Exploring Agent-Based Simulations in Political Science Using Aggregate Temporal Graphs

R. Jordan Crouser* Tufts University
Jeremy G. Freeman † Tufts University
Andrew Winslow ‡ Tufts University
Remco Chang§ Tufts University

ABSTRACT

Agent-based simulation has become a key technique for modeling and simulating dynamic, complicated behaviors in social and behavioral sciences. As these simulations become more complex, they generate an increasingly large amount of data. Lacking the appropriate tools and support, it has become difficult for social scientists to interpret and analyze the results of these simulations. In this paper, we introduce the Aggregate Temporal Graph (ATG), a graph formulation that can be used to capture complex relationships between discrete simulation states in time. Using this formulation, we can assist social scientists in identifying critical simulation states by examining graph substructures. In particular, we define the concept of a *Gateway* and its inverse, a *Terminal*, which capture the relationships between pivotal states in the simulation and their inevitable outcomes. We propose two real-time computable algorithms to identify these relationships and provide a proof of correctness, complexity analysis, and empirical run-time analysis. We demonstrate the use of these algorithms on a large-scale social science simulation of political power and violence in present-day Thailand, and discuss broader applications of the ATG and associated algorithms in other domains such as analytic provenance.

Index Terms: G.2.2 [Discrete Mathematics]: Graph Theory—Graph algorithms; H.1.2 [Information Systems]: User/Machine Systems—Human information processing;

1 INTRODUCTION

Modeling and simulating dynamic, complicated behaviors is a key component of research in the social and behavioral sciences. In recent years, stochastic agent-based simulation has become a vital technique for modeling and exploring these behaviors [20]. As these simulations become more fine-grained and complex, they continue to generate increasingly large data sets which must later be analyzed. A complete exploration of these simulation results could help social and political scientists understand complicated social behaviors, find patterns of violence and socioeconomic repression, predict catastrophic events and better inform the decision-makers who influence global policy. Unfortunately, the existing methods and tools available to social scientists for analyzing these results scale poorly to large simulations, making it difficult to effectively interpret and analyze the results of these simulations.

Challenges facing social scientists in utilizing large-scale agent-based simulations to model complex behaviors are not limited to unmanageable data size and dimensionality. Because these simulations are stochastically seeded and run hundreds or thousands of times, it is also important to social scientists to be able to compare simulated behaviors between and across distinct runs, and to be able to piece together many simulation runs into a single, cohesive view. Due to the stochastic nature of these simulations, it may be impossible to predict how many runs are necessary to fully capture the behavior being simulated, and additional simulation runs are often expensive, sometimes taking hours to days to complete. Because of this, it is also important to be able to perform analysis on the fly.

In this paper, we begin to address these needs by developing a new theoretical framework and visualization techniques for analyzing complex dynamic behaviors. We begin by defining the Aggregate Temporal Graph (ATG). In this graph, a vertex represents a unique configuration state in the simulation, and a directed edge denotes a temporal transition between two states. We construct a directed graph by aggregating a large set of sample sequences from the agent-based simulation. Each of these sequences can be thought of as a pathway through the "simulation space", the set of reachable configurations in the high-dimensional parameter space. By aggregating these sequences and combining their common states into shared vertices, we are able to reconstruct the overall shape of the simulation space. This formulation provides a compact representation of the high-dimensional data space that preserves temporal relationships between simulation states. It is generalizable across various domains and is both structurally and developmentally modular: analysts can drill down to focus on specific components or subgraphs of interest, and analysis can begin at any point in the development of the ATG.

Framing the analysis of these data sets as an ATG problem, we can apply novel graph analysis techniques to the constructed graph to detect structural and transitional features such as points of high revisitation, repeated substructures, reachable subgraphs and anomalous paths. These features can be mapped to semantic meaning, providing a novel way of analyzing complex dynamic behavior in various contexts. In this paper, we focus on structures that illuminate causal relationships between states in the simulation. Because the simulation may still be running live as the analysis begins or the analyst may choose to modify the graph to focus on interesting features, the topology of the graph at any point in the analysis is uncertain, rendering effective precomputation impossible.

In order to facilitate the analytical process even as the topology of the graph is changing, we develop provably efficient algorithms for identifying interesting graph substructures on the fly. In addition to providing a formal theoretical analysis, we demonstrate the use of these algorithms in a real-time, fully functional prototype visual analytics system on an ATG-encoding of a large political science simulation and discuss implications for future work.

This work makes several important contributions toward interactive, real-time analysis of massive agent-based simulations.

1. We present the Aggregate Temporal Graph (ATG) structure that efficiently encodes temporal relationships between simulation states using a directed graph structure. We also identify novel substructures whose identification within an ATG structure has powerful cross-disciplinary semantic implications.

2. We pose efficient algorithms that can be used to identify these substructures and provide formal proofs of correctness as well as worst-case bounds.

*e-mail: rcrous01@cs.tufts.edu
†e-mail: jeremy.freeman@gmail.com
‡e-mail: awinslow@cs.tufts.edu
§e-mail: remco@cs.tufts.edu

3. We demonstrate the utility of the ATG framework and presented algorithms for real-time analysis.

To evaluate the ATG formulation and associated algorithms, we implement a functional prototype utilizing an existing graph visualization system and perform an empirical evaluation on the results of a real-world 1000-run simulation on political violence and power systems in present-day Thailand. We formulate the simulation data as an ATG containing 14,887 unique vertices and 59,000 edges, and we demonstrate the analytical process using this structure. Our findings provide strong evidence for the utility of the ATG framework for real-time analysis of massive simulation datasets.

2 Agent-Based Simulations as an ATG

Current research in the social and behavioral sciences relies heavily on stochastic simulation methods for modeling complex dynamic problems in political theory [20, 22], evolutionary biology [25], and criminology [9]. In order to capture the intricate nature of these real-world phenomena and produce a viable simulation, these models often contain thousands of interacting agents, each with hundreds of variables controlling its behaviors and relationships. Because they are stochastic rather than deterministic, each of these simulations must be run many times over the same data. As simulation methods become more and more powerful, the required running time and volume of generated data grows dramatically. As such, there exists a rapidly growing need for better analytical methods for making sense of the results generated by these simulations.

