the judges. Nonetheless, in both experiments, we evaluated different types of provenance graphs and two different numeric behaviors: (1) random noise and (2) monotonic noise.

In a brief study, we narrowed down the types of graphs that provenance information can be produced based on different game genres into three categories: (1) Directed Acyclic Graphs (DAG), (2) Tree Graphs and (3) Linear Graphs. DAG is the most common form of provenance graph. Role-playing games, action, racing, sports games, and any other game with multiple actors will generate a DAG graph. However, provenance graphs can also result in a tree-graph under special occasions. For example, a provenance graph can be a tree when the information is still being gathered or when analyzing the provenance data from multiple sessions at the same time to know how and where each player experience diverged. Lastly, the provenance graph can be a linear-graph when only one agent/actor is involved and past actions have no influence on the current action, besides the most recent one. Games that might generate a linear-graph are puzzle games. Linear graphs can also be generated when there are multiple agents but they do not interact with other agents. When this occurs, the resulting graph will contain multiple disconnected linear graphs. Lastly, analyzing game data strictly from a single actor might generate a linear graph.

Each graph category was based on a template (a graph of the clustering result) and different graphs were generated through the addition of noises in order to verify if the algorithm would correctly omit these added elements through collapses. These templates were based on already known graphs for each category being evaluated in this experiment and represent the most common graph structures that can be resulted from real tracked provenance data. The template vertex values were randomized, following a uniform distribution with integer values ranging from negative to positive values, thus simplifying mathematical calculations and minimizing numerical precision errors. Moreover, each template vertex also represents a collapse resulting from the ideal clustering and its value could be seen as the median of the cluster. The inserted noise vertex act as a reverse engineering of the collapse, where each inserted noise would represent an element that belonged to the same collapse as the template vertex.

The noise insertion in the graph is made through insertions of new vertices (and edges connecting these vertices) with values between two existing vertices of the template graph. This type of insertion preserves the original graph layout since new vertices will always be between two existing vertices. Considering that each template vertex is used as the oracle for the summarization and represent a *collapse group*, then the added noise is required to have a compatible value to belong to the *collapse group*. Otherwise, the original premise that each template vertex represents a collapse of similar vertices would be invalid.

As mentioned before, we divided the experiment graphs to include two different numeric behaviors for the graphs. For the random noise values, the insertion of new noise vertex in the graph was accomplished by five steps: (1) define n, which is the number vertex to be added as noise; (2) for each noise vertex to be added, randomly pick a template vertex; (3) calculate the minimum distance to the closest template vertex neighbor from the selected template vertex and define this distance as three-sigma; (4) generate the random value for the noise vertex using a Gaussian distribution, where the median is the selected template vertex value and three-sigma is the minimum distance; and (5) create the noise vertex with the generated value and randomly insert it between the selected template vertex and one of its neighbors, inserting new edges accordingly.

We choose the Gaussian distribution because it allows generating values that are considered similar to the template vertex due to the probabilistic nature of the distribution. The minimal distance from the template vertex to another template vertex is the limiter in the Gaussian distribution (or the three-sigma), ensuring that 99.7% of the generated values will orbit around the median (template vertex value) with the maximum distance of three-sigma, according to the 3-sigma rule. Furthermore, the minimal distance also ensures that the initial premise is valid, which has a sequence of similar values. If we use a value higher than the minimum distance, then its value would be closer to another template vertex neighbor than the selected template vertex. This would either result in a splitting point or encapsulate the other template vertex inside the same *collapse group* if there are many more noise vertices with values higher than the minimum distance. We can impose this restriction because we are simulating an expansion of a collapsed group to try to reverse engineer the original members that led to that collapsed (template) vertex. Nonetheless, outliers are possible to be generated, since the 3-sigma rule only guarantees 99.7% of the generated values to be lower than the minimum distance. However, they will be few in numbers and sparse in the graph, which basically describes the meaning of an outlier.

The monotonic noise behavior is slightly different and was accomplished by only four steps: (1) randomly select a template vertex; (2) randomly select an edge that connects the selected template vertex; (3) generate a random value that is between the vertices' values from the edge; and (4) insert the noise vertex with the generated value between the vertices that belongs to the selected edge, replacing the edge with new edges to correctly connect the three vertices. The generated noise values will always follow the original behavior of the two original template vertices: if the second template vertex value is higher than the first, then all noise values between those two template vertices will follow an increasing function, otherwise, they will follow a decreasing function. The resulting graphs have smoother value transitions due to the monotonic nature of the noise value generation.

4.1. Automatic experiment execution

In order to answer the proposed research question, we must analyze the reduction of the graph size after applying the automatic collapse of the graph and verify if the non-collapsed vertices represent events that have had great variations in the state. This analysis is done through the precision, recall, and F-measure metrics to evaluate the effectiveness of automatic collapse by comparing with the synthetic oracle that was used to generate the graphs.

4.1.1. Materials and method

This experiment was executed through the use of synthetically generated provenance graphs as described in the previous section. We evaluated the three different categories of graphs to know which situations the proposed algorithms were more effective, considering that different domains can lead to graphs of different types. With this experiment, we can map which domains are most appropriate for each heuristic. Thus, we calculated the dependent variables (precision, recall, and F-measure) when applying the algorithm in each graph category (Direct Acyclic Graph, tree, and linear). Moreover, we also analyzed the results with two different numeric behaviors for the noise insertion (random and monotonic). This results in six different categories to be analyzed: (1) randomized DAG; (2) randomized tree; (3) randomized linear; (4) monotonic DAG; (5) monotonic tree; and (6) monotonic linear. In this experiment, we compare the collapse suggested by each algorithm with the template graph (*i.e.*, the synthetic oracle, which is an artificially created oracle), which represents the collapse baseline. We measured the dependent variables (precision, recall, F-measure) for each of the six different categories. The precision metric tells us how many of the generated *collapse groups* were correct in relation to the Oracle. The recall metric tells us how many of the correct *collapse groups* were generated by the algorithm. Finally, the F-measure tells us the overall performance of the algorithm based on the compromise of precision and recall metrics.

We defined a cluster to be correct based on the oracle if the proposed *collapse group* had only one oracle (i.e., template) vertex in it. For example, if we used a template graph with ten template vertices to generate the noise graph and the algorithm collapsed all vertices from the noise graph into a single group, then there will be a 0% precision since it had zero correct *collapse groups* out of ten, and its recall would also be 0%.

We ran seven different iterations for each one of the six categories. Each iteration had forty template graphs that each spawned five noise graphs, totaling in 200 noise graphs to be analyzed for each iteration. The size of the noise graph grew by a factor of two with an initial size of ten times the number of vertices in the template graph (*i.e.,* 10×, 20×, 40×, 80×, 160×, 320×, 640× the size of the template graph). All four algorithms analyzed the same graph, in parallel, before generating the next graph of the experiment. This allows us to better measure their effectiveness since all algorithm result was based on the same graph used by the other three algorithms.

The experiment execution plan was divided into three stages: (1) Train the algorithms through parameter tuning in order to find the optimal configurable parameter values for each category, (2) execute the experiment using the resulting parameter values from stage 1, and (3) analyze the results. We trained all algorithms for each category in order to find the best values for their configurable parameters to maximize their F-measure result. Each training session used a smaller sample of randomized graphs. Nevertheless, the training session also used multiple template graphs to spawn different noise graphs during each iteration. The number of iterations was also reduced, removing some of the intermediate iterations for the training session.

