Problem decomposition for evacuation simulation using network flow

Seán Óg Murphy[12], Kenneth N. Brown[2], Cormac Sreenan[1]

1. Mobile and Internet Systems Laboratory, 2. Cork Constraint Computation Centre,
Department of Computer Science, University College Cork, Ireland
jmm3@student.cs.ucc.ie, k.brown@cs.ucc.ie, cjs@cs.ucc.ie

Abstract. Simulation of building evacuations can be a powerful tool for predicting evacuation outcomes, but for this prediction to be useful it must be produced in a timely manner. The building evacuation outcomes are dependent on the movement decisions of the occupants, but simulating all possible combinations of occupant decisions is infeasible. Our contribution is a novel technique using building structure knowledge in the form of a Network Flow Graph to determine where and when occupants might interact with one another. We decompose the problem into non-interacting groups, to be simulated separately, which leads to a significant simulation workload reduction.

Keywords: evacuation monitoring, multi-agent simulation, distributed simulation, network flow graph, problem decomposition

1 Introduction

Simulation of building evacuation during emergencies is a powerful tool for evaluating building safety and prediction of evacuation outcomes[1]. Simulating the movement of evacuating pedestrians can provide useful information to building designers and to fire safety officials, including evacuation time estimates and locations at risk of traffic congestion or evacuee injury[1-7]. We make use of a 2D vector-space pedestrian simulator to simulate the individual movements of pedestrians in the building. This simulator provides a realistic model of pedestrian movement but is computationally intensive compared to simpler models such as Cellular Automata [2,4,10] or Graph-based simulations [5]. The usefulness of any prediction information is contingent on both the timeliness and the accuracy of the simulation that generated it. Since a real evacuation involves autonomous human actors and uncertain hazard spread, to ensure that the simulations cover likely scenarios we must run many different scenarios. The more we can reduce the simulation time, the more scenarios we can consider, and the more useful the results will be. In this paper, we investigate the problem of how to cover a wide range of scenarios without an excessive increase in computation time.

We assume that clusters of individuals in an evacuation will behave as a group, and once they have formed a group all members will follow the same route out of the building. We reduce the amount of simulation by limiting our attention to the most likely routes for each initial group. However, different groups may meet during the evacuation and cause congestion, and may form into larger groups to continue the evacuation. Since congestion slows down the evacuation, and puts the participants at risk, we must investigate the interaction of the different groups. Even for small numbers of likely routes, this group interaction quickly produces an excessively large simulation space. We reduce the complexity of the problem by quickly generating possible group interactions using Network-Flow Graph analysis, identifying groups and routes which do not interact, distributing the simulation of those local scenarios to different processors, and then combining the results to provide global evacuation analyses. For those groups and routes which do interact, we simulate them together in a single process. Decomposing the problem in this manner and computing it using parallel computation resources allows many more possible futures to be simulated in a time-frame that allows for predictive feedback to fire-safety personnel during the emergency. Further, we handle the uncertainty with an agile simulation strategy, by responding to new evacuation data and quickly decomposing the problem space to drive a new cycle of simulation.

We implemented this problem decomposition methodology and evaluated its performance in a typical building topology, with a mix of population densities and hazard positions. In our experiments our methodology identified a manageably small number future states that need be simulated. Using this problem decomposition strategy in our experiments we successfully filtered out all but a few dozen future states of the evacuation, significantly reducing the scale of the simulation task.

This work was funded by HEA PRTLI IV as part of the Nembes project.

2 Overview and Assumptions

Our approach is to perform a series of filtration operations to produce the parameters for a set of several simulations. As there are limitations on time and computational resources, we make some simplifying assumptions as to occupant behaviour and analyze the building structure to identify where occupants are likely to meet each other in the building. While occupants during emergencies are inherently unpredictable, we believe it is pragmatic to disregard the possibility of particularly unusual behaviour (such as occupants heading away from exits or splitting up groups) until we receive information that this unusual behaviour is actually occuring in the building via sensor data or firefighter reports.

By analyzing the building structure (in the form of a network flow graph), we can determine which occupants in the building are *orthogonal* in time and space *before* we go to the trouble of simulating them together at once. Orthogonal groups of occupants can be simulated in separate simulation instances as their members are not likely to interact with each other. For instance, occupants on opposite ends of a building are not likely to meet each other when evacuating, and can be safely simulated separately without concern for modeling interactions between the groups.

