5-2022

A Novel Data Lineage Model for Critical Infrastructure and a Solution to a Special Case of the Temporal Graph Reachability Problem

Ian Moncur
University of Arkansas, Fayetteville

A Novel Data Lineage Model for Critical Infrastructure and a Solution to a Special Case of the Temporal Graph Reachability Problem

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science

by

Ian Moncur
University of Arkansas
Bachelor of Science in Computer Science, 2020

May 2022
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

_____
Brajendra Panda, PH.D.
Thesis Director

_____
Dale Thompson, PH.D.
Committee Member

_____
Qinghua Li, PH.D.
Committee Member

ABSTRACT

Rapid and accurate damage assessment is crucial to minimize downtime in critical infrastructure. Dependency on modern technology requires fast and consistent techniques to prevent damage from spreading while also minimizing the impact of damage on system users. One technique to assist in assessment is data lineage, which involves tracing a history of dependencies for data items. The goal of this thesis is to present one novel model and an algorithm that uses data lineage with the goal of being fast and accurate. In function this model operates as a directed graph, with the vertices being data items and edges representing dependencies. Additionally, data is grouped into multiple layers which allows for faster partial damage assessment. Lower layers of the graph consist of more granular data items, while higher layers consist of containers of lower layer data items. By assessing a higher layer, one can immediately conclude that certain portions of the system are undamaged, and those portions may begin operation again. In practice, graph creation is a front-loaded operation that allows immediate action at the time of damage assessment. Depending on the system, this graph will often be cyclic which causes standard assessment to be a computationally slow problem. By tracking the time of dependencies, our graph operates as a subclass of temporal graph, which are graphs that change over time. By taking advantage of unique properties of this subclass, our algorithm is able to function in a way that is nearly only dependent on the number of edges. Put together, the model can run quickly, free up undamaged portions of the system during assessment, and find the minimum amount of damage which needs to be manually assessed.

TABLE OF CONTENTS

1 INTRODUCTION 1

2 BACKGROUND AND RELATED WORK 3

2.1 Critical Infrastructure 3

2.2 Data Lineage 4

2.3 Temporal Graphs 6

3 THE DAMAGE ASSESSMENT MODEL 8

3.1 Definitions 8

3.2 Dependency Graphs 8

3.3 Pay Calculation Example 11

3.4 Propagation of Damage 12

3.5 Separation Of Edges 13

3.6 Static Assessment 14

3.6.1 Static Assessment Examples 14

3.7 Multiple Layers 16

3.7.1 Multiple Layer Example 17

3.8 The Damage Assessment Algorithm 19

3.8.1 Algorithm for Damage Assessment within a layer 20

3.9 Damage Assessment Example 21

3.10 A Similar Reachability Problem 22

3.11 Suggestions for Implementation, Creation, and Storage of Model Data 23

4 SIMULATION AND ANALYSIS OF RESULTS 25

4.1 Number of Edges per Layer Vs. Number of Vertices at the First layer 26

4.2 Number of Edges per Layer Vs. Number of Vertices at the Second Layer 29

4.3 Number of Edges per Vertex Vs. Number of Vertices at the First layer 30

4.4 Number of Top Layer Vertices Vs. Number of Vertices at the First Layer 32

4.5 Number of Children Vs. Number of Vertices at the First Layer 34

4.6 Comparison of Total Damage Found at Different Layers 36

5 CONCLUSIONS AND FUTURE WORK 38

6 WORKS CITED 39

1 INTRODUCTION

As the reach of technology continues to grow, so too does our dependence on it. Data storage is easier than ever, and processing power continues to become cheaper. As a result, more data than ever is being processed and stored. For the vast majority of systems, the scope of data has become too large for a human to parse, much less perform the same operations the system is performing at a given time. This means that full system understanding is far more complex, and should something go wrong in a system, significant amounts of work are required to determine what went wrong. Logging and techniques such as data lineage can mitigate the necessary amount of work by reducing the amount of human review required. There are a variety of potential issues that can arise to create data corruption that range from viruses to hardware issues. If the situation is not noticed or resolved quickly the system may be reusing damaged data, and what may have been a small problem can become significantly worse.

Should a system continue running after some data is corrupted, damage can propagate from just a single data item to potentially the entire system. Whenever data is dependent on damaged data, it may itself become damaged. As a result, damage can spread quickly as more of the system becomes damaged. Without proper preparation, even a small amount of corrupted data can make it so a full evaluation of the system is required for repair, as the scope of damage is unknown. Again, due to the size of these systems, full manual review is impractical, and some technological assistance becomes essential. This is particularly true in systems that require minimal downtime, as a system will need to cease operation when damaged in order to halt damage propagation.

Downtime is always crucial in critical infrastructure. In creating algorithms, a balance typically needs to be struck between the speed of the algorithm and its use of computing resources. However, in critical infrastructure the assumption is often made that speed is the primary goal, and therefore a larger than typical amount of resources can be contributed to

processing and preprocessing. Additionally, even marginal gains in speed or efficiency, particularly with repairs, can become valuable. This is particularly true when doing damage assessment to limit the amount of time the system is down.

Some damage or attacks to a system may be hard to detect and as a result will not always be caught initially causing damage to propagate. The goal of this thesis is to provide a model for data lineage to help detect all potentially damaged data as quickly as possible. Moreover, the damage detected should include only information that is potentially damaged as this requires the minimal amount of manual review for repair. This model may be particularly useful for critical infrastructure as portions of the system may be freed up early during assessment, allowing partial functionality.

The rest of the thesis is organized as follows. Chapter 2 consists of background and related work. Chapter 3 defines the model and the algorithm used to evaluate the model. Chapter 4 shows experiments and results. Chapter 5 presents conclusions and future work.

2 BACKGROUND AND RELATED WORK

2.1 Critical Infrastructure

According to The CISA (Cybersecurity and Infrastructure Security Agency) of the United States “Critical infrastructure describes the physical and cyber systems and assets that are so vital to the United States that their incapacity or destruction would have a debilitating impact on our physical or economic security or public health or safety. The Nation's critical infrastructure provides the essential services that underpin American society.” (Infrastructure Security) Many countries have a similar definition of critical infrastructure. In this definition, critical infrastructure could represent anything ranging from the electrical grid to roads. Throughout this thesis critical infrastructure will focus on cyber systems specifically and could refer to all such systems where minimizing downtime for repairs is crucial. Having downtime is not the goal of any system, but the more users that are harmed by downtime, the more crucial a particular system is as typically more resources will be dedicated for maintenance.

These are typically larger systems or networks of computers that have an important role with many dependent users or perhaps other dependent systems. Technological advancements have made critical infrastructure more prominent particularly as certain technologies become a part of everyday life. Critical infrastructure is vulnerable to a variety of attacks including DDOS (Genge and Siaterlis), Malware (Langer), and spyware. In the case of the SolarWinds attack (Alkhadra et al.) malware made the systems of nine US federal agencies (“Background Press Call”) and at least 100 technology companies vulnerable (most notably to spyware) after creating a backdoor in software that was distributed to them.

Additionally, as dependency on critical infrastructure increases vulnerability and attack frequency does as well. Not only are there more providers of critical infrastructure, but the infrastructure becomes a larger target, as spying can provide more information, disabling the

system will cause more people to be affected, and the effects of ransomware make it more likely for a large company to give in to demands. Additionally, more personnel involved means that fishing attacks are more likely to work, and insider threats are more likely to happen. Much of the current work on critical infrastructure involves risk analysis and preventative measures.

Some examples of this work include (Ouyang) which provides suggestions for modeling interconnected systems in critical infrastructure, as the failure of one system (such as electrical grid) could cause another failure (such as the internet). (Husnoo et al.) discusses privacy for IoT critical infrastructure. (Alcaraz and Zeadelly) provide general suggestions and discuss the modern challenges of critical infrastructure. NIST, the US National Institute of Science and Technology, provide a framework for improving cybersecurity for critical infrastructure (Barrett). In these guidelines they provide 5 core functions in regard to threats; these are identify, protect, detect, respond, and recover. This thesis focuses only on the penultimate tenet, though proper maintenance of critical infrastructure involves all of these. Additionally, there is much work that is closely related to the other guidelines that does not specifically refer to critical infrastructure, such as work that finds a widespread vulnerability, work on system privacy, or work that seeks to prevent or mitigate cyber-attacks.

2.2 Data Lineage

Data lineage (also called data provenance) is a technique used to trace dependencies of data items within a system. Often data lineage is used to trace errors or discrepancies, and in this thesis data lineage will be used to trace propagation of damage through the data items of a system. This information can also be gathered to assign responsibility to the entity that created the data item or learn how it was collected. As a result, data lineage information can also be used to assess quality and integrity of data. (Buneman and Tan) discuss the uses of provenance data in the context of databases. There are several techniques for data lineage.

One such technique is the W3C Open Provenance model, see (Khalid et al.). As this paper primarily focuses on the lineage of data items and their dependencies, the following is a brief discussion about how the W3C Open Provenance model would look at similar data. This model uses a graph to represent data lineage. The vertices in this graph can be data items, people, entities, or functions. When these things interact with a data item a new vertex is created with an edge from the entity modifying the data to the new vertex. Edges represent the action taken to create that edge. An example provided by the Open Provenance primer is shown in figure 1.

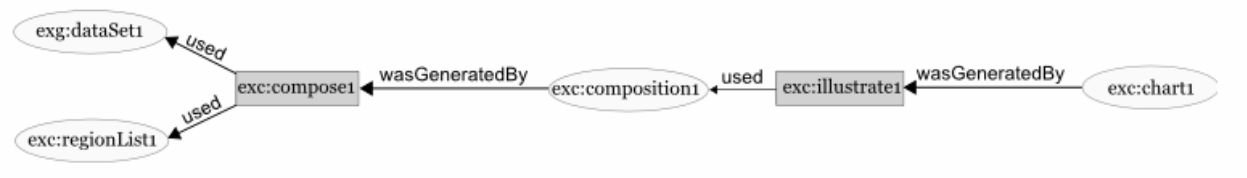

Fig. 1. *An example W3C open provenance graph (Belhajjeme et al.)*

In this way, directed edges point forward in time, and following an edge backwards shows their relation based on the edge. For example, in figure 1 dataSet1 used compose1. As a result of storing this information one can now see that dataset1 and regionlist1 are both dependent on chart1, and should it later be realized that chart1 was damaged both of these data items are likely to be damaged as well. Such damage tracing is the focus of the novel model provided in this thesis and further distinctions will be discussed later.

Work in data lineage includes increasing the efficiency of storage, see (Boa et al) and (Amsterdamer et al.). Storage costs can increase by a significant factor particularly when holding a large amount of metadata about each data item. This is particularly true in cases where data lineage needs to be traced all the way back to when the data item was created, or in cases where a large breadth of information about a data item needs to be stored. One technique used in (Anand et al.) does this by removing old copies of data items from the graph.