Because of the large volume of data generated and the inherent temporal nature of these simulations, agent-based simulation results are an ideal candidate for representation as an ATG. As agent-based simulations are discrete, transitions between states during each timestep can be represented by an edge between vertices in the ATG that represent these states. After representing the simulation space as an ATG, we can apply both well-studied and novel graph algorithms to help analysts locate meaningful underlying structures. For example, it is of interest to political scientists to be able to detect critical decision points preceding either desirable or undesirable states. In the latter case, the military terminology often used to describe these decision points is "the left of boom", wherein there is still an opportunity to prevent catastrophe if the appropriate actions are taken. Analysts are similarly interested in being able to detect inevitable (or highly likely) events given some intelligence describing a starting point in the simulation space. With the ability to easily detect unavoidable events in a well-described simulation space, analysts could better inform policy-makers regarding the consequences of a set of actions and better prepare for events that are highly likely to occur in the real world.

In this paper, we propose two algorithms to identify these structures and provide proof of correctness, complexity analysis, and empirical run-time analysis. We demonstrate the use of these algorithms on a large-scale social science simulation of political power and violence in present-day Thailand. Finally, we discuss applications of the ATG framework and algorithms in domains such as web analytics and analytic provenance.

3 Related Work

This project extends existing work on agent-based simulation in the social and behavioral sciences, as well as work on graph theoretic techniques for use in visual analytics.

3.1 Agent-Based Simulation in Social and Behavioral Science

Agent-based simulation is a key technique for modeling complex dynamic systems in political science, cognitive science, and other social and behavioral sciences. Increasing computing power means that systems of increasing complexity can be simulated. For example, agent-based simulation is widely used in political science to model collaboration [2], conflict [28], violence [4], and population change [3]. Agent-based simulations have also been used to identify a country's political patterns, which might indicate the imminence of civil unrest and help predict catastrophic events [21]. Such prediction requires models of a complexity far greater than those widely used to test single political theories. Lacking the tools to thoroughly analyze these results, predicting upcoming political structures using agent-based simulation is imprecise at best.

Similarly, there are gaps in computational modeling for biological and cognitive simulations. For example, agent-based simulation systems like NetLogo [29], SWAGES [26], Mace3j [10], and RePast [6] have been used to explore large parameter spaces of various cognitive and biological models. In the large-scale simulations run in these systems, even a relatively simple agent-based model can produce so much data that it cannot be stored in a regular file system. Instead, they require a more efficient integrated data management system that supports parallel insertions into and queries of the database or the utilization of distributed computation methods. While tools such as MDSViz and SocialViz [7] provide critical first steps toward robust visual analytics tools for this domain, there remains a critical need for tools to support real-time analysis and iteratively improve agent-based models.

3.2 Graphs in Visual Analytics

Visualization of graph structures is a well-studied area in computational science. For a survey see work by Herman et al. [13]. However, the study of graphs in visual analytics has primarily been restricted to the study of entity relationships. There has been much interest in tools for visualizing these relationships in the context of social networks [12, 14, 32], protein interaction [1, 8, 15], and more. There has also been recent interest in visualizing entity-relationship graphs with a temporal component in the area of information propagation through a network [11, 17, 18, 19] While these graphs do indeed encode a temporal component of the data they represent, they differ from the aggregate structure we propose in this work in that they represent the temporal transitions between a known set of static entities, rather than simulation states being explored.

Our work is also informed by contributions in the area of state space visualization, such as those made by Van Ham et al. [30, 31], Pretorius et al. [24, 23], and Blaas et al. [5]. Recent work in this area explored the visualization of large state spaces, where traditional node-link diagrams become quickly over-cluttered and fail to support the analyst in generating insight. As the state space increases, computational support during the visual exploration process becomes increasingly more important. Exploring massive simulation spaces involves similar data-density bounds and visualization challenges. To our knowledge, there exists no research on the utilization of graph structures for aggregated temporal simulation data in the area of political science.

4 Finding Meaning through Structure

An Aggregate Temporal Graph (ATG) is a directed graph constructed by aggregating a large set of sample sequences. In this graph, a vertex represents a unique configuration state in the simulation, and a directed edge denotes a temporal transition between two states. In formulating the analysis of agent-based simulation results as an ATG, we encode critical information about both the transitions between individual simulation states as well as the relationships between simulation runs. Consequently, substructures in the graph can be mapped to semantically meaningful relationships between simulation states. By identifying underlying structures in the ATG, we can assist domain experts in uncovering key and potentially subtle patterns in their simulations. In the following sections, we present two novel substructures that map to state relationships of interest to domain experts in social and political science.

4.1 Gateways

Agent-based simulations in political and social science are often used to simulate the conditions surrounding desirable or undesirable situations. In either case, it is of great interest to political scientists to be able to detect pathways of vertices that inevitably lead into these situations, and to identify critical decision points preceding these paths. In this section, we define the concept of a *Gateway* that captures this relationship between vertices.

Definition 1. *A* maximal path of vertex v *is a simple (non-intersecting) path ending at a vertex r, such that either: 1. r has no out-edges, or 2. there exists an edge (r,s), with s a vertex in the path.*

Given an ATG $G = (V,E)$ and vertex set $V' \subseteq V$, we consider the set of maximal paths of vertex $v \in V$ and whether every such maximal path intersects V' (see Fig. 1):

Definition 2. *A vertex v is a Gateway* to a set of vertices V' *if v has at least one outgoing edge and every maximal path starting at v contains some vertex in V'.*

In the special case that V' consists of a single vertex r, every *Gateway* v to r is a vertex whose maximal paths share a common set of vertices containing r, i.e. every maximal path of v contains r.

