Evacuation in Buildings: Path-Finding for Real-Time Evacuation Systems

Po-Han Chen[1] and Feng Feng[2]

Abstract—**Finding the shortest path between any two nodes in a network is a classical problem in a number of research areas. Although some general shortest path algorithms have already been developed, such as Dijkstra's shortest path algorithm, in some situations, their computation complexity is too high to be of any practical use.**

In some real-time computation systems, computing the shortest path in a short time is a basis on which the computation systems could be applied to solve real world problems. The real world problem specifies the requirements on the computation speed, software and hardware limitations and the accuracy level for an algorithm. As this research is to find the shortest path in a large indoor area and the algorithm developed would be implemented on a real-time emergency evacuation system, it is required that the computing speed be fast enough even the network is large in terms of the numbers of nodes and arcs.

To solve the above problem, some common inside-building topologies have to be analyzed, followed by the development of fast shortest path finding algorithms for the topologies, which frequently appear in indoor areas like campus and commercial buildings. This series of algorithms combined classical shortest path algorithms in graph theory with interval routing techniques in computer networks in order to get a high computation speed especially for large scale networks.

The developed fast algorithms could serve as the main shortest path finding tool in emergency evacuation systems for large indoor areas, and extend traditional evacuation systems from virtual simulation to real-time problem solving.

Index Terms—**1-IRS, dynamic interval routing schemes, large indoor area routing, shortest path finding.**

I. Introduction

THE emergency evacuation system, for which we are developing path finding algorithms, is a mobile and wearable computing system. It supports location-aware computing, and offers services that are relevant to the mobile user's current location [1]. Mobile user's outdoor location can be given by GPS. And by now, some commercial location trackers have been able to give indoor location information with a wireless network, like IEEE 802.11 Wi-Fi wireless LAN. The average accuracy of this type of location tracker can be up to 1 meter. When a building's indoor area, especially passageways and staircases, is abstracted to be a mini transportation network in certain way, the shortest path from the mobile user's current location to any location in the network can be given by certain algorithms in graph theory. Dijkstra's shortest path algorithm [2] is the classic one, which can be used for the networks in any topologies. But for some special topologies, other algorithms can be applied independently from any Dijkstra's thoughts. Interval routing schemes (IRS) in computer network routing are in this category and is the main topic of this paper.

In part II, the network abstraction method of a building's indoor structure is introduced. On the basis of this abstraction, some frequently appearing indoor topologies are taken out as the major subject for the shortest path algorithms.

In part III, literatures on shortest path finding algorithms are reviewed. Static IRS algorithms are introduced for above frequently appearing indoor topologies.

In part IV, Dynamic IRS algorithms are given for the topologies above.

In part V, Dynamic IRS algorithms are combined with Dijkstra's shortest path algorithms in order to give shortest paths in networks with much more complext topologies.

In part VI, conclusions are given and possible future works are discussed.

II. Network Abstraction and Frequently Appearing Topologies inside Buildings

A. Network Abstraction

Most large indoor areas consist of one or more than one builds connected by some passageways or doors. A path between any two points in the area should be within its free space. The free space refers to all the areas within this large indoor area, which allow people to walk through. All the passageways, staircases and any empty space confined by doors, internal and external walls of the building may be deemed as free space. As for our path finding problem, passageways and staircases are our concerns; and it is assumed that our algorithms only deals with paths within passageway and staircase areas.

In robot motion planning, path finding is a traditional problem and this free space, denoted by $C_{free,}$ is called work space or configuration space for the movement of a robot.

[1] Assistant Professor, School of Civil & Environmental Engineering, Nanyang Technological University, Singapore 639798

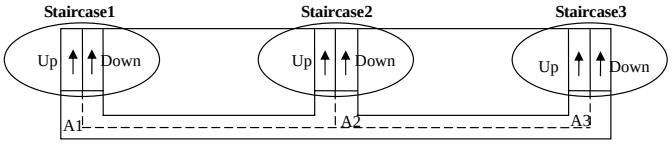
[2] Research Student, School of Civil & Environmental Engineering, Nanyang Technological University, Singapore 639798

There are a number of ways to convert C_{free} to be a network, which is a convenient data structure for shortest path calculation. These include some exact cell decomposition methods, such as triangular and trapezoidal decomposition [3-7]. Some road map methods, such as visibility graphs [8] and Voronoi diagrams [9, 10], are also commonly used for the path calculation in robot motion planning. For the same free space, there are a number of methods to do the network conversion. And the resulting networks for the same C_{free} may be much different due to the methods.

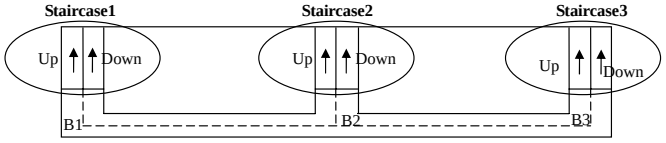
Exact cell decomposition and road map methods and visibility graphs work when the location information and motion control are very accurate. But in our case, the location tracker can't provide the accuracy of this high level and the evacuation task's property does not need so high accuracy either. So a simplified version of accessibility graph [11] is used for this research.

B. Frequently Appearing Topologies inside Buildings

One topology appearing frequently in most flat buildings is 2D grid. Let look at a frequently appearing layouts in most flat buildings first. In Figure 1 (a) and (b), the accessibility graphs for two neighboring floors A and B are shown. The building spatial maps are drawn after the formal graduate apartment at Block 553, Upper Jurong, Singapore, which is a 10-floor flat building.



