# A-RESCUE 2.0: A High Fidelity, Parallel, Agent-Based Evacuation Simulator

**Preprint** *in* Journal of Computing in Civil Engineering · December 2018

DOI: 10.1061/(ASCE)CP.1943-5487.0000802



**6 authors**, including:

Hemant Gehlot
Purdue University
**15** PUBLICATIONS **22** CITATIONS
SEE PROFILE

Xianyuan Zhan
Microsoft
**29** PUBLICATIONS **701** CITATIONS
SEE PROFILE

Xinwu Qian
University of Alabama
**34** PUBLICATIONS **345** CITATIONS
SEE PROFILE

# A-RESCUE 2.0: A High Fidelity, Parallel, Agent-Based Evacuation Simulator

Hemant Gehlot[1], Xianyuan Zhan[1], Xinwu Qian[1], Christopher Thompson[1], Milind Kulkarni[1], and Satish V. Ukkusuri*[1]

[1]Purdue University, USA

**Abstract**

Effective evacuation of residents in hurricane affected areas is essential in reducing the overall damage and ensure public safety. However, traffic flow patterns in evacuation contexts is far more complex than normal traffic and is usually accompanied with severe congestion due to the presence of evacuees. In such scenarios, agent-based simulation can accurately capture evacuation traffic patterns, which can be extremely useful in evacuation management. However, existing simulators are not fully capable of simultaneously handling highly detailed household behaviors as well as large-scale traffic in evacuation contexts. In this study, we develop a parallelizable large-scale version of A-RESCUE (An Agent based regional evacuation simulator coupled with user enriched behavior) called A-RESCUE 2.0. Detailed household evacuation behaviors are modeled using a comprehensive decision making module. Computation loads induced by the large amount of evacuation vehicles are distributed by a parallelization scheme that involves partitioning the road network into subnetworks such that traffic updates in each subnetwork are simultaneously updated in parallel. Dynamic load balancing among different subnetworks is ensured by periodically repartitioning the network using a multi-level graph partitioning algorithm. A predictive network weighing scheme is developed that assigns weights (reflecting computational load) to the roads of network based on current and predicted future network traffic loadings. An on-demand routing strategy is also developed that allows effective rerouting computation based on changing traffic patterns. A-RESCUE 2.0 is capable of representing non-evacuee background traffic as well as uncertain events on the network like road closures. In addition, real-time monitoring of traffic patterns is made possible using a visualization module that is connected to the simulator through an efficient data communication layer. Comprehensive experiments are conducted on Miami-Dade county network to validate the applicability of developed simulator on real-world networks. Findings from experimental tests confirm that parallelization scheme is effective in improving computational performance.

## Introduction

Hurricanes are among the mostly costly and dangerous natural disasters in the United States. In 2012, Hurricane Sandy caused 147 direct casualties along its path and brought damage in excess of $50 billion for the United States (Blake et al., 2013). Other direct impacts of Sandy include destruction of 570,000 buildings, cancellation of 20,000 airline flights, 8.6 million power outages in states among others. These disastrous experiences have compelled various stakeholders such as emergency managers, crisis planners and researchers to uncover the critical role of evacuation logistics. However, effective evacuation planning is highly dependent on understanding the traffic patterns of evacuees. Development of an evacuation strategy for populations at risk from natural disaster is often problematic due to unstable and hectic traffic conditions that accompany a natural disaster. Traditional static modeling tools are not sophisticated enough to accommodate realistic scenarios that include dynamic conditions and

*(Corresponding author) Email: sukkusur@purdue.edu

varying evacuee response rates typically found in an evacuation. Due to these reasons, traffic evacuation simulators provide the right tools to model evacuation traffic patterns.

There has been a considerable amount of work in simulating traffic during evacuation. Some of the prominent works include NETVAC (Sheffi et al., 1980), TEDSS (Sherali et al., 1991), IMDAS (Franzese, 2001), OREMS (Rathi and Solanki, 1993), MASSVAC (Hobeika and Jamei, 1985; Hobeika and Kim, 1998) and CEMPS (Pidd et al., 1993). The limitations of these works are that they are specific to certain particular evacuation scenarios. For instance, TEDSS is specially designed for evacuating people around nuclear power stations and macroscopically simulates evacuation scenarios without going into microscopic details. OREMS does macroscopic traffic simulation using static traffic assignment, and thus, is not a good representation of dynamic evacuation flows. MASSVAC requires evacuation routes to be provided as input from all origins to destinations. CEMPS copes well with rectangular grid-type road networks but has not been tested for general road networks. Also, travel time of vehicles across network links in CEMPS is considered to be independent of congestion, which is not realistic.

Recently, some of the dynamic traffic simulation frameworks like such as INTEGRATION (Mitchell and Radwan, 2006), PARAMICS (Cova and Johnson, 2003), CORSIM (Williams et al., 2007), VISSIM (Han and Yuan, 2005), MITSIMLab (Jha et al., 2004), DYNASMART (Murray-Tuite, 2007), DynaMIT (Balakrishna et al., 2008), DynusT (Noh et al., 2009) and INDY (Klunder et al., 2009) have also been used to study evacuation problems. However, the aforementioned frameworks either use macroscopic traffic flow models, which lack sufficient precision in capturing traffic details, or are restricted to small-scale simulation that are not capable of handling large number of agents. Therefore, there is an urgent need to develop high fidelity evacuation frameworks that are scalable for large-scale city-wide agent-based simulations.

Agent-based simulation is a class of computational methods to model systems composed of interacting autonomous agents situated in an artificial environment (Macal and North, 2005). These autonomous agents are self-directed with the capability of making decisions and reacting to the environment. Many agent-based frameworks have been developed in recent years, such as SUMO (Soares et al., 2013), MATSim (Balmer et al., 2009), TRANSIMS (Smith et al., 1995), FLAMEGPU (Richmond, 2011), D-MSAON (Luke et al., 2005), ParamGrid (Klefstad et al., 2005), SMARTS (Ramamohanarao et al., 2016) and A-RESCUE (Ukkusuri et al., 2017). Frameworks like TRANSIMS (Smith et al., 1995) use primitive traffic flow representations such as cellular automata technique, which lack sufficient realism. In addition, a few existing agent-based frameworks like FLAMEGPU (Richmond, 2011) do not have a well-developed message passing scheme, which cannot capture complex interactions between agents. Some of these frameworks do not support distributed computing, such as SUMO (Soares et al., 2013), A-RESCUE (Ukkusuri et al., 2017). Frameworks like ParamGrid (Klefstad et al., 2005) that support distributed computing only divide computational load on the traffic network based on equally sized spatial regions without balancing actual load across the traffic network. SMARTS (Ramamohanarao et al., 2016) divides computation based on actual but static load on the traffic network and restricts the movement of vehicles to pre-defined routes. Therefore, vehicles are not allowed to choose adaptive routes based on varying traffic conditions. A-RESCUE (Ukkusuri et al., 2017) considers adaptive routing but computation slows down significantly under large number of vehicles due to lack of parallelization and an inefficient visualization interface (VI) module.

Table 1: Comparison of A-RESCUE 2.0 with other simulators

| Feature | SUMO | TRANSIMS | VISSIM | MATSIM | ParamGrid | SMARTS | A-RESCUE | A-RESCUE 2.O |
|---|---|---|---|---|---|---|---|---|
| Distributed computing | No | Yes | No | No | Yes | Yes | No | Yes |
| OpenStreetMap road data | Yes | No | No | Yes | No | Yes | Yes | Yes |
| Dynamic load balancing | No | No | No | No | No | No | No | Yes |
| Continuous spatial automation | Yes | No | Yes | Yes | Yes | Yes | Yes | Yes |
| Add-on visualization module connected though WebSockets | No | No | No | No | No | No | No | Yes |

In this paper, A-RESCUE 2.0, a large-scale, parallel, agent-based traffic simulation framework for

hurricane evacuation using the Repast Simphony toolkit (North et al., 2005) is developed. Table 1 shows major differences between A-RESCUE 2.0 and other simulators. The agents in our simulator are as follows: 1) vehicles are agents as they have complex driving behavior: lane changing, car-following, routing and 2) roads are also agents since we can model complex changing behaviors: lane closure, change of speed limit due to background traffic, connectivity to other roads. Also, detailed household evacuation behaviors are modeled using a comprehensive evacuation decision making module, which allows realistic characterization of city level evacuation demand patterns. We use a high fidelity microscopic car-following model and a lane changing model to capture detailed traffic flow characteristics. The relevance of evacuation scenario in developing this simulator is that many evacuees depart by the time evacuation warning is issued (Lindell et al., 2005). It has also been observed that there is synchronization in the departure of people and many people evacuate during a particular period (Wolshon, 2002). During this period, congestion on road networks increases. Simulation of heavy traffic during these times significantly reduces down the computational performance of existing evacuation simulators. In order to improve the computational performance of simulator a parallelization scheme is introduced. Parallelization is achieved by partitioning the traffic network into subnetworks such that traffic updates within each of the subnetworks is simultaneously updated in parallel by a Java thread. The partitioning of the network is achieved through a multilevel network partitioning algorithm called Metis (Karypis and Kumar, 1995). The implemented network partitioning algorithm is very efficient and provides high quality partitions, which boosts the parallel performance of the simulation framework. However, traffic patterns across road networks continuously change over time, leading to frequent changes in loads distributed to various subnetworks. In order to ensure equal load distribution, A-RESCUE 2.0 handles dynamic load balancing by periodically repartitioning the traffic network into sub-networks such that each sub-network has approximately the same computational load and communications between different partitions are minimized. While repartitioning the network, the simulator takes into account both current as well future load (till next repartitioning period) in the traffic network through a predictive network weighing scheme. In addition, A-RESCUE 2.0 allows adaptive routing of vehicles so that vehicles can change their paths as conditions vary across the network. Note that these uncertain conditions, like traffic jams, are common in evacuation scenarios. The simulator provides an event handling scheme that supports uncertain events like road closures. Thus, A-RESCUE 2.0 simulates real world conditions of roads jams and subsequent strategic rerouting by vehicles.