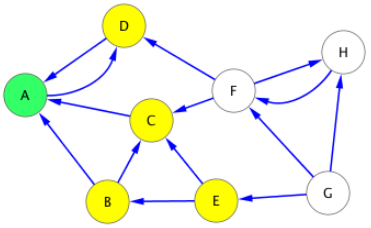

Figure 1: Each of the yellow vertices is a *Gateway* to the vertex set $\{A\}$. That is, every maximal path leaving a yellow vertex contains A.

To identify the set of *Gateways* to the input vertex set, we use an iterative set-building algorithm. By repeatedly growing the set of known *Gateways* rather than checking the outgoing paths of every vertex from which the input set is reachable, we greatly reduce the average computation time. We start by initializing the set S of *Gateways* equal to the input set (Fig. 2L), as each vertex contained in this set is trivially a *Gateway* to the set (see Definition 2). We then iteratively grow S by traversing each untraversed edge coming into the set of known *Gateways* and processing its source, either adding the source vertex to S or labeling it with the number of its outgoing edges with a destination that is not a currently known *Gateway* (Fig. 2C), with labels maintained by a recursive subroutine when any vertex is added to S. The algorithm terminates when no additional vertices can be added to S (Fig. 2R). For a more formal definition of this process, see Algorithm 1: *FindGateways*.

4.1.1 Correctness of the *FindGateways* Algorithm

Lemma 4.1. *During the execution of FindGateways, if a node u has a label then the label value equals the number of edges (u,r) with r not in S.*

Proof. By induction on the (decreasing) label value $Label(u)$. Initially $Label(u)$ is set to be exactly the number of edges leaving u whose destinations are not in S (Line 9 of FindGateways). When the destination of an edge (u,r) is added to S (Line 4 of FindGateways and Line 3 of UpdateLabel) UpdateLabel is called on u, decreasing $Label(u)$ by one. Finally, no node is ever removed from S and $Label(u)$ never increases. □

Algorithm 1: FindGateways

```
   Data: A set of vertices V'
   Result: A set S of all Gateways to a set of vertices V'
1  Initialize set S = V'.
2  while ∃ unlabeled vertex w ∉ S with an outgoing edge to a vertex in S do
3      if all edges whose source is w have destination vertices ∈ S then
4          Add w to S and set Label(w) = 0.
5          for each labeled vertex u with an edge whose destination is w do
6              UpdateLabels(u)
7          end
8      else
9          Set Label(w) = to the number of edges whose source is w and whose
           destination ∉ S.
10     end
11 end
12 Return S.
```

Recursive Subroutine: UpdateLabel

```
   Data: A vertex u
1  Decrement Label(u) by 1
2  if Label(u) == 0 then
3      Add u to S
4      for each labeled vertex u' that has an edge to u do
5          UpdateLabel(u')
6      end
7  end
```

Lemma 4.2. *Every vertex w added to the set S during the execution of FindGateways is a Gateway to V'.*

Proof. By induction. The set S is initialized to be V', and all vertices in V' are Gateways to V' by definition. A vertex w is added to S in two ways: 1. w is unlabeled and for every edge (w,r), the vertex r is in S (Line 3 of FindGateways), or 2. w is labeled and has label 0.

In case (1), w is also a Gateway as every maximal path of w consists of a subpath of a maximal path of a vertex $r \in S$, prepended with the edge (w,r). For case (2), first observe that $Label(w)$ always corresponds to the number of outgoing edges from w whose destinations are not in S by Lemma 4.1. Since $Label(w)$ is immediately checked for value 0 upon any change in $Label(w)$ (Line 2 of UpdateLabel) and the initial value of $Label(w)$ is non-zero, w is always added to S when all outgoing edges have destinations in S. As shown for case (1), every maximal path of w then must intersect V', so w is a Gateway to V'. □

We now prove two lemmas used in the proof of Lemma 4.5.

Lemma 4.3. *If a Gateway v to a vertex set V' lies on a cycle then some vertex $r \in V'$ also lies on the cycle.*

Proof. Let v be a Gateway to a vertex set V' and let v lie on a cycle. One maximal path of v consists of the cycle with a single edge removed: the edge whose destination is v. Since v is a Gateway to V', some vertex $r \in V'$ must lie on this path and thus on the cycle. □

Lemma 4.4. *Let v be a Gateway to a vertex set V' with $v \notin V'$. Then v has at least one outgoing edge, and for every edge (v,r) the vertex r is also a Gateway to V'.*

Proof. If v has no outgoing edges, then v has a single maximal path consisting of itself and is not a Gateway to V'. Next, suppose by contradiction that r is not Gateway to V'. Then r has some maximal path that does not intersect V'. If v does not appear in this path, then prepending v to the path forms a maximal path of v that does not intersect V'. On the other hand, if v does appear in this path, then prepending v to the path forms a non-simple path (that contains a maximal path of v) that does not intersect V'. So in either case, v has a maximal path that does not intersect V', contradicting the assumption that v is a Gateway to V'. Thus r is a Gateway to V'. □

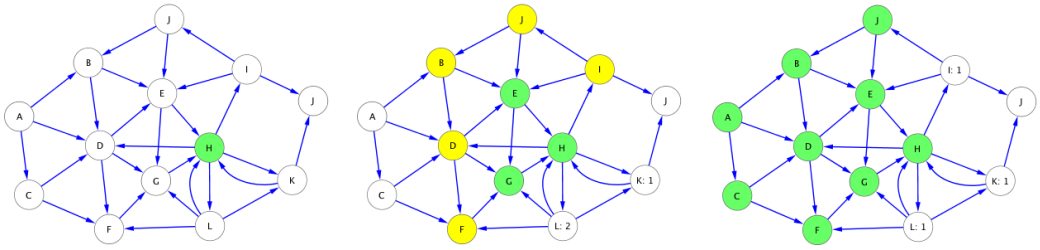

Figure 2: Execution of the *FindGateways* algorithm. (L) An interesting simulation state has been identified, and we would like to determine the set of vertices that inevitably lead to this state. A vertex H representing this state is given as input to the algorithm. (C) After a few iterations, vertices E and G have been added to the set of *Gateways* (shown in green) and vertices K and L have been processed and labeled. Yellow vertices are now in the queue to be processed. (R) Upon completion of the algorithm, the set of *Gateways* has been iteratively grown as large as possible and the final set of *Gateways* (shown in green) is returned.

