

# Evacuation Route Planning: Scalable Heuristics

Sangho Kim  
Department of Computer  
Science  
University of Minnesota  
sangho@cs.umn.edu

Betsy George  
Department of Computer  
Science  
University of Minnesota  
bgeorge@cs.umn.edu

Shashi Shekhar  
Department of Computer  
Science  
University of Minnesota  
shekhar@cs.umn.edu

## ABSTRACT

Given a transportation network, a vulnerable population, and a set of destinations, evacuation route planning identifies routes to minimize the time to evacuate the vulnerable population. Evacuation route planning is a vital component of efforts by civil authorities to prepare for both natural and man-made disasters (e.g., hurricanes, terrorist acts, etc). However, evacuation route planning is computationally challenging due to the size of transportation networks, the large number of evacuees, and capacity constraints. For example, the number of evacuees often far exceeds the bottleneck capacity, i.e., the minimum cut of a given network. Current approaches (e.g., linear programming and Capacity Constrained Route Planner (CCRP), a recently proposed evacuation planning algorithm) do not scale well because of intensive computation needs in order to produce the schedules of evacuees as well as routing plans.

This paper presents innovative heuristics scalable to very large transportation networks. The Intelligent Load Reduction heuristic accelerates the routing computation by the reduction of evacuees using the bottleneck saturation. The performance of Intelligent Load Reduction is evaluated using real world scenarios. Results show that the Intelligent Load Reduction heuristic significantly improve the runtime of CCRP. We propose another heuristic named Incremental Data Structure. While the Intelligent Load Reduction gains performance increase by giving up the schedules of evacuees, the Incremental Data Structure heuristic can reduce calculation time of the CCRP algorithm by the enhanced data structures without affecting the outputs.

## Categories and Subject Descriptors

F.2.0 [Theory of Computation]: Analysis of Algorithms and Problem Complexity—*General*

## General Terms

Algorithms, Experimentation

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ACMGIS '07, November 7-9, 2007, Seattle, WA

Copyright 2007 ACM ISBN 978-1-59593-914-2/07/11 ...\$5.00.

## Keywords

evacuation route planning, spatial network, algorithm

## 1. INTRODUCTION

Given a transportation network with capacity constraints, the initial occupancies and destination nodes, evacuation route planning generates a set of evacuation routes and a schedule for the movement of people and vehicles along these routes, such that the evacuation is completed in the shortest possible time. This is a critical step in disaster emergency management and homeland defense preparation. The recent catastrophes caused by hurricanes on the Gulf coast underscore the importance of evacuation route planning. Route planning is computationally challenging because the number of evacuees often far exceeds the capacity, *i.e.* the number of people that can move along the road segments in a unit time regardless of transportation method. Also, route planning algorithms deal with transportation networks that are very large.

Research in the area of Evacuation Route Planning fall into three categories: (1) Linear Programming methods that generate optimal evacuation plans which minimize the total evacuation time [2, 5, 6, 8, 12, 13], (2) Simulation methods that models traffic flow at single vehicle level [3, 15], and (3) Heuristic methods such as the Capacity Constrained Route Planner algorithm [14]. Linear programming based approaches use flow network algorithms to evaluate the routes. The transportation network is transformed into a time-expanded network by duplicating the original evacuation network  $G$  for each discrete time unit  $t = 0, 1, \dots, T$ . Then, it defines the evacuation problem as a minimum cost network flow problem [1, 4] on the time-expanded network  $G_T$ . Finally, it feeds the expanded network  $G_T$  to minimum cost network flow solvers, such as NETFLO [11], to find the optimal solution. Although these evacuation planning algorithms generate optimal plans, they are expensive with respect to memory and take a long time to solve problems of the sizes usually encountered in urban evacuation scenarios. This method requires a prior knowledge of an upper bound on the evacuation time  $T$  to generate the time-expanded network, which might be hard to estimate precisely. An under-estimated bound  $T$  would result in a failure to reach a solution, whereas, an overestimated value for  $T$  would result in an over-expanded network, leading to unnecessary storage and run-time. In practice, LP based approaches have been used in scenarios that involve small sized networks such as, in small building evacuations.

Simulation methods assume that the behavior of individ-

ual drivers is under the influence of vehicles in their proximity. This simulation models are often accompanied with labor intensive network coding and significant running time, making it difficult to take advantage of spatial databases or easily compare alternative configurations. Thus, they may be inappropriate for large evacuation scenarios.

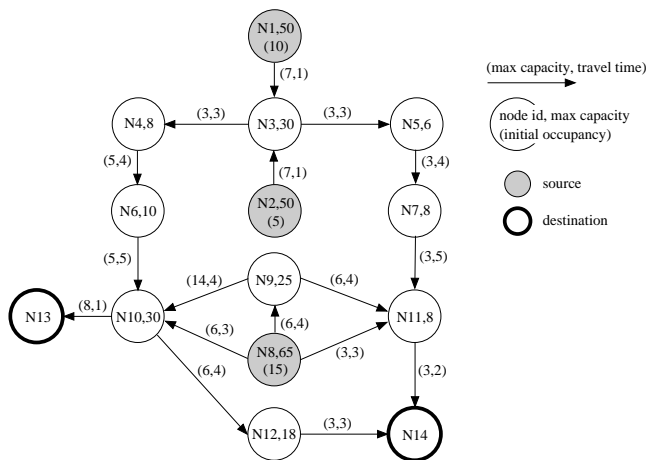
Though the methods in the third category based on heuristics do not always generate optimal evacuation routes, they have been able to reduce the computational cost of the process dramatically. A well-known approach that falls in this category is the Capacity Constrained Route Planner (CCRP) [14]. This method makes use of well known shortest path algorithms and extends them by incorporating capacity constraints. It models capacity as a time series to account for the time dependent nature of the networks. It uses only the original evacuation network instead of the time-expanded network used by the LP based approach and thus requires less memory. Although this method has been effectively used in medium-sized networks, it might not scale up efficiently to an evacuation scenario that involves millions of evacuees and a transportation network of hundreds of thousands of nodes (e.g., evacuation of the city of Houston during Hurricane Rita in September, 2005). For example, CCRP took more than a day of computation time to identify evacuation routes for more than a million people within a 300 square mile area of Twin Cities in Minnesota.

