Effective evacuation route planning algorithms by updating earliest arrival time of multiple paths

Manki Min
Dept. of EECS
South Dakota State University
Brookings, SD 57007, USA
manki.min@sdstate.edu

Jonguk Lee
LG Electronics
LG Gasan Digital Center, 189,
Gasan digital 1-ro
Geumcheon-gu, Seoul
153-803, Korea
peace0114@gmail.com

Sunho Lim
T²WISTOR: TTU Wireless
Mobile Networking Laboratory
Dept. of CS
Texas Tech University
Lubbock, TX 79409, USA
sunho.lim@ttu.edu

ABSTRACT

More and more natural disasters are happening around the globe and in such urgent situations, effective evacuation planning is one of the most critical tools for human safety. Evacuation planning algorithms are different from traditional network routing algorithms in the sense that the objective is to minimize the time when the last evacuee arrives at the destination. Among the existing evacuation planning algorithms, CCRP++ finds good solutions but its computation time is not scalable in terms of the number of the evacuees. In addition, in order to improve the computation time from the base algorithm CCRP, CCRP++ makes non-global update of earliest arrival time of each source node which results in unnecessarily large evacuation time.

In this paper we investigated the two factors (evacuation time and computation time) of CCRP++ and proposed three algorithms, DMP, SMP, and EET, that significantly improve both factors. All four algorithms take the transportation network as the input and output the complete evacuation scenarios in which individualized evacuation plan for each evacuee is determined. Our first algorithm DMP significantly reduces the number of the shortest path findings during the evacuation iteration which is the main reason of large computation time of CCRP++. In addition, by updating the earliest arrival time of each found path efficiently, the evacuation time of DMP output is on average reduced compared to that of CCRP++ output. Our second algorithm SMP further reduces the computation time by first finding the limited number of shortest paths before the evacuation iteration while maintaining the reduced evacuation time. Our third algorithm EET focuses on reducing the evacuation time of SMP output even in the worst-case scenarios by efficiently and effectively estimating the evacuation time and by using it to determine the source node to be evacuated in each round. The computation results show that EET successfully reduces the evacuation time, especially when DMP/SMP outputs were worse than CCRP++. Due to its algorithmic complication, EET has slightly increased computation time compared to SMP, but still remains comparable to DMP.

SIGSPATIAL '14, November 04-07 2014, Dallas/Fort Worth, TX, USA

http://dx.doi.org/10.1145/2675316.2675326.

Keywords

Scalable, Suboptimal, Evacuation Routing

1. INTRODUCTION

For effective evacuation, contraflow routing and evacuation planning algorithms are required. Contraflow routing algorithm is to reverse edges (lanes) to minimize the evacuation time, which is the time required to move all evacuees to the destinations. Evacuation planning algorithm is to find evacuation scenarios to minimize the evacuation time. In both contraflow routing and evacuation planning algorithms, the evacuation time and the execution time are two important objectives.

Contraflow routing algorithm considers reversing edges to decrease or increase the capacities of the edges. Discussions on the contraflow schemes have been published in literature [1, 14, 15, 16]. Effective contraflow route is important to achieve better evacuation time. For example, there is a contraflow greedy heuristic algorithm in [10]. The evacuation time of a contraflow generated by the greedy heuristic algorithm based on CCRP++ [17] is shorter than that of CCRP++ without contraflow on average.

There have been many researches on evacuation planning based on network flow model [2, 5] and linear programming approach [6, 7, 8, 11]. Getting evacuation time in a large network takes longer time than that in a small or medium network. Thus, execution time is also important especially for time-critical disaster happening. The algorithm CCRP++ is known as an efficient evacuation planning algorithm and it is an improved version of CCRP. It shows shorter execution time than the execution time of CCRP [12]. The evacuation time of CCRP++ is similar to that of CCRP.

A greedy heuristic algorithm [10] is a contraflow routing algorithm. The greedy algorithm uses evacuation scenario generated by an evacuation planning algorithm such as CCRP++ to calculate the factors that are used to select the edges to be reversed. The reversed edge makes contraflow that increases the capacity of the evacuation flow and decreases the evacuation time. The contraflow is the result of the greedy algorithm. The result can be used as

the input of an evacuation planning algorithm to determine its evacuation time.

A contraflow algorithm MTFC [13] shows similar evacuation time as the greedy algorithm when CCRP++ is used as the evacuation planning algorithm. The algorithm MTFC allocates more capacity to the shortest paths and has shorter execution time than the greedy algorithm.

The main contribution of this paper is to propose three new evacuation planning algorithms to achieve short execution time and similar evacuation time; static multiple paths (SMP), dynamic multiple paths (DMP), and estimated evacuation time (EET). The result by all of SMP, DMP, and EET confirm that the new algorithms are significantly more time-efficient than CCRP++ with decreased evacuation times. Especially SMP and EET outperform other algorithms, DMP and CCRP++, in terms of the computation time and evacuation time respectively.

The remainder of this paper is composed as follows. In section 2 we'll discuss related work of both contraflow routing and evacuation planning algorithms that are used in our computation. In section 3 we'll firstly describe how to globally update the earliest arrival time of the found paths efficiently. Then the three algorithms DMP, SMP, and EET are presented and issues and solutions found in those algotihms are discussed in section 3. The computation results are presented and analyzed in section 4. Finally section 5 concludes this paper.

2. RELATED WORK

Our main algorithm is evacuation planning algorithm but some contraflow route algorithms, such as the greedy algorithm or MTFC (Maximum Throughput Flow-Based Contraflow), are helpful for better evacuation plan [9, 10, 13]. Contraflow evacuation routing algorithms such as the greedy algorithm in [10] require a complete evacuation scenario and that is another reason of why we study evacuation routing problem itself.

Another related work category is evacuation planning algorithm. Our algorithms are an improved version of CCRP++ [17] which is known as an efficient evacuation planning algorithm. CCRP++ is an improved version of CCRP [12] that is another evacuation planning algorithm. CCRP++ with MTFC shows shorter or similar evacuation time than evacuation time from CCRP++ with Greedy contraflow algorithm.