(a) Accessibility Graph: Floor A



(b) Accessibility Graph: Floor B

Figure 1 Accessibility Graphs for a 2D Grid Structure

The dashed lines are the accessibility graphs for floors A and B. A is one storey above B. Floors A and B are connected by three staircases. Each staircase connects the all the floors of the building. In the accessibility graph, the path between node A1 and B1 are a sequence of connected segments from A1 to staircase 1 to B1. If we virtually straighten this segment (convenient for path calculation) sequence to make it one arc from A1 to B1, and do the same manipulations on the paths A2 to B2 and A3 to B3 on all other floors, the accessibility graphs of the all floors can be simplified logically into one overall accessibility graph which is looked as in Figure 2.

Each vertical segment in Figure 2 is actually a sequence of connected segments, but this straightening manipulation simplifies the path finding calculation significantly and does not affect the accuracy of calculation result. When a path is calculated and there are any this kind of vertical arcs included in the path, only a simply conversion from the arcs' logical forms in the calculation process to its physical forms in reality is needed in order to present the path found on virtual reality display devices.

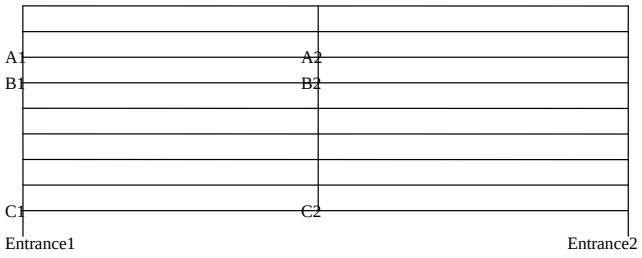


Figure 2 the Simplified Accessibility Graph for a 2D Grid

Some other structures can be found in the form of trees and rings. One example of a tree structure is a typical unit of most of blocks on Jurong West, Singapore. It is a unit in a 12-story apartment building. All floors of the unit are connected by a staircase. On each floor of the unit there are 4 apartments. The simplified accessibility graph for one floor is in Figure 3 and for the whole building in Figure 4.

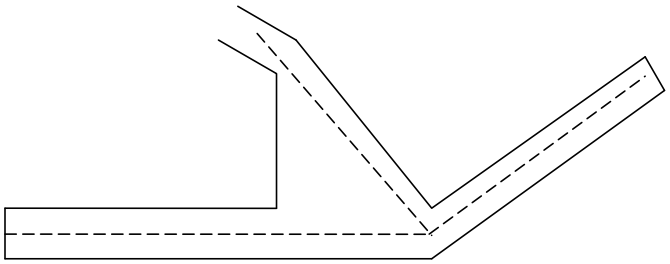


Figure 3 Single Floor Layout for a Tree

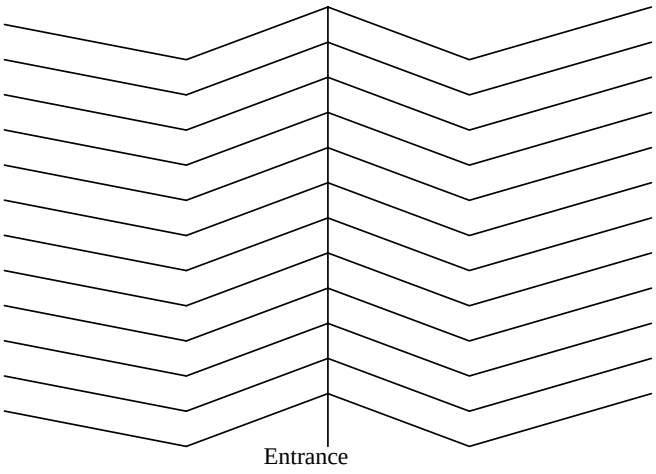


Figure 4 Tree Structure of a Unit in a Block

Rings also frequently appear in inside-building topologies. Some on-campus residence halls of Nanyang Technological University, Singapore, take this form.

2D grids, trees and rings are high regular topologies.

Therefore, some efficient algorithms can be developed in consideration of the properties of these topologies. The Interval routing schemes (IRS) are a series of routing techniques for high regular topologies and the literatures on IRS are reviewed in part III.

III. Literature Review

For high regular topologies, such as grids, trees and rings, shortest path finding calculation may need no Dijkstra's algorithm involved. And alternative routing methods, usually used in computer network routing can be used. The routing method significantly decreases the real-time path computation load on the system and makes it a good alternative for Dijkstra's algorithm, which takes no consideration of the network topology.

Interval routing is a way of implementing routing schemes on arbitrary networks. A routing scheme is a strategy that assigns to every source-target pair the path that a message from the source to the target should take. The concept of routing scheme was originally developed for computer network. For example, one possible routing scheme is to store a complete routing table in each of the n vertices of the network, specifying for each target the next edge in some shortest path over which the message must be forwarded. But with the number of the nodes increases, the size of the routing tables stored at each node would increase as well.

Another way of implementing routing schemes, called interval routing, has been discussed in a number of research articles, which has been summarized by Gavoille [12]. Actually interval routing scheme is based on representing the routing table stored at each node in a compact manner. In this method, each node is assigned a distinct label from the set {1, 2… n}. Arcs are bi-directional and are labeled with one or several subintervals of the interval [1…n] so that for any node v the intervals associated with outgoing edges from v are pair-wise disjoint and their union covers [1…n]. When a message with destination v arrives at node u, the message is forwarded on the unique outgoing edge labeled with an interval containing the label of v.

In most cases, [1…n] is the cyclic interval, i.e., all subintervals are understood to wrap around. Such a scheme is the called a circular interval routing scheme (CIRS). Variants of the scheme include linear interval routing schemes (LIRS), in which [1…n] is viewed as a linear interval; k-interval routing schemes, in which edges can be labeled with at most k intervals (k-IRS). An interval routing scheme for which all messages are routed along shortest paths is called an optimal scheme.

