A Scalable Approach for Dynamic Evacuation Routing in Large Smart Buildings

†‡Van-Quyet Nguyen, †Huu-Duy Nguyen, §Quyet-Thang Huynh, ¶Nalini Venkatasubramanian, †Kyungbaek Kim*
†Department of Electronics and Computer Engineering, Chonnam National Universisty
‡Faculty of Information Technology, Hung Yen University of Technology and Education
§Hanoi University of Science and Technology
¶Department of Computer Science, University of California Irvine
Email: quyetict@utehy.edu.vn, huuduyce@gmail.com, thanghq@soict.hust.edu.vn,
nalini@ics.uci.edu, kyungbaekkim@jnu.ac.kr

***Abstract*—This paper considers the problem of dynamic evacuation routing in large smart buildings. We investigate a scalable routing approach which not only generates effective routes for evacuees but also quickly updates routes as the disaster status and building conditions could change during the evacuation time. We first design a flexible and scalable evacuation system for large smart buildings with multiple levels of computational support. Given such a system, we develop a novel distributed algorithm for finding effective evacuation routes dynamically by using an LCDT (Length-Capacity-Density-Trustiness) weighted graph model, which is built upon the current disaster information and building conditions. Finally, we propose a caching strategy which expedites dynamic route generation with the current effective route part(s) in order to improve the performance of dynamic evacuation in large buildings. To validate our approach, we test the proposed algorithm with our implementation of an evacuation simulator and compare the results with other approaches. Experimental results show that our approach outperforms other ones in the aspect of the evacuation time reduction and the maximum number of people being evacuated in each time span.**

***Index Terms*—Dynamic Evacuation Routing, Emergence Evacuation Systems, Evacuation Simulator, Distributed Graphs**

I. Introduction

With the development of the Internet of Things (IoT), smart buildings are becoming a reality with the support of smart devices such as smart indicators, smart sensors, smart cameras, and RFIDs. These smart devices play an important role in monitoring and tracking the events/conditions inside the building to provide useful information which helps the building management systems to make the right decisions in emergency situations. One of the indispensable systems in a large smart building is an emergency evacuation system, which guides evacuees to pass through exit gates as fast as possible. In that system, finding effective evacuation routes, which help to reduce the evacuation time and maximize the number of evacuees arriving at exit gates, is a hard problem because of uncertain of hazard conditions and possibility of congestions during the time of evacuation.

Recently, the weighted graph based approaches using IoT data in smart buildings have emerged to dynamically find the evacuation routes more effectively [1]–[3]. Nikolaos et al. [1] proposed a building evacuation system that computes the optimal evacuation routes in real-time. The authors use indicators (smart panels) as decision nodes to provide directions to evacuees, and a network of sensors in order to update the hazard intensity between decision nodes. The major drawback of this approach is the lack of consideration in the capacity (e.g., corridor width) and the density of evacuees in the corridors/rooms during evacuation time. Hence, it is the cause of crowd congestion on the evacuation routes. Sai-Keung et al. [2] proposed an optimized evacuation route based on crowd simulation. The authors use capacity and physical length to make a weighted graph, then they determine evacuation routes based on the shortest path algorithm. To reduce congestion on a route having high density, they develop an algorithm to find a *division point* which divides evacuees into two groups and evacuate in opposite directions. However, this approach does not take hazard intensity on the routes into account. A distributed evacuation guidance in large smart buildings has been proposed by Marin et al. in [3]. The authors consider building conditions and hazard intensity to recommend the best evacuation route for each evacuee through a smart sensor network and personal mobile devices. This approach has some limitations: (1) exploiting sensor data from evacuees' smartphones in order to count the number of people located at a certain space is a time consuming job (e.g., 15 minutes for batch processing as presented in their work); (2) their smartphone-based approach requires evacuees to carry devices for monitoring and tracking, which limits the usage scope of their system.

The existing systems have considered distributed approaches for finding evacuation routes in large smart buildings. The distributed approaches are required for handling massive data generated by a large number of devices (e.g., sensors, cameras) in a smart building to provide an evacuation system with real-time situation awareness [3]. Specifically, they separate an evacuation procedure in a large smart building into multiple small processes on each floor/sub-region. However, these approaches lack of the consideration of *global view* information, which is about the hazard intensity and the crowd congestion in the whole building. Consequently, in some cases, the evacuees following an evacuation route from a higher floor could face a hazardous area or crowd congestion when they move to

*Corresponding author.

the lower floors. Another limitation of the existing approaches is the late updating about the information of disaster events and congestion since obtaining them often takes much time. For instance, from a single image captured by a camera, using CNN-based technique for counting people density on a route segment of a building could take 100 milliseconds [4]. So, the duration of 10 seconds is needed for counting people on 100 route segments. Hence, determining only affected regions/nodes which should be updated is necessary to reduce the computation cost and quickly update the evacuation routes.

In this paper, we propose an LCDT (Length-Capacity-Density-Trustiness) layered graph mode in order to find the most effective evacuation routes dynamically in a distributed manner without losing global information. We first represent a network of smart indicators in a smart building as a distributed graph, where the subnetwork of indicators on each floor is represented as a subgraph, and a cross-graph is defined to represent the connection between floors. We then consider the factors: physical length, capacity, hazard intensity, and people density to weight the distributed graph. Next, each subgraph is locally evaluated to find the local evacuation routes on each floor. Afterward, we update all the weights of the cross-graph and find out effective global evacuation routes. Finally, we map local evacuation routes and global evacuation routes to find the final effective routes for every indicator. Moreover, a caching strategy is proposed to improve the effectiveness of evacuation routes. Our work makes the following contributions:

- We design a flexible and scalable evacuation system for large smart buildings based on the integration of IoT devices and a distributed graph based evaluation technique (Section III).
- We propose a so-called LCDT model, which represents the impact of disaster conditions and building conditions on every route segment in the building (Section IV).
- We propose a distributed algorithm that exploits the LCDT model for efficient evacuation route planning (Section V).
- We propose a caching strategy which generates effective evacuation routes more frequently in a given time, and it increases situation awareness of dynamic routes to improve their effectiveness (Section VI).
- We implement a simulation tool to evaluate and prove that our approach outperforms other ones (Section VII).