The existing methods, optimal, simulation and heuristic do not provide adequate scalability required in some exceptionally large evacuation scenarios. The computation bottleneck in the heuristic algorithm has been identified as the repeated shortest path computations to find the evacuation routes, subject to the capacity constraints of the network. To address this issue, this paper proposes heuristics scalable to large networks. The Intelligent Load Reduction (ILR) is based on a load reduction technique to skip multiple iterations in CCRP that do not serve to find new routes. The ILR approach exploits the saturation of edges on the min-cut partitioning (called the bottleneck edges) of the network. Analysis shows that this approach improves the scalability of the algorithm with respect to the number of evacuees. However, the ILR heuristic does not compromise the full schedules of evacuees due to the skipping mechanism exploited in the algorithm. Our preliminary experimental results show that the runtime can be reduced by up to 95% while its solution quality (i.e., evacuation time by the routes from the algorithm) is comparable to that of CCRP.

To overcome the limitation of ILR, we are developing another heuristic called the Incremental Data Structure (IDS) that reduces the computational cost of the evacuation planning algorithm by reducing the total cost of shortest path computations. This is achieved through the reuse of parts of shortest path trees between successive iterations. This approach makes the heuristic method scalable to large networks. The IDS heuristic produces exactly same results with CCRP because the IDS approach is an optimization technique using advanced data structures and algorithms.

**Scope and Outline of the Paper** The paper proposes two evacuation route planning heuristics that are scalable to large problem sizes. The heuristics are based on macroscopic traffic simulation model and focused on reducing the computational effort involved in repeated shortest path computations.

The rest of the paper is organized as follows. Section 2



**Figure 1: Modeling of simple building evacuation situation using graph**

provides the problem definition and an outline of the existing solutions. Section 3 describes the ILR heuristic based on the bottleneck saturation. Section 4 provides the description of IDS heuristic that reuses previously computed shortest paths. We are planning to perform experimental evaluations on IDS algorithm as of writing. Section 5 outlines the conclusions and the future work.

## 2. PROBLEM DEFINITION / SOLUTIONS

In transportation science, there are various ways (e.g., microscopic, mesoscopic, and macroscopic) to interpret and formulate an evacuation situation. We decided to use the macroscopic model using mathematical graphs (i.e., flow network) to describe the evacuation situation due to its increased public attention, improved techniques and computational capacity [7]. A precise formulation of the research problem on evacuation route planning is as follows.

- Given:** A transportation network with
- (1) integer capacity constraints on nodes and edges,
  - (2) integer travel time on edges,
  - (3) number of evacuees and their initial locations,
  - (4) locations of evacuation destinations.
- Find:** An evacuation plan consisting of a set of origin destination routes and a scheduling of evacuees on each route to minimize evacuation egress time.
- Objective:** Minimize the computational cost of producing the evacuation route plan.
- Constraint:**
- (1) The scheduling of evacuees on each route should observe the capacity constraints.
  - (2) Edge travel time preserves First-In First-Out.
  - (4) Limited amount of computer memory.

The problem definition described above is illustrated with a situation of a simple building evacuation in Figure 1. Each node is shown by a circle with two attributes: maximum node capacity and initial node occupancy. For example, at node N1, the maximum capacity is 50, which indicates that this node can hold at most 50 evacuees at any time instant. If the node capacity is not meaningful (e.g., node as a city

**Table 1: Example evacuation plan based on the evacuation network in Figure 1**

Source	# of Evacuees	Route with Schedule node_id (arrival_time) – node_id (arrival_time) – ...	Destination Arrival Time
N8	6	N8 (0) – N10 (3) – N13 (4)	4
N8	6	N8 (1) – N10 (4) – N13 (5)	5
N8	3	N8 (0) – N11 (3) – N14 (5)	5
N1	3	N1 (0) – N3 (1) – N4 (4) – N6 (8) – N10 (13) – N13 (14)	14
N1	3	N1 (0) – N3 (2) – N4 (5) – N6 (9) – N10 (14) – N13 (15)	15
N1	1	N1 (0) – N3 (1) – N5 (4) – N7 (8) – N11 (13) – N14 (15)	15
N2	2	N2 (0) – N3 (1) – N5 (4) – N7 (8) – N11 (13) – N14 (15)	15
N2	3	N2 (0) – N3 (3) – N4 (6) – N6 (10) – N10 (15) – N13 (16)	16
N1	3	N1 (1) – N3 (2) – N5 (5) – N7 (9) – N11 (14) – N14 (16)	16