During the second stage, we analyzed 200 different noise graphs in each one of the seven iterations, generating a total of 1400 graphs for each category. Thus, for the randomized behavior, we analyzed the results from all four algorithms from 4200 different graphs (*i.e.*, 1400 DAG, 1400 trees, 1400 linear). Analogously, we also analyzed other 4200 different graphs for the partially monotonic behavior, using the same structure of seven iterations, where each iteration had forty different template graphs and five noise graphs spawned from each template graph. Table 1 describes the graph size used during each one of the seven iterations of the experiment for both randomized and partially monotonic behaviors.

Table 1
Graph size for each iteration of the automatic experiment.

		Graph size (vertices)					
		Random noise			Monotonic noise		
		DAG	Tree	Linear	DAG	Tree	Linear
Iteration	1	50	90	100	50	90	100
	2	100	180	200	100	180	200
	3	200	360	400	200	360	400
	4	400	720	800	400	720	800
	5	800	1440	1600	800	1440	1600
	6	1600	2880	3200	1600	2880	3200
	7	3200	5760	6400	3200	5760	6400

The results of each algorithm were compared with the synthetic oracle, which is the template graph used to generate the noise graph that was given to the algorithms, to calculate their precision, recall and F-measure values. We then used F-measure to define the most qualified algorithm for each analyzed situation.

4.1.2. Results and discussion

First, we ran a normality test to verify if the data followed a normal distribution. According to the results of the Shapiro–Wilk test [21], the normality assumption was violated for all obtained results from the experiment since each dataset had a p-value $< 2.2 \times 10^{-16}$. Therefore, non-parametric tests were adopted for statistical analysis. The non-parametric test used to compare the means was Wilcoxon Matched-Pairs Signed-Rank test. Although there are other non-parametric tests, such as Chi-2 and Kruskal–Wallis, Wilcoxon Matched-Pairs Signed-Rank was chosen because it compares two means from two different samples against the same alternative hypothesis over the same (paired) observation (i.e., graph), which fits our experiment design.

Considering we have four different algorithms to compare and we decided to use Wilcoxon test, it is necessary to run six analyses: (1) DBSCAN vs. IC; (2) DBSCAN vs. VE; (3) DBSCAN vs. ICVE; (4) IC vs. VE; (5) IC vs. ICVE; and (6) VE vs. ICVE. We decided to use the Bonferroni correction in the alpha-value to compensate, which translates to $\alpha = 0.00833$, since we have six comparisons. We adopted the following format for the hypothesis in our tests, naming *alg1* as the first algorithm used in the comparison and *alg2* the second algorithm used in the comparison:

$H_0 : \mu_{alg1} = \mu_{alg2}$
$H_1 : \mu_{alg1} \neq \mu_{alg2}$

It is possible to assert that there is a difference in mean if the null hypothesis is rejected. The null hypothesis is not rejected if the *p-value* is greater than the significance level α. In other words, there would not be enough evidence to assert a difference between results. When the null hypothesis is rejected (p-*value* $< \alpha$), we can use the *box plots* to determine the superior method. All tests in this section were made using *R*, which is an open-source tool commonly used for statistical analysis.

The *box plots* of Fig. 4 summarize the distributions of all four approaches for the 8400 analyzed graphs. In these graphics, the boxes represent part of the central distribution, which contains 50% of data. Thus, the data scattering is proportional to the box's height. A black line inside the box represents the median. This way, 25% of the data is between the box's edges and the median. The median location indicates if the distributions are symmetrical in the experiments. Lastly, circles indicate outliers. The *box plots* for each algorithm measures the precision, recall, and F-measure of the algorithms.

The *box plots* show the VE algorithm having the lowest median (0.288) and IC with the highest median (0.414) for F-measure. However, all algorithms have high F-measure amplitudes. Looking at the precision *box plot*, we can see that IC has the highest median

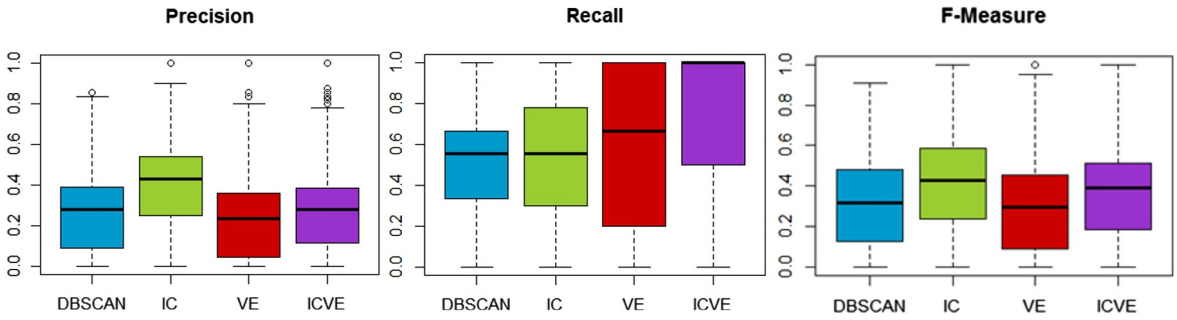

Fig. 4. Box plot of the automatic results for each algorithm.

Table 2
F-measure results from Wilcoxon test and Cliff's Delta effect size for the automatic experiment. The sign between parenthesis represents if the CI is positive or negative when comparing the algorithms.

F-measure	DBSCAN vs. IC	DBSCAN vs. VE	DBSCAN vs. ICVE	IC vs. VE	IC vs. ICVE	VE vs. ICVE
p-value	2.2×10^{-16} (−)	1.291×10^{-16} (+)	2.2×10^{-16} (−)	2.2×10^{-16} (+)	2.2×10^{-16} (+)	2.2×10^{-16} (−)
Effect size	−0.2587116 (small)	0.08436762 (negligible)	−0.1062623 (negligible)	0.3150409 (small)	0.1450793 (negligible)	−0.1806847 (small)

as well. However, IC has the lowest recall, while VE and ICVE have the highest recall of all algorithms, with ICVE winning due to his higher median at the maximum value. By analyzing the *p-values* from Table 2, the IC algorithm provided higher F-measure results than the other three algorithms ($p\text{-value} < \alpha$), even after applying the Bonferroni correction. This is also supported by the effect size calculated using Cliff's Delta method, where IC has a small difference from DBSCAN and VE.

Considering the high amplitude of all algorithms, we decided to split the analysis into two groups: one considering only random noise graphs and another for using the monotonic noise graphs. Fig. 5 illustrates the *box plots* of both groups for each algorithm. The *box plots* show that each algorithm had considerable different results for each group, especially the DBSCAN algorithm, showing worst results when using random noise and a much better result when dealing with monotonic noise. These observations can be confirmed by looking at Table 3, showing that DBSCAN indeed had worse results with random noise graphs and better results than VE and ICVE algorithms with monotonic noise. However, the ICVE algorithm proved to be better with random noise due to its higher recall and above average precision, while the IC algorithm proved to be better with monotonic noise since it got a higher precision in comparison with the rest. This is supported by the effect size between the algorithms shown in Table 3. The high amplitude of the algorithms, even after dividing into two groups (random and monotonic noise) is due to the seven iterations used in the experiment, which increases the graph sizes by doubling the graph's size from the previous iteration.

These results indicate that the IC and ICVE algorithms provide overall better collapses than the other algorithms, including the DBSCAN, in both random and monotonic noise graphs, independently of the graph size. Thus, we can conclude that the inter-cluster verification does indeed provide better clusters when considering DAG, Linear, or Tree graphs due to its nature — it better adapts to the neighborhood by shaping the cluster through distance checks to the farthest existing member already inside the cluster instead of checking only the direct nearest member of the candidate neighbor. This results in a behavior that creates close-knit clusters and consequently avoids overgrowing the cluster to encompass the majority of the graph when nodes have small and progressive variations in relation to neighbors. Meanwhile, the DBSCAN compares the candidate neighbor only with the nearest member of a cluster and therefore tries to compensate this deficiency by reducing the distance threshold in order to create correct clusters according to the oracle. This results in an increased recall at the cost of precision, since it will create many clusters and only a few of these are equivalent to the ones in the oracle. On the other hand, the ICVE stayed behind the IC algorithm because it refined too much in the cluster construction, resulting in more close-knit clusters than necessary and, consequentially, farther from the oracle than IC. This is even more apparent in the monotonic noise graphs by looking at the effect size metric, where the IC algorithm provided the best results in comparison to all the other algorithms with a good margin. However, the variable epsilon approach (VE), when used alone, proved to be detrimental and achieved only better results than the DBSCAN when dealing with monotonic noise, which would correlate to a graph with smooth value transitions.