We next cover related work on evacuation routing for large buildings in the following section.

II. Related Work

Building evacuation methods have been studied in very early based on mathematical models [5], [6]. Dressler et al. [7] applied network flow techniques to find a good exit gate for evacuees in an emergency situation. A framework using the genetic algorithm for the evacuation of large-scale pedestrian facilities with multiple exit gates is presented by Abdelghany et al. in [8].

In recent years, the weighted graph based systems using IoT data in smart buildings have attracted much attention for dynamic evacuation routing. Nikolaos et al. [1] proposed a building evacuation system that computes the optimal evacuation routes in a real-time manner. Their approach calculates the weights of graphs based on physical length and hazard intensity obtained from smart sensors. An extension of this work with capacity consideration is presented by Desmet et al. in [9]. Other designs of intelligent indoor emergency evacuation systems is described in [10] [11].

A dynamic approach for optimal evacuation routing in hazardous environments is described in [12]. The information about density and hazard location is used to calculate optimal paths. The authors have not considered the scalability of their approach yet. Marin et al. [3] presented a distributed building evacuation system that considers building conditions and hazard intensity to recommend the best evacuation route for each localized evacuee. This approach has a limitation which is the smartphones carrying dependence of the evacuees. Another optimization technique can be applied to reduce computational cost of evacuation routing on large graphs is using caches. Balasubramanian et al. [13] proposed a multi-geography overlay structure that supports weighted least cost path queries with sources and destination in different geographies. The authors use cached-paths, which are the most frequently accessed, to prune search space. Other cache-based approaches for dynamic routing are presented in [14] [15].

III. Emergency Evacuation System for Large Smart Buildings

In this section, we present a design architecture of emergency evacuation system for large smart buildings. Then, we describe the modules of the proposed system in detail.

A. Overview of System Architecture

The proposed system architecture is given in Fig. 1, which includes three main modules: *Smart Indicators*, *Smart Guidance Agents* (SGAs), and *Global Coordinator* (GC). They operate based on the IoT data obtained from sensors and cameras. Specifically, a network of smart indicators on each floor is presented as a subgraph, where a node represents a smart indicator (called *Indicator Node*), and an edge represents a route segment between two indicators. In the subgraphs, the node represents a smart indicator at a stair of the building is defined as a *Stair Node*, while the one represents a smart indicator at an exit gate is defined as an *Exit Node*. The *SGAs* perform weighting the edges of these subgraphs based on disaster conditions and building conditions. Then, they locally evaluate the cost of evacuation routes on their responsibility floor from every *Indicator Node* to the *Stair Nodes* and between *Stair Nodes*. Afterward, the cost between *Stair Nodes* is used to globally evaluate the cost of evacuation routes from every *Stair Node* to the *Exit Nodes*. The global information is sent back to the *SGAs* to decide the optimal evacuation routes from the *Indicator Nodes* to the *Exit Nodes*.

B. Smart Indicators

The main function of a smart indicator is direction guidance based on effective evacuation routes which are found by the

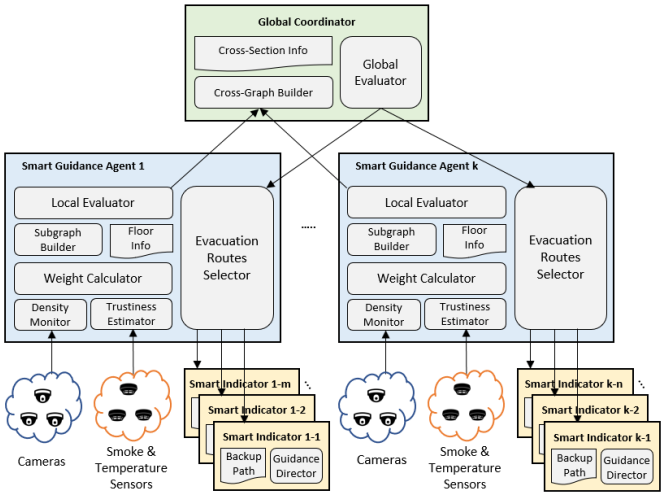


Fig. 1: Overview of system architecture.

corresponding SGA. It is performed by *Guidance Director* component which communicates with the corresponding SGA to receive the information about the next smart indicator. Besides, the *Guidance Director* can use a backup path for the guidance in case of disconnection with the SGA due to disaster events (e.g., SGA is broken). For doing this, a backup battery is required for the smart indicators.