An example of circular interval routing scheme is showed in Figure 5.

In Figure 5, nodes are first labeled with numbers from 1 to 7. Each arc is given an interval for each direction. If we are trying to find the path from node 5 to node 2, first scan all intervals on all arcs outward from node 5. We find that the number 2 exists in the circular interval [6, 4], (the circular interval [6, 4] includes {6, 7, 1, 2, 3, 4}) so arc from node 5 to node 4 is chosen. With similar methods, the arc from node 4 to node 3 is chosen. At node 3, there are 3 intervals, among which [1, 2] includes number 2. Therefore the path is constructed as 5-4-3-2. At each node, the next arc to follow is determined by whether the destination node's label is bracketed by the interval of that arc.

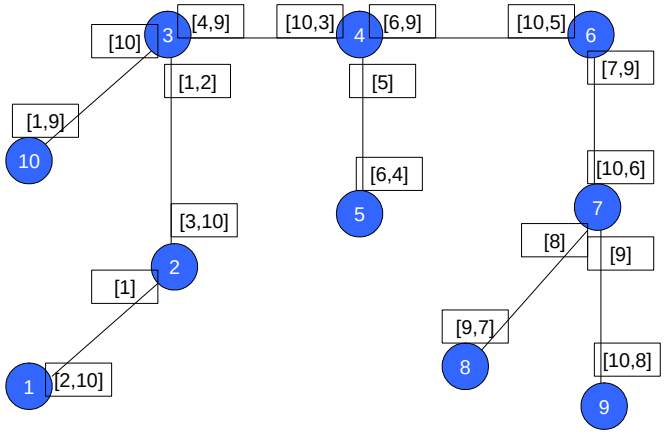


Figure 5 An Example of 1-CIRS

Some highly regular networks, such as 2D grids (alias meshes), admit optimal 1-LIRS [13]. Trees and rings admit optimal 1-CIRS [14].

Interval routing techniques was originally introduced by Santoro and Khatib [15]. This scheme was subsequently generalized by Leeuwen and Tan to more than one interval per edge [13]. The interval routing method has been implemented in the INMOS C104 and RCube routers [16, 17]. A short survey was presented by Leeuwen and Tan [18].

IV. Dynamic IRS Algorithms

A. *IRS for 2D Grids – Optimal 1-LIRS*

With the labeling method described by Leeuwen and Tan, any 2D grid allows an optimal 1-LIRS [13]. Each static node in the grid is labeled with a number from 1 to n. n is the number of static nodes in the grid. The labeling sequence is from top to the bottom rows and from the left to the right in the same row [13]. Each segment connecting two adjacent nodes represents two arcs in different directions. Each arc is labeled with a linear interval.

If the source and target points are just two static nodes, given the labeling information of these two points and follow the interval information on the arcs, a shortest path can be found very efficiently, because no useless nodes would be visited at all. Suppose *pathstatic(N1, N2)* performs this simple static 1-IRS shortest path finding function. N1 is the source node and N2 the target node of this path finding task. They are both static nodes, which respectively contain a member variable named *label*, i.e. N1.label is the node label of N1 and N2.label N2's label of 1-LIRS labeling scheme for this grid. We will use *pathstatic()* directly in the following demonstration of our dynamic 1-IRS algorithms.

In indoor shortest path findings, the source of the path may

not correspond to any static dummy node in the accessibility graph. And in most of cases the source is located on a segment and adjacent to two static nodes directly. And the target is the same as the source. So we call this kind of nodes dynamic nodes. We call the two static nodes directly adjacent to a dynamic node *gating nodes* (GN for short) for a dynamic location. If the source and the target nodes are not on the same segment, the shortest path from or to the source node must pass one of the gating nodes of the source node; and this path must pass one of the gating nodes of the target node as well. We call the static node, from or to which the dynamic node's shortest path pass, *optimal gating node* (OGN for short). Because dynamic nodes like the source or target may be any points on any segments of the network, they can't be given routing information priori as we have done to static dummy nodes. But the information stored for GNs of two dynamic end nodes of the shortest path may help to determine the OGNs very simply. When the OGNs of the source and target nodes are determined, the path between OGNs can be find with the above *pathstatic()* function. Therefore the complete shortest path from the source node S to the target node T can be constructed very efficiently with the IRS information stored for the network.

There are three cases with dynamic nodes involved; they are

(1) S is dynamic, T is static;

(2) S is static, T is dynamic

(3) S and T are both dynamic

Case (1) and (2) can be treated in the same way as in case (3), because any static node can be looked as a dynamic node very close to this static node. If S and T are on the same segment, there is no need to do further shortest path finding computing. So we only discuss the situations, in which S and T are on different segments here.