In addition, an efficient visualization interface (VI) module is developed as an add-on component, which can run on a separate machine. This module queries data from the simulator and does not slow down simulation. The visualization module is connected to simulation through efficient data communication via WebSockets (Wessels et al., 2011). Note that the existing agent-based frameworks like A-RESCUE use a built-in VI module, that is not capable of visualizing large scale simulations. Our proposed simulation framework overcomes this limitation. In short, this paper makes the following contributions:

1. A high-fidelity, agent-based, parallel traffic simulation framework is developed that distributes computational load by periodically repartitioning road network into various sub-networks.

2. A predictive network weighing scheme is developed that allows dynamic load balancing across different network partitions.

3. An adaptive on-demand routing scheme is developed that allows efficient routing of vehicles based on changing traffic conditions.

4. Background traffic and event handling schemes are incorporated for representing non-evacuee traffic and dynamic events like road closures, respectively.

5. An efficient VI module connected to simulator through a high-fidelity network connection provides real-time monitoring of traffic conditions.

6. Computational experiments are conducted to test the applicability of proposed framework on a real-world network.

7. Developed a high fidelity evacuation framework that can conduct analysis on both large-scale city (macro) level and individual household (micro) level at the same time.

This study is organized as follows. The next section presents the simulation framework in detail. Then, experimental results on the developed simulator are presented. The final section summarizes important findings from the research and presents future directions.

# Simulation Features

This section presents the details of A-RESCUE 2.0. First, the overall framework of the simulation model is introduced. Details of the components will be described in later sections.

## Framework of the agent-based simulation model

Figure 1 illustrates the overall framework of A-RESCUE 2.0. The framework is an integration of household level behavioral models, detailed traffic networks and background traffic patterns with high fidelity traffic flow models to capture the complexity of hurricane evacuation.

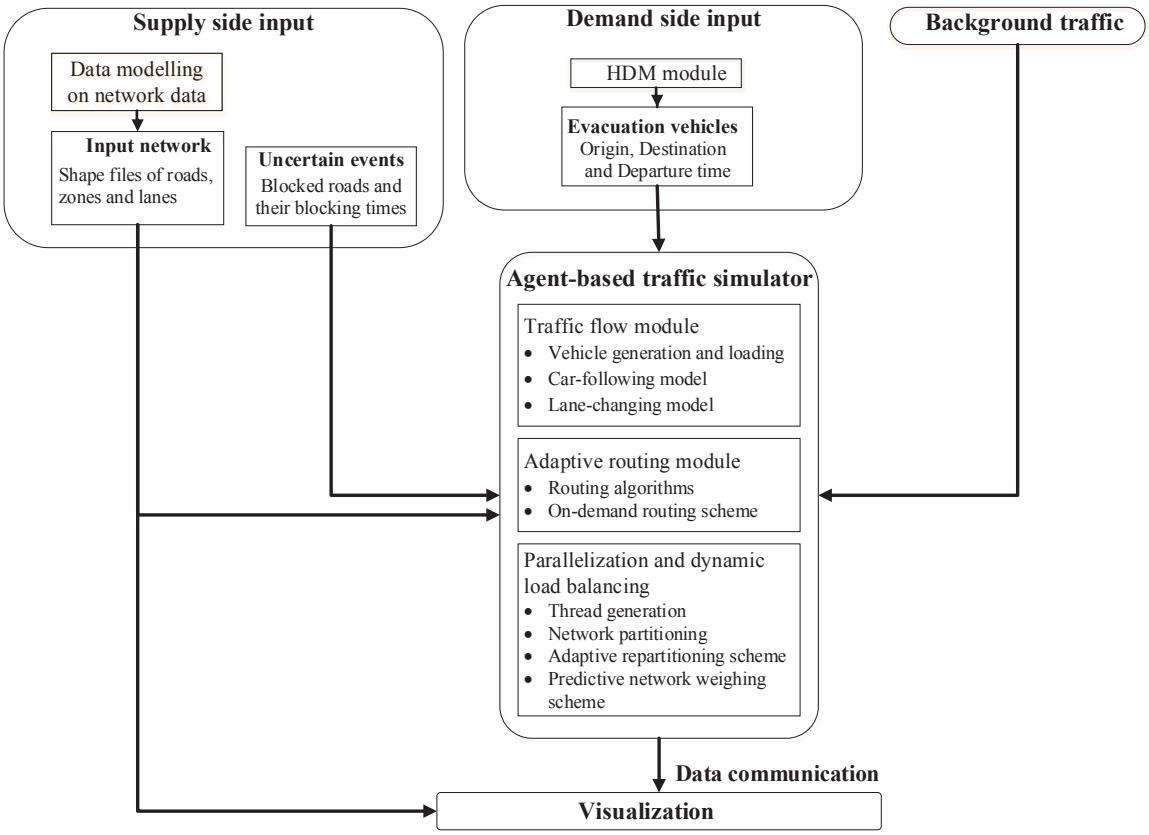



Figure 1: Simulation framework

The simulation framework uses three different types of input information, namely, demand side input, supply side input and background traffic to encode all information needed to perform simulation. The demand side input uses household behavior models that fuse various information sources to predict the number of evacuation agents and their detailed evacuation behavior (e.g. departure time and destination). Behavior models take into account the effects of personal and social relations

surrounding households that impact their information access and decision-making, households' information gathering from social media, hurricane characteristics, people's perception towards hurricane threat etc. Note that we assume that evacuation decision is made at the household level. There has been much evidence in the literature (Hasan et al., 2010; Mesa-Arango et al., 2012; Sadri et al., 2014) that many evacuation related decisions happen at the household level. Dash and Gladwin (2007) concluded that factors such as presence of children or elderly persons in the household play important roles in evacuation decision making. These factors can encourage or restrain evacuation of an individual depending on the situation. For example, the presence of children in the household may influence parents to protect them from danger through evacuation, yet the presence of a disabled person may hinder their ability to take the essential steps for evacuation. In addition, it has been documented in the evacuation literature that decisions like evacuation destination are decided considering whether all household members (including any medical patients or pets) would be accepted or not at the chosen destination (Lindell et al., 2005). Supply side input provides the high quality traffic network data which is preprocessed using a data modeling step. Supply side uncertainties, such as disruptions on the network (e.g. road closure) can also be incorporated using an event handling scheme of the simulation model. The last type of input is the background traffic information. Background traffic represents local traffic generated due to the movement of non-evacuees. In our platform, the background traffic data is collected through Google Maps API (Svennerberg, 2010).

Once all the inputs are available, the simulation platform simulates the movement of vehicles on the road network. This movement is governed by car-following and lane-changing rules. An adaptive vehicle routing scheme routes vehicles based on varying traffic patterns with the minimum computation cost. Due to the nature of microscopic simulation, the amount of computation involved in running the simulator can become very expensive as the number of vehicles increase. A parallelization strategy is developed where the network is partitioned into different subnetworks such that traffic updates corresponding to each subnetwork are updated in parallel by a separate thread. Since traffic patterns continuously change across the network, a dynamic load balancing scheme is developed to ensure approximately equal computational load across different subnetworks. This scheme periodically repartitions the network into different partitions taking into account both current and future loads till next repartitioning period. Empirical experiments show that this scheme is scalable and capable of handling large-scale hurricane evacuation simulation.

The VI module of A-RESCUE 2.0 is designed as a separate add-on module. The vehicle movements are transferred from simulator to VI module using a network data communication component. This design can allow the VI module to run on a separate machine, which reduces the computation and memory consumption on the simulation machine. The details of data communication implementation and VI module would be provided later.

## Input and data modeling

The simulation platform requires a high quality GIS road network map as input. It is crucial that the network of the simulator represents the real-world traffic condition well. A pre-processing module called data modeling (DM) takes a road network shapefile as input and generates an accurate lane-based network file that is required for the simulator and for our VI module. The first part of our DM module is developed in Visual Basic as an add-on for ArcMap. The main functionality of the DM module can be summarized as follows.

1. *Lane-offset:* Most of the existing GIS road shapefile data only contains information at the link level, while the number of lanes is an attribute of one particular link. In order to capture realistic lane-changing behaviors in the network, a detailed lane-based network with accurate lane connectivity information is needed. The lane-offset function is therefore introduced to build a lane-based network from the link shapefile. An example of lane-offset is given in Figure 2. The link object in the example is bi-directional and has 2 lanes in each direction. The lane-offset function results in 4 lane objects of unique IDs, and their topology is inherited from the link connectivity in the original shapefile. In our simulation, the vehicle routing is still conducted on the link level, and the corresponding lane to drive on will be assigned to the corresponding vehicle depending on the direction it heads to. The vehicle traveling towards right direction

will use lane 10013 for straight and left turning movements, and lane 10014 for right turning movement in this example.

