Graph-based approaches for simulating pedestrian dynamics in building models

Mario Höcker & Volker Berkhahn

Institut für Bauinformatik, Leibniz Universität Hannover, Callinstr. 34, 30167 Hannover, Germany

Angelika Kneidl, André Borrmann

Lehrstuhl für Computation in Engineering, Technische Universität München, Arcisstr. 21, 80290 München, Germany

Wolfram Klein

Siemens AG, CT T DE TC3, 80200 München, Germany

ABSTRACT: This paper presents different path-finding algorithms for simulating pedestrian dynamics in building models. Starting from a given model (scenario), we show how to automatically derive a visibility graph. This graph is used as the underlying structure for routing pedestrians from their sources to their destinations. Based on this graph, we search for fastest routes by means of a conventional Dijkstra algorithm where we assign dynamically changing travel times as edge weights. To update the shortest paths due to changing edge weights, we introduce a heuristic A* algorithm, which is faster in finding optimal paths. We compare the results of our approach to a variant where we assign Euclidean distances as static edge weights. Additionally, we show that the A* algorithm has a better performance in finding the shortest path for most cases.

1 INTRODUCTION

Pedestrian simulations are used for a wide variety of applications, namely the identification of possible conflict points or bottlenecks in buildings and surroundings, the determination of evacuation times, the determination of optimal evacuation routes etc. To simulate pedestrian behavior, a wide variety of different models is used as summarized in [Schadschneider et al. 2009]. Our objective is to describe individual movement and navigation processes in buildings with a microscopic graph-based model.

2 MODEL DESCRIPTION

To simulate pedestrian movements, we use two different model approaches. The first approach combines a cellular automaton [Burstedde et al. 2001, Emmerich and Rank 1997, Kinkeldey 2003, Klüpfel 2003, Kretz 2007] and a force model according to [Schadschneider et al. 2009]. The second approach [Helbing et al. 2001, Höcker et al. 2009] describes pedestrian movement by means of space continuous interactions. The individual interaction force contains a driving term directed to a destination as well as a repulsive term originating from other pedestrians and obstacles.

One challenge of this model is the calibration between the different force terms to achieve the most realistic walking and navigation behavior in complex buildings. To solve this challenge, we configure the individual driving force by introducing a visibility graph [de Berg et al. 2000] of the scenario topography. This graph is used to navigate pedestrians from their sources to their destinations by applying different routing criterions and algorithms.

3 GRAPH DERVIATION FROM THE SCENARIO TOPOGRAPHY

To map pedestrian movements within a scenario, we introduce a method to construct a visibility graph as the underlying structure for routing algorithms. We automatically derive the graph from the scenario topography consisting of sources and destinations as well as different kinds of obstacles such as walls, polygons (e.g. desks inside buildings) etc. From this initial geometry we start extracting the graph by applying the following two steps.

First, we place so-called orientation points on the bisector of each convex obstacle corner (refer to Fig. 1 top). We choose the distance to the corners according to an appropriate measure such that “artificial” congestions at corners can be avoided when pedestrians are trying to pass these orientation points.

In order to prevent that a point is not visible from the related corner, it has to be checked if another obstacle, e.g. the line of a wall, intersects the imaginary sight line between the corner and the point. In case of an intersection, we move this point closer to the corner.

In detail, the orientation point is placed midway of the shortest distance vector between the cutting edge and the corner (see Fig. 2).

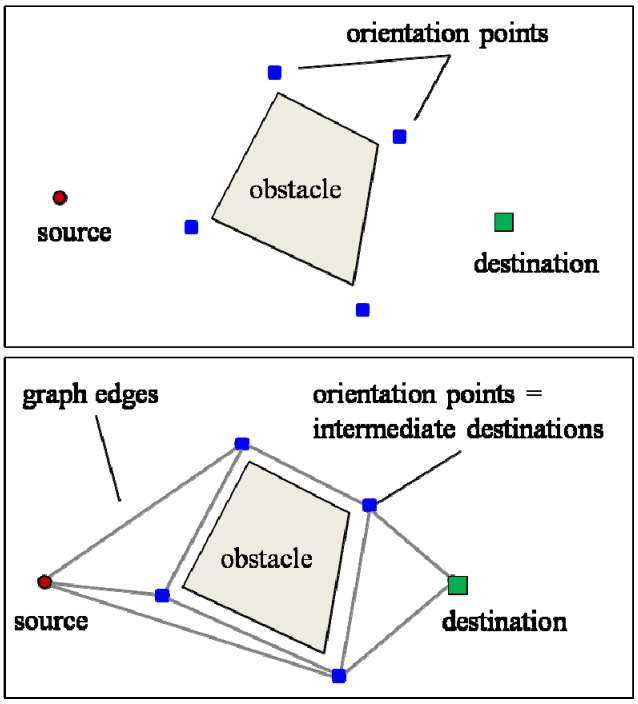

Fig. 1. Graph derivation from the scenario topography.

