CHESTER COLLEGE
A College of the University of Liverpool

Evaluation of a lift control algorithm for the emergency evacuation of tall buildings.

By

Keith Ennion

Dissertation submitted in accordance with the requirements of Chester College; in partial fulfilment of the Degree of Master of Science in Information Systems.

ABSTRACT

This paper considers the hypothesis that a suitably designed lift system can be used for the automatic evacuation of tall buildings. It will identify the key features that should be provided to ensure that lifts are controlled efficiently during this task and so ensure that the evacuation time is reduced to a practical minimum. An algorithm is described which has been designed in line with these principles and its performance is evaluated by simulation. The performance is then assessed against a standard control algorithm and also against a theoretical best possible solution. The initial results indicate that a dedicated evacuation algorithm can provide significant benefits above a more generalised control strategy for this particular traffic situation and that it is less susceptible to variations in passenger arrival patterns.

DECLARATION

This work is original and has not been previously submitted in support of any course or qualification.

Signed: -

ACKNOWLEDGMENTS

I would like to express my appreciation for the help and encouragement given by my tutor Garfield Southall, during the various flights of fancy that lead me to this project. His patience and tolerance has been greatly appreciated. I would also like to thank Mohammed Saeed for his gentle prompting, without which this project would never have been attempted. I would also like to thank Gary Malbon of LiftStore Ltd, who made the Elevate Simulation software available to me, without which the analysis could not have been performed.

Finally, but by no means least, I would like to thank my wife, Marian for her understanding, encouragement and efforts in proof reading this work.

CONTENTS

FIGURES

TABLES

1 Introduction

The Lift has been an integral part of medium and high-rise architecture since its inception. Indeed without it, it is doubtful that buildings would ever rise higher than about 6 floors. In the public's mind however, this form of transport has been seen as an unfortunate necessity; often giving rise to apprehension and mistrust. A plummeting lift car, or being trapped in a stuck lift in an empty building, has been the theme of many films, or drama series.

As is usually the case the reality is somewhat different from the public perception. In fact a modern lift system is subject to a high degree of regulation, making it one of the safest forms of transport available to the public. Modern building regulations also provide a high degree of security for the occupants of tall buildings, however the nature of their design carries the potential for a high level of fatality, should a disaster befall them, for this reason a means of evacuation is an integral part of the building design. Most building regulations disregard the lifts as a safe means of evacuation during an emergency and rely on protected stairways. Historically this was a prudent view to take of the lift system, however modern standards are such that lifts can be designed to operate safely under emergency conditions.

In the UK, legislation has been put in place in the form of the Disability Discrimination Act and in BS5588. These are currently being introduced to ensure disabled access to all buildings. In terms of the lift system, this ensures wheelchair users can gain access to the lift, that they can reach the landing and car-call buttons and that Braille legends enable buttons to be identified by the partially sighted

passengers etc. BS5588 recommends standards for the construction of fire fighting and evacuation lifts, ensuring they can be used safely during a fire and so enable their use in the managed evacuation of disabled people from the building. Unlike the regulations on access, it makes no recommendation that all buildings should be provided with this type of lift. This leaves the provision of a means of evacuation for disabled people, to the discretion of the building owner.

The Canadian Institute for Research in Construction has looked at this general area (Guylene, 2002) and has identified the use of "Safe Lifts" in conjunction with refuge areas (areas of high fire protection, isolated from the main floor area) as enhancing the safe evacuation of disabled occupants. The report also identifies special buildings such as the Stratosphere Tower in Las Vegas, where the lifts are seen as the primary route for evacuation for all occupants, not just the disabled.

In the UK the issue has also been raised. In his paper "In the event of fire – use the elevator" (Hawkins, 2000) points out that valuable evacuation time is often wasted by current practice. When a positive fire signal is received, it is common practice to return the lifts to the home floor and then lock them out until the emergency services arrive to provide managed control of the system. The time the emergency services take to reach the building is likely to be variable. In a city centre during working hours a response time of 5 – 10 minutes would be considered good. Modern lift systems are designed with a capacity to move a good proportion of the maximum building population (10% - 25%) in a 5-minute period. It can be seen therefore that if the lift system were considered safe for automatic operation during this critical period, it could contribute significantly to the evacuation before any rescue service arrives.

Given that construction standards have now been introduced, to make a lift safe for the use of fire fighters and the supervised evacuation of the disabled, it seems a natural further step to protect the whole lift system and enable it to play a role in the automatic evacuation of the building. To achieve this the normal operating behaviour of the lift will need to be superseded by an algorithm optimised for evacuation. It will be the purpose of this paper to look at the issues involved in such an automatic control algorithm, to implement an algorithm based on this analysis and to evaluate its performance using a commercially available lift simulation program.

Following this introduction the paper will outline the basic features of a lift control system and the general features that have a bearing on the evacuation issue. It then moves on to identify specific aspects of the situation that must be considered in the design of an evacuation strategy. In section four, the features of the proposed algorithm are explained. This is followed by a description of the Elevate simulation software to be used in its evaluation. A discussion of the tests performed and an analysis of the results obtained is then given. The paper then draws together these results in the conclusions and future work is identified.

2 Lift Control Principles

The operation of a lift system may at first appearance seem to be a simple matter. The prospective passengers simply press a button; a lift arrives and the doors open. They get in, select a destination floor and are delivered to their destination quickly and safely. This apparent simplicity is in fact the culmination of over 150 years of development, which has encompassed developments in Mechanical, Electrical, Electronic and Computer engineering. By the 1950s the introduction of automatic doors removed the need for a lift attendant and the fully automatic lift had arrived.

2.1 Collective Control

At every floor of the building each lift has it's own *landing-call* button. The term *landing* referring to the area on each floor immediately outside the lift and from which the passenger enters the lift. Pressing the landing-call button summons the lift car and allows the passenger to enter. The passenger then selects a destination using buttons within the car; the *car-call* buttons. The lift car travels to its destination and the passenger alights at this point the lift is free to accept further calls. This is the simplest type of automatic control, known *as non-collective control.* It can be implemented relatively cheaply, requiring only simple control logic and a single call button at each landing. It is however only suitable for low demand situations, such as freight lifts, because it can only handle one call at a time.

By enabling the lift to remember landing-calls and by providing two buttons at the landing, one to register a desire to travel up the building and the other to travel down, it has been possible to significantly improve the performance of the lift system. A lift

committed to travel in one direction, is able to detect other calls to move in the same direction and is therefore able to pick up or "collect" further passengers as it reaches these intermediate floors. The provision of this ability in both directions of travel is known as *full-collective control* or *directional collective control.* As with most lift control systems, priority is given to the passenger in the lift who registers their desired destination through the car-call buttons. Landing-calls are only collected if; there is room in the lift, they are registered in the lift's direction of travel ahead of its current position and the lift has sufficient time to stop before the landing is reached. When all the car-calls have been served and there are no further landing-calls ahead of it in its direction of travel, the lift will answer the furthest landing-call in the opposite direction. When it reaches that call it will collect a new passenger and start its run in the opposite direction.

2.2 Group Control

As passenger demand increased, buildings were equipped with multiple lifts, often collected together in the same area of the building. Using individual control systems for each lift showed up further areas of inefficiency. A person in a hurry wishing to travel down the building may well register a down call with each lift. As the lifts are working independently they would each try and respond, however when the first car arrives the passenger gets in and completes the journey, meanwhile the other lifts also responded and travelled to an empty landing. It was clear that to improve matters some form of co-operation between the lifts was required and this came in the form of a group controller. Instead of wiring the landing-call directly to the lift control system, it was brought into the group controller. This control element has an overview of the lift system; knowing the current position and direction of each of the

lifts and the current state of demand from the car and landing-call buttons. With this information, it is possible to allocate the landing-calls to the lift that is best positioned to serve the call. A passenger placing a landing-call on this type of system may press the call button by one set of doors and find that the lift arrives at any of the entrances. How this decision is arrived at is known as the *car allocation strategy* and has been the basis of much research over the past twenty years, using both conventional and AI techniques.

2.2.1 Nearest Car Allocation

One of the simpler conventional techniques for car allocation is termed *Nearest Car.* This works on the full-collective principle, providing two landing-call buttons at each floor apart from the terminal floors, where only one direction is required. The control scheme is expected to space the lift effectively around the building in order to provide an even service. It will also follow a parking policy to pre-position the cars, commonly at the ground floor or other high demand areas, when the traffic is light.

Any car-calls are specific to an individual lift and are treated as the priority calls for that lift. This prevents the group controller from sending the lift in the opposite direction to that requested by the passenger. A landing-call on the other hand is allocated to a lift that the group controller considers to be the best placed to answer it, the nearest car. The search for the nearest car is continuously performed until the call is serviced. This is possible due to the simple nature of the calculation and so enables the system to adapt to the changing dynamics of the situation. The term "Nearest Car" is somewhat misleading in that it is not simply concerned with how far the car is from the landing-call. Apart from the separation distance the algorithm takes into account the requested direction of travel for the landing-call and the current direction

of travel of the lift. The exact nature of the algorithm will be specific to the individual manufacturer, but Barney (2003) gives a sample solution

He measures distance 'd' in terms of the number of floors between the landing-call and the position of the car under consideration. The number of floors above the ground floor is given as 'N', so that the total number of floors in the building is (N+1). The objective of the algorithm is to calculate a figure of suitability 'FS' for each lift that has space available. It then allocates the landing-call to the car with the highest FS value. The basic equation is given by: -

$$\mathbf{FS = (N+1) - d}$$

This is used for cars that currently have no allocated landing or car-calls and are consequently idle; or for a car that is committed to moving towards the landing-call, but in the opposite direction to that required by the landing.

For the condition when the car is moving towards the landing-call and in the direction requested by it a factor of 1 is added, giving the equation: -

$$\mathbf{FS = (N+2) - d.}$$

The final possible condition is when the lift is moving away from the landing-call. For this situation the figure of suitability is set to one i.e.

$$\mathbf{FS = 1}$$

Having made this calculation for each lift the landing-call is allocated to the lift with the highest FS value. In the case of more than one lift having the same value the call is allocated to the first to be calculated. The process is then repeated for any other

landing-calls that may be present. Having allocated all the calls the system waits for the next time slice, typically 100 milliseconds and the whole process is then repeated. This allows the system to adapt to changes in movement or direction caused by the car-calls made by passengers. As the number of lifts and size of building increase, the number of possible calls increases rapidly and so the time to perform the calculation can become critical.