C. Smart Guidance Agents

The SGAs are used for finding the evacuation routes in their physical space. The main components of a SGA include:

- *Weight Calculator*: This component performs weighting the impact of disaster and building conditions on the route segments associated with the smart indicators. A weight is calculated based on people density, location trustiness, and route segment information between two indicators which are gotten from *Floor Info*. The people density on each route segment is estimated by *Density Monitor* module. Meanwhile, the trustiness of a route segment is a real number in the range [0,1] that indicates the likelihood of the impact of a disaster event on the route segment, which is estimated by *Trustiness Estimator*. The weights of route segments will be used to update a weighted graph constructed by *Subgraph Builder*.
- *Local Evaluator*: This component evaluates the cost of evacuation routes from every *Indicator Nodes* to *Stair Nodes* by applying Dijkstra algorithm on a weighted subgraph generated by *Subgraph Builder*. It also calculates the cost between the *Stair Nodes*, and sends this information to the *GC*. Note that, the next smart indicator of each *Indicator Node* is not decided by *Local Evaluator* because we need to use the global view information before selecting the final effective evacuation routes.
- *Evacuation Routes Selector*: It is responsible for selecting an effective evacuation route for each *Indicator Node* in its subgraph. It combines the results from *Local Evaluator* and *Global Evaluator* in order to select the routes which have minimal cost from the *Indicator Nodes* to the *Stair Nodes*.

D. Global Coordinator

This module evaluates the overall evacuation routes by considering a graph of all *Stair Nodes* and *Exit Nodes*. The objective of providing global view information is to help the SGAs find the optimal routes which avoid hazardous and crowd congestion areas. To obtain this objective, the GC uses the following components.

- *Cross-Graph Builder*: It constructs a cross-graph of smart indicators based on the information in *Cross-Section Info*. In which, a node represents a *Stair Node* or an *Exit Node*, and an edge represents a route between two *Stair Nodes* or from a *Stair Node* to an *Exit Node*. The weights of edges in the cross-graph are assigned and updated by the information received from SGAs in every update interval (e.g., 10s, 30s).
- *Global Evaluator*: It computes the shortest paths over the cross-graph from every *Stair Node* to all *Exit Nodes*. The information about the shortest paths from a *Stair Node* to *Exit Nodes* is sent back to the corresponding *Evacuation Route Selector* in the SGAs for making final decisions.

IV. LCDT-based Weighted Graph Model for Providing Situation Awareness

We consider a network of smart indicators in the building as an undirected graph $G = (V, E)$, where V is a finite set of nodes, and E is a finite set of edges, $E \subseteq V \times V$. A node in the graph G represents a smart indicator. An edge (v_i, v_j) represents a route segment in the building between two adjacent smart indicators v_i and v_j.

To evaluate evacuation routes, our system performs assigning the evacuation costs on every edge of G. We can define a weighted graph of G as follows.

Definition 1 - Weighted Graph: A graph $G^w = (V, E, \omega)$ is a weighted graph if it is an undirected graph, and the function ω defined on E maps each edge $e \in E$ to a real number.

As we mentioned in the previous section, SGAs will calculate the weights of route segments. A weight value indicates the impact of disaster conditions and building conditions to the evacuation time on a route segment. The higher value of the weight, the higher evacuation time is needed. In order to estimate the weight of a route segment, we consider four factors: (1) physical length, (2) physical width, (3) hazard intensity, and (4) people density. Obviously, physical length is a fixed value, and the longer physical length, the more evacuation time is needed. For the physical width, it is used together with physical length to model the capacity (the number of people can move together) of a route segment. We now consider a route segment $R_{(i,j)}$ between two adjacent smart indicators v_i and v_j, which has the physical length $L_{(i,j)}$ and the physical width $H_{(i,j)}$. Assuming that an evacuee needs a minimum lane space $X \times Y$ (e.g., $X = Y = 1.0m$) for

moving, where X, Y are the physical length and the physical width of the lane space, respectively. So, the capacity of the route segment, C(i,j), can be calculated by using Equation 1.

$$C_{(i,j)} = (L_{(i,j)}/X) \times (H_{(i,j)}/Y) \quad (1)$$

Next, we consider the hazard intensity on a route segment, which is affected by a fire event. We model it by calculating trustiness of location $T_{(i,j)}$ based on the data received from a smoke sensor and a temperature sensor located at the route segment. Since this is not the main topic of this paper, we refer to our previous work which is a sensor-based approach for calculating trustiness of location presented in [16]. The people density of a route segment, denoted as $D_{(i,j)}$, is determined by the number of people on that segment. It can be obtained by implementing a CNN-based approach of people counting from a single image captured by a camera [4].

Finally, we introduce a so-called LCDT model to calculate the weight of every route segment $R_{(i,j)}$ using Equation 2.

$$\omega_{(i,j)} = \frac{L_{(i,j)}}{T_{(i,j)} \times (C_{(i,j)} - D_{(i,j)} + 1)} \quad (2)$$

In Equation 2, $T_{(i,j)} \in [0, 1]$, if $T_{(i,j)} = 1$, there is no affected of disaster event on $R_{(i,j)}$. In contrast, $T_{(i,j)} = 0$ means that the route segment $R_{(i,j)}$ has been strongly affected by disaster event. As the result, when $\omega_{(i,j)}$ get an infinite value, the evacuation routes should not include $R_{(i,j)}$. We assume that $D_{(i,j)}$ is always less than or equal to $C_{(i,j)}$.

V. A Distributed Approach for Finding Dynamic Evacuation Routes

In this section, we present a distributed approach for finding optimal evacuation routes with global view consideration and multiple levels of computational support (i.e. local and global computation). To do this, we first define a distributed weighted graph based on the concept of the weighted graph in the previous section. We then propose the algorithms for local and global evaluation of evacuation routes in SGAs and GC, respectively.

Definition 2 - Distributed Weighted Graph: A distributed weighted graph, G^d, for a weighted graph $G^w = (V, E, \omega)$ is a set of N subgraphs $\{G_i = (V_i, E_i, \omega) \,|\, i \leq N, V_i \subset V, E_i \subset E\}$ and a cross-graph $G_c = (V_c, E_c, \mu)$, where E_c is a set of edges that connect subgraphs, called cross-edges; V_c is a set of nodes that have cross-edges to or from subgraphs; μ is the function defined on E_c which maps each edge $(u, v) \in E_c$ to a real number.