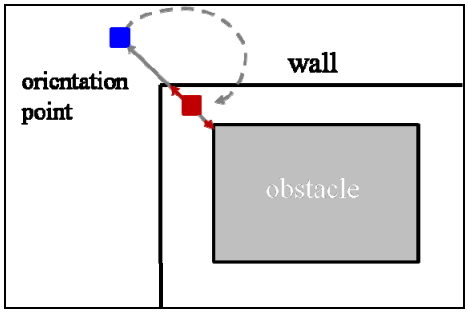

Fig. 2. Repositioning of orientation points.

To avoid orientation points that are too close to each other, we prune these redundant points by melting these neighboring points into one new point as shown in Fig. 3.

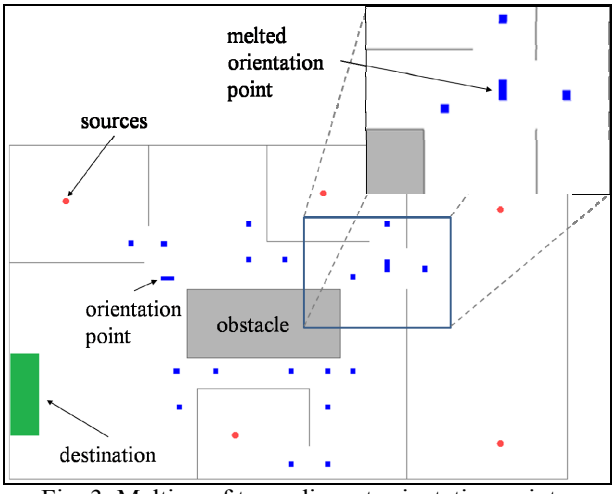

Fig. 3. Melting of two adjacent orientation points.

Finally, the resulting orientation points represent the graph nodes.

Second, we start connecting these graph nodes with each other subsequently. Two orientation points are connected by means of a graph edge, if they are in sight of each other. In addition, they are connected to sources and destinations in the same manner.

After the graph generation we perform a check for dispensable edges to achieve a most efficient navigation process. For individual routing, there are only edges necessary leading around the obstacle corners. This characteristic can be detected with a so-called corner related sight criterion illustrated in Fig. 4. The obstacle corner is connected to a sight crossing area bounded by the imaginary sight lines placed along the two corner sides.

We define an edge as necessary if it leads from the orientation point out of the sight crossing area. If this criterion is not fulfilled, the edge will be disconnected. In this manner, up to 75 percent of the initial edges can be reduced.

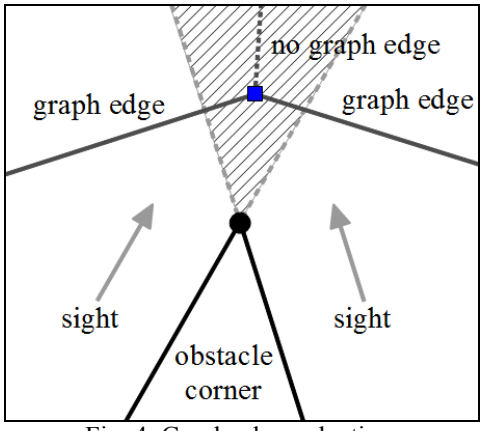

Fig. 4. Graph edge reduction.

The resulting graph (as shown in Fig. 1 bottom) can be used for the routing algorithms defining at least one path between the sources and destinations if it exists. This follows directly from the construction procedure.

4 ROUTING ALGORITHMS

We apply different algorithms, which are based on the developed visibility graph, to route the pedestrians from their sources to their destinations.

More precisely, we combined two different routing strategies: The first one is a common shortest path finding algorithm [Dijkstra 1959] with dynamic edge weight variation. The second one is a heuristic algorithm according to [Russel and Norvig 2003], which is able to find as well an optimal shortest path, but in most cases more efficiently – depending on the heuristic values. We start with the description of the common Dijkstra algorithm with adaptation to our problem, followed by the A* algorithm with the heuristics we used.

4.1 *Dijkstra algorithm with dynamic edge weights*

Since the Dijkstra algorithm considers edge weights to compare two edges while traversing the graph from a given node (source) to a designated node (destination), we focus on determining the edge weights to match pedestrian behavior.

Our approach is to take travel times as edge weights instead of Euclidean distances. The travel time is derived as follows: For each edge, we define an area associated with it (see Fig. 5). Then, we count the number of pedestrians that are traversing along that edge. From the number of pedestrians in relation to the whole accessible size of the area, we derive the density factor as in (1):

$$density\_factor = \frac{number\_of\_pedestrians}{accessible\_edge\_area} \quad (1)$$

As edge areas can overlap, we have to associate pedestrians not only to one edge area but to all surrounding edges whose areas are affected.

