Generation and use of sparse navigation graphs for microscopic pedestrian simulation models

Angelika Kneidl[1], André Borrmann[1], Dirk Hartmann[2]
[1] Computational Modelling and Simulation Group, Technische Universität München, Germany
[2]Siemens AG, Corporate Technology, Germany
kneidl@tum.de

Abstract. For the spatial design of buildings as well as for the layout of large event areas, the crowd behaviour of the future users plays a significant role. The designing engineer has to make sure that potentially critical situations, such as high densities in pedestrian crowds, are avoided in order to guarantee the integrity, safety and comfort of the users. To this end, computational pedestrian dynamics simulations have been developed and are increasingly used in practice. However, most of the available simulation systems rely on rather simple pedestrian navigation models, which reflect human behaviour only in a limited manner. This paper contributes to enhancing pedestrian simulation models by extending a microscopic model by a navigation graph layer serving as a basis for different routing algorithms. The paper presents an advanced method for the automated generation of a spatially embedded graph which is on the one hand as sparse as possible and on the other hand detailed enough to be able to serve as a navigation basis. Three different pedestrian types were modelled: pedestrians with good local knowledge, pedestrians with partly local knowledge and those without any local knowledge. The corresponding algorithms are discussed in detail. To illustrate how this approach improves on simulation results, an example scenario is presented to demonstrate the difference between results with and without using a graph as constructed here. Another example shows the application of the extended simulation in a real-world engineering context. The article concludes with an outlook of further potential application areas for such navigation graphs.

Keywords: navigation graph, visibility graph, microscopic pedestrian simulation, A* Algorithm, cellular automaton.

1 Introduction

For the spatial design of buildings as well as for the layout of large event areas, the crowd behaviour of the future users plays a very important role. The designing engineer has to make sure that potentially critical situations, such as high densities of pedestrian crowds, are avoided in order to guarantee the integrity, safety and comfort of the users. In today's engineering practice, rough approximate calculations are used to determine the space required by pedestrian streams. However, these methods are neither able to capture the precise geometric setup of the investigated scenario nor can they consider the complex way-finding and walking behaviour of individual pedestrians. Accordingly, local phenomena are disregarded and potentially critical situations are easily ignored. To overcome these shortcomings, computational pedestrian dynamics simulations have been developed and are increasingly used in practice.

However, most of the available simulation systems systems either rely on rather simple pedestrian navigation models, which reflect real human behavior only in a very limited manner, or are computationally expensive. This paper contributes to enhancing computationally cheap pedestrian simulation models by presenting a sophisticated graph-based approach for modelling navigational behaviour of humans. This allows engineers and architects to quickly and effortlessly evaluate different layout options. The implementation of this approach includes an advanced technique for generating sparse navigation graphs from a given spatial layout of the scenario under investigation.

2 Related work

The simulation of pedestrian crowds has been widely examined using a variety of approaches that focus on different details depending on the objective of the simulation [1]. For example, to determine minimum evacuation times for buildings or areas, macroscopic models are typically used. These focus on the overall situations of the simulated scenario and are based on mean values. Examples of such models are network flow models [2], fluid-dynamic models [3] or gas kinetic models [4]. To simulate the individual behaviour of pedestrians on the other hand, microscopic models have been developed. These models consider the movements of each individual and focus on the interaction between individuals. Force models (e.g. Social Force Model by Helbing and Molnár [5]) as well as cellular automata [6] or agent-based models [7] belong to this category.

One central aspect of microscopic pedestrian simulation is to simulate the different movement strategies of individuals. Pelechano and Malkawi [8] categorize “virtual human technologies” into different features, such as appearance, function, time, autonomy and individuality.

The focus of this contribution lies on the latter: to differ between individual behaviour as a factor of sex and age, and – the authors’ main focus – sense of orientation and familiarity with a location. The aim is to simulate large pedestrian crowds while taking into account different movement behaviours. An important constraint considered for the development of the corresponding algorithms is the requirement of high computational performance which allows for real-time simulations even on standard hardware. This provides the possibility to use the simulator as training facility for preparing and training the security staff of major events. – a feature strongly demanded by security authorities.

In order to assign individual behaviour to pedestrians, agent-based models are common. These assign different behavioural patterns to each individual, which results in different movement behaviour. Reynolds [9] models the perception of individuals with three different layers, namely a locomotion layer, a steering layer and an action selection layer. Musse and Thalmann [10] developed a human crowd behaviour model, consisting of a random behavioural model, which can be described by a few parameters. In [11], a personality model is mapped into a simulation model. Taking this a step further, Lerner [12] uses tracking from video data to obtain possible movements and trajectories. As these models have to calculate the new position of each pedestrian according to a complex set of rules in every time step, they are very computationally intensive and are capable of simulating only few pedestrians in real time. Another, faster way to assign individual behaviour is to use a navigation graph with different routing algorithms according to the individuals’ preference. Since the objective is to simulate a large crowd in a large area in real time, the latter approach has been chosen.

Combining a microscopic layer with such graphs or networks was proposed by [13]. Here, a continuous microscopic model is used as operational model, i.e. to model the microscopic pedestrians’ movement, in combination with a tactical model implemented as a network, for pedestrians’ route assignment. The network consists of uniform square cells, which are connected by links. [14] combines an agent-based approach with a macroscopic network. In spite of taking cost functions and optimizing the flow, agents move through this network choosing the next vertex based on different criteria. The authors call this algorithm route choice self organisation (RCSO). However, both approaches do not focus on the derivation of a graph from a given geometry but take either such a network as given or simply divide space into uniform squares, the latter resulting in unrealistic wayfinding behaviour.

In contrast, this paper describes a technique for generating navigation graphs based on navigation points, which precisely reflect human navigational behaviour. At the same time, the graph consists of a minimum number of edges and vertices, enabling a high computational efficiency of the corresponding navigation algorithms.

A variety of alternative techniques have been proposed to create a navigation graph or roadmap from a given topography. Most of these techniques have been developed in the field of Robotics. [15] gives a good overview of the most common techniques of space decomposition. [16] describes all kind of planning algorithms, including motion planning algorithms. One technique for deriving a roadmap is to divide the space with Generalized Voronoi Diagrams [17] and to use the resulting lines as graph edges and the intersection points of the lines as graph nodes. The resulting graph consists of edges which are equidistant to each obstacle. A similar approach has been proposed in [18]: Here, agents navigate along combined Voronoi diagrams, which include not only obstacles but other moving agents as well. The intersection of the regions of the first order Voronoi diagram with the second order Voronoi diagram forms the navigation graph. The authors call this graph Multi-agent Navigation Graph (MaNG), which provides maximal clearance for each agent. This kind of graphs is suitable for steering robots, however they do not reflect human navigational cognition and are therefore of only limited applicability for pedestrian simulation.

Approaches which are capable to more accurately model human perception and cognition are based on visibility graphs [15]. A visibility graph consists of vertices defined by sources, destinations and obstacles within a scenario. Two nodes are connected if they are in line-of-sight. To avoid redundant edges, a reduced visibility graph can be constructed by categorizing edges into supporting and separating edges [15]. In [19], such a visibility graph is used to navigate agents through a scenario. Based on this visibility graph, a pre-computed shortest path map is stored. If other moving agents are located on the pre-calculated path, a recalculation has to be performed. Since this recalculation is very computational intensive, the focus of Choset's work lies on the approximation of agents' positions in order to minimize the number of recalculations by excluding agents which are outside the viewable region of the subject under examination. Gloor et al. [20] propose to construct a visibility graph by placing nodes at a certain distance from convex corners. This approach prevents simulated pedestrians from walking too close around a corner, but it also produces many nodes, which are dispensable.

In this paper we describe a novel navigation graph generation algorithm which is based on the idea of placing nodes at a certain distance from each corner, but discards all superfluous nodes. Furthermore, the resulting graph is not as dense as a common visibility graph because geometrically close edges are omitted.

3 Model setup

An important requirement is that the simulator is able to run in real time, as the simulator is designed as a training tool. To achieve such high performance, a cellular automaton model for space discretization in combination with a conservative force model [21] has been chosen, i.e. a model based on energy potentials that describe the influencing forces on each pedestrian (attracting force of the destination, repellent forces of obstacles as well as the repellent forces of other pedestrians). Combining the cellular automaton with these forces, the navigation of single pedestrians can be modelled efficiently [22]. This combination makes it possible to quickly update pedestrians' positions while taking into account the interactions between them. However, using this approach key aspects of pedestrian movement are neglected, namely the individual navigational behaviour of pedestrians as well as the large-scale orientation of pedestrians. Using only the cellular automaton model, the simulated pedestrians appear as being short-sighted, since only neighbouring cells are considered in each update step.