In this section, two cotnraflow algorithms, Greedy and MTFC, are introduced and two evacuation planning algorithms, CCRP and CCRP++, are introduced as well.

2.1 Greedy Contraflow Algorithm

The greedy algorithm is a contraflow route algorithm [10]. The greedy algorithm needs an evacuation route planner algorithm such as CCRP++ in [17]. To get flow history and evacuation time, the greedy algorithm uses evacuation planner at first. Evacuation time is defined as a required time to evacuate all evacuees to destination node. The greedy algorithm calculates congestion index by dividing flow history of edge by multiplication of edge's capacity and evacuation time. The congestion index values of edges are sorted in descending order. Each edge is flipped if congestion index of the edge is greater than opposite edge's congestion index. Since the greedy contaflow algorithm needs the complete flow history, it requires as part of an input, a complete evacuation scenario for the given problem instance. Using more efficient evacuation planning algorithms will be helpful to reduce the computation time of such contraflow evacuation planning algorithm too.

2.2 CCRP

CCRP in [12] is a sub-optimal evacuation planner to reduce the execution time of optimal evacuation planner algorithm. According to [12], heuristic approaches of CCRP are based on Dijkstra's algorithm in [4, 3]. Since optimal evacuation planner requires a time expanded graph and hence more memory and longer running time are required than CCRP. But CCRP does not require a time-expanded graph. It builds evacuation plan based on "Earliest Arrival time (EA)" from source nodes. CCRP finds the shortest paths from all source nodes to all destinations based on travel time by using Dijkstra the shortest path algorithm. After that, it sends evacuees on source node of a path whose EA is minimum among found paths to a destination node of the path. The number of the evacuees sent to the destination node is maximum possible evacuees based on capacities of edges on the path. That is, it is minimum capacity among the edges of the paths because greater evacuees than minimum capacity of edges cannot be sent because of an edge with minimum capacity.

2.3 CCRP++

CCRP has the overhead of finding the shortest paths for all source nodes and destination nodes in every iteration. As a result, the paths that may be helpful in the future are not considered and overhead occurs until all evacuees are evacuated to destination nodes. To solve this problem, CCRP++ uses a priority queue based on EA. Key in the queue is EA and value in the queue is source node information such as path from the source node. To avoid the overhead of CCRP, CCRP++ does not find the shortest paths per every iteration. Instead, it updates EA of source node by checking if updating EA is required.

To distinguish non-reserved source nodes from reserved source nodes, CCRP++ uses two priority queues. An auxiliary priority queue for non-reserved source nodes is called PreRQ and a priority queue for reserved source nodes is called RQ. Their <key, value> are <EA_i, S_i >. Top element of a priority queue has the smallest EA in the queue.

At first, CCRP++ finds the shortest paths from all source nodes and found EA of each source node is inserted into PreRQ. Top of PreRQ is inserted into RQ and it is popped.

Evacuation iteration runs until all evacuees are evacuated to destination nodes. If the EA of PreRQ's top is less than the EA of RQ's top, availability of PreRQ's top is checked. If the PreRQ's top is available, it is reserved and pushed into RQ. Otherwise, its EA is updated and pushed into PreRQ again.

If the EA of RQ's top is less than or equal to the EA of PreRQ's top, RQ's top element is popped and its EA is updated and the element is reserved until its next element in RQ has less EA than its EA. But RQ is empty, it is assumed that the next element has maximum EA.

Whenever availability of element in priority queue is checked or EA of the element is updated, the process to get the shortest path is required because reserving path process may affect EA information. Reserved elements are the elements having the smallest EA in all the queues.

3. OUR ALGORITHMS

In this section, we first introduce two algorithms DMP/SMP and discusses the differences between them and CCRP++. Then we present our third algorithm EET as a solution to the issues related to the first two algorithms.

There are four differences between our algorithms and CCRP++. The first difference is the selection of the path whose EA is updated. CCRP++ only updates the EA of the top element in the priority queue but ours update all elements in the priority queue.

The second difference is the number of priority queues. CCRP++ uses two queue called PreRQ and RQ but ours use only one priority queue.

The third difference is the frequency of finding the shortest path. CCRP++ repeatedly finds the shortest path for each evacuee but ours stop finding the shortest paths when no more new path is found.

The fourth difference is the order of the priority queue. The priority queues in CCRP++ has the smallest EA as top element but in our DMP and SMP priority queue has the largest EA as top element to reduce evacuation time in a particular case. The particular case will be introduced in analysis section in detail. This ordering motivates the study of our third algorithm and it will be discussed later again.

The common algorithms shared by DMP and SMP are given in Figure 1.

```
ReservePaths()
 1:  do
 2:    Q1 = Q.dequeue()
 3:  while evacuee doesn't exist on Q1.SrcNode
 4:  ComparedEA = Q1.EA
 5:  for each path p in Q1
 6:    if p.EA = ComparedEA AND evacuee exists on
     Q1.SrcNode
 7:      if p.EA > evacTime
 8:        evacTime = p.EA
 9:      Update p.capacity
10:      p.Evacuees -= p.capacity
11:      for each edge e on the path p:
12:        Update e.timeCapacity
13:        for each path p' sharing e:
14:        IncreaseStartTime(p')
15:      IncreaseStartTime(p)
16:      Update p.EA

UpdateEAsInQ()
 1:  Q.enqueue(Q1)
 2:  Update EA of all elements in Queue
 3:  Update sequence in Q by heapifying priority queue

IncreaseStartTime(p)
 1: Increment p.startTime until every edge e on p has non-
    zero capacity at the time of travel time sum before e +
    p.startTime
```

Figure 1: Algorithm Reserve Paths And Update EA

3.1 DMP

The algorithm DMP (Dynamic Multiple Paths) is presented in Figure 2. The motivation of DMP is the inefficiency of repeated path finding by CCRP++. From the generated evacuation scenarios, we observed that in most of the evacuation iteration the same path is selected and reserved successively. Similar to CCRP++, it finds the shortest paths during evacuation iteration. Evacuation iteration in DMP is performed during the evacuees needed be evacuated exist.