In our context, N subgraphs are corresponding to the number of floors as well as the number of SGAs. G_i represents a network of smart indicators at the floor i^{th}, and G_c represents a network of smart indicators located at stairs and exit gates in the smart building.

Finding evacuation routes involves three phases as follows.

Local Evaluation. Algorithm 1 illustrates the procedure of finding the options of evacuation routes for very indicator in a given subgraph. We have to find the effective routes from all nodes in the subgraph to *Stair Nodes* or *Exit Nodes*. To end this, we employ Dijkstra algorithm on the weighted graph G_i with source nodes are a set of *Stair Nodes* S_i, and the destination nodes are all nodes in V_i, resulting in a complexity of $O(|S| \cdot |E_i| \cdot log|V_i|)$, with $|S|$ is the number of *Stair Nodes* in the G_i (every G_i has the same number of *Stair Nodes*) (lines 4-5). We get the weight and the next node from each indicator to each *Stair Node* during the searching (lines 7-8). This information is kept to be used for selecting the final evacuation route for each indicator in *Evacuation Route Selection* phase (line 9). Also, the weights among pairwise of *Stair Nodes* are calculated and stored in a weight matrix W (lines 10-12). The matrix W represents the weighted-based shortest distances between *Stair Nodes* in S_i ($i > 1$) or from *Stair Nodes* to Exit Nodes in case of evaluation on G_1 (exit gates on the first floor). Once the local evaluation phase is done, W is sent to the GC.

Algorithm 1 *Local Evaluation*

Input: A subgraph: $G_i = (V_i, E_i, \omega)$; a set of stair nodes in the floor i^{th}: S_i
Output: W is a matrix which contains the weight between pairs of stair nodes.

1: **for each** ind in V_i **do**
2: ind.NextOptions $\leftarrow \emptyset$;
3: **end for**
4: **for each** s in S_i **do**
5: $SPT \leftarrow$ Dijkstra(s, G_i); *// SPT: Shortest Path Tree*
6: **for each** ind in V_i **do**
7: $weight2S \leftarrow$ GetWeight(ind,s,SPT);
8: $next \leftarrow$ GetNextNode(ind,s,SPT);
9: ind.NextOptions.Add($< next, weight2S, s >$);
10: **if** ($ind \in S_i$ **and** $ind \neq s$) **then**
11: $W_{(s,ind)} \leftarrow weight2S$;
12: **end if**
13: **end for**
14: **end for**
15: **return** W;

Algorithm 2 *Global Evaluation*

Input: $G_c = (V_c, E_c, \mu)$ and a set of exit nodes Φ
Output: ω is a matrix which contains the weight from stair nodes to exit nodes.

1: **for each** e in Φ **do**
2: $SPT_c \leftarrow$ Dijkstra(e, G_c);
3: **for each** s in $V_c \setminus \Phi$ **do**
4: $weight2E \leftarrow$ GetWeight(s,e,SPT_c);
5: $\omega_{(s,e)} \leftarrow weight2E$;
6: **end for**
7: **end for**
8: **return** ω;

Global Evaluation. This phase performs evaluating the effective evacuation routes from every *Stair Node* to every

Algorithm 3 *Evacuation Routes Selector*

Input: $V_i, \{\omega(s,e)\}$, with $s \in S_i$ and $e \in \Phi$
Output: The next nodes for all indicator nodes in V_i

```
1:  for each ind in V_i \ Φ do
2:      ind.Weight2E ← ∞;
3:      for each opt in ind.NextOptions do
4:          for each e in Φ do
5:              weight ← opt.weight2S + ω(opt.s, e);
6:              if (weight < ind.Weight2E) then
7:                  ind.Weight2E ← weight;
8:                  ind.Next ← opt.next;
9:              end if
10:         end for
11:     end for
12: end for
```

Exit Nodes. The main procedure is presented in Algorithm 2. The input of this algorithm is a cross-graph, G_c, which is built as the following: (1) each *Exit Node* e is connected to every *Stair Node* in G_1; (2) each *Stair Node* in G_i, with $i > 1$, is connected to each other; (3) the connections among *Stair Nodes* in two adjacent floors are set up based on layout information of the building which is stored in *Cross-Section Info* component of GC; (4) the weights of the edges in G_c are assigned based on cost matrices W received from SGAs. After constructing the cross-graph, we apply Dijkstra algorithm to find the effective evacuation routes from every *Exit Node* to all *Stair Nodes* in G_c (lines 1-2). The weights from from every *Stair Node* to every *Exit Node* are calculated and stored in a weight matrix ω (lines 3-4). These weights will be sent back to the corresponding SGA.

Evacuation Routes Selection. This is the last phase in finding the optimal evacuation routes, and it is illustrated in Algorithm 3. The algorithm calculates the weight-based shortest distances from each node $v \in V_i$ to all *Exit Nodes* based on the weight matrices W and ω (lines 1-5). The route which has minimal weight will be selected for each v (lines 6-8). We formalize the evacuation routes selection as the following equations:

$$P_{(v,e_t)} = argmin_{e_t}\{W_{(v,e_t)}\} \quad \textit{if} \quad i = 1 \quad (3)$$

$$P_{(v,S_i{}^j,e_t)} = argmin_{e_t}\{W_{(v,S_i{}^j)} + \omega_{(S_i{}^j,e_t)}\} \quad \textit{if} \quad i > 1 \quad (4)$$

where e_t is an *Exit Node* and $e_t \in e$; $S_i{}^j$ is a *Stair Node* and $S_i{}^j \in S_i, j \leqslant |S|$. Note that, the case of $i = 1$ means that we are considering node v on the first floor. Based on the selected route P, the next indicator is assigned for each node v in V_i.