2. *Lane connectivity:*. After offsetting the lanes, the next task is to define the connectivity of offset lanes to ensure proper vehicle movements at each intersection. We developed rules to automatically assign downstream and upstream lanes for each offset lane based on network topology. In particular, if there are more than three lanes, the right-most lane will be used as a dedicated right-turning lane and the left-most lane will be used as a dedicated left-turning lane. Otherwise, a lane will represent both straight and left/right turning lanes.

3. *Error diagnostic:* Note that the raw road network shapefile may have erroneous connectivity and attribute information for certain links and nodes. So, an error diagnostic tool to automatically filter out potentially erroneous nodes and links is developed. This includes filtering out nodes with more than 5 links, links that are not connected to other parts of the network, and links with impractical geometry shape. These mistakes are corrected before going through the DM procedure, which ensures that the output network is perfectly functional for our simulator.

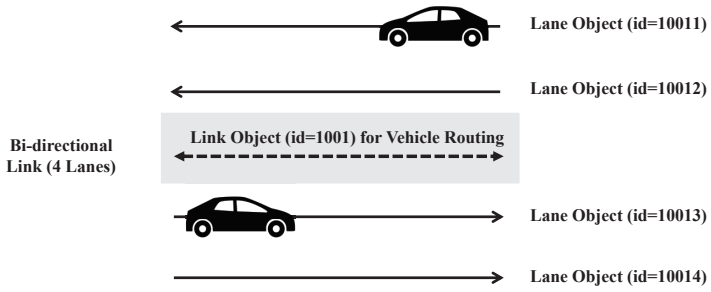



Figure 2: Lane-offset for a bi-directional link (number of lanes is 4)

Besides preparing the input network for our simulator, the other step of our DM module is to prepare the OpenStreetMap (OSM) file for our VI module that is consistent with the input of our simulator. The framework of input preparation for VI module is presented in Figure 3. We first extract the raw OSM file from the OSM server with predefined bounding box (corresponding to the study area). This file contains multiple layers as shown in the PostGIS database section in Figure 3. The OSM file may have very different roadway information as compared to the input of our simulation, as it is generated from different data source. In order to deliver a consistent visualization, the existing roadways of the OSM file are further replaced with the generated shapefile for the simulator. This gives the final OSM file that is used for our VI module.

## Household decision making module

The household decision making (HDM) module is an important component in A-RESCUE 2.0. The households located in the area under hurricane threat often make multiple decisions for their evacuation. Our HDM module is build upon several recent studies (Ukkusuri et al. (2017), Hasan et al. (2010), Mesa-Arango et al. (2012), Sadri et al. (2014), Murray-Tuite and Wolshon (2013), Hasan et al. (2013)) that examine realistic household evacuation behavior and evacuation-related factors, such as hurricane trajectory and evacuation warning influence. The HDM module considers five main decisions that households make: (1) whether to evacuate, (2) accommodation type selection (friends/relatives homes, public shelters or hotels), (3) evacuation destination (the specific destination), (4) evacuation mode (auto-based evacuation or carpooling) and (5) departure time. First, the decision of a household

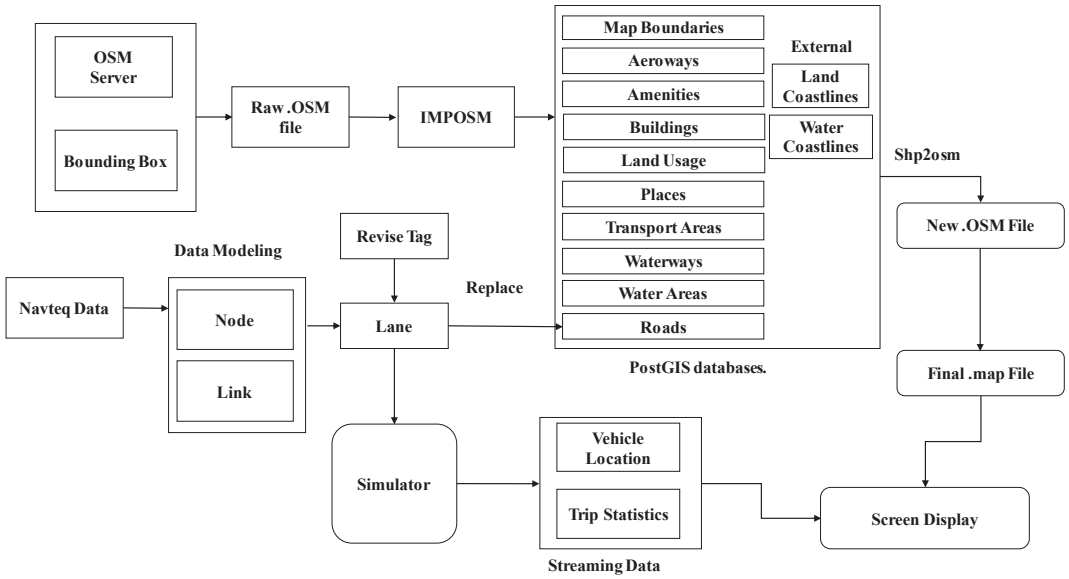



Figure 3: Overall framework for preparing input file for the VI module

to evacuate was modeled using a random parameter binary logit model (Hasan et al., 2010). Some of the variables that were found to be significant in this model are age, education status, income level, household characteristics etc. After this, accommodation type and evacuation destination are modeled using logit models (Mesa-Arango et al., 2012). Variables like household geographic location, income, preparation time, changes in evacuation plans, race etc., were found to be significant in influencing this decision. Next, mode choice is modeled using a nested logit model (Sadri et al., 2014) depending on whether the household owns vehicles. If not, they are assigned as a passenger to be picked up by a family member, friend etc., or to transit. If the household owns vehicles, the number of vehicles used is determined from a truncated model. In mode choice model, variables like gender, marital status, number of aged people in household, previous hurricane experience etc. were found to be significant. Finally, we model departure time using a hazard based duration model (Hasan et al., 2013). Here, variables like geographic location, whether received an evacuation notice, education status, income level, race etc. were found to be significant. The outputs from HDM module for each household is a schedule for evacuation trip. Each trip is described by an origin, destination, departure time. This trip information is an input to the simulator, based on which vehicles are generated and assigned into the traffic network. Note that these behavioral models are validated using the data from Hurricane Ivan including the households from Alabama, Louisiana, Florida and Mississippi (evacuate/stay decision, number of household vehicles used), and data from a behavioral-intention survey for Miami (accommodate type, evacuation destination, mode assignment for non-personal vehicles, departure time for the ultimate evacuation trip, activity participation and scheduling (Ukkusuri et al., 2017).

## Traffic flow module

The traffic flow (TF) module is another important component in our simulation model which mimics the movement of vehicles on the traffic network. The TF module is implemented in three components. The first component involves the creation and loading of vehicles. After this, vehicle movements are simulated using car-following and lane-changing models for traveling across the network.

### Vehicle generation and loading

Vehicles are generated based on the output of the HDM model. Once vehicles are generated they are stored in a pre-trip first-in first-out (FIFO) queue that sorts the vehicles based on increasing departure time i.e., vehicles that depart early are placed at the top, before vehicles that depart late. The entrance road link for a vehicle is the one that is nearest to the household location. Before entering the network, vehicles check for the availability of adequate leading spaces in the entrance road links. If adequate leading space is not available then vehicles wait in the queue for a later simulation step. The initial position and speed of the vehicles are determined based on vehicles' desired speeds, simulation step size and prevailing traffic conditions. Once a vehicle successfully enters the network it is removed from the pre-trip queue of the entrance link.

### Car-following model

The car-following model governs how vehicles follow one another on a roadway. Our simulation framework implements the car-following model developed by Yang et al. (2000). The acceleration of a vehicle is determined based on its relationship with the leading vehicle. This relationship is a function of the headway with the leading vehicle. There are three regimes, depending on the magnitude of headway: free-flowing, car-following and emergency decelerating (Herman et al., 1959). Once acceleration/deceleration rate is computed using the car-following model, the vehicle's speed and position are updated at each simulation tick. See Ukkusuri et al. (2017) for the details of various regimes of car-following model.

### Lane-changing model

Lane changing behavior has significant impact on traffic flow and thus affects network congestion level. Our implementation is based on the lane changing model developed by Yang et al. (2000) and Gipps (1986), which consider two types of lane changing behavior: 1) Mandatory lane changing (MLC): this arises when vehicle needs to change the lane to get into correct lane to cross over an intersection and 2) discretionary lane changing (DLC): this arises when a vehicle decides to move to a different lane with better perceived traffic conditions. MLC has priority over DLC and therefore MLC is mainly practiced in the region closer to the downstream junction. We implement MLC in the half of road that is closer to downstream and DLC in the remaining half. Note that for both MLC and DLC, the lane changing vehicle checks its gap distance from the leading vehicle as well as its gap distance from the lagging vehicle in the target lane. Only when these gap distances are greater than a predefined minimum distance, can the vehicle change lanes (Ahmed, 1999).

## Adaptive routing module

An adaptive routing scheme has been implemented that considers updated traffic pattern across the network. Note that link travel times are periodically updated across the network to consider time varying traffic conditions. At every tick of the simulation, all the vehicles need to perform routing check if travel times have changed since the last time when routing was performed. If so, they seek for new travel paths using a routing algorithm. In A-RESCUE 2.0, two routing algorithms have been developed: shortest path and stochastic k-shortest path algorithms. For both the algorithms, threads safe libraries have been used to consider the possibility of multiple vehicles (belonging to different partitions) simultaneously running routing algorithms. Once routes are obtained, all the vehicles store their paths until link travel times are again updated.