GetMoreShortestPath is performed for all source nodes. In GetMoreShortestPath phase, GetElementInQBySource function returns element for the source from priority queue. If the element does not exist in the queue, new element is created and its EA and findingPathDone flag are initialized. After that, the new element is added into the queue. DMP requires findingPathDone flag of element in priority queue.

The findingPathDone flag is initialized to false value. When no new path is found any more from source node of the element, findingPathDone flag of an element in priority queue is set as true, and DMP algorithm does not find the shortest path from the source of the element. While findingPathDone is false, finding the shortest path is performed from the source node.

The routine of finding the shortest path repeats by reducing the capacities of the edges on the shortest path by the capacity of the path until new path is found. If a new path is found, it is added in the queue. Otherwise, findingPathDone flag is set to true and getting the shortest path is no longer performed about the source node. After one step about source node and capacities of edges on paths are restore. After loop about all source nodes is terminated, sequence of priority queue is updated by heapifying the heap of the queue.

Next step is reserving paths and updating EA. The processes are introduced in Figure 1 and it is used in SMP as well. To reserve path, the element with largest EA in a priority queue is extracted as Q1. But Q1 should have evacuees on its source node. ComparedEA variable has Q1's EA.

If the EA of the path equals to ComparedEA and its source node has one or more evacuees, current evacuation time evacTime variable is updated in the case of current EA is greater than evacTime. Then, the capacities of Paths in Q1 are updated and the paths are reserved.

Reserving path evacuates evacuees from a source node of Q1 at the current start time of the path. The number of evacuees evacuated from the source node is capacity of the path. In reserving path, start time of the path is increased for next reserving the path. Also other paths using the edges with zero capacity on the reserved path at same time have updated EA by increasing the start time of other paths. Note that updating capacity of path before reserving the path is required to consider shared edge by other paths. If some edges on reserved path are used by other paths, EA of path may have wrong value.

After reserving paths is done, Q1 is inserted to queue again. The EAs of all elements in the queue are updated and sequence in queue is updated by heapifying heap of priority queue.

Reserving paths and Updating EA in Figure 1 repeats until there is no evacuee at any source nodes.

3.2 SMP

The algorithm SMP (Static Multiple Path) is presented in Figure 3. The motivation of SMP is the unnecessary path findings in each evacuation iteration that DMP performs assuming the repeated finding of the same path in each try. SMP does not find the shortest paths during evacuation iteration because paths are found before evacuation iteration. At first, it makes a new element to be inserted in priority

```
// Initialization
 1: exitNum,evacNum = 0
 2: for each source node s:
 3:   evacNum = evacNum + s.evacueeNum
// Evacuation Iteration
 4: while exitNum < evacNum
// GetMoreShortestPaths
 5:   for each source s
 6:     rq = GetElementInQBySource(s)
 7:     if rq is NULL
 8:       rq = new element for source s
 9:       Initialize rq.EA
10:       rq.findingPathDone = false
11:       Add rq in Q
12:     if rq.findingPathDone is false
13:       p = NULL (p is current path)
14:       do
15:         if p is not NULL
16:           Store original capacity of p if it was not stored
17:           Update p.capacity
18:           for each edge of p e
19:             e.capacity = e.capacity - p.capacity
20:         p = GetShortestPath(rq.SrcNode)
21:       while p is not NULL and p is in rq
22:       if p is NULL (New path is not found any more)
23:         set that rq.FindingPath is done
24:       else if p is new path
25:         Add p into rq
26:         Update EA of rq
27:       Restore capacities of paths in rq
28:   Update sequence in Q by heapifying priority queue
29:   ReservePaths()
30:   UpdateEAsInQ()
```

Figure 2: Algorithm DMP

queue and GetShortestPath (based on Dijkstra's algorithm) is performed. The element in the priority queue has multiple paths and the minimum EA value among the paths. GetShortestPath finds the shortest path from source node and adds the new shortest path in the element. Then it updates the EA of the element and returns the element. The EA of the element is updated to the minimum EA among the paths in the element.

Finding the shortest path routine stops when new path is no longer found. After that, the reduced capacities of the edges are restored and the elements in the priority queue is updated by heapifying the heap of the priority queue.

Other two processes are reserving paths and updating EAs in queue. They are the same as the processes of DMP. The processes are introduced in Figure 1. The two processes are performed until all evacuees are evacuated by comparing exitNum and evacNum.

3.3 Analysis

When comparing our algorithms DMP/SMP to CCRP++, most evacuation time results are shorter than CCRP++ and all execution time results are dramatically shorter than CCRP++. CCRP++ finds the shortest path when it checks whether the element in PreRQ is available or EA of RQ's top is updated. But SMP algorithm statically finds the shortest paths at first and DMP finds paths dynamically and stops when new path is no longer found. Thus, CCRP++ takes longer running time than our algorithms. CCRP++ updates only EA of top element in priority queues but our algorithms update EA of all paths from all source nodes after reserving paths. Thus, our algorithm achieved better evacuation time results in most cases. But few cases show worse evacuation time results than CCRP++. We analyzed the results in worse and better cases in the following subsections.

```
// Initialization
 1: exitNum,evacNum = 0
 2: for each source node s:
 3:   rq = new element for queue element
 4:   rq = GetShortestPath(rq.SrcNode, rq)
 5:   Q.enqueue(rq)
 6:   evacNum = evacNum + s.evacueeNum
// GetMoreShortestPaths
 7: for each element of Q rq
 8:   do
 9:     for each path in rq p
10:       Store original capacity of p if it was not stored
11:       Update p.capacity
12:       for each edge of p e
13:         e.capacity = e.capacity - p.capacity
14:     oldPathsCount = The number of paths in rq
15:     rq = GetShortestPath(rq.SrcNode, rq)
16:     newPathsCount = The number of paths in rq
17:   while newPathsCount > oldPathsCount
18:   Restore capacities of paths in rq
19:   Update sequence in Q by heapifying priority queue
// Evacuation Iteration
20: While exitNum < evacNum
21:   ReservePaths()
22:   UpdateEAsInQ()
```

Figure 3: Algorithm SMP

3.3.1 *Distribution of evacuees from the same source node*

If many evacuees can be distributed by many paths from the source node without long delay of other paths, evacuation time can be shorter.