In order to take into account different degrees of knowledge of efficient routes towards a destination as well as individual navigational behaviour (e.g. keep as close as possible to the direction to the destination, choose routes with less turns, etc.) without losing computational speed, the basic model is extended in our proposal by a navigation graph. Pedestrians move according to the cellular automaton from one graph node to the next. The graph models the large-scale orientation of pedestrians. Thus the pedestrians' decisions can now also be influenced by events at the edges. An overview of the hierarchical setting of the model is illustrated in Figure 1. Using this graph approach, different route choice behaviours can be implemented and thus different pedestrian types (e.g. pedestrians who are / are not familiar with a particular environment).

The navigation graph is based on the concept of visibility graphs. However, our method differs from the method described in [15] and [20]. First, navigation points are constructed that will serve as graph vertices and then these vertices are connected according to criteria depending on the location of the sources and destinations of a scenario.

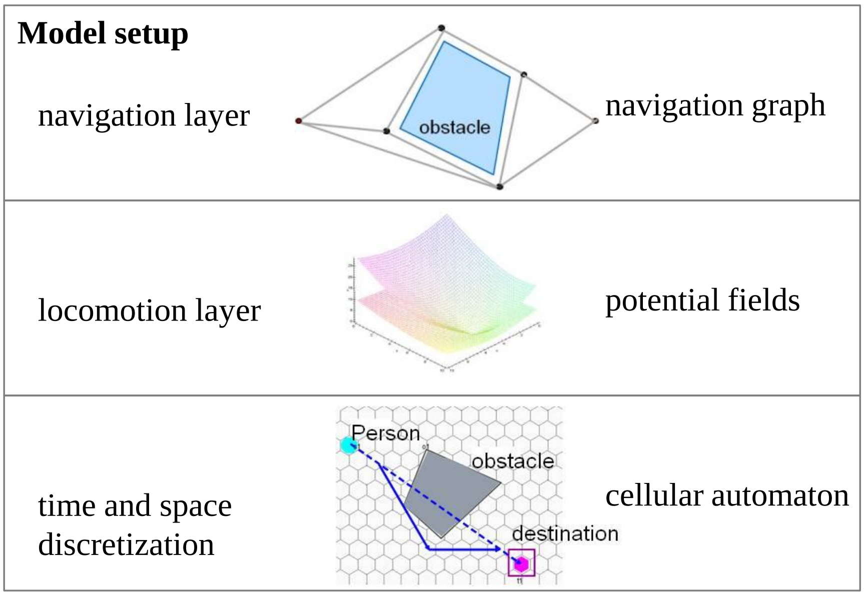

Figure 1 Hierarchical setup of simulation model

4 Construction of a navigation graph from a geometric scenario setup

During the simulation, pedestrians are routed from vertex to vertex on the navigation graph, until they reach their destination. Thus, one requirement for a navigation graph is that the graph represents real routes of pedestrians as closely as possible. The Generalized Voronoi Diagram (GVD) forms a graph with equidistant edges from each obstacle. By taking a GVD as a navigation graph, pedestrians would walk equidistant to all obstacles. This would result in large detours. With visibility graphs, the resulting routes are close to obstacle corners and thus reflect human behaviour more realistically. Hence, we rely on visibility graphs as navigation graphs.

The derivation of the visibility graph from a given geometry is achieved in multiple steps, which are described in detail below.

4.1 Derivation of navigation points (vertices)

In the first step, the geometry of a given simulation scenario is examined and navigation points are placed around each obstacle at each convex corner. Each of these points will correspond to a vertex in the visibility graph.

Since there are two different types of obstacles – one-dimensional (e.g. walls) and two-dimensional obstacles (e.g. houses) – placement of navigation points has to distinguish between these two cases, as illustrated in Figure 2.

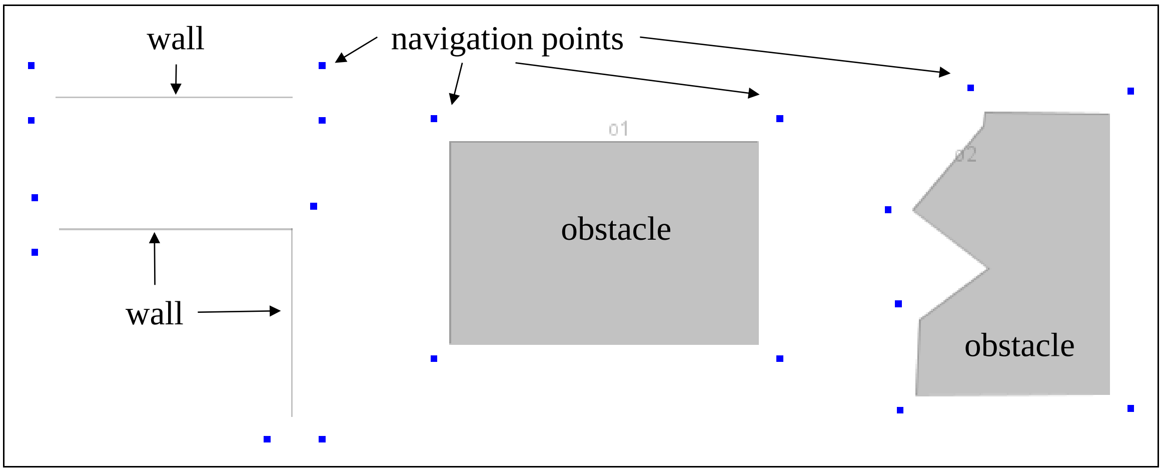

Figure 2 Navigation point placement for walls and obstacles

Depending on the obstacles' geometry, an overlap of two neighbouring navigation points may occur or a corner point is not visible from the corresponding corner. As a result different consistency checks on navigation points are carried out:

- One criterion for a valid navigation point is that the imaginary sight line between the point and the corresponding obstacle corner is not obstructed. If this check fails, the corresponding points are re-located in such a way that they fulfil this criterion. An illustration is given in Figure 3. If the distance between two obstacles is smaller than the length of a cell of the cellular automaton (i.e. no pedestrians is able to walk between the two obstacles), no navigation point is placed. The exact algorithm is described in [23] and [24].
- To avoid redundant points, i.e. points which are located geometrically close to one another, a second check is performed. If two or more nodes are located close together, the simplest approach would be to merge them into one single point. But, as can be seen in Figure 4 (left), this may result in non-reachable destinations, since the criterion of visibility between two nodes is not fulfilled (see Section 4.2 Connecting vertices: a cone-based search methodfor more details). To accommodate this, the following check is implemented: for each navigation point, the corresponding obstacle corner is stored. If another corner point lies closer to the corresponding corner than the navigation point

itself, the corresponding corner is added to this point. If the same is true vice versa, then both points are merged into a new navigation point. This ensures that there is always a way around each corner. An example of this approach is shown in Figure 4 (right).

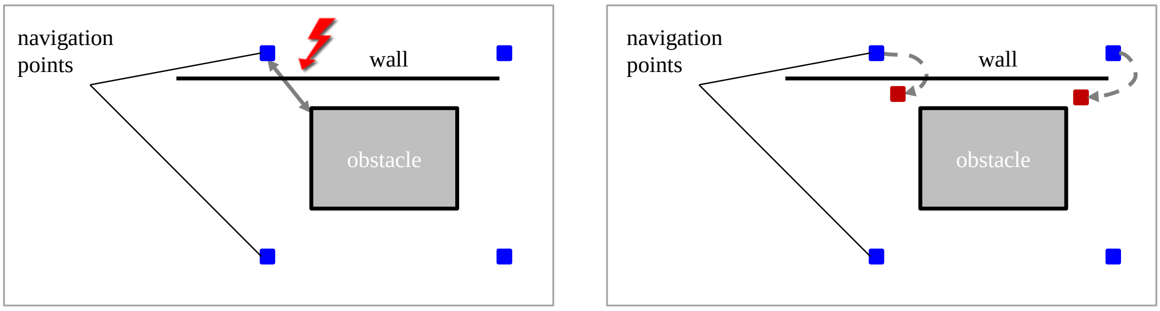

Figure 3 Replacement of navigation points so that there are no obstructions between the point and corresponding obstacle corner