VI. Caching Strategy for Effective Dynamic Evacuation Routes

A. Observation that Motivate Caching

We have a key observation that evacuees usually keep following an effective route until a disaster event or crowd congestion happens on that route. Thus, the weights of effective routes will be changed as the people moving while other routes may be not much changed. Therefore, caching the effective routes for updating their weights and finding only affected indicators to immediately update their direction could help reducing the computation cost of the proposed algorithm as well as improving the efficiency of evacuation.

B. A Caching Strategy

We consider caching effective routes on every subgraph. It is not difficult to find that caching the effective routes between every pair of nodes or the long routes is impossible due to the large size of the subgraphs and the cache might not fit into available storage space. Therefore, choosing some nodes in the subgraphs and caching the effective routes among them could be preferred as a solution.

Choosing cache nodes. The fact that the stairs and exit gates are located at positions where the evacuees can quickly find out when disasters happen. During that time, if any stairs and exit gates are not available, the evacuees have to find other available ones. Thus, we first choose the stairs and exits nodes in the subgraphs as the cache nodes and the routes among them as the cache paths. However, in case of large smart buildings, the distance between two stairs/exit nodes u and v could be long, caching a long path between them may not efficient as we discussed above. In that case, some intermediate nodes u' should also be chosen as the cache nodes that satisfy the following conditions: (1) most of the paths from u to v passing through these nodes (e.g. nodes at intersections), (2) the distances $d(u,u')$ and $d(u',v)$ are less than a given distance (e.g., 50 meters), and (3) the distance $d(u,v) = d(u,u') + d(u',v)$ is minimal.

Finding cache paths. We prefer using k paths between each pair of cache nodes with priority consideration. We identify top-k shortest paths such that they are dissimilar with each other and their total weight is minimized. This is necessary to switch from the current shortest path to another path when disaster event or crowd congestion happens. Here, we consider physical length and width between two indicator nodes for weighting the edges of the subgraphs when finding top-k shortest paths. A simple way to weight is using weight function $\omega_{(i,j)} = L_{(i,j)}/H_{(i,j)}$. We define the length of a path P, denoted as $L(P)$, is the sum of the edge weights in P. To find dissimilar paths, we first formally define the path similarity.

Definition 3 - Path Similarity: The similarity between two paths P_i and P_j, denoted as $S(P_i, P_j)$, is a real number ranged from 0 to 1 that indicates the ratio between the length of the shared edges and the length of the union of edges.

There are several similarity functions that can be applied to measure the path similarity. In this paper, we use *cosine similarity* function, and the similarity between two paths P_i and P_j can be formalized as follows.

$$S(P_i,P_j) = \frac{\sum\limits_{(u,v)\in(P_i\cap P_j)} \omega^2_{(u,v)}}{\sqrt{\sum\limits_{(a,b)\in P_i} \omega^2_{(a,b)}}\sqrt{\sum\limits_{(c,d)\in P_j} \omega^2_{(c,d)}}} \quad (5)$$

Algorithm 4 *Find k effective paths between two cache nodes*

Input: A subgraph G, source s, destination t, number of cache paths k, upper bound length θ
Output: The top-k efficient paths for caching

$\Psi \leftarrow$ FindAllPaths(s, t, G, θ)
$\Psi \leftarrow$ SortByLength(Ψ)
$P \leftarrow$ SelectTopKShortestPaths(Ψ, k)
$\vartheta \leftarrow$ ScoreKPaths(P)
$\Omega \leftarrow \Psi \setminus P$
for $i = k$ **to** $|\Omega|$ **do**
  for $j = k - 1$ **to** 0 **do**
    $P' \leftarrow P \setminus \{P[j]\}$
    $P' \leftarrow P' \cup \{\Omega[i]\}$
    $\sigma \leftarrow$ ScoreKPaths(P')
    if $\sigma > \vartheta$ **then**
      $P \leftarrow P'$
      $\vartheta \leftarrow \sigma$
    end if
  end for
end for

We can select *k* dissimilar paths by using path similarity. However, this is insufficient to guarantee the total length of these *k* paths is minimized. Therefore, we use another parameter named *coefficient of length deviation*. The coefficient of length deviation between two paths P_i and P_j, denotes as $\Delta(P_i, P_j)$, is calculated by using the following equation.

$$\Delta(P_i, P_j) = \frac{L(P_j) - L(P_i)}{L(P_j)} \tag{6}$$

Note that, the value of $\Delta(P_i, P_j)$ is in the range [0,1] since we sort the list of paths by length so that $L(P_j)$ is always greater than or equal to $L(P_i)$ before calculating the $\Delta(P_i, P_j)$.

We propose a heuristic algorithm for finding k effective paths between two cache nodes as shown in Algorithm 4. This algorithm evaluates possible sets of k paths by scoring them based on the path similarity and the coefficient of length deviation. The scoring procedure is presented in Algorithm 5.

Cache Structure. We cache k effective paths of each pair of cache nodes, (a, b), into memory in the structure of $< (a, b), \rho, \kappa_{(a,b)} >$ where $\kappa_{(a,b)}$ is list of k cache paths from a to b; ρ is the current selected path from a to b and $\rho \in \kappa_{(a,b)}$. We assume that the size of available cache memory of SGAs is enough for storing their own cache paths.