To determine which occupants interact with each other, we first assume that occupants that begin the evacuation near to each other will form a group. We assume this group will stick together and may follow one of several routes out of the building. We also assume that when separate groups meet each other during the evacuation, they merge together and continue using one of the constituent group's routes out. By grouping occupants in this manner we reduce the problem complexity significantly, as we focus on permutations of group decisions rather than of individual occupant decisions.

2.1 EvacSim Evacuation Simulator

EvacSim is a Java-based building evacuation simulation tool (Figure 1). This tool models a building in 2.5 dimensions (2D floors stacked on top of each-other) and algorithmically extracts a topology graph from this structure by packing traversable spaces with rectangles representing rooms and corridors, and then connecting adjacent rectangles with graph edges. EvacSim represents building occupants as individual agents with dynamic movement behaviour (flocking[16], group formation, obstacle avoidance) and path planning capability. Occupant Agents move through in the building according to their plan, can respond to visible signposting or hazards and form groups that move and plan travel together. Occupant Agents travel in 2-dimensional space, directing their travel through manipulation of a 2D vector which they use to steer towards their goal while avoiding obstacles.

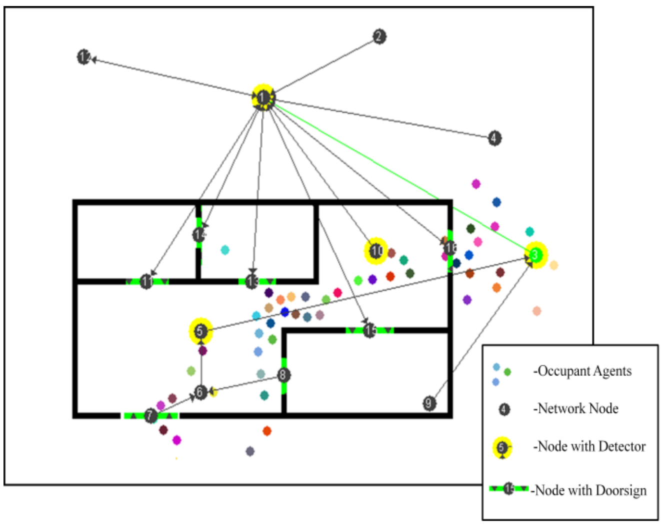

Figure 1. EvacSim Evacuation Simulator screenshot

EvacSim also models individual sensor and actuation nodes as simulated sensor network components. Simulated sensors can determine the occupancy of spaces in the building and identify the presence of a hazard such as a fire or smoke.

Simulated occupants and sensors provide occupancy data as the initial state from which future states are predicted. This sensor data also provides the evacuation route planner (Section 2.3) with the occupancy information which drives the dynamic occupant routing algorithm.

Each occupant agent in the simulation needs to periodically check for collisions and interactions with other agents in the world. As a consequence, the complexity of a naive simulation is $O(n^2)$ as each occupant checks each other occupant for collisions. EvacSim makes use of a Cell List mechanism [2,21] which limits these collision and interaction checks to locally surrounding occupant agents. Using this mechanism, the computational complexity relates to the *density* of occupants (i.e. the number of occupants in each Cell) rather than the overall number of occupant agents in the world.

2.2 Network Flow Graph

Analysis of the building structure produces a building Topology Graph. This graph represents the spaces in the building (rooms, corridors etc) as graph nodes and connects nodes together with edges if it is possible to traverse between them. Edge weights correspond to the walking distance between nodes. Routes from a group's starting position to an exit are considered as a sequence of nodes to visit on this graph. The sum of edge weights between each node gives the traversal time for an unimpeded individual on that route.

This Topology Graph is extended as a Network Flow Graph[13] by placing a flow capacity constraint on each edge. This constraint dictates the number of occupants that can traverse the edge simultaneously, i.e. how many people can walk abreast without impeding each other. Network Flow Graphs are often used to model road traffic in emergencies[14,15] and as a basis for dynamic evacuation planning[10,11,12]. In this research, we use the flow capacity constraint to determine how long it takes for an entire group of occupants to completely traverse the edge, analogous to calculating the amount of time it takes to transfer a given amount of data across a network link of limited bandwidth

2.3 Route Allocation

Occupant groups have a number of options as to how they may proceed out of the building; in fact if we allow looping there are an infinite number of routes to exits. To reduce the problem space, we again make some assumptions as to likely occupant behaviour. We assume there are a limited number of routes the occupants are likely to take; these routes may be observed from day-to-day sensor readings of the building, drawn from k-shortest paths or based on evacuation guidance present in the building.