Figure 4 shows an example about the distribution from a source node and way to expand path in DMP/SMP. Network #3 can evacuate many evacuees than Network #1 from a source node. SMP finds the next shortest path by iteratively reducing the capacities of the edges on the found paths and finding the shortest path. In this way SMP expands the shortest path pool. The algorithm DMP finds one path per source node in every iteration. When DMP finds a new path from the second iteration, reservation of the paths in the first iteration may affect what paths will be found in the current iteration. So if possible we should find many paths all at once instead of incrementally finding multiple paths. However there is clearly tradeoffs between many paths and smaller number of paths. Having a larger pool of paths not only increases the path finding time but also increases the path maintaining time such as the time to maintain EAs of each path. With smaller pool of paths, on the other hand, we may lose opportunities to further reduce the evacuation time.

Some cases of shorter evacuation time output in DMP/SMP than CCRP++ show that in DMP/SMP more evacuees are distributed from a source node whose evacuees lastly arrive at destination nodes than CCRP++. Unlike CCRP++, our algorithms update EA of all source nodes after reserving path. Therefore, more appropriate paths than CCRP++ are reserved when evacuees are distributed by reserving path.

3.3.2 *Useful blocked edges in DMP and SMP*

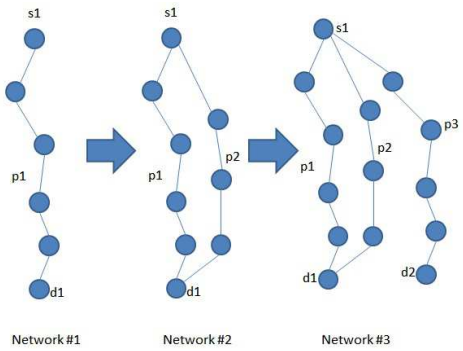


Figure 4: The example for distribution in DMP and SMP

Since CCRP++ does not block edges by reducing the edge capacity as is the case with DMP/SMP, it shows the case that some shared edges could be useful at different time. In DMP/SMP, edges with zero capacity obtained by reducing capacity cannot be used when finding the shortest path. However, those edges may be useful if they are used at different time.

Figure 5 shows shared edges. Green edges are the shared edges for path p2 and p3. Red edge is the shared edge for the paths p1 and p2. Both DMP and SMP block the first edge among green edges and red edge because the edges have the minimum capacity of found path. That is, the three paths cannot be found in DMP and SMP but CCRP++ finds them. The three paths found by CCRP++ contribute shorter evacuation time than the paths found by DMP/SMP.

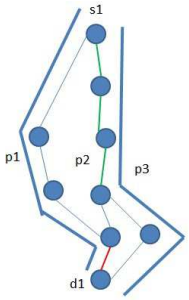


Figure 5: Blocked shared edges in DMP and SMP but useful in CCRP++

3.3.3 *Source node with largest EA considered at first*

Unlike CCRP++, DMP/SMP use a priority queue with EA in descending order. That is, its top element has the largest EA among elements in the queue. This is useful to reduce the evacuation time in some cases of paths having shared edge from different source nodes.

For example, there are two paths that share one edge and have different EAs but same capacity in Figure 6

Figure 6 can be expressed as an arrival graph. An arrival graph consists of capacity (y-axis) and arrival time (x-axis). Since two evacuees are sent from a source node in every evacuation iteration, a path p1 has arrival time between 50 and 150 and a path p2 has arrival time between 100 and 200. The arrival graph for the example is introduced in Figure 7.

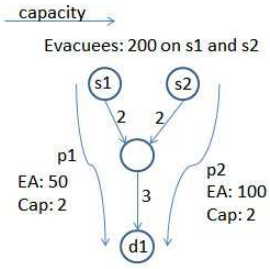


Figure 6: Two paths example to reduce evacuation time by using the largest EA queue

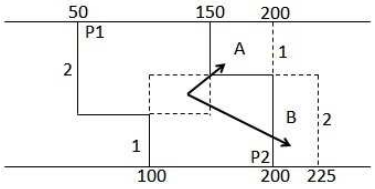


Figure 7: Arrival graph for the largest EA and the smallest EA queue

In Figure 7, the overlap area of p1 and p2 needs to be moved to either A or B area. If priority queue's top element has the smallest EA like CCRP++, evacuation time will be 225 and it can be expressed as an arrival graph in Figure 8.

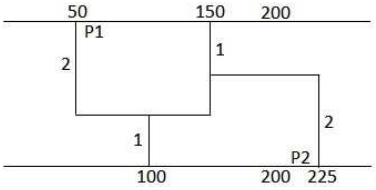


Figure 8: Arrival graph for the smallest EA queue

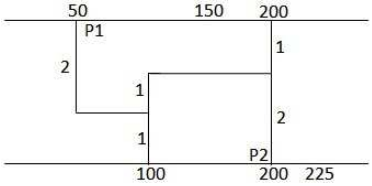


Figure 9: Arrival graph for the largest EA queue

However, evacuation can be 200 which is less than 225 if the top element of priority queue is the largest EA in the queue. Its arrival graph is shown in Figure 9. It is noted that not every problem instance will find this approach useful, for example if s2 has 120 evacuees instead of 200 as shown in Figure 6, then the area A in Figure 7 will be expanded to reach the time of 200 while B will reach only 185. Moreoever, in that case the actual evacuation time should be 174 since the overlap areas can be moved to A and/or B.

3.4 EET

In this section, we describe how to further reduce the evacuation time by effectively considering the evacuation time of

each source node and introduce our algorithm EET (Estimated Evacuation Time).

In the previous subsection we discussed blocked edges problem and distribution problem from a source node. In order to solve those problems while keeping the computation time low, we statically find as many paths as possible from each source node in the following way: After iteratively finding the shortest paths as in SMP algorithm, we keep finding the new shortest paths by temporarily resetting the capacity of each edge included in the found paths. This process is repeated until there is no more new path found.