### Routing algorithms

In the simulator, two routing algorithms are implemented. The first algorithm is the shortest path ($SP$) algorithm that computes the path with lowest travel time from one node to another node. Thus, route decision in SP is deterministic. The other algorithm is known as K shortest paths ($KSP$) algorithm that computes K paths that have lowest travel times from one node to another. A vehicle chooses path

$i$ with probability $p_i$ determined by a *logit* based function that assigns higher probability to a path that has lower travel time, $\psi_i$:

$$p_i = \frac{\exp(-\theta\psi_i)}{\sum_i \exp(-\theta\psi_i)}$$

where $\theta$ is a travel time weighing factor. Due to computation of multiple paths in *KSP* in comparison to single path computation in *SP*, routing through *KSP* is computationally move expensive than *SP*.

#### Low cost on-demand routing scheme

An on-demand routing scheme is implemented to minimize the overall routing computation in simulation. Whenever a vehicle reaches a junction, it checks if travel times have changed since the last time when routing was preformed. If travel times change, the vehicle recomputes path from the current junction to destination using one of the routing algorithms based on the updated travel times and stores this path. Otherwise, the vehicle chooses the next link based on the stored path. As routing is only preformed when necessary and stored routes are used until the network condition is refreshed, it is termed as on-demand routing. Note that a vehicle having reached an intersection may not be able to instantly travel along the updated route because of the restrictions on choosing the next link due to the current lane of vehicle. In that case, vehicle chooses the next link based on the old route and travels along an updated route thereafter.

## Parallelization and dynamic load balancing

Since the amount of computation of simulation can significantly increase when the number of vehicles is large, a parallelization strategy is introduced to enhance the computational performance of A-RESCUE 2.0. The traffic network is partitioned into various subnetworks such that traffic updates within each of the sub-networks are simultaneously computed in parallel by a Java thread. Properly balancing the computational load across sub-networks is essential to achieve the best parallelization performance. Therefore, the simulator uses a dynamic loading balancing strategy that periodically repartitions the road network into sub-networks with approximately equal computational load using a predictive network weight scheme. The following subsections present the thread generation, network partitioning and load balancing schemes in detail.

#### Thread generation

A-RESCUE 2.0 implements a Java Thread pool to efficiently manage parallel threads and reduce computation overhead due to thread creation and destruction. A thread pool is a fixed group of threads that constantly waits for computation jobs to execute. The idea is to have the threads always existing to reduce computation overhead due to thread creation and destruction. At every simulation tick, each thread is assigned to a particular network partition, and updates vehicle movements on all roads in the network partition.

#### Network partitioning

The road network is partitioned every network repartitioning period, $T_{part}$, based on the weights of the computation load graph. The computation load graph is a weighted graph, with weights on nodes and edges approximating the computational load. More details related to the computation load graph are presented later. The network partitioning is performed using an existing Java implementation of the Metis algorithm called GMetis (Pingali et al., 2011). Metis is an efficient multi-level network partitioning algorithm, which is widely used for balancing load on distributed computing tasks. The main objective of Metis is to partition the network into a set of partitions, such that each partition has approximately equal total node weight and in-between partition communications (total weights on the boundary edges) are minimized. Metis partitions the network using a three-stage process (Karypis and Kumar, 1995). The network first collapses a local cluster of connected nodes into a contracted node, hence coarsen the origin network to a much smaller graph. Next, a series of bisections of the coarsened graph are performed to obtain the desired number of partitions based on the weights of

contracted nodes. Finally, partitions are projected back towards the original graph (finer graph), and gradually refining each partition towards the partitioning objective (equal node weights and minimum boundary edge weights). For a more detailed description of Metis algorithm, the readers can refer to Karypis and Kumar (1995), Pingali et al. (2011).

On completion of the GMetis algorithm, the road network is divided into various partitions, where each road belongs to one of the categories: in-partition road or boundary road. In-partition roads are roads whose both upstream and downstream nodes belong to the same partition of network and boundary roads are roads whose upstream and downstream nodes belong to different partitions. The partition assignment for boundary roads is sequentially performed by merging each road to the neighboring partition with lower total edge weight, which ensures the computational load of each partition remains approximately equal.

**Adaptive repartitioning scheme**

An efficient repartitioning scheme is implemented that considers dynamically changing traffic conditions. Repartitioning of the network is scheduled every fixed interval of time ($T_{part}$), but the actual repartitioning is executed only when the number of vehicles in the network is above a threshold, $N_{min}$. The idea behind is that if the amount of traffic on the network is low, then the computational load as well as the amount of variation in the partitioning weights as compared to the previous repartitioning period would be low. Thus, repartitioning computation could be saved by simply using the previous network partitioning. To avoid prolonged periods of non-partitioning, a maximum duration ($T_{max}$) is introduced to implement network repartitioning using the most recent traffic condition.

**Predictive network weighting scheme**

To partition the traffic network into subnetworks/partitions with approximately equal computational load, the computational load in the traffic network within the future time period (till the next repartitioning period) needs to be predicted. The prediction is performed on a computation load graph, where edges are the roads and nodes are the intersections. The computation load graph is a weighted graph, with weights on nodes and edges approximating the computational load. The main computation on vehicles' movement update can be decomposed to two parts, which are 1) car-following and lane changing updates and 2) routing computation. The routing computation is more costly compared to the car-following and lane changing updates. Therefore, computational loads on roads (represented as edge weights) is estimated as the sum of three components: number of current vehicles on the roads (denoted as $N_c$), predicted number of vehicles that would be traveling on the road until the next repartitioning period (denoted as $N_t$) and the predicted number of vehicles that would perform routing on the road until the next repartitioning period (denoted as $N_r$). The following equation represents the weight of an edge:

$$W_e = \alpha N_c + \beta N_t + \gamma N_r \tag{1}$$

where $\alpha, \beta$ and $\gamma$ are scale parameters for different components. $N_c$ can be easily obtained by counting the number of vehicles on each road. The computation of $N_t$ and $N_r$ involves future prediction, which are estimated based on the current route of each vehicle and network travel times based on the most recent network condition. The following procedure is used for computing these two terms:

1. Predicted number of vehicles that would travel the road until the next repartitioning period ($N_t$): For each vehicle, up to $N_{lookup}$ (a predefined threshold) downstream reachable roads on the vehicle's current route are tracked. Reachability is examined by computing the cumulative travel time starting from the current road. If the cumulative travel time on a downstream road segment is smaller than the length of the repartitioning period $T_{part}$, the count $N_t$ will increment by 1 on that road segment. $N_t$ is an approximation of car-following and lane changing updates' computational load in the next network repartitioning period, as road travel times and vehicles' routes are based on current network traffic condition.

2. Predicted number of vehicles that would perform routing on the road until the next repartitioning period ($N_r$): The routing computation is very costly compared to car-following and lane changing

updates. Knowing where routing computation occurs is important to accurately capture computational load associated with routing. Similar to $N_t$ computation, up to $N_{lookup}$ downstream reachable roads are tracked using cumulative travel time starting from current road. However, only those roads for which cumulative travel time exceeds an integral multiple of network refresh period $T_{ref}$ ($T_{ref} \leq T_{part}$) are considered, and then road's $N_r$ is incremented by one. This is because routing is performed after every time the network is refreshed. The maximum possible future routing roads of a vehicle is equal to $\lfloor T_{part}/T_{ref} \rfloor$. Once this procedure is performed for every vehicle, the $N_r$ value for each road in the network is obtained.

Both $N_t$ and $N_r$ are approximations of car-following and lane changing updates' loads as well as the routing computational loads. This is because that road travel times and vehicles' routes are based on current network traffic condition rather than actual future network traffic conditions. Moreover, finding the proper values of the scale parameters $\alpha, \beta$ and $\gamma$ for $N_c$, $N_t$ and $N_r$ is important to accurately estimate the computational load. In our simulation model, the scale parameters $\alpha, \beta$ and $\gamma$ are determined by a series of parameter tuning experiments. The details of these experiments are presented later. After edge weights are computed, node weights are computed as follows:

$$W_n = \frac{1}{2} \sum_{e' \in \Gamma(n)} W_{e'} \tag{2}$$

where $\Gamma(n)$ represents the set of adjacent edges (roads) of node (intersection) $n$.

## Background traffic and event handling

Background traffic represents local traffic generated by non-evacuees. Inclusion of background traffic makes A-RESCUE 2.0 more reliable and robust. City-scale background traffic states are included in terms of hourly link travel times in the traffic simulator. We utilize Google Maps Directions API to extract road travel time data by querying travel times between road starting and end locations on Google Map. We collected hourly travel times of more than 4,000 roads in the study region for a normal day (i.e. a day not experiencing a hurricane event). In future, a similar dataset for a day during hurricane events will be collected to reflect real background traffic. Figure 4 shows the three main steps of the inclusion of background traffic into simulation platforms, including data collection and periodically refreshing road free flow speeds using the hourly travel speed observed in the background traffic. Moreover, background traffic update on a road will be overridden if there is an event simultaneously occurring on the road (e.g. road closure, which sets the road free flow speed to 0).

An event handling scheme is also developed that can simulate events like road closure, which is useful to simulate disruptions on road network during hurricanes. The input to the scheme is the list of events with their starting and ending times. In the beginning of the simulation, all the events are loaded into a queue. At each time interval new events that need to be started are checked from the queue and are put in a different queue containing running events, which is sorted in the increasing order of ending times of events. The finished events are removed from the running events queue at their ending times. When an event is activated, the road corresponding to the event is found and the event is executed. If an event is deactivated at its ending time, then original status (e.g. free-flow speed of the road) is restored.