In our experiments we use a hazard-aware safe evacuation planner to produce routes which are communicated to the occupants (e.g. via mobile device or dynamic signposts) [8- 10]. This evacuation planner accounts for the location of hazards and produces evacuation routes that minimize evacuation time and exposure to the hazard. Alternative route generation schemes could be used, such as dynamic flow-based planners [10-12] or exploiting day-to-day sensor data to discover common exit routes.

2.4 Complexity and orthogonality

The number of potential futures simulated is contingent on the number of possible routes the occupant groups could take, and the amount of interaction between groups. As we assume groups that meet in the building could proceed according to either constituent group's route, there is the potential for a great deal of branching future states. In this research we perform initial analysis of the Network Flow Graph and Routes in order to filter out combinations which are unlikely to occur, and only simulate the remaining combinations. By exploiting the network flow graph to determine the times at which crowds arrive at nodes, as well as the duration they spend at the node, we can identify when and where crowds might meet each other. Crowds which don't meet at any time in the graph are considered orthogonal to each other and do not need to be simulated together (Figure 2). This characteristic is crucial to reducing the problem space, as we do not need to simulate the product of the permutations of multiple orthogonal crowds.

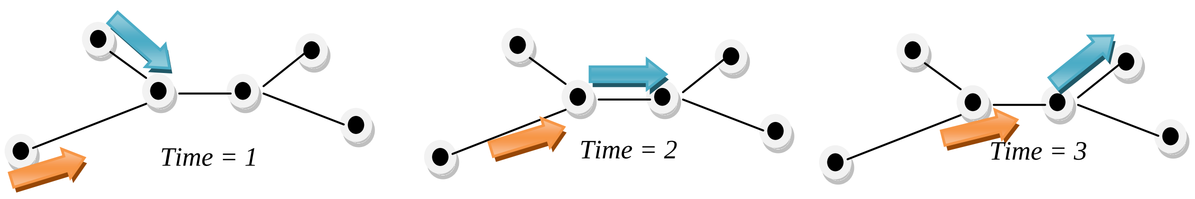

Figure 2. Two crowds occupying the same space but at different times are orthogonal to one another

In the case that two groups meet each other in the building, we need to account for the branching possibilities that proceed from the interaction. Where two crowds meet, they interact, share route information and lead to greater queues and delays on the two groups. In our approach we merge the groups together and investigate the branching futures where they proceed along any one of the original groups' routes. For instance 3 groups of 10 meeting leads to investigation of 1 group of 30, and the 3 possible routes it could take (as drawn from the original 3 groups).

3 Simulation Framework

3.1 Network Flow Graph

Building structure analysis produces a Network Flow Graph which represents the traversal characteristics of the building. In this graph, Nodes represent spaces such as rooms or corridors, and Edges represent the adjacency of those spaces. A key feature of Network Flow graphs is that edges feature not only a weight representing distance, but also have a capacity value which represents how many occupants can traverse the edge in a given time period. Intuitively, the capacity value represents the *bandwidth* of the edge and the distance represents the *traversal time*.

Capacity and Distance values can be set based on simple formulae (e.g. straight line distance for the Distance value, and a Capacity estimate based on the size of the area the edge passes through), but in this work we determine these values by simulation of crowds passing over the edge similar to the macro-micro coupling work of Kneidl et al[3].

3.2 Evacuation Route Generation

EvacPlan is a dynamic crowd evacuation planner[11] which takes as input a network flow graph with set initial occupancy, and location of hazards and exits. EvacPlan computes an optimum assignment of evacuation routes to individual occupants, minimizing total evacuation time and individual exposure to the hazard. EvacPlan produces a route in the form of a sequence of Nodes in the Network Flow graph, occupants are communicated these routes and follow them to escape the building.

In this research we assume that it is not possible to provide individualized routes to each occupant separately; instead we communicate the full set of applicable routes to *all* the occupants at a node and assume that the group will stick together and follow just one of these routes.

4 Discovering Orthogonal Future States

4.1 Crowd-Route generation

Each Node in the Network Flow Graph has an initial occupancy value, reflecting the presence of occupants as reported by the sensor network. This occupancy information initializes a "Crowd" object for each occupied Node in the network. EvacPlan provides sets of likely routes which could be followed by occupants at nodes, each combination of a route with a crowd generates a CrowdRoute object. For example, 12 occupants at Node A have 3 routes assigned to their members. As we assume that the Crowd doesn't split up, this results in the generation of 3 CrowdRoute objects of size 12, accounting for the 3 routes that the Crowd may take in exiting the building.