2.2.2 Bunching

According to Siikonen (1997) one of the drawbacks of this system and others that use the collective control principle, is the *bunching of lifts*. During heavy traffic periods, when people are entering or leaving the building, there are a lot of calls to serve and the lifts have a tendency to move side by side, i.e. they start to bunch together at the same level within the building. This happens because lifts always stop at the nearest call and by-pass hall calls only when fully loaded. Early systems coped with this by running the lifts in a scheduled manner, in a similar way to a bus service. A car arriving at the ground floor would not be permitted to leave until a scheduled time. This had the effect of distributing the cars more evenly around the building, but reduced overall capacity. Other schemes giving priority to calls registered for a long time and the ability to bypass calls registered for a short period, also had a desirable result. During peak traffic conditions the application of Floor Zoning, Channelling (Powell, 1992) and "Next car up" (Barney and dos Santos, 1985) have all been used to tackle this issue.

2.2.3 Traffic Patterns

As mentioned in the previous section the actual flow of people around a building can have a significant influence on the efficiency of the lift system. For the majority of the day the traffic is likely to have no discernible pattern and is therefore termed *Random inter-floor traffic.* For this type of traffic a full-collective control system is well adapted and together with an intelligent parking policy can provide rapid response to the moderate levels of traffic during the course of the day.

The most demanding routine traffic pattern is widely recognised as being the morning *Up-peak.* This condition occurs as people arrive for work in the morning, enter at the ground floor and travel up the building. The designation of an official start time causes a synchronisation of the occupant's movements and so places a heavy demand on the lift system. Being considered the worst case condition, it is the condition around which the lift system's design calculations are based. The basic premise being that if the system can provide an acceptable service for this condition, it will cope with any other traffic condition. It is also a traffic pattern, which is amenable to theoretic calculation and as such has been the basis of much research.

Under the up-peak traffic condition full-collective control does not provide an optimal solution and special up-peak control strategies can be applied. If all the people entering the building are trying to travel up, it would be wasteful to allow a lift car to depart the ground floor with just a few occupants. Delaying the normal door closing time may allow further people to enter and so fill the lift. Similarly when an empty car arrives back at the ground floor, it may be wise for it to remain with doors closed until a lift, still filling, is ready to depart. These and many other specialised strategies have been adopted to improve performance under this traffic condition.

A similar traffic condition occurs at the end of the day, as people leave the building, the *Down-peak.* Again the simple application of full-collective control does not perform well under these conditions. When the car empties at the ground floor the car allocation algorithm looks for up calls, or the furthest away down call. As most of the floors in the building will have down calls, the algorithm sends the car to the highest calling floor, so that it can collect from other floors on the way down. The problem arises from the volume of people trying to travel. The lift car will fill at the first or possibly second floor it stops at and will then travel down the building bypassing the lower and intermediate floors. By the time the car is empty more passengers have placed calls on the higher floors and the process repeats, leaving the lower floors with poor service. Again the solution is to force a different traffic handling strategy. Techniques that prove useful for the up-peak condition can also be applied usefully to the down-peak.

2.3 Zoning

The division of a building into a number of Zones, each served by dedicated lifts has been known for some time to improve overall system throughput under peak traffic conditions. Barney (Barney et al, 1985) shows that for a 24 floor high-rise building, the up-peak performance can be improved, using two less lifts, if the building is split into two zones. The first zone has 6 dedicated lifts to serve floors 1- 13, while the other zone serves floors 14 – 24 with a further 6 lifts. The benefits of zoning arise from the reduction in the number of times a lift has to stop during a round trip, the fact that it forces a more even distribution of lifts around the building and the equalisation of traffic load between zones.

Reducing the number of times a lift must stop, is the most important benefit. As a significant amount of the journey time is taken with the acceleration/deceleration of the lift and the door operating times. By zoning the lifts, passengers with common destinations are forced to use the same lift cars and therefore as a proportion of the capacity of the lift, the number of people travelling to a single floor is increased. Another benefit of the zoning scheme described above is that for the high-rise zone, the lifts are not expected to stop at the lower floors. This means the lift can always reach and maintain maximum speed for over half the height of the building, this would be significantly less likely, if zoning were not in place. A further benefit is a more even distribution of lifts around the building, making the system more responsive and so reducing average waiting times. It is in effect an anti-bunching feature.

Finally provided the zone boundaries are selected correctly the passenger traffic can be shared out equally. As can be seen from the zoning example above the lower zone or *Local Zone* serves 13 floors while the second zone or *high-rise Zone* serves 11. Obviously there is further to travel to service the floors in the high-rise zone and this is balanced by the fact that there are fewer floors to service. The selection of this zone separation point has been the focus of much research in dynamic zoning and will be discussed shortly.

2.3.1 Zoning Methods

There are three methods by which Zoning is commonly implemented in buildings; static zoning, layout-time zoning and dynamic zoning.

Static zoning is the permanent allocation of certain lifts to serve a restricted number of floors within the building and is often a feature of the architecture of the building. The local zone lifts only have lift shafts, which reach partway up the building, while the higher zone lifts will not have entranceways to the lower floors, just to the ground floor. In fact for many high-rise buildings the high-rise zone may not have a direct link to the ground floor at all. People wishing to reach one of these higher floors will take an express lift to a restricted number of sky lobbies, where they transfer to lifts servicing that zone. The World Trade Centre for example had a main entrance lobby at ground level and two sky lobbies at floors 44 and 78. Each sky lobby was linked to the ground by its own group of express lifts and a further express group linked the two sky lobbies. Each of the three lobby floors then served the floors above them, which were divided into four zones, giving a total of twelve zones to service 110 floors.

Layout – time zoning is a temporary form of zoning set to operate at particular times of the day, to cope with peak traffic conditions. An information panel or indicator at the entrance to the lift informs the passenger when zoning is in operation and which floors the particular lift car is serving. While the use of zoning can be seen to provide increased handling capacity during peak flows of people into and out of the building, it does have disadvantages for other traffic conditions. For most of the working day, random inter-floor traffic will form the bulk of the journeys within the building. In these circumstances zoning can be seen to reduce the performance of the system for journeys between zones. This is because a passenger has to travel to an intermediate transfer floor, before being able to enter a lift capable of travelling to the required destination. By engineering the building so that all lifts can service all floors and then allowing the control system to zone the lifts just at peak times of day, it is possible to provide an efficient means of coping with both types of traffic condition.

Dynamic zoning is a more flexible method of implementing zoning. With the previous types of zoning, the zone boundaries are either permanently fixed by the design of the building, or at least pre-programmed into the controller. The introduction of modern control systems enables closer monitoring of the prevailing traffic pattern and provides the potential for adapting the control regime to suit the traffic pattern. It aims to dynamically allocate the zone boundaries in an optimal manner for the current traffic condition. Chan & So (1995) first introduced the concept for up-peak and down-peak traffic conditions and then further developed it for random inter-floor traffic (So & Chan, 1997). Their approach is to provide as many zones as there are lifts and adjust the zone boundaries in such a manner that it minimises the Round Trip Time (RTT) for all the lifts within their respective zones and so ensure an even traffic load. The strict rule: -

$$\mathbf{0 < n_1 < n_2 .... < n_j < .... < n_{m-1} < N}$$

Defines the **m-1** zone boundaries between the Ground Floor = **0** and the highest floor in the building = **N**. The mathematical problem becomes the minimisation of the following cost function: -

$$\underset{n_1,\ldots,n_{m-1}}{\mathbf{Min}} \quad \sum_{j=1}^{m} \mathbf{RTT_j^4}$$

Where: -

The number of lift cars = the number of zones	= **m**
The number of floors above the ground floor	= **N**
The highest floor served by car 1 in zone 1	= $\mathbf{n_1}$
The round trip time for the j^{th} car	= $\mathbf{RTT_j}$

The solution of this problem is however non trivial. Chan & So initially described a solution, based on a standard Lagrange function together with Kuhn-Tucker

conditions and then in their second paper a solution based on a genetic algorithm. Further work has however criticised these solutions for practical implementation into real time systems, due to the intensive nature of the calculations involved. Qun has proposed two alternative methods (Qun, Zen-Shi & Xiago-guang, 2001) and (Qun, Xin-Yu & Zen-Shi, 2002) involving a direct search numerical method and a dynamic programming approach. Which if any of these solutions will be deemed practicable is beyond the scope of this paper.

3 Factors affecting the performance of an automated evacuation.

If we accept the hypothesis that in the future a suitably designed lift control system can be used to provide a safe means of evacuating medium and high-rise buildings, we must then consider what control strategy it should adopt and the factors that will govern its operation

3.1 Everybody-Out vs. Prioritised Evacuation.

The optimum strategy may well depend on the type of emergency and how it's perceived by the occupants of the building. Certain threats such as fire may present an immediate threat to a relatively small proportion of the buildings population, under these circumstances it can be argued that the everybody-out option is not the best strategy, as it could delay the evacuation of those at most risk (Guylene, 2002) also (Howkins, 2000). They propose a sequential evacuation, where floors are evacuated by priority. Starting with the affected floors and those directly threatened. It is however conceded that this type of evacuation requires training and good communications, to be effective. Guylene describes three stages of training, consisting of talk-through, announced drills and finally surprise drills. It is also noted however, that maintaining the motivation of occupants to take part can be a difficult issue. Favro (1997) quotes studies in human behaviour, which clearly show that people in emergency situations do what they are used to. They tend to follow their familiar route out of the building, or if they are just visiting, to reverse the way they came into the building. In both cases they tend to gravitate towards the lifts, even if warned not to do so. Keating and Loftus have found that under heightened anxiety,

people's attention becomes narrowly focused, making them aware of only the most obvious aspects of their environment. Under these circumstances habit will tend to take over and specific emergency procedures tend to be forgotten. Favro (1997) goes on to question the concept of partial evacuation, considering it to be naive. He points out that people faced with a threat, either real or perceived, will attempt to flee. It is unlikely that they will be convinced that they are not in danger, if they see what they consider to be evidence of an immediate problem. He quotes an incident at the World Trade Centre in 1975. In this case a fire involving a wastebasket, which was confined to a single room, caused the spontaneous evacuation of 11 floors, though the smoke was light, non-lethal and the occupants were told that evacuation was unnecessary.