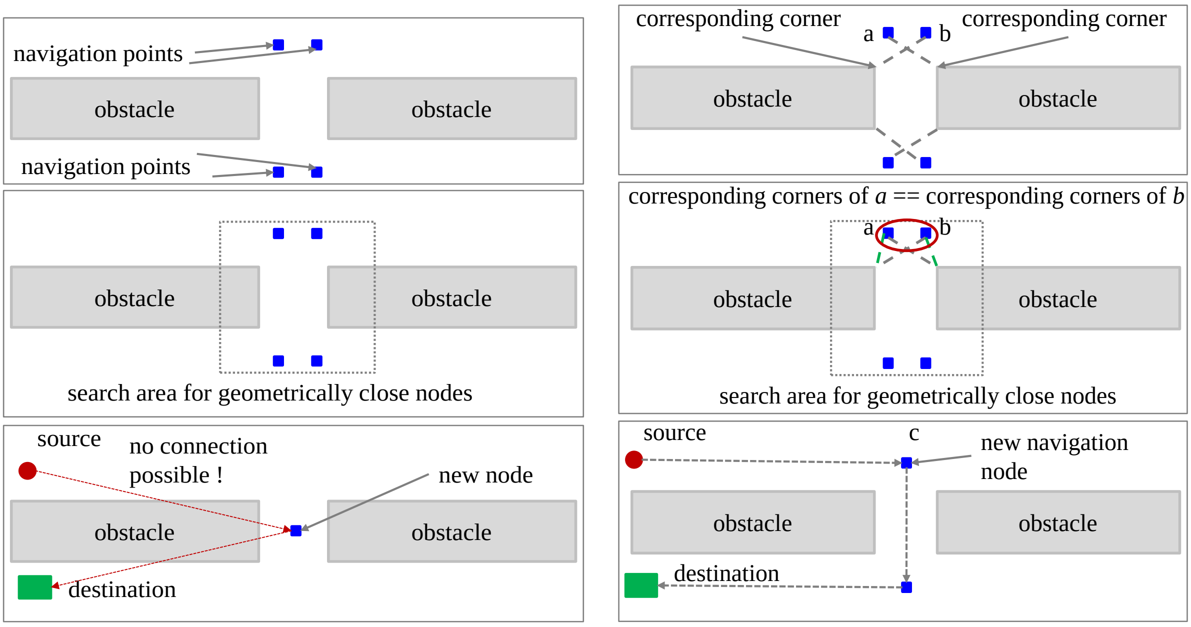

Figure 4 Left: If all geometrically close points are merged, this can lead to non-visibility graphs: as a result, there is no connection around a corner. Right: Correct merging of two adjacent nodes by checking the corresponding corners of each node; only if two nodes have the same corresponding corners are they merged into a new, single point.

4.2 Connecting vertices: a cone-based search method

Once all navigation points have been created, these points are connected according to certain rules. The first rule is that two vertices can only be connected if there is a sight line between them. This is the definition of a visibility graph and a basic requirement for the routing algorithms, since the Euclidean distance is used to determine edge weights.
The objective is to construct a directed graph, which is able to model human navigation behaviour in as much detail as possible. However, the inclusion of all possible navigation options, i.e. all possible edges fulfilling the criterion of visibility, would result in a very dense navigation graph, which would dramatically decrease computational efficiency. Since the aim

is to simulate large crowds in real time, the resulting graph should cover major aspects of navigation decisions while at the same time being as sparse as possible. To obtain a sparse graph, redundant edges are avoided. Redundant edges are those that form a loop, i.e. do not lead to the destination, as well as those that are geometrically located very close to each other. An example of such edges is shown in Figure 5.

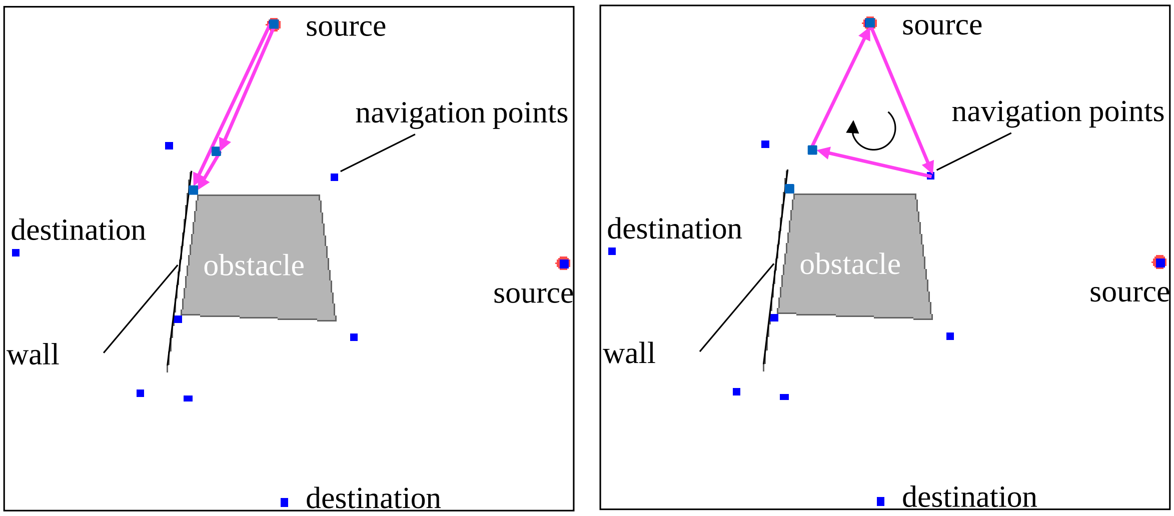

Figure 5. Redundant edges:
left, geometrically close edges; right, edges that lead pedestrians back to the source.

A spatial index, namely an R*-Tree [25] is used to make visibility graphs more sparse. Using this data structure, nearest neighbours of a point can be found efficiently in a search space that exists of one- and two-dimensional objects such as polygons and points. In [24] we propose to always connect a node with its three nearest neighbour nodes via an edge. However, this procedure is not sufficiently flexible for arbitrary geometries since cases can occur where individual vertices have more than three neighbours that need to be connected in order to cover the full directional range. One example is illustrated in Figure 6: the picture on the left shows the complete visibility graph, the picture in the centre the resulting edges from the former method: here, the source is connected with the three vertices inside the red circles, but the vertices to the far left and far right are not connected. The picture on the right shows the result of our improved method: the connecting edges lead in every direction. This improved method is outlined in more detailed below.

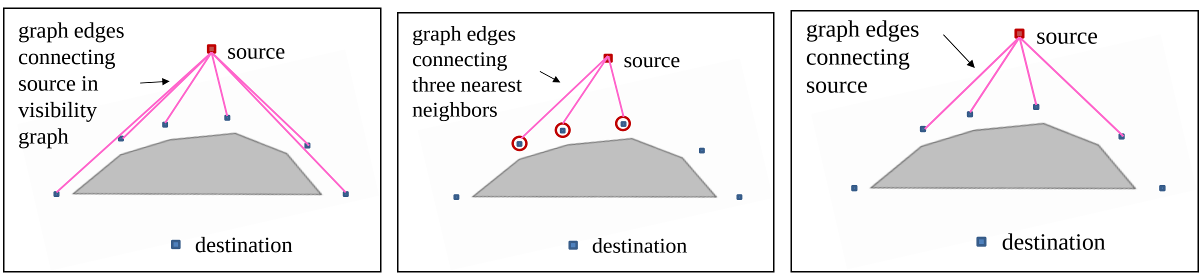

Figure 6: Examples of a geometry, where the algorithm presented in [24] would not find all important neighbouring vertices of a source. Left: a complete visibility graph; middle: connecting the three nearest vertices; right: resulting edges for cone-based method.

The algorithm consists of a cone-based search for finding the most relevant neighbours to be connected. The basic idea is that the angle between two outgoing edges has to be larger than a certain threshold. If the angle is smaller, the longer edge is discarded. The algorithm works as follows: The graph is initialized by inserting all navigation points as vertices. It starts with an arbitrary vertex v_i. For this vertex, a rectangular search area is defined, such that the inspected vertex v_i as well as all destinations of the given scenario are located inside that area. Furthermore, if any obstacle obstructs the sight lines between v_i and each of the reachable destinations, the search area is extended until it encompasses these obstacles (Figure **9**a). Inside this area, a search is conducted for all vertices within sight of v_i, sorted according to their distance to v_i (Figure **9**b). An edge is created between the closest vertex and v_i. Starting from v_i, a cone-shaped area is defined around the edge with an angle α_{cone}. This cone-shaped area (Figure **9**c) is then subtracted from the search area and the next vertex chosen with the smallest distance to the vertex v_i. The same procedure is conducted with this vertex, i.e. this vertex is connected and the corresponding cone-shaped area removed from the search area. The resulting edges of v_i are shown in Figure **9**d. The algorithm is repeated for every graph vertex. Figure 7 shows the pseudo code of the algorithm.

Algorithm: Graph construction

Parameters: Set of orientation points; Set of destinations $D \subseteq V$; Obstacles O

Output: Graph G (V, E)

- Add all orientation points as vertices V to G
- **For** each vertex v_i in $V \backslash D$ **Do**
  - Define search area A_{search}, such that: $v_i \in A_{\mathrm{search}}, \forall_{d_i \in D} \in A_{\mathrm{search}}$
  - **For** all obstacles o_i in line of sight between v_i and $d_i \in D$ **Do**
    - Expand A_{search}, such that all orientation points of $o_i \in A_{\mathrm{search}}$
  - **End For**
  - Search inside A_{search} for all orientation points in sight $\{n\}$ and sort them according to their distance to v_i.
  - **For** each $n_i \in \{n\}$ **Do**
    - **If** n_i inside valid area A_{search} **Then**
      - insert edge (v_i, n_i) to G
      - cut cone-shaped section A_{cone_vi} around edge (v_i, n_i): $A_{search} = A_{search} \backslash A_{cone\_vi}$
    - **End If**
  - **End For**
- **End For**

Figure 7: Pseudo-code for graph generation algorithm

The number of the resulting edges can be varied by changing the value of the angle α_{cone}, which defines the cone-shaped sections. Larger angles correspond to sparser graphs, since larger cones are removed from the search area. In Figure 8, different graphs for different values α_{cone} are illustrated. For all further examples, we chose an angle $\alpha_{cone} = \pi/20$.

