Synchronized Flow-Based Evacuation Route Planning[*]

Manki Min

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

Abstract. When a disaster occurs, we need a routing plan to evacuate all the people in the affected area as soon as possible. For this purpose, we can model the transportation network as a graph of nodes and edges with occupancy on nodes and capacity and travel time on edges, where nodes represent places such as cities and edges represent roads. Given a transportation network graph, we can compute routes to evacuate all the people in the dangerous area by selecting paths from the source nodes (the nodes of which residents need to be evacuated) to the destination nodes (the nodes where the evacuees can be transported to). With capacity and travel time constraints on the roads (or edges), calculation of the evacuation time on the graph requires the use of time-expanded graphs. The use of time-expanded graphs, which are merely duplications of the given graph flagged with discrete time stamps, explodes the time and space complexities of the calculation of evacuation times. This drawback results in low scalability, especially when the evacuation time or the number of evacuees is relatively big compared to the size of the graph, such as the number of nodes, edges, and paths. In this paper, we present a scalable algorithm, SYNChronized FLOw Evacuation(SyncFloE), to plan the evacuation routes based on synchronized flows. The novel concept of synchronized flows replaces the use of time-expanded graphs and provides higher scalability in terms of the evacuation time or the number of evacuees. SyncFloE has computation time that only depends on the number of source nodes and the size of the graph itself, such as the number of nodes, edges, and paths. The computational results that support our claim are presented and discussed.

1 Introduction

With recently increasing occurrence of disasters such as hurricanes, tsunamis, earthquakes, and nuclear meltdowns all over the globe, the more importance is put on the efficient and effective operation of evacuation process. When such a disaster happens, the people who stay in the area that can be affected by the disaster (this area is will be different depending on the types of the disasters) need to move to safer places (often times shelters). Since there can be a lot of people that need to be evacuated, without the help of well-established evacuation routes it will be a very time-consuming and ineffective evacuation. Depending on the types of the disasters, the time that we have to compute such evacuation routes and at the same time the time that the last evacuees are evacuated, called evacuation time, can vary. Nonetheless, regardless of the type of

[*] Research is partly supported by NSF Award CCF-0729182.

disasters, the algorithm must be scalable so that reasonable amount of time could be used to compute the evacuation routes and the evacuation time could be minimized.

The computation of the evacuation routes and evacuation time is based on the abstraction of the transportation network as a graph of nodes and edges with occupancy on nodes and capacity and travel time on edges, where nodes represent places such as cities and edges represent roads. Given a transportation network graph, we can compute routes to evacuate all the people in the dangerous area by selecting paths from the source nodes to the destination nodes. In this paper, we present our scalable algorithm, SYNChronized FLOw Evacuation (SyncFloE), which can compute the evacuation time without the help of time- expanded graphs. There are algorithms to compute the evacuation time in literature, but they are not scalable due to the fact that either they are based on time-expanded graphs or their computations repeat over time. SyncFloE uses the novel concept of *Synchronized Flow* to replace the use of the time-expanded graphs and/or the repetitions over time. A synchronized flow is the flow of evacuees from the source node(s) to destination node(s) that can have the same evacuation time by redistributing the evacuees over the paths along the flow from either the same source node or different source nodes. To ensure the computation of accurate evacuation times, we need to introduce *Virtual Evacuees* into the synchronized flows. Virtual evacuees are the evacuees that do not actually exist and they are added to the synchronized flow just to make the synchronized flow have the same evacuation time for all the paths.

Our contribution in this work is three-fold: firstly, our algorithm provides a new efficient way to compute the evacuation time and plan the evacuation routes. SyncFloE can compute the evacuation time extremely quickly so it can be directly used in real evacuation situation. This real-time calculation with the latest information about the roads and the cities (or places) will ensure always the most accurate evacuation routes. Secondly, our algorithm can be used in the computation of contraflow-based evacuation routes. In one of our previous work, we presented algorithms for the computation of contraflow-based evacuation routes and one of them is using the evacuation routes in its computation of contraflows. This algorithm inherits the scalability from evacuation routing algorithms and the existing evacuation routing algorithms cannot make it scalable. In addition, we are planning to extend SyncFloE to design another efficient and effective contraflow-based evacuation routing algorithm. Thirdly, the novel concept of synchronized flows will make a very powerful tool in the network flow study. In case of single packet routing, only the information of the edge such as the travel time or the capacity is required in the routing. However when we consider multiple packet routing, there's more than just the edge information in the routing computation and the synchronized flows (or time-expanded graphs as an old way) will play the key role in the routing.