Other types of threat may require evacuation in a less specific manner. A terrorist warning of a bomb may well require the full evacuation of a building by the quickest means possible, presenting no scope for prioritising floors. It must also be conceded that at the start of an emergency, precise information on the threat is a rare commodity. While it may be feasible to think of an automatic transfer of information from the fire control system to the lift control system, this is only likely to happen in buildings with highly integrated systems.

With these points in mind, we must consider the everybody-out option as being the more general strategy. But to allow a prioritised evacuation to be implemented if and when it becomes clear that the threat is localised within the building and the floors of higher risk are identified. Our algorithm will need to be flexible to cope with both conditions.

3.2 Effects of Bunching

Since fully automated lift groups have been in use, it has been known that under peak traffic conditions there is a tendency for their movements to synchronise. Al-Sharif (1995) has described a method of determining a bunching factor, which shows when bunching comes into effect for a group of lifts serving the same floors. Using simulation techniques, he has been able to demonstrate the process starting to take effect in an eight car group, at system traffic levels of as little as 40% of the total lift capacity and at 60% for a two car group. His findings indicated that the main result of bunching was to increase the average waiting time for the system, but not to materially affect the system handling capacity. Previous work done by Poschel (1994) reported on the dynamics of a jamming effect as large numbers of occupants leave the building. He describes it as the "Feierabend Effect", or end of day down peak. These conditions are a less extreme example of what could occur during an emergency evacuation. As with Al-Sharif's work he also describes a threshold effect as passenger movements increase. His work was again based on simulation, but in this case he implemented two forms of control strategy. Both were based on Nearest Car allocation with full collective control. One algorithm was able to make use of car weighing to ensure full cars are not stopped at intermediate floors. He refers to this strategy as "smart" control, while the other algorithm without car weighing was referred to as "naive". He showed that a naive control algorithm demonstrates the bunching threshold earlier than the smart algorithm and that passenger throughput falls back after the threshold is reached. Thus reducing the overall capacity of the system just at the point it is needed the most.

For an evacuation strategy these effects must be avoided, therefore our algorithm will attempt to avoid bunching and will implement car weight to measure the level of occupancy of the lift, as used in the smart control algorithm.

4 Features of the proposed evacuation algorithm

An automatic evacuation strategy has the task of moving the building population to the evacuation floor, typically the ground floor, in the fastest time possible. This essentially means minimising the number of times a lift stops during the round trip from the evacuation floors and back again, while ensuring it returns with a full load. As a general principle it will assume an everybody-out policy is being adopted, but will make provision for a prioritised strategy to be imposed should the nature of the emergency make that approach more appropriate. This decision would need to be taken as part of the evacuation management plan and would be made by a suitably authorised member of staff. The control equipment will be assumed to include a car weight sensor, which enables the level of occupancy of the lift to be estimated. A pre-recorded announcement should be played to inform all passengers that the lift is operating as an evacuation lift and will be travelling to the evacuation floor. In the event of overloading, or door obstruction further messages should be played to inform the passengers of the current status of the lift and so allow corrective action to be taken. As with the current standards (BS5588) it is assumed that the landing, from which the evacuees are to enter the lift, is a refuge area with adequate protection from smoke and fumes and therefore panic should not be a prime concern.

The evacuation algorithm will be designed to meet and cope with a number of potential conditions during the various stages of an evacuation. Initially the algorithm will allocate all the floors in the building to a number of zones; one lift per zone. This will prevent any form of bunching and also ensure an even distribution of lifts throughout the building. Should a prioritised evacuation from above a designated fire

floor be called for, then the zoning routine may be called again to remove the zoning allocation from low priority floors and redistribute the priority floors amongst the zones.

The next phase is to start allocating cars to landing-calls. For this exercise all landing-calls are treated in the same manner, regardless of direction. An up landing-call means that a person is at the landing and should therefore be moved to the evacuation floor in just the same way as if they had placed a down call. Initially as people start arriving at the floors a lift may not fill up completely before leaving the floor. This implies that there are no passengers left waiting at the floor and this factor is used to lower the floors priority for an immediate revisit. The partially filled car will then be allocated another floor to pickup from until it is full.

As the evacuation progresses the number of people waiting at the landing ensures that the lift is filled, each time it visits a floor. The objective now is to ensure that all the floors are visited regularly. The zoning of floors ensures that there is a lift prioritised to serving those floors, however within the zone there may well be a number of floors requiring service from just one lift. Conventional collective control tends to send the lift to highest floor first and then collect passengers on the way down. In this traffic condition it would result in a very poor service for the lower floors in the zone (Barney et al 1985). The lift would pass the lower floors on the way up; then fill to capacity at the highest floor and so return directly to the evacuation floor. Only when the higher floors are cleared would the lower floors in the zone see service. To avoid this happening the allocation process needs to track the last time each floor is visited and prioritise floors that have not had recent service.

As the evacuation proceeds it is likely that one zone will be emptied before the others. With a conventional zone this would result in the lift being designated as free and it would remain at a parking floor. In our case the zoning algorithm gives weight to landing-calls in surrounding zones, thus enabling the lift to continue to assist in other zones. This can then lead to the floors with higher populations being serviced more rapidly.

5 Elevate Simulation Software

The Elevate Simulation software is a commercially available software package, which enables designers to trial different lift configurations and control strategies in all types of buildings (Peters, 2001). It provides various modes of operation, including the ability to link in a custom designed control strategy to the main simulation program. This is the mode of operation used for evaluating the evacuation strategy.

Initially the simulation software needs to be configured with details of the building, its occupancy and the characteristics of the lift system. Having set these physical characteristics, the dynamics of the system can also be configured. This consists of the arrival rate of people at the landings, a probability factor for their destination floor, a time slice for each simulation step and finally a control strategy. This can be the basic algorithm provided as a standard with the package, or a custom package developed using Microsoft Visual C++ and accessed as a Dynamic Link Library (DLL).

For all the tests performed the simulator was configured to cause the total population of each floor, to arrive at its landing and try to travel to the ground floor, during the first five minutes of the simulation. A random number generator schedules the individual passenger arrival times using a seed value from the configuration, this determines which passenger arrives at what floor and when. Using the same seed value in subsequent runs will produce the same passenger arrival sequence and so enables a consistent comparison to be made between different control algorithms. By varying the seed value a different arrival pattern is produced, but in all cases the

overall arrival rate follows a Poisson distribution, which is generally accepted as a good model for passenger arrivals (Barney et al, 1985). As a passenger arrives at the landing, the control strategy is notified by the registering of a landing-call, it then responds by allocating a lift car to service the call. The dynamics of these processes, time to close the doors, car acceleration, travel speed and deceleration, are all simulated in accordance with the configuration information provided initially. The passenger is then simulated entering the lift and placing a car-call to the destination floor, in accordance with the probability factor designated. For our simulation this is 100% for the ground floor, forcing all cars to empty when this floor is reached.

As the simulation progresses, a mimic display, as shown in figure 1, shows an animation of the lift operations.

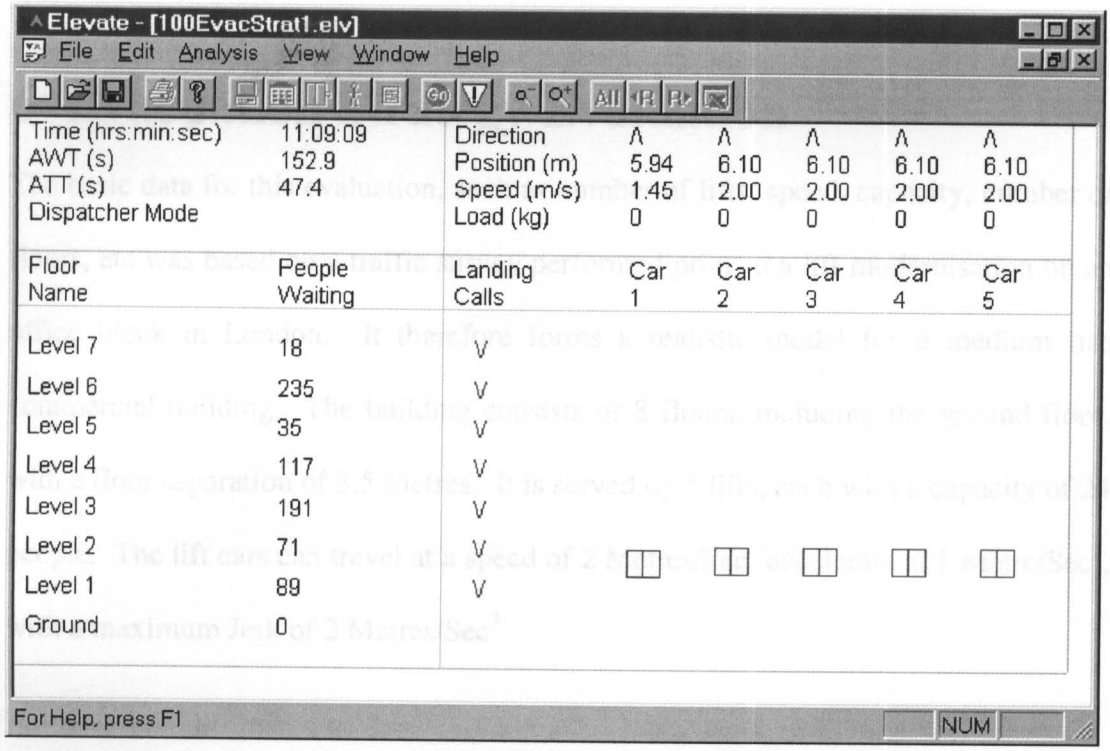

Figure 1 – Elevate simulation screen.

This allows a visual inspection of the system's operation, while at the same time statistics on each passenger's journey are being recorded. This information can then be used, at the end of the simulation, to provide more objective measures of performance, in the form of an analysis report. Appendix B shows typical reports generated during testing.

6 Evacuation Strategy Evaluation

The basic data for this evaluation, such as number of lifts, speed, capacity, number of floors, etc was based on a traffic survey performed prior to a lift modernisation on an office block in London. It therefore forms a realistic model for a medium rise commercial building. The building consists of 8 floors, including the ground floor, with a floor separation of 3.5 Metres. It is served by 5 lifts, each with a capacity of 24 people. The lift cars can travel at a speed of 2 Metres/Sec; accelerate at 1 Metre/Sec^2, with a maximum Jerk of 2 Metres/Sec^3

The floor population is significantly different for each floor and is shown in the following table.