or an intersection), it can be set to infinity. The initial occupancy is shown to be 10, which means there are 10 evacuees at this node when the evacuation starts. Each edge, shown as an arrow, represents a link between two nodes. Each edge also has two attributes: maximum edge capacity and travel time. For example, at edge N1-N3, the maximum edge capacity is 7, which means at each time point, at most 7 evacuees can start to travel from node N1 to N3 through this link. The travel time of this edge is 1, which means it takes 1 time unit(s) to travel from node N1 to N3. The time unit is defined according to the requirement of precision in evacuation time (e.g., minute or hour). The example in Figure 1 has 10 evacuees at node N1, 5 at node N2, and 15 at node N8. The goal is to compute an evacuation plan that evacuates the 30 evacuees to the two destinations (thick lined, nodes N13 and N14) using the least amount of time.

Table 1 shows an example evacuation plan for the evacuation network in Figure 1. Each row in the table describes the schedule of a group of evacuees moving together with a series of node IDs and arrival time of each node. Take source node N8 for example; initially there are 15 evacuees at N8. They are divided into 3 groups: 6, 6 and 3 people. The first group starts from node N8 at time 0 and moves to node N10, then moves from node N10 at time 3 to node N13, and reaches destination N13 at time 4. The second group follows the same route of the first group, but has a different schedule due to the capacity constraints of this route. The second group moves from N8 at time 1 to N10, then moves from N10 at time 4 to N13, and reaches destination N13 at time 5. The third group takes a different route. It starts from N8 at time 0 and moves to N11, then moves from N11 at time 3 to N14, and reaches destination N14 at time 5. The procedure is similar for other groups of evacuees from source node N1 and N2. The whole evacuation egress time is 16 time units since the last groups of people reach their destination at time 16. This evacuation plan is an optimal plan for the evacuation scenario shown in Figure 1.

**Existing Solutions:** Various approaches to solve the evacuation route planning problem have been proposed in domains such as transportation science, mathematics and computer science. One of the most widely discussed is the Linear Programming (LP) approach. In order to apply an LP approach, the evacuation network needs to be converted to a time expanded network [6] which consists of multiple copies of the same network along the time dimension. The multiple copies are connected by an edge cost (e.g., travel time). In this approach, the evacuation route planning problem then becomes a minimum cost flow problem. The following factors, however, hinder application of the LP approach to large transportation networks. First, the use of

time expanded networks significantly increases the problem size. If we solve a problem with 1,000 nodes and a maximum 300 minute evacuation time, the time expanded network will consist of 300,000 nodes. In addition, building a time-expanded network requires guessing the upper bound of the evacuation time. If the upper bound is over-estimated, a significant waste of calculation time and memory is expected. If the upper bound is under-estimated, it will lead to the failure of finding a solution.

Simulation methods are also widely used in civil engineering. The methods describe traffic flow at vehicle level, modeling the interaction between cars with car-following models. Theodoulou and Wolshon [17] used CORSIM microscopic traffic simulation to model the freeway evacuation around New Orleans. With the help of a micro scale traffic simulator, they were able to suggest alternative evacuation routes in a detailed manner. However, their microscopic simulation model requires a labor intensive network coding using aerial photographs and significant running time for each scenario, making it difficult to take advantage of spatial databases or easily compare alternative route sets. Evacuation route planning with other microscopic traffic simulation (e.g., MITSIMLab [9]) have shown similar limitations.

To overcome the limitations of LP methods as well as simulation methods, [14] proposed the Capacity Constrained Route Planner (CCRP). While the LP approaches maintain available capacity over the edges of the time expanded graph, the CCRP maintains the available capacity information using a time series data structure attached to each edge. The CCRP makes use of well known shortest path algorithms and extends them by incorporating capacity constraints using the available capacity information. The Algorithm 1 is an overall algorithm structure of the CCRP. In each iteration of *while* loop, the algorithm first searches for the route with the earliest arrival time from any source to any destination, taking available edge capacity into consideration. The next step finds the minimum flow along the path found which is equivalent to the size of a group of evacuees who travel through the path. Then, the last step reserves the minimum flow. It is a task of updating available capacity.

---

**Algorithm 1** Capacity Constrained Route Planner

---

- 1: add super source node;
  - 2: **while** any source node *s* has evacuees **do**
  - 3: calculate earliest arrival path;
  - 4: calculate minimum flow along the path;
  - 5: reserve the minimum flow;
  - 6: **end while**
- 

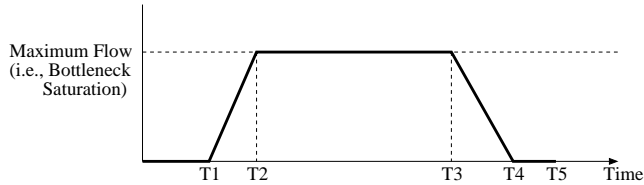
### 3. INTELLIGENT LOAD REDUCTION

The Intelligent Load Reduction (ILR) is designed to reduce the load (i.e., number of evacuees) of a given evacuation scenario to reduce the run time. The load reduction heuristic is based on insights related to the properties of evacuation route computation. The load reduction heuristic finds only evacuation routes regardless of evacuation schedule. Before we go into the details of the Intelligent Load Reduction (ILR) heuristic, it would be useful to briefly explore the background of the proposed heuristic.