The rest of this paper is as follows: Section 2 will define the problem of evacuation routing and discuss briefly the existing algorithms. In section 3, the novel concept of synchronized flows are explained and discussed. The algorithm SyncFloE is introduced in section 4 and explained. The computation results are presented and discussed in section 5 and section 6 concludes this paper.

2 Related Work

Given a transportation network graph of nodes and edges with occupancy on nodes and capacity and travel time on edges, where nodes represent places such as cities and edges represent roads, the evacuation routing problem is to find the routes, in other words a set of paths, for evacuation so that the time the last evacuees arrive the destination, called the evacuation time, become minimum. The evacuation routing problem and the traditional network routing problem are two different problems in the following reason: the traditional network routing problem considers the routing for single packet and so each edge will be used at most once for the packet and hence the attributes of the edges are enough for the routing computation. However the evacuation routing problem has multiple packets and an edge can be used more than once over time and hence the attributes of the edges are not enough for the computation. One way or another, we need to take care of this time-related attributes and the easiest way is to use the time-expanded graphs which are the duplications of the same graph tagged with discrete time-stamps with proper inter-links between them. Or one can record the usage logs for each edge and the logs become bigger as the evacuation time grows. This difference makes the use of traditional routing algorithms for evacuation routing impractical.

There isn't much work toward the evacuation routing in literature. Capacity Constrained Routing Planner (CCRP) [7–9] and its improvement CCRP++ [11] are reported in literature. CCRP is simply repeated operations of Dijkstra' shortest path algorithm in time-expanded graphs. The algorithm finds the shortest path that is available at current time repeatedly and if it cannot find a path, it increases the current time and continues finding the shortest paths until there is no more evacuees. Since CCRP is based on the time-expanded graphs, 1) it is not scalable, and 2) even worse its running time will also depend on how many duplications the time-expanded graphs have. It is not easy to have a tight estimation of the evacuation time (this number is the number of the duplications) before actually running the algorithm, and this is a kind of dilemma. CCRP++ was proposed as an improvement to CCRP and CCRP++ has much shorter running time than CCRP by reducing the number of shortest path findings. The algorithm uses the maximum capacity of each shortest path found and keeps the usage logs for each edge so that later when the same path is found, it can start from the earliest available time. Anyway CCRP++ also uses the time attributes and its computational complexity is not free from the inherent dependency to the evacuation time. As discussed in section 5, as the evacuation time increases, more precisely as the number of paths that are used grows, the running time becomes extremely higher. Even for a graph with 100 nodes and roughly 300 edges, it takes impractically long time.

We proposed our algorithm Quickiest Path Evacuation Routing (QPER) in [10]. The quickest path is the path that gives the minimum evacuation time when combined with the existing paths by redistributing evacuees. QPER is using quickest paths instead of shortest paths and hence the number of paths that are used in the evacuation is much smaller in QPER than CCRP++ and the running time of QPER is more scalable than CCRP++. QPER is finding the evacuation routes for a single source node and this result became the motivation of this work, synchronized flows.

Slightly different focuses have been put on works in [1, 3, 4]. In [1], the authors discussed the problem of lane-based routing to minimize the traffic delay. The main effort of the proposed work is focused on the minimization of the crossing conflicts at the intersections of the roads. In this work, they considered maximum flow (which will give the maximum constant rate of evacuation flow) to solve the problem and hence it may not lead to an evacuation routing with minimum evacuation time. The quickest transshipment problem in [3] is the problem of minimizing the number of paths to complete the transshipment with demands that exceeds the capacity of the network capacity. By only minimizing the number of paths, we may not be able to get the minimum evacuation time, so this problem cannot be applied to our problem. [4] presents an extensive survey of the mathematical modelling of evacuation problems with different goals under different network configurations.

3 Synchronized Flows

In this section, we explain the synchronized flow that is the basis of the proposed algorithm.