It’s also important in many cases to increase the speed of queries to allow provenance data to be used more efficiently, see (Karvounarakis et al). (Chapman et al.) focuses both on reducing the volume of data and increasing the query speed of lineage information. As data lineage has been around for some time now, other work suggests new techniques for data lineage, such as (Buneman and Tan) which proposes a technique for data lineage that uses blockchain. Much of this work, such as (Heinis and Alonso), is focused on maintaining data integrity, however it should be clear that this is not the only use for this information. (Zimmerman and Nagappan) and (Cao et al.) both use dependency graphs to assess and mitigate damage in a system.

2.3 Temporal Graphs

(Thulasiraman and Swamy 1) define a graph as follows. “A graph G = (V, E) consists of two sets: a finite set V of element called vertices and a finite set E of elements called edges. Each edge is identified with a pair of vertices. If the edges of a graph G are identified with ordered pairs of vertices, then G is called a directed or an oriented graph.” This paper makes use of directed graphs or digraphs. Further, these graphs are all temporal graphs. Temporal graphs are distinct in that the set of edges E, is now dependent on the time T. i.e., E(T) is a function of time that generates a set of edges, see (Othon). Whereas a typical or static graph may represent roads between cities, a temporal graph may be used to represent communications between satellites. Often satellites (the vertices in this example) will enter and exit the range of other satellites, causing a graph representing the communication capabilities of these satellites to be dependent on time.

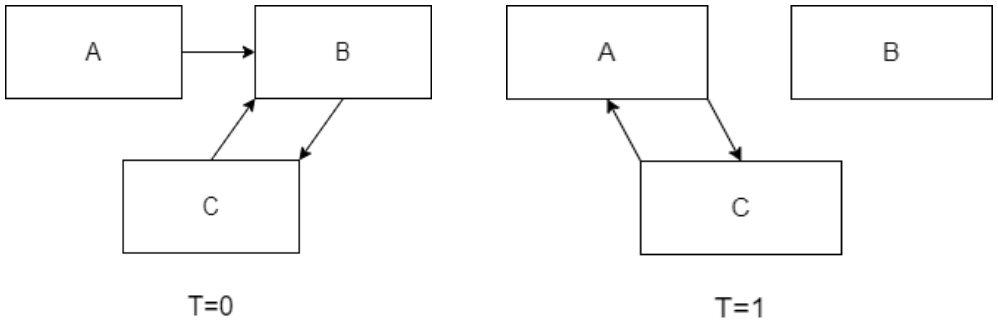

Fig. 2. *A temporal graph with distinct edges for two values of T*

Figure 2 visualizes a simple temporal graph at two different times, T=0 and T=1. This graph contains 3 vertices (A, B, and C) and 5 edges. Notably at a fixed time, a temporal graph can be viewed as a static graph. Should edges exist for continuous time frames rather than discrete time frames, one could still model this structure as a discrete temporal graph. This is done by separating these continuous times into discrete times at which the graph is distinct. As a result, all temporal graphs discussed are shown using discreet times.

Many of the problems for static graphs are also common problems when dealing with temporal graphs. Much of the work on temporal graphs seeks to solve these problems. (Huanhaun et al. “Path Problems”) discusses techniques used to solve path problems in the context of temporal graphs which along typical shortest path, also include earliest arrival, fastest, and latest-departure paths. The problem of earliest arrival for two vertices seeks to learn the earliest time a specific vertex can be reached from another. (Huanhuan et al. “2016 IEEE”) and (Enright et al.) both focus on reachability. The problem of reachability asks for the set of vertices that can be reached from the first and, for the purposes of this thesis, can be viewed as extension of earliest arrival in the case of temporal graphs. This thesis will discuss earliest arrival and reachability in a special case of temporal graphs.

3 THE DAMAGE ASSESSMENT MODEL

3.1 Definitions

Data Item - Throughout this thesis data item will be a catchall term for an amount of data that makes sense to be grouped. This can be a single bit, a file, all data on a computer, or data on a collection of computers depending on the specific use case.

System - A system will refer to some collection of computing resources, whether it's a process, a computer, or a collection of computers. Regarding data lineage information, system will refer to the entirety of computing resources whose data lineage information is being tracked.

Dependency - A dependency B→A (read from B to A) occurs when a data item A is dependent on data item B. For example, if A and B are integers setting A = B+1 creates a dependency B→A. As another example, in a system that calculates payroll there may be a dependency between the hours worked data item and the gross pay data item for each employee. Dependencies will be assumed to occur instantaneously.

Set - A set is a list that can contain no duplicates. If a set already contains an element, and that element is added again, the set is unchanged.

3.2 Dependency Graphs

The Damage Assessment Model is represented as a temporal graph consisting of vertices representing data items, and edges representing a dependency that occurred at a specific time.

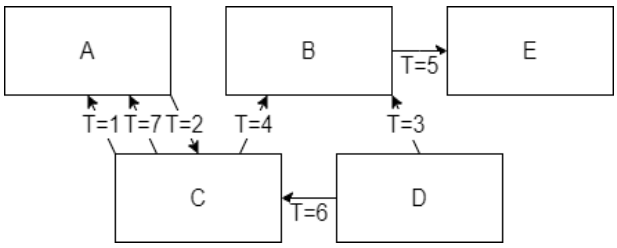

Fig. 3. *A temporal dependency graph*

Figure 3 shows a graph representative of 5 data items after 7 dependencies have occurred. This is a relatively simple dependency graph with only 5 vertices to be illustrative. Typical use cases likely have something in the range of one hundred to one billion vertices, but the general structure remains the same. In this case the vertices or nodes consist of the set V = {A, B, C, D, E}. The set of Edges E = {C→A at T=1, A→C at T=2, D→B at T=3, C→B at T=4, B→E at T=5, D→C at T=6, C→A at T=7}

Because the vertices represent data items, in general, discussing Vertex A and the data item it represents are distinct. However, throughout this section the vertex and its corresponding data item should be thought of as the same. For simplicity a dependency for the data item represented by vertex A to the data item represented by vertex B is synonymous with a dependency from A to B (shown as A→B).

The same edge can exist at different times (i.e., C→A exists at T=1 and T=7). Additionally, Edges can exist from a vertex to itself, but this does not assist in damage assessment as these edges would only matter when the vertex was already damaged. Such an edge could however assist when repairing but will largely be ignored for the purposes of the model.

Note that figure 3 represents the data in a temporal graph. This same data could be represented in one static graph for each discrete time that a dependency occurred. Also, because this is a temporal graph, if continuous times were used, they could be converted into

discrete times. Further, due to the data being represented one could always create a single static graph per edge. This is due to the fact that if two dependencies were to ever occur at the same time atomicity of operations requires that these two dependencies not affect each other. This prevents chains of dependencies occurring at the same time. For example, A→C and C→B cannot both occur at the same time because C was already being modified. Multiple edges can still occur at the same time, as long as no occurrence of such a chain exists. Edges A→C and B→C can still occur at the same time, as long as these dependencies are part of the same function to modify C.

One may now begin to notice some key differences between this representation and the W3C Open Provenance representation of data. First is that the Damage Assessment Model contains vertices of only data items and edges only of dependencies between them. As a result, the Damage Assessment Model cannot assess integrity of data sources as integrity information is not stored.

The Open Provenance model is acyclic. Acyclic in this case meaning the graph contains no cycles. A cycle exists in a directed graph when a vertex is reachable from itself. The Damage Assessment Model does not create a new vertex each time a dependency occurs, only a new edge. This means that the Damage Assessment Model is cyclic and as a result many algorithms for static graphs become significantly more complex. One such cycle can be seen in figure 3, at vertex C with edges C→A at T=1 and A→C at T=2. This simplicity comes with the advantage of significantly reducing the amount of required storage, and the Damage Assessment Model being temporal will later mitigate its cyclic nature.

3.3 Pay Calculation Example

Table 1

Information about an employee at an hourly job

Employee #	Hourly Pay in USD	Hours Worked	Taxes %
1	15	40	15

Say then the system calculates both gross and net pay. First gross pay is calculated as Hourly Pay * Hours Worked. Net pay is then calculated as Gross pay*(1-Taxes%/100).

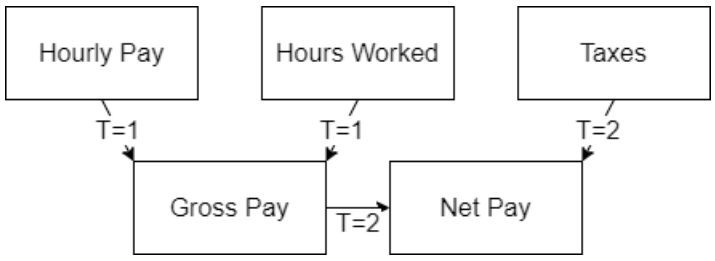

Fig. 4. *Dependency graph after operation on data in table 1*

The dependency graph of this system is now shown in figure 4. One important thing to note is that though the dependencies Hourly Pay→Gross Pay and Hours Worked→Gross Pay occurred at the same time, in terms of damage tracing either could be looked at first and the same result would be found. The same is true for the dependencies from Taxes→Net Pay and Gross Pay→Net Pay. As a result, figure 5 is identical in function to figure 4, at least in terms of damage assessment. This truth will be a key in the algorithm used to assess damage in the model.

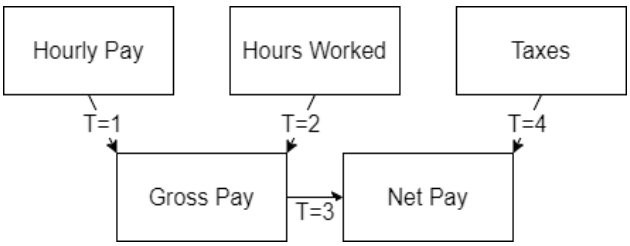

Fig. 5. *A dependency graph displaying damage found*

The simple justification for this is that if vertex C is dependent on vertices A and B at the same time, they could also be represented as edges at adjacent times. The full justification requires that a definition of damage be provided.

3.4 Propagation of Damage

In evaluating damage, a vertex is said to be damaged (or potentially damaged as the true state of the data item is unknown by the model) if it has a dependency on another damaged vertex. IE if Vertex X is damaged at time T and an edge exists X→Y after time T then Y is also damaged. In this case the edge from X→Y shows that Y is dependent on X. This means that, in some capacity, Y is used to create or modify the value of X. Thus, changing Y before this calculation can change the value of X. Note that this is not always the case.

Say at time T=1 X is set to a value of Y^2. This operation creates the edge Y→X at T=1. Say then, the value of Y is changed to (-Y) through damage at T=0. Though Y is now damaged, and X is dependent on Y, $X=Y^2=(-Y)^2$ so the value of X is unchanged with this knowledge. As a result, it's important to note that the damage assessment detects potential damage as opposed to actual damage. Due to this nature, further references to damage will refer to potential damage unless otherwise specified. By the nature of the model, only potential or probable damage is able to be found. After assessing damage further review is necessary to determine and repair actual damage.

3.5 Separation Of Edges

In the case of two vertices A and B modifying vertex C at the same time there are four possible cases shown by table 2.