4.1.3. Threats to validity

We identified internal and external factors that may influence the results. In relation to internal validity, the graph generation algorithm and noise insertion can affect the results because they are all synthetic in nature. Another threat is related to the automatic evaluation of the generated clusters from each algorithm with the template graph. We are not aware of other work that proposed such method of automatic analysis, which synthetically expand graphs with noise insertion and then use clustering algorithms in order to omit the noise vertex, trying to return to the original graph, using precision, recall, and F-measure to evaluate each algorithm in order to determine the best solution. Lastly, another threat is related to the training sessions of the algorithms. All algorithms were trained for each category with varying graph sizes, selecting the best overall parameters values for it depending on the graph type and noise insertion (i.e., random or monotonic). We did not split the training sessions to find the best configurations for each different graph sizes used by the iterations of each category. This can change the results of the algorithms when the graph size differs too much between iterations, especially when comparing the initial iterations with the two last iterations, where the graphs have thousands of more vertices than the previous iterations. Thus, the training session selected the parameters that provided the best overall quality for clustering the graph independently of its size, which could have clusters differing from a few dozen vertices in small graphs to hundreds of vertices in larger graphs since only the

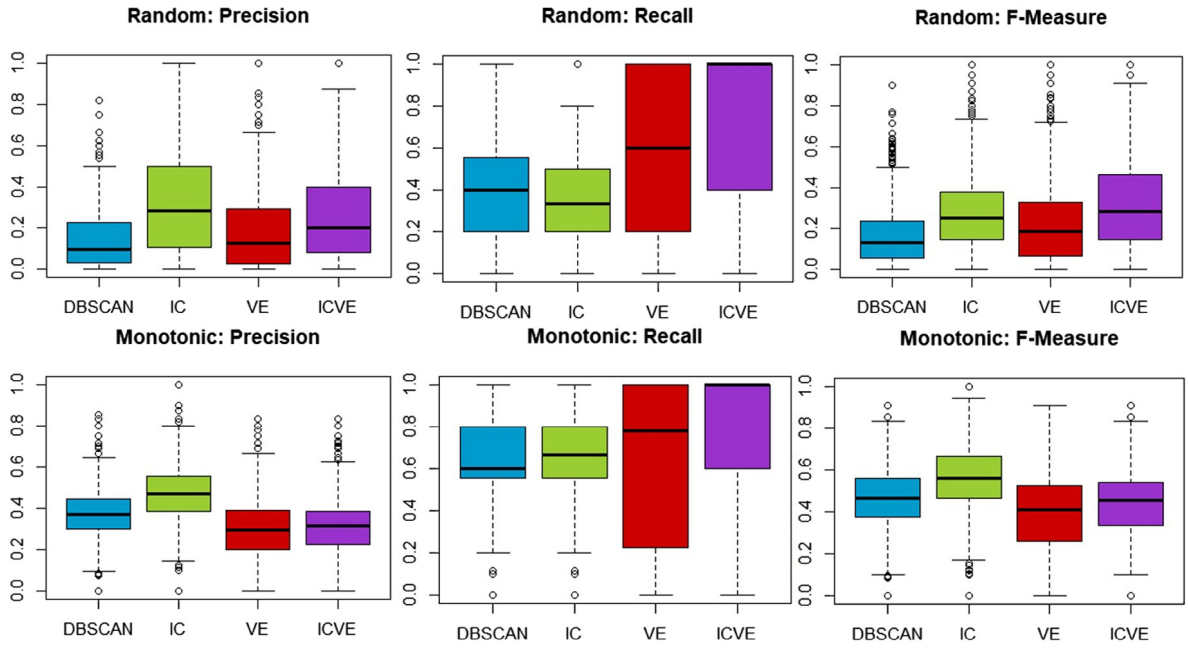

Fig. 5. Divided *box plots* of the automatic experiment for the random and monotonic noise.

Table 3
F-measure results from Wilcoxon test and Cliff's Delta effect size for the automatic experiment divided into two groups: Random and Monotonic noise.

F-measure		DBSCAN vs. IC	DBSCAN vs. VE	DBSCAN vs. ICVE	IC vs. VE	IC vs. ICVE	VE vs. ICVE
Random noise	*p*-value	2.2×10^{-16} (−)	2.2×10^{-16} (−)	2.2×10^{-16} (−)	2.2×10^{-16} (+)	2.2×10^{-16} (−)	2.2×10^{-16} (−)
	Effect size	−0.3997387 (medium)	−0.1289706 (negligible)	−0.4000328 (medium)	0.2138566 (small)	−0.08052234 (negligible)	−0.2605855 (small)
Monotonic noise	*p*-value	2.2×10^{-16} (−)	2.2×10^{-16} (+)	2.2×10^{-16} (+)	2.2×10^{-16} (+)	2.2×10^{-16} (+)	2.2×10^{-16} (−)
	Effect size	−0.3449252 (medium)	0.2465062 (small)	0.1366312 (negligible)	0.5193248 (large)	0.4567062 (medium)	−0.1076311 (negligible)

noise varied in each iteration, while the number of correct clusters in the Oracle maintained the same independently of the graph size. This behavior is reflected in the high amplitude in all algorithms and resulted in having similar precision and recall in all algorithms, even after splitting the experiment results between the two noise groups (random and monotonic). However, it might be possible to achieve better tuning for each algorithm by changing some of the factors used during the training session. For example, we aimed at maximizing the overall F-Measure for each algorithm and graph type, independently of the graph size. Furthermore, since training is an exhaustive and time-consuming operation, we also limited the training sample size to only 5 iterations, with 20 oracle graphs in each iteration and each oracle graph generating only three noise graphs to be used for training. It might be possible to achieve better tuning by increasing the training sample size and specializing each training session to a specific graph size category.

Regarding external validity, we mitigated sample bias by randomly generating multiple different template graphs for each iteration of the experiment. Each template graph was also used to spawn multiple noise graphs, where each one was used by all four clustering algorithms, mitigating any bias in the algorithm evaluation since all four algorithms were using the same noise graphs during each step of the experiment.

4.2. Experts experiment execution

Our research question aims at identifying the most effective similarity summarization algorithm for reducing the information to be analyzed, while still preserving the graph semantics. To do so, we also executed experiments with human judges to evaluate each one of the four algorithms in order to select the most appropriate, based on their opinion, for each category of graph.

4.2.1. Materials and method

In this experiment, due to the nature of using human beings as an oracle, we had to use a smaller set of graphs with a hundred vertices instead of using thousands of graphs with thousands of vertices like in the previous experiment. We used the same types of graphs from the automatic experiment: three graph types (DAG, tree, linear) and two graph characteristics (random and monotonic noise). The combination results in six different categories to be analyzed by each subject: (1) randomized DAG; (2) randomized tree; (3) randomized linear; (4) monotonic DAG; (5) monotonic tree; and (6) monotonic linear. Furthermore, we used the same graph generation technique described in the previous section to generate all synthetic graphs, with only one numeric attribute for the vertex value. Thus, the similarity between neighbor vertices would be based only on this numeric value.