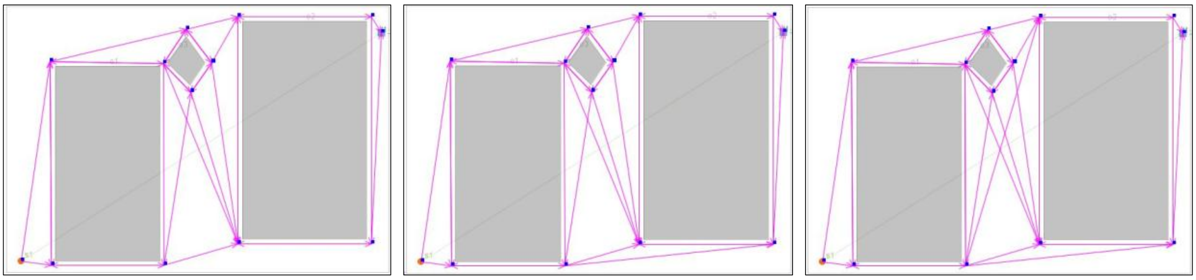

Figure 8: Resulting graphs for different alpha values: Left: $\alpha_{cone}= \pi/15$; Middle: $\alpha_{cone}= \pi/20$; Right: $\alpha_{cone}= \pi/25$

The resulting graph provides at least one route from each source to every destination if there is one. This follows directly from the algorithm: by definition, there is at least one navigation point in the line of sight in each search area. Thus each vertex is connected to at least one other vertex. This connected vertex refers either to the destination itself or it is connected to a vertex which leads around an obstacle that obstructs the sight line between this vertex and the destination. Accordingly, there is either a path from the start vertex to the destination or no connection at all between source and destination. Since the search is directed, the order of the inspected vertices can be chosen arbitrarily, but the resulting graph always remains the same.

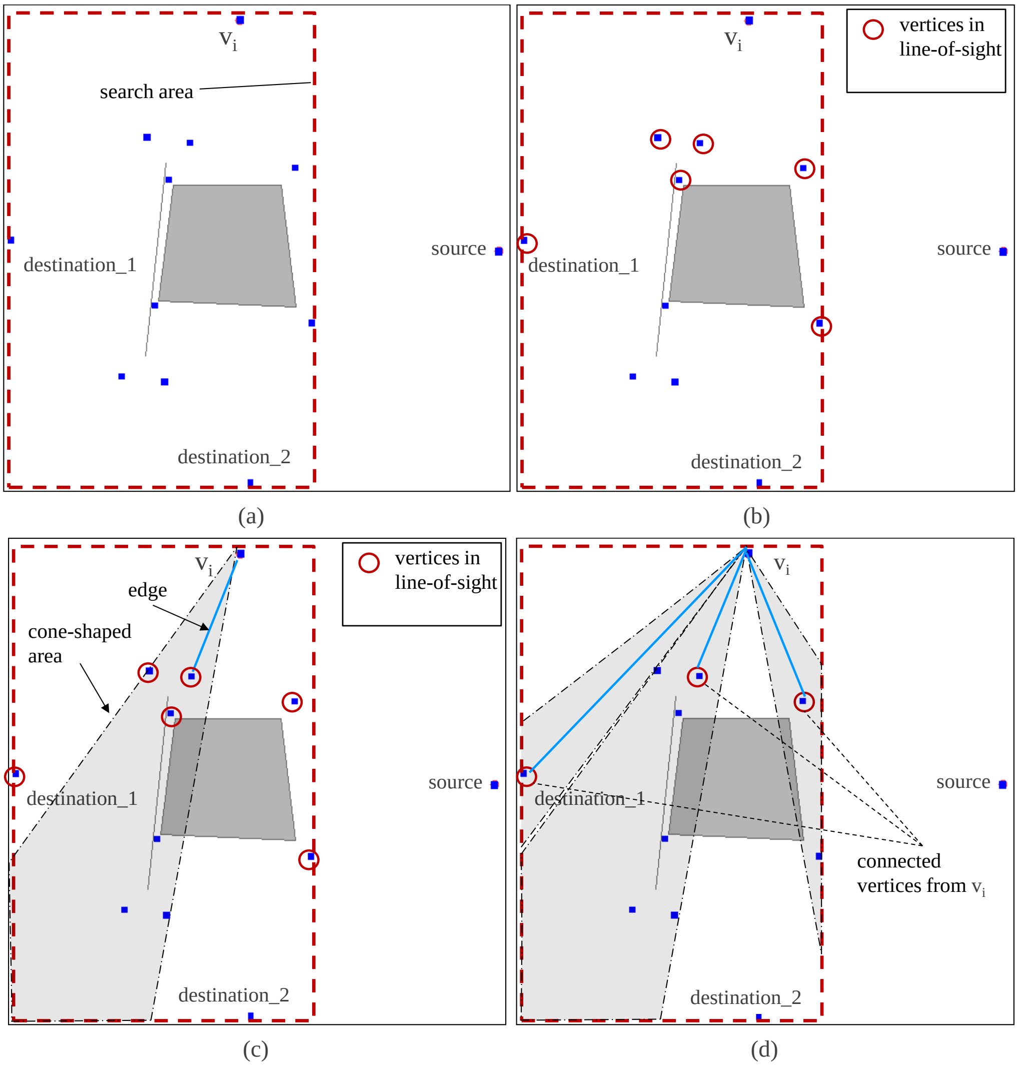

Figure 9: Steps connecting node v_i with neighbours: (a) defined search area (b) all vertices visible from v_i (c) cone-shaped section for excluding vertices in same direction (d) resulting edges from vertex v_i

4.3 Connectivity check

An essential feature of the visibility graph is to provide at least one route leading from all sources to all assigned destinations. A check is undertaken to ascertain if there are any vertices that are not connected to any source or destination. This is likely because the algorithm inspects every given navigation point (i.e. vertex). These vertices and their corresponding edges can be discarded. To remove these, the connected components are first identified within the graph [26]. This is done using a breadth-first iterator which starts from each source vertex and checks if at least one destination can be reached. If so, the source and destination belong to a connected component and all vertices of this connected component are going to be kept.

Figure **10** shows an example of a graph consisting of two connected sets, but only one set contains source and destination: the second set can, therefore, be discarded.

Note, that by applying this technique the resulting graph is no longer generic but directly depends on the locations of sources and destinations within the scenario. This reduced graph improves the performance of the route-finding algorithms.

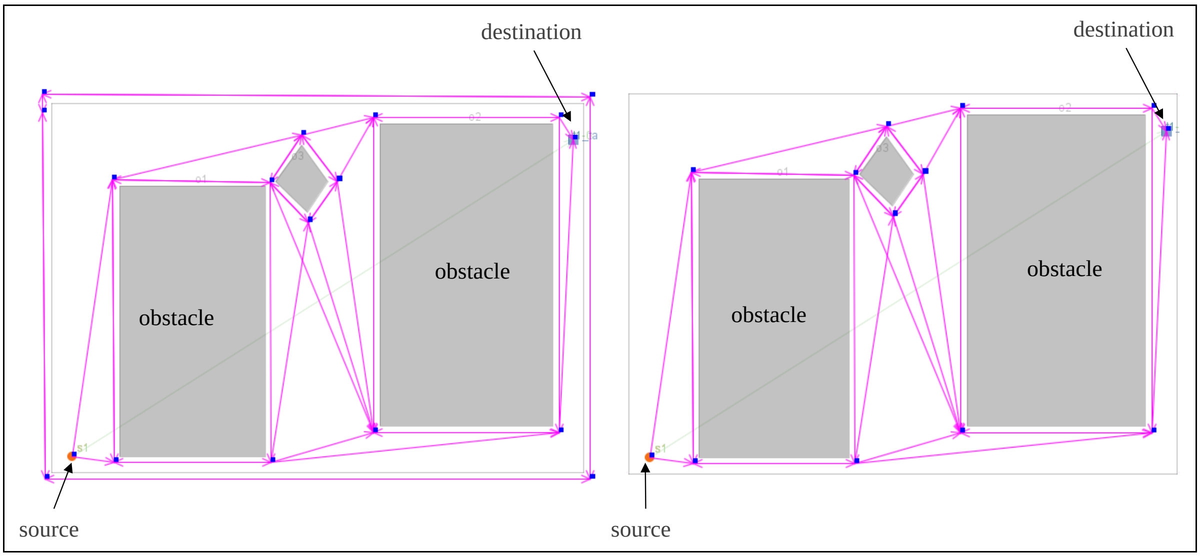

Figure 10: A graph with two connected sets (left) and a graph with only one connected set that contains source and destination (right)

5 Application areas for the extended plain simulation with a navigation graph

Using this visibility graph as a navigation graph, more complex situations can be modelled as well as different pedestrian behaviours with respect to large-scale orientation. The pedestrians of the simulation are no longer shortsighted, but can react to situations which occur further away (such as congestion). An example of using this graph to assist security staff can be found in [27]. In the following, we introduce the mapping of different walking behaviours using this graph.