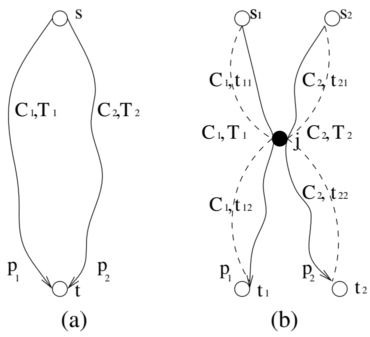


Fig. 1. Multiple paths in a synchronized flow

Let's begin with the simplest case when a source node has more than one paths as in figure 1 (a). In this case, we can freely redistribute the evacuees from one path to another to make the evacuation times of the two paths equal as in figure 2 (a). The combined evacuation time (CET) [10] is computed as follows:

$$CET = \frac{n + C_1 \cdot T_1 + C_2 \cdot T_2}{C_1 + C_2} - 1 \tag{1}$$

where n is the number of evacuees in the source node, C and T represent capacity and travel time of each path and the evacuation time of the source node is $\lceil CET \rceil$. This free redistribution of evacuees is possible at any time as long as the source node still has remaining evacuees and this separates the two cases in figure 1. In (b), the paths from different source nodes cannot have freely redistributed evacuees. In addition, the redistribution of evacuees and the resulting equalized evacuation time distinguish the evacuation routing from the traditional network routing. Figure 2 shows the arrival

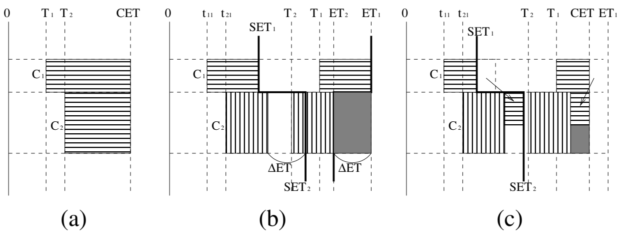


Fig. 2. Arrival Graph

graphs that represent the flows of evacuees that arrive the destination nodes and the joint node of the two paths if there's any.

Now let's consider the multiple paths from different source nodes as in figure 1 (b). In this case, unfortunately we cannot directly apply the CET calculation to equalize the evacuation times, instead we have two evacuation times. In this figure, the black colored node j is the joint node between two paths in different synchronized flows and s_1 has path p_1 and s_2 has path p_2. p_1 has capacity of C_1 and travel time of T_1 and the travel time on p_1 from s_1 to j is t_{11} and the travel time from j to t_1 is t_{12}. We can observe that the grey-shaded are in the arrival graph (figure 2 (b)) is wasted and if we could move some of the rightmost evacuees from s_1 in the upper box to this grey area, we would reduce the evacuation time of s_1. In fact, we need to look at the thick lines which shows the ending time of each path to get the same evacuation time, we call such ending time as synchronized ending time (SET). After the end time of the leftmost box (representing arrival at the joint node) until SET$_2$, we can put evacuees and still get the two evacuation times equal. Note that when we move the evacuees from s_1 to s_2, then SET$_1$ will also change and so does SET$_2$. As a result of this redistribution, we get figure (c) and we get CET less than ET$_1$ (Evacuation Time of s$_1$) and we can say those two paths to have synchronized flow. Still the grey-shaded area exists in (c) and this means that we need to put virtual evacuees into synchronize flows on those two paths.

The concept of virtual evacuees is the key point in synchronizing flows on the paths from the different source nodes. Next we discuss the six cases of synchronizing flows, two for the paths from the same source and four for the paths from different sources. Before we begin, let's define some notations.

- For a node n, $n.id$ means the id of n, $n.Occ$ means the occupancy (or the number of people staying) of n, $n.pSF$ means the list of SF (Synchronized Flow) that pass through n, and $n.pSF(FId)$ means the SF in $n.pSF$ with the flow id of FId.
- For a path p, $p.Cp$ means the capacity of the path p, $p.TT$ means the travel time of the path p;
  $p.Src$ means the origin node of the path p and $p.Dst$ means arrival node.
- For a synchronized flow SF, $SF.CpSum$ means the sum of capacities of paths in SF, $SF.CpTTSum$ means the sum of products of capacity and travel time;
  $SF.Srcs$ is the list of source nodes that evacuates along SF, $SF.FId$ is the flow id of SF;

$SF.EV$ is the number of evacuees on SF, $SF.VE$ is the number of virtual evacuees, and $SF.ET$ is the evacuation time of SF.
$n.pSF(FId)$ maintains $min\Delta T$, $minC$, $max\Delta T$, and $maxC$ that are used to calculate SETs.
For a node n in a path p, we maintain $min\Delta T/max\Delta T$ and $minC/maxC$ to be the minimum/maximum value of $T - t$ among the paths in $n.pSF$ with maximum capacity and its corresponding capacity among all the paths on the synchronized flow with flow id of FId that pass through n, where T is $p.TT$ and t is the travel time on p from $p.Src$ to n;
$n.pSF(FId)$ has SET of $n.pSF(FId).ET - n.pSF(FId).max\Delta T$ if FId is the flow id of p and $n.pSF(FId).ET - n.pSF(FId).min\Delta T$ otherwise.
- S means the set of source nodes and T means the set of destination nodes of the evacuation.
- ΔN means the number of evacuees that need to be moved from one path to another, $VirtEvac$ means the number of virtual evacuees.