We decided to use two different samples for each one of the six categories to reduce bias, resulting in having the judge to look and analyze 48 graphs in total, since each sample consists of the original graph and other four collapsed graphs, one for each algorithm. However, at the same time, we could not increase this number to three samples since each new sample would result in an additional 24 graphs to be analyzed by the judge. Then, the subject selects one of the four collapsed graphs that represents, in his opinion, a good collapse for that specific case.

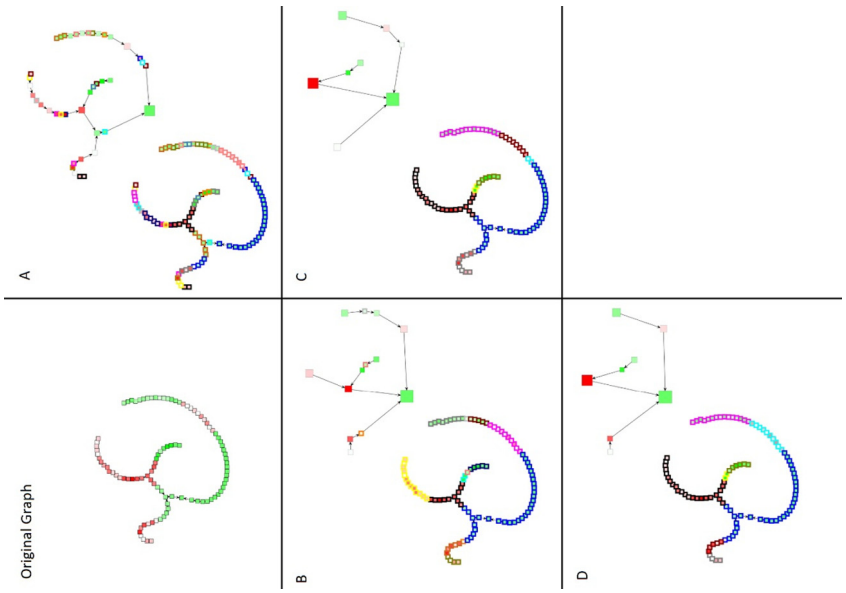

Fig. 6. Example of a sheet given to the judge for his analysis of the algorithms. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

In order to reduce the length of the experiment, we previously trained all algorithms using the same principle from the automatic experiment since we were also using synthetic graphs and also considering the graph sizes that were going to be used in this experiment. Thus, the judge does not need to fine-tune each algorithm for each graph and only need to select the algorithm that generated the best result according to his opinion. We also used gradient colors instead of numbers to represent the vertex's value in the graph, in order to make easier for the subjects, resembling what a tool such as *Prov Viewer* would show. Red gradient represents negative values, green gradient represents positive values, and white gradient represent values close to zero.

We presented all the graphs in a comprised way, with the original graph at the top left. Fig. 6 illustrates the format used in the experiment, where the top left graph (mostly using only green, red, and white gradients) is the original graph and the other four colored graphs are the summarization results from each algorithm. Note that each output section of the sheet has two graphs: a graph with colored borders, showing the composition of each *collapse group*, and a summarized graph, which is the result after following the proposed collapses. Sequential vertices that had the same border color belong to the same *collapse group*. For example, in the output "C" from Fig. 6, we can see a chain of blue-colored vertices near the middle of the graph. All these blue-colored vertices belong to the same *collapse group* since they all share the same border color. Thus, each colored chain of vertices represents a different *collapse group*. Vertices with colored borders that appear in the smaller summarized graph represent outliers, which are *collapse groups* comprised of only a single vertex. Moreover, the size of the vertex in the summarized graph is proportional to the number of vertices in the *collapse group* that generated it. Thus, we can see in the same example that the blue chain of vertices from letter "C" generated a big green vertex in the summarized graph due to the high quantity of vertices in the chain and their predominant color in the original graph is a bright green. We decided to use these printed versions of the graph to expedite the analysis process, allowing the judge to compare side-by-side the different outputs.

For this experiment, we decided to use a simple vote process: each subject should point which category represents the most appropriate algorithm in his opinion. Thus, the experiment execution was divided into two stages: (1) a pilot experiment to detect any issues that needed to be addressed and (2) the experiment itself. During the pilot, volunteers were required to analyze each graph and select one of the four algorithms, picking the one that proposed the most appropriate *collapse group* s in his/her opinion. The pilot experiment was applied to three volunteers from the university.

In order to avoid biased answers, we randomized the algorithms order, forcing the subject to analyze all the four algorithms' output. We also adopted a speak-aloud approach in order to identify how each volunteer reached their final decision. Their recorded decision-making process can also be used to fine-tune the algorithms in the future. We incorporated this approach because the volunteers were not giving any feedback to back up their selection during the pilot.

Considering that all twelve graphs were synthetic, we decided to add two new graphs in the experiment. These graphs were real provenance graphs generated from a gameplay session of a racing game using the PinG approach. Both provenance graphs belonged to the same game session, and thus had the same vertices. They differentiated only in the displayed graph layout. The first graph used a simple graph layout (the same used for all the previous graphs) where we omit all the domain information, displaying it as another synthetic domain-less graph, with each vertex having a single numeric value without any semantics. The second graph used a spatio-reference layout, placing all vertices in their spatial location when the action represented by the vertex was executed. Moreover, we explained to the volunteer the semantics of this graph, as well as the domain it belongings. By asking the volunteer to analyze the same provenance graph twice, but having a different perception from each, we were able to discern if the graph domain and spatio-referencing of the data generated a significant impact in the summarization process.

Lastly, we made a final change in the experiment by introducing a post-experiment questionnaire,[2] designed to be answered at the

[2] All questionnaires are available at: https://github.com/gems-uff/prov-viewer/tree/master/Documents.

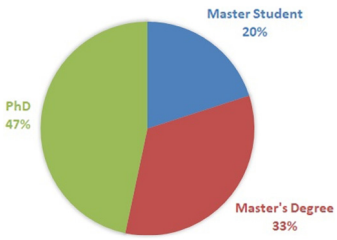

Fig. 7. Volunteers' characterization chart.

Table 4
Volunteers' characterization table.

Subject	Academic education	Conclusion year	Graph knowledge
P1	Master's program	2017	No formal knowledge
P2	**Ph.D.**	2010	Studied
P3	Master's degree	2017	No formal knowledge
P4	**Ph.D.**	2012	Studied
P5	Master's degree	2019	Studied
P6	**Ph.D.**	2011	Studied
P7	Master's degree	2016	Studied
P8	**Ph.D.**	1998	**Expert**
P9	Master's program	2017	Studied
P10	**Ph.D.**	2014	**Expert**
P11	Master's degree	2017	Read about
P12	**Ph.D.**	2003	Read about
P13	**Ph.D.**	1998	**Expert**
P14	Master's degree	2017	Studied
P15	Master's program	2017	Read about

end of the session and allowing us to gather additional feedback and further insights from the subject that were not captured by the talk-aloud strategy. This questionnaire included two questions related to the final two real provenance graphs, allowing us to verify if the domain information had any impact on the analysis.

After changing the original experiment structure used during the pilot, the resulting experiment plan was divided into three stages: (1) Generating the graphs, (2) running the experiment with judges, and (3) analyzing the results. We executed the first stage before running the experiment. In this stage, we trained all the algorithms for each category. Then, we generated two graphs for each category and ran the algorithms for each graph, creating their corresponding paper sheets with the original graph and the four different outputs.