4.2 Time expansion of Network Flow Graph (TimeGraph)

In this scheme, we filter out impossible future states of the evacuation within a given time period. To achieve this we divide the time period into individual timepoints (e.g. at 1-second intervals) and each Node has an array of occupancy records, which will be used to represent occupancy of the Node at each timepoint. This allows us to recognise that two groups passing through the same node at different times are orthogonal in time and hence don't collide. From now on, we refer to this time-expanded network flow graph as a "TimeGraph".

4.3 First- and Last-Arrival calculation

We determine that two CrowdRoutes interact if they ever occupy the same space at the same time. To determine if this is the case, we work the CrowdRoute through its route on the graph, determining the earliest and latest arrival time at each node during its journey and noting the presence of the CrowdRoute at each Node for any timepoints between these two values. The earliest arrival of a member of the group at each node is given by the *traversal* time over the connecting edge. The latest arrival time is dictated by the size of the Crowd and the bandwidth of the edge, computed as (EarliestArrivalTime + size/capacity). As such, a narrow corridor may take longer for a large crowd to traverse than a wide open space, due to queuing. This queuing leads to members of the CrowdRoute arriving at the next node over the course of an extended period and they are more likely to collide with other CrowdRoutes as a result. Note that this computation is carried out on the TimeGraph before we execute any expensive multi-agent simulation.

First and Last Arrival calculation

```
T = 0; TimeGraph TG; maxdur = 30;

for each CrowdRoute CR {

//starting from the CR's current position at the start of route
     Node currNode = CR.Route.get(0);

//Work CR through the graph, marking presence at each Node
     for each other Node N in CR.Route {
            Edge E = TG.GetEdge(currNode,N);

//Calculate the first and last arrival times for the next Node
            firstArrival += E.traversalTime;
            throughput = CR.size/E.capacity;
            lastArrival = firstArrival + throughput;


            for (int i = time, i < throughput; i++) {
//Mark CR as present on current node until all occupants departed
                   currNode.addAtTime(i,CR);
                   if (i+firstArrival < maxdur)
//Mark CR as present on N from first to last arrival, up to maxdur
                          N.addAtTime(firstArrival+i,CR);
            }
//Move the clock ahead to the time of arrival at next node
            T = T+traversal;
            currNode = N;
     }
}
```

4.4 CrowdRoute collision discovery and merging

Having worked each CrowdRoute through the graph, we can then examine each time-point on each Node to determine if and where CrowdRoutes simultaneously occupy a node. In cases where CrowdRoutes interact, we merge the CrowdRoutes together, and repeat the calculations in 4.2 (unless the CrowdRoutes involved contain the same Crowds as each other, as a Crowd cannot collide with itself).

In the case of a merge, all the colliding CrowdRoute Crowds are combined together into a combined Crowd object. This Crowd has a size equal to the sum of the constituent crowd objects. Each original CrowdRoute's Route is used in the creation of a new set of CrowdRoute objects which are inserted into the TimeGraph nodes at the timepoint of the earliest first-arrival among the colliding CrowdRoutes at that node. The merged CrowdRoute contains a record of the constituent Crowd objects and the routes they were following before the merge occured.

Example:
CrowdRoute A-a, B-b and C-c collide at Node N.
This produces 3 new CrowdRoute objects (one for each route they proceed on):
CrowdRouteA-a,B-b,C-c(a)
CrowdRouteA-a,B-b,C-c(b)
CrowdRouteA-a,B-b,C-c(c)

The new CrowdRoutes are inserted into the graph on Node N, at the earliest timepoint any of the constituent CrowdRoutes was present at that node and the first-last arrival operations are performed for this new merged CrowdRoutes. This may in turn lead to the discovery of more collisions and generate further merged groups; this continues until no new collisions are discovered (the maxduration value provides a natural termination point for this cycle, indicating how far into the future we intend to simulate).

Collision Detection and Merge Operation

```
for each Node N {
    for each timepoint T in N {
//if there are multiple crowdroutes present
      if (N.T.contents.size > 1)
//and if we haven't merged on a previous iteration
        if (!exists(CrowdRoute(t.contents))) {
//find the earliest arrival of any colliding crowd
            int arrival = earliestArrival(N,t.contents)
//get the set of all member routes
            Set Routes = getAllRoutes(t.contents)
//remove any duplicate routes
            Routes.enforceUnique()
//remove any nodes in the routes before the current node
            Routes.startFrom(N)
            for each route R in Routes {
//make a merged CrowdRoute for each possible route
                CrowdRoute merged = new CrowdRoute(t.contents);
                merged.setRoute();
//insert the merged route into the graph
                N.addAtTime(arrival,merged);
            }
        }
    }
}
```