Table 2

Potential damage in Vertex C in varying potential damage in Vertices A and B when they both have an edge to C at the same time

Vertex A	Vertex B	Vertex C
Undamaged	Undamaged	Undamaged
Undamaged	Damaged	Damaged
Damaged	Undamaged	Damaged
Damaged	Damaged	Damaged

We now compare this with evaluating the state of vertex C having separated the edges into different times. Without loss of generality, we evaluate damage at vertex C given the state of vertex A first. If vertex A is damaged then vertex C is damaged, otherwise it is currently undamaged. We then evaluate vertex C regarding vertex B. If C is already damaged it remains so, if not we check if B is damaged. If it is, we then say that C is damaged, otherwise C is undamaged. In the end we find that if A or B is damaged for adjacent edges to the same vertex, then vertex C is damaged yielding the same result as evaluating them at the same time. This same idea can be extended to any number of edges that occur at the same time, and it's found that the order in which edges to the same vertex are assessed does not change the result. This, along with atomicity of operations allows these edges to be separated into new discreet times without a change in the result after assessment is completed. Again, this separation will be

important later as the algorithm used for this evaluation requires that this separation be possible.

In terms of a graph, rather than individual dependencies, damage will start at a single vertex at a given time and will be traced until all potential damage is known. In general, this is the reachability problem from the damaged vertex. In damage assessment the goal is to find all damaged vertices while excluding as many undamaged vertices as possible. This is to minimize the number of vertices that need to be reviewed and repaired. More specifically, the desired result is the set that contains all damaged vertices that can be found by the model and nothing else.

3.6 Static Assessment

Due to the time of dependencies being tracked, all edges that occurred before the time of damage can be ignored. Approaching the data as a static graph (i.e. after removing edges before the time of damage and ignoring dependency times afterward) allows for a simple approach to damage assessment as well established algorithms for reachability can be used.

3.6.1 Static Assessment Examples

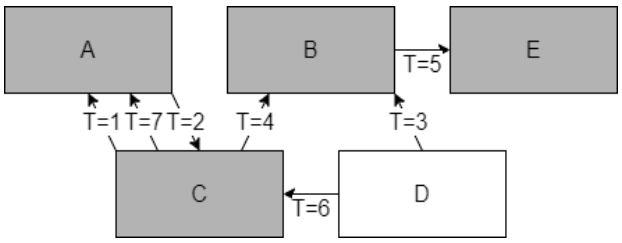

Fig. 6. *A dependency graph displaying damage found after assessment*

Starting at the damaged vertex, all outgoing edges that were created after time of damage are used and a breadth first search is done from the damaged vertex. All vertices reached are added to the list of damaged vertices.

Figure 6 shows one such damage assessment of figure 3 after Vertex C was found to be damaged at time T=3. All vertices that have been darkened are damaged. In this case only D is found to be undamaged. C has an edge to both A (using C→A at T=7) and B (using C→B at T=4), thus both are damaged and added to the list of damaged vertices. Finally, E is damaged because B is damaged, and assessment is complete. The set of damaged vertices V_d = {A, B, C, E}. We can now also use the time of damage for these vertices by looking at when their dependency occurred. C is known to be damaged at T=3, A is damaged at T=7 (as earlier edges are ignored), B is damaged at T=4, and E is damaged at T=5.

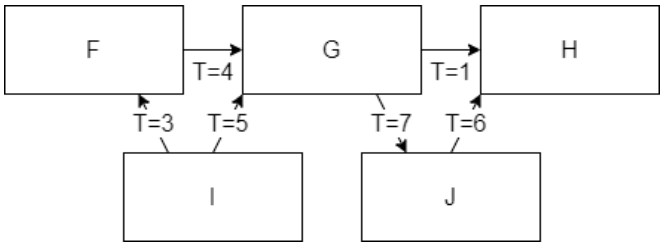

Fig. 7. *A dependency graph*

Figure 7 shows the dependency graph of another system. Say vertex I is damaged at time T=2. This system is now assessed in the same manner as before. Edges before T=2 are ignored, in this case G→H at T=1. We now see that both F and G are damaged by vertex I and both are added to the list of damaged vertices. J is then damaged by vertex G. Finally, H is damaged by vertex J. The set of damaged vertices V_d = {F, G, H, I, J}. However, when timestamps are accounted for, and time of damage is observed, one may notice several key discrepancies. The first is that vertex G was initially damaged by vertex F at T=4 rather than

vertex I. In doing this assessment, the earliest time of damage is sought as once a vertex is damaged it can begin to propagate that damage. For example, vertex J was damaged at T=7 by vertex G and was undamaged prior. This means that once timestamps are accounted for, vertex H should also be undamaged, due to the dependency J→H occurring before J was damaged. Thus, not only does reachability matter, but the earliest time a vertex can be reached from a damaged vertex matters as well. This is the earliest arrival problem from the damaged vertex and must also be solved to know if damage is propagated. In this case, the true set of damaged vertices is V_t= {F, G, I, J}, as assuming all dependencies are tracked vertex H cannot be damaged.

This approach of treating the graph as a static graph or static assessment is representative of the case where some data lineage is used, but the time of damage is ignored. This approach will later be compared to the approach of the Damage Assessment Model in the analysis and results section. Notice that static assessment can still eliminate some vertices from the scope of potential damage but is overall inefficient. This approach creates a superset of damaged vertices as some edges will be used in this approach that will not be used once time of damage starts being accounted for. Thus, more damage is found and as a result more damage needs to be manually reviewed, which can prolong downtime. Again, the goal is to find the minimum set containing all potentially damaged vertices.

3.7 Multiple Layers

Another crucial aspect of the Damage Assessment Model is that it operates in multiple layers. One important feature of the way data items have been defined is that it is possible for one data item to be a container for several smaller data items. This aspect is used to define these layers. Data items at each layer above the first represent a mutually exclusive group of data items at the layer below it. That is if data item B is contained in a higher layer data item A,

then A is the only data item at that layer which contains B. Data items at the bottom layer, or first layer, are referred to as granular. This is because data items at this layer could be viewed as partitions of higher layer data items, but there are no such partitions for data items at this layer. That is, data at the first layer is the smallest subdivision of data items in a system. If data item X is part of higher layer item Y, we will say that Y is the parent of X or that X is the child of Y.

3.7.1 Multiple Layer Example

Imagine the case of implementing the model for a database. This database contains a collection of tables, and each of these tables contain a collection of records. Each record contains a value for each field in the table. Then to implement the Damage Assessment Model one may decide that these values for fields are granular or first layer data items. Then second layer data items could be records, and third layer data items could be tables.

Another potential implementation of this structure could have files representing first layer data items, folders or collections of files as second layer data items, and individual computers representing third layer data items.

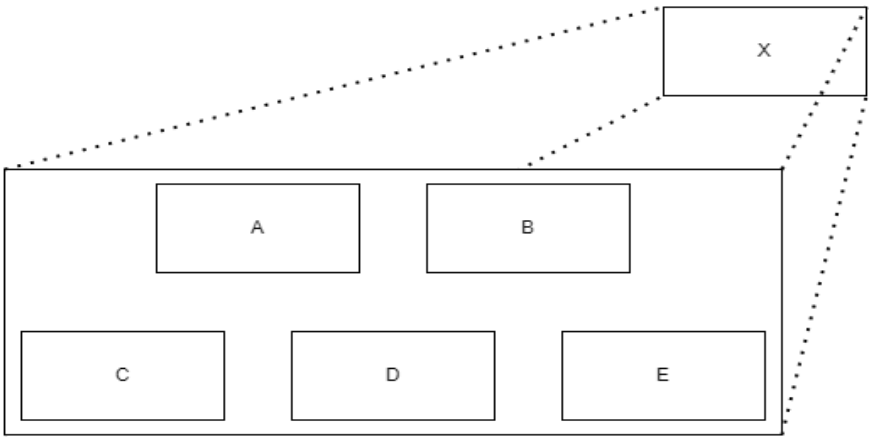

Fig. 8. *Visualization of Vertex X functioning as a container*

Figure 8 is a visualization of this concept. Data items A, B, C, D, and E are all on the same layer. Data item X exists at one layer above the others. X contains data items A, B, C, D, and E. If A, B, C, D, and E are integers, then X would be representative of a list of integers. If there was also a data item Y at the same layer as X, Y could not contain any of A, B, C, D, or E as the contents of X and Y are mutually exclusive.

This thesis focuses primarily on the case with 3 layers; however, this concept of layers can be extended into less or more depending on the particular use case. Key things to note are that all data items within a layer are defined at the same scope, and that this grouping should be chosen to closely represent the reality of the data.

This layering is done to allow partial damage assessment of a system. Each layer has its own dependency graph. It is at this point that some assumptions about the system need to be made. The first is that all operations that cause a dependency can be condensed into one or several edges between granular data items. The second is that higher layer data items consist only of lower layer data items. The last is that all dependencies that affect the system are tracked. If these assumptions were untrue, then there would be some dependencies or data items not tracked by the model, and as a result the model would be unable to represent and find all damaged data items.

When a dependency is created from one data item to another within a layer, if these data items are not part of the same data item in the layer above (rephrased these vertices do not share a parent vertex), then an edge must also be created from one parent to the other. This is the primary reason that the layered structure should represent reality as grouped data items should be more likely to have dependencies to one another. This limits the number of dependencies on higher layers and makes it more likely for some vertices to be found undamaged after assessment.

By creating these edges, in function, we now have a dependency graph for these higher layer items. When a data item X is found to be damaged, all higher layer data items that contain X are also found to be damaged and there will be exactly 1 such data item per layer. By assessing the damage at a higher layer, if a vertex is found undamaged, then both the vertex and its children will also be undamaged at all lower layers. Additionally, there will be less dependencies at higher layers allowing for faster assessment. Put together, this means that damage assessment at a higher layer has the potential to make portions of the system usable without propagating damage before the slower first layer assessment occurs. Also, because the graph of each layer is independent the assessment of each layer can be done in parallel, allowing higher layers to be assessed without slowing the overall assessment in comparison to an approach with a single layer.

3.8 The Damage Assessment Algorithm

With layers now accounted for, the model can be represented as several temporal graphs. More specifically, one temporal graph per layer. These graphs are temporal as they change over time, however they are unique in that no edges can be added. This is because, during assessment, the system is halted and cannot create new edges. Over time each of these graphs can only lose edges. Lose in this sense meaning that as T increases, less edges are able to propagate damage. Further, for reasons previously discussed, these graphs can be precisely represented as a series of static graphs each with a single edge. The algorithm will function by assessing if damage can be propagated by these single edges.

3.8.1 Algorithm for Damage Assessment within a layer

Input: A set of Vertices V, an ordered list of Edges E, a damaged Vertex D, and a time of damage T

Output: A set of Damaged vertices V_t

1. Insert D into V_t
2. For each Edge in E after time T
    a. If the independent vertex is in V_t
        i. Insert the dependent vertex into V_t
3. Return V_t

The set of vertices V will be all vertices within a layer. Also note that both V and V_t are sets and contain only unique items. The ordered list of Edges E consists of all edges that occurred within that layer. These edges consist of a dependent vertex and an independent vertex. In the edge A→B, A is independent, and B is dependent. Additionally, these edges are sorted chronologically. By sorting them, the operation on line 2 of the algorithm can function using only a binary search. Sorting the edges is little work as they should be close to sorted when stored immediately after a dependency occurred. Because the number of vertices plays no role in the complexity of the algorithm it functions in linear time with respect solely to the number of edges. If necessary or helpful, one could also track the earliest time each vertex was damaged by looking at the edge that added that vertex to the set of damage. This information could be useful in viewing logs for repairs after assessment.