Floor	Population
1	110
2	191
3	287
4	237
5	131
6	259
7	138

Table 1 – Floor Populations

6.1 Optimum evacuation.

To provide an assessment of the best possible performance, the spreadsheet shown in Appendix A was developed. This calculates the minimum time to evacuate the building, using the lift system alone, for an everybody-out evacuation. It takes the population for each floor and calculates the number of full car journeys required to remove the majority of people from that floor. The simulator was used to measure the

time to evacuate one person from each floor in turn, thus giving a round trip time for one person from that floor. The simulator allows 2.4 seconds for one person to enter and leave the lift. As the lift capacity is 24 the single person round trip time is extended by 55.2 Seconds (23 * 2.4) to give the round trip time for a full car to that floor. By inspection, the number of full journeys to nearly evacuate that floor can be allocated to various columns of the spreadsheet, representing the 5 lifts. The aim being to keep the total journey times for each lift as near as possible the same. To assess this, the average journey time and its variance is calculated for each lift.

Having allocated all the full car journeys, there are still people remaining at the floors. As the numbers will not fill a lift car, the car is allowed to stop at more than one floor until it is full. The journey times are calculated in a similar way to that when full, but allowing for the additional stop. These journey times are shown in the "Journey time for people remaining" column and are allocated to the lift columns as described previously.

The resulting prediction is the time each lift would be in continuous operation and provided the difference between these journey times is small, when compared to the individual journeys, then the times achieved will be the best that can be achieved from that system.

As can be seen from the spreadsheet Lift 2 will deliver the last person after 15.32 = 15 minutes 19 seconds. This should be considered the optimum value we are trying to achieve for this building.

6.2 Basic Simulation

To provide a base reference, the standard control algorithm supplied with the Elevate software was used to perform a series of runs. This is a nearest car allocation system, with full collective control evacuating the whole building population. The building and population data was configured into the system, with all the passengers arriving at the landings within the first five minutes and all passengers travelling to the ground floor as their destination. The simulation was allowed to run until all the passengers had been delivered to the ground floor and the elapsed time was recorded. The test was then repeated a further four times using a different seed number for the random passenger arrival generator. This change alters the pattern of passenger arrivals while retaining the overall Poisson distribution for the five-minute arrival period. Having made five simulation runs the average evacuation time was calculated to be 23minutes 31seconds as shown in Table 2. This is the base that any improvement can be measured against. A value for the standard deviation on the data was also made, which gives a measure of the variability between results.

Seed Value	Evacuation time
1	23 : 26
5	23 : 11
10	22 : 59
15	23 : 59
20	24 : 02
Average	**23 : 31**
Standard Deviation	**28.3**

Table 2 – Evacuation times using the basic simulation.

6.3 Evacuation Strategy Simulation

The Elevate simulation software allows custom algorithms to be evaluated by linking them in as a Dynamic Link Library (DLL) of the name "DispatchX.dll". This routine can be developed in Microsoft Visual C++ and is based on a skeleton routine

provided as part of the Elevate development software. The development pack provides header files for various modules within the Elevate software, making variables and functions available to the DLL. It also provides a skeleton version of the DispatchX.dll source code, which defines the interface routines that must be provided to the Elevate software for it to operate correctly. The Reset routine is called once at the start of the simulation to initialise any working variables within the DLL, while the Update routine is called every 100 milliseconds to allow the algorithm to update its calculations in line with the new dynamic position of the lifts. Before the simulation started the operator was able to select between alternative control algorithms, the case statement within this routine simply reflects that decision, calling the actual algorithm under test, in our case "Algorithem0". Section 4 described the features that the evacuation algorithm would seek to achieve; we will now describe how these features have been provided. The full algorithm is shown in Appendix C.

6.3.1 Zone Allocation

The Reset routine is used to initialise any working variables and to load certain building data constants. It then calls the Zone Allocation routine. For this test we are considering an everybody-out strategy, so the routine is passed the first and top floors, as the area to allocate zones between. In the case of a prioritised evacuation, the routine would be passed the lowest and highest floors to be evacuated, so that zoning would only be permitted for this area of the building.

The zone allocation process is essentially the problem addressed by dynamic zoning in Section 2.3.1. and will allocate a zone for each of the available lifts. However in our case we are not concerned with changing traffic patterns, the intention being to consider the zone allocation as just one of a number of factors for car allocation. We are therefore less interested in discovering the exact mathematical optimum solution,

than with providing a simple and quick calculation that will provide a fair allocation. Indeed even with an exact mathematically optimum solution, the zoning boundaries are not integer values, so must be rounded to the nearest floor, introducing a level of approximation. The method of allocation used in the zone allocation routine, is based on estimating the amount of traffic associated with the floors being zoned and then dividing this across the number of lifts available. This therefore allocates as many zones as there are lifts, as is the case with dynamic zoning. The zone allocation routine is shown below.

```
//-------------------------------------------------------------------------
// Allocate a zone number to each floor level

void dispatchX::ZoneAllocation(int StartFloor, int EndFloor)
{
int i;

   // Initialise all floors
   for ( i=1; i<= TopFloor; i++) {

      m_User1[ZONE_FOR_FLOOR + i] = 0;

   } // for i <= TopFloor

   // Cumulative Floor Population Distance.
   m_User1[CUMULATIVE_POP_DISTANCE] = 0;
   for ( i= StartFloor; i<= EndFloor; i++) {

      m_User1[POP_DISTANCE + i] = m_User1[POP_OF_FLOOR + i] *
                                  m_User1[FLOOR_HEIGHT + i] ;

      m_User1[CUMULATIVE_POP_DISTANCE + i] = m_User1[POP_DISTANCE + i] +
                               m_User1[CUMULATIVE_POP_DISTANCE + i - 1];

   } // for i <= EndFloor

   // Calculate an average population to be moved by each lift.
   m_User1[AV_FLOOR_POPULATION] = m_User1[CUMULATIVE_POP_DISTANCE + TopFloor]
                                    / (m_NoOfLifts - 1);

   // Allocate the zone to a floor based on how many times the average value
   // is contained in the cumulative value for that floor.
   for ( i= StartFloor; i<= EndFloor; i++) {

     m_User1[ZONE_FOR_FLOOR + i] = 1 + m_User1[CUMULATIVE_POP_DISTANCE + i]
                                  / m_User1[AV_FLOOR_POPULATION];
   }
} // Zone Allocation
```

This routine is going to calculate what zone each floor is a member of. This is done by allocating one element of an array for each floor and allocating it the zone number chosen for that floor. The initial stage of the routine is to initialise this array to zero for each floor in the building. If a prioritised evacuation were to be used the subsequent phases would allocate zones only to the floors involved in the evacuation.

The remaining floors will remain initialised to zero and can be ignored for call allocation during the initial evacuation phase.

The next stage in the routine is to calculate a cumulative floor population x distance. To estimate the amount of traffic involved in evacuating a floor, the product of the number of people on the floor and the distance they have to travel is calculated. A cumulative value is then calculated for each floor, starting at the first floor. This is a total for the current floor and all floors below it. Consequently the top floor holds the total traffic for all the floors under consideration. As each zone is to be served by only one lift, taking this total traffic value and dividing it by the number of lifts can calculate the average traffic for each zone.

The final stage is to make the zone allocation. Again starting at the first floor the cumulative value for that floor is divided by the average traffic value. The integer result, ignoring any remainder is then increased by one to give a zone number. The average value represents the capacity of one zone, while the cumulative value represents the total demand to that level and so the division process gives the number of zones worth of capacity required at that floor. Basing the zone allocation on that value ensures that each zone has roughly the average traffic allocated to it.

6.3.2 Car Allocation

The car allocation process is where the decision about which car to send to a specific floor is made. This decision is based on the dynamics of the situation and is therefore revisited each time the update program is called. All previous allocations are cancelled and the landing-calls are held as primary calls. The allocation process considers each available lift car in turn and reviews all the primary landing-calls. A selection factor is calculated for each of these calls and the call with the highest selection factor has the lift allocated to it. The status of that landing-call is then

changed to that of secondary allocation. The decision about which lift cars are available is based on the current status of the lift car. If the car has its doors closed i.e. it is not loading or unloading and it is not full, the car weight is less then the 24 person maximum of 1800 Kg, then it can be considered for allocation. For most of the evacuation period this process is sufficient to allocate all the lift cars available, however as the evacuation draws to a conclusion, there are more lifts available than there are floors to be served. Rather than let these spare lifts stand idle, a second car allocation is performed. This uses the same process as the primary allocation however the selection factor is based on a different factor, an estimate of the floor with the highest number of people remaining, the Population factor.

6.3.3 Selection Factors

The heart of the allocation process is the calculation of the selection factor. This calculation will be repeated many thousands of times during the calculation and should therefore be as efficient as possible. The main selection factor routine SelFactor combines the results of three selection functions as a weighted sum. However the weighting factors for the duration of this test were set to 1, giving each factor equal significance. The SelFactor routine is shown below, followed by descriptions of each of the selection functions.

```
//-------------------------------------------------------------------------
// Calculate a selection Factor for calling this lift to this floor. Based on
// the set criteria.

double dispatchX::SelFactor(double CurrentTime, lift l[MAX_LIFTS],
                            int iLiftNo,      int iFloor)
{
   iFloor--;

double dFactor =  FillWeight * FillFactor(iFloor) +
                  DurationWeight * DurationFactor(CurrentTime, iFloor) +
                  ZoneWeight * ZoneFactor(iLiftNo,iFloor);
   return dFactor;
}
```

The FillFactor is of significance at the start of the evacuation. In the early stages of an evacuation the number of people waiting may be less than the capacity of the lift. If this is the case then the lift car will not be full on leaving the floor and by implication the floor will be empty. Under these circumstances the floors priority for allocation of a further visit should be down rated for a period of time. The routine below calculates a % factor for how full the lift was the last time it left the floor.

```
//-----------------------------------------------------------------------------
// % of Car Capacity used when leaving this floor last time

double dispatchX::FillFactor(int iFloor)
{
double dFactor = 100 * m_User2[WEIGHT_ON_LEAVING_FLOOR + iFloor]/ WEIGHT_OF_FULL_CAR;
   return dFactor;
}
```

The DurationFactor provides a measure of how long it has been since the floor was last visited. It is calculated as a % of the round trip time to that floor and is aimed at ensuring an even service within a zone. It also works as a way of timing out the depressive effect of the fill factor, ensuring that a revisit is made and that a full lift does eventually leave the floor.

```
//-----------------------------------------------------------------------------
// Time since last pickup as a % of the round trip time
double dispatchX::DurationFactor(double dNow, int iFloor)
{
double dFactor = 100 * (dNow - m_User2[LAST_PICKUP_FROM_FLOOR+iFloor]) /
m_User1[RTT_FOR_FLOOR+iFloor];

   return dFactor;
}
```

The ZoneFactor is the most significant factor during the main part of the evacuation. Its purpose is to provide an even distribution of lifts around the floors to be evacuated.