## Data communication

The simulator and VI module are remotely integrated via a network communication layer built upon the widely-known and industry standard WebSocket library (Wessels et al., 2011). This provides an efficient channel through which the simulator can pass information to the VI module. Before the simulator can pass data to the VI module, it must first collect the data. The following section describes this process and then the way collected data is transferred by the network communication layer is discussed.

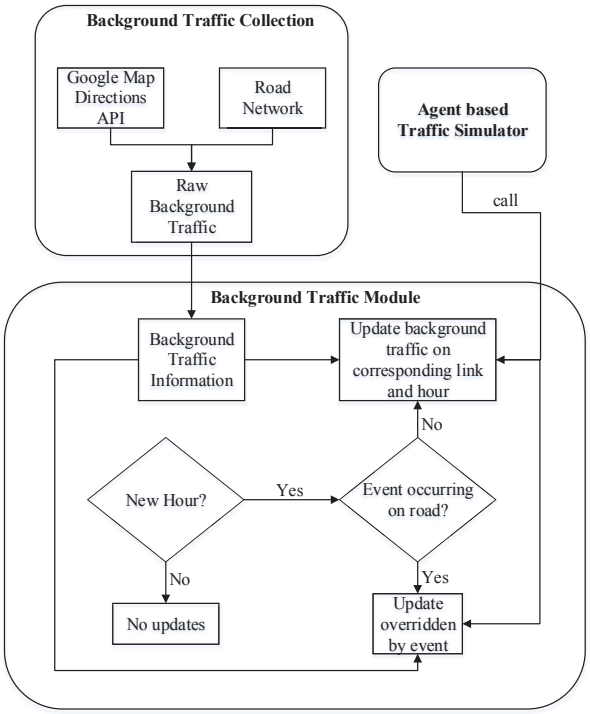



Figure 4: The inclusion of background traffic into simulation platform

### Data collection

Data collection within A-RESCUE 2.0 follows the organization scheme of collecting together all the data points generated during each tick of the simulation. The data generated in each tick is placed into a queue that stores the data for different ticks. The proposed data collection framework design is efficient, scalable, and expandable because the framework passively waits for the simulator to provide new data points. Thus, there is no wasted effort to check unchanging objects within the model when no new data is generated. For example, vehicles in the network might not move for some time steps, therefore for these time steps simulation does need not provide any input to the collection framework as there is no update in the information.

### Network communication

The network communication between simulator and the add-on VI module is built using WebSocket library (Wessels et al., 2011). A survey of common networking platforms identified REST (Fielding, 2000) and WebSocket as two good candidates based upon industry accepted standards. Both feature a robust collection of libraries for easy implementation in a wide variety of modern programming languages. Due to the volume of data being generated and greater efficiencies inherent in its design for continuously streaming data, WebSocket is selected as the platform for A-RESCUE 2.0. In addition, WebSocket connections are persistent between data transmissions and avoid the overhead issues of protocols like REST that need to establish a new connection with each request of data transfer.

Once the data collection framework begins to receive the data generated during different ticks, data is periodically streamed through WebSocket to the VI module. WebSocket connections send each tick data as one message. Then, VI module receives communication from the simulator as a stream of messages to be processed. When a new data message is received, it is decoded into an appropriate data structure to be utilized by the VI module. The decoded data is placed into a queue from which the module can read and process each simulated tick sequentially to recreate the visualization of simulator's

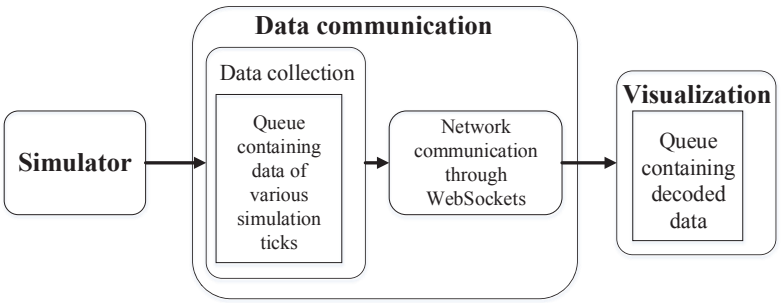



Figure 5: Data communication layer connecting simulator with VI module

execution history. Figure 5 presents the data communication framework connecting simulator and VI module.

## Visualization

We developed our VI module to address the drawbacks associated with the performance of built-in VI module of the Repast Simphony simulation platform, where the displayed simulation information is very limited and is found to be a performance bottleneck of our simulator. The VI module obtains the streaming data from simulator through the data communication protocol, visualizes the vehicle movements and efficiently monitors the network status of the simulation. The VI module is based on Java platform, for which the input file is obtained as explained before. We use Mapsforge library (Hansen et al., 2016) to render the OSM file as the underlying map, and GEOS library (Steiniger and Hunter, 2013) to handle the fundamental geometries of shapefile. A screenshot of the visualization interface is shown in Figure 6.

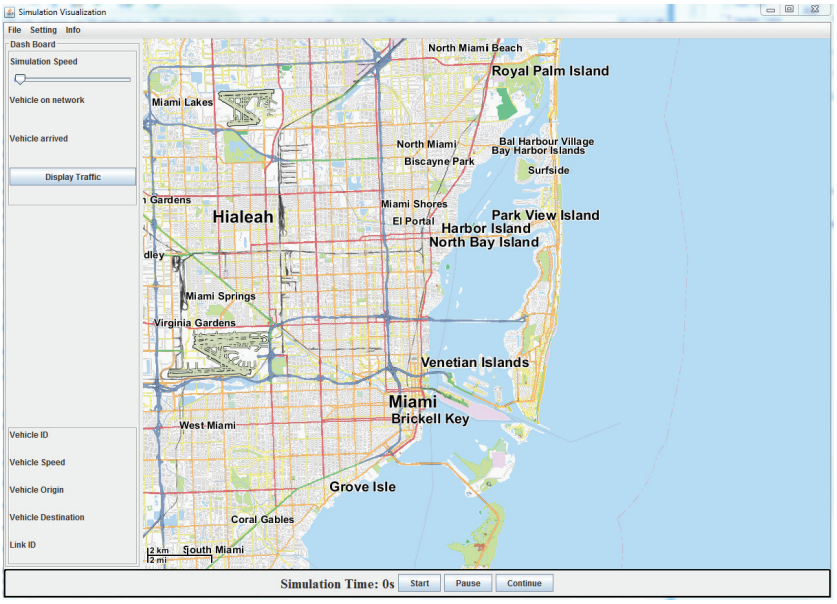



Figure 6: Visualization interface

The VI consists of three main sections: the control panel on top left corner, the information panel on lower left corner, and the entire map view of the study area which occupies most part of the

interface. We can input map file, adjust simulation speed, and change vehicle color and vehicle size using the control panel. The information panel shows the summary of statistics for the entire network including the total departed and arrived vehicles, as well as information of selected vehicles (e.g., ID, speed, origin and destination). A snapshot of the control panel and selected vehicle on the network is shown in Figure 7.

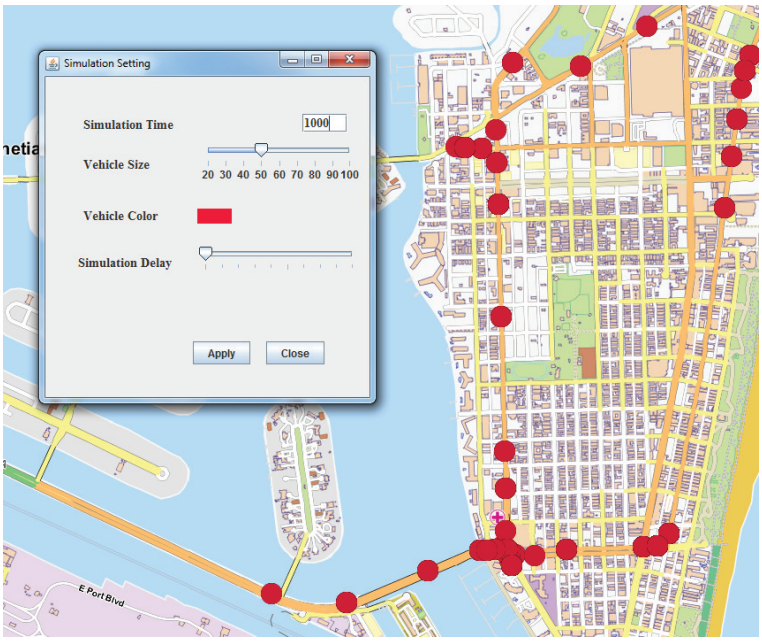



Figure 7: Screen shot of visualization setting and running vehicles

Besides the basic functionality for visualizing vehicle movements, an additional component to inspect network status is also developed. The component is named as shapefile viewer, which displays the underlying shapefile of the network and is synchronized with the view of vehicle movements, as shown in Figure 8. Since the rendered OSM file is a tile map without geometry information, the shapefile viewer provides the detailed link-level information, including link IDs, number of lanes, speed limits, and free flow travel times etc. In addition, it can display dynamic background traffic information at link level (see Figure 9), which helps to monitor the traffic condition of the network for the evacuating vehicles.

Note that computational performance is a major concern of this research, which is also a motivating reason for developing the VI module. Therefore, the computational performance of the VI module is tested. We conduct the experiments with various load levels, from 1,000 vehicles up to 100,000 vehicles. The test network is the Miami-Dade network with more than 4,000 nodes and 8,000 links. For the worst case (100,000 vehicles), only the vehicle loading part consumes most of the time (around 30 seconds) since all vehicles were loaded at the same time. As soon as all vehicles are loaded properly, the visualization of their movement is very smooth and efficient, with CPU usage being consistently lower than 20% on a desktop with @3.4GHz processor and 16GB RAM.