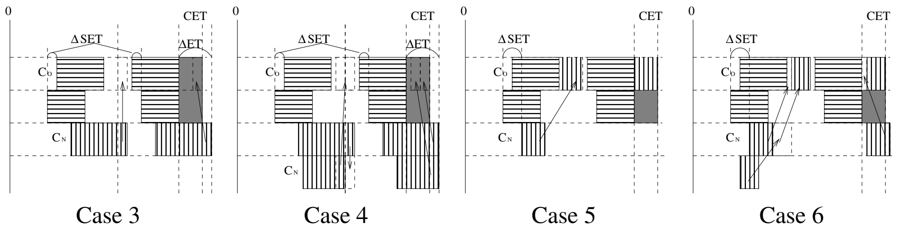


Fig. 3. Arrival graphs for cases 3 $\sim$ 6

Let's say p is the path that is newly added to the graph.

Case 1: If $p.Src$ does not have any synchronized flow yet and $p.Dst \in T$, we can simply make the path to have a new synchronized flow SF such that $SF.FId := p.Src.id$, $SF.CpSum := p.Cp$, $SF.CpTTSum := p.TT$, $SF.EV = p.Src.Occ$, $SF.ET = (SF.EV + SF.CpTTSum)/SF.CpSum - 1$, and $SF.Srcs := \{p.Src\}$. There is no ΔN nor $VirtEvac$.

Case 2: If $p.Src$ has the synchronized flow SF, we can add p into SF such that $SF.CpSum := SF.CpSum + p.Cp$, $SF.CpTTSum := SF.CpTTSum + p.Cp \cdot p.TT$, $SF.ET = (SF.EV + SF.VE + SF.CpTTSum)/SF.CpSum - 1$, and $SF.Srcs := SF.Srcs \cup \{p.Src\}$. There is no ΔN nor $VirtEvac$.

Case 3 (subcase of case 1): if p has a joint node j which has a synchronized flow SF_{old}, then $VirtEvac$ can be calculated as follows:
$jSF_N = j.PSF(SF.FId)$, $jSF_O = j.PSF(SF_{old}.FId)$,
$C_N = p.Cp$, $C_O = \min(jSF_O.minC, C_N)$,
$\Delta ET = SF.ET - jSF_O.ET$,

$\Delta SET = \Delta ET - (jSF_N.maxT - jSF_O.minT)$,
$\Delta N_1 = \dfrac{C_N \cdot C_O}{C_N + C_O} \cdot \Delta ET,\ \ \Delta N_2 = C_O \cdot (\Delta ET - \Delta SET)$,
$CS_O = SF_{old}.CpSum.$
$$VirtEvac = \begin{cases} CS_O \cdot \frac{\Delta N_1}{C_O} - \Delta N_1, & \text{if } \Delta N_1 \geq C_N \cdot \Delta SET, \\ CS_O \cdot \left(\Delta SET - \frac{\Delta N_2}{C_N} + \frac{\Delta N_2}{C_O}\right) - \Delta N_2, & \text{o.w.} \end{cases}$$

Case 4 (subcase of case 2): if $p.Dst \in T$ and p has a joint node j which has a synchronized flow SF_{old}, then $VirtEvac$ can be calculated as follows:
$C_N = SF.CpSum.$
$$VirtEvac = \begin{cases} CS_O \cdot \frac{\Delta N_1}{C_O} - \Delta N_1, & \text{if } \Delta N_1 \geq C_N \cdot \Delta SET, \\ CS_O \cdot \left(\Delta SET - \frac{\Delta N_2}{C_N} + \frac{\Delta N_2}{C_O}\right) - \Delta N_2, & \text{o.w.} \end{cases}$$
The undefined variables are defined in the same way as case 3.

Case 5: If $p.Dst = j \notin T$ and $p.Src$ does not have a synchronized flow yet and j has another synchronized flow SF_{old}, then $VirtEvac$ can be calculated as follows:
$\Delta SET = p.TT - (jSF_O.ET - jSF_O.minT).$
$$VirtEvac = \begin{cases} CS_O \cdot p.Src.Occ - p.Src.Occ, & \text{if } \Delta SET \leq 0, \\ (CS_O + \Delta SET) \cdot p.Src.Occ - p.Src.Occ, & \text{o.w.} \end{cases}$$
The undefined variables are defined in the same way as case 4.