5.1 Modelling different pedestrian behaviour

This graph can be used to model individual pedestrian behaviour. There are different algorithms, which take into account the different behaviour of simulated pedestrians.

Pedestrians are categorised into three different main types:

- Pedestrians who are very familiar with the environment and will choose an alternative route if their current route is very crowded.
- Pedestrians with no detailed local knowledge who only know the direction of the destination but no details about the specific route.
- Pedestrians who are not familiar with the location and make their decisions based on local criteria: The choice of the next turn depends on the characteristics of the outgoing edges (long edges vs. short edges) and the route choices of other pedestrians.

Pedestrians with detailed local knowledge are modelled using the *Fastest Path Algorithm*. The fastest path is calculated using the Dijkstra Shortest Path Algorithm [28] with dynamic edge weights, taking travelling times instead of distances as edge weights. Hence the assigned edge weights can change over time. We derive this travel time from the density on an edge and the corresponding mean velocity. A detailed description of the algorithm can be found in [23].

Pedestrians, who employ an air-line (as the crow flies) distance between their current location and the destination for navigating through a scenario they are not familiar with, are modelled by applying a variant of the *A* Algorithm* [29]. The basic idea of this heuristic algorithm is to take given information into account and combine it with assumptions about missing information. In our case, the given knowledge is the direction to the destination (the exact route is outside the field of vision) and the current distance to the next navigation point, i.e. vertex (the visibility graph ensures that the information is always available).

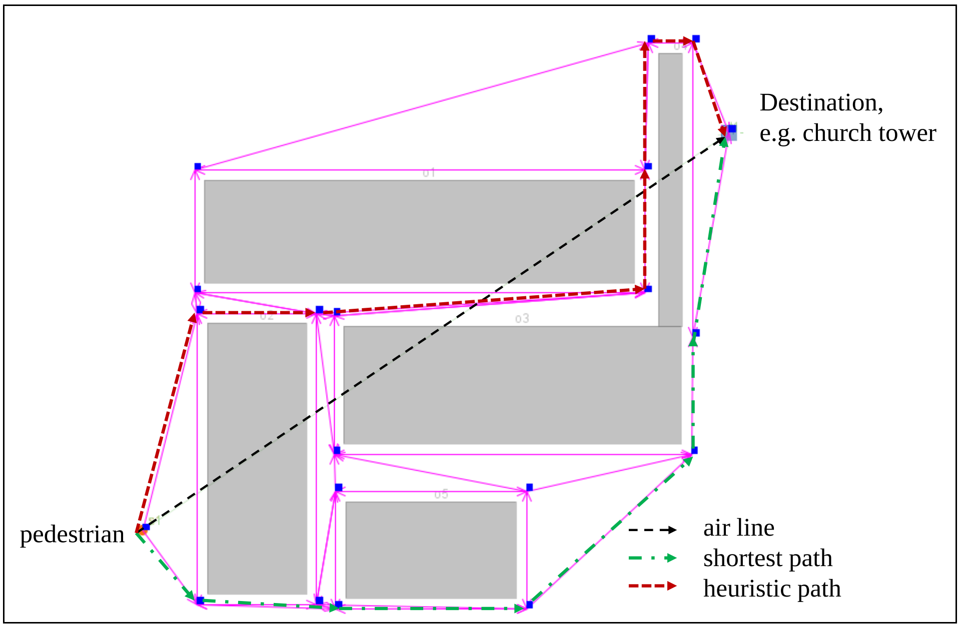

Figure 11 Example for the heuristic A* algorithm: due to the air-line estimation, the algorithm does not find the shortest path, but the path that most approximates the air-line to the destination

To implement this, the algorithm uses real distance edge weights (or as an alternative travel times) for the known part of the route and an additional heuristic measure for the unknown part. This unknown part refers to the air-line distance to the destination to estimate the unknown part. Figure 11 shows an example that illustrates the basic idea of the algorithm.

The navigation behaviour of the third type of pedestrians is modelled by a *Probabilistic Choice Algorithm*. With this algorithm, local-based decisions as well as non-deterministic route-choice behaviour of pedestrians are modelled. Furthermore, it reflects the "trail behaviour" of pedestrian, i.e. the tendency to follow paths chosen by other pedestrians. The algorithm is implemented according to [30]:
Different values of an edge, such as its derivation from the air-line to the destination as well as the length of the edge and improvement of distance to destination, are totalled. This combined value is its edge weight. At each node, the weight of each outgoing edge is scaled to a value between 0 and 1, such that the sum of all edge weights is 1. Using roulette wheel selection [31], one of the possible outgoing edges is chosen: The higher the value of an edge, the higher the probability that it is selected.

To map the "trail behaviour", each chosen edge is assigned an amount of pheromone, which evaporates over time according to Ant Colonization Optimization algorithms [32].

A detailed definition including validation of individual behaviour as well as the description of the algorithms can be found in [33].

This algorithm differs from the Route Choice Self Organization (RSCO) algorithm proposed in [14] in such way, that Teknomo defines three different principles for edge selection: the permission (whether an edge is accessible), the interaction (avoidance of crowded edges) and navigation (relative enhancement to the destination). The *Probabilistic Choice Algorithm* does not take into account any densities on the edges as a parameter for avoiding congested edges, but instead it models the trail behaviour by means of evaporating pheromone, meaning that pedestrians, who are unfamiliar with the location, will choose an edge more likely, if there are already pedestrians walking along this edge. Secondly and more importantly, in contrast to the RSCO algorithm the *Probabilistic Choice Algorithm* is not deterministic with respect to edge choice.

5.2 Test case

The developed graph generation method and navigation strategies have been applied on an example scenario shown in Figure 10 (right). Here, pedestrians walk from the lower left corner to the upper right corner within a room. Three obstacles are located inside this room. In total, 1200 pedestrians were simulated with a generation rate of 6 pedestrians per second.

In a first simulation the graph layer was not used. Simulation runs using the navigation graph were then conducted: pedestrians walked according to the three algorithms *Fastest Path, A** and *Probabilistic Choice*. The last simulation was based on a combination of all three algorithms.

Figure 12 to Figure 16 show screenshots of the simulations at different points in time: Without using a graph, each simulated pedestrian takes the same route. As shown in Figure 12, this results in significant congestion in front of the bottleneck of the right obstacle. This is because simulations without a navigation graph update a pedestrian's position solely using the cellular automaton. During the simulation, all neighbouring cells are examined for each pedestrian at each time step, choosing the cell with the lowest potential value if it is not

occupied by either an obstacle or another pedestrian. This selection process is conducted until all pedestrians reach their destination. From this setup it is obvious that each pedestrian is short sighted and chooses a similar route to walk to his destination. The only varying factor is the number of pedestrians that walk the same way and occupy cells.
Figure 13 shows the pedestrians walking according to the *Fastest Path Algorithm*. One can see a wider spread of routes resulting from the dynamic routing. To begin with, the fastest path is identical to the shortest path. After a while, the route becomes too crowded. The route south of the first obstacle becomes faster, since pedestrians taking the shortest route have to slow down in response to the intensity of its use. Later in the simulation, a third route becomes the fastest, as the last part of the two former routes are crowded on the last common segment.

In Figure 14, the results of the *A* Algorithm* are illustrated. In spite of the results of the *Fastest Path Algorithm*, the most likely way to the destination is south of the left obstacle. This is explained by the fact that the lower route is located closer to the air-line to the destination. However, as with the *Fastest Path Algorithm*, the route becomes too crowded after a while and pedestrians start to walk on the west side of the left obstacle. This happens, since travel times are used as fixed edge weights instead of Euclidean distances. At some point, the bottleneck north of the right obstacle becomes crowded and pedestrians start to walk along the south of the right obstacle.

The results of the *Probabilistic Choice Algorithm* are demonstrated in Figure 15. Initially, there is a wide spread of the pedestrians' route choices, since edges are chosen by probability. After a while, the path north of both obstacles becomes more likely, due to greater pheromone deposition. Not taking into account the densities on the edges, the pattern seen in the plain simulation happens, namely congestion in front of the upper left corner of the right obstacle. This reflects exactly what was supposed to be modelled: if a person is not familiar with a place, he will most likely stay on his path, since he does not know alternative routes that lead to the destination.

Each algorithm on its own maps only one kind of pedestrian behaviour. Combining all algorithms, the simulation produces the results shown in Figure 16. The number of pedestrians for each type is chosen according to a distribution rate, which has been derived by the experiment presented in [33], in which large scale orientation is investigated: Students are sent to a well-known location in the Munich city centre (Germany) without a map. On their return, they document the path they have chosen. Additionally, each participant fills out a questionnaire regarding way-finding behaviour and orientation. From the analysis of the documented paths and the questionnaire, a distribution rate for the different types of orientation has been derived.