# Results

In this section, the results of simulation scenarios conducted on a real world road network are presented. The default test configuration is set to 20,000 vehicles, 250000 simulation ticks or 20.8 hours of real world time on Miami Dade county network that contains about 8580 roads. We also conduct tests with a larger vehicle demand whose details would be later provided. After completing extensive computational tests both $T_{part}$ and $T_{ref}$ are set equal to 1000 simulation ticks. Also, parameters $N_{min}$

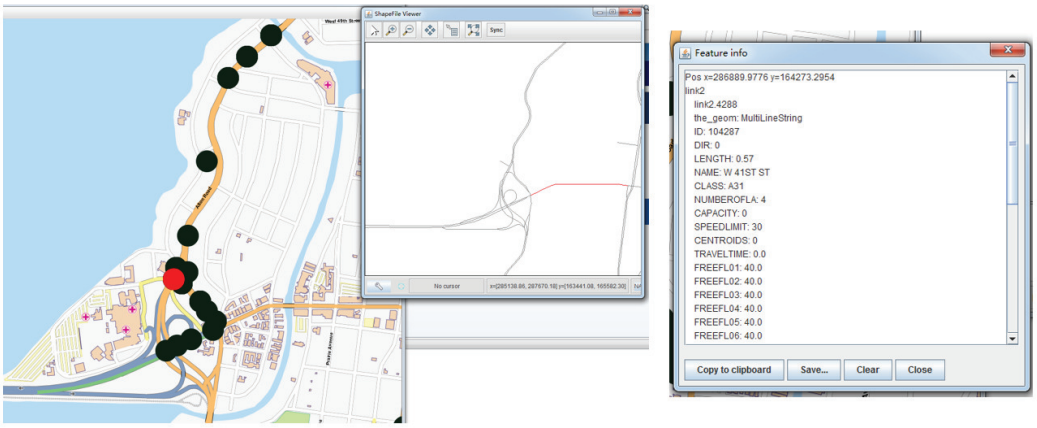



Figure 8: The shapefile viewer

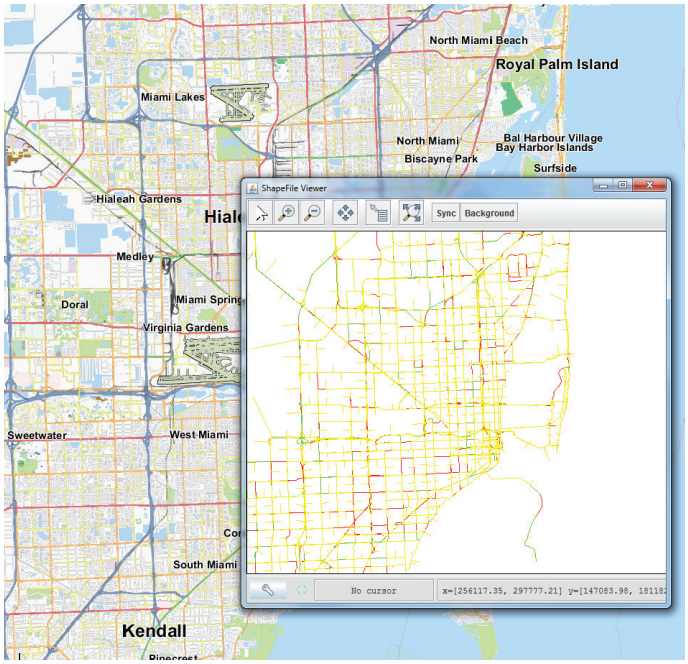
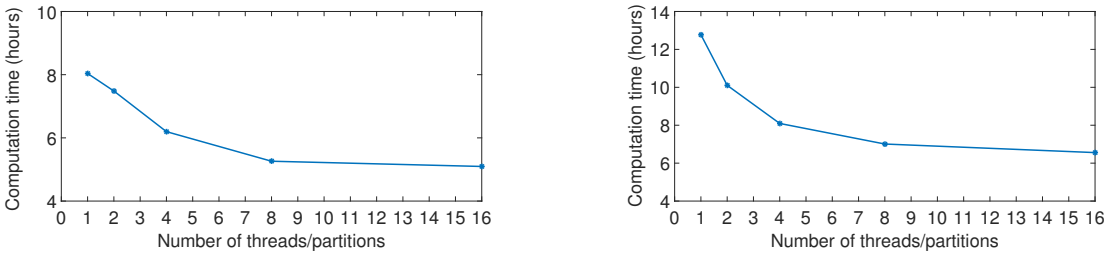



Figure 9: The shapefile viewer displaying background traffic information

and $T_{max}$ of adaptive network partitioning are set to 200 vehicles and 10000 ticks, respectively. In order to compute scale parameters $\alpha$, $\beta$ and $\gamma$ for predictive network weighting scheme comprehensive experimental tests are conducted. We discuss more about these tests in a later section. Parameters $(\alpha, \beta, \gamma)$ were selected equal to $(15, 5, 10)$ and $(10, 5, 10)$ for *SP* and *KSP*, respectively. The tests are conducted on a machine having 32 logical cores, memory of 252 GB RAM and Intel Xeon processors with 2.90 GHz CPU speed.

## Computational performance with the number of partitions and routing strategies

In this section, variation of computational performance with the number of partitions is discussed. Figures 10a and 10b provide the plots of computation time for routing strategies *SP* and *KSP*, respectively. Note that the number partitions are always equal to some powers of two because partitioning in GMetis is done using a recursive algorithm that creates partitions by bisecting existing partitions. It can be observed that computation time reduces with the number of partitions for both the routing algorithms. However, the improvement in computation time is more for *KSP* as compared to *SP* because of high routing computation involved in *KSP* as compared to *SP*. This computation gets distributed among the threads of different partitions and overall computation time reduces.




(a) *SP* (b) *KSP*

Figure 10: Variation of computation time with the number of partitions

We also conducted scalability tests with a larger demand file that consisted of around 80,000 vehicles for the duration of 1 million simulation ticks or 83 hours of real world time on the same network. Figures 11a and 11b present the plots of computation time for routing strategies *SP* and *KSP*, respectively. It can be observed that improvement in computation time for *KSP* is higher as compared to *SP* for the same reasons mentioned before.

## Variation of computation with repartitioning period

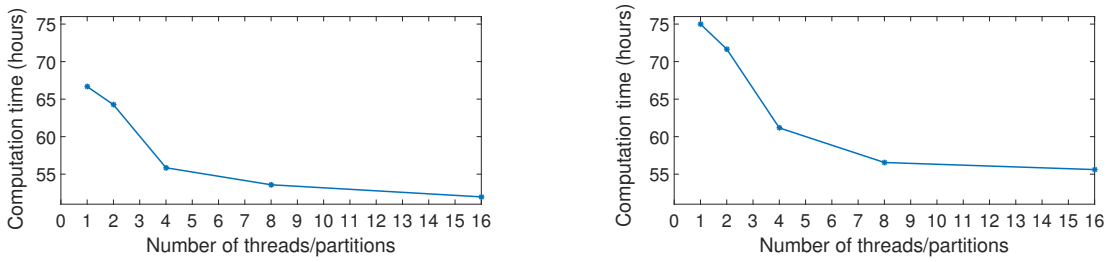
In this section, the effect of repartitioning period, $T_{part}$, on computational performance is analyzed. As mentioned before, the length of this period determines how frequently the network is repartitioned. Table 2 presents the variation of computational performance with $T_{part}$ while other parameters are kept constant including $T_{ref}$ (set to 1000 simulation ticks). It can be observed that simulation time increases with repartitioning period. That is because although the computation corresponding to repartitioning reduces with increasing $T_{part}$, the computational imbalance across different partitions increases when repartitioning is done less frequently. The latter overshadows the effect of former and hence, the overall computation increases with $T_{part}$.

Table 2: Variation of simulation time with repartitioning period

| $T_{part}(ticks)$ | Computation time (hours) |
|---|---|
| 1000 | 7.01 |
| 2000 | 7.21 |
| 3000 | 7.42 |
| 4000 | 7.60 |
| 5000 | 7.63 |

## Variation of computation with network refresh period

We now understand the effect of network refresh period $T_{ref}$ on computational performance. Table 3 presents the variation of computation time with with $T_{ref}$ (other parameters are kept fixed including




(a) *SP* (b) *KSP*

Figure 11: Variation of computation time with the number of partitions for larger demand file

repartitioning period $T_{part}$ which is set to 2000 simulation ticks). It can be observed that computational performance reduces with increasing length of network refresh period. That is because every time the network is refreshed, routes of vehicles are recomputed. Since the amount of computation associated with routing is significant, the reduction in computation time is significant with increasing length of network refresh period.