Case 6 (subcase of case 2): If $p.Dst = j \notin T$ and $p.Src$ has a synchronized flow SF and j has another synchronized flow SF_{old}, then $VirtEvac$ can be calculated as follows:
$\Delta SET = p.TT - (jSF_O.ET - jSF_O.minT)$,
$\Delta N_2 = \frac{C_N \cdot C_O}{C_N + C_O} \cdot (\Delta ET \Delta SET).$
$$VirtEvac = \begin{cases} CS_O \cdot \frac{\Delta N_1}{C_O} - \Delta N_1, & \text{if } \Delta SET \leq 0, \\ CS_O \cdot \left(\Delta SET + \frac{\Delta N_2}{C_O}\right) - \Delta N_2, & \text{o.w.} \end{cases}$$
The undefined variables are defined in the same way as case 4 or 3.

The proof of the correctness of the above formula is omitted due to the space limitation. However it is derived from the observation that the redistribution of evacuees is possible only when the evacuees are available. The condition in cases 3 $\sim$ 6 basically means that the SET of the old synchronized flow at the joint node is greater than the SET of the new synchronized flow so that the redistribution can happen right after the old synchronized flow ends passing by the joint node. Otherwise we need to put a pause time before redistributing the evacuees as in figure 3 case 3 or 4.

4 Synchronized Flow Evacuation (SyncFloE)

Figure 6 shows our novel algorithm SyncFloE. As we can see from the description, computation time of SyncFloE will depend on the number of shortest paths with updated available capacities of edges and it does not depend on any other factor such as the number of evacuees, capacities of the edges (and hence the capacities of the paths),

```
Initialization
1: $S$:= the set of source nodes, $PS$:= the set of poorest source nodes, $T$:= the set of destinations.
2: preEvactime := curEvacTime := $\infty$.

Iteration1 while a shortest path $p$ from a node $s \in PS$ to a node $t \in T$ is found using available capacities of edges:
1: if ($p.TT >$ preEvacTime)
2:     $PS := PS \setminus \{s\}$.
3: else
4:     (preEvactime, curEvacTime) = manipulate($p, S, PS, T$, preEvacTime, curEvacTime).

Iteration2 while a shortest path $p$ from a node $s \in PS$ to a node $t \notin T$ is found:
1: (preEvactime, curEvacTime) = manipulate($p, S, PS, T$, preEvacTime, CurEvacTime).
```

Fig. 4. SyncFloE

and so on. This independence gives our algorithm dramatically improved scalability and we believe that this is a lower bound for the computational complexity of evacuation routing.

Adding a path p into a synchronized flow SF involves updating $SF.CpTTSum$, $SF.CpSum$, $SF.EV$, $SF.VE$, and $SF.ET$. In addition, tracking all the nodes in p, the synchronized flows passing the node n will be updated; if the flow is not already included in $n.pSF$, then include SF into $n.pSF$, add $p.Src$ into $SF.Srcs$, update $min\Delta T$, $minC$, $max\Delta T$, $maxC$ properly as discussed in section 3. The removal of p from SF is almost reverse activity of the addition; rolls back the $min\Delta T$, $minC$, $max\Delta T$, $maxC$, removes SF from $n.pSF$ if it was included by the path addition, removes $p.Src$ from $SF.Srcs$ if it was included by the path addition, and finally restores $SF.CpSum$, $SF.CpTTSum$, $SF.EV$, $SF.VE$, $SF.ET$.

Merging two synchronized flows (SF_N into SF_O) involves adding $SF_N.CpSum$, $SF_N.EV$, $SF_N.VE$, and $SF_N.CpTTSum$ to $SF_O.CpSum$, $SF_O.EV$, $SF_O.VE$, and $SF_O.CpTTSum$ and adding $SF_N.Srcs$ into $SF_O.Srcs$. Split of a synchronized flow into two flows is almost reverse activity of merging; rolls back the two synchronized flows to the ones before the merging, and removes $p.Src$ from $SF_N.Srcs$ if it was included by the merging.