Unlike conventional zoning we are not attempting to make an absolute allocation of a floor to a zone, we simply provide a degree of membership for a particular floor.

The algorithm starts with a designated floor and lift, we are attempting to find if this floor is to be served by this lift. The first line of the calculation finds the difference between the zone the floor is allocated to and the zone for the lift. A value of 0 indicates the floor is in a zone served by this lift. A value of +1 or –1 indicates the floor is in a zone adjacent to the zone served by the lift. The next step is to normalise the fit value giving the zoning selection factor. If the floor is a member of the lift's zone then the selection factor is 100%, while neighbouring zones score 93% and zones once remove score 75%

$$\textit{Zone Factor} = 100 * ((\textit{No of Zones}-1)^2 - \textit{Value}^2) / (\textit{No of Zones}-1)^2$$

This form of allocation gives the floor a degree of membership to all of the zones however the further away it is from the lift's zone the less likely it is to be served by it. The allocation profile can be seen from figure 2.

```
//---------------------------------------------------------------------------
// Factor for membership of the zone
int dispatchX::ZoneFactor(int iLiftNo, int iFloor)
{
int iValue = m_User1[ZONE_FOR_FLOOR + iFloor] - iLiftNo;
int iFactor = 100 * (16 - iValue*iValue)/16;

   return iFactor;
}
```

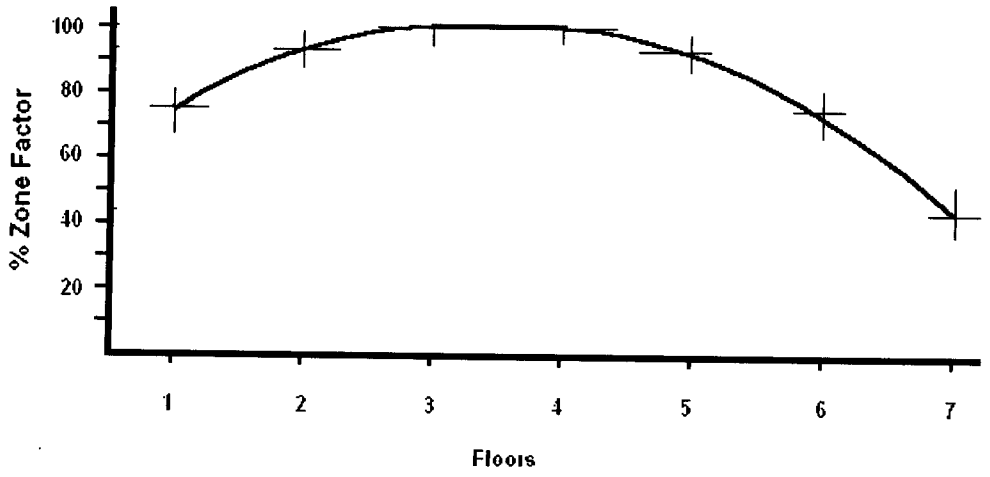

Figure 2 – Zone factor for each floor

Unlike the other factors PopulationFactor is called by the main algorithm, but only when there is a second lift available to visit a floor. This condition can arise towards the end of the evacuation, as the floors are cleared, and is intended to provide a bias for floors likely to have more people on them. It initially works from the designated floor population as a % of the building population, the floor population is however reduced by the lift capacity each time a full lift leaves the floor, so a running estimate is maintained of the number of people remaining at the floor. A floor with the highest estimated people remaining will get allocated a second car if one is available.

```
//-----------------------------------------------------------------------------
// % of Building population possibly remaining at this floor.
int dispatchX::PopulationFactor(int iFloor)
{
int iFactor = 100 * m_User1[POP_OF_FLOOR + iFloor]/ m_User1[POP_OF_BUILDING];

   return iFactor;
}
```

6.3.4 Simulation Results

The evacuation algorithm was compiled and linked as a DLL and run as a custom control, repeating the tests outlined in Section 6.2. Again five runs were performed and the average and standard deviation were calculated. As can be seen from Table 3 the average evacuation time has been reduced by 4 minutes 38 seconds to give **18minutes 53 seconds.**

Seed Value	Evacuation time
1	18 : 58
5	18 : 49
10	18 : 51
15	18 : 58
20	18 : 50
Average	**18 : 53**
Standard Deviation	**4.4**

Table 3 – Evacuation times using the evacuation simulation.

7 Analysis and Conclusions

A simple inspection of the results obtained shows a significant improvement in the evacuation time. Table 4 shows the results obtained, as compared to the Basic Simulation, which is to be used as a base line for the evaluation.

	Basic Simulation	Evacuation Strategy	Theoretic Optimum
Time to evacuate	23:31	18:53	15:19
Improvement on Basic Simulation		4:38	8:12
% Improvement		20%	35%
Standard Deviation	28.3	4.4	

Table 4 – Comparison of simulation results.

From these figures it can be seen that the Evacuation strategy has produced a 20% improvement on the basic algorithm supplied with the simulation software. This is 3 minutes 34 seconds short of the optimum evacuation time, but still a significant improvement. The other significant factor to be drawn from these test results is the value of standard deviation. This factor is a measure of the variability in evacuation time that arises from different arrival patterns. It shows clearly that the Evacuation strategy is much less susceptible to variation in the arrival pattern and therefore gives a more predictable outcome.

While the objective data of evacuation time, gives a measure of the improvement seen, it was also possible to make a number of observations on the causes of this improvement. From simply watching the lift animation screen it is possible to infer

what may be happening. Figure 3 shows a snapshot of the Basic simulation just over 9 minutes into the first run, with a seed value of 1.

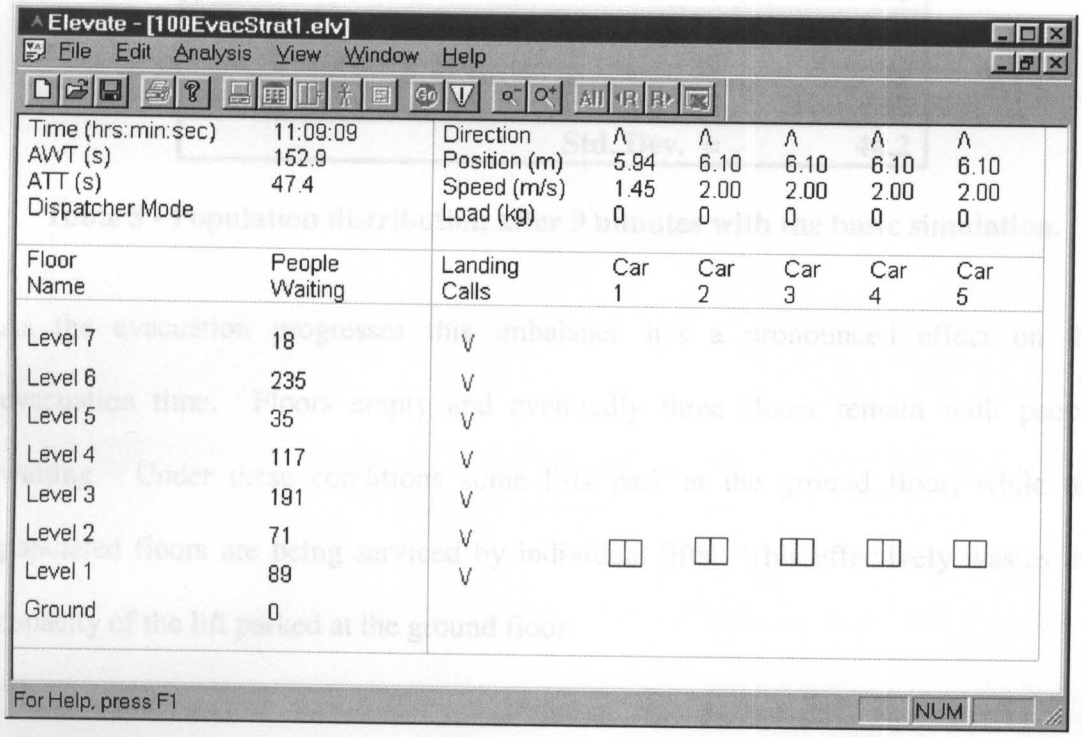

Figure 3 – Basic simulation after 9 minutes

As can be seen from this screen snapshot, the effect of bunching is pronounced and has been for a significant period up to this point. The evacuation pattern has been very repetitive, with the same floors being serviced at the expense of others. This can be seen from the pronounced difference in the number of people waiting at the different floors. Table 5 shows how uneven the evacuation has been so far.

Floor	Initial Population	Population after 9minutes	People Evacuated
7	138	18	120
6	259	235	24
5	131	35	96
4	237	117	120
3	287	191	96
2	191	71	120
1	110	89	21
Total	**1,353**	**765**	**597**
		Std. Dev. =	**44.2**

Table 5 - Population distribution after 9 minutes with the basic simulation.

As the evacuation progresses this imbalance has a pronounced effect on the evacuation time. Floors empty and eventually three floors remain with people waiting. Under these conditions some lifts park at the ground floor, while the populated floors are being serviced by individual lifts. This effectively wastes the capacity of the lift parked at the ground floor.

For the Evacuation Strategy the picture is somewhat different. Figure 4 shows a screen snapshot at approximately the same point in time for the equivalent run with the evacuation algorithm.