Table 3: Variation of simulation time with network refresh period

| $T_{ref}$ (ticks) | Computation time (hours) |
|---|---|
| 500 | 9.29 |
| 1000 | 7.21 |
| 2000 | 5.96 |

## Performance with developed vs built-in VI module

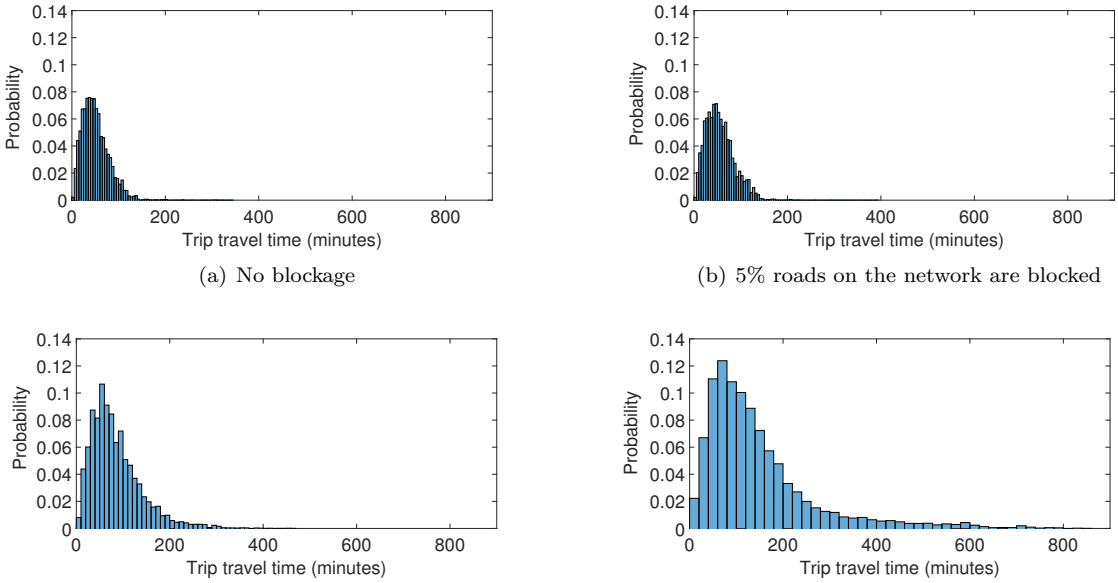
In this section, the computational performance of A-RESCUE 2.0 when using the developed VI module is compared to when the built-in VI module of Repast Simphony is used. Table 4 shows simulation computation for parameter setting times when no VI module is used, a separate VI module is used (connected through the simulator through data communication) and when built-in VI module is used. It can be seen that using the developed VI module reduces the additional computational load on simulation machine due to the use of visualization to 2% as compared to the corresponding 9% load for built-in visualization. This is because of the efficient data communication scheme and also because of the fact that VI module running on a separate machine reduces the computation and memory consumption on the simulation machine.

Table 4: Simulator computation time for different visualization settings

| Setting | Computation time (hours) | Percentage increase from no visualization |
|---|---|---|
| No visualization | 7.94 | - |
| Developed add-on VI module | 8.09 | 1.9% |
| Built-in VI module | 8.64 | 8.8% |

## Trip travel time analysis with varying network conditions

We now analyze the effect of road blockages on trip travel times of evacuees during hurricane evacuation. Such blockages can occur due to flooded roads, due to sudden influx of evacuees on roads at same time etc. As mentioned before, such blockages are introduced in A-RESCUE 2.0 using event handling scheme. We set the free-flow speeds of blocked roads to a small value in order to simulate blocking. Figure 12 and Table 5 present the distribution and average of trip travel times, respectively. We consider four scenarios: first setting does not contain any blocked road in the network and remaining




(a) No blockage

(b) 5% roads on the network are blocked

(c) 15% roads on the network are blocked

(d) 25% roads on the network are blocked

Figure 12: Distribution of trip travel time under different road blockage settings

three settings consider different proportions of network roads to be blocked. The blocked roads are chosen randomly across the network. In no blockage setting, free-flow speeds are solely governed by background traffic updates whereas in the remaining settings free-flow speeds are also governed by event handling scheme. It can be seen that with increasing blockage trip travel time distribution gets skewed towards higher travel time values. This is also confirmed by the increasing average travel time values with blockage percentage.

Table 5: Average trip travel time under different road blockage/congestion settings

| Network congestion setting | Average trip travel time (minutes) |
|---|---|
| No blockage | 50.7 |
| 5% roads on the network are blocked | 55.7 |
| 15% roads on the network are blocked | 84.6 |
| 25% roads on the network are blocked | 145.9 |

## Sensitivity of computation towards predictive network weighing scale parameters

As mentioned before, extensive experimental tests are carried out to finalize predictive network weighing parameters $\alpha$, $\beta$ and $\gamma$. It involves conducting an iterative process of varying one parameter while fixing the other two and repeating the process for all parameters until convergence is reached. Parameters that allow highest computational performance are finally chosen. The tests are conducted for both routing algorithms *SP* and *KSP*. Table 6 presents the variation of computation time for different parameter settings (bold parameters represent optimal parameters for the corresponding iteration). We proceed in the following order of varying parameters: $\gamma \rightarrow \alpha \rightarrow \beta \rightarrow \gamma$ and observe that convergence is reached.

Table 6: Variation of computation time for *SP* and *KSP* with different scale parameters settings (bold parameters represent optimal parameters for the corresponding iteration)

| Iteration | Parameters for SP $(\alpha, \beta, \gamma)$ | Computation time for SP (hours) | Parameters for KSP $(\alpha, \beta, \gamma)$ | Computation time for KSP (hours) |
|---|---|---|---|---|
| 1 | $2, 5, 2$ | 6.41 | $2, 5, 2$ | 8.91 |
| | $2, 5, 5$ | 6.70 | $2, 5, 5$ | 8.49 |
| | **2, 5, 10** | 6.37 | **2, 5, 10** | 8.47 |
| | $2, 5, 20$ | 6.64 | $2, 5, 20$ | 8.52 |
| 2 | $2, 5, 10$ | 6.37 | $2, 5, 10$ | 8.47 |
| | $5, 5, 10$ | 6.28 | $5, 5, 10$ | 8.40 |
| | $10, 5, 10$ | 6.58 | **10, 5, 10** | 8.09 |
| | **15,5,10** | 6.20 | $15, 5, 10$ | 8.16 |
| 3 | $15, 1, 10$ | 6.23 | $10, 1, 10$ | 8.35 |
| | **15,5,10** | 6.20 | **10, 5, 10** | 8.09 |
| | $15, 10, 10$ | 6.21 | $10, 10, 10$ | 8.84 |
| | $15, 15, 10$ | 6.21 | $10, 15, 10$ | 8.53 |
| 4 | $15, 5, 2$ | 6.22 | $10, 5, 5$ | 8.69 |
| | $15, 5, 5$ | 6.48 | **10, 5, 10** | 8.09 |
| | **15,5,10** | 6.20 | $10, 5, 15$ | 8.43 |
| | $15, 5, 20$ | 6.36 | $10, 5, 20$ | 8.35 |

# Conclusions and future directions

This paper develops A-RESCUE 2.0, a parallelizable large-scale agent-based traffic simulator that is capable of simulating large-scale traffic patterns during hurricane evacuation. Realistic household evacuation behaviors are captured using a series of evacuation decision making models. A parallelization scheme is introduced to improve the computational performance of the simulator. The scheme involves partitioning the network into subnetworks using a multi-level graph-partitioning algorithm such that computation corresponding to different subnetworks is handled in parallel by separate Java threads. Due to changing traffic conditions across the network, equal distribution of load across subnetworks is ensured through periodic repartitioning of the network. A predictive network weighing scheme is introduced that takes into account current and future loads during repartitioning. Weights corresponding to current and future load are fixed through extensive experimental tests to ensure minimal computational load. Also, an adaptive strategy determines the period of repartitioning based on current traffic load on the network. Experimental tests demonstrate significant improvement in computation time due to parallelization.

Vehicles can travel on the network through either of the two routing algorithms: Single shortest path (SP) and K shortest paths (KSP). An on-demand routing strategy is proposed that minimizes the overall routing computation in simulation. In addition, a background traffic scheme updates hourly free-flow speeds that takes into account traffic congestion due to non-evacuee traffic. The simulator is also capable of simulating events like road closures or blockages on roads. We also conduct trip travel time analysis for varying road blockages on the network and verify that average trip travel time increases with increasing blockage on the network.

An add-on visualization interface module is developed that allows real-time monitoring of traffic patterns across the network. This module communicates to the simulator using a network communication layer of WebSockets that routinely transmits vehicle trajectory information to the module. This layer also allows simulator and visualization module to work on separate machines, thereby distributing the overall computation. Computational tests suggests that the developed visualization and network communication layer add lesser computational load to the simulation machine as compared to using the built-in visualization module.

There are some limitations of this study that should be of interest while conducting future research: 1) In A-RESCUE 2.0, the event information like road blockage are predefined and are taken as input from the beginning. In order to simulate real time evacuation and enrich user experience, the data communication should also be able to communicate from visualization to simulator so that

users/evacuation officials prioritize certain road closures by observing the visualization interface; 2) A-RESCUE 2.0 assumes that all users update their routes in a uniform manner but that need not be true. Some users might not update their routes based on traffic conditions, some might update their routes less frequently and some might update their routes very frequently. The simulator should be extended to take into account multiple routing behaviors; 3) A-RESCUE 2.0 should be extended to take into account multiple vehicle types having different acceleration rates, deceleration rates and other characteristics. This would allow modeling of different types of vehicles like trucks, passenger cars etc; 4) On the traffic network side, there can be various traffic flow restrictions (e.g. contra-flow) and signal priorities for managing evacuation traffic efficiently. It will be particularly useful to test such traffic options in an integrated simulation system; 5) Currently the simulation system does not model how the warning information is propagated and processed and how it influences the evacuation decisions. Considering an underlying social network among the agents and modeling a dynamic decision-making context, such phenomena can be simulated.