After adding/removing a path into a synchronized flow or merging/splitting two synchronized flows, we calculate the evacuation time of the whole graph by calculating the evacuation time of each synchronized flow and updating evacuation time of the source nodes that belong to the synchronized flow. Then curEvacTime becomes the maximum evacuation time calculated and preEvacTime is the evacuation time of the previous calculation. The curEvacTime and preEvacTime updated after the calculation of evacuation time are used to determine whether or not to roll back the changes that we introduced into the graph. When rolling back the changes, the source node of the path will be removed from the poorest source node to give a chance to the other poorest source nodes. If the poorest source node list becomes empty or there are no more paths, the algorithm terminates.

```
1: if $p.Dst \in T$,
2:   if $p.Src$ has a synchronized flow $SF_{new}$, // case 2
3:     add $p$ into $SF_{new}$.
4:     if (curEvacTime > preEvacTime), remove $p$ from $SF_{new}$, and $PS = PS \setminus \{p.Src\}$.
5:     else if $p$ has a joint node with another synchronized flow $SF_{old}$, // case 4
6:       calculate VirtualEvac using Case 4, add VirtEvac to $SF_{old}$, and merge $SF_{new}$ into $SF_{old}$.
7:       if (curEvacTime > preEvacTime), remove VirtEvac from $SF_{old}$, and split $SF_{new}$ from $SF_{old}$, and $PS = PS \setminus \{p.Src\}$.
8:   else, // case 1
9:     add $p$ into a new synchronized flow $SF_{p.Src.id}$.
10:    if $p$ has a joint node with another synchronized flow $SF_{old}$, // case 3
11:      calculate VirtEvac using Case 3, add VirtEvac to $SF_{old}$, and merge $SF_{s.id}$ into $SF_{old}$.
12:      if (curEvacTime > preEvacTime), remove VirtEvac from $SF_{old}$, and split $SF_{p.Src.id}$ from $SF_{old}$, and $PS = PS \setminus \{p.Src\}$.
13: else
14:   if $s$ has a synchronized flow $SF_{new}$, // case 2
15:     add $p$ into $SF_{new}$.
16:     if (curEvacTime > preEvacTime), remove $p$ from $SF_{new}$, and $PS = PS \setminus \{p.Src\}$.
17:     else if $p$ has a joint node with another synchronized flow $SF_{old}$, // case 6
18:       calculate VirtEvac using Case 6, add VirtEvac to $SF_{old}$, and merge $SF_{new}$ into $SF_{old}$.
19:       if (curEvacTime > preEvacTime), remove VirtEvac from $SF_{old}$, and split $SF_{new}$ from $SF_{old}$, and $PS = PS \setminus \{p.Src\}$.
20:   else, // case 5
21:     calculate VirtEvac using Case 5, add VirtEvac to $SF_{old}$, and add $p$ into $SF_{old}$
22:       if (curEvacTime > preEvacTime), remove VirtEvac from $SF_{old}$, and remove $p$ from $SF_{old}$, and $PS = PS \setminus \{p.Src\}$.
```

Fig. 5. Manipulate(p, S, PS, T, preEvacTime, curEvacTime)

SyncFloE repeatedly calls manipulate function (figure 5) which will have activities such as addition (or removal) of a path into a synchronized flow and merge (or split) of two synchronized flows. The computational complexity of manipulate function is bounded by the computational complexity of those activities that are bounded by the number of node because in all activities, we simply traverse the nodes at most once.

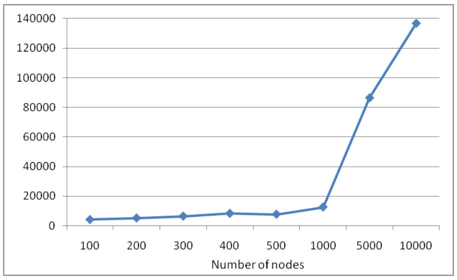
In this paper we haven't included the construction of evacuation routes but it can be obtained by storing additional information such as paths, when the path is used, etc. With the additional information we can easily reconstruct the paths and the flows that depict the evacuation routes.

5 Computational Results

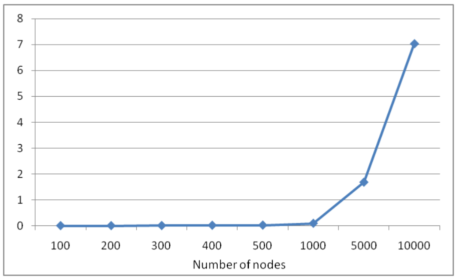
We implemented two algorithms, CCRP++ and SyncFloE. For comparisons, we randomly generated n nodes in a $n \times n$ area with random occupancy for each node, with the value of n in {100, 200, 300, 400, 500, 1000, 5000, 10000}. The number of the source nodes, m_s, and the destination nodes, m_t, are randomly determined between 1 and 10 and between 1 and 5, respectively. Then the point of the disaster is randomly generated and the m_s nodes that are closest to the disaster point are marked as source nodes and the m_t nodes farthest from the disaster point are marked as destination nodes. For each source node, randomly pick edges to generate at least one path. Up to 1.5 $\sim$ 3 times n edges are generated randomly. Each edge is assigned random capacity in between 1 and 5 and travel time proportional to the distance between two endpoint nodes. And we ran each transportation network setting 10 times to get the average results. For the computation we used gcc and g++ as compilers on a Linux machine with dual 2.33 GHz dual core CPU's and 4GB of RAM.