C. Updating Evacuation Routes using Caches

The basic idea of using caches to update the evacuation routes is that we check the conditions of all current selected cache paths on SGAs in every given checking interval, τ, if any current selected cache path, ρ, has congestion and/or strong affected by a disaster event (low trusted value), we will select the next one in corresponding k cache paths. Then, the indicators which belong to ρ will have their direction updated. Here, the checking interval τ is much smaller than the update interval of finding evacuation routes in the whole building. Therefore, calculating and updating routes using cache paths do not take much time while quickly update effective routes for evacuees. The procedure updating routes using caches is shown in Algorithm 6.

Algorithm 5 *ScoreKPaths(P)*

Input: A set of k paths P, the coefficient α of weighting path similarity and coefficient of length deviation, $\alpha \in [0, 1]$
Output: The score of P

$\vartheta \leftarrow 0$
for $i = 0$ to $|P|$-1 **do**
  for $j = i + 1$ to $|P|$ **do**
    $\vartheta \leftarrow \vartheta + \alpha \cdot (1 - S(P[i], P[j])) + (1 - \alpha) \cdot \Delta(P[i], P[j])$
  end for
end for
return ϑ

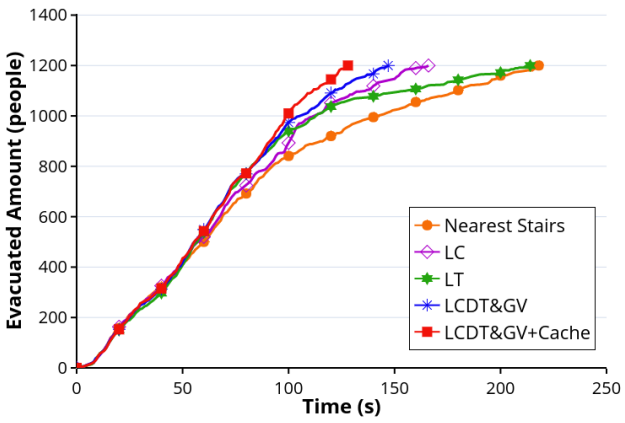
Algorithm 6 *Update Evacuation Routes with Caches*

Input: A set of cache paths κ, the threshold for checking the strong affecting of a disaster event ξ, the threshold for checking crowd congestion β.
Output: Update direction for affected indicators

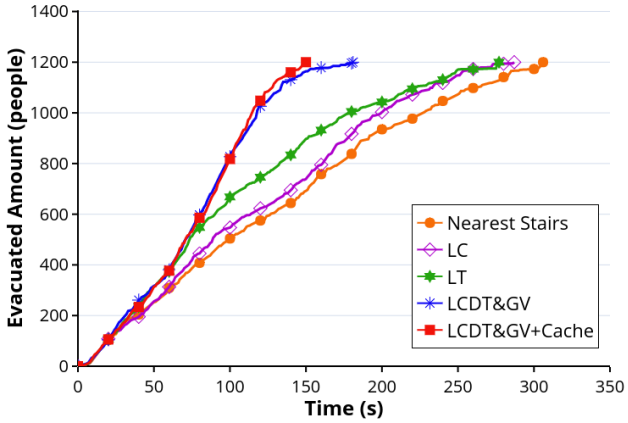
for each path ρ in κ **do**
  $a \leftarrow$ the source node
  $b \leftarrow$ the destination node
  $density \leftarrow 0$
  $capacity \leftarrow 0$
  $minTrust \leftarrow 1$
  for each edge (i, j) in ρ **do**
    $density \leftarrow density + D_{(i,j)}$
    $capacity \leftarrow capacity + C_{(i,j)}$
    if $T_{(i,j)} < minTrust$ **then**
      $minTrust \leftarrow T_{(i,j)}$
    end if
  end for
  */*Check conditions and update direction*/*
  if $minTrust < \xi$ **or** $density > \beta \cdot capacity$ **then**
    $\rho \leftarrow \kappa_{(a,b)}$.Next
    $\varphi \leftarrow$ list nodes in ρ
    for $i = 0$ to $|\varphi|$-1 **do**
      $\varphi[i]$.Next = $\varphi[i + 1]$
    end for
  end if
end for

VII. Evaluation

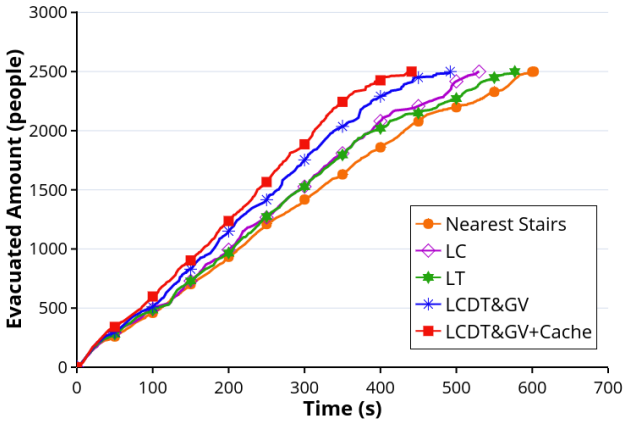
To evaluate the effectiveness of our proposed approach, we implemented a simulation tool for emergency evacuation in smart buildings. We generate two graphs of indicators based on the layouts of two large buildings. The first one is generated by mimicking the layout of a real building named Donald Bren Hall at University of California Irvine that is a six-story building (called DBH). The second one is generated by using a synthetic building that is a ten-story building (called TSB) for evaluating the scalability of our approach as well as for deeper test. The former has 3 stairs, 4 exit gates, and