3.9 Damage Assessment Example

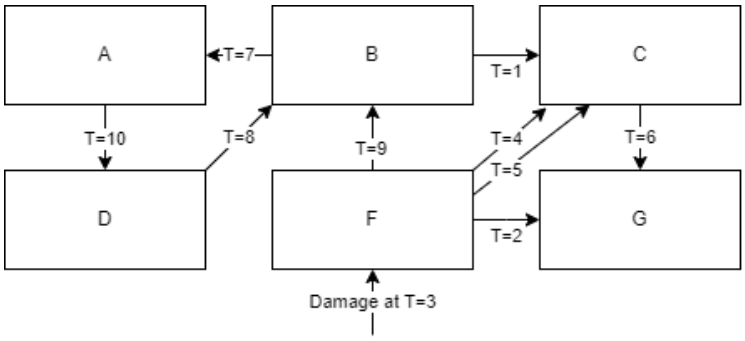

Fig. 9. *Dependency graph with damage at T = 3 to vertex F*

The system shown in figure 9 has vertices V = {A, B, C, D, F, G}. There are 9 edges recorded within the system (ignoring the edge showing damage). The sorted list of Edges E = (B→C, F→G, F→C, F→C, C→G, B→A, D→B, F→B, A→D). The set of damage is V_t = {F}. The time of Damage is at T=3, so after accounting for time our set of Edges becomes E' = (F→C, F→C, C→G, B→A, D→B, F→B, A→D). Edge F→C is now assessed, and because F is damaged vertex C is also damaged. V_t = {F, C}. F→C is assessed again, but C is already damaged. C→G is now assessed, C is damaged, therefore G is as well. V_t = {F, C, G}. B→A is assessed, but B is undamaged, so no damage is propagated. D→B is assessed, but again no damage can be propagated. F→B is assessed, and B is now marked as damaged. V_t = {F, C, G, B}. Finally, A→D is assessed, but no damage can be propagated. All damage has now been found and vertices F, C, G, and B need to be further examined and repaired. Additionally, all computing resources held by A and D can now be used (as long as access to damaged portions of the system is restricted) as they have been found undamaged.

This algorithm could be expanded to handle the case where multiple data items are damaged with only minor changes. Rather than inserting D into V_t on line 1, all items that are damaged need to be inserted into V_t. Before any edges are assessed, V_t must be the set of

initial damage at time T. This algorithm is then run per layer, and if possible, in parallel with all layers.

Once one layer has been assessed, all undamaged portions of that layer can again be used in operation without causing further damage. Additionally, because higher layers will have less edges and edges are the determining factor in runtime, assessment at higher layers is faster. In particular, the highest layers assessment should complete faster, allowing the largest partitions of the system to be freed first where possible. Then the same is done on lower layers. Once done, further review is required for all first layer data items marked as damaged and the system can be repaired.

When a vertex at a high layer is marked as damaged, its operation cannot resume until after assessment at the layer below. This is because once assessment at a layer is complete, all damaged items at that layer are known. For example, assessment shows second layer vertex X damaged. Now, once first layer assessment completes X no longer needs to be viewed as damaged as all damage to first layer items is known. X is a container of first layer items, and therefore as long as no damaged portions of X are used, X as a whole need not be viewed as damaged, particularly in cases where no data items in X are damaged.

3.10 A Similar Reachability Problem

Imagine a scenario where a salesman sits in an airport. The salesman is sure he has completed all the sales he is going to in this town and wishes to travel somewhere new to find customers. He decides that he only cares to go to a destination he can fly to. He wants to look at all destinations he can reach, and if possible, he wants to arrive at the earliest possible time.

If the salesman were to view airports as vertices and flights as edges, he now runs into the same special case of the temporal graph reachability problem. In the algorithm he can view the airport he resides in as the initial damage, and he begins to look at flights that start as soon

as he could reach a gate. One primary difference is that his edges have a travel time. This travel time necessitates he also keep track of when he arrives after a flight and can board another. Then if a flight from that airport happens after that time, then he can board it. If the algorithm would see a vertex as damaged, then that denotes an airport he can reach. Should he also keep track of arrival times, he can also know the earliest time he can reach any of them. He can now make an educated decision about where to go.

3.11 Suggestions for Implementation, Creation, and Storage of Model Data

Though vertices are representative of data items, the contents of these data items are irrelevant to the model. This means that in regard to the model very little data needs to be stored. As long as the ability remains to translate a vertex into its corresponding data item, vertices can be referred to by an integer. Another important note is that the only necessary metadata required for a vertex is whether or not the vertex is damaged. Thus, along with the understanding of which vertices correspond to which data items, all necessary storage can be condensed into a single file of bit flags, one per vertex. The position of the vertex in this file corresponds to its number, and the bit flag represents whether or not it has been marked as damaged. This file can be used to represent V_t and adding a vertex to V_t will simply mean flagging that vertex as damaged.

Edges, then, require 2 integers and a time stamp. For the binary search to function, these edges should all be of the same size. In general, this means that each edge stored requires $2*\log_2$(Total Vertices) bits to store references to both vertices along with a number of bits for the timestamp. Further, because they are already sorted during evaluation only one edge needs to be read at a time.

Whenever a dependency occurs, an edge must be created. In addition, if the vertices involved do not share a parent, an edge between the parent vertices must also be created and

so on until the top layer is reached and the vertices have no parents. Edges should be created chronologically and typically the creation of a new edge requires simply appending the edge to the list of edges.

Damage assessment requires all edges after time of damage. As a result, if the system is found to be consistent, then all current edges can be deleted as they will not affect assessment. However true consistency of a system is very difficult to determine, and one recommended practice is deleting edges with a timestamp that is older than a chosen time in order to prevent cases where edges relevant to assessment are deleted. Alternatively, these edges could function as a circular log in a fixed size file, where a new edge will simply overwrite the oldest edge. As this list is already sorted, finding which edge to replace can either be done with a binary search or by storing the location of the last write.

4 SIMULATION AND ANALYSIS OF RESULTS

Sample models were generated, and the algorithm was run to find how much of the system would be found damaged given different initial parameters. The amount of damage found by the model is compared with the total vertices and damage found by static assessment. For further detail on static assessment, this method is discussed in section 3.6.

To make the models function as they would in reality, whenever an edge is created at a higher layer, an edge is also created at lower layers between children of the vertices in that edge. This means that creating 1000 edges at the 2nd layer also creates 1000 at the 1st. Though such data could be operated on, the trivial case where a high layer data item has only 1 child is not considered. Thus, for all layers above the first, all vertices have at least 2 children. In order to reduce variance in the dataset, and have a more consistent representation of data, the first edge created is guaranteed to propagate damage. All tested graphs operate using 3 layers.

Additionally, edges created within their parameters are randomized. This causes the average model to perform worse than most real systems, as often more isolation between data items causes damage propagation to be less likely.

Variables studied include the maximum number of edges, maximum number of children, and top layer vertices for cases where the number of edges is varied based on a total, and where the number of edges is varied based on the number of vertices in a layer.

4.1 Number of Edges per Layer Vs. Number of Vertices at the First layer

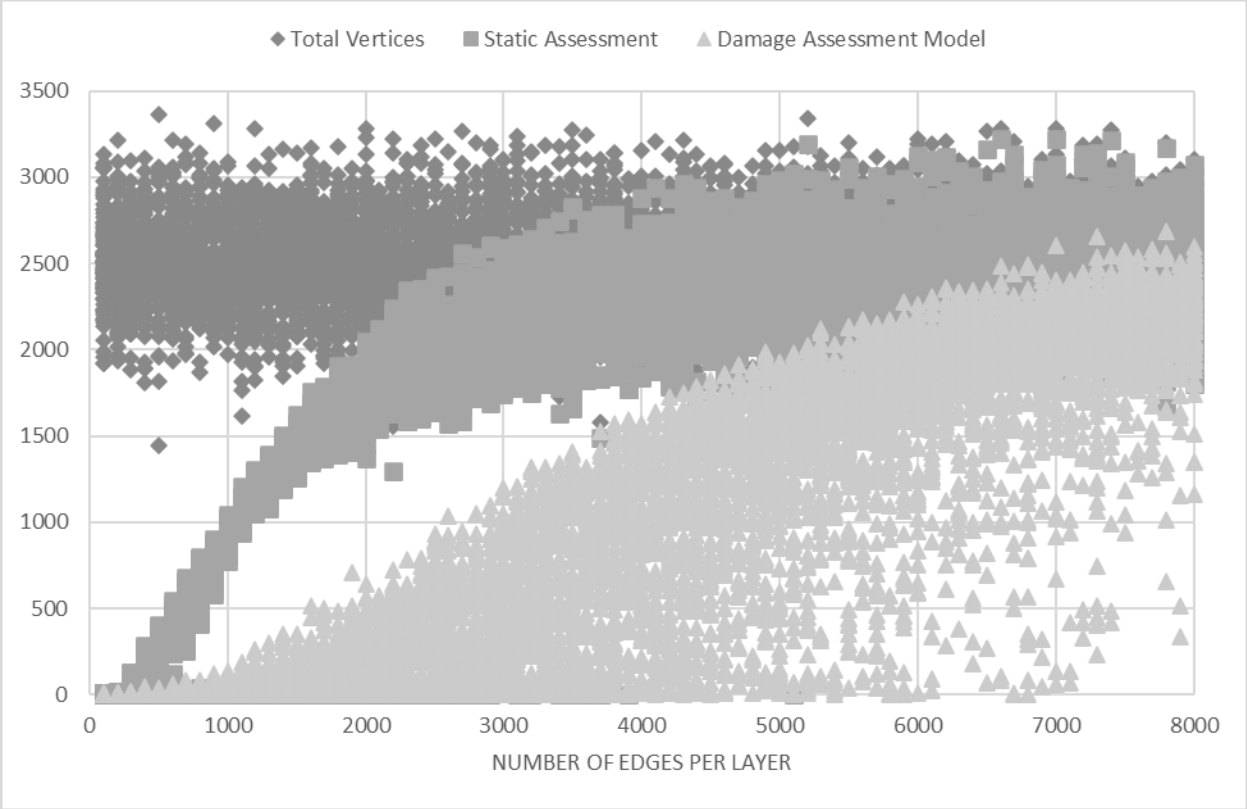

Fig. 10. *Scatterplot of all datapoints for generated models showing a comparison between total vertices and total damage found by static assessment and the Damage Assessment Model on the first layer*

100 models were created for each increment of 100 edges (per layer) with otherwise identical parameters. There can be a large amount of variance between 2 models with the same parameters, but overall trends can be seen through the average of these results. Damage was assessed at the first layer and compared to the total number of edges. Each model created had 25 3rd layer vertices and an average of 10 children for each lower layer.