One can observe that no high densities occur at any location. Likewise, the different navigation behaviours seem to be reflected well, since one can identify the preferred routes of each algorithm.

Nevertheless, the results of the experimental simulation need to be validated to ascertain how realistic they are. We can, however, clearly see that the simulation becomes more realistic when we apply a navigation graph, since unlikely congestions no longer occur. Realistic in this context means that assuming a certain distribution rate of the simulated pedestrians' local knowledge as given, the simulation is able to reflect these different route choice behaviours accordingly.

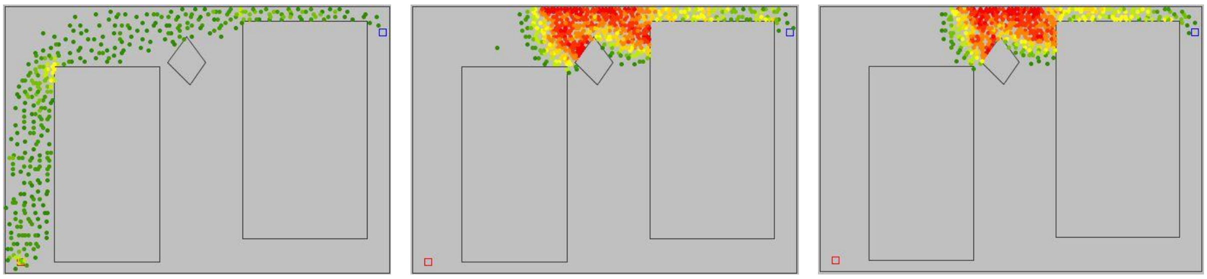

Figure 12: Screenshot of a simulation of 1200 pedestrians walking from the lower left corner to the upper right corner *without using a navigation graph* after 63 seconds (left), 263 seconds (middle) and 284 seconds (right).

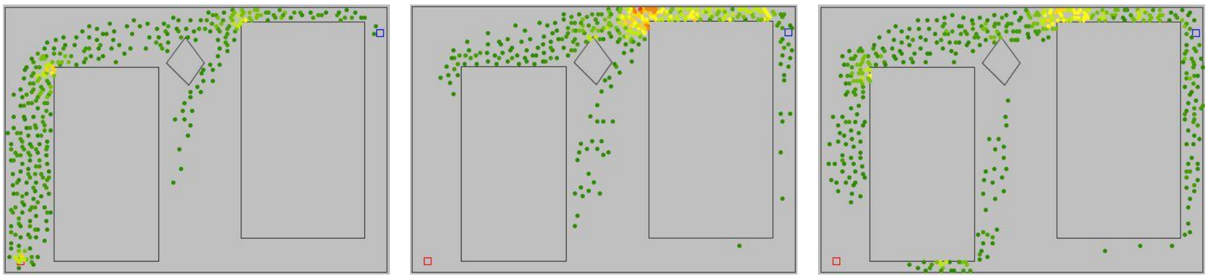

Figure 13: Screenshot of the same simulation using a navigation graph with *Fastest Path Algorithm* after 63 seconds (left), 263 seconds (middle) and 284 seconds (right)

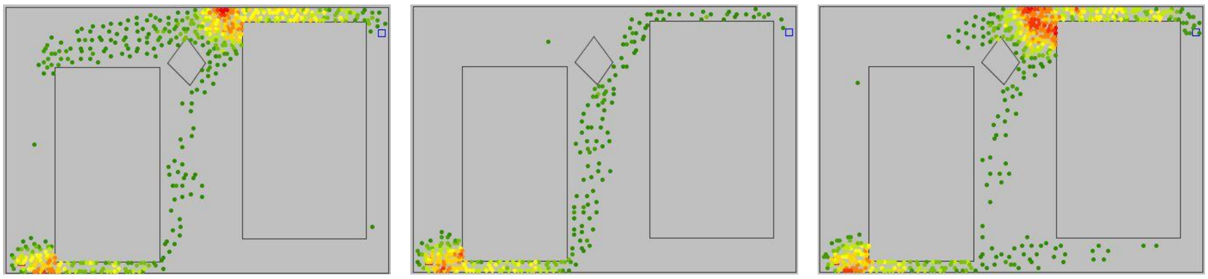

Figure 14: Screenshot of the same simulation using a navigation graph with *A* algorithm* after 63 seconds (left), 263 seconds (middle) and 284 seconds (right)

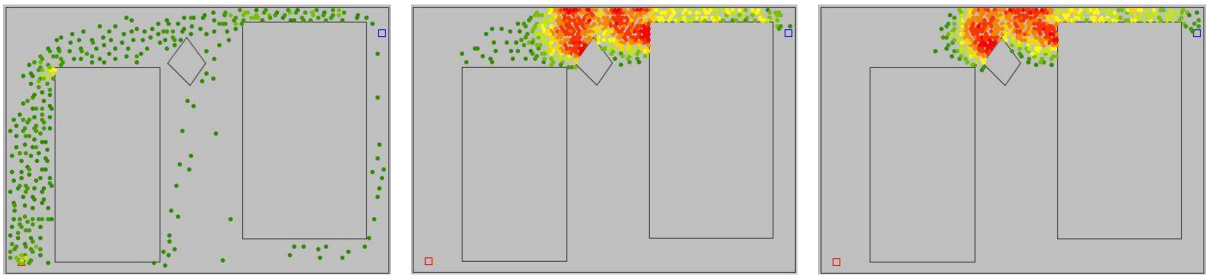

Figure 15 Screenshot of the same simulation using a navigation graph with *Probabilistic Choice* after 63 seconds (left), 263 seconds (middle) and 284 seconds (right)

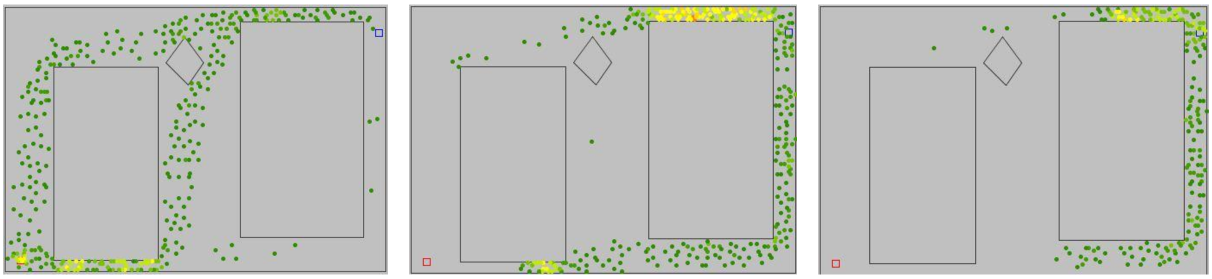

Figure 16 Screenshot of the same simulation using a combination of the above algorithms after 63 seconds (left), 263 seconds (middle) and 284 seconds (right)

5.3 Real-world application scenario

To illustrate the application of the developed simulation approach, we are discussing a real-world application scenario. Here, an architect or engineer responsible for the layout of an office building is using the graph-extended pedestrian simulation to investigate different options with respect to the number and localisation of (emergency) exits. The office building has 4 floors. The investigated floor plan comprises 41 offices (Figure 17). For each office, the number of expected occupants is known. In the first layout option, there is one exit stair located on the west side and another one located on the east side (see Figure 17).