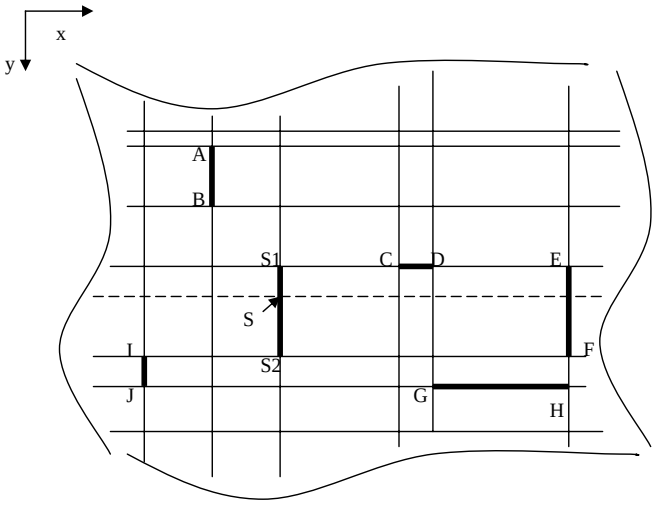


Figure 6 Dynamic S and T in a 2D Grid

If S and T are both dynamic and on different segments, there are two GNs for S and T respectively, the situation is a little more complicated. Let's illustrate it with a piece of a 2D grid in Figure 6. x and y are the column and row location coordinates of the static node in the grid. Let S be a dynamic location on a vertical segment. Let S1 be the upper GN and S2 be the lower GN (S1.y < S2.y and S1.x = S2.x). If T is on a vertical segment, let T1 be the upper GN and T2 be the lower GN (T1.y < T2.y and T1.x = T2.x). If T is on a horizontal segment, let T1 be the left GN and T2 be the right GN (T1.x < T2.x and T1.y = T2.y).

Cut the grid into two parts along a horizontal line passing S. If both of T1 and T2 are in the upper part like AB or CD, S1 is the OGN of S. If T1 and T2 are both in the lower part like GH and IJ, S2 is the OGN of S. If T1 is in the upper part and T2 is in the lower part like EF, then we have to compare |SS1|+|ET| with |SS2|+|TF|. If |SS1|+|ET| is smaller, then the OGN of S is S1 and the OGN of T is E; otherwise, the OGN of S is S2, and the OGN of T is F.

This method can also be used to choose OGNs when S1S2 is horizontal (S1.x < S2.x and S1.y = S2.y). In this case the cutting is vertical, and the similar procedures to find OGNs can be used.

The procedure can be described with the following pseudo-code in Appendix. In *findpath2d(), pathstatic()* runs for only one time, so the worst case time complexity of the program *findpath2d()* is O(r+c). r is the number of rows of the static nodes in the grid; c is the number of the columns of the static nodes in the grid.

B. IRS for Trees – Optimal 1-CIRS

A tree has three important properties:

(1) it has no cycles and

(2) when a tree is cut into two parts at any segment, each part is a new tree

(3) there is only one single path between any two nodes on a tree, no matter if the nodes are dynamic or static. (A single node is a trivial tree.)

Based on these properties, it has been proved that a tree allows optimal 1-CIRS [46]. The 1-CIRS node labeling follows the depth first procedure. Arcs' interval labeling is based on property 2 and 3. For trees we only discuss the situation, in which S and T are both dynamic nodes and on different segment.

We see that we only need to observe the intervals of the arcs between GNs in order to determine the OGNs for S and T. These arcs are S1-S2, S2-S1, T1-T2 and T2-T1. If S1-S2's interval includes T1 and T2, i.e. S2 is the node S must pass in order to reach T. So S2 is the OGN for S; otherwise, S1 is the OGN for S. Similarly, if T1-T2's interval includes S1 and S2, T2 is the OGN for T; otherwise, T1 is the OGN for T. The algorithm is described by the *treedynamic()* function in Appendix. It's worst case time complexity is $O(\Phi)$. Φ is the diameter of the tree.

C. IRS for Rings – Optimal 1-CIRS

Many indoor structures are in circular form, which in calculation is often treated like a ring. A ring consists of a series of straight segments. Each node in the ring is adjacent to two segments; therefore, each node has two arcs starting from itself. The node and arc interval labeling method is based on "hops", which is a concept used only in computer networks [13]. The

distance from one node to another is measured with hops, not with Euclidean length. In our case, we don't concern about the hop numbers between nodes. Euclidean length is our concern. So we have to develop new node and arc interval labeling method based on IRS concepts and Euclidean distance in stead of hop numbers. Each static node in the ring is adjacent to two segments; therefore, each static node has two arcs starting from itself. A ring may appear as in Figure 7. Dashed curves represent all segments not shown in the Figure.

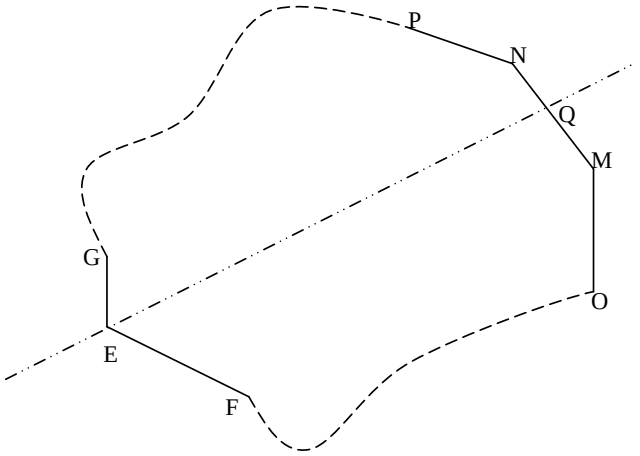


Figure 7 Nodes and Arcs Labeling for Rings

The labeling of nodes is simple. Start from any arbitrary node and choose an arbitrary direction (clockwise or counter-clockwise along the ring).Label all the nodes one by one from 1 to n. n is the total number of the nodes in the ring. As for arc interval labeling is little more complicated because Euclidean distance is used instead of hops.

Choose an arbitrary node E to start labeling for the ring in Figure 7. Draw a dot line passing E, which cuts the ring into two halves. Another intersection point of the dot line and the ring is Q on segment MN. Make sure that the half {EG…PNQ} is equal to {EF…OMQ} in terms of Euclidean length. Then the shortest paths from E to any nodes in {EG…PNQ} all take the clockwise direction; and the shortest paths from E to any nodes in {EF…OMQ} all take the counter-clockwise direction. So the interval of the clockwise arc EG should include the labels of all nodes in {EG…PNQ}; and the interval of the counter-clockwise arc E-F should include the labels of all nodes in {EF…OMQ}. Because the characteristics of the ring topology and our node labeling method, no matter how to choose E, the labels of all nodes on one of its two halves of the ring can be bracketed with only one circular interval. That means, for each of its two arcs to different directions, only a circular interval is needed. Accordingly, 1-LIRS can be applied not only to abstract rings in computer networks, but also to physical Euclidean rings.