#### 3.1 Background Information

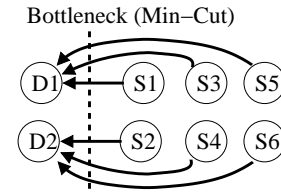
**Bottleneck Saturation:** In graph theory, a given network can be divided by a cut, a partition of vertices into two sets. It is natural to introduce the concept of 'cut' to the evacuation problem because the evacuation situation is described as flows from a set of sources to a set of destinations. The max-flow min-cut theorem states that "The maximal amount of flow is equal to the capacity of a minimal cut" [4, 10]. In other words, the maximum flow is dictated by the bottleneck of a given network. Identifying a bottleneck (i.e., min-cut, or a set of edges whose sum of capacity is minimum) of a specified network inside an evacuation zone is useful to predict congested road segments and analyze the behavior of network flows during the evacuation process.

Figure 2 illustrates the bottleneck (i.e., min-cut) saturation by flows during an evacuation process. That is, the available bottleneck capacity is filled by flow (e.g., moving vehicles) as evacuees move across the bottleneck boundary. During the period from 0 to  $T_1$ , the bottleneck has 0 flow because it takes time for evacuees to reach the bottleneck edges from the sources. During the period from  $T_1$  and  $T_2$ , evacuees go through the bottleneck edges using different paths and the saturation level goes up. After time  $T_2$ , the bottleneck is fully saturated. Although evacuees may find detour routes from sources to the bottleneck edges, the evacuees will still likely encounter congestion between  $T_2$  and  $T_3$ . This fully saturated period is proportional to the load (i.e., number of evacuees) of a given scenario. During the period from  $T_3$  and  $T_4$ , the congestion through the bottleneck is mitigated. During the period from  $T_4$  and  $T_5$ , remaining evacuees leave the bottleneck edges and reach their destinations.



**Figure 2: The behavior of flows with regard to bottleneck saturation**

**Why Does Naive Early Termination Not Work?** Suppose that we are only interested in finding a set of evacuation routes regardless of the full evacuation schedule from time 0 to  $T_5$ , as shown in Figure 2. One may guess that it would be possible to acquire the evacuation routes if we stop the CCRP algorithm at time  $T_2$  because almost no routes are found after  $T_2$  due to the fully saturated bottleneck. We initially designed this heuristic based on the early termination approach at time  $T_2$ . However, the following problem emerged when we tried this approach. Suppose that the evacuees in six sources go to the two destination nodes shown in Figure 3. The flows from  $S_1$  and  $S_2$  may fill the bottleneck in the early phase because the mechanism of the CCRP algorithm is based on earliest arrival path finding. If we stop CCRP at time  $T_2$  in the bottleneck saturation graph, only source nodes close to the bottleneck (e.g.,  $S_1$ ,  $S_2$ ) serve to fill the cut. This phenomenon hampered the correctness of the early termination approach. Thus, we needed more intelligent techniques to reduce the iterations of CCRP while preserving the correctness of the algorithm.



**Figure 3: The problem of naive early termination approach**

### 3.2 Load Reduction Formula

A parameter to measure the global load of a given evacuation scenario is defined. The *overload\_degree* is the ratio between the number of evacuees and the bottleneck capacity. A scenario with many initial evacuees or low bottleneck capacity is considered a problem with high computational load because the bottleneck saturation period is long, resulting in long scheduling calculation.

$$overload\_degree = \# \text{ of evacuees} / \text{bottleneck capacity}$$

Another parameter is *node\_degree\_with\_flow*. This is a local information pertinent to each source node. This parameter determines if a source has diverted routes to destinations. For example, if a source has *node\_degree\_with\_flow* 3, then the source has a higher chance of having more detour routes than those with value 1.

$$node\_degree\_with\_flow(n) = \# \text{ of edges from a node } n \text{ with flow history}$$

The following is a load reduction formula using the global parameter *overload\_degree* and local parameter *node\_degree\_with\_flow*:

$$occupancy(n) \leftarrow occupancy(n) / (overload\_degree \times node\_degree\_with\_flow(n))$$

When the load (i.e., number of remaining evacuees at each source node) is reduced, the global and local parameters are used as denominators. If the given scenario has a heavy load, the parameter *overload\_degree* can aggressively cut down the load. Thus, our intelligent load reduction heuristic can be less sensitive to the load. The local parameter *node\_degree\_with\_flow* means that when a source has a larger outdegree (i.e., the number of edges from the source) with flow history, there is a higher chance that the source node has alternative detours. Thus, the load of the source needs to be more aggressively reduced.

### 3.3 Algorithm

Algorithm 2 is a pseudocode in which the ILR is applied to the CCRP algorithm. The basic structure of the algorithm is similar to that of CCRP, but the bottleneck saturation checking and load reduction formula are added. The set variable *used\_source\_set* keeps track of a set of sources that serve to fill the bottleneck until the bottleneck is saturated (line 11). The load of the sources in the *used\_source\_set* is reduced (line 12, 13, and 14) by the load reduction formula. When the bottleneck is saturated, the available capacity of the edges in  $G$  is reset (line 15) so that other sources can have

a chance of filling the bottleneck edges. While the CCRP calculates the full schedule of the evacuation process, our algorithm using the ILR technique can skip many iterations that do not produce new routes.