The next stage was the experiment execution with the judges. We applied the experiment with fifteen volunteers closely related to computer science. Fig. 7 illustrates the characterization of the judges by their degree. It is important to note that three of the Ph.D. volunteers were considered specialists in graphs. As we can see by looking at this figure, 47% of the volunteers had a Ph.D. degree and were professors in the university. The remaining volunteers were divided in 33% having the Master's Degree and currently working on their Ph.D. degree and the remaining 20% being students under the master's program. Table 4 shows in more detail the characterization of each volunteer.

Each experiment had an average duration of one hour and a half and was conducted individually due to the talk-aloud strategy. The volunteers analyzed 14 paper sheets, where each contained one of the fourteen graphs and the resulting outputs from each algorithm (as illustrated in Fig. 6), marking their answers in a spreadsheet. The post-experiment questionnaire was handed to the volunteers after they finished analyzing the 14 sheets.

The last stage of the experiment was the result analysis. We performed a statistical analysis over the results by means of hypothesis test in order to compare the obtained results of each

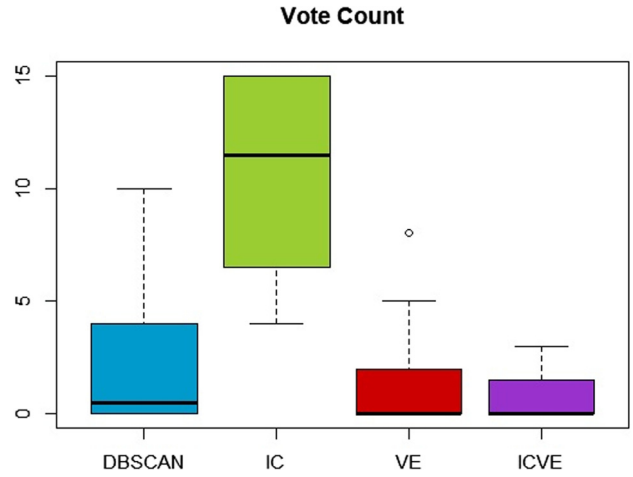

Fig. 8. Box plot of the judge's results for each algorithm.

collapse algorithm. An important factor for the design of the experiment concerns the definition of the significance level used during statistical analysis. We used a confidence interval of 95%, which translates to $\alpha = 0.05$, where α is the probability of rejecting the null hypothesis given that it is true (Type I error) [22].

4.2.2. Results and discussion

Fig. 8 summarizes the results of the experiment for all four algorithms considering all twelve synthetic graphs analyzed by the fifteen judges. By analyzing the box plot, we can visually see that the IC algorithm had better results than any of the other three algorithms, confirming the results from the automatic experiment. The DBSCAN algorithm was ranked in the second position with some situations almost reaching the median of IC. However, it is only possible to assert this assumption by running statistical tests.

Similar to the automatic experiment, we ran a normality test using Shapiro–Wilk test [21] and noticed that the collected data does not follow a normal distribution. Therefore, we once again used the Wilcoxon Matched-Pairs Signed-Rank test because it compares two means from two different samples over the same observation (*e.g.*, graph), which fits our experiment design.

Again, we run six analyses to compare the algorithms: (1) DBSCAN vs. IC; (2) DBSCAN vs. VE; (3) DBSCAN vs. ICVE; (4) IC vs. VE; (5) IC vs. ICVE; and (6) VE vs. ICVE. Considering that we are using six comparisons, we also decided to use the Bonferroni correction in this experiment, which translates to $\alpha = 0.00833$. We adopted the same format from the automatic experiment for the hypothesis in our tests, naming *alg1* as the first algorithm used in the comparison and *alg2* the second algorithm used in the comparison:

$$H_0 : \mu_{alg1} = \mu_{alg2}$$
$$H_1 : \mu_{alg1} \neq \mu_{alg2}.$$

By analyzing the *p-values,* CI, and the effect size from Table 5, we can see that the IC algorithm provided better results using $\alpha = 5\%$, even after applying the Bonferroni correction. Therefore, the null hypothesis was rejected in all comparisons involving the IC algorithm. However, there is not enough evidence ($p\text{-value} > \alpha$) to assert the difference between results when comparing the other algorithms to provide a ranking, only enough to know which of all four is the best. This finding matches with our initial visual analysis of the box plot from this experiment (Fig. 8), where we could clearly see that algorithm IC had better results than all other analyzed algorithms. Furthermore, this result also coincides with

Table 5
Results from Wilcoxon test for the judge experiment. The sign between brackets represents if the CI is positive or negative when comparing the algorithms.

F-measure	DBSCAN vs. IC	DBSCAN vs. VE	DBSCAN vs. ICVE	IC vs. VE	IC vs. ICVE	VE vs. ICVE
p-value	**0.008062** (−)	0.6741 (−)	0.1434 (−)	**0.004673** (+)	**0.002328** (+)	0.4568 (+)
Effect size	−0.8611111 (large)	0.1666667 (small)	0.2291667 (small)	0.9236111 (large)	1 (large)	0.05555556 (negligible)

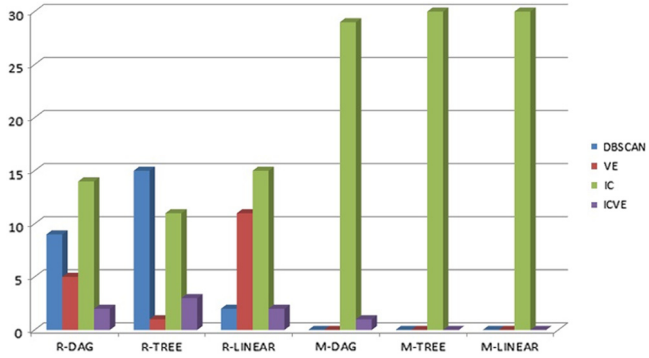

Fig. 9. Experiment results from each one of the categories in the experiment.

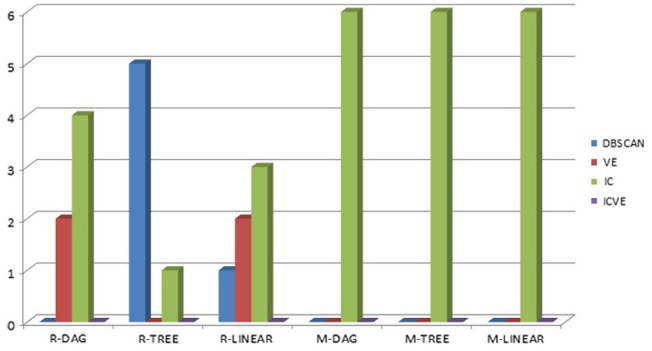

Fig. 10. Expert's results.

the initial findings from the automatic experiment, where the IC algorithm was also the one that had a highest F-measure. Thus, the initial findings from both experiments are in tune.

However, if we examine the box plot closely, we can see that the DBSCAN amplitude almost reaches the IC median. This might mean that in some cases the DBSCAN can be at least equal to IC. If we break down the results for each category, as illustrated by Fig. 9, we can see a large difference between random and monotonic noise results. The IC algorithm dominates in monotonic noise graphs and is adequate in random graphs, thus making it an ideal general-purpose algorithm for summarizing provenance graphs. Meanwhile, the ICVE algorithm appears to be always inappropriate. The judges did not like this algorithm because in most cases, as the majority of the judges summarized, it is "*too sensitive to changes and creates too many different clusters, not summarizing much the data*" (P8).

Considering the random categories, the DBSCAN appears to be only inappropriate when dealing with linear graphs. This occurred because it overextended the reach of a cluster, grouping almost half of the graph in a single cluster because it failed to detect the slope variations (both increasing and decreasing), stopping only when encountered an extremely sharp slope. As one of the experts said (P13): "*by identifying this entire region, I remove it from the dispute*" when he/she was referring to half the graph that the algorithm failed to divide that was composed of a light green segment followed by a dark green, then another light green segment and ending with a whitish red segment. Despite VE showing to be inappropriate with tree-graphs, the resulting clusters were very similar to those from DBSCAN, showing small differences that led them to not be chosen when in doubt between DBSCAN and VE.