When a ring is labeled in above way, the following dynamic algorithm can be applied. As in 2D grids and trees, we only discussed the situation, in which S and T are both dynamic nodes and on different segements.

In the ring in Figure 8, S1 is the counter-clockwise GN of S; and S2 is the clockwise GN of S. T is the clockwise GN of T; and T2 is the counter-clockwise GN of T. For our convenience to illustrate the algorithm, draw a line passing S1 and T1; and draw another line passing through S2 and T2. If the interval of arc S1-S2 includes all of S2, T2 and T1, it means the length of the ring below the line S1-T1 is shorter than that above the line. So the clockwise path from S1 to T1 is shorter than the counter-clockwise path. In this case, S to T's shortest path should be clockwise; and therefore, S2 and T2 are OGNs for S and T respectively.

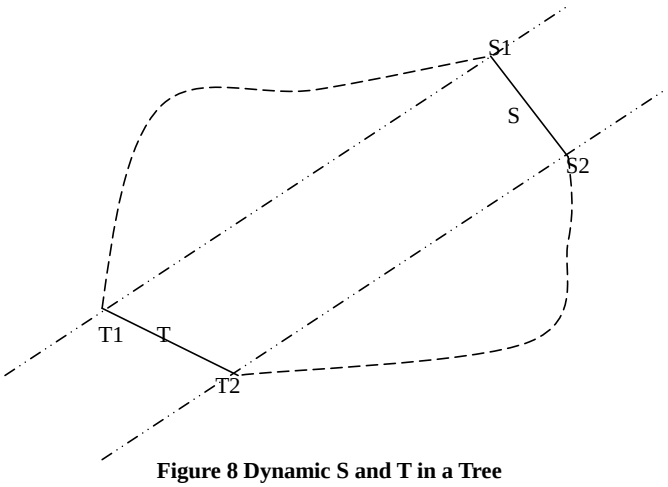


Figure 8 Dynamic S and T in a Tree

If the interval of arc S2-S1 includes all of S1, T1 and T2, it means the length of the ring above the line S2-T2 is shorter than that below the line. So the counter-clockwise path from S2 to T2 is shorter than the clockwise path. In this case, S to T's shortest path should be counter-clockwise; and therefore, S1 and T1 are OGNs for S and T respectively.

If none of the intervals of arc S1-S2 or S2-S1 includes all of S1, T1 and T2, the Euclidean length of the path from S to T in counter-clockwise direction must be compared with that in clockwise direction. The smaller one is the direction of the shortest path from S to T. The dynamic program can be described with the pseudo code of the function *ringdynamic()* in Appendix. The time complexity of this program is O(n). n is the number of static nodes in the ring.

V. Dynamic Shortest Path Algorithm for Complex Networks Figures and Tables

A. Element Classes and Non-Classified Segments

With the algorithms developed for the 1-IRS structures including grids, trees and rings, we can apply the IRS techniques in static computer networks to dynamic Euclidean networks. We have analyzed common inside-building structures and find that most of structures of single buildings can be modeled into the basic structures like grids, trees and rings. We call them Element Classes (EC for short). And one

single node can be treated as a trivial EC here.

Sometimes the path finding source S and T may exists in different buildings, and the shortest path may pass through arcs and nodes belonging to multiple ECs. In emergency evacuation, this does not happen very often. Usually the users of our system are supposed to be rescue team members. They often have been in the emergency building before they are equipped and issue shortest path finding command to the system. But we still need to consider more complicated structures than ECs.

Some buildings have more complicated topologies than any 1-IRC EC. In this case we can treat this building as a collection of ECs connected by some special segments. These special segments don't belong to any EC; therefore they are called *Non-Classified Segments* (NCS for short). All segments included in each EC are called *Classified Segments* (CS for short).

A more complicated network structure is shown in Figure 9. EC0 is a trivial EC. Five ECs are connected by a collection of NCS as shown. CSs are not shown because we don't need them for the global Dijkstra's algorithm.

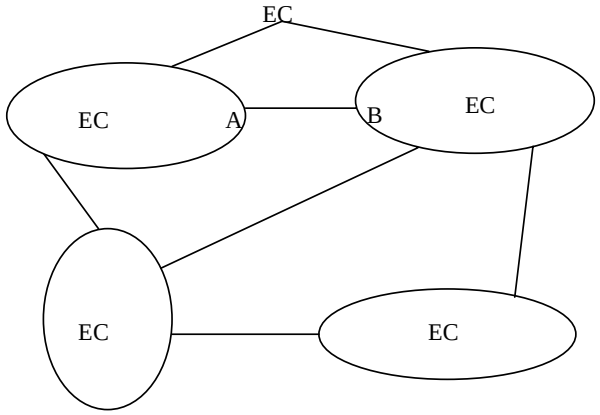


Figure 9 A Complex Structure

Any node of an EC, which is one end of any NCS, is called *Boundary Node* (BN) of this EC or the boundary node to the EC at the other end of the NCS. For Example, A is a BN of EC1, and a BN to EC2. B is a BN of EC2, and a BN to EC1.

B. General Procedure Description

When an Element Class ECa is input to the global Dijkstra's algorithm, it may take two forms. Because the source location S and target location T are not in the same location, when S is in ECa, the global Dijkstra's algorithm only cares about the length of the path from S to each BN of ECa. So we can draw ECa in a star shape as shown in Figure 10.