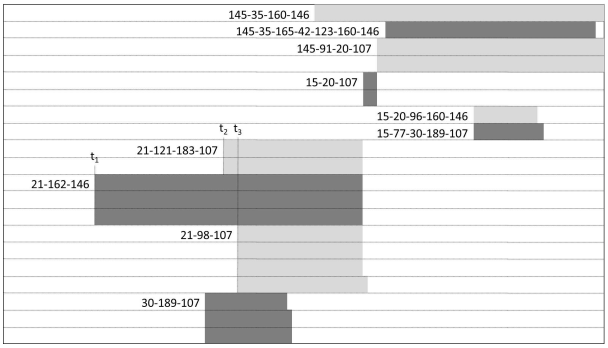


Figure 10: Example arrival graph by CCRP++

As we observed in subsection 3.3.3, the resulting evacuation plan may have different evacuation time depending on which source node is evacuated first. And such decision should be made so that the final evacuation time can be made minimum. Since calculation of the actual final evacuation time is hard itself, we use the estimated evacuation time defined based on the temporal locality. We have observed the temporal locality of the evacuation scenarios in the evacuation results generated by the algorithms CCRP++, SMP, and DMP. Figure 10 shows one such example of temporal locality in the arrival graph generated by CCRP++ for a 200 node input. As can be seen in the graph, each path is successively used in the next discrete time unit which clearly shows the temporal locality. In addition, due to the overlaps of the edges between the paths, for example between the two paths 145-91-20-107 and 15-20-107, some paths may not be used for evacuation at the same time. This makes an exception to the temporal locality and hence makes the problem of computing the evacuation time hard.

The estimated evacuation time of each source node is updated after reserving a path in each iteration in the algorithm EET and is defined as follows:

$$EET_S = S.EA + \frac{S.Evacuees}{S.B} - 1,$$

where S is the source node that we consider in the current iteration, EET_S is the estimated evacuation time in the current iteration, $S.EA$ is the earliest arrival time of S (i.e., the smallest value of earliest arrival time among paths from S), $S.Evacuees$ is the number of evacuees remaining at S in the current iteration, and $S.B$ is the temporal bandwidth of S in the current iteration. $S.B$ is incremented by the capacity of the reserved path so that it represents the temporal bandwidth of the current EA paths from S. For example in Figure 10, at time t_1, the source node 21 has the temporal bandwidth (B) as 3, at t_2, it increases to 5, and at t_3, it becomes 9 and maintains 9 as the temporal bandwidth until the end.

```
// Initialization
 1: exitNum = 0
 2: evacNum = GetNewShortestPaths()
// Evacuation Iteration
 3: while exitNum < evacNum
 4:   UpdateEET(Q.top())

GetNewShortestPaths()
 1: for each source node s:
 2:   rq = new element for queue element
 3:   Reset NE
 4:   do
 5:     for each path p in rq
 6:       Push a new edge e on p into NE
 7:       Determine p's timed capacity
 8:       for each edge e of p
 9:         e.capacity = e.capacity - p.capacity
10:     rq = GetShortestPath(rq.SrcNode, rq)
11:   while a new path is found
12:   Restore capacities of paths in rq
13:   do
14:     for each edge e in NE
15:       Reset e.capacity
16:       rq = GetShortestPath(rq.SrcNode rq)
17:       Add a new edge e on the new path into NE
18:       Restore e.capacity
19:     while a new path is found
20:   Update sequence in Q by heapifying priority queue
21:   Q.enqueue(rq)
22:   evacNum = evacNum + s.evacueeNum
23:   return evacNum

UpdateEET(rq)
 1: cCap = rq.path.capacity
 2: if (rq.tEA < rq.EA)
 3:   rq.B = cCap
 4:   rq.tEA = rq.EA
 5: else
 6:   rq.B += cCap
 7: ReservePath(rq)

ReservePath(rq)
 1: cCap = min(rq.path.capacity, rq.Evacuees)
 2: rq.Evacuees -= cCap
 3: for each edge e on the path rq.path:
 4:   Update e.timeCapacity
 5:   for each path p' sharing e:
 6:     UpdateEA(p')
 7: UpdateEA(rq.path)

UpdateEA(p)
 1: pTT = 0
 1: for each edge e on p starting from its source node s:
 2:   delayCapacity = {}
 3:   for each item (t, c) in e.timeCapacity:
 4:     add (t − pTT, c) into delayCapacity
 5:   pTT += e.TravelTime
 6: Union delayCapacity and find the smallest non-
    negative delay d with non-zero capacity
 7: p.EA = p.TravelTime + d
 8: Reorder p in the path list of rq such that rq.SrcNode
    = s
 9: rq.EA = rq.path.EA
10: rq.EET = rq.EA + rq.Evacuees / rq.B - 1
11: Update sequence in Q by heapifying priority queue
```

Figure 11: Algorithm EET

The complete algorithm EET is given in Figure 11. The priority queue used in EET keeps the source node with the largest EET to be placed at the front and in the paths in each queue item are always sorted based on their EA values in ascending order. For example, $Q.top()$ will return the queue item with the source node with the largest EET value and rq.path will return the path with the smallest EA from the source node of rq.SrcNode. GetShortestPath algorithm is a simple Dijkstra algorithm with some modification to assign proper values such as EA to the queue item rq.

4. COMPUTATIONAL RESULTS

We implemented four algorithms (CCRP++, DMP, SMP, and EET) in C++ on a RedHat Linux machine with dual 2.33 GHz dual core CPUs and 4GB RAM. The algorithms are compared in terms of the evacuation time and the computation time using the minimum measurement among the results of the four algorithms to normalize the measurement of each algorithm.

The input data have n nodes in an $n \times n$ network and the range of n is $\{100, 200, 300, 400, 500\}$. There are 15 different input data per $n \times n$ network. The number of source nodes, m_s, is randomly chosen between 1 and 10 and the number of destination nodes, m_t, is between 1 and 5. The disaster point is randomly generated and the m_s closest nodes to the disaster point are set as the source nodes and the m_t farthest nodes are the destination nodes. The ratio between the number of edges and the number of nodes in the network is set between 1.5 and 3. We assigned randomly chosen capacity between 1 and 5 to each edge and the travel time is set proportional to the distance of the edge. The network is constructed in the way that every source node has at least one path to a destination node and about 95 % of the edges are bidirectional with the same capacity/travel time on both directions.