4.5 EvacSim instance generation

The Collection of all CrowdRoute objects generated during the collision detection and merging phases represents the possible ways occupants in the building could meet and proceed together. Each compound CrowdRoute generated features the constituent Crowds and the routes that they took in order to produce the compound CrowdRoute. CrowdRoutes generated from the merging of other compound CrowdRoutes also hold the information describing which route they followed after each previous merge. For each CrowdRoute, we can create a seperate EvacSim instance. Each instance contains occupant agents representing the members of the Crowd objects present in the CrowdRoute. Each occupant agent is given a sequence of routes to follow; initially following their original route (e.g. occupants from CrowdB follow route b) until they meet occupants from other Crowds in the world. When they meet, they proceed according to the record in the compound CrowdRoute object.

For example. following the collision and merge operations, we may be left with 7 CrowdRoutes:

CrowdRouteA-a CrowdRouteB-b CrowdRouteC-c	*The original CrowdRoutes*
CrowdRouteA-a B-b(a) CrowdRouteA-a,B-b(b)	*A meets B*
CrowdRoute[A-a,B-b(b)],C-c(b)	*A+B on B's route meets C, uses b*
CrowdRoute[A-a,B-b(b)],C-c(c)	*A+B on B's route meets C , uses c*

This leads to the generation of 7 seperate EvacSim instances, one for each original CrowdRoute object and one for each merge operation that occurs. This accounts for groups merging, and allows for the possibility that two groups might fail to meet eachother.

5 Experiments

We evaluate our problem decomposition method by executing it on realistic evacuation scenarios. We used a realistic building model and occupant population, varying the position of the emergency hazard (fire) to provide a number of different evacuation scenarios within this building model.

5.1 Experiment Description

These experiments were performed using a model based on the 3rd floor of the Kane Science Building at University College Cork with initial populations of 85 and 340 occupants , spread among various office and lecture spaces, constituting 19 initial occupant groups. This population features a mix of large and small groups, and the floor layout produces a variety of routes to exits and varying flow capacities due to the variety of corridor widths and room sizes (Fig 3 (i)). This floor features 6 exits and in this experiment we produce orthogonal simulation sets based on a 40-second prediction period. We repeated this experiments adjusting the population density and varying the location of the hazard to produce a variety of different initial states and outcomes.

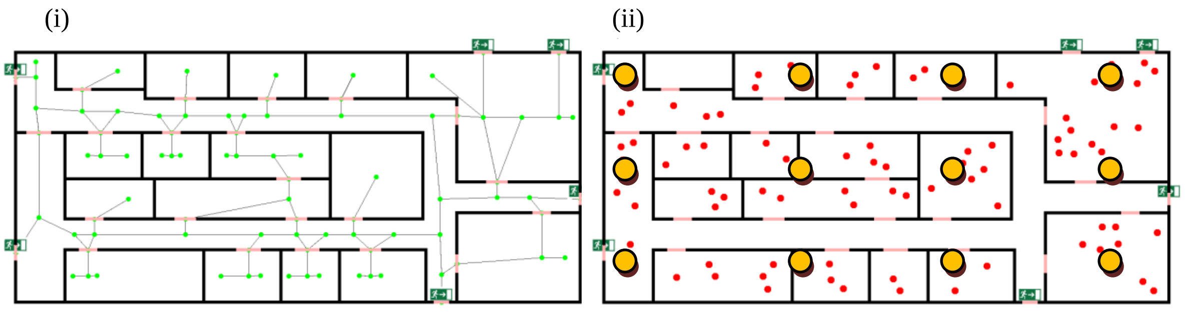

Figure 3. Experiment building with graph (i), building with occupants (small circles) and hazard start locations (large, outlined circles) (ii)

The initial fire was placed in one of 12 starting spots drawn from a 4x3 grid (Fig 3 (ii)). In each case, we noted the running time for our parallelization scheme, the number of orthogonal simulation instances produced, and the largest and average size of each simulation instance in terms of constituent groups. These experiments were repeated for an 80-second look-ahead period, to determine what impact increasing the duration of time investigated had on the number of simulation sets and on runtime of the problem decomposition routines.

We also timed individual EvacSim instances using a variety of crowd densities, to determine how the number of occupants in a simulation set impacts the running time of the simulation instance. This experiment timed the amount of computation time it took to calculate 40 or 80 seconds worth of evacuation time through a range of population densities. Experiments (decomposition task, and EvacSim runtime test) were performed on a single core of a Intel Core 2 Duo 3.0Ghz processor using 1Gb of RAM.