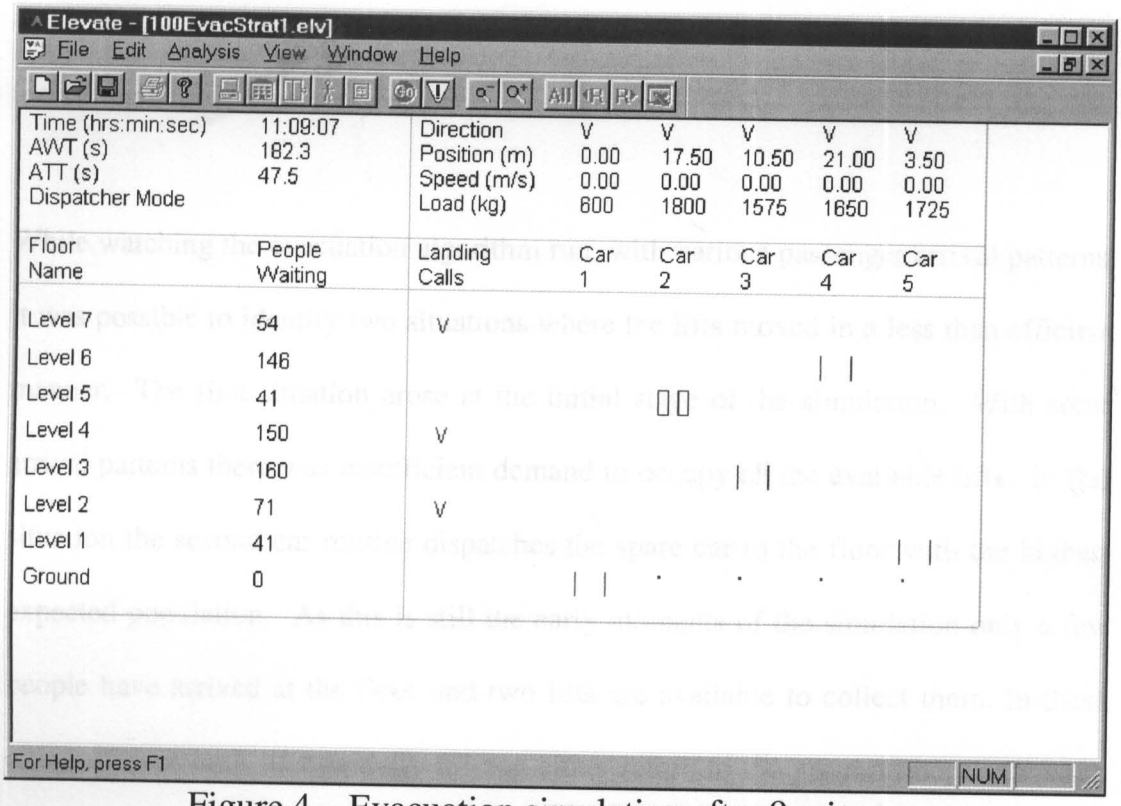

Figure 4 – Evacuation simulation after 9 minutes

Here we can see no evidence of bunching and that has been apparent from the start of the simulation. It is also apparent that the floor populations are not as badly out of balance as for the previous case. If we calculate the number of people evacuated we have the following results.

Floor	Initial Population	Population after 9minutes	People Evacuated
7	138	54	84
6	259	146	113
5	131	41	90
4	237	150	87
3	287	160	127
2	191	71	120
1	110	41	69
Total	**1,353**	**672**	**690**
	0	**Std. Dev. =**	**21.5**

Table 6 - Population distribution after 9 minutes with the evacuation simulation.

Here we see some variation in the numbers evacuated from each floor, but the variation is not as marked. Comparing the standard deviation for the numbers

evacuated, it is seen to be just less than half the value of the Basic Simulation, indicating a more even service.

While watching the evacuation algorithm run, with various passenger arrival patterns, it was possible to identify two situations where the lifts moved in a less than efficient manner. The first situation arose at the initial stage of the simulation. With some arrival patterns there was insufficient demand to occupy all the available lifts. In this situation the second car routine dispatches the spare car to the floor with the highest expected population. As this is still the early moments of the simulation only a few people have arrived at the floor and two lifts are available to collect them. In these circumstances both lifts partially fill and either return to the ground floor, or have to make a second stop to collect passengers from an intermediate floor. In either case a small amount of capacity has been lost from the system. The second situation also arises when the lift car partially fills. Because of the way the simulator operates the lift car is only permitted to travel down the building when passengers are onboard. It has been seen on occasions where a lift only partially full ignored a landing-call on a floor immediately above it and the call had to be serviced by a lift travelling from the ground floor.

In conclusion a 20% improvement has been achieved with this dedicated evacuation algorithm and it has a more consistent operation when faced with different passenger arrival patterns. From a theoretic assessment of the situation it is possible to say that further improvement should be possible, but this improvement could only be of the order of 3 minutes 34 seconds. It is felt that the benefits seen stems mainly from improving the balance of service to the individual floors and this may be an area to develop if further improvements are to be achieved.

If we consider 9 minutes as being a reasonable response time for the emergency services to respond to the building. It can be seen that allowing the lift system to operate in automatic evacuation mode has allowed 690 people, over half the building population, to leave the building before the emergency services arrive to provide supervised evacuation. This is a strong argument for allowing a suitably designed lift system to operate automatically during an emergency situation. It however remains to be seen if this added benefit will be recognised within building codes of practice.

8 Future Work

The results described in the previous section should only be seen as an encouraging first step in the development of a specialised evacuation control algorithm. So far the algorithm has only been tested for one specific building layout, with an everybody-out strategy. As such any benefits seen can only safely be said to apply to a building of that particular size. The next step should therefore be to perform a series of further tests to investigate the effectiveness of the algorithm under different operating conditions. There are several areas to consider; the building construction needs to be varied in terms of the height, population and lift performance, here the use of actual building examples would be of great assistance, to ensure realistic scenarios are tested. In addition to this the evacuation strategy should also be considered, for our test we opted for an everybody-out approach, however the algorithm is capable of operating over a restricted number of floors, to give a prioritised evacuation. Its performance in this area has not yet been evaluated and as such is another area for attention.

A third and less obvious area is the assumption made about building and floor populations. The tests performed so far used the building design populations for calculating the zoning requirements and for the second car allocation. These values were also used to configure the simulator, so in effect we presented the algorithm with an idealised situation. In a real building it is unlikely the design population and the actual population would be the same; visitors come into the building, the staff take holidays etc. It would therefore be of some interest to measure the effect of varying simulated floor populations, away for the design norm.

A further area of investigation, that had been envisaged, is to try and optimise the car selection factors. Provision was made in the original design to weight each of the individual selection factors. For the purpose of these tests however, these weighting factors were given a common value of 1 and hence had no effect on the selection process. The application of different weighting factors could be investigated to see if a critical factor could be identified and enhanced within the car selection process.

The final area to be considered, is the decision to use the Basic Simulation as a baseline to measure improvement from. This form of control is a general-purpose nearest car strategy, which was selected for the purely practical reason that it was available. A more considered approach would suggest that an algorithm designed to cope with a normal down peak traffic flow, would be a better benchmark to judge the evacuation algorithm against.

Appendix A

Best Evacuation time calculation

Floor Level	Floor Population	Car Size	No of Full Journeys	People Remaining	Time to Load/unload People Remaining	Single Person Round trip time	Time to fill / empty the car	Full Car Round trip time	Time for all Full Car journeys I	Journey time for people remaining
		24				Sec	Sec	Sec	Sec	Sec
Level 7	138	24	5	18	43.20	38.50	55.20	93.70	468.50	86.35
Level 6	259	24	10	19	45.60	35.00	55.20	90.20	902.00	
Level 5	131	24	5	11	26.40	31.50	55.20	86.70	433.50	57.90
Level 4	237	24	9	21	50.40	28.00	55.20	83.20	748.80	70.90
Level 3	287	24	11	23	55.20	24.50	55.20	79.70	876.70	
Level 2	191	24	7	23	55.20	21.00	55.20	76.20	533.40	
Level 1	110	24	4	14	33.60	17.60	55.20	72.80	291.20	51.20
Ground										
Total	1,353		51						4,254.10	4.44
									70.90	Min(Total)
									14.18	Min (Avr)
						2.4	Sec (Time to load & unload 1 person)			

	Lift 1	Lift 2	Lift3	Lift 4	Lift 5	
1	93.70	90.20	90.20	83.20	83.20	
2	93.70	90.20	90.20	83.20	83.20	
3	93.70	90.20	90.20	83.20	83.20	
4	93.70	90.20	90.20	83.20	83.20	
5	93.70	86.70	90.20	83.20	79.70	
6	86.70	86.70	79.70	79.70	79.70	
7	76.20	86.70	76.20	79.70	79.70	
8	76.20	76.20	76.20	79.70	79.70	
9	72.80	76.20	72.80	79.70	79.70	
10	70.90	72.80	86.35	79.70	76.20	
11	51.20	72.80	57.90	86.70	90.20	
Sum (Sec)	902.50	918.90	900.15	901.20	897.70	4,520.45
(Minutes)	15.04	15.32	15.00	15.02	14.96	

	Lift 1	Lift 2	Lift3	Lift 4	Lift 5	
Average	904.09	904.09	904.09	904.09	904.09	
Variance	-1.59	14.81	-3.94	-2.89	-6.39	0.00
Var Sq	2.53	219.34	15.52	8.35	40.83	16.93

Appendix B

Standard control algorithm results.

ELEVATE
Version 4.0
© Peters Research Ltd. 2001
JOB DATA

Job Title	Internal Group Collective
Job No	
Calculation Title	
Made By	
File	100EvacStrat1.elv
Date	17-Nov-02

ANALYSIS DATA

Analysis Type	Simulation
Measurement system	Metric
Dispatcher Algorithm	Group Collective
Time slice between simulation calculations (s)	0.1
No of time slices between screen updates	1
Random number seed for passenger generator	1
Start at	11:00:00
End at	11:23:26

BUILDING DATA

Floor Name	Floor Height (m)
Ground	3.5
Level 1	3.5
Level 2	3.5
Level 3	3.5
Level 4	3.5
Level 5	3.5
Level 6	3.5
Level 7	

ELEVATOR DATA

No of Elevators	5
Capacity (kg)	1800
Door Pre-opening Time (s)	0
Door Open Time (s)	1.8
Door Close Time (s)	2.9
Speed (m/s)	2
Acceleration (m/s^2)	1
Jerk (m/s^3)	2
Motor Start Delay (s)	0.5
Home Floor	Ground

PASSENGER DATA (Period 1)

Start Time	11:00
End Time	11:05
Loading Time (s)	1.2
Unloading Time (s)	1.2
Passenger Mass (kg)	75
Capacity Factor (%)	100
Stair Factor (%)	0

Evacuation algorithm results.