---

**Algorithm 2** CCRP with Intelligent Load Reduction

---

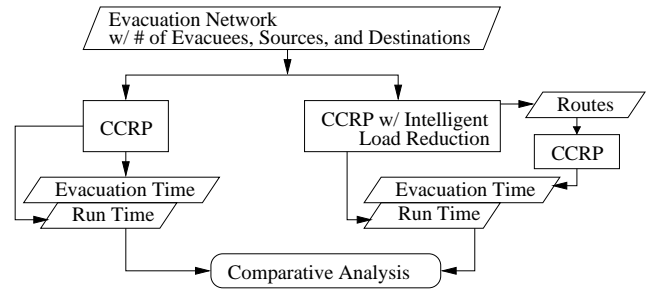
- 1:  $bottleneck \leftarrow \text{mincut}(G)$ ;
  - 2:  $overload\_degree = total\_occupancy / bottleneck$ ;
  - 3:  $used\_source\_set \leftarrow null$ ;
  - 4: add super source node to  $G$ ;
  - 5: **while** any source node has evacuee **do**
  - 6: calculate earliest arrival path;
  - 7: calculate minimum flow along the path;
  - 8:  $used\_source\_set \leftarrow$  insert the root of the path;
  - 9: reserve the minimum flow;
  - 10:  $saturation \leftarrow (bottleneck - available\_capacity(\text{mincut\_edges}(G))) / bottleneck$ ;
  - 11: **if**  $saturation == 1.0$  **then**
  - 12: **for all** node  $s$  in  $used\_source\_set$  **do**
  - 13:  $occupancy(s) \leftarrow occupancy(s) / (overload\_degree \times node\_degree\_with\_flow(s))$ ;
  - 14: **end for**
  - 15: reset available capacity of edges in  $G$  to max capacity of edges;
  - 16:  $used\_source\_set \leftarrow null$ ;
  - 17: **end if**
  - 18: **end while**
  - 19: **return** edges with flow history;
- 

### 3.4 Experiment

**Experiment Setup:** The computational environment in which we performed our experiments consisted of a Pentium 4 CPU with 1GB memory running Windows. The software was implemented in C++ language with Microsoft Visual Studio. In this experiment, two different types of real world networks were prepared. The first dataset was a virtual scenario of a failure at the Monticello nuclear power plant located to the northwest of Minneapolis-St. Paul, Minnesota. The sources were cities inside a 10-mile radius around the nuclear power plant as defined by Minnesota Homeland Security and Emergency Management [16]. This scenario had many sources but a single destination because evacuees needed to pass radiation clearance check at a specified facility. This special case resulted in a very limited number of bottleneck edges around the destination. Thus, the choking edges of this network occurred at the road segments around the destination. The second dataset consisted of Minneapolis downtown scenarios with evacuation zones of different radii. The main purpose of this dataset was to evaluate the scalability of the proposed heuristics.

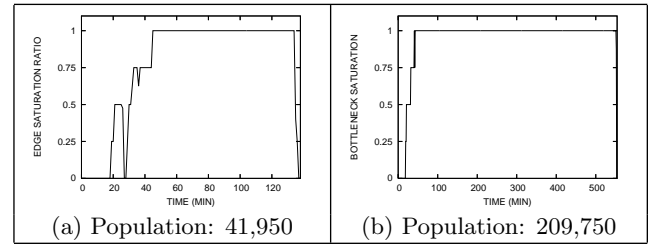
Figure 4 illustrates the overall experiment design. The datasets of the Monticello and Minneapolis downtown networks with the evacuees, sources, and destinations are the input to the system. The primary metrics are evacuation time and runtime. Evacuation time is used for the comparison of solution quality, and runtime is used for the comparison of performance. In the case of CCRP with ILR, the only output from the heuristic is routes. Thus, we have to run the original CCRP against the output routes to measure the solution quality (i.e., evacuation time) of CCRP with ILR.

**Results:** It is worthwhile to observe the bottleneck saturation patterns of the given datasets. Figure 5 illustrates



**Figure 4: Experiment Design**

the bottleneck saturation patterns of the Monticello scenario with different numbers of evacuees. The original number of evacuees of the scenario was 41,950 based on census data. We purposely increased the number of evacuees by five times to see the effect of the difference in load. The Monticello scenario showed patterns similar to those illustrated in Section 3.1. In Figure 5(a), the bottleneck saturation is 0 between time 0 and 20 minutes because it takes time for evacuees to travel from the sources to the bottleneck edges. Then, the evacuees gradually fill up the bottleneck capacity until time 45. After the flow fills the bottleneck capacity, the saturated period lasts for two thirds of the entire evacuation process time. When the evacuation process ends, the saturation drops fast. In Figure 5(b), the fully saturated period becomes longer as the size of the population increases.