5.2 Results

For a population of 85, with both look-ahead durations, we found that EvacSim simulated the *entire* population in a fraction of "real" time, 1.5 seconds for a 40-second simulation look-ahead period and less than 3 seconds for an 80-second period. Similarly we found that EvacSim was capable of producing simulation results at a fast rate for the population of 350, with runtimes of 7 seconds for simulating 40 seconds of evacuation, and 13 seconds for an 80-second simulation period.

These runtimes (Figure 4) indicate that the upper bound on the run time of a single simulation instance for these population sizes is quick enough for predictive purposes. We also found that for small occupant numbers, EvacSim simulates the look-ahead period much more quickly, suggesting that a large number of low-density simulation instances can be completed in a short period of time.

Additionally, we found that increasing the population (and thereby, occupant density) the simulation running times quickly become too high to provide predictions within the time-frame. This suggests that it is much preferable to produce a large number of low-population simulation instances rather than a small number of high-population/high density simulations each of which would take a significantly longer time to complete.

We recorded the decomposition running time and the number of simulation instances suggested in each case (Table 1). We also noted the number of occupant agents that would be present in each of these simulation instances, to gauge the running times of the simulation instances based on the running time results shown in Figure 4

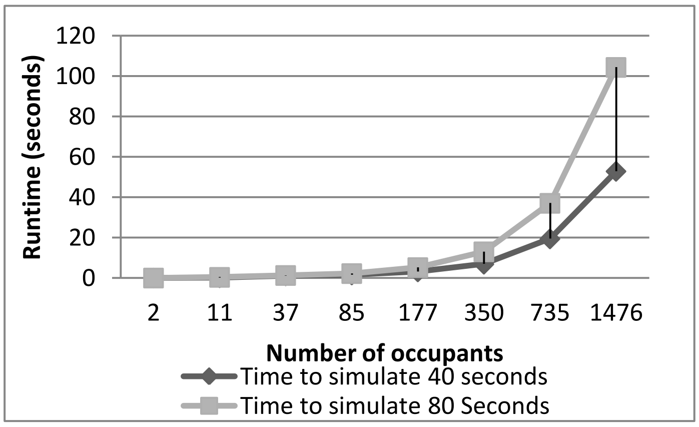

Figure 4. EvacSim running times for a range of occupant populations, simulating 40 and 80 seconds of evacuation time

	AVERAGE DECOMPOSITION RUNTIME(SECONDS)	AVG. NUMBER OF SIMULATION INSTANCES	AVG. OCCUPANTS PER SIMULATION INSTANCE
85 occupants, 40 second look-ahead	Average: 0.1420 Standard Deviation : 0.0984	Average: 25.8 Standard Deviation: 1.5	Average: 5.7 Standard Deviation : 0.4
85 occupants, 80 second look-ahead	Average: 0.1724 Standard Deviation: 0.117	Average: 29.8 Standard Deviation: 7.2	Average: 6.8 Standard Deviation: 2.3
350 occupants, 40 second look-ahead	Average: 0.3531 Standard Deviation: 0.1875	Average: 40.2 Standard Deviation: 2.3	Average: 32.4 Standard Deviation: 1.5
350 occupants, 80 second look-ahead	Average: 0.9741 Standard Deviation: 1.0832	Average:64.9 Standard Deviation: 8.2	Average: 51.2 Standard Deviation: 8.3

Table 1: Problem decomposition experiment results

We found that the decomposition method suggests average simulation instance counts well below 100, and for the lower crowd density and lookahead period the number of instances was very low. For the most part these simulation instances feature a population count which is a fraction of the total building occupancy, suggesting running times between 0.4 and 1 seconds. For a small 20-core computer, the worst case total set of simulation instances would be computable in less than 10 seconds, and with 0.2-3 second decomposition computation times, this should provide evacuation outcome predictions well within the time-frames we are simulating for.

For long look-ahead periods, the tendency is for crowds to merge together more often as queues begin to form at exits. We found that for an 80-second look-ahead period, the average number of simulation instances is higher relative to 40-seconds, reflecting the tendency for large merged groups to form as CrowdRoutes converge on exits. As a consequence, this produces a relatively high number of Simulation Instances due to the greater degree of route sharing that could occur between the interacting groups.

For some hazard positions, only a limited subset of the building exits appear in evacuation routes as the hazard blocks some escape routes. In these cases, there is a tendency to produce a large merged group as the majority of occupants head towards just two or three remaining exits. In these cases, there is a greater runtime for the problem decomposition software relative to other hazard positions (3 seconds vs 0.2-0.4) as there is a greater amount of crowd merging operations to compute. A hazard spawning in the leftmost position in the second row cuts off a number of exits. In the experiment using 80-second look-ahead and 350 occupants, this led to 80 EvacSim instances and with an average of 62 occupants in each one. This result suggests that a balanced decomposition of the problem might not always possible using this technique alone, as in some scenarios (e.g. limited exits or highly restricted flow) there is a low degree of orthogonality.