All spokes of the star, SA, SB, SC, and SD are virtual segments. They are constructed with the fast dynamic IRS algorithms illustrated before. And the length of each virtual segment is the length of the real shortest path between the two end nodes of the virtual segment. Similarly, if T but not S is in ECa, the global Dijkstra's algorithm only cares about the length of the path from each BN of ECa to T. So in this case, ECa still takes the star form to join the global Dijkstra's algorithm.

Another form ECa may take to join the global Dijkstra's algorithm is an n-partite. n is the number of ECs directly connected to ECa by some NCS. This happens when neither S nor T is located in EC. In Figure 11, ECa is in a bi-partite form, when A and D are BNs to the same EC, and B and C are BNs to another EC. AB, AC, DB and DC are all virtual segments; and the length of each virtual segment is the length of the real shortest path between the two end nodes of the virtual segment.

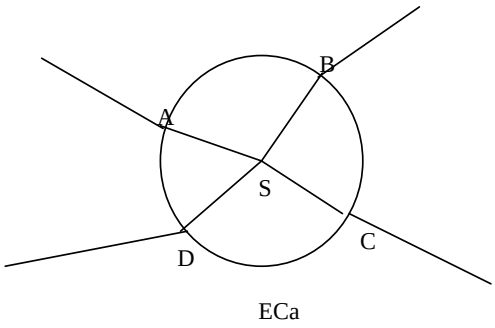


Figure 10 Star Form of an EC

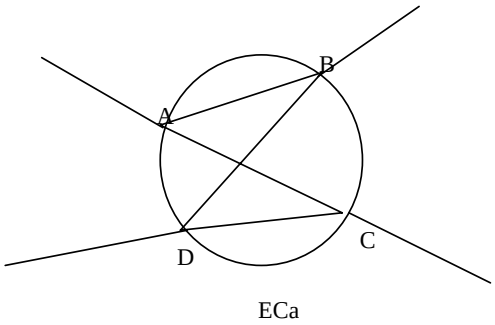


Figure 11 2-Partite Form of an EC

As S and T are assigned in real time, the star form of an EC must be constructed in real time. The n-partite form of an EC can be constructed in advance in order to save run-time computing resources. Before S and T are assigned, EC of the network constructed in advance all take the form of n-partite. In this stage, the global network consists of n-partite form ECs connected with NCSs. We call the graph of the global network in this stage *Primary Complete Connectivity Graph* (PCCG).

When S and T are assigned two different ECs, EC1 and EC2, these two ECs are transferred into star form through real time path calculations. We call the graph of the global network in this stage *Complete Connectivity Graph* (CCG). With CCG, Dijstra's algorithm can be applied and a path p from S to T can be found. p is a sequence of arcs. Some arcs represent virtual segments, so each of these virtual arcs should be converted to real sub-path. Then the virtual path p is transferred to be the real path P. The general procedure of this method is described in the flow chart in Figure 12.

C. Performance Considerations

To evaluate the performance of the algorithm, we compare its computing process with that of pure Dijkstra's algorithm

applied to the whole real network.

In our algorithm, IRS dynamic algorithm is very fast. So Dijkstra's algorithm part is supposed to be the most time consuming part. So comparing the performance of this part to the pure Dijkstra's algorithm applied to the whole real network is meaningful for us. An algorithm's computing time is roughly proportional to the input size, which for a network algorithm is segment and node numbers.

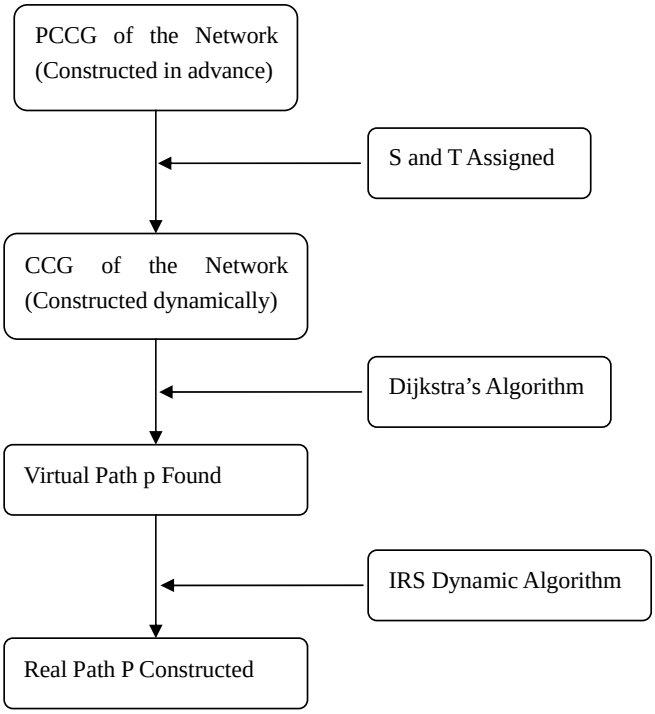


Figure 12 The Flowchart for Multi-EC Algorithm

Suppose we have an network with x ECs; they are ECi (i=1,2…x). In ECi, the node number is Ni and the segment number is Ai. The number of NCSs is a constant K. If we input the whole real network to pure Dijkstra's algorithm, the total node number N and total segment number is A; they are:

$$N = \sum_{i=1}^{x} Ni$$

$$A = K + \sum_{i=1}^{x} Ai$$

In CCG of a network, most of ECs take n-partite form except the ECs where S or T stays. So node and segment numbers of PCCG can represent those of CCG's. For ECi, CNi is the node number of its n-partite form and CA_i is the segment number of its n-partite form. When we input the CCG to Dijkstra's algorithm, the total node number is CN and total segment number is CA; they are:

$$CN = \sum_{i=1}^{x} CNi$$

$$CA = K + \sum_{i=1}^{x} CAi$$

So to allow our algorithm to have better performance, at least it is required that CN<=N and/or CA<=A. If it is forced that $CN_i <= N_i$ and $CA_i <= A_i$ (it is called boundary requirements) for each i, the requirements is met. If an EC's CCG doesn't meet the boundary requirements, all real nodes and segments in the EC should join the Dijkstra's algorithm directly without any virtual form like star or n-partite. That means, in this case, this EC should be ungrouped and is not treated as an EC in the following Dijkstra's algorithm.