**Figure 5: Bottleneck saturation patterns [dataset: Monticello scenario]**

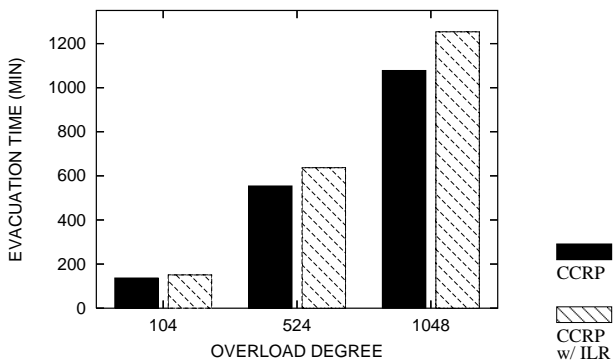
Table 2 is the runtime comparison between original CCRP and CCRP with the ILR heuristic with regard to different loads. The overload (e.g., 104) is the total number of evacuees (e.g., 41,950) divided by the bottleneck capacity (e.g., 403). One observation is that as the overload becomes larger, the performance gain becomes larger because the portion of additional overheads inside CCRP with ILR (e.g., bottleneck calculation and load reduction) gets relatively smaller as the load increases. Another observation is that the runtime of CCRP with ILR did not change between the overload 524 and 1048. This means that the proposed heuristic is not sensitive to the change of load.

The second metric to compare is the evacuation time with regard to different overload degrees. The evacuation time is linearly proportional to the overload degree. One observation is that as the overload becomes larger, the evacuation time gap between original CCRP and CCRP with ILR becomes bigger. This pattern can be interpreted as follows. As the proposed heuristic is insensitive to the load change, the number of iterations that are skipped by the heuristic in

**Table 2: Runtime reduction by Intelligent Load Reduction (ILR) [dataset: Monticello scenario]**

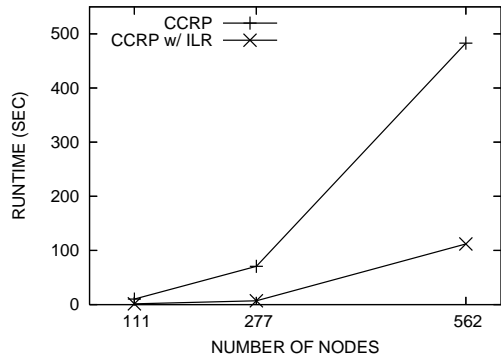
Overload Degree	104	524	1048
Runtime of CCRP (sec)	1.49	8.45	21.94
Runtime of CCRP w/ ILR (sec)	0.64	1.08	1.08
Runtime Reduction %	57%	87%	95%

comparison with original CCRP gets larger. In other words, the probability of finding the routes using the ILR heuristic that exist in the results from original CCRP gets lower as the load increases. There is an obvious tradeoff between performance gain and solution quality. However, it seems that the quality degradation (e.g., around 20% at overload 1048) by the heuristic is within a reasonable range in comparison with performance gain (e.g., 95%) when we increased the load up to 10 times from the original load.



**Figure 6: Evacuation time comparison [dataset: Monticello scenario]**

The last experiment using the ILR heuristic was the scalability test. We used the Minneapolis downtown scenarios with different network sizes. As the network size becomes larger, the runtime quadratically increases. Due to the increment pattern, the performance gain becomes larger as the given network size becomes larger. Thus, the scalability of the proposed heuristic is obvious in this experiment.



**Figure 7: Scalability results by ILR [dataset: Minneapolis scenario]**

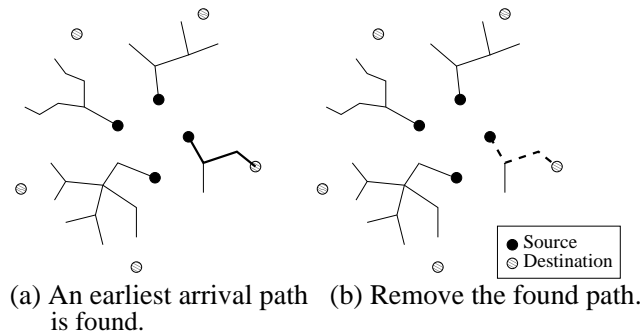
## 4. INCREMENTAL DATA STRUCTURE

The Incremental Data Structure (IDS) heuristic reduces the number of redundant computations between path calculations inside the existing CCRP algorithm. The heuristic is based on the optimization of data structures. This incremental data structure approach produces both evacuation routes and schedules.

### 4.1 Background Information

Although the CCRP significantly improves the performance of previous LP approaches, we are still interested in reducing the runtime of CCRP to make it more scalable for the evacuation scenarios based on very large transportation networks and very large population sizes. In the CCRP algorithm, "calculate earliest arrival path" is the most expensive processing routine because it is called multiple times by the *while* loop under the capacity constraints (see Algorithm 1). In the current implementation of CCRP, the expensive routine creates, initializes and destroys data structures necessary for the calculation of earliest arrival paths. The components of the required data structures are similar to those of Dijkstra's algorithm: node arrival time, parent pointer, and flag for 'visited' mark.

The approach we propose in this section is based on the following insightful idea. When we calculate a shortest path, the data structure (e.g., simple array, binary heap, Fibonacci heap, and double bucket implementation [18]) grows from the sources toward the destinations in a tree shape as shown in Figure 8. When the tree is fully grown enough to touch any destination, the algorithm stops because we are only interested in the earliest arrival path from any source to any destination. In the next iteration, a similar tree is grown from scratch. The only difference is the change of available capacity along the path found in the previous iteration (i.e., dotted line shown in the Figure 8). Our idea is to remove only the earliest arrival path found in the previous iteration and reuse the grown tree data structures. This idea does not affect the correctness of the CCRP algorithm because the change in available capacity only occurs along the path found. This incremental data structure, however, causes the following two problems.