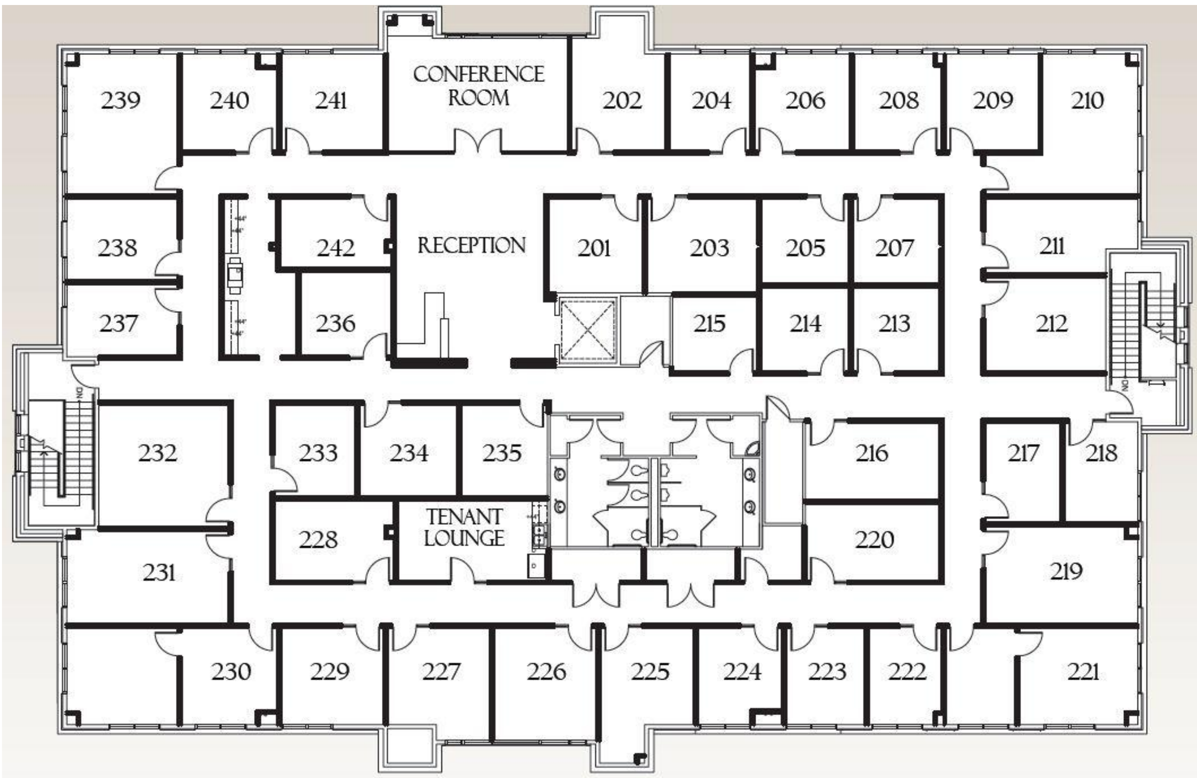

Figure 17 Floor plan of the investigated office building with 41 offices

By means of the methodology introduced in Section 4, a navigation graph is automatically generated for the given floor plan. The result is depicted in Figure 18. The scenario has been simulated for a total number of 161 pedestrians. At the start of the simulation they are placed according to the given number of occupants for each individual room. Each exit stair has been assigned a suitable capacity. Each pedestrian is routed towards the exit which is closest to his/her original position.

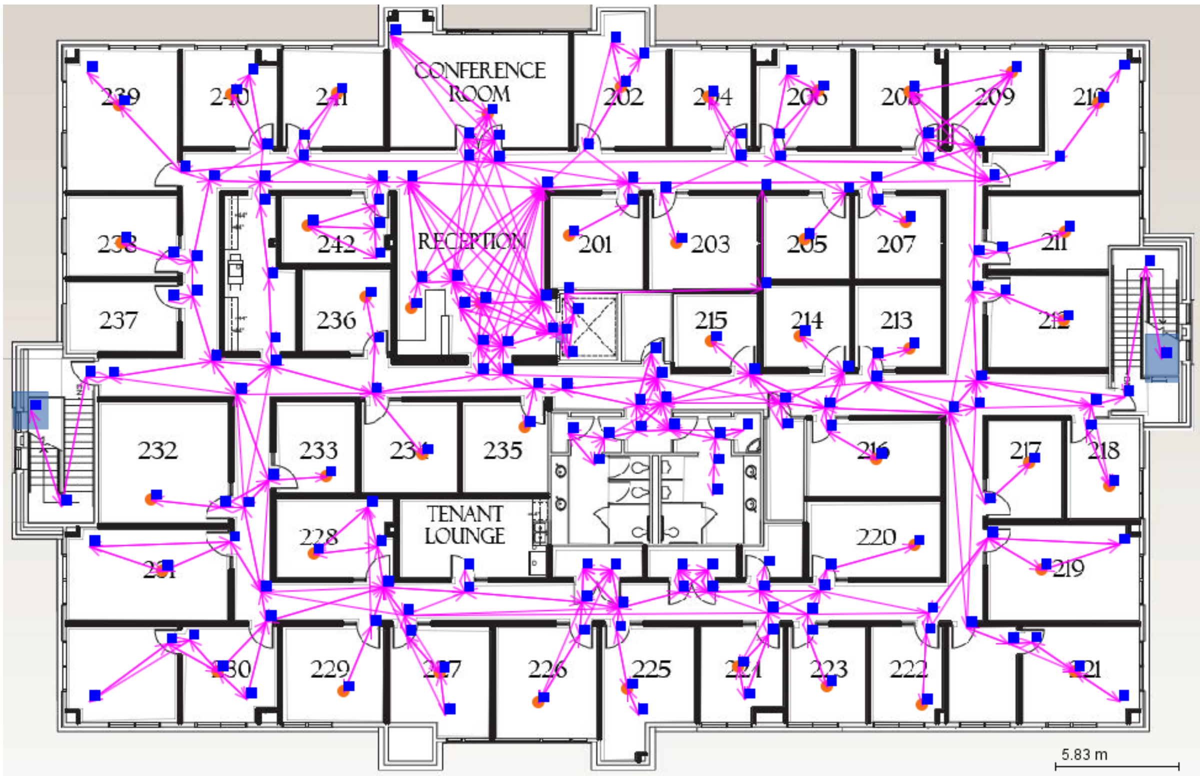

Figure 18 The automatically generated navigation graph.

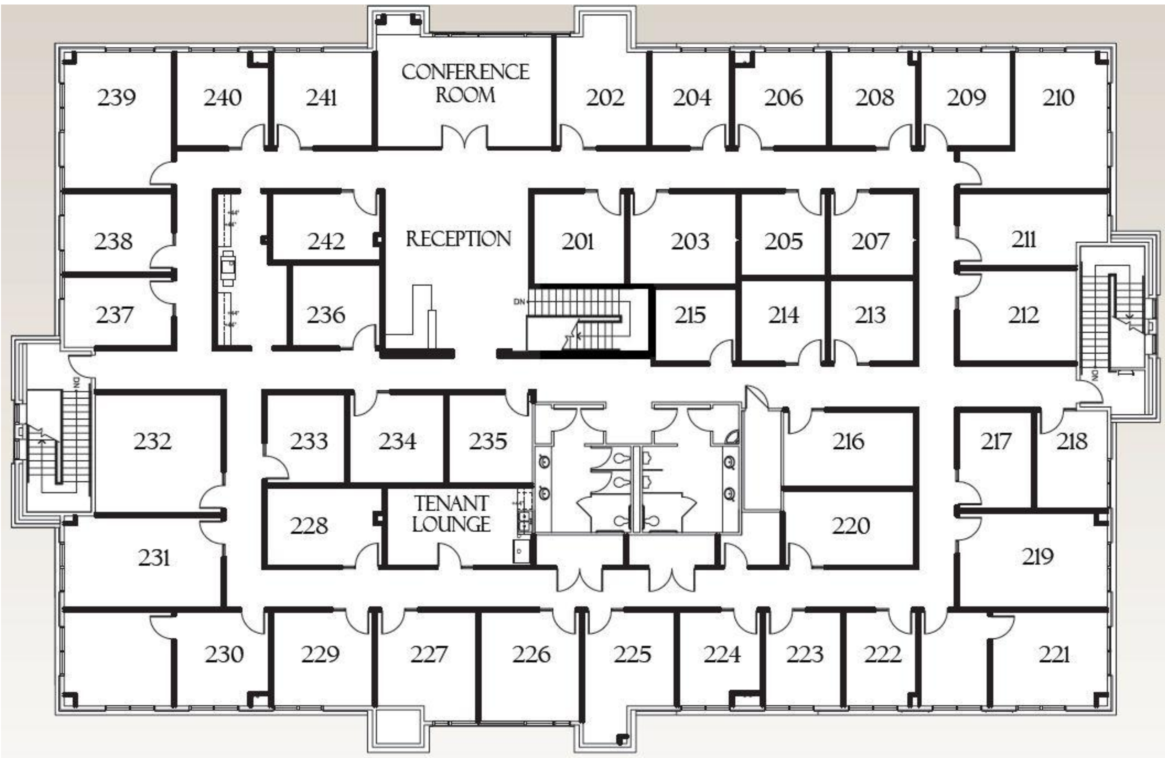

Figure 19 Modified floor plan: An additional stair has been placed at the central position.

The simulation results, depicted in Table 1 left-hand side, show that this setup results in congestion in front of the stair. In an emergency situation, such constellations must be avoided.

For this reason, in the second option an additional stair has been placed in the floor at a central position (Figure 19). Again, the navigation graph is automatically generated. The results of the subsequent simulation run are depicted in Table 1, right hand side. It can be seen that in this option, no congestions occur and thus a critical situation is avoided.

The engineer can take these simulation results into account for the final decision regarding the number and position of the exit stairs. The automated navigation graph generation method introduced in this paper provides the possibility for a quick and effortless evaluation of floor plan alternatives with respect to evacuation situations.

Time	Option 1: Two exit stairs	Option 2: Three exit stairs
After 15 seconds		
After 30 seconds		
After 60 seconds		

Table 1 Results of the simulation: The left column depicts the simulation results for the two exit stairs option, the right columns shows the results for the three exit stairs option. Whereas for the first option critical congestion occur during an evacuation, they do not so for the second option.