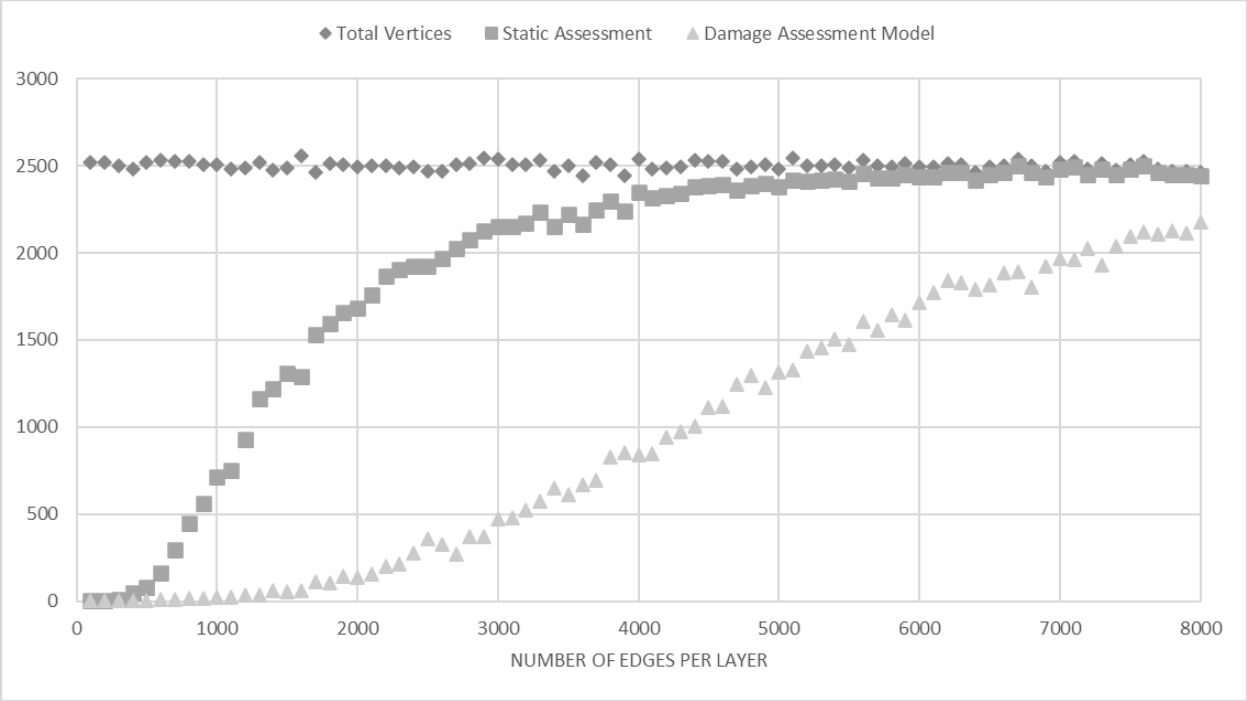

Fig. 11. *Scatterplot of averaged data for generated models showing a comparison between total vertices and total damage found by static assessment and the Damage Assessment Model on the first layer*

Figure 11 shows the average of data in figure 10. Predictably, total vertices remain approximately 2500 when averaged throughout the dataset.

Edges per layer represents the case where the total number of dependencies that occur strictly within one layer is similar to the number in other layers. As an example, in the case that the modeled system represents a database, 3rd layer vertices represent a table in a database and 2nd layer vertices represent records, the total number of dependencies between tables is the same as the sum of the number of dependencies that occur between records within the same table.

In this case when an edge is specified to be on a specific layer, that means that the vertices of the edge either share a parent or are at the top layer. As an example, if an edge is

created at the 2nd layer, both vertices share the same 3rd layer parent vertex. Additionally, a first layer edge is created between children of these vertices with respect to their order.

In reading the graph, *M* edges per layer means there are *M* edges generated on each layer. This means that *M* edges are generated at the 3rd layer by selecting 2 random 3rd layer vertices per edge. This creates *M* 1st and 2nd layer edges. Afterward, another *M* edges are generated at the 2nd layer by selecting 2 random children of a randomly selected 3rd layer vertex per edge. This does not create any further edges between 3rd layer vertices but does create another *M* edges at the first layer. Finally, *M* more edges are created on the first layer by selecting 2 random children of a randomly selected 2nd layer vertex per edge. Again, this creates no edges between higher layer vertices. Once all edges are created, they are shuffled randomly. In total *M* edges per layer represents 3**M* total edges for the first layer, 2**M* for the second layer, and *M* for the third layer. Under this structure, if it takes *N* time to assess *M* vertices, the third layer takes *N* time to assess, the second 2**N*, and the first 3**N* time.

There is a clear positive trend between the number of edges and the amount of damage found by both methods of assessment. Notably, the highest rate of growth in damage comes when approximately half of the vertices (in this case 1250) vertices are damaged for each assessment type. This can likely be attributed to the propagation of damage requiring an edge from a damaged vertex to an undamaged vertex and the inverse relationship between damaged and undamaged vertices. Intuitively, very little of the system is damaged initially so edges are unlikely to start at a damaged vertex, and once a majority of the system is damaged, edges are less likely to propagate damage to a vertex that was previously undamaged.

For the Damage Assessment Model, the first layer finds half of its vertices damaged at roughly 5000 edges per layer. Given that there are 2500 average vertices at this layer and 15000 total edges, there are on average 6 dependencies per vertex before half of the system is found damaged.

4.2 Number of Edges per Layer Vs. Number of Vertices at the Second Layer

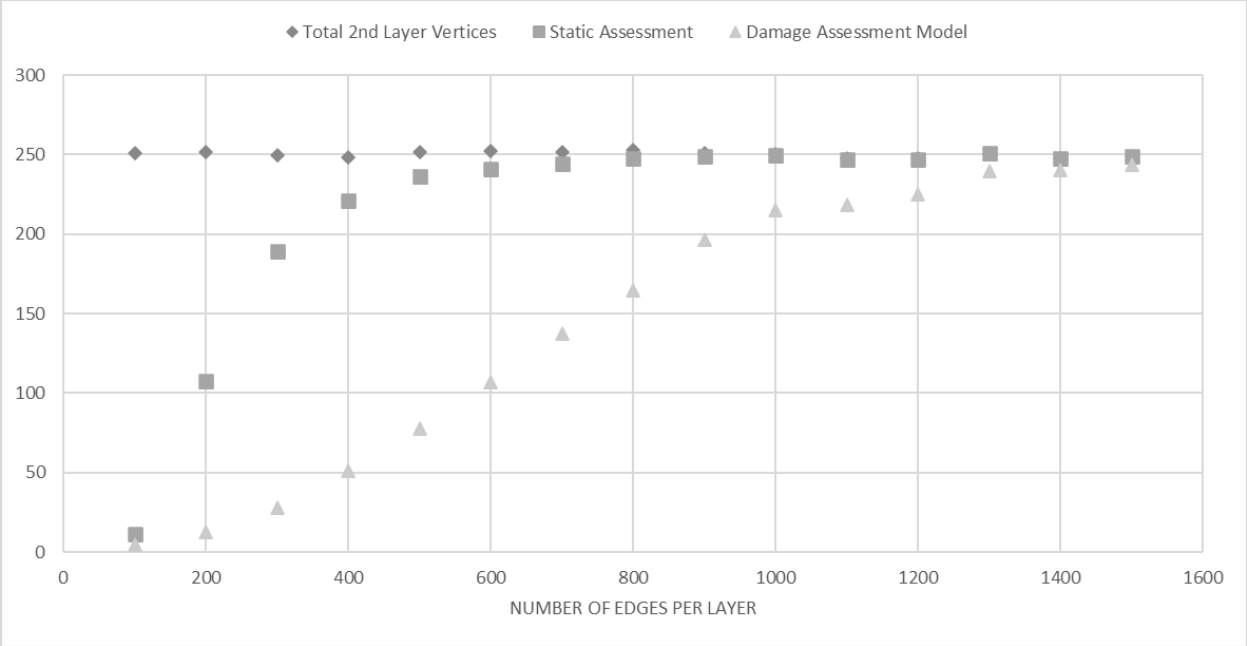

Fig. 12. *Scatterplot of averaged data for generated models showing a comparison between total vertices and total damage found by static assessment and the Damage Assessment Model on the second layer*

Figure 12 shows that same data at the second layer and similar growth patterns are seen here. In both layers, the Damage Assessment Model significantly outperforms static assessment in reducing the number of data items that require repair or validation. In this case, the Damage Assessment Model finds a similar amount of damage when given 3 times the number of edges when compared to static assessment.

For the Damage Assessment Model, the 2nd layer finds half of its vertices damaged at roughly 625 edges per layer. Given that there are 250 average vertices at this layer and 1250 total edges at this layer, there are on average 5 dependencies per vertex before half of the system is found damaged. This difference between the 2nd and the 1st layer could be caused by damage being more often propagated between children of the same data item. In order for an edge at the 1st layer where both vertices share a parent to spread damage, damage must

already be present in the first vertex, and not present in the second. Such edges represent a third of the total edges, and as a result some clustering of damage occurs where nearly all children of some 2nd layer data items are damaged, while for others nearly none of their children are. It seems likely that this clustering has a more profound effect at the 1st layer as clustering can occur at both the 2nd and 3rd layers.