ELEVATE

Version 4.0

JOB DATA

Job Title	Evacuation Routine
Job No	
Calculation Title	
Made By	
File	100EvacStrat1.elv
Date	20-Nov-02

ANALYSIS DATA

Analysis Type	Simulation
Measurement system	Metric
Dispatcher Algorithm	+2nd car to Populous floor
Time slice between simulation calculations (s)	0.1
No of time slices between screen updates	1
Random number seed for passenger generator	1
Start at	11:00:00
End at	11:18:58

Appendix C

Evacuation Control Algorithm.

```
/*
--------------------------------------------------------------------------------
Filename    xDISPATCH.CPP

Purpose     Implementation of dispatchX class
      For the testing of an Evacuation stratergy.

Version        8/10/2002

--------------------------------------------------------------------------------
*/

#include "stdafx.h"
#include "arraysize.h"
#include "xdispatch.h"
#include "math.h"
#include "FLOAT.H"

#define max(a, b)  (((a) > (b)) ? (a) : (b))


// Index values into the User1 array where these values are held.
#define POP_OF_BUILDING           10
#define POP_OF_FLOOR              10
#define ZONE_FOR_FLOOR            40

#define LAST_PICKUP_FROM_FLOOR    60
#define WEIGHT_ON_LEAVING_FLOOR   80
#define RTT_FOR_FLOOR            100

#define POP_DISTANCE       120   // For each floor the number of people * how
                                 //far they have to travel
#define CUMULATIVE_POP_DISTANCE 140  // Cumulative value of above.
#define FLOOR_HEIGHT          160  // Floor Height

#define WEIGHT_OF_FULL_CAR   1400..// Consider lifts above this weight to be full(Kg)
#define AV_FLOOR_POPULATION          1401 // Average floor population.
#define TopFloor         7  // Highest floor


//FILE * fDebug;

//constructor
dispatchX::dispatchX() {}

//member functions

//--------------------------------------------------------------------------
// This routine is called before the simulation starts

void dispatchX::Reset(building b)    {

//FILE * fDebug = fopen("debug.txt","a+");
//fprintf(fDebug, "This is in Reset routine  \n");



   m_NoOfAlgorithms=1;  //enter here the number of algorithms you have defined
            //and want to be accessed by the user from Elevate

   m_AlgorithmName[0]="2nd car to Populus floor";  // Identify the routine


   for (int ctr=1;ctr<=b.m_NoFloors;ctr++) //this ensures there are no left
                                                  // over calls
   {                 //registered with the dispatcher when the
      m_UpLandingCalls[ctr]=0;         //simulation commences
      m_DownLandingCalls[ctr]=0;
   }

   for (ctr=0;ctr<1000;ctr++)          //reset the user variables
```

```
    {
       m_User1[ctr]=0;
       m_User2[ctr]=0;
    }

    // Allocate Floor Population.
    m_User1[POP_OF_FLOOR + 1] = 110;
    m_User1[POP_OF_FLOOR + 2] = 191;
    m_User1[POP_OF_FLOOR + 3] = 287;
    m_User1[POP_OF_FLOOR + 4] = 237;
    m_User1[POP_OF_FLOOR + 5] = 131;
    m_User1[POP_OF_FLOOR + 6] = 259;
    m_User1[POP_OF_FLOOR + 7] = 138;

    for (int i=1; i<= TopFloor; i++) {
       m_User1[POP_OF_BUILDING] = m_User1[POP_OF_BUILDING] +
                   m_User1[POP_OF_FLOOR + i];
    }


    // Floor Height
    m_User1[FLOOR_HEIGHT  + 1] =  35;
    m_User1[FLOOR_HEIGHT  + 2] =  70;
    m_User1[FLOOR_HEIGHT  + 3] = 105;
    m_User1[FLOOR_HEIGHT  + 4] = 140;
    m_User1[FLOOR_HEIGHT  + 5] = 175;
    m_User1[FLOOR_HEIGHT  + 6] = 210;
    m_User1[FLOOR_HEIGHT  + 7] = 245;

    ZoneAllocation(1,TopFloor);

// fclose(fDebug);
    return;
} //Reset



//---------------------------------------------------------------------------
// This routine contains the dispatching algorithm(s),
// and is called every time slice (0.1 to
// 0.01 seconds as entered by the user)


void dispatchX::Update(double CurrentTime,lift l[MAX_LIFTS],double SimulationTimeStep,
    building b,int NoLifts, CString message, CString &mode)

{

    //call the algorithm selected by the user
    switch(m_Algorithm)
    {
       case 0:
          Algorithm0(CurrentTime,l,SimulationTimeStep,b,NoLifts,message,mode);
          break;
       case 1:
          Algorithm1(CurrentTime,l,SimulationTimeStep,b,NoLifts,message,mode);
          break;
       case 2:
          Algorithm2(CurrentTime,l,SimulationTimeStep,b,NoLifts,message,mode);
          break;
       case 3:
          Algorithm3(CurrentTime,l,SimulationTimeStep,b,NoLifts,message,mode);
          break;
       case 4:
          Algorithm4(CurrentTime,l,SimulationTimeStep,b,NoLifts,message,mode);
          break;

    }
```

```
//--------------------------------------------------------------------------

void dispatchX::Algorithm0(double CurrentTime,lift l[MAX_LIFTS], double SimulationTimeStep,
              building b,int NoLifts,CString message,CString &mode)
{   // Algorithem 0

int iLiftNo;
int iFloor;
int iBestFloor;
int iDemandSeen;
double dBestFactor;



   for (int i=1; i<= b.m_NoFloors; i++)
   {
      for (int LiftNo=1; LiftNo<=NoLifts; LiftNo++)
      {
                if (l[LiftNo].m_UpLandingCalls[i]==1    && i!=l[LiftNo].m_DestinationFloor)
        {
           m_UpLandingCalls[i]=1;
           l[LiftNo].m_UpLandingCalls[i]=0;
        }

                if (l[LiftNo].m_DownLandingCalls[i]==1 && i!=l[LiftNo].m_DestinationFloor)
        {
           m_DownLandingCalls[i]=1;
           l[LiftNo].m_DownLandingCalls[i]=0;
        }
      }
   }

     // For Each lift car find a floor to send it to
   for ( iLiftNo=1; iLiftNo<=NoLifts; iLiftNo++) {

      //  If we have reached the destination floor cancel the call
      if ( l[iLiftNo].m_DoorStatus == OPENNING  &&
          l[iLiftNo].m_DestinationFloor == l[iLiftNo].FloorAt() ) {

         l[iLiftNo].m_UpLandingCalls   [l[iLiftNo].FloorAt()] = 0;
         l[iLiftNo].m_DownLandingCalls[l[iLiftNo].FloorAt()] = 0;
         l[iLiftNo].m_Direction = DOWN;
      }

      // Now chose the best floor to send this lift to
      iBestFloor = -1;
      dBestFactor = -1;
      iDemandSeen = 0;
      for ( iFloor=1; iFloor <= b.m_NoFloors; iFloor++ ) {

         double FloorFactor = -1;

         // If there is a demand on this floor
         if ( m_UpLandingCalls  [iFloor] > 0 ||
             m_DownLandingCalls[iFloor] > 0 ) {
            iDemandSeen++;

            // If the lift can collect passengers
            if( l[iLiftNo].m_DoorStatus == CLOSED &&
               l[iLiftNo].m_CurrentLoad < WEIGHT_OF_FULL_CAR ) {

// Can the lift collect from this floor.  If so find out how important it is
               if ( l[iLiftNo].m_Direction == UP &&
                  l[iLiftNo].QuickestFloorStopFloor() <= iFloor )
                  FloorFactor = SelFactor (CurrentTime,l,iLiftNo,iFloor);

               if ( l[iLiftNo].m_Direction == DOWN &&
                  l[iLiftNo].QuickestFloorStopFloor() >= iFloor ) {
                  FloorFactor = SelFactor (CurrentTime,l,iLiftNo,iFloor);
               }

               if ( l[iLiftNo].m_Direction == NONE &&
                   l[iLiftNo].m_CurrentLoad == 0 )
                     FloorFactor = SelFactor (CurrentTime,l,iLiftNo,iFloor);


               // See if this floor is more important than the others so far
```

```
            if ( FloorFactor > dBestFactor ) {
               dBestFactor = FloorFactor;
               iBestFloor = iFloor;

            } // new best factor

         } // if lift available for allocation.
      } // If Landing Call
   } // For Each floor


   // If a best floor chosen then allocate a lift to it. & set the landing
   // demand as a possible candidate for a second car visit.
   if ( iBestFloor > 0 ) {
      l[iLiftNo].m_DownLandingCalls[iBestFloor] = 1;
      m_UpLandingCalls[iBestFloor] = 0;
      m_DownLandingCalls[iBestFloor] = -1;
      l[iLiftNo].Update(CurrentTime);
   } // if best floor found


//  2nd Car Allocation
    // No unallocated landing calls available for this lift, so
    // check back over any allocated demand to find which floor
    // would be served best by a second car visiting it.

    if ( iDemandSeen == 0 ) { // If no primary demand 2nd car allocation

       for ( iFloor=1; iFloor <= b.m_NoFloors; iFloor++ ) {
       double FloorFactor = -1;

          if ( m_DownLandingCalls[iFloor] == -1 ) {

             // If the lift can collect passengers
             if( l[iLiftNo].m_DoorStatus == CLOSED &&
                l[iLiftNo].m_CurrentLoad < WEIGHT_OF_FULL_CAR ) {

    // Can the lift collect from this floor.  If so find out how important it is
                if ( l[iLiftNo].m_Direction == UP &&
                    l[iLiftNo].QuickestFloorStopFloor() <= iFloor )
                      FloorFactor = PopulationFactor (iFloor);

                if ( l[iLiftNo].m_Direction == DOWN &&
                   l[iLiftNo].QuickestFloorStopFloor() >= iFloor ) {
                      FloorFactor = PopulationFactor (iFloor);
                }

                if ( l[iLiftNo].m_Direction == NONE &&
                    l[iLiftNo].m_CurrentLoad == 0 )
                      FloorFactor = PopulationFactor (iFloor);


                // See if this floor is more important than the others so far
                if ( FloorFactor > dBestFactor ) {
                   dBestFactor = FloorFactor;
                   iBestFloor = iFloor;

                } // new best factor
             } // if the lift can collect passengers.
          } // Down Landing Call = -1   2nd car allocation
       } // For each floor again

       // If a floor chosen send a second lift to it
       if ( iBestFloor > 0 ) {
          l[iLiftNo].m_DownLandingCalls[iBestFloor] = 1;
          m_UpLandingCalls[iBestFloor] = 0;
          m_DownLandingCalls[iBestFloor] = 0;
          l[iLiftNo].Update(CurrentTime);
       } // if best floor found

    } // No primary demand seen
 } // For Each lift


 // Clean up.
 for ( iFloor=1; iFloor <= b.m_NoFloors; iFloor++ ) {
       if ( m_DownLandingCalls[iFloor] == -1 ) {
          m_DownLandingCalls[iFloor] = 0;
```

```
        }
    }

    return;

}   // Algorithem 0


//--------------------------------------------------------------------------

void dispatchX::Algorithm1(double CurrentTime,lift l[MAX_LIFTS], double SimulationTimeStep,
                  building b,int NoLifts,CString message,CString &mode)
{   // Algorithem 1

    return;
}   // Algorithem 1



//--------------------------------------------------------------------------

void dispatchX::Algorithm2(double CurrentTime,lift l[MAX_LIFTS], double SimulationTimeStep,
                  building b,int NoLifts,CString message,CString &mode)
{   // Algorithem 2

    return;
}   // Algorithem 2

//--------------------------------------------------------------------------
void dispatchX::Algorithm3(double CurrentTime,lift l[MAX_LIFTS], double SimulationTimeStep,
                  building b,int NoLifts,CString message,CString &mode)
{  // Algorithem 3

    return;

}  // Algorithem 3

//--------------------------------------------------------------------------
void dispatchX::Algorithm4(double CurrentTime,lift l[MAX_LIFTS], double SimulationTimeStep,
                  building b,int NoLifts, CString message,CString &mode)
{  // Algorithem 4

}  // Algorithem 4



//--------------------------------------------------------------------------
// Cancel Landing calls for a lift when it arrives at floor.
// Skip canceling Landing coall alocation to other lifts.

void dispatchX::CancelLandingCalls(lift l[MAX_LIFTS],int NoLifts)
        //This function is called every time slice
        //in order to cancel landing calls that have
        //been answered.
        //We recommend that you do not edit this
        //function.

{  // CancelLandingCalls


    for (int ctr=1; ctr<=NoLifts; ctr++)
    {
    int i=l[ctr].FloorAt();  //find out what floor lift is at

       if (l[ctr].m_TravelStatus==0  //not travelling and doors closed/closing
       && (l[ctr].m_DoorStatus==CLOSED||l[ctr].m_DoorStatus==CLOSING))
       {
       m_User2[WEIGHT_ON_LEAVING_FLOOR + i] = l[ctr].m_CurrentLoad;
       m_User2[LAST_PICKUP_FROM_FLOOR  + i] = l[ctr].m_CurrentTime;
       } // Stopped & Doors Closing


       if (l[ctr].m_TravelStatus==0  //not travelling and doors open/openning
       && (l[ctr].m_DoorStatus==OPEN || l[ctr].m_DoorStatus==OPENNING))
       {
          if (l[ctr].m_Direction==UP) //travelling up
          {
             for (int ctr2=1;ctr2<=NoLifts;ctr2++) //cancel calls
```

```
                l[ctr2].m_UpLandingCalls[i]=0;
            m_UpLandingCalls[i]=0;

         }

         if (l[ctr].m_Direction==DOWN)  //travelling down
         {
            for (int ctr2=1;ctr2<=NoLifts;ctr2++)
               l[ctr2].m_DownLandingCalls[i]=0;
            m_DownLandingCalls[i]=0;

         }

         if (l[ctr].m_Direction==NONE) //no direction
         {
            //up call at landing
            for (int ctr3=1;ctr3<=NoLifts;ctr3++)
            if (l[ctr3].m_UpLandingCalls[i]==1||m_UpLandingCalls[i]==1)
            {
               for (int ctr2=1;ctr2<=NoLifts;ctr2++)
                  l[ctr2].m_UpLandingCalls[i]=0;
               m_UpLandingCalls[i]=0;

               l[ctr].m_Direction=1;
            }

            //down call at landing
            for (ctr3=1;ctr3<=NoLifts;ctr3++)
            if (l[ctr3].m_DownLandingCalls[i]==1||m_DownLandingCalls[i]==1)
            {
               for (int ctr2=1;ctr2<=NoLifts;ctr2++)
                  l[ctr2].m_DownLandingCalls[i]=0;
               m_DownLandingCalls[i]=0;

               l[ctr].m_Direction=-1;
            }
         }
      }
   } // For each Lift
} // CancelLandingCalls


//      End of Sceleton code.
//##############################################################################

//------------------------------------------------------------------------------
// Calculate a selection Factor for calling this lift to this floor. Based on
// the set criteria.

double dispatchX::SelFactor(double CurrentTime, lift l[MAX_LIFTS],
                            int iLiftNo,      int iFloor)
{
   iFloor--;

double dFactor =  FillWeight * FillFactor(iFloor) +
                  DurationWeight * DurationFactor(CurrentTime, iFloor) +
                  ZoneWeight * ZoneFactor(iLiftNo,iFloor);

   return dFactor;
}


//------------------------------------------------------------------------------
// % of Building population possibly remaining at this floor.
int dispatchX::PopulationFactor(int iFloor)
{
int iFactor = 100 * m_User1[POP_OF_FLOOR + iFloor]/ m_User1[POP_OF_BUILDING];

   return iFactor;
}

//------------------------------------------------------------------------------
// Factor for membership of the zone
int dispatchX::ZoneFactor(int iLiftNo, int iFloor)
{
int iValue = m_User1[ZONE_FOR_FLOOR + iFloor] - iLiftNo;
int iFactor = 100 * (16 - iValue*iValue)/16;
```

```
   return iFactor;
}

//-------------------------------------------------------------------------
// Time since last pickup as a % of the round trip time
double dispatchX::DurationFactor(double dNow, int iFloor)
{
double dFactor = 100 * (dNow - m_User2[LAST_PICKUP_FROM_FLOOR+iFloor]) / m_User1[RTT_FOR_FLOOR+iFloor];

   return dFactor;
}



//-------------------------------------------------------------------------
// % of Car Capacity used when leaving this floor last time

double dispatchX::FillFactor(int iFloor)
{
double dFactor = 100 * m_User2[WEIGHT_ON_LEAVING_FLOOR + iFloor]/ WEIGHT_OF_FULL_CAR;
   return dFactor;
}


//-------------------------------------------------------------------------
// Allocate a zone number to each floor level

void dispatchX::ZoneAllocation(int StartFloor, int EndFloor)
{
int i;

   // Initialise all floors
   for ( i=1; i<= TopFloor; i++) {

      m_User1[ZONE_FOR_FLOOR + i] = 0;

   } // for i <= TopFloor

   // Cumulative Floor Population Distance.
   m_User1[CUMULATIVE_POP_DISTANCE] = 0;
   for ( i= StartFloor; i<= EndFloor; i++) {

      m_User1[POP_DISTANCE + i] = m_User1[POP_OF_FLOOR + i] *
                                  m_User1[FLOOR_HEIGHT + i] ;

      m_User1[CUMULATIVE_POP_DISTANCE + i] = m_User1[POP_DISTANCE + i] +
                              m_User1[CUMULATIVE_POP_DISTANCE + i - 1];

   } // for i <= EndFloor

   // Calculate an average population to be moved by each lift.
   m_User1[AV_FLOOR_POPULATION] = m_User1[CUMULATIVE_POP_DISTANCE + TopFloor]
                                   / (m_NoOfLifts - 1);

   // Allocate the zone to a floor based on how many times the average value
   // is containd in the cumlative value for that floor.
   for ( i= StartFloor; i<= EndFloor; i++) {

     m_User1[ZONE_FOR_FLOOR + i] = 1 + m_User1[CUMULATIVE_POP_DISTANCE + i]
                                / m_User1[AV_FLOOR_POPULATION];
   }
} // Zone Allocation
```