Table 1. CCRP++ VS SyncFloE

Capacity	CCRP++		SyncFloE	
Range	Evac. Time	Comp. Time	Evac. Time	Comp. Time
$1 \sim 5$	8787	12498.6	9105	0.02
$10 \sim 50$	5719	21.21	6527	0.02
$100 \sim 500$	5656	0.4	6216	0.02



Evacuation Time of SyncFloE



Computation Time of SyncFloE

Fig. 6. Computational results of SyncFloE

CCRP++ was not run for all network settings, in fact for most settings CCRP++ took too long time, so we are just presenting one interesting results that explains why CCRP++ is not scalable. We ran CCRP++ for a rather small sized network with 500 nodes (including 7 source nodes and 4 destination nodes) and 829 edges. To evacuate 19856 evacuees from the 7 source nodes to 4 destination nodes took 12498.6 seconds while SyncFloE took 0.02 seconds. However when we multiplied each edge's capacity by ten so that CCRP++ finds less paths, it took 21.21 seconds while SyncFloE took 0.02 seconds. When we multiplied the edge capacities by ten again, CCRP++ took 0.4 seconds and SyncFloE still runs in 0.02 seconds. For this network setting, the computation time has increased by at least 50 times with the decrease of the edge capacity; 50 times from 100 to 10, and 500 times from 10 to 1. The evacuation time of CCRP++ was slightly better than that of SyncFloE and we think it might be the result of inaccurate update of synchronized flows that pass through each node and careful investigation in this direction is one of our future works. The evacuation time and computation time comparison of two algorithms is given in table 1.

Table 2. Evacuation Time and Computation Time by SyncFloE

run index	n	Evac.	Comp.	n	Evac.	Comp.	n	Evac.	Comp.	n	Evac.	Comp.
1	100	887	0.01^+	200	2785	0.01^+	300	3037	0.01	400	8648	0.02
2	100	1354	0.01^+	200	4498	0.01^+	300	1420	0.01^+	400	10911	0.01^+
3	100	3069	0.01^+	200	4185	0.01^+	300	3659	0.01	400	15792	0.01^+
4	100	1674	0.01^+	200	7310	0.01^+	300	5695	0.01	400	3858	0.01
5	100	1901	0.01^+	200	2681	0.01^+	300	8951	0.01^+	400	2482	0.02
6	100	7988	0.01^+	200	5930	0.01^+	300	11213	0.01^+	400	5159	0.01
7	100	1775	0.01^+	200	2618	0.01^+	300	15340	0.01^+	400	6530	0.01
8	100	11861	0.01^+	200	4364	0.01^+	300	4936	0.01^+	400	8964	0.01
9	100	987	0.01^+	200	12413	0.01^+	300	4044	0.01^+	400	13808	0.03
10	100	1697	0.01^+	200	5114	0.01^+	300	6223	0.01	400	7206	0.02
1	500	9105	0.02	1000	7861	0.11	5000	118409	1.15	10000	136811	7.63
2	500	7431	0.02	1000	11520	0.12	5000	107008	1.51	10000	157855	7.79
3	500	7865	0.04	1000	11665	0.1	5000	86369	1.15	10000	145413	5.18
4	500	8541	0.01	1000	12399	0.11	5000	64732	1.72	10000	148794	8.61
5	500	11536	0.01	1000	13540	0.01	5000	50052	1.72	10000	137801	4.28
6	500	10387	0.02	1000	13299	0.1	5000	90902	2.22	10000	107904	8.15
7	500	2688	0.02	1000	10651	0.14	5000	67160	2.17	10000	137496	7.66
8	500	3680	0.01	1000	10025	0.12	5000	117433	2.62	10000	120094	7.69
9	500	7950	0.02	1000	20953	0.07	5000	86681	1.63	10000	137625	5.57
10	500	8028	0.03	1000	14223	0.02	5000	74652	0.93	10000	136771	7.86

$^+$ the running time was shorter than 0.01 seconds.