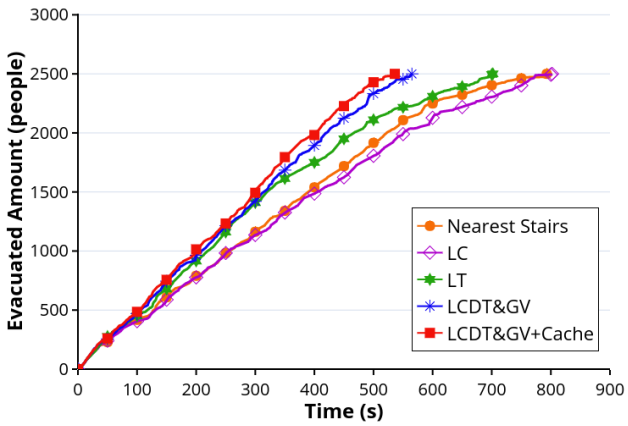
(a) Normal regions in Donald Bren Hall



(b) Critical regions in Donald Bren Hall



(c) Normal regions in Ten-Story Building



(d) Critical regions in Ten-Story Building

Fig. 2: Comparison of the effectiveness among evacuation methods with varied locations of disaster events

the area of each floor is approximately 2,300 square meters (24,800 square feet). While the later one has 4 stairs and 3 exit gates, and the area of each floor is approximately 8,300 square meters (89,300 square feet). We assume that elevators are not available during evacuation time.

We simulate fire events in the buildings in two scenarios: (1) the fire event occurs at normal regions and (2) the fire event occurs at critical regions. In the normal case, the fire event affects a stair on the first floor and its close regions. Regarding the critical case, the fire event affects a stair on the first floor as well as 2 exit gates and their close regions. We assume that the impact of fire events on route segments are estimated and assigned by random trustiness values in the range [0.1,0.9]. The fire spreading is simulated by randomly updating trusted values and expanding the affected regions with a given spreading rates in every ten seconds of the first minute after the fire event happened. We consider fire spreading problem in smart buildings as the future work to make our approach more realistic in evacuation systems. The number of evacuees on each floor in DBH is 200, while the one in TSB is 250. The position of evacuees are randomly generated that each person is close to an indicator, and there are not more than 20 people near an indicator. The initial speed of each person is also randomly generated which ranges from 2.0 m/s to 5.0 m/s. The moving step of each person is updated in every 200 milliseconds.

In the first experiment, we evaluate the efficiency of our proposed approach by comparing with other evacuation approaches. We implement our LCDT-based approach for evacuation routing with global view information consideration (called LCDT&GV) and the improvement algorithm using caches (called LCDT&GV+Cache). We also implement three other approaches: Shortest-Path-Length based approach which evacuees use the nearest stairs to pass through exits (called *Nearest Stairs*), Length-Capacity-based approach (called LC) which mentioned in [2], and Length-Trustiness-based approach (called LT) in [1]. The update interval for calculating the weights of all route segments in the building is set to 10s for our approaches and LT approach. With our LCDT&GV+Cache approach, the checking interval for using caches is set to 2s; the coefficient α for weighting path similarity and coefficient of length deviation is set to 0.5; the threshold for checking the strong affecting of a disaster event ξ equals to 0.3; the threshold for checking crowd congestion β

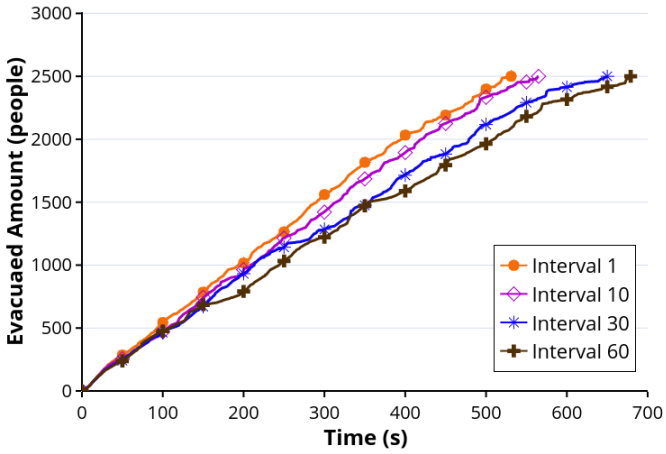


Fig. 3: The impact of update interval on evacuation system.

equals to 0.5. We run simulations on a computer with 32.0 GB of RAM and 12 CPU cores, and each SGA is run on different CPU in parallel.

Figure 2 illustrates the comparison of the effectiveness among evacuation methods. In overall, our LCDT-based approaches outperform other approaches in all simulation scenarios with both the evacuated amount and the total evacuation time. In which, our approaches take less time, which the performance is up to 2 times faster compared to other approaches, to evacuate all the people. Note that, in the simulations, we assume that all people could be evacuated with unlimited simulation time. Obviously, using *Nearest Stairs* approach is not efficient in most of the simulation scenarios because when the evacuees choose the nearest stairs, they could face with the area having fire events. As a result, they have to find other stairs or exit gates, and congestion could occur in this case, which causes the time-consuming. In the respect of LC approach, since it considers the length and capacity of the routes, it almost achieves a better result comparing to *Nearest Stairs* approach. However, its performance is not good enough because it still does not take the disaster areas into account. Therefore, similar with *Nearest Stairs* approach, the evacuees will need more time to pass to other routes, which the drawback is significantly shown in Fig. 2(d). With LT approach in the cases that disaster events happened at critical regions, it achieves better performance compared to LC and *Nearest Stairs* because it considers the impact of disaster events on the routes. Nevertheless, in the case that disaster event happened at the normal regions, its performance is not much different with LC approach. Moreover, at the later stage of the evacuation time, LT could not perform a better result comparing to LC as shown in Fig. 2(a) and 2(c), since with the latter there is not much congestion like with the former. Regarding our LCDT&GV approach, we achieve better results because we consider more necessary parameters with global view information. Furthermore, LCDT&GV+Cache approach, which employs cache paths, enhances the efficiency of the original one in all the cases.