4.3 Number of Edges per Vertex Vs. Number of Vertices at the First layer

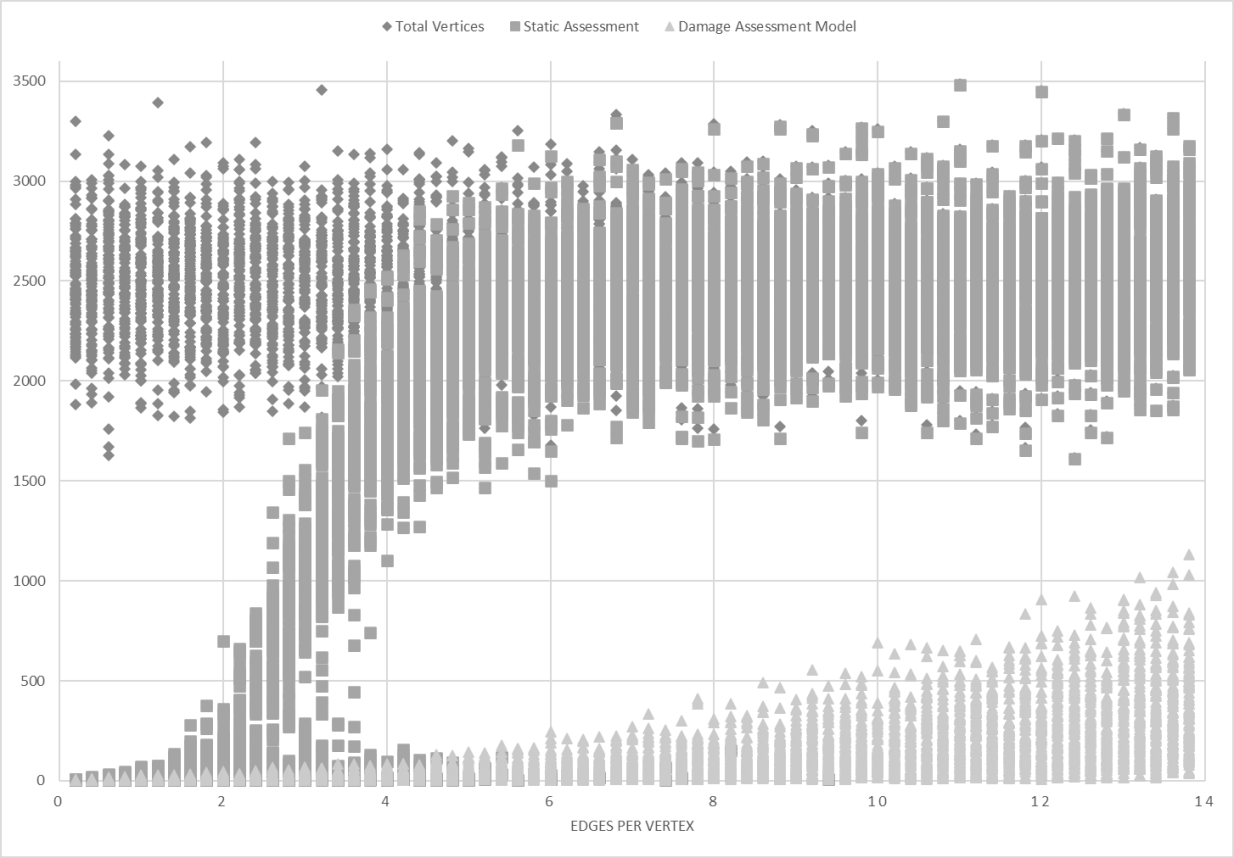

Fig. 13. *Scatterplot of all datapoints for generated models showing a comparison between total vertices and total damage found by static assessment and the Damage Assessment Model on the first layer*

100 Models were created with 25 top layer vertices and an average of 10 children for each increment of .2 edges per vertex. In contrast with edges per layer edges per vertex means that the number of edges on a given layer are directly proportional to the total number of vertices at that layer. 2 edges per vertex means that for a layer with 1000 vertices, there are 2000 edges at this layer. This method of creating edges represents the case where dependencies often share parent vertices as at any layer any 2 vertices are just as likely to share a dependency. Figure 17 can be directly contrasted with figure 10.

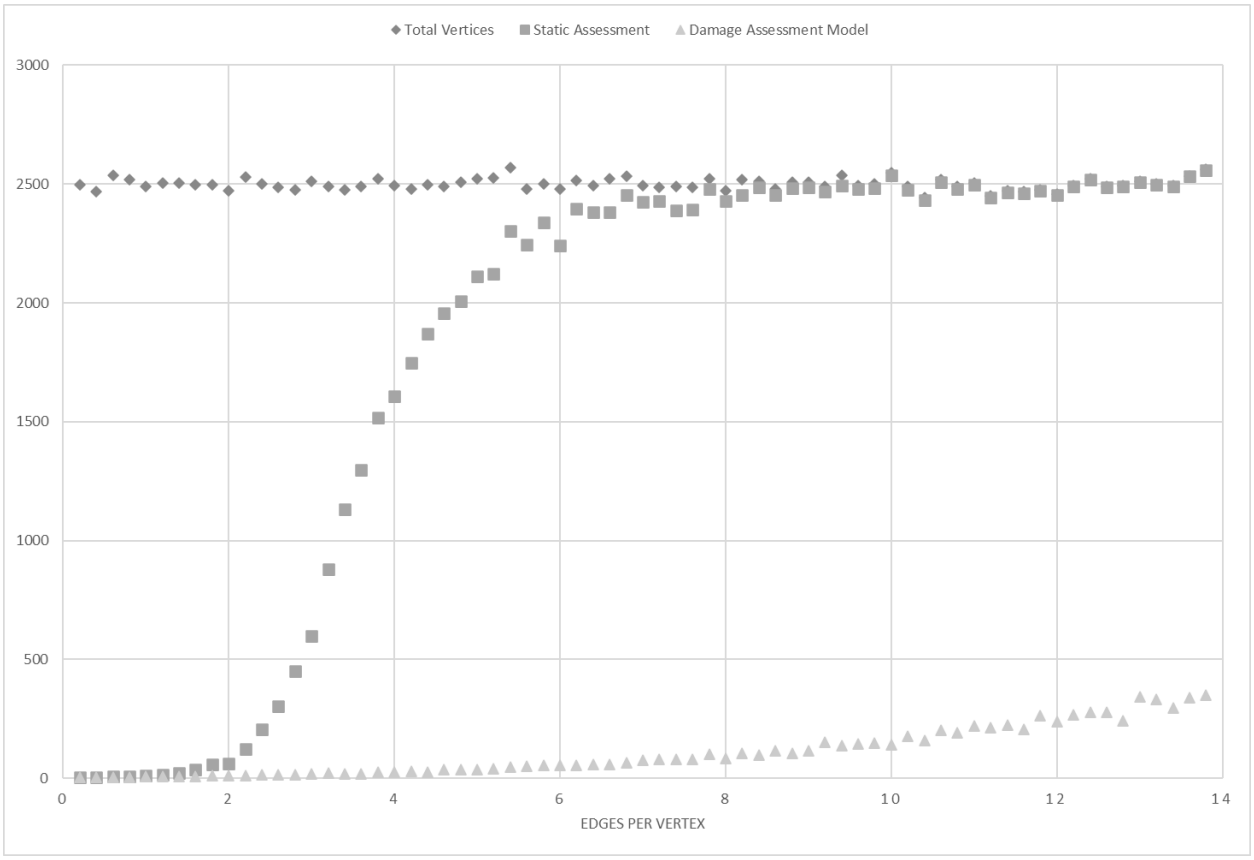

Fig. 14. *Scatterplot of averaged data for generated models showing a comparison between total vertices and total damage found by static assessment and the Damage Assessment Model on the first layer*

When averaged there is a clear increase in performance when compared to edges per layer. With edges per layer at least half of the vertices were found damaged by both assessment types at 6 edges per vertex. However, because there are far fewer edges at both the second and third layers, significantly more clustering is caused, reducing the likelihood of any edge propagating damage. Even at 14 edges per vertex the Damage Assessment Model has not found even 20% of the total system to be damaged.

4.4 Number of Top Layer Vertices Vs. Number of Vertices at the First Layer

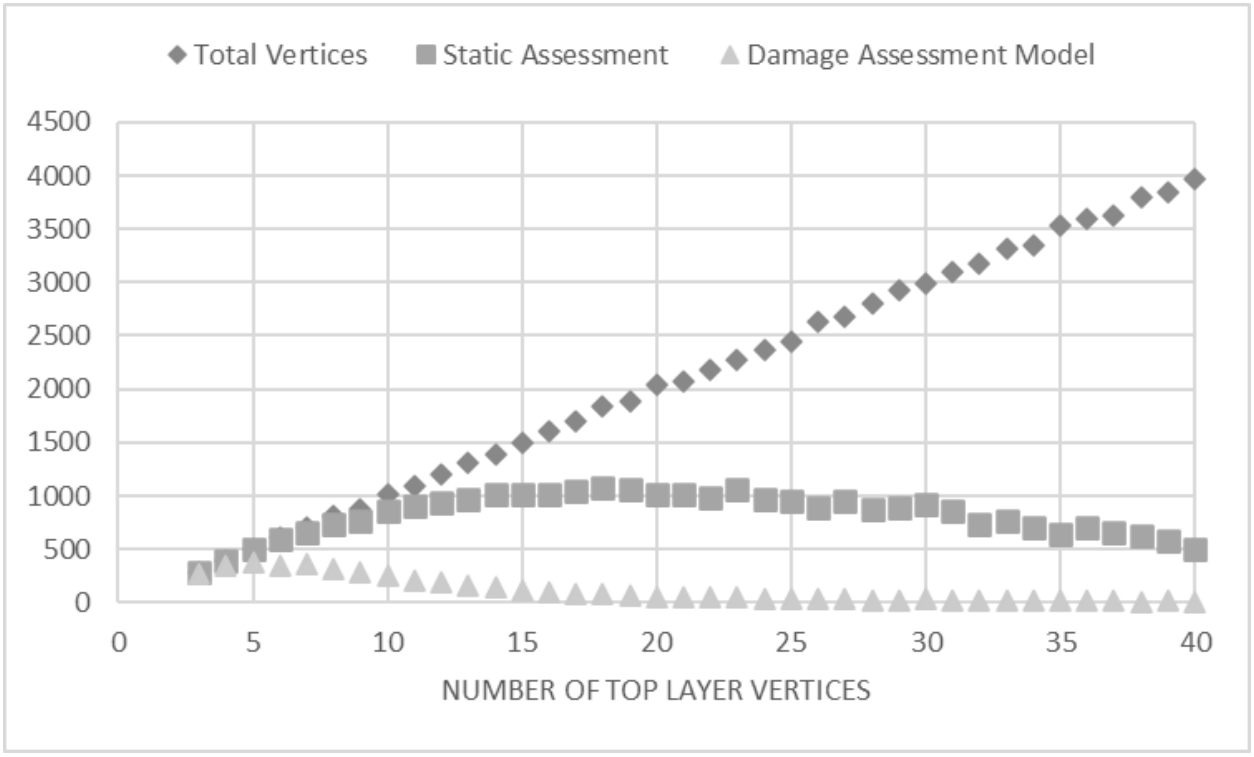

Fig. 15. *A comparison of total vertices and damage found using both types of assessment for differing numbers of top layer vertices averaged using 100 distinct generated models per data point*

Figure 15 shows total vertices and predicted damage for models with varying numbers of 3rd layer vertices, 10 average children, and 1200 total edges per layer. Increasing the number

of top layer vertices causes a linear increase in total vertices. Additionally, as a certain point adding more vertices causes less damage to be found using either type of assessment. When seen as a percentage of the total vertices, there is clear decline as more top layer nodes are added.