Lemma 4.5. *Every Gateway v to V' is added to the set S during the execution of FindGateways.*

Proof. By contradiction. Assume that some Gateway v is not added to S. All Gateways to V' in V' are added to S during initialization, so $v \notin V'$. By Lemma 4.4 every destination of an outgoing edge from a Gateway $v \notin V'$ is another Gateway. So there exists a Gateway $v \notin V'$ such that all outgoing edges from v are in S, otherwise there is a cycle containing at least one Gateway and no node in V' (contradicting Lemma 4.3). By Lemma 4.1, the label of the Gateway v became 0 during the execution and so was added to S. □

Theorem 4.6. *The set S returned by FindGateways is the set of Gateways to the input vertex set V'.*

Proof. By Lemma 4.2 and Lemma 4.5. □

4.1.2 Complexity of the *FindGateways* Algorithm

We maintain a queue of unlabeled nodes with an outgoing edge whose destination is in S. Nodes are added to the queue at the same time UpdateLabel is called, and removed in Line 2 of FindGateways. The set S is implemented as a simple linked list, and the edges are implemented as an undirected adjacency list data structure with a flag in the source and destination entries indicating the direction of the edge.

The execution of *FindGateways* involves $O(1)$ processing for each vertex or edge in the graph a constant number of times, yielding a total running time of $O(|E|)$ (Theorem 4.7). In practice, the bound on running time is significantly lower, as the class of cases where all edges in the graph must be considered are extremely rare in both randomized directed graphs and ATGs constructed from actual data (see Section 7).

Theorem 4.7. *The FindGateways algorithm runs in $O(|E|)$ time for an ATG $G = (V,E)$.*

Proof. First, note that every node of an ATG has at least one incoming or outgoing edge (since every state has either a predecessor or successor), so $V = O(|E|)$. We use an amortized analysis based on the edges of G, and show that each edge induces $O(1)$ work.

Line 1 of *FindGateways* takes $|V| = O(|E|)$ time to add the elements of V' to S and initialize the queue. Line 2 of *FindGateways* takes $O(1)$ time for each vertex added to the queue. A vertex is added for every incoming edge to a vertex added to S, so the queue has a total of $O(|E|)$ vertices added (with possibly many duplicates) and executes $O(|E|)$ times. However, since each vertex is immediately labeled (Line 4 or 9 of *FindGateways*) it is only processed within the `while` loop once. Executing the `if-else` statement in Lines 3 through 9 of *FindGateways* requires checking the outgoing edges of a unique vertex and so takes $O(|E|)$ total time. Finally, *UpdateLabels* is called at most once for each edge (u,w) when w is added to S, and takes $O(1)$ time, using $O(|E|)$ time total. □

4.2 Terminals

In the previous section, we described a tractable method for finding the set of *Gateways* to any selected set of vertices. In many circumstances, it is also useful to be able to reverse this process: that is, given a starting vertex representing a configuration of the simulation, determine the set of inevitable outcomes (see Fig. 3). In this section, we define the concept of a *Terminal* in an ATG, and present an efficient algorithm for their computation.

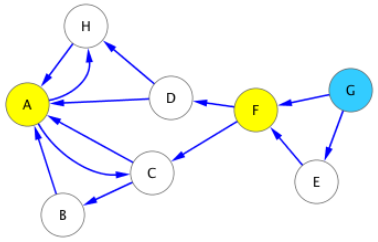

Figure 3: Vertex G is a *Gateway* to each of the yellow vertices, or *Terminals*. That is, every maximal path leaving G contains each of the yellow vertices.

Definition 3. *A Terminal of a vertex v is any vertex for which v is a Gateway.*

Because a starting vertex v is a *Gateway* to a reachable vertex w if and only if every path out of that vertex eventually passes through w, it therefore follows that any *Terminal* of a starting vertex v must lie on the intersection of all the maximal paths leading out of v. Unfortunately, while the proof of complexity is beyond the scope of this paper, it can be shown that even counting the number of maximal paths leaving a vertex in an arbitrary graph is intractable.

Another possibility would be to simply determine the subgraph reachable from the starting position, and then use a naïve approach, running *FindGateways* algorithm on each vertex in the subgraph to determine whether or not the starting vertex is contained in its *Gateway* set. Because the reachable subgraph could span the entire graph, and because *FindGateways* algorithm may need to traverse each edge in the graph twice each time it is run, this method could take up to $O(|V||E|)$ time on a graph $G = (V,E)$. While this is certainly a dramatic improvement over the previous method, it is still not efficient enough to be run in real time.

In order to be able to compute these values in real time, we present a heuristic algorithm that leverages the benefits of both

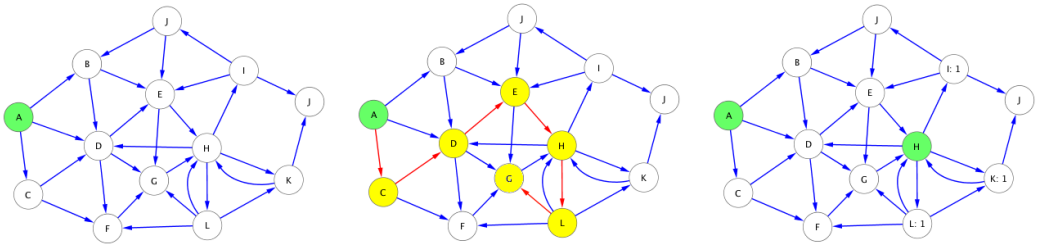

Figure 4: Execution of the *FindTerminals* algorithm. (L) A simulation state that closely resembles a real-world interaction has been identified, and we would like to determine any inevitable future events. A vertex A representing this state is given as input to the algorithm. (C) To minimize computation, we begin by pruning parts of the graph. We accomplish this by selecting a random maximal path, knowing that only vertices that appear along this path can meet inevitability criteria. Vertices along this path (shown in yellow) are now in the queue to be processed. (R) After the execution of the algorithm, the final set of *Terminals* representing inevitable future states is returned.