References

Al-Sharif L.R. (1995). Bunching in Lift systems
Elevator Technology 5 -Proceedings of ELEVCON 93 (pp 88-97) IAEE

Barney, G.C. & dos Santos, S.M. (1985) Elevator Traffic, Analysis, Design and Control
Peter Peregrinus, London

Barney, G.C. (2003) Elevator Traffic Handbook, Theory and Practice
Spon Press. Taylor & Francis Group. London

BS5588 Fire precautions in the design, construction and use of buildings: Part 5 (1991)
Code of practice for firefighting stairs and lifts

BS5588 Fire precautions in the design, construction and use of buildings: Part 8 (1991)
Code of practice for evacuation of disabled people

Chan, W.L. & So, A.T.P (1995) Dynamic zoning in elevator traffic control
Elevator Technology 6 - Proceedings of ELEVCON 95 (pp 132- 140) IAEE

Favro, P.C (1997) Elevators as part of a building evacuation scheme
Elevator World, March 1997

Guylene, P. (2002) Evacuation Planning for Occupants with Disability
Fire Risk Management Program, Internal Report No 843
Institute for Research in Construction Retrieved 2nd September 2002, from
www.cnrc.ca/irc/fulltext/ir843/ir843.pdf

Howkins, R.E. (2000) In the Event of Fire – Use the Elevators
Elevator Technology 10, (pp 11-21) IAEE

Howkins, R.E. (2002) Elevators for emergency evacuation and egress
Elevator World, February 2002 (p124 – 121)

Peters,R.D. (2001) Getting started with Elevate [User Manual]
Peters Research www.peters-reserch.com

Poschel T & Gallas J.A.C. (1994) Synchronization effects in the dynamical behaviour of elevators. The American Physical Society. Physical Review E. Volume 50 No4 (pp 2654-2659)

Powell B.A (1992) Important issues in up peak traffic handling
Elevator Technology 4 - Proceedings of ELEVCON 92 (pp 207- 218) IAEE

Qun Z,. Zen-Shi W. & Xiao-guang, S. (2001) A direct-search approach to Dynamic Zoning optimization – Problem of EGCS Elevator World, February 2001

Qun Z, Xin-Yu L. & Zen-Shi, W. (2002) The research dynamic programming zoning control method in elevator up-peak Mode Elevator World, June 2002

So, A.T.P & Chan, W.L. (1997) Comprehensive dynamic zoning algorithms
Elevator World, September 1997