**Figure 8: Insightful idea to reduce redundant calculations in multiple calls of shortest path in CCRP**

### 4.2 Orphan Branch Problem

If we remove the path that is found in the previous iteration, orphan branches are left, as shown in Figure 9. If we do not remove the orphan branches, they will grow like

other branches which have roots at the source nodes. We can consider two approaches to clean up the orphan branches. First is a brute force method in which each node in the network is scanned and checked to determine whether it is connected to any source using a parent pointer. This method may not be effective, however, for either very large networks or database driven networks (i.e., where nodes are not in memory buffer). The second method involves using descendant pointers. When the tree is grown, each node maintains descendant pointers as well as a parent pointer. The descendant pointers point to the neighbors of a node whose level is under the node in the tree. When we remove the path found, we check each node along the path. If it has descendant pointers, we can remove the orphan branches following the descendant pointers. The maximum number of descendant pointers may be small because a transportation network usually has a vertex degree of up to 5 (i.e., the maximum number of road segments from an intersection).

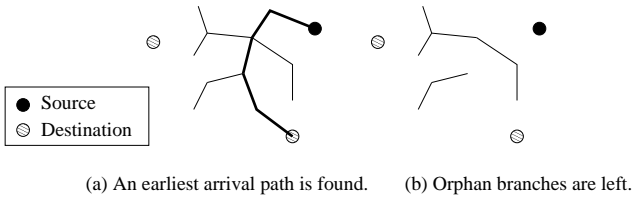


Figure 9: Orphan branch problem after path removal

### 4.3 Non-growing Source Problem

Another critical problem that occurs after the path removal step is a non-growing source problem. Figure 10-(a1), (a2), (a3) shows the steps of finding a shortest path using Dijkstra’s algorithm on a simple network with two sources S1 and S2 and a destination D1. Figure 10(a2) shows a state after a shortest path (i.e., S2-N2-N3-D1) is found. Each node has a node arrival time and parent pointer information which have been used to calculate the shortest path. The path is removed in Figure 10(a3) to perform the incremental search. Then, a part of the tree, S1-N1, remains, but permanently loses a chance to grow again because both nodes S1 and N1 in the partial tree are marked as ‘visited’ with thick line in Figure 10(a3). That is, source S1 is rendered a kind of non-growing source after this step.

To resolve the non-growing source problem, we introduce a sorted list data structure for each node to maintain the history of the node arrival time updates while calculating Dijkstra’s shortest path. In the traditional Dijkstra’s algorithm, a distance value is updated only when the new value is shorter than the old value. However, our approach inserts the new value into the sorted list (in increasing order by node arrival time) without the comparison used in Dijkstra’s algorithm. As shown in Figure 10(b2), node N3 has a sorted list (4,N2) and (5,N1) in the format of (node arrival time, parent pointer). When the tree is expanded, the first element in the sorted list, i.e., least node arrival time, is used to calculate the node arrival times around the neighborhood nodes. Similarly, when the found path is removed, the elements at the front of the sorted list are removed as shown in Figure 10(b3). We observe that the grown history from node N1 remains at node N3 in the Figure 10(b3). In

the next iteration, the source S1 has a chance to grow again using the remaining history data as shown in Figure 10(b4).

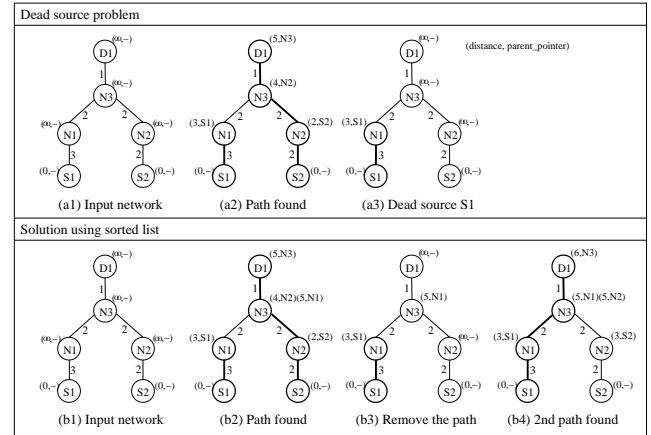


Figure 10: Non-growing source problem and solution

### 4.4 Algorithm

Algorithm 3 summarizes the changes in “calculate earliest arrival path” routine in the CCRP algorithm. The set  $Q$  and  $S$  describe ‘unvisited’ and ‘visited’ nodes in the graph. The element inserted into the sorted list follows the form of (node arrival time, parent pointer). The incremental data structures used in the Algorithm 3 (i.e., sorted list, bitmap for visited mark) are created and initialized outside of the while loop in Algorithm 1 for the sake of continuity.

#### Algorithm 3 Earliest Arrival Path using Incremental Data Structures

- 1: remove the route found in previous iteration;
- 2: orphan branch clean up;
- 3: **while**  $u$  is not a destination **do**
- 4:  $u := \text{Extract\_Min}(Q)$ ;
- 5:  $S := S \cup u$ ;
- 6: **for all** edge( $u, v$ ) outgoing from  $u$  **do**
- 7:  $t := \text{earliest\_edge\_entry\_time}(u, v)$  by capacity constraints;
- 8: insert  $(t + \text{edge\_travel\_time}(u, v), u)$  into sorted list;
- 9: **end for**
- 10: **end while**

## 5. CONCLUSION AND FUTURE WORK

In this paper, two scalable heuristics for the calculation of evacuation route planning are presented. Today’s evacuation route planning often involves large transportation networks (e.g., interstate highways across several states during hurricane evacuation) and large numbers of evacuees (e.g., millions of evacuees from several cities). Thus, the computational challenges for the solutions have been critical issues. Previous linear programming methods produce optimal plans for small size scenarios. Heuristic methods such as CCRP produce sub-optimal plans for medium size scenarios. However, there is an obvious need for scalable evacuation route planning heuristics along with further advances

in geographic information science, contemporary computing power and continuous research in risk management.