6 Discussion

Engineers responsible for the layout of public buildings and large event areas have to consider the movement and behaviour of pedestrian crowds in order to prevent critical situations. Recently, computational simulations have been increasingly used to predict the dynamics of pedestrian crowds. However, most of the available simulation systems either rely on rather simple pedestrian navigation models, which reflect human behaviour only in a very limited manner, or are computationally very expensive. In this paper, a sophisticated graph-based approach has been presented, which allows integrating advanced navigational behaviour with computationally efficient simulations.

The implementation of this approach includes an advanced technique for generating sparse navigation graphs from a given spatial layout of the scenario under investigation. This graph is a subset of a standard visibility graph. It can be used to not only map individual pedestrian behaviour in pedestrian simulations, but also to model the large-scale orientation of pedestrians.

The advantage of using such a navigation graph instead of using an agent-based approach is an enormous reduction in computational effort. Furthermore, a scenario can be easily changed during runtime. Closing doors or making routes non-accessible, for example due to fire, can be modelled simply by deleting the corresponding edges.

The paper introduces a new method for constructing a navigation graph from a given geometry by reducing a visibility graph with a cone-based search method. The main advantage of the method is that the resulting graph is very sparse. Unlike standard visibility graphs, geometrically close edges are merged into a single edge, while at the same time maintaining broad spatial coverage for ensuring a better modelling of navigational behaviour. Furthermore, all vertices are discarded that are not part of any connected component with sources as well as destinations.

To measure the quality of the resulting graph, a next step in our research will be to define a metric and evaluate the navigation graph according to this metric in comparison to existing graphs.

To demonstrate the advantage of a graph-extended simulation, the results of a set of sample simulation scenarios have been presented. Different levels of local knowledge are modelled using three different routing algorithms: Pedestrians who are familiar with a location are simulated using a *Fastest Path Algorithm*. Pedestrians with partial knowledge of a location are modelled according to the heuristic *A* Algorithm*. The movements of pedestrians with no local knowledge are modelled using the *Probabilistic Choice Algorithm* – a derivation of an Ant Colonization Algorithm. The application of these three algorithms within the simulation improved the simulation results significantly, since artificial congestions produced by the static cellular automaton model could be eliminated. Furthermore, a wide range of diverging route choice behaviour is realized, i.e. not just the shortest routes are taken by the simulated persons, but also non-optimal ones, which reflects human navigation more naturally. In addition to the modelling of different route choices, the application of the navigation graph resolves the problem of short-sightedness in conventional simulation models and makes it possible to consider the pedestrians' sense of large-scale orientation.

In future research we plan to continue validating the simulation results to ensure the quality of the different routing algorithms. According to these validation results, we will further improve the algorithm and incorporate interactions between the different types of pedestrians.

References

[1] A. Schadschneider, W. Klingsch, H. Klüpfel, T. Kretz, C. Rogsch, A. Seyfried, Evacuation dynamics: Empirical results, modeling and applications, Encyclopedia of Complexity and System Science (2009) 3142–3176.

[2] H.W. Hamacher, S.A. Tjandra, Mathematical modelling of evacuation problems: A state of the art, in: M. Schreckenberg (Ed.), Pedestrian and Evacuation Dynamics, Springer, Berlin, 2002, pp. 227–266.

[3] D. Helbing, A fluid-dynamic model for the movement of pedestrians, in: Complex Systems, 1992, pp. 391–415.

[4] L.F. Henderson, The Statistics of Crowd Fluids, Nature 229 (1971) 381–383.

[5] D. Helbing, P. Molnár, Social Force Model for Pedestrian Dynamics, Physical Review E 5 (1995) 4282–4286.

[6] C. Burstedde, K. Klauck, A. Schadschneider, J. Zittartz, Simulation of pedestrian dynamics using a two-dimensional cellular automaton, Physica A: Statistical Mechanics and its Applications 3-4 (2001) 507-525.

[7] N. Ronald, L. Sterling, M. Kirley, An Agent-Based Approach To Modelling Pedestrian Behaviour, Lecture Notes in Computer Science (2007) 145-156.

[8] N. Pelechano, A. Malkawi, Evacuation simulation models: Challenges in modeling high rise building evacuation with cellular automata approaches, Automation in Construction 4 (2008) 377–385.

[9] C. Reynolds, Steering Behaviors for Autonomous Characters, in: Game Developers Conference, 1999.

[10] S.R. Musse, D. Thalmann, A Model of Human Crowd Behavior: Group Inter-Relationship and Collision Detection Analysis, in: Workshop Computer Animation and Simulation of Eurographics, 1997, pp. 39-52.

[11] F. Durupinar, J.M. Allbeck, N. Pelechano, N. Badler, Creating crowd variation with the OCEAN personality model, in: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 3, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2008, pp. 1217-1220.

[12] A. Lerner, Y. Chrysanthou, D. Lischinski, Crowds by Examples, in: D. Cohen-Or, P. Slavík (Eds.), EUROGRAPHICS 2007, 2007.

[13] M. Asano, T. Iryo, M. Kuwahara, Microscopic pedestrian simulation model combined with a tactical model for route choice behaviour, Transportation Research Part C: Emerging Technologies 6 (2010) 842–855.

[14] K. Teknomo, Modeling mobile traffic agents on network simulation, in: Proceeing of the 16th Annual Conference of Transportation Science Society of the Philippines (TSSP), Manila, 2008.

[15] H.M. Choset, Principles of robot motion, MIT Press, Cambridge, Mass., 2005.

[16] S.M. LaValle, Planning Algorithms, Cambridge University Press, Cambridge, U.K, 2006.

[17] F. Aurenhammer, Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure, ACM Comput. Surv. (1991) 345-405.

[18] A. Sud, E. Andersen, S. Curtis, M.C. Lin, D. Manocha, Real-Time Path Planning in Dynamic Virtual Environments Using Multiagent Navigation Graphs, IEEE Transactions on Visualization and Computer Graphics (2008) 526-538.

[19] O. Arikan, S. Chenney, D.A. Forsyth, Efficient multi-agent path planning, in: In Proceedings of the 2001 Eurographics Workshop on Animation and Simulation, 2001, pp. 151-162.

[20] C. Gloor, P. Stucki, K. Nagel, Hybrid Techniques for Pedestrian Simulations, in: P. Sloot, B. Chopard, A. Hoekstra (Eds.), Cellular Automata, Springer Berlin / Heidelberg, 2004, pp. 581–590.

[21] W. Klein, G. Köster, A. Meister, Towards The Calibration of Pedestrian Stream Models, in: J. Weglarz, R. Wyrzykowski, B. Szymanski (Eds.), 8th Int. Conf.on Parallel Processing and Applied Mathematics, Springer, Berlin, 2010.

[22] D. Hartmann, Adaptive pedestrian dynamics based on geodesics, New Journal of Physics 4 (2010) 43032.

[23] M. Höcker, V. Berkhahn, A. Kneidl, A. Borrmann, W. Klein, Graph-based approaches for simulating pedestrian dynamics in building models, in: University College Cork (Ed.), 8th European Conference on Product & Process Modelling (ECPPM), Cork, Ireland, 2010.
[24] A. Kneidl, A. Borrmann, D. Hartmann, Einsatz von graphbasierten Ansätzen in einer mikroskopischen Personenstromsimulation für die Wegewahl der Fußgänger, in: T. Krämer, S. Richter, F. Enge, B. Kraft (Eds.), Forum Bauinformatik 2010, Shaker, Aachen, 2010.
[25] N. Beckmann, H.-P. Kriegel, R. Schneider, B. Seeger, The R*-tree: an efficient and robust access method for points and rectangles, SIGMOD Rec. 2 (1990) 322-331.
[26] A. Gibbons, Algorithmic graph theory, Cambridge Univ. Press, Cambridge, 1999.
[27] A. Kneidl, M. Thiemann, D. Hartmann, A. Borrmann, Combining pedestrian simulation with a network flow optimization to support security staff in handling an evacuation of a soccer stadium, in: Proceedings of European Conference Forum 2011, Cork, Cork, 2011.
[28] E.W. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1959) 269.
[29] Y. Li, M. Höcker, Heuristische Navigationsalgorithmen für Fußgängersimulationen, in: P. von Both (Ed.), Forum Bauinformatik 2009, Universitätsverl., Karlsruhe, 2009, pp. 25–36.
[30] D. Angus, Solving a unique Shortest Path problem using Ant Colony Optimisation.
[31] T. Bäck, Evolutionary algorithms in theory and practice, Oxford Univ. Press, New York, 1996.
[32] M. Dorigo, Optimization learning and natural algorithms, Dissertation, Mailand, 1992.
[33] A. Kneidl, A. Borrmann, How Do Pedestrians find their Way? Results of an experimental study with students compared to simulation results, in: W. Jaskolowski, P. Kepka (Eds.), EMEVAC, The Main School of Fire Service, Warsaw, 2011.