When all ECs' CCG meet the boundary requirements, it can not make it assure that the algorithm with ECs involved performed better than pure Dijkstra's algorithm. Because in order to implement EC involved algorithm, the data structure needed is unavoidably bigger than that of pure Dijkstra's algorithm. The space complexity of the EC involved algorithm is higher than that of pure Dijkstra's algorithm. Data structure size affects the speed of data transiting between computing resources on the computer. When the data structure is too big, it will significantly affect a program's running speed.

The performance of the algorithm with EC involved will be much better than pure Dijkstra's algorithm when the nodes and/or segments in the network can be significantly reduced by constructing CCG with virtual forms of the ECs. If it is not the case, its performance should be tested for arbitrary network.

VI. Conclusions and Future Research

In this research, common inside-building topologies are discussed first with their simplified accessibility graphs. According to their characteristics, IRS techniques in static computer networks are extended to the dynamic Euclidean networks. The dynamic algorithms for some frequent appearing structures, such as 2D grids, 3D grids, trees and rings are discussed and described by pseudo codes in Appendix. These codes can be converted to other object-oriented language codes.

Complicated structures can be treated as a collection of connected ECs. Each EC can be a grid, a tree or a ring. In this view, dynamic 1-IRS and Dijkstra's shortest path algorithm are integrated into a multi-EC algorithm. Its performance is compared with pure Dijkstra's algorithm, which is the classic algorithm in calculating Euclidean shortest paths.

Computer networks and buildings internal structures often take similar high regular structures. The reason may be that they are both artificial products of human beings, who prefer orderliness, standardization and easiness of republication especially for engineers. Frequent appearances of high regular topologies in network designs of a variety of engineering fields may reflect this preference broadly existing in engineering area. Due to the characteristics of computer networks, a variety of routing techniques have been applied very early. This research is attempting to explore the possibility and potential of the usage of certain routing techniques on inside building topologies with the presence of location trackers with affordable accuracy level.

The implementation of IRS on inside building topologies

may provide other functions than finding the shortest path. With the small routing table stored for all dummy nodes, routing information can be given for all directions from any point in real-time in the network. This implies that some Virtual Reality tools, like automatic road guide or real-time route suggest, can be implemented with the help of IRS.

In this article, only 1-IRS techniques are discussed for indoor networks. Other routing techniques like k-IRS may also be explored in future research.

APPENDIX

A. 2D grid code:

Class Gridnode {Arc [] arclist; int nodelb; int x; int y}

/* Arclist is an array of the arcs starting from the node. Nodelb is the 1-LIRS label for the node. x and y are the coordinates of the node in the grid*/

Class Gridarc {Gridnode stnode, ennode; float arclength; int sint, eint;}

/* stnode and ennode are the start and end nodes of the arc. arclength is the Euclidean length of the arc; sint and eint are respectively the first and second number of the linear interval label. */

```
Program pathstatic (node s, node t)
{ Gridarc arc; Arc [] result; Gridnode ndtemp; ndtemp=s;
While (ndtemp.nodelb <> .nodelbt) do
{for each arc in ndtemp.arclist
      if (t.nodelb >= arc.sint and t.nodelb <= arc.eint) then
      {result.add(arc); ndtemp=arc.ennode; break;} }
return result [];} // result[] is the arc array of the final path

Program findpath2d (S, T, G)
/*S, T are the source and G is the network consisting of
nodes and arcs. S and T are on different segments. S1 is the
upper or left GN of S, S2 is the lower or right GN of S.T1 is the
upper or left GN of T, T2 is the lower or right GN of T.*/

{ Arc [] result; Gridnode ndtemp;
// S and T are on different horizontal segments
if (S1.y = S2.y and T1.y = T2.y and T2.x<=S1.x) then
result=pathstatic(S1,T2); endif;
if (S1.y = S2.y and T1.y = T2.y and T1.x>=S2.x) then
result=pathstatic(S2,T1); endif;
   if (S1.y = S2.y and T1.y = T2.y and T1.x=S1.x and
T1.y<>S1.y) then
{if ((|SS1|+|TT1|) <= (|SS2|+|TT2|)) then
result=pathstatic(S1, T1);
else result= pathstatic(S2, T2); }
// S and T are on different vertical segments
if (S1.x = S2.x and T1.x = T2.x and T2.y<=S1.y) then
result=pathstatic(S1,T2); endif;
if (S1.x = S2.x and T1.x = T2.x and T1.y>=S2.y) then
result=pathstatic(S2,T1); endif;
   if (S1.x = S2.x and T1.x = T2.x and T1.y=S1.y and
T1.x<>S1.x) then
{if ((|SS1|+|TT1|) <= (|SS2|+|TT2|)) then
result=pathstatic(S1, T1);
else result= pathstatic(S2, T2);}
// S on vertical segment and T on horizontal segment
if (S1.x = S2.x and T1.y = T2.y and T1.y<=S1.y and
T2.x<=S1.x) then result=pathstatic(S1,T2);
if (S1.x = S2.x and T1.y = T2.y and T1.y<=S1.y and
T1.x>=S1.x) then result=pathstatic(S1,T1);
if (S1.x = S2.x and T1.y = T2.y and T1.y>=S1.y and
T2.x<=S1.x) then result=pathstatic(S2,T2);
if (S1.x = S2.x and T1.y = T2.y and T1.y<=S1.y and
T1.x>=S1.x) then result=pathstatic(S1,T1);
// S on horizontal segment and T on vertical segment
if (S1.y = S2.y and T1.x = T2.x and T1.x<=S1.x and
T2.y<=S1.y) then result=pathstatic(S1,T2);
if (S1.y = S2.y and T1.x = T2.x and T1.x<=S1.x and
T1.y>=S1.y) then result=pathstatic(S1,T1);
if (S1.y = S2.y and T1.x = T2.x and T1.x>=S1.x and
T2.y<=S1.y) then result=pathstatic(S2,T2);
if (S1.y = S2.y and T1.x = T2.x and T1.x>=S1.x and
T1.y>=S1.y) then result=pathstatic(S1,T1);
return result[];} // end of the program findpath2d()
```