of these approaches. Recall that the definition of a *Terminal* implies that a *Terminal* lies on the intersection of all maximal paths out of the source vertex. Therefore, any maximal path leaving the source vertex is guaranteed to pass through any and all *Terminal* vertices that exist. By choosing a random maximal path and running *FindGateways* algorithm only on each vertex along the path rather than the entire subgraph, it is likely that we will eliminate a large portion of the computation due to performing a preliminary set reduction (see Fig. 4). Further, the construction of a random maximal path is guaranteed to find any existing *Terminals* before repeating any vertex; any vertex not yet encountered could be avoided indefinitely traversing this cycle. As such, the construction of a random maximal path may stop upon traversing any edge leading back to a vertex already in the path with full confidence that the path contains all possible *Terminals*. Our empirical results indicate that, in practice, randomly chosen maximal paths are overwhelmingly short, due perhaps to the small-world phenomena apparent in many real-world graphs (see Section 7). We formalize our prune-and-verify strategy in Algorithm 3: *FindTerminals*.

4.2.1 Correctness of the *FindTerminals* Algorithm

By Definitions 2 and 3, all *Terminals* of a specified source must appear on all paths leaving the source, and thus *FindTerminals* will run *FindGateways* on all possible terminals of an input vertex v. We showed *FindGateways* to be correct in the previous section (Thm. 4.6) and so the set of *Gateways* for each vertex that is evaluated will be correctly returned.

4.2.2 Complexity of the *FindTerminals* Algorithm

In the worst case, finding a maximal path takes $O(|V|)$ time and produces a path of length $|V|-1$. Executing *FindGateways* on each vertex of the path takes $O(|E|)$ time, and all executions take $O(|V||E|)$ total time. So the algorithm takes a total of $O(|V||E|)$ time, or $O(|V|^3)$ time in a graph with $|V|$ vertices. In practice, however, this algorithm is remarkably fast. For an empirical analysis, see Section 7.

5 Visualization System

To evaluate the ATG framework and validate our claim that the algorithms presented in the previous section are useful in an interactive visual analysis environment, we implemented a functional prototype within an existing graph visualization system. Cytoscape [27] is designed for use with large, complex networks and their associated attribute data. It offers a complete, intuitive plugin API that readily facilitated the implementation of our algorithms. Using the Cytoscape parser, we can generate ATGs from standard comma-delimited text files, and augment the vertex-edge structure with additional vertex and edge attributes representing dimensions within the original simulation data such as violence, protest, and dominant political group.

Algorithm 3: FindTerminals

```
Data: A starting vertex v
Result: A set T of all vertices to which v is a Gateway
Initialize set T = {v}.
Use GetMaximalPath to find a maximal path P starting at v.
for each vertex w along P do
    Run FindGateways algorithm on V' = {w}
    if v is in the set returned by FindGateways algorithm then
        Add w to T.
    end
end
Return T.
```

Subroutine: GetMaximalPath

```
Data: A starting vertex v
Result: A maximal path P starting at v
Add v to P.
Set currentVertex equal to v
while currentVertex has no self edges and has outdegree > 0 do
    Choose an outgoing edge at random, and set currentVertex equal to its
    destination.
    if currentVertex ∈ P then
        Return P.
    else
        Add currentVertex to P.
    end
end
Return P.
```

Leveraging this system, analysts are able to interactively perform modifications to the visual representation of the graph including layout and clustering, as well as focus on specific attributes using zooming and the application of native filters. This enables the analyst to explore the simulation space, identifying areas of interest for deeper exploration and finding patterns among simulation runs. Using our algorithms implemented as Cytoscape plugins, we are able to identify *Gateways* and *Terminals* to a user-generated input set in real time. This close coupling of interactive exploration with robust computational support for identifying interesting substructures in the graph provides support throughout the analytical sense-making process. In the following section, we will discuss several use-case scenarios and demonstrate how this prototype visual analytics system can be used to derive insights on real-world data.

6 Applications in Political Science

We have identified meaningful usage scenarios for each of our algorithms that are of interest to our collaborators in political science. Preliminary presentation to domain experts indicates that using our

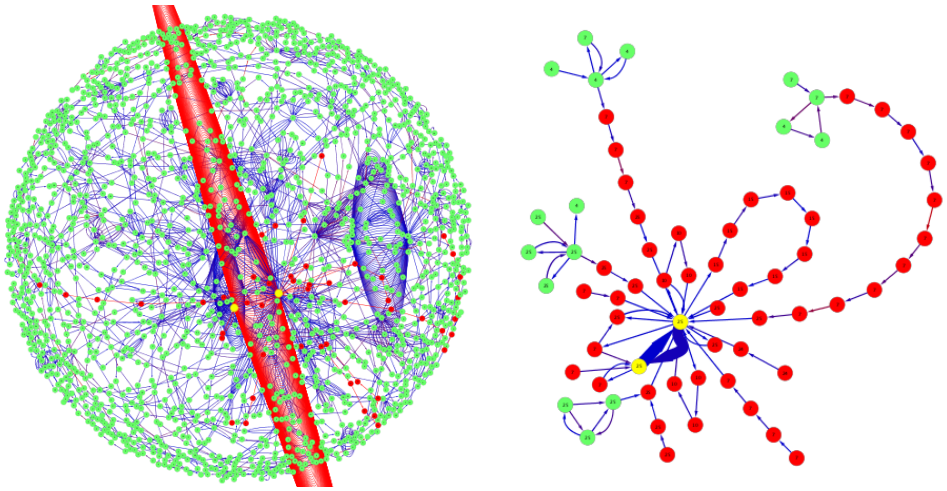

Figure 5: (L) Visualization of an ATG generated from 100 runs of an agent-based simulation of political hierarchies. The two yellow vertices represent an interesting feature: a highly stable vertex pair (note the high degree of revisitation). In red are the results of the *FindGateways* algorithm. (R) A simplified graph showing only the input vertices V' in yellow, the set of *Gateway* vertices to this set in red, and critical decision points and alternatives in green. By extracting and examining this subgraph, the analyst is able to determine that in this case there many independent paths representing distinct sequences of events leading into the yellow vertices.

prototype system, analysts are able to interact meaningfully with the data, visually and computationally explore the simulation space, and derive insight in these areas.