Table 6 illustrates the statistical analysis of the Random Noise graphs used in this experiment. We omitted the analysis of the Monotonic Noise since the IC algorithm dominated in this situation. The results show that the IC algorithm is indeed better than VE and ICVE algorithms. However, there is not enough statistical evidence to make clains with the DBSCAN since the null hypothesis was not rejected (p-value $= 0.0855 > 0.00833$). Overall, this finding also matches with the ones from the previous experiment, where IC algorithm proved to be better in both monotonic and random noise.

Fig. 10 illustrates the results only from the experts' point of view and Fig. 11 provides a summary of these results in the form of box plots. None of the expert judges selected the ICVE algorithm in any of the categories, corroborating that it is the least favorable algorithm among the judges. This figure also shows that IC is generally the most appropriate algorithm, with the exception when dealing with the random tree category, where the DBSCAN showed better results. Nonetheless, these results are similar to those presented in Fig. 9, where the random tree was the only case when IC lost to another algorithm, which was also the DBSCAN. However, despite these observations for the random graphs, there are not enough statistical data to determine the most appropriate algorithm for each type of graph since each category only had two graph samples and 15 human judges. Thus, having an external threat to validity against generalization of the results and few data samples for each graph type (i.e., only 30 and the ideal is at least 50 samples). Table 7 shows the statistical analysis, which corroborates our assumptions of not having enough data for analyzing each graph type since almost no one had the null hypothesis violated.

As mentioned before, all those 12 graphs were synthetic in nature, although based on structures frequently found on provenance generated from games. However, in the same experiment, we had one real provenance graph with two different visualizations, which was based on a car game. We added this graph to have an initial analysis of how the algorithms would behave in a real scenario. The first time it appeared in the experiment, the judges treated it like any of the other synthetic graphs since the real game data looked exactly like any other of the artificially generated graphs from the templates and no additional information was given to them regarding the origin of the graph. However, unlike the previous graphs, the car's provenance graph only contained green colored vertices due to the nature of the game since the value represented the car's speed. Furthermore, since only the player was present, the resulting provenance graph was from the linear type. Fig. 12 illustrates the results before and after revealing the details about the graph, where "Linear-Car" represents the judges' first interaction with the graph, thinking it was a synthetic graph, and "Geo-Car" represents their second interaction, where they had domain information and the graph's vertices were displayed over the race

Table 6
F-measure results from Wilcoxon test and Cliff's Delta effect size for the random noise of the expert experiment. Used Bonferroni correction (0.00833).

F-measure		DBSCAN vs. IC	DBSCAN vs. VE	DBSCAN vs. ICVE	IC vs. VE	IC vs. ICVE	VE vs. ICVE
Random noise	*p*-value	0.0855 (−)	0.1721 (+)	**0.0009764** (+)	**0.002351** (+)	**1.529 × 10^{-6}** (+)	0.04288 (+)
	Effect size	−0.1555556 (small)	0.1 (negligible)	0.2111111 (small)	0.2555556 (small)	0.3666667 (medium)	0.1111111 (negligible)

Table 7
F-measure results from Wilcoxon test and Cliff's Delta effect size for the random noise of the expert experiment divided into three groups: DAG, Tree, and Linear Graphs. Used Bonferroni correction (0.00833).

F-measure		DBSCAN vs. IC	DBSCAN vs. VE	DBSCAN vs. ICVE	IC vs. VE	IC vs. ICVE	VE vs. ICVE
Dag	*p*-value	0.3053 (−)	0.3014 (+)	0.03937 (+)	0.04117 (+)	**0.002972** (+)	0.2986 (+)
	Effect size	−0.1666667 (small)	0.1333333 (negligible)	0.2333333 (small)	0.3 (small)	0.4 (medium)	0.1 (negligible)
Tree	*p*-value	0.4413 (+)	Error	**0.005053** (+)	Error	0.03551 (+)	Error
	Effect size	0.1333333 (negligible)	0.4666667 (medium)	0.4 (medium)	0.3333333 (medium)	0.2666667 (small)	−0.06666667 (negligible)
Linear	*p*-value	0.001772 (−)	0.01403 (−)	1 (+)	0.4413 (+)	**0.001772** (+)	0.01403 (+)
	Effect size	−0.4333333 (medium)	−0.3 (small)	1.853986 × 10^{-18} (negligible)	0.1333333 (negligible)	0.4333333 (medium)	0.3 (small)

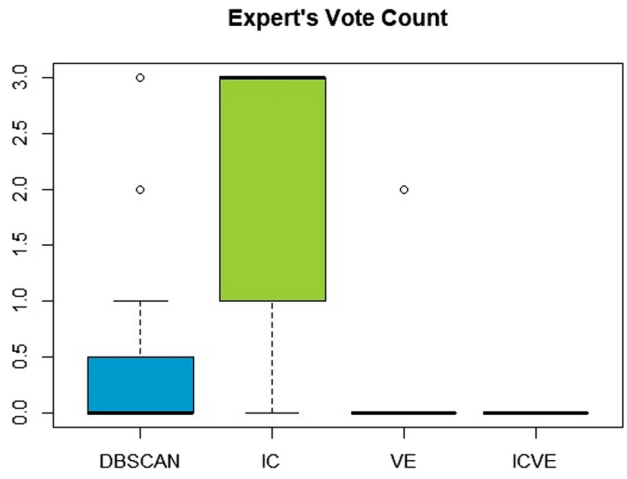

Fig. 11. Box plot of only the three experts for each algorithm.

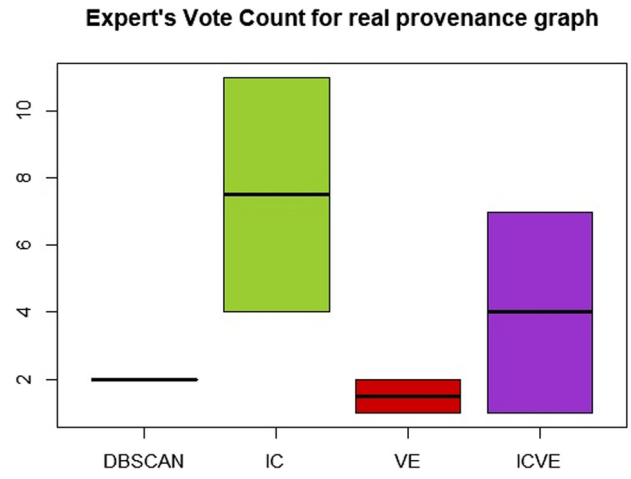

Fig. 13. Box plot of both Linear and Geo car provenance graphs.

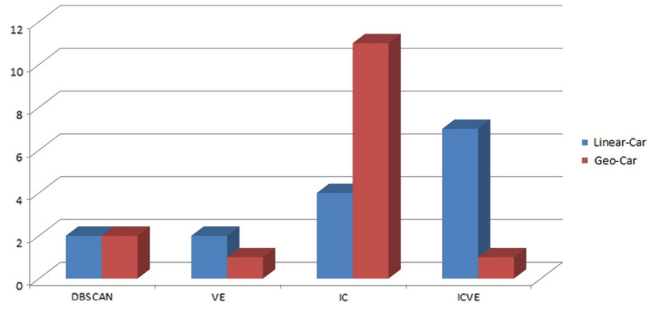

Fig. 12. Results from the real provenance graph.

track map, showing their exact locations. Fig. 13 also provides a summary of these results in the form of box plots.