The results of SyncFloE is given in table 2 and the average values over 10 runs were plotted on the graphs in figure 6). The computation time of SyncFloE for smaller sized networks with up to 1000 nodes was extremely short as less than one second. The increase rate of computation time up to 1000 nodes is negligible but it starts to grow from more than 1000 nodes by at most 20 times; less than 20 times from 1000 to 5000 and four times from 5000 to 10000. This increase of computation time at large-scale networks is inevitable since with more edges, we can expect to have more paths and as a result the algorithm will run longer. This result confirms that the computational complexity of our algorithm depends on the number of paths.

The evacuation time also increases in a similar pattern as the computation time and this is because as the number of nodes grows, the graph will have more edges and longer paths. In fact, when the number of nodes was increased by ten times from 1000, the evacuation time became roughly ten times bigger. However when the number of nodes is small as 500 or less, the number of evacuees will play more important role in determining the evacuation time and hence we don't observe the similar increase pattern for the evacuation time.

6 Conclusions

In this paper, we proposed a truly scalable algorithm for evacuation routing that is not affected by factors outside of the graph itself such as the number of evacuees and the capacity of the paths used. The existing algorithms in literature, CCRP and CCRP++ were not scalable in most cases due to their dependency on the timed information such as in time-expanded graphs or in repeated computations of the same path over time. The virtual evacuees were added to the synchronized flows to equalize the evacuation times of different paths. This process gets rid of the dependency on the timed information by allowing paths from different source nodes to be grouped as a synchronized flow. We strongly believe that our novel concept of synchronized flow will play a very important role in the evacuation routing in the future.

Our future works include the careful revision on the implementation of SyncFloE and the more thorough study on the synchronized flows and the design of contraflow evacuation routing based on the synchronized flows. We expect to improve the performance in terms of the evacuation time by carefully revising the implementation especially regarding correct logging of synchronized flows on the nodes, correct roll backs of changes, and more careful calculation of the virtual evacuees.

References

1. Cova, T.J., Johnson, J.P.: A network flow model for lane-based evacuation routing. Transportation Research Part A 37, 579–604 (2003)
2. Yang, F., Yan, X., Xu, K.: Evacuation Flow Assignment based on Improved MCMF Algorithm. In: Proc. First International Conference on Intelligent Networks and Intelligent Systems, pp. 637–640 (2008)
3. Fleischer, L.K.: Faster Algorithms For The Quickest Transshipment Problem. SIAM J. Optim. 12/1, 18–35 (2001)
4. Hamacher, H.W., Tjandra, S.A.: Mathematical Modelling of Evacuation Problems: A State of Art. Technical Report Nr. 24, Berichte des Fraunhofer ITWM (2001)
5. Kim, S., Shekhar, S.: Contraflow Network Reconfiguration for Evacuation Planning: A Summary of Results. In: Proc. Proceedings of the 13th ACM Symposium on Advances in Geographic Information Systems, pp. 250–259 (2005)
6. Kim, S., Shenkhar, S., Min, M.: Contraflow Transportation Network Reconfiguration for Evacuation Route Planning. IEEE Transactions on Knowledge and Data Engineering 20/8, 1115–1129 (2008)
7. Lu, Q., Huang, Y., Shekhar, S.: Evacuation Planning: A Capacity Constrained Routing Approach. In: Chen, H., Miranda, R., Zeng, D.D., Demchak, C.C., Schroeder, J., Madhusudan, T. (eds.) ISI 2003. LNCS, vol. 2665, pp. 111–125. Springer, Heidelberg (2003)
8. Lu, Q., George, B., Shekhar, S.: Capacity Constrained Routing Algorithms for Evacuation Planning: A Summary of Results. In: Medeiros, C.B., Egenhofer, M., Bertino, E. (eds.) SSTD 2005. LNCS, vol. 3633, pp. 291–307. Springer, Heidelberg (2005)
9. Lu, Q., George, B., Shekhar, S.: Evacuation Route Planning: Scalable Heuristics. In: Proc. 15th International Symposium on Advances in Geographic Information Systems (2007)
10. Min, M., Neupane, B.C.: An Evacuation Planner Algorithm in Flat Time Graphs. In: Proc. of ACM International Conference on Ubiquitous Information Management and Communication, ICUIMC (2011)
11. Yin, D.: A Scalable Heuristic for Evacuation Planning in Large Road Network. In: Proc. the Second International Workshop on Computational Transportation Science, pp. 19–24 (2009)