6.1 Keeping to the "Left of Boom"

One important task facing decision makers in political science is maintaining an awareness of threat scenarios and developing strategies for efficient, cost-effective disaster avoidance. In military terminology, this is known as keeping to the "Left of Boom". Using filtering tools, analysts can identify states that exhibit high levels of violence or states where a known oppressive regime is in power. Using these states as input to the *FindGateways* algorithm, they are able to identify the set of states in the graph that inevitably lead to dangerous or unfavorable outcomes (see Fig. 5L). This subgraph can be easily extracted using the visualization tool and reprojected to help the analyst see the relationships at play (see Fig. 5R). These states may themselves appear non-threatening, leading traditional analysis to overlook them, though they in fact represent a favorable balance of conditions for the rise to power of the oppressive organization or violent uprisings. By comparing the configurations on the periphery of the set of *Gateways*, analysts are able to make recommendations regarding early warning signs to watch for in the real world and to identify critical decision points.

6.2 Forecasting Outcomes

Another important area of concern for decision-makers is preparation for upcoming events. Using an ATG built from their simulation data, it is possible for analysts to locate a state in the system that closely resembles current conditions and to detect unavoidable future states using the *FindTerminals* algorithm (see Fig. 6). As arrival in each of the states yielded by this algorithm is inevitable, early detection will allow for more informed decisions and the most specific planning possible.

7 Evaluation

The formal analyses provided in Section 4 demonstrate worst-case computational running times with respect to the number of edges in the graph. For both algorithms, we proved that the running time is tractably polynomial even in pathological graphs. This indicates that identifying *Gateway* and *Terminal* vertices in real-world graphs is computationally reasonable on a standard machine. In the following empirical analysis we demonstrate that in an ATG generated from real data, the expected running time of both algorithms is significantly lower than the worst-case bound, and we are able to correctly identify *Gateway* and *Terminal* vertices in real-time.

To evaluate the ATG formulation as well as our implementation of the *FindGateways* and *FindTerminals* algorithms, we performed an empirical evaluation on the results of a real-world political science simulation on political violence and power systems in present-day Thailand generated by our collaborators in political science at the University of Pennsylvania. This simulation models the interaction of 31 political identities, tracking their popularity, level of violence, protest, power and influence. The simulation was run 1,000 times, with each run spanning 60 discrete time steps simulating a period of several months.

The raw data was condensed using the Dynamic Political Hierarchy (DPH) model developed by Lustick et al. [21], and then formulated as an ATG containing 14,887 unique vertices and 59,000 edges. Each vertex represents a unique DPH configuration of the 31 interacting political identities, and a directed edge represents a temporal transition between two DPH configurations over the course of one time step. Upon formulating this simulation space as an ATG, we identify some interesting characteristics of the graph using Cytoscape's native network analysis tools, and then perform a runtime analysis of our implementation of both algorithms. The following simulations were run on an Intel Core 2 Quad 2.66Ghz desktop with 2GB of RAM running Windows XP.

7.1 Small World Structure of the ATG

The characteristic path length[1] of the resulting ATG is 11.94. This indicates that despite being a relatively sparse graph (containing few edges with respect to the number of vertices) the ATG generated from this data exhibits small-world characteristics [33]. The short characteristic path length with respect to the size of the graph suggests the existence of centralized "hub states" through which many of the simulation runs pass. These states can be identified explicitly by filtering to show only vertices with a large number of incoming edges. In Fig. 7L, we provide a filtered view of some of

[1]For any connected graph the *characteristic path length* is the average distance between pairs of vertices in the graph.

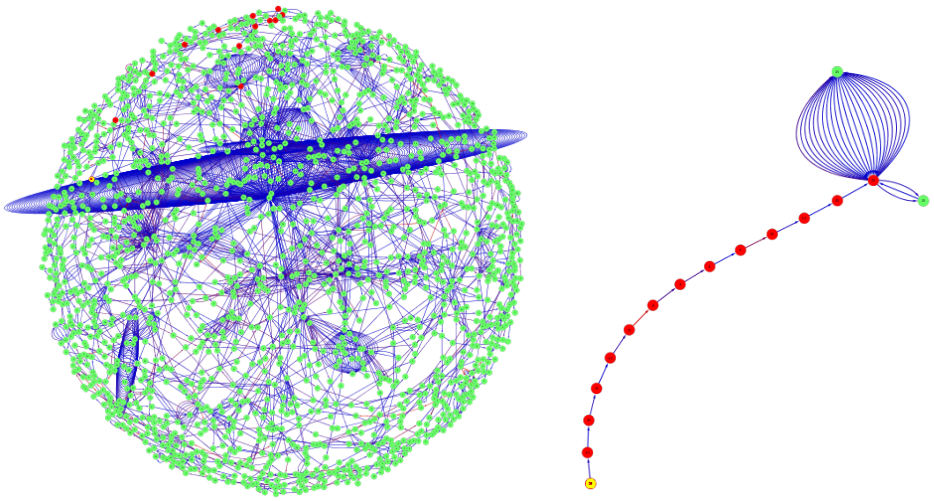

Figure 6: (L) Visualization of an ATG generated from 100 runs of an agent-based simulation of political hierarchies. The yellow vertex represents a simulation state similar to the current political climate. In red are the results of the *FindTerminals* algorithm. (R) A simplified graph showing only the input vertex in yellow and the set of *Terminal* vertices in red. By extracting and examining this subgraph, the analyst is able to see clearly that in this case there is is only one pathway predicted by the simulation once the yellow vertex has been reached.

these central hubs, condensing duplicated edges and representing the combined edge count using the weight of the remaining edge. In this ATG, there exists a set of 20 extremely centralized vertices (representing $< 0.14\%$ of the vertices in the graph) that appear as endpoints to 35% of all edges in the graph. It is also interesting to note that in this simulation, these highly centralized vertices are dominated by only two of the 31 political identities, demonstrating that these identities are particularly powerful.