4.1 Evacuation time results

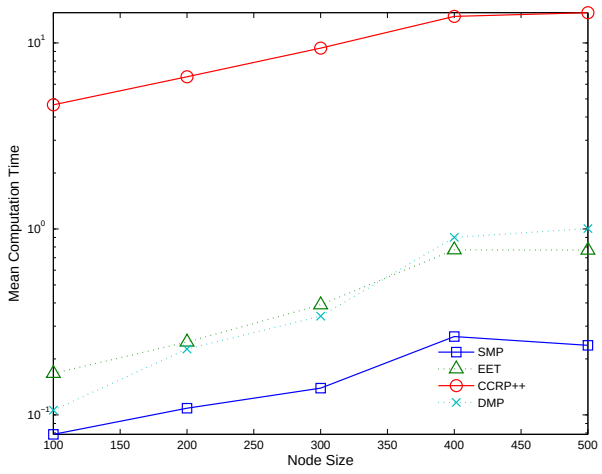


Figure 14: Average computation time of CCRP++, DMP, SMP, and EET

Figure 12 shows the results of the four algorithms in term of the evacuation time. We normalized the evacuation time of each algorithm using the minimum value among them for each problem instance. For example, in the 13-th problem

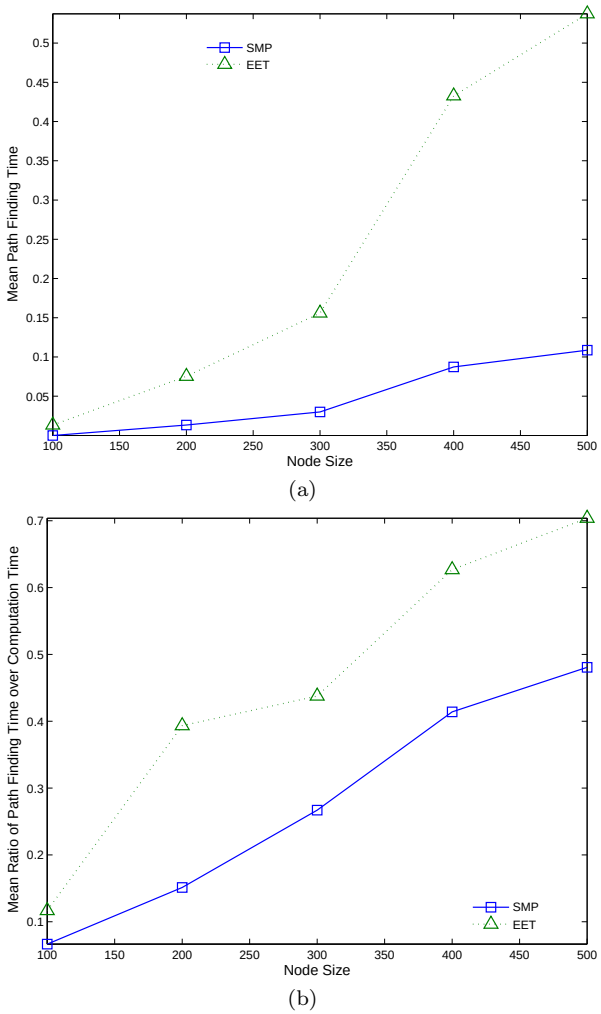
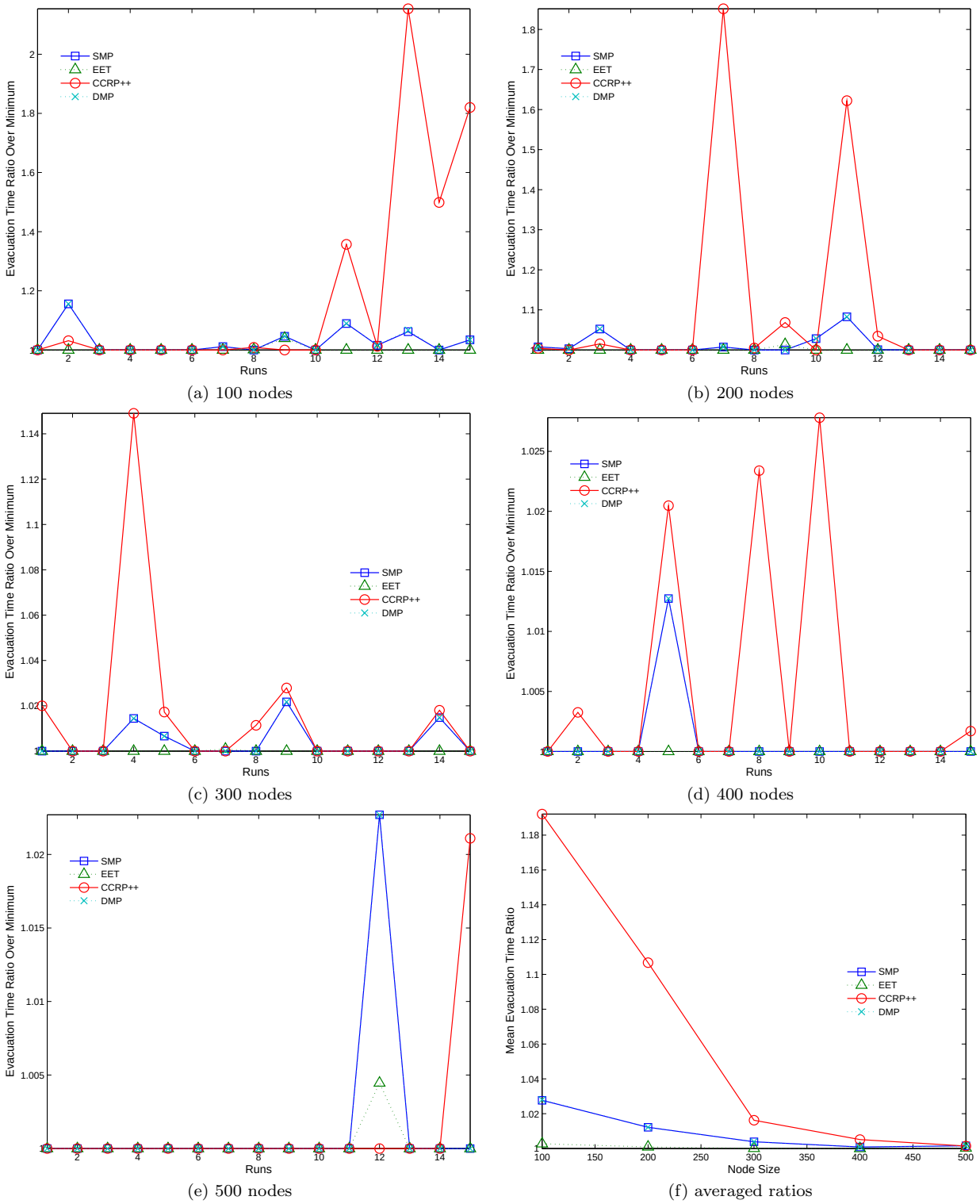