B. Tree code:

```
Program treedynamic(S,T,G)
// S, T are dynamic nodes on two different arcs. S1, S2 are
GNs of S. T1, T2 are GNs of T.
{  Arc [] result;
{if (interval of arc S1-S2 includes labels of T1 and T2)
then (S2 is the OGN) else (S1 is the OGN); //Find OGN of S
if (interval of arc T1-T2 includes labels of S1 and S2)
then (T2 is the OGN) else (T1 is the OGN); //Find OGN of S
result=pathstatic(OGN of S, OGN of T);
return result[];} // end of the program treedynamic()
```

C. Ring code:

```
Program ringdynamic (S, T, G)
/* S, T are dynamic nodes on two different arcs. S2is the
clockwise GN of S, S1 is the counterclockwise GN of S. T1is the
clockwise GN of T, T2 is the counterclockwise GN of T.*/
{Arc [] result;
if (the interval of arc S1-S2 includes labels of T1 and T2)
then
{ result=pathstatic(S2,T2);return result[]; }
if (the interval of arc S2-S1 includes labels of T1 and T2)
then
{ result=pathstatic(S1,T1);return result[]; }
if (SS1+S1T1+T1T<=SS2+S2T2+T2T) then
{ result=pathstatic(S1,T1);return result[];}
else { result=pathstatic(S2,T2);return result[]; }
} //end of the program ringdynamic()
```

REFERENCES

[1] H. Beadle, B. Harper, G. Maguire Jr., and J. Judge. Location aware mobile computing. In *Proc. ICT '97 (IEEE/IEE Int. Conf. on Telecomm.)*, Melbourne, Australia, 1997.

[2] Dijkstra EW. A note on two problems in connection with graphs, *Numerische Mathematic 1959;*1:269-71.
[3] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan. Triangulating a simple polygon. Inform. *Process. Lett.* 7: 175-179, 1978.
[4] D. T. Lee and F. P. Preparata. Location of a point in a planar subdivision and its applications. *SIAM J. Comput.*, 6: 594-606, 1977.
[5] D. Avis and G. T. Toussaint. An efficient algorithm for decomposing a polygon into star-shaped polygons. *Pattern Recogn.*, 13:395-398. 1981.
[6] B. Chazelle. A theorem on polygon cutting with applications. In *Proc. 23rd Annu. IEEE Sympos. Found. Comput. Sci*., pages 339-349, 1982.
[7] B. Chazelle. Approximation and Decomposition of Shapes. In *Algorithmic and Geometric Aspects of Robotics*, Lawrence Erlbaum Associaties, Hillsidale, NJ, 1987., page 145-185.
[8] N. J. Nilsson. A mobile automation: an application of artificial intelligence techniques. *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, Washington D. C., 509-520, 1969.
[9] F. Aurenhammer. Voronoi diagrams: properties, algorithms and applications. *SIAM J. Comput..*, 16:78-96, 1987.
[10] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Chichester, U.K., 1992.
[11] T. Hollerrer, D. Hallaway, N. Tinna and S. Feiner. Steps toward accommodating variable position tracking accuracy in a mobile augmented reality system. *AIMS'01: Second Int. Workshop on Artificial Intelligent in Mobile Systems*, Seatle, WA, Aug. 4, 2001, pp.31-37.
[12] C. Gavoille. A survey on interval routing. *Theoret. Comput. Sci*. 245 (2000) 217-253.
[13] J. Van Leeuwen, R.B. Tan. Interval routing. *The Computer Journal*, 30 (1987) 298-307.
[14] N. Santoro, R. Khatib. Labeling and implicit routing in networks. *The Computer Journal*, 28 (1985) 5-8.
[15] N. Santoro, R. Khatib. Routing without routing tables. *Technical Report SCS-TR-6 School of Computer Science*, Carleton University, Ottawa, 1982.
[16] D. May, P. Thompson. *Transputers and Routers: Components for concurrent machines*, INMOS Ltd., 1990.
[17] B. Zerrouk, V. Reibaldi, F. Potter, A. Greiner, D. Anne. RCube, a gigabit serial links low latency adaptive router, in the *Records of the IEEE Hot Interconnects IV*, Palo Alto CA, U.S.A, August 1996.
[18] J. van Leeuwen, R.B. Tan. Compact routing methods: a survey, in: P. Flocchini, B. Mans, N. Santoro (Eds.), *1st Internat. Coll. on Structural Information & Communication Complexity (SIROCCO)*, Carleton University Press, May 1994, pp. 99-110.