The second experiment investigates how the update interval of the weights in evacuation system affects the efficiency of our LCDT&GV approach. To do this, we tested with four update intervals: 1, 10, 30, and 60 second(s) in case of fire events happened at critical regions. We run simulations on TSB dataset with the conditions described above. For each update interval, we run simulation three times and get the average value of both evacuation time and evacuated amount. Figure 3 shows the impact of the update interval on the effectiveness of the evacuation system. We observed that a small update interval could help the system to reduce the evacuation time while increasing the number of evacuees passing to exit gates in each time span. For instance, the evacuated amount at time span 400(s) is around 2000, 1900, 1700, 1600 corresponding to the update intervals 1s, 10s, 30s, and 60s, respectively. As a result, the total of evacuation time with the small update intervals is less than the larger ones.

VIII. Conclusion

This paper proposed a scalable approach for dynamic evacuation routing by using an LCDT weighted graph model, which is built upon the disaster information and building conditions. We designed a flexible and scalable evacuation system for large smart buildings based on the integration of IoT devices and a distributed graph based evaluation technique. We then proposed a distributed algorithm that exploits the LCDT model for dynamic evacuation routing. Moreover, we presented a caching strategy to improve the performance of the proposed approach. We evaluated our proposal on both real building and synthetic building with an evacuation simulator. Experimental results indicated that our approach outperforms other ones in the aspect of the evacuation time reduction and the maximum number of people being evacuated in each time span.

Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning(NRF-2017R1A2B4012559). This work is supported by NIST under award No.70NANB17H285. This work is supported by DARPA under grant FA8750-16-2-0021.

References

[1] N. Dimakis, A. Filippoupolitis, and E. Gelenbe, "Distributed building evacuation simulator for smart emergency management," *The Computer Journal*, vol. 53, no. 9, pp. 1384–1400, 2010.

[2] S.-K. Wong, Y.-S. Wang, P.-K. Tang, and T.-Y. Tsai, "Optimized evacuation route based on crowd simulation," *Computational Visual Media*, vol. 3, no. 3, pp. 243–261, 2017.

[3] M. Lujak, H. Billhardt, J. Dunkel, A. Fernández, R. Hermoso, and S. Ossowski, "A distributed architecture for real-time evacuation guidance in large smart buildings." *Comput. Sci. Inf. Syst.*, vol. 14, no. 1, pp. 257–282, 2017.

[4] V. A. Sindagi and V. M. Patel, "A survey of recent advances in cnn-based single image crowd counting and density estimation," *Pattern Recognition Letters*, vol. 107, pp. 3–16, 2018.

[5] P. A. Thompson and E. W. Marchant, "A computer model for the evacuation of large building populations," *Fire safety journal*, vol. 24, no. 2, pp. 131–148, 1995.

[6] G. G. Lovas, "On the importance of building evacuation system components," *IEEE Transactions on Engineering Management*, vol. 45, no. 2, pp. 181–191, 1998.

[7] D. Dressler, M. Groß, J.-P. Kappmeier, T. Kelter, J. Kulbatzki, D. Plümpe, G. Schlechter, M. Schmidt, M. Skutella, and S. Temme, "On the use of network flow techniques for assigning evacuees to exits," *Procedia Engineering*, vol. 3, pp. 205–215, 2010.

[8] A. Abdelghany, K. Abdelghany, H. Mahmassani, and W. Alhalabi, "Modeling framework for optimal evacuation of large-scale crowded pedestrian facilities," *European Journal of Operational Research*, vol. 237, no. 3, pp. 1105–1118, 2014.

[9] A. Desmet and E. Gelenbe, "Capacity based evacuation with dynamic exit signs," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*. IEEE, 2014, pp. 332–337.

[10] Q. Zhang, T. Chen, and X.-z. Lv, "New framework of intelligent evacuation system of buildings," *Procedia engineering*, vol. 71, pp. 397–402, 2014.

[11] J. Liu, F. Lin, E. Chu, and J.-L. Zhong, "Intelligent indoor emergency evacuation systems: Reference architecture and data requirements," in *Future Technologies Conference (FTC)*. IEEE, 2016, pp. 600–609.

[12] P. Boguslawski, L. Mahdjoubi, V. Zverovich, and F. Fadli, "A dynamic approach for evacuees' distribution and optimal routing in hazardous environments," *Automation in Construction*, vol. 94, pp. 11–21, 2018.

[13] V. Balasubramanian, D. V. Kalashnikov, S. Mehrotra, and N. Venkatasubramanian, "Efficient and scalable multi-geography route planning," in *Proceedings of the 13th International Conference on Extending Database Technology*. ACM, 2010, pp. 394–405.

[14] J. R. Thomsen, M. L. Yiu, and C. S. Jensen, "Effective caching of shortest paths for location-based services," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, pp. 313–324.

[15] Y. Zhang, Y.-L. Hsueh, W.-C. Lee, and Y.-H. Jhang, "Efficient cache-supported path planning on roads," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 4, pp. 951–964, 2016.

[16] V.-Q. Nguyen and K. Kim, "Study on location trustiness based on multimodal information," in *Proceedings of the 4th International Conference on Smart Media and Applications (SMA)*, 2016.