Figure 15: Average time spent for path finding by SMP and EET

instance in (a), CCRP++'s evacuation time is roughly 2.2 times bigger than the minimum evacuation time (which is generated by EET). Subfigures 12 (a)∼(e) are the individual run results for 100∼500 node cases and Subfigure 12 (f) shows the averaged ratios for each node size.

As in Subfigure 12 (f), we can see that on average CCRP++ produces the largest evacuation time, DMP/SMP generates pretty close evacuation time, and EET outputs the best evacuation time. When the node size increases, the difference between algorithms reduces and we think this is because of the network generation settings that we used. We maintained the similar settings for the number of evacuees and the ratio of the number of edges. As a result, when we have more nodes in the graph, it seems that the graph tends to have less, but longer paths which does not leave much room for improvement.

In addition to the averaged results, we would like to point out some of the individual run results. Normally CCRP++'s evacuation time was the worst, but occaisionally other algorithms are worse or even CCRP++ is the best as in 2-nd instance of 100 nodes in Subfigure 12 (a) and 12-th instance of 500 nodes in Subfigure 12 (e). This may reflect that



(a) 100 nodes (b) 200 nodes

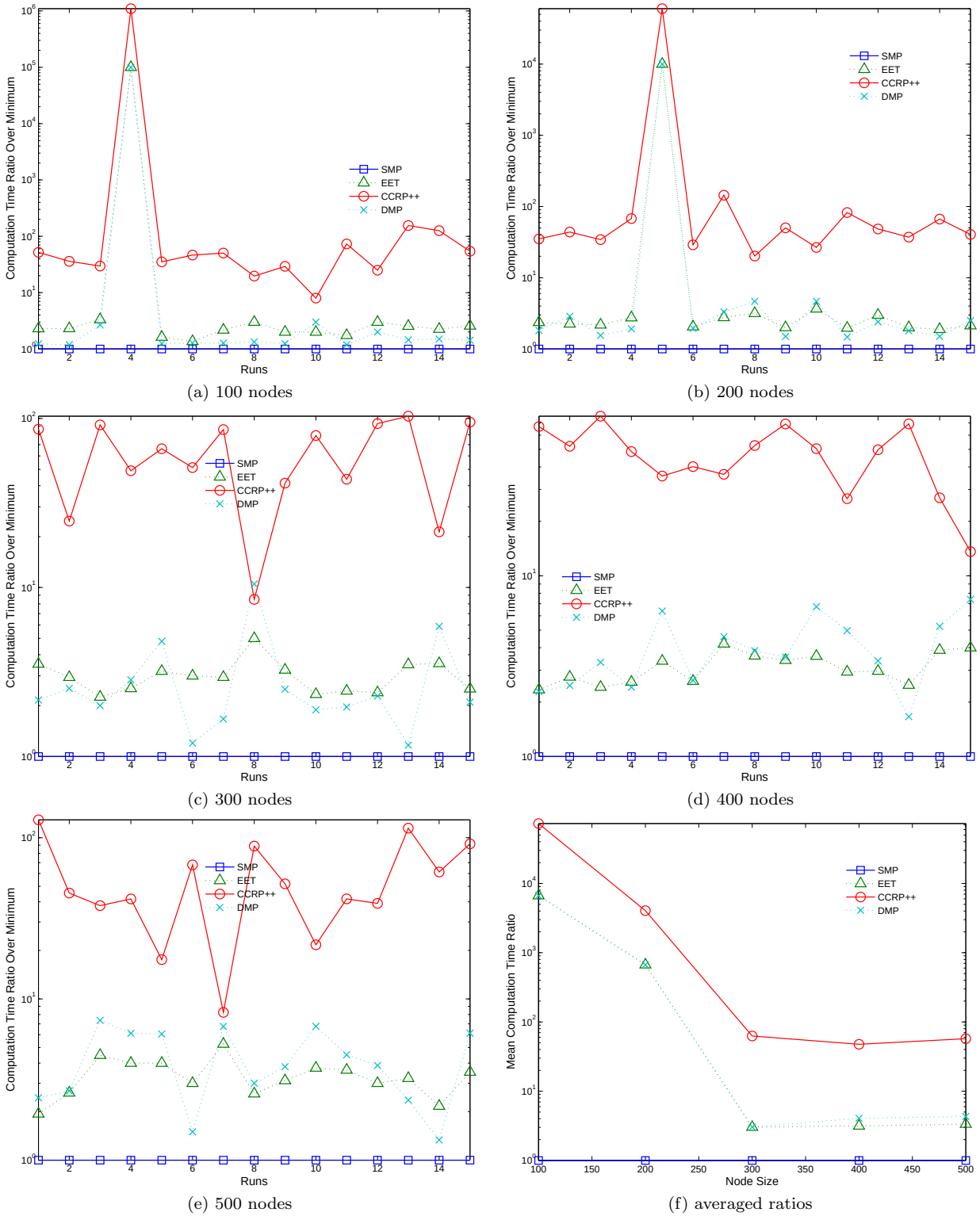
(c) 300 nodes (d) 400 nodes

(e) 500 nodes (f) averaged ratios

Figure 12: Evacuation time ratios of CCRP++, DMP, SMP, and EET over minimum

intricate overlapping of edges was not effectively handled by out algorithms and needs further careful investigation. Bright side is that EET could lower the evacuation time even when CCRP++ was the best as in 12-th instance in Subfigure 12 (e). This shows the effectiveness of the usage of additional path finding and the new ordering of the priority queue based on the estimated evacuation time.