## Acknowledgements

The authors are grateful to National Science Foundation for the award CMMI 1520338 to support the research presented in the paper. However, the authors are solely responsible for the findings presented in this study. The authors also thank Wenbo Zhang for his help in the development of background traffic part.

## References

Ahmed, K. I. (1999). *Modeling drivers' acceleration and lane changing behavior*. PhD thesis, Massachusetts Institute of Technology.

Balakrishna, R., Wen, Y., Ben-Akiva, M., and Antoniou, C. (2008). Simulation-based framework for transportation network management in emergencies. *Transportation Research Record: Journal of the Transportation Research Board*, 2041(1):80–88.

Balmer, M., Rieser, M., Meister, K., Charypar, D., Lefebvre, N., and Nagel, K. (2009). Matsim-t: Architecture and simulation times. In *Multi-agent systems for traffic and transportation engineering*, pages 57–78. IGI Global.

Blake, E. S., Kimberlain, T. B., Berg, R. J., Cangialosi, J. P., and Beven Ii, J. L. (2013). Tropical cyclone report: Hurricane sandy. *National Hurricane Center*, 12:1–10.

Cova, T. J. and Johnson, J. P. (2003). A network flow model for lane-based evacuation routing. *Transportation research part A: Policy and Practice*, 37(7):579–604.

Dash, N. and Gladwin, H. (2007). Evacuation decision making and behavioral responses: Individual and household. *Natural Hazards Review*, 8(3):69–77.

Fielding, R. (2000). Representational state transfer. *Architectural Styles and the Design of Netowork-based Software Architecture*, pages 76–85.

Franzese, O. (2001). Traffic modeling framework for hurricane evacuation. In *Proceedings of 80th Annual Meeting of TRB, Washington, DC, 2001.*

Gipps, P. G. (1986). A model for the structure of lane-changing decisions. *Transportation Research Part B: Methodological*, 20(5):403–414.

Han, L. D. and Yuan, F. (2005). Evacuation modeling and operations using dynamic traffic assignment and most desirable destination approaches. In *84th Annual Meeting of the Transportation Research Board, Washington, DC*, pages 2401–2406. Washington: Transporta tion Research Board.

Hansen, M., Petersen, M. N., Kokfelt, T. F., and Stensgaard, B. M. (2016). afieldwork-an android app for offline recording of geological information and data display. *Geol. Surv. Den. Greenl. Bull*, 35:99–102.

Hasan, S., Mesa-Arango, R., and Ukkusuri, S. (2013). A random-parameter hazard-based model to understand household evacuation timing behavior. *Transportation research part C: emerging technologies*, 27:108–116.

Hasan, S., Ukkusuri, S., Gladwin, H., and Murray-Tuite, P. (2010). Behavioral model to understand household-level hurricane evacuation decision making. *Journal of Transportation Engineering*, 137(5):341–348.

Herman, R., Montroll, E. W., Potts, R. B., and Rothery, R. W. (1959). Traffic dynamics: analysis of stability in car following. *Operations research*, 7(1):86–106.

Hobeika, A. G. and Jamei, B. (1985). Massvac: A model for calculating evacuation times under natural disasters. *Emergency Planning*, pages 23–28.

Hobeika, A. G. and Kim, C. (1998). Comparison of traffic assignments in evacuation modeling. *IEEE transactions on engineering management*, 45(2):192–198.

Jha, M., Moore, K., and Pashaie, B. (2004). Emergency evacuation planning with microscopic traffic simulation. *Transportation Research Record: Journal of the Transportation Research Board*, 1(1886):40–48.

Karypis, G. and Kumar, V. (1995). Metis–unstructured graph partitioning and sparse matrix ordering system, version 2.0. *Citeseer*.

Klefstad, R., Zhang, Y., Lai, M., Jayakrishnan, R., and Lavanya, R. (2005). A distributed, scalable, and synchronized framework for large-scale microscopic traffic simulation. In *Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE*, pages 813–818. IEEE.

Klunder, G., Terbruggen, E., Mak, J., and Immers, B. (2009). Large-scale evacuation of the randstad evacuation simulations with the dynamic traffic assignment model indy. *Citeseer*.

Lindell, M. K., Lu, J.-C., and Prater, C. S. (2005). Household decision making and evacuation in response to hurricane lili. *Natural Hazards Review*, 6(4):171–179.

Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., and Balan, G. (2005). Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527.

Macal, C. M. and North, M. J. (2005). Tutorial on agent-based modeling and simulation. In *Simulation Conference, 2005 Proceedings of the Winter*, pages 14–pp. IEEE.

Mesa-Arango, R., Hasan, S., Ukkusuri, S. V., and Murray-Tuite, P. (2012). Household-level model for hurricane evacuation destination type choice using hurricane ivan data. *Natural hazards review*, 14(1):11–20.

Mitchell, S. and Radwan, E. (2006). Heuristic priority ranking of emergency evacuation staging to reduce clearance time. *Transportation Research Record: Journal of the Transportation Research Board*, 1(1964):219–228.

Murray-Tuite, P. (2007). Perspectives for network management in response to unplanned disruptions. *Journal of urban planning and development*, 133(1):9–17.

Murray-Tuite, P. and Wolshon, B. (2013). Evacuation transportation modeling: An overview of research, development, and practice. *Transportation Research Part C: Emerging Technologies*, 27:25–45.

Noh, H., Chiu, Y.-C., Zheng, H., Hickman, M., and Mirchandani, P. (2009). Approach to modeling demand and supply for a short-notice evacuation. *Transportation Research Record: Journal of the Transportation Research Board*, 1(2091):91–99.

North, M. J., Howe, T. R., Collier, N. T., and Vos, J. R. (2005). The repast simphony runtime system. In *Proceedings of the agent 2005 conference on generative social processes, models, and mechanisms*, volume 10, pages 13–15. ANL/DIS-06-1, co-sponsored by Argonne National Laboratory and The University of Chicago.

Pidd, M., de Silva, F. N., and Eglese, R. W. (1993). Cemps: a configurable evacuation management and planning systema progress report. In *Proceedings of the 25th conference on Winter simulation*, pages 1319–1323. ACM.

Pingali, K., Nguyen, D., Kulkarni, M., Burtscher, M., Hassaan, M. A., Kaleem, R., Lee, T.-H., Lenharth, A., Manevich, R., Méndez-Lojo, M., et al. (2011). The tao of parallelism in algorithms. In *ACM Sigplan Notices*, volume 46, pages 12–25. ACM.

Ramamohanarao, K., Xie, H., Kulik, L., Karunasekera, S., Tanin, E., Zhang, R., and Khunayn, E. B. (2016). Smarts: Scalable microscopic adaptive road traffic simulator. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):26.

Rathi, A. K. and Solanki, R. S. (1993). Simulation of traffic flow during emergency evacuations: a microcomputer based modeling system. In *Simulation Conference Proceedings, 1993. Winter*, pages 1250–1258. IEEE.

Richmond, P. (2011). Flame gpu technical report and user guide (cs-11-03). *Department of Computer Science, University of Sheffield, Tech. Rep.*

Sadri, A. M., Ukkusuri, S. V., Murray-Tuite, P., and Gladwin, H. (2014). Analysis of hurricane evacuee mode choice behavior. *Transportation research part C: emerging technologies*, 48:37–46.

Sheffi, Y., Mahmassani, H. S., and Powell, W. B. (1980). *NETVAC1: A Transportation Network Evacuation Model*. Center for Transportation Studies, Massachusetts Institute of Technology.

Sherali, H. D., Carter, T. B., and Hobeika, A. G. (1991). A location-allocation model and algorithm for evacuation planning under hurricane/flood conditions. *Transportation Research Part B: Methodological*, 25(6):439–452.

Smith, L., Beckman, R., Anson, D., Nagel, K., and Williams, M. (1995). Transims: Transportation analysis and simulation system. Technical report, Los Alamos National Lab., NM (United States).

Soares, G., Macedo, J., Kokkinogenis, Z., and Rossetti, R. (2013). An integrated framework for multi-agent traffic simulation using sumo and jade. In *SUMO2013, The first SUMO user conference*, pages 15–17.

Steiniger, S. and Hunter, A. J. (2013). The 2012 free and open source gis software map–a guide to facilitate research, development, and adoption. *Computers, environment and urban systems*, 39:136–150.

Svennerberg, G. (2010). *Beginning Google Maps API 3*. Apress.

Ukkusuri, S. V., Hasan, S., Luong, B., Doan, K., Zhan, X., Murray-Tuite, P., and Yin, W. (2017). A-rescue: An agent based regional evacuation simulator coupled with user enriched behavior. *Networks and Spatial Economics*, 17(1):197–223.

Wessels, A., Purvis, M., Jackson, J., and Rahman, S. (2011). Remote data visualization through websockets. In *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on*, pages 1050–1051. IEEE.

Williams, B. M., Tagliaferri, A. P., Meinhold, S. S., Hummer, J. E., and Rouphail, N. M. (2007). Simulation and analysis of freeway lane reversal for coastal hurricane evacuation. *Journal of urban planning and development*, 133(1):61–72.

Wolshon, B. (2002). Planning for the evacuation of new orleans. *Institute of Transportation Engineers. ITE Journal*, 72(2):44.

Yang, Q., Koutsopoulos, H., and Ben-Akiva, M. (2000). Simulation laboratory for evaluating dynamic traffic management systems. *Transportation Research Record: Journal of the Transportation Research Board*, 1(1710):122–130.