Surprisingly, almost half of the judges picked the ICVE algorithm on the "Linear-Car" graph. However, all of them reported difficulties when analyzing the graph due to the higher number of vertices and also because all vertices were green-colored. After knowing the graph origins and what it represented, thirteen of the judges selected another algorithm when analyzing the "Geo-Car" graph, with the majority choosing the IC algorithm. As the majority of the judges said, knowing additional data information, such as domain, changed their perception because "*the contexts made me (judge P2) think about the causes of variation*". Moreover, the judges also agreed that, in this specific case, geo-referencing the data helped them in their decision-making process because "*the relative position of the vertices in relation to their neighbors helps*" (P8) when they were trying to see the speed difference between vertices due to their distance. Moreover, some judges were also in agreement that "*the context of the game shows that position matters to analyze the number of curves* (from the race track), *which impacts in the performance analysis* (of the player)" (P1). However, one of the experts (P10) said that he "*abstracted the application*" and "*the additional information didn't impact in my (his) analysis*". Nevertheless, that same specialist selected the IC algorithm in both graphs (Linear-Car and Geo-Car). Fig. 14 shows the "linear-car" graph superimposed in the race track, which was shown to the judges when explaining the graph's domain and Fig. 15 show both graphs presented to the judges for their evaluation. Notice that we

Fig. 14. Provenance Graph from the racing game used in the experiment. Figure (a) illustrates the "linear-car" graph superimposed in the race track, generating the "Geo-Car" visualization and (b) illustrates an example of one of the outputs from the IC algorithms. Vertices color represent the car's speed. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

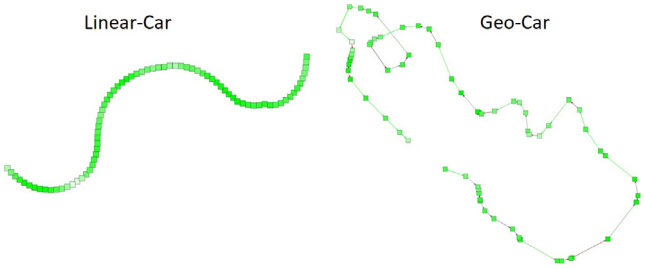

Fig. 15. Linear-Car and Geo-Car graphs presented to the judges. Both graphs are the same as the only difference being in the vertex positioning. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

removed the background from Geo-Car after presenting it to the judges so they could easily detect and contrast the vertices to select one of the four collapses. However, they could consult the original Geo-Cal graph (with the race track background) anytime.

These results point out that the IC algorithm provides overall better collapses than the other algorithms in both random and, in particular, monotonic noise graphs. However, when considering only Random Noise graphs, the DBSCAN tied with the IC algorithm and there were not enough data to determine which is the best solution. These results match with the ones from the automatic experiment, where the IC algorithm also proved to be the better algorithm. Therefore, with the findings of both experiments, we can assume that the new IC algorithm for clustering DAG, Tree, and Linear graphs is more adequate than the other algorithms. Furthermore, since the results from the automatic experiment matched with the one using human judges, then we can also conclude that this automatic validation is also a viable method for measuring clustering effectiveness of the algorithms.

4.2.3. Threats to validity

Despite the care in reducing the threats to the validity of the experiment, there are factors that can influence the results. In relation to internal validity, the selection of participants can affect the results because of the natural variation in human preference in picking the answers, since there are no right or wrong answers in this experiment. Furthermore, the experiment was executed with volunteers, since they generally are more motivated for executing tasks. Any volunteer could choose to be dismissed from the experiment and be released earlier. One possible threat is related to each individual perception of different colors and their shades in the graph. To minimize this problem, we also printed different versions of the material for color-blind individuals. Nonetheless, one of the experts in Visual Computing correctly remarked that the different shades used (i.e., red, green, and white) could also be a threat due to their different chromatic distance when analyzing the graphs. Another threat is related to graph sizes used. Unlike the automatic experiment, we used graphs that could be considered small, with around a hundred vertices, in order to not confuse the volunteer during the analysis.

5. Related work

Our related works were selected from those used by the game industry for clustering and summarizing gameplay telemetry data and other graph-based research for visualization of gameplay data that uses some kind of clustering technique to group graph nodes. We are only interested in this moment in using summarization for noise reduction and aiding the visual exploration of the graph.

Play-Graph [23] is a graph-based approach to formally describe and visualize game session data by using a graph visualization. It has multiple variables and their interrelations along with the temporal progression of players. It uses spatial information to render the graph in a game scene and cluster nearby nodes to form a single node that provides statistical information in the scene's region (*e.g.*, player race distribution at that location). However, its clustering method disregards temporal information and indiscriminately cluster similar states that are spatially close, losing the real sequence of events.

Another approach is *Playtracer* [24], which is a visual tool designed to illustrate how groups of players move through the game space. Thus, *Playtracer* aids the designer by showing common pathways and alternatives that players used to succeed or fail in their tasks, identifying pitfalls and anomalies in the scene. However, *Playtracer* does not take into consideration temporal information or the actual game map. The temporal information would allow stating the order of events in the game, shedding more light in the player's behavior, while the map of the scene would show exactly where these pathways, pitfalls, and game anomalies were in the game. Moreover, in order to solve problems related to the number of visible states, *Playtracer* uses an aggressive technique to cluster nearby states together to make a cleaner visualization, disregarding sequence of events and thus forming cycles in the graph.

Both Play-graph and Playtracer approaches are focused on state-changes analysis and therefore deal with a different type of game data, which is much coarser grained than event-based approaches. Furthermore, due to the different type of telemetry tracking (i.e., states instead of events), the tracked data possess much less variance since the tracking is limited to visited states instead of tracking the sequence of executed actions. Nevertheless, those approaches use Quality Threshold clustering technique that is very similar to the DBSCAN clustering algorithm, which we used to compare our algorithms.

Other related works include common approaches adopted by the game industry and game research, such as heat maps [25] or trajectory analysis [26,27], which display paths in a map. These approaches use visualizations based on the evaluation of one or two variables from the game data, providing an easy to read and intuitive interpretation of the data distribution. However, heat maps only aggregate variables that follow specified restrictions (*i.e.*, death locations) to show density distribution over the scene. Similarly, trajectory maps also aggregate equivalent paths. Nevertheless, they do not provide any insight on the executed actions, hindering influences between events. However, we did not compare them with our proposed algorithms because they are used for a different type of data analysis. The heat map is commonly used to aggregate data within the same 2D spatial region to analyze the

most visited sections of the map by players or can also be used for specific occurrences such as places where players died the most. Similarly, trajectory analysis is used to map the player's navigation in the game scenario.

6. Conclusion

This paper introduced new perspectives on game session analysis through graphs, leveraging the current state of the art based on game session analysis along with some common techniques to deal with information overload. Furthermore, we proposed new graph algorithms based on the DBSCAN that take into consideration the temporal sequence of information of a provenance graph to summarize tracked data to a more manageable size through the usage of collapses strategies. None of the existing approaches considered using temporal information in their collapse strategies. These collapses intend to reduce the overall graph size by hiding sections in the graph that alone were not significant enough to produce a meaningful impact on the game and did not offer any useful information for analysis.

Our proposal enhances the identification of sections or vertices that are different from its neighbors or the expected behavior. In a game context, vertices that were not collapsed represent drastic changes in the game state and are worth for displaying at the analysis, while all collapsed ones fluctuate around the same state for a given attribute. Therefore, the resulting collapsed graph is useful to confirm the initial hypothesis of sections that the player had difficulties in the game by identifying sections with multiple nearby vertices because each vertex in the collapsed graph represents a major variation in the game state. However, as shown by the judge experiment, this type of collapse can also work on graphs that are outside of the game domain since the majority of the analyzed graphs were not related to any domain.