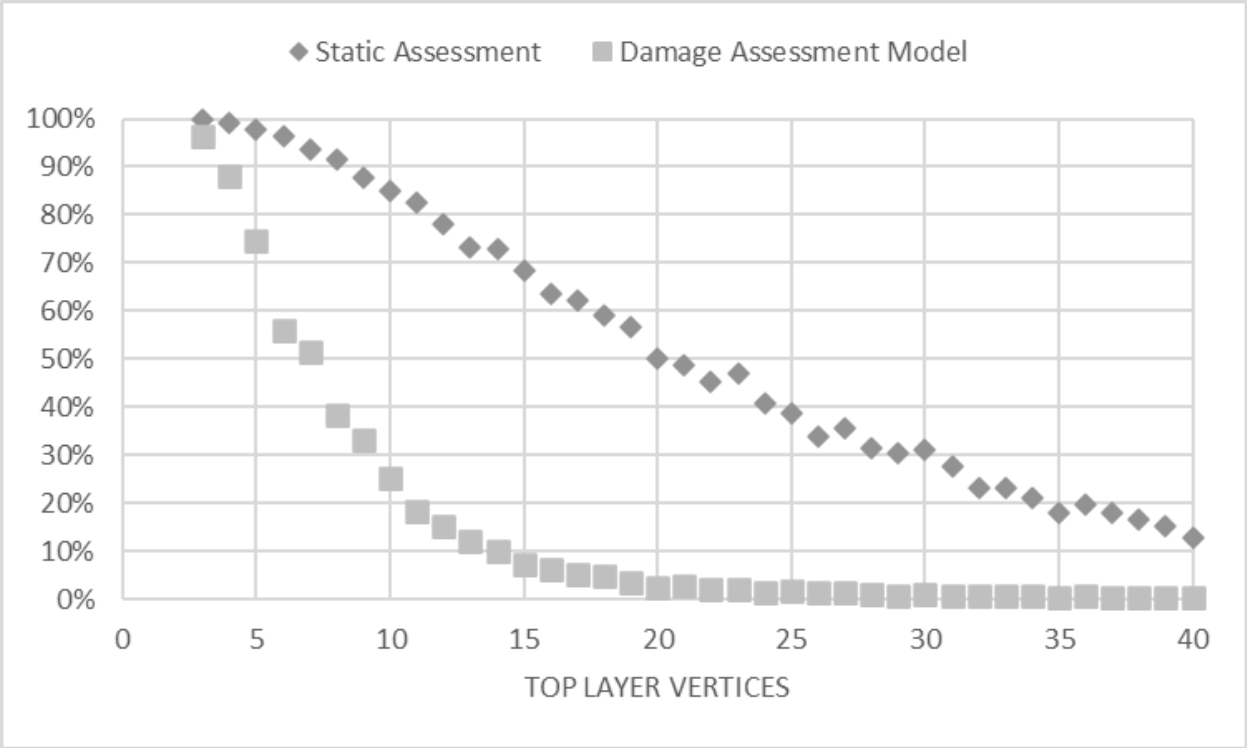

Fig. 16. *A comparison of damage found using both types of assessment as a percentage of total vertices for differing numbers of top layer vertices averaged using 100 distinct generated models per data point*

Figure 16 shows the percentage of total vertices found damaged by both assessment types. The decrease in amount of damage found by static assessment is nearly linear, while the Damage Assessment Model shows a much more rapid decline to 0.

4.5 Number of Children Vs. Number of Vertices at the First Layer

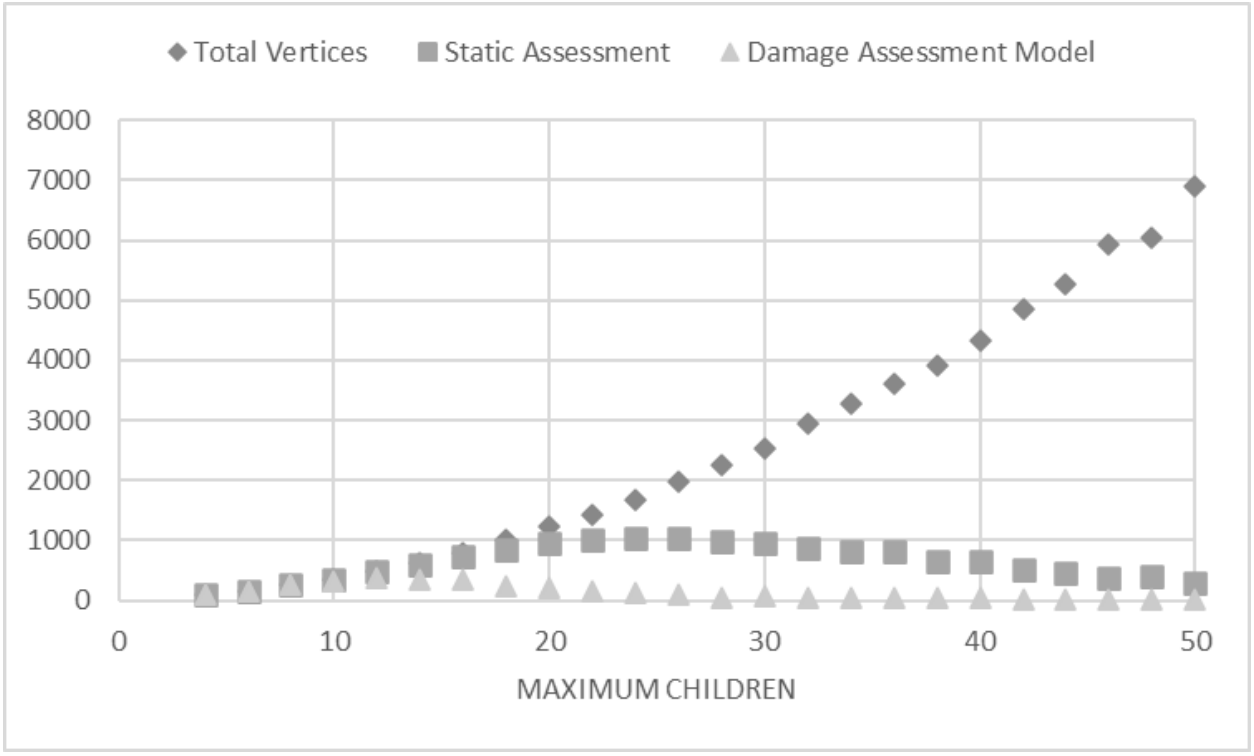

Fig. 17. *A comparison of total vertices and damage found using both types of assessment for differing numbers of maximum children for vertices averaged using 100 distinct generated models per data point*

Figure 17 shows how the maximum number of children can affect assessment and total vertices. For the models, each high layer vertex has between 2 and the maximum number of children. Each model has 10 vertices at the 3rd layer and 1200 edges per layer were assessed. The trends in both figure 15 and figure 17 are very similar, but increasing the maximum children causes an exponential increase in total vertices.

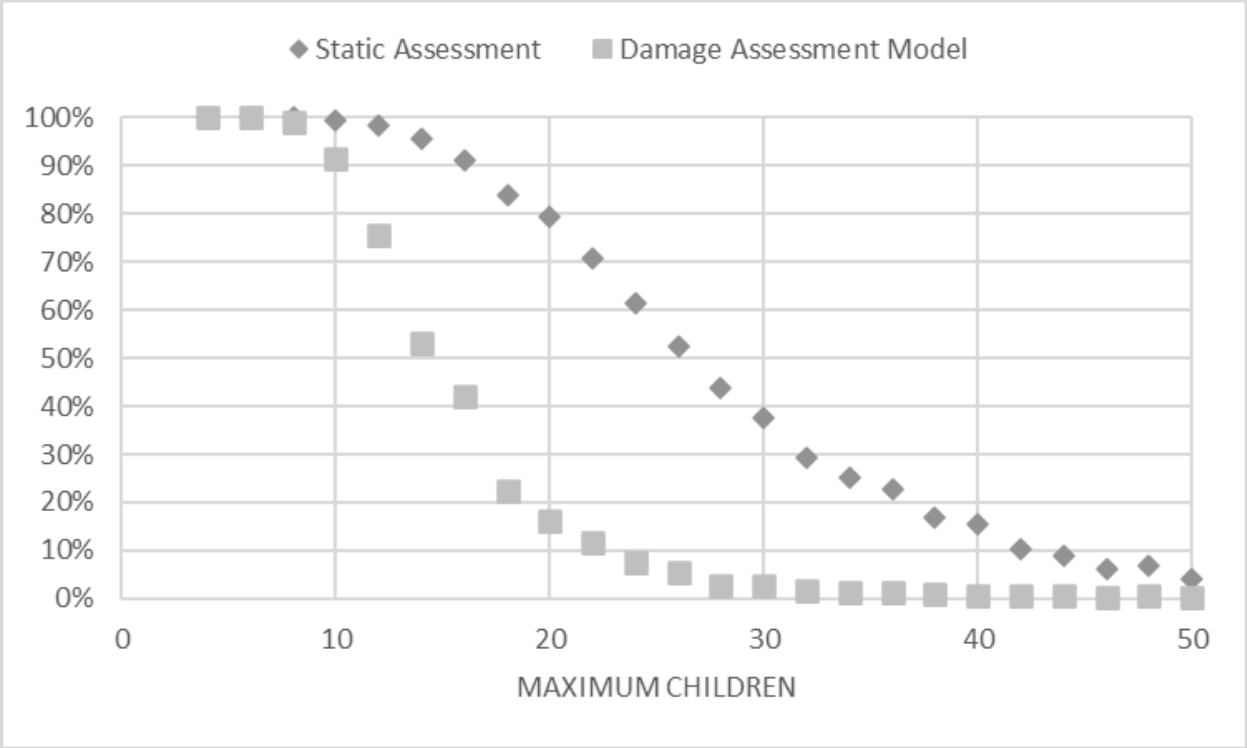

Fig. 18. *A comparison of damage found using both types of assessment as a percentage of total vertices for differing numbers of maximum children for vertices averaged using 100 distinct generated models per data point*

In Figure 18 this same damage is viewed as a percentage of total vertices. For both assessment types a clear s-curve is formed as damage becomes less likely to propagate with more vertices and a static number of edges.

4.6 Comparison of Total Damage Found at Different Layers

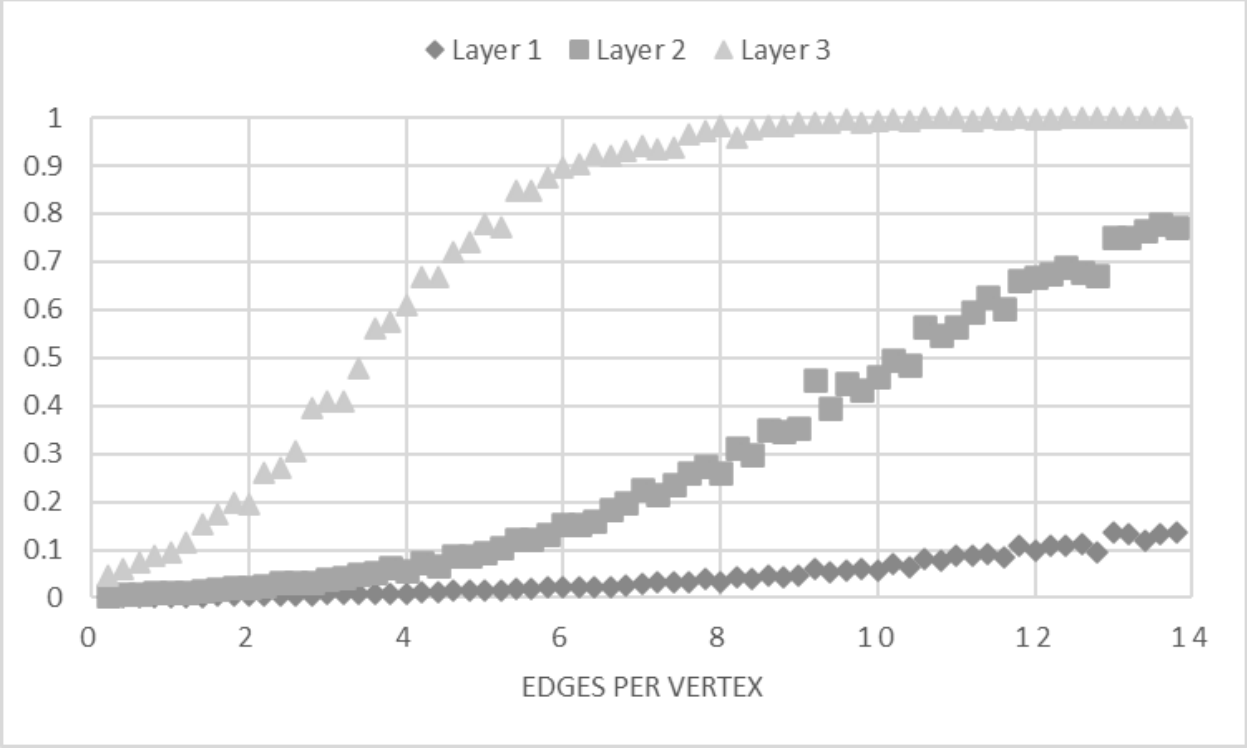

Fig. 19. *Total percentage of vertices found damaged by the Damage Assessment Model at different layers averaged using 100 distinct generated models per data point*