Hazard position has a significant impact on the balance of the simulation sets; a single high-population simulation instance is more computationally expensive to compute than a small set, and may constitute a simulation bottleneck as we wait for the large set to finish simulating before we can make use of its prediction results.

6 Related Work

Efficient distributed evacuation simulation requires a balanced division of work. Examples of domain decomposition techniques in this field generally divide the simulation space based on spatial proximity of occupant agents. Mohedeen [6] investigates spatial decomposition approaches based on balancing the number of occupants simulated at each processor; achieved via space partitioning and exit-count balancing. This thesis briefly suggests the possibility of an initial simulation to predict evacuation trends (which could be used to partition the simulation domain), but discounts this as too time-intensive during an emergency scenario.

Wagoum et. al present a force-based distributed evacuation simulator [2]. This simulator makes use of a 2D vector-force based simulation model similar to the EvacSim simulator used in our work and parallelizes the simulation task via spatial decomposition of the building based on exit placements. This simulator modeled a population of 22,500 occupants evacuating a sports arena, and demonstrated the ability to simulate events at up to 2.5x faster than real-time when using a large number (160) of parallel cluster nodes; demonstrating the feasibility of distributed simulation as a predictive tool.

Our work differs from these methodologies as it is based on spatial and temporal proximity, and this approach allows us to accommodate a significant amount of occupant decision branching. To our knowledge, network-flow graph analysis as a parallelization technique is a novel approach.

A variety of multi-agent evacuation simulators have been developed to model building evacuations. Sharma [4] described modeling of occupant personality as an agent behaviour strategy, featuring panic states. Filippoupolitis et al's [5] distributed simulator incorporated a simulated sensor network in a graph-model based simulation. Korhonen et al's [7] work combined a fluid-dynamics-based fire simulation with a field-based pedestrian model. Zia[20] investigated agent cognitive models when interacting with technological assistance in the form of a guidance belt [17] during emergency evacuation. These agents model emotional states and exchange of route information and leader-following.

Kooa et. al. [19] developed a framework for rapidly simulating fire spread in a reduced-fidelity simulator, and respond to updated information from building sensors. This updated information adjusts their fire simulator model *during* the emergency to match the fire observations from the sensor network. This cyclical "simulate-and-adjust" approach share similarities with our work as it acknowledges that some emergency features cannot be simulated to a high degree of fidelity within the short time frames involved.

7 Conclusions and Future Work

We developed a novel methodology of filtering future states of an emergency evacuation based on assumptions about occupant behaviour, and analysis of the building structure in the form of a Network Flow Graph. In this work, we demonstrated that this approach can filter out a vast number of future states which are not likely to happen in the evacuation, based on spatial and temporal proximity of occupants evacuating. In our experiments we found that this problem decomposition approach produces a manageable set of future states to simulate, for a typical building and population size. Our network flow graph-based problem decomposition computes quickly and produces sub-problems for simulation that can individually simulate fewer occupants than the overall evacuation scenario.

For future work, we intend to investigate a greater variety of building topologies and emergency scenarios at, such as large scale evacuation of stadia or cities. Our experiments showed that our decomposition technique works well for a 350-occupant evacuation scenario, on larger scales there may be many thousands of agents and greater computational resources.

Our experimental results show that in cases of dense populations, there is a tendency to produce a number of large population simulation instances due to large combined groups forming. In the case that a large group like this forms, the likelihood is that the individuals involved will move slowly, and this may be a fruitful area for exploiting spatial-locality domain subdivision methods such as those suggested in [2,6].

In this work, we developed and implemented a methodology that produces numbers of evacuation simulation instances to compute. In future, we intend to integrate this with a distributed computation platform to determine how appropriate this methodology is when dealing with the constraints and features of distributed computers.