The experimental results show that the inter-cluster verification variant (IC) provides better results than the *DBSCAN* for collapsing similar segments in the graph. The statistical analysis of both experiments shows that this is true in all the studied cases, including random and monotonic behavior and different graph types. The variable epsilon variant (VE), which is responsible for generating a customized epsilon for each cluster, is inferior to the IC variant when used alone. However, in the automatic experiments, the combination of both variants, resulting in the ICVE algorithm, provided better results when dealing with more randomized values. Unfortunately, the judges disliked the algorithm due to its sensitivity and thus it requires more refining before being used. Answering our research question, the IC variant showed to be the most effective algorithm for reducing the information while also preserving its overall semantics in all instances according to the judges and the automatic experiments.

Future work includes further refinement of the variants and resulting algorithms especially the VE variant. Another future work is related to the novel approach of the automatic experiment for evaluating graph summarization, further improving it and running more experiments to validate it. However, the initial result from both experiments shows an alignment of algorithm selection, which can be an indication of its possible effectiveness when dealing with graphs with similar behavior of those we used, which can be very common across multiple domains.

Acknowledgments

We would like to thank CNPq, Brazil, FAPERJ, Brazil, and CAPES, Brazil for the financial support.

References

[1] M. El-Nasr, A. Drachen, A. Canossa (Eds.), Game Analytics - Maximizing the Value of Player Data, Springer Science & Business Media, 2013.
[2] A. Drachen, R. Sifa, C. Bauckhage, C. Thurau, Guns, swords and data: Clustering of player behavior in computer games in the wild, in: Conf. Comput. Intell. Games CIG, 2012, pp. 163–170.
[3] A. Drachen, A. Canossa, Towards gameplay analysis via gameplay metrics, in: Int. MindTrek Conf. Everyday Life Ubiquitous Era, pp. 202–209, 2009.
[4] A.R. Gagné, M. Seif El-Nasr, C.D. Shaw, Analysis of telemetry data from a real-time strategy game: A case study, Comput. Entertain. CIE 10 (3) (2012) 2:1–2:25.
[5] C. Pedersen, J. Togelius, G.N. Yannakakis, Modeling player experience for content creation, Trans. Comput. Intell. AI Games T-CIAIG 2 (1) (2010) 54–67.
[6] A. Drachen, C. Thurau, R. Sifa, C. Bauckhage, A Comparison of methods for player clustering via behavioral telemetry, Found. Digit. Games FDG (2013).
[7] B.G. Weber, M. John, M. Mateas, A. Jhala, Modeling player retention in madden NFL 11, in: Innov. Appl. Artif. in: Intell. Conf. IAAI, 2011.
[8] J. Sander, M. Ester, H.-P. Kriegel, X. Xu, Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications, Data Min. Knowl. Discov. 2 (2) (1998) 169–194.
[9] R. Sibson, SLINK: An optimally efficient algorithm for the single-link cluster method, Comput. J. 16 (1) (1973) 30–34.
[10] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise, in Proceedings of the Second, in: International Conference on Knowledge Discovery and Data Mining, Portland, Oregon, 1996, pp. 226–231.
[11] T. Kohwalter, E. Clua, L. Murta, Game flux analysis with provenance, Adv. Comput. Entertain. ACE (2013) 320–331.
[12] T. Kohwalter, E. Clua, L. Murta, Provenance in Games, in: Braz. Symp. Games Digit. Entertain. SBGAMES, 2012, pp. 162–171.
[13] L. Moreau, P. Missier, PROV-DM: The PROV Data Model, 2010. [Online]. Available: http://www.w3.org/TR/prov-dm/ [Accessed: 21.03.13].
[14] PREMIS Working Group, Data Dictionary for Preservation Metadata, Implementation Strategies (PREMIS), OCLC Online Computer Library Center & Research Libraries Group, Final report, 2005.
[15] L. Moreau, I. Foster, J. Freire, J. Frew, P. Groth, D. McGuiness, IPAW, 2002. [Online]. Available: http://www.ipaw.info/ [Accessed: 02.04.13].
[16] L. Moreau, et al., The open provenance model core specification (v1.1, Future Gener. Comput. Syst. 27 (6) (2007) 743–756.
[17] S. Miles, J. Heasley, A. Szalay, L. Moreau, P. Groth, Provenance Challenge WIKI, 2010. [Online]. Available: http://twiki.ipaw.info/bin/view/Challenge/ [Accessed: 26.03.13].
[18] The use of multiple measurements in taxonomic problems - Fisher - 1936 - Annals of Human Genetics - Wiley Online Library. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1111/j.1469-1809.1936.tb02137.x/full [Accessed: 14.12.17].
[19] F.R.S. Karl Pearson, LIII. On lines and planes of closest fit to systems of points in space, Lond. Edinb. Dublin Philos. Mag. J. Sci. 2 (11) (1901) 559–572.
[20] D. Birant, A. Kut, ST-DBSCAN: An algorithm for clustering spatial–temporal data, Data Knowl. Eng. 60 (1) (2007) 208–221.
[21] S.S. Shapiro, M.B. Wilk, An analysis of variance test for normality (complete samples), Biometrika 52 (3/4) (1965) 591.
[22] G.R. Norman, D.L. Streiner, Biostatistics: The Bare Essentials, 3 Pap/Cdr ed., People's Medical Publishing House, Shelton, Conn, 2012.
[23] G. Wallner, Play-Graph: A methodology and visualization approach for the analysis of gameplay data, Found. Digit. Games FDG (2013) 253–260.
[24] Y.-E. Liu, E. Andersen, R. Snider, S. Cooper, Z. Popović, Feature-based projections for effective playtrace analysis, Found. Digit. Games FDG (2011) 69–76.
[25] A. Drachen, A. Canossa, Analyzing spatial user behavior in computer games using geographic information systems, in: MindTrek Conf. Everyday Life Ubiquitous Era, pp. 182–189, 2009.
[26] H.-K. Pao, K.-T. Chen, H.-C. Chang, Game bot detection via avatar trajectory analysis, IEEE Trans. Comput. Intell. AI Games 2 (3) (2010) 162–175.
[27] J.L. Miller, J. Crowcroft, Avatar movement in world of warcraft battlegrounds, in: Netw. Syst. Support Games NetGames, 2009, pp. 1–6.

Troy Kohwalter holds a degree in Computer Science from Universidade Federal Fluminense (2011) and M.S. in Computer also from Universidade Federal Fluminense (2013). Is currently a doctoral student in the Computing Graduate Program at Universidade Federal Fluminense. Operates in Visual Computing Research with an emphasis in Digital Entertainment. Acts on the following topics: provenance, software engineering, education, serious games, game analytics.

Leonardo Murta is an Associate Professor at the Computing Institute of Universidade Federal Fluminense. He holds a Ph.D. and a M.S. degree in Systems Engineering and Computer Science from COPPE/UFRJ, and a B.S. degree in Informatics from IM/UFRJ. He published over 150 papers in journals and conferences and received an ACM SIGSOFT Distinguished Paper Award at ASE 2006. He has served as program committee member of ICSE 2014, associate editor of JBCS since 2013 and JSERD since 2016. His research area is software engineering, and his current research interests include configuration management, software evolution, software architecture, and provenance.

Esteban Clua is professor at Universidade Federal Fluminense and coordinator of Medialab, Young Scientist of the State of Rio in 2009 and 2013. He is undergraduate in Computer Science by Universidade de São Paulo and has master and doctor degree by PUC-Rio. His main research areas are Digital Games, GPUs and Visualization. He is one of the founders of SBGames (Brazilian Symposium of Games and Digital Entertainment) and was the president of Game Committee of the Brazilian Computer Society from 2010 through 2014. He was nominated NVIDIA Fellow in 2015.