The two scalable heuristics are focused on improving the performance of an existing heuristic, the Capacity Constrained Route Planner (CCRP), developed by the University of Minnesota. The first heuristic, named Intelligent Load Reduction, is engineered for evacuation routes. It does not produce full evacuation schedules. The heuristic uses global (i.e., bottleneck saturation) and local (i.e., sources' outdegree) information to reduce the load on sources. This heuristic shows a tradeoff between performance and solution quality. The experimental validation of the presented ILR shows considerable performance gains over the existing CCRP algorithm. The second heuristic, named Incremental Data Structure, is a set of optimization techniques for CCRP. It exploits incremental data structures to avoid the redundant calculation required during the multiple calls of the shortest path function in the CCRP algorithm. Like CCRP, it can produce both evacuation routes and schedules.

Future work should include the following tasks on evacuation route planning algorithms: experimental evaluation on IDS, analytical validation of the proposed heuristics (e.g., runtime analysis), memory consumption experiments, experimental evaluation on various evacuation scenarios (e.g., geometry variations and very large scenarios), including temporal dynamics during evacuation (e.g., change in hurricane direction or toxic plume propagation), and implementation on spatial databases to effectively handle large datasets. Especially, we conjecture that both approaches are independent and highly likely to be combined to have better performance.

## 6. ACKNOWLEDGEMENTS

We are particularly grateful to members of the Spatial Database Research Group at the University of Minnesota for their helpful comments and valuable discussions. We would like to thank Dr. Wee-Liang Heng at ESRI for his initiative ideas on the incremental data structure. Kim Koffolt helped increase the readability of our paper.

## 7. REFERENCES

- [1] R. Ahuja, T. Magnanti, and J. Orlin. Network Flows: Theory, Algorithms, and Applications. *Prentice Hall*, 1993.
- [2] L. Chalmet, R. Francis, and P. Saunders. Network Model for Building Evacuation. *Management Science*, 28:86–105, 1982.
- [3] M. -A. et al. Development of a Deployable Real-Time Dynamic Traffic Assignment System: DynaMIT and DynaMIT-P User's Guide. *Massachusetts Institute of Technology*, 2002.
- [4] L. Ford and D. Fulkerson. Flows in Networks. *Princeton University Press*, 1962.
- [5] R. Francis and L. Chalmet. A Negative Exponential Solution To An Evacuation Problem. *Research Report No.84-86, National Bureau of Standards, Center for Fire Research*, October 1984.
- [6] H. Hamacher and S. Tjandra. Mathematical Modeling of Evacuation Problems: A State of the Art. *Pedestrian and Evacuation Dynamics*, 227–266, 2002.
- [7] S. Hoogendoorn and P. Bovy. State of the Art of Vehicular Traffic Flow Modelling. *Proceedings of the Institution of Mechanical Engineers*, 215(1):283–303, 2001.
- [8] B. Hoppe and E. Tardos. Polynomial Time Algorithms For Some Evacuation Problems. *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, 433–441, 1994.
- [9] M. Jha, K. Moore, and B. Pashaie. Emergency Evacuation Planning with Microscopic Traffic Simulation. Technical Report Transportation Research Record 1886:40–48, *The Journal of Transportation Research Board*, 2006.
- [10] D. Karger and C. Stein. An  $O(n^2)$  Algorithm for Minimum Cuts. In *Proceedings of the 25th ACM Symposium on the Theory of Computing*, 757–765, 1993.
- [11] J. Kennington and R. Helgason. Algorithm for Network Programming. *Wiley and Sons*, 1980.
- [12] T. Kisko and R. Francis. Evacnet+: A Computer Program to Determine Optimal Building Evacuation Plans. *Fire Safety Journal*, 9:211–222, 1985.
- [13] T. Kisko, R. Francis, and C. Nobel. EVACNET4 User's Guide. *University of Florida*, 1998.
- [14] Q. Lu, B. George, and S. Shekhar. Capacity Constrained Routing Algorithms for Evacuation Planning: A Summary of Results. *Proceedings of 9th International Symposium on Spatial and Temporal Databases*, 291–307, 2005.
- [15] H. Mahmassani, H. Sbayti, and X. Zhou. DYNAMART-P Version 1.0 User's Guide. *Maryland Transportation Initiative, University of Maryland*, 2004.
- [16] Minnesota Homeland Security and Emergency Management. *Monticello Evacuation Planning web site*. <http://www.hsem.state.mn.us/>, 2004.
- [17] G. Theodoulou and B. Wolshon. Alternative Methods to Increase the Effectiveness of Freeway Contraflow Evacuation. Technical Report Transportation Research Record 1865:48–56, *The Journal of Transportation Research Board*, 2004.
- [18] F. Zhan and C. Noon. Shortest Paths Algorithms: An Evaluation Using Real Road Networks. *Transportation Science*, 32:65–73, 1998.