8 References

1. M. J. Akhlaghinia, A. Lotfi, C. Langensiepen and N. Sherat Occupant behaviour prediction in ambient intelligence computing environment. Journal of Uncertain Systems Vol 2, Issue 2, p85-100. (2008)
2. A. U. K. Wagoum, M. Chraibi, J. Mehlich, A. Seyfried and A. Schadschneider: Efficient and validated simulation of crowds for an evacuation assistant. Computer Animation and virtual Worlds Volume 23, Issue 1, p3–15, January/February (2012)
3. A. Kneidl, M. Thiemann, A. Borrmann, S. Ruzika, H. W. Hamacher,G. Köster,E. Rank. Bidirectional Coupling of Macroscopic and Microscopic Approaches for Pedestrian Behaviour Prediction. Pedestrian and Evacuation Dynamics. (2010)
4. Sharad Sharma. Modeling and simulation of multi-agent systems for emergency scenarios. Doctoral dissertation, Wayne State University Detroit, MI, USA. (2006)
5. Avgoustinos Filippoupolitis , Laurence Hey , Georgios Loukas , Erol Gelenbe , Stelios Timotheou, Emergency response simulation using wireless sensor networks, Proceedings of the 1st international conference on Ambient media and systems, p.1-7, February 11-14, (2008)
6. Y Mohedeen. Domain Partitioning and software modifications towards the parallelisation of the buildingEXODUS evacuation software. PhD Thesis, University of Greenwich (2011)

7. Korhonen, T., Hostikka, S., Heliövaara, S., Ehtamo, H.,Matikainen, K.: Integration of an Agent Based Evacuation Simulation and the State-of-the-Art Fire Simulation. Proceedings of the 7th Asia-Oceania Symposium on Fire Science & Technology, pp.20-22 (2007)
8. Chih-Yung Chen: A Fuzzy-Based Approach for Smart Building Evacuation Modeling. Innovative Computing, Information and Control (ICICIC). 2009 Fourth International Conference Issue Date: 7-9 Dec. 2009 pp. 1200 - 1203 (2009)
9. YanYan Chu, Hui Zhang, ShiFei Shen, Rui Yang and LiFeng Qiao. Development of a model to generate a risk map in a building fire. Science China Technological Sciences Volume 53, Number 10, p2739-2747 (2010)
10. D. Dressler, M. Groß, J. Kappmeier, T Kelter, J. Kulbatzki, D. Plümpe, G.n Schlechter, M. Schmidt, Martin Skutella and S.Temme: On the use of network flow techniques for assigning evacuees to exits. Procedia Engineering Volume 3, 2010. pp. 205-215. 1st Conference on Evacuation Modeling and Management, (2010)
11. Tarik Hadzic, Kenneth N. Brown, Cormac J. Sreenan. Real-Time Pedestrian Evacuation Planning During Emergency. ICTAI 2011: Proceedings 23rd IEEE International Conference on Tools with Artificial Intelligence, November 2011, Boca Raton, Florida. (2011)
12. S Opasanon and E Miller-Hooks. The Safest Escape problem. Journal of the Operational Research Society, Volume 60, p1749–1758 (2009)
13. Jay E. Aronson. A survey of dynamic network flows. Annals of Operations Research Volume 20, Number 1, p1-66 (1989)
14. Sangho Kim, Shashi Shekhar. Contraflow network reconfiguration for evacuation planning: a summary of results. Proceedings of the 13th annual ACM international workshop on Geographic information systems, , p250-259 GIS (2005)
15. Makino, K. Takabatake, T. Fujishige, S. The evacuation problem, dynamic network flows, and algorithms. SICE 2003 Annual Conference. p2807-2811 Vol.3. (2003)
16. Reza Olfati-Saber, Flocking for Multi-Agent Dynamic Systems: Algorithms and Theory. IEEE Transactions on Automatic Control, Volume 51, Issue 3, p401-420, March (2006)
17. A Ferscha, K Zia. Lifebelt: Silent Directional Guidance for Crowd Evacuation. ISWC09, p19-26, 4-7 September, (2009)
18. Luca Chittaro, Daniele Nadalutti . Presenting evacuation instructions on mobile devices by means of location-aware 3D virtual environments. MobileHCI '08 Proceedings of the 10th international conference on Human computer interaction with mobile devices and services, p 395-398, September 02-05, (2008)
19. Sung-Han Kooa, Jeremy Fraser-Mitchell, Stephen Welch. Sensor-steered fire simulation. Fire Safety Journal, Volume 45, Issue 3, April 2010, p193–205 (2010)
20. K. Zia, A. Riener, A. Ferscha, A. Sharpanskykh. Evacuation simulation based on a cognitive decision making model in a socio-technical system. DS-RT 2011 15th International Symposium on Distributed Simulation and Real Time Applications. Salford/Manchester, UK. September4-7, (2011)
21. William Mattson, Betty M. Rice. Near-neighbor calculations using a modified cell-linked list method. Computer Physics Communications, Volume 119, Issue 2-3, p135-148 (1999).