In summary all our algorithms, especially EET, generate the best (or close to the best) evacuation time solutions.



(a) 100 nodes (b) 200 nodes

(c) 300 nodes (d) 400 nodes

(e) 500 nodes (f) averaged ratios

Figure 13: Computation time ratios of CCRP++, DMP, SMP, and EET over minimum

4.2 Execution time results

In all results of execution time, all our algorithms DMP, SMP, and EET show significantly shorter computation time than that of CCRP++ as in Figure 13. We normalized the computation time of each algorithm using the minimum value among them for each problem instance. Since some algorithm's computation time is less than 0.0000001 seconds and is recorded as 0 for some instances, we used 0.0000001

seconds for those cases. Subfigures 13 (a)∼(e) are the individual run results for 100∼500 node cases and Subfigure 13 (f) shows the averaged ratios for each node size.

In terms of the computation time, SMP is the best without any exception and considering that the path finding is a very time-consuming action compared to path reservation, it is an anticipated result. The two algoithms DMP and EET were close each other in terms of the computation time, regardless of the different structure of the algorithms. Computation time alone makes SMP the best algorithm but the evacuation time (or even the combination of the two times) makes EET the best algorithm among the compared algorithms. Several unusual peaks by DMP/EET as in 4-th instance in Subfigure 13 (a) are made simply because SMP has too small computation time and as described above, SMP's computation time in that case is set as 0.0000001. Interesting observation is in one case (8-th instance of Subfigure 13 (c)), DMP took longer than CCRP++ and even in that case EET didn't take longer than CCRP++.

Figure 14 shows the comparison of average computation time of the four algorithms and Figure 15 shows the comparison of average time spent for path finding by the two best algorithms SMP and EET. Subfigure 15 (a) shows the average path finding time spent by SMP and EET and Subfigure 15 (b) shows the average ratio of path finding time over the computation time. As in Subfigure 15 (a), we can see that EET spent larger amount of time than SMP in all cases which results in increased pool of paths. The gap between those two algorithms in Subfigure 15 (a) seems to increase as the network size grows. However Subfigure 15 (b) reveals that considering the increase of the computation time, the gap between the ratio of two algorithms remains pretty consistent which confirms the efficient use of the time for path finding by EET.

5. CONCLUSIONS

In this paper, we proposed three algorithms DMP, SMP, and EET to reduce the execution time and to improve the evacuation time of CCRP++ in [17]. They are evacuation planning algorithms that can be also used to generate the input to a contraflow evacuation planning algorithm.

Our algorithms outperform CCRP++ in terms of the execution time by significantly reducing the number of path finding. As a side-effect of reducing the number of path finding, DMP and SMP suffered the blocked edge issues which still gave very similar evacuation time as CCRP++. They also outperform CCRP++ in terms of the evacuation time by the efficient global update of the EA information. The algorithm SMP outperforms others in terms of the computation time and EET outperforms others in terms of the evacuation time.

The study on the distribution of evacuees problem and blocked useful edges problem lead to our third algorithm EET. The temporal locality in the arrival graph is also useful to define the estimation of the evacuation time which was the heart of EET algorithm.

Based on the findings, we are planning to carefully study the tradeoffs between path pool size and the evacuation time/computation time. In addition, more efficient and better contraflow routing algorithms based on SMP/EET are to be studied in the future.

6. REFERENCES

[1] State of florida contraflow plan. Florida Department of Transportation, http://www.onewayflorida.org/, 2007.

[2] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[3] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. MIT Press, 2009.

[4] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, November 1959.

[5] L. Ford and D. Fulkerson. *Flows in Network*. Princeton University Press, 1962.

[6] R. Francis and L. Chalmet. A negative exponential solution to an evacuation problem. Technical Report Research Report No.84-86, National Bureau of Standards, Center for Fire Research, October 1984.

[7] H. Hamacher and S. Tjandra. Mathematical modelling of evacuation problems: A state of the art. In M. Schreckenberg and S. D. Sharma, editors, *Pedestrian and Evacuation Dynamics*, pages 227–266. 2002.

[8] B. Hoppe and E. Tardos. Polynomial time algorithms for some evacuation problems. In *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 433–441, 1994.

[9] S. Kim and S. Shekhar. Contraflow network reconfiguration for evacuation planning: A summary of results. In *Proceedings of the 13th ACM Symposium on Advances in Geographic Information Systems*, pages 250–259, 2005.

[10] S. Kim, S. Shekhar, and M. Min. Contraflow transportation network reconfiguration for evacuation route planning. *IEEE Transactions on Knowledge and Data Engineering*, 20:1115–1129, 2008.

[11] T. Kisko, R. Francis, and C. Nobel. Evacnet4 user's guide. University of Florida, 1998.

[12] Q. Lu, B. George, and S. Shekhar. Capacity constrained routing algorithms for evacuation planning: A summary of results. In *Proceedings of the 9th International Symposium on Spatial and Temporal Databases*, pages 291–307, 2007.

[13] M. Min and J. Lee. Maximum throughput flow-based contraflow evacuation routing algorithm. In *Proceedings of Pervasive Computing and Communications Workshops, 2013 IEEE International Conference on*, pages 511–516, 2013.

[14] G. Theodoulou. Contraflow evacuation on the westbound i-10 out of the city of new orleans. Master's thesis, Louisiana State University, 2003.

[15] B. Wolshon. One-way-out: Contraflow freeway operation for hurricane evacuation. *Natural Hazards Review*, 2(3):105–112, 2001.

[16] B. Wolshon, E. Urbina, and M. Levitan. National review of hurricane evacuation plans and policies. Technical Report Technical report, Hurricane Center, Louisiana State University, 2002.

[17] D. Yin. A scalable heuristic for evacuation planning in large road network. In *Proceedings of the Second International Workshop on Computational Transportation Science*, pages 19–24, 2009.