Figure 19 shows total system damage found after assessment at different layers of these same models with varying numbers of edges per vertex. Higher layers will always have more damage found as a total proportion of the system when assessed. As a result, higher layers begin to see rapid growth in found damage earlier, but assessment can still allow some of the system to resume operation. For any particular number of edges per vertex, the total amount of the system that can be freed after assessing each layer can be seen. For 6 edges per vertex a third layer assessment finds roughly 90% of the system damaged, so 10% can be immediately freed up. After a second layer assessment finds roughly 15% of the system then damaged, 85% of the system's resources can now be used. Given that assessment at each layer can be done

in parallel, assessment at higher layers is not a costly operation particularly in critical infrastructure.

5 CONCLUSIONS AND FUTURE WORK

Should damage occur within a system and propagate, a technique such as data lineage can drastically reduce the amount of data that needs to be checked for damage. This is particularly true for critical infrastructure where preventing any amount of downtime is crucial. By reviewing the time dependencies occurred between data items, searching for damage can become the temporal graph reachability problem. Additionally, when assessing, no new edges are added to the graph allowing separation of edges and assessment of static graphs with a single edge.

The Damage Assessment Model is one such temporal graph and operates in multiple layers which can assist in allowing some portions of the system to resume operation before the full assessment is complete. Assessment can be done on each layer in parallel, and it can be guaranteed that items found undamaged in a higher layer assessment will also be undamaged in the full assessment. Its primary advantages include speed and a predictable and consistent runtime.

The model and algorithm were tested and compared to a static graph approach of assessing the same information and significantly outperformed it. As more edges are created assessment takes linearly more time, and more damage is likely to be found. Growth of damage in terms of edges functions as an s-curve, seeing most rapid growth when half of the system is damaged.

Future work includes an implementation where high layer assessment is taken into account for lower layer assessment by removing edges that are from undamaged vertices in a higher layer. This could be done by merging edges at a lower layer into a single list as their shared parents are found damaged. However, with the current algorithm, unless a large portion of the system is isolated, this method will likely underperform.

6 WORKS CITED

Alcaraz, Cristina, and Sherali Zeadally. “Critical Infrastructure Protection: Requirements and Challenges for the 21st Century.” *International Journal of Critical Infrastructure Protection*, vol. 8, 2015, pp. 53–66. *ScienceDirect*, https://doi.org/https://doi.org/10.1016/j.ijcip.2014.12.002. Accessed 8 Apr. 2022.

Alkhadra, Rahaf, et al. “2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT).” IEEE Xplore, 2021, https://ieeexplore.ieee.org/abstract/document/9579611. Accessed 8 Apr. 2022.

Amsterdamer, Yael, et al. *On Provenance Minimization*, vol. 37, no. 4, 2012. *ACM Digital Library*, https://doi.org/10.1145/2389241.2389249. Accessed 10 Apr. 2022.

Anand, Manish Kumar, et al. “Efficient Provenance Storage over Nested Data Collections.” *Proceedings of the 12th International Conference on Extending Database Technology Advances in Database Technology*, ACM, New York, NY, 2009, pp. 958–969. *ACM Digital Library*, https://dl.acm.org/doi/10.1145/1516360.1516470. Accessed 8 Apr. 2022.

“Background Press Call by Senior Administration Officials on the Administration's Response to the Microsoft and SolarWinds Intrusions.” *The White House*, The United States Government, 31 Mar. 2021, https://www.whitehouse.gov/briefing-room/press-briefings/2021/03/12/background-press-call-by-senior-administration-officials-on-the-administrations-response-to-the-microsoft-and-solarwinds-intrusions/.

Bao, Zhifeng, et al. “Efficient Provenance Storage for Relational Queries.” *CIKM'12: The Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, The Association for Computing Machinery, New York, NY, 2012, pp. 1352–1361. *ACM Digital Library*, https://doi.org/10.1145/2396761.2398439. Accessed 10 Apr. 2022.

Barrett, Matthew. “Framework for Improving Critical Infrastructure Cybersecurity Version 1.1.” NIST Cybersecurity Framework, 2018.

Belhajjeme, Khalid, et al. “PROV Model Primer.” Edited by Yolanda Gil and Simon Miles, *Prov Model Primer*, 2013, https://www.w3.org/TR/prov-primer/.

Buneman, Peter, and Wang-Chiew Tan. “Provenance in Databases.” *SIGMOD 2007. Proceedings of the ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA: June 12-14, 2007, Beijing, China*, ACM, New York, NY, 2007, pp. 1171–1173. *ACM Digital Library*, https://dl.acm.org/doi/abs/10.1145/1247480.1247646?casa_token=r5ZCXlwVQxoAAAAA:ujzfZtb-mPZ_a-ooba_OOuynIh26HH_qUxAQjxoPsVGi87glaBzGGCmfWv_3yN5DIHsDVj6MRfhRqQ. Accessed 8 Apr. 2022.

Cao, Chen, et al. “Assessing Attack Impact on Business Processes by Interconnecting Attack Graphs and Entity Dependency Graphs.” *Data and Applications Security and Privacy XXXII 32nd Annual IFIP WG 11.3 Conference, DBSEC 2018, Bergamo, Italy, July 16-18, 2018, Proceedings*, edited by Florian Kerschbaum and Stefano Paraboschi, Springer International Publishing, 2018, pp. 330–348. *SpringerLink*, https://link.springer.com/chapter/10.1007/978-3-319-95729-6_21#citeas. Accessed 9 Apr. 2022.

Chapman, Adriane P., et al. “Efficient Provenance Storage.” *Proceedings of the 2008 ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA: 2008, Vancouver, Canada, June 09-12, 2008*, Association for Computing Machinery, New York, NY, 2008, pp. 993–1006. *ACL Digital Library*, https://dl.acm.org/doi/pdf/10.1145/1376616.1376715.

Enright , Jessica, et al. “Assigning Times to Minimise Reachability in Temporal Graphs.” *Journal of Computer and System Sciences*, vol. 115, 2021, pp. 169–186. *ScienceDirect*, https://doi.org/https://doi.org/10.1016/j.jcss.2020.08.001. Accessed 9 Apr. 2022.

Genge, Béla, and Christos Siaterlis. “Analysis of the Effects of Distributed Denial-of-Service Attacks on MPLS Networks.” *International Journal of Critical Infrastructure Protection*, vol. 6, no. 2, 2013, pp. 87–95. *ScienceDirect*, https://doi.org/https://doi.org/10.1016/j.ijcip.2013.04.001. Accessed 8 Apr. 2022.

Heinis, Thomas, and Gustavo Alonso. “Efficient Lineage Tracking for Scientific Workflows.” *Proceedings of the 2008 ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA: 2008, Vancouver, Canada, June 09-12, 2008*, Association for Computing Machinery, New York, NY, 2008, pp. 1007–1018. *ACM Digital Library*, https://dl.acm.org/doi/abs/10.1145/1376616.1376716. Accessed 8 Apr. 2022.

Husnoo, Muhammad Akbar, et al. “Differential Privacy for IoT-Enabled Critical Infrastructure: A Comprehensive Survey.” *IEEE Access*, vol. 9, 2021, pp. 153276–153304. *IEEE Xplore*, https://doi.org/10.1109/ACCESS.2021.3124309. Accessed 8 Apr. 2022.

“Infrastructure Security.” *INFRASTRUCTURE SECURITY*, Cybersecurity and Infrastructure Security Agency, https://www.cisa.gov/infrastructure-security.

Karvounarakis, Grigoris, et al. “Querying Data Provenance.” *Proceedings of the International Conference on Management of Data, SIGMOD '10, Indianapolis, IN, USA, 06-11.06.2010*, A.C.M., New York , NY, 2010, pp. 951–962. *ACM Digital Library*, https://doi.org/10.1145/1807167.1807269. Accessed 10 Apr. 2022.

Langner, Ralph. *IEEE Security Privacy*, vol. 9, no. 3, 2011, pp. 49–51. *IEEE Xplore*, https://doi.org/10.1109/MSP.2011.67. Accessed 8 Apr. 2022.

Liang, Xueping, et al. “2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID).” 13 July 2017, https://ieeexplore.ieee.org/abstract/document/7973733. Accessed 8 Apr. 2022.

Michail, Othon. “An Introduction to Temporal Graphs: An Algorithmic Perspective*.” *Internet Mathematics*, vol. 12, no. 4, 2016, pp. 239–280., https://doi.org/10.1080/15427951.2016.1177801.

Ouyang, Min. “Review on Modeling and Simulation of Interdependent Critical Infrastructure Systems.” *Reliability Engineering & System Safety*, vol. 121, 2014, pp. 43–60. *ScienceDirect*, https://doi.org/https://doi.org/10.1016/j.ress.2013.06.040. Accessed 8 Apr. 2022.

Thulasiraman, K., and M.N.S. Swamy. *Graphs: Theory and Algorithms*. Wiley.

Wu, Huanhuan, et al. “2016 IEEE 32nd International Conference on Data Engineering (ICDE).” IEEE Xplore, 23 June 2016, https://ieeexplore.ieee.org/abstract/document/7498236?casa_token=zfuUJXByjcYAAAAA:Qpj8b2JETTOrvaDir1Cth48mQ7vd0Ep0-Sp0ASNIjqeVWz1OphezNjO2k5HZuWQFkxqHTj8KzIE. Accessed 8 Apr. 2022.

Wu, Huanhuan, et al. “Path Problems in Temporal Graphs.” *Proc. VLDB Endow.*, vol. 7, no. 9, May 2014, pp. 721–732. *ACM Digital Library*, https://doi.org/10.14778/2732939.2732945. Accessed 8 Apr. 2022.

Zimmermann, Thomas, and Nachiappan Nagappan. “Predicting Defects Using Network Analysis on Dependency Graphs.” *Proceedings of the 30th International Conference on Software Engineering*, Association for Computing Machinery, New York, NY, 2008, pp. 531–540. *ACM Digital Library*, https://doi.org/10.1145/1368088.1368161. Accessed 9 Apr. 2022.