7.2 Stable Vertices and Vertex Pairings

Another interesting feature of the ATG built from this data set is the presence of single vertices and small two-vertex clusters with extremely large numbers of local edges and very few edges to other vertices (see Fig. 7C and Fig. 7R). These may be considered local minima in the parameter space where the simulation tends to "settle" at a particular configuration, and may represent stable configurations of Thailand's political hierarchy.

7.3 Real-Time Utility of *FindGateways* and *FindTerminals*

To demonstrate the applicability of our algorithms for use in real-time analysis, we systematically ran our implemented version on each vertex in the graph and tracked its running time. In all instances, the *FindGateways* algorithm terminated in under $1ms$ and the *FindTerminals* algorithm terminated in under $2ms$. Compare this with $16.7ms$, the amount of time allotted per frame on average to achieve 60 frames per second, and this analysis provides strong evidence for the real-time utility of these algorithms. As a control, we also tracked the running time of the naïve version of the *FindTerminals* algorithm (without first computing a maximal path). Despite having no theoretical advantage, the *FindTerminals* algorithm as implemented reduces the actual running time by approximately three orders of magnitude, requiring under $2ms$ in all instances as compared to over $1000ms$ using the naïve approach.

8 Discussion and Future Work

In the above sections, we have made an argument for the utility of ATGs in the analysis of agent-based simulation data in social and political science. However, we believe that this representation as well as the substructures and algorithms presented are much more broadly applicable. We propose that this same formulation can be applied to other problems involving similarly large parameter spaces that can be explored in pieces. For instance, in collaborative analysis, we can aggregate the analytical trails of all analysts when using a visualization into an ATG. In this structure, a vertex would represent a configuration of a visualization based on the P-Set Model [16], and the aggregation of the analytic trails will correspond to insights and data representations that have been explored. The *FindGateways* algorithm can then be applied to this ATG structure to assist a novice analyst joining the collaboration.

In addition to collaborative analysis, there are many other problems that could be formulated as an ATG where the *Gateway* and *Terminal* structures can correspond to meaningful discoveries. While we have identified several rich applications of this framework, one persistent challenge for working within the ATG framework is identifying an appropriate means for describing states representing their data. If each configuration of the entire parameter space is represented as a unique vertex, then it is highly likely that the resulting ATG will be highly disconnected, making patterns impossible to detect. If the description of the set of vertices is too coarse, then the loss of detail could predicate patterns with little or no semantic value. Under a well-formed encoding, highly similar vertices are merged while still maintaining the integrity of the overall shape of the space. In future work, we would like to explore the semi-automatic generation of distance functions that could help analysts identify meaningful vertex encodings.

9 Conclusion

In this paper, we introduced a graph formulation to capture complex relationships between discrete simulation states in time called an Aggregate Temporal Graph (ATG). We defined the concept of a *Gateway* and its inverse, a *Terminal*, which capture the relationships between pivotal states in the simulation and their inevitable outcomes. We proposed efficient algorithms to identify these relationships and provide a proof of correctness, complexity analysis, and empirical run-time analysis. Using this formulation, we demonstrated the use of these algorithms on a large-scale social science simulation of political power and violence in present-day Thailand. Finally, we discussed broader applications of the ATG and *Gateway*/*Terminal* algorithms in domains outside social and political science and proposed areas for future research.

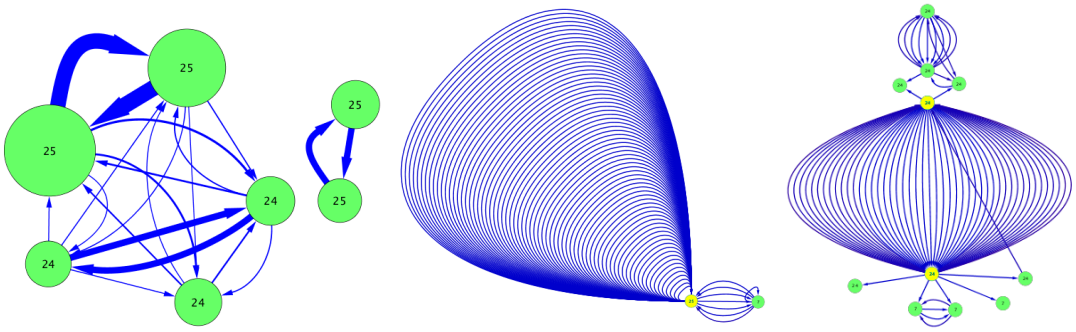

Figure 7: Examples of interesting structures seen in the ATG generated from an agent-based simulation of political violence in Thailand. (L) A tightly interconnected group of "hub" vertices. In this view, repeated edges have been represented using edge weight, vertex indegree is mapped to vertex size, and the vertex is labeled with the dominant identity in the configuration the vertex encodes. (C) A stable vertex, highlighted in yellow. The majority of the edges incident to this vertex are self-loops, suggesting that the simulation often remains in this same configuration state over a period of several timesteps. (R) A stable pairing, highlighted in yellow. Note that the majority of the edges incident to this pair are shared, suggesting that the simulation often bounces between these same two configuration states over a period of several timesteps.

ACKNOWLEDGEMENTS

This material is based in part upon work supported by the National Science Foundation under Grant Number BCS-0904646. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The authors also wish to thank Ian Lustick, Miguel Garces, and Brandon Alcorn of Lustick Consulting.

REFERENCES

[1] A. Adai, S. Date, S. Wieland, and E. Marcotte. LGL: creating a map of protein function with an algorithm for visualizing very large biological networks. *Journal of Molecular Biology*, 340(1):179–190, 2004.

[2] R. Axelrod. *The Complexity of Cooperation*. Princeton Univ. Press, 1997.

[3] R. Axtell, J. Epstein, J. Dean, G. Gumerman, A. Swedlund, J. Harburger, S. Chakravarty, R. Hammond, J. Parker, and M. Parker. Population growth and collapse in a multiagent model of the Kayenta Anasazi in Long House Valley. *Proc. National Academy of Sciences of the United States of America*, 99(Suppl 3):7275, 2002.

[4] R. Bhavnani and D. Backer. Localized Ethnic Conflict and Genocide. *Journal of Conflict Resolution*, 44(3):283, 2000.

[5] J. Blaas, C. Botha, E. Grundy, M. Jones, R. Laramee, and F. Post. Smooth graphs for visual exploration of higher-order state transitions. *Visualization & Computer Graphics, Trans. on*, 15(6):969–976, 2009.

[6] N. Collier. Repast: An extensible framework for agent simulation. *The Univ. of Chicagos Social Science Research*, 36, 2003.

[7] R. Crouser, D. Kee, D. Jeong, and R. Chang. Two visualization tools for analyzing agent-based simulations in political science. *Computer Graphics and Applications*, 32(1):67–77, 2012.

[8] A. Enright and C. Ouzounis. BioLayoutan automatic graph layout algorithm for similarity visualization. *Bioinformatics*, 17(9):853, 2001.

[9] A. S. Flowers. The Strategic Use of Public Higher Education to Reduce Individual Poverty, Spur Economic Development, and Increase Social Capital: Exploring the District of Columbia and its public university as a Case Study, 2006.

[10] L. Gasser and K. Kakugawa. MACE3J: Fast flexible distributed simulation of large, large-grain multi-agent systems. In *Proc. 1st Int'l. conference on Autonomous agents and multiagent systems: part 2*. ACM, 2002.

[11] D. Gruhl, R. Guha, D. Liben-Nowell, and A. Tomkins. Information diffusion through blogspace. In *Proc. 13th Int'l. Conf. on World Wide Web*. ACM, 2004.

[12] J. Heer and D. Boyd. Vizster: Visualizing online social networks. 2005.

[13] I. Herman, G. Melançon, and M. Marshall. Graph visualization and navigation in information visualization: A survey. *Visualization & Computer Graphics, IEEE Trans. on*, 6(1):24–43, 2000.

[14] W. Huang, S. Hong, and P. Eades. Effects of sociogram drawing conventions and edge crossings in social network visualization. *Journal of graph algorithms and applications*, 11(2):397–429, 2007.

[15] F. Iragne, M. Nikolski, B. Mathieu, D. Auber, and D. Sherman. ProViz: protein interaction visualization and exploration. *Bioinformatics*, 21(2):272, 2005.

[16] T. Jankun-Kelly, K. Ma, and M. Gertz. A model and framework for visualization exploration. *Trans. on Visualization & Computer Graphics*, 2007.

[17] V. Kostakos. Temporal graphs. *Physica A: Statistical Mechanics and its Applications*, 388(6):1007–1023, 2009.

[18] J. Leskovec, M. McGlohon, C. Faloutsos, N. Glance, and M. Hurst. Cascading behavior in large blog graphs. *Arxiv preprint arXiv:0704.2803*, 2007.

[19] D. Liben-Nowell and J. Kleinberg. Tracing information flow on a global scale using Internet chain-letter data. *Proc. National Academy of Sciences*, 105(12):4633, 2008.

[20] I. Lustick. PS-I: A user-friendly agent-based modeling platform for testing theories of political identity and political stability. *Journal of Artificial Societies & Social Simulation*, 5(3), 2002.

[21] I. Lustick, B. Alcorn, M. Garces, and A. Ruvinsky. From Theory to Simulation: The Dynamic Political Hierarchy in Country Virtualization Models. In *American Political Science Association (APSA) 2010 Annual Meeting*, 2010.

[22] I. Lustick and D. Miodownik. Abstractions, ensembles, and virtualizations simplicity and complexity in agent-based modeling. *Comparative Politics*, 41(2):223–244, 2009.

[23] A. Pretorius and J. van Wijk. Bridging the semantic gap: Visualizing transition graphs with user-defined diagrams. *Computer Graphics & Applications*, 27(5):58–66, 2007.

[24] A. Pretorius and J. Van Wijkk. Visual analysis of multivariate state transition graphs. *Visualization & Computer Graphics, Trans. on*, 12(5):685–692, 2006.

[25] M. Scheutz and P. Schermerhorn. Predicting population dynamics and evolutionary trajectories based on performance evaluations in alife simulations. In *Proc. 2005 Conf. on Genetic & Evolutionary Computation*. ACM, 2005.

[26] M. Scheutz, P. Schermerhorn, R. Connaughton, and A. Dingler. SWAGES–an extendable parallel grid experimentation system for large-scale agent-based alife simulations. *Proc. of Artificial Life X*, 2006.

[27] P. Shannon, A. Markiel, O. Ozier, N. Baliga, J. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498, 2003.

[28] A. Srbljinovic, D. Penzar, P. Rodik, and K. Kardov. An agent-based model of ethnic mobilisation. *Journal of Artificial Societies & Social Simulation*, 6(1):1, 2003.

[29] S. Tisue and U. Wilensky. Netlogo: A simple environment for modeling complexity. In *Proc. of Int'l. Conf. on Complex Systems*, 2004.

[30] F. Van Ham, H. Van De Wetering, and J. Van Wijk. Visualization of state transition graphs. In *Proc. of Information Visualization*, 2001.

[31] F. Van Ham, H. Van de Wetering, and J. Van Wijk. Interactive visualization of state transition systems. *Visualization & Computer Graphics, Trans. on*, 8(4):319–329, 2002.

[32] F. Viégas and J. Donath. Social network visualization: Can we go beyond the graph. In *Workshop on Social Networks, CSCW*, volume 4, 2004.

[33] D. Watts and S. Strogatz. Collective dynamics of small-worldnetworks. *Nature*, 393(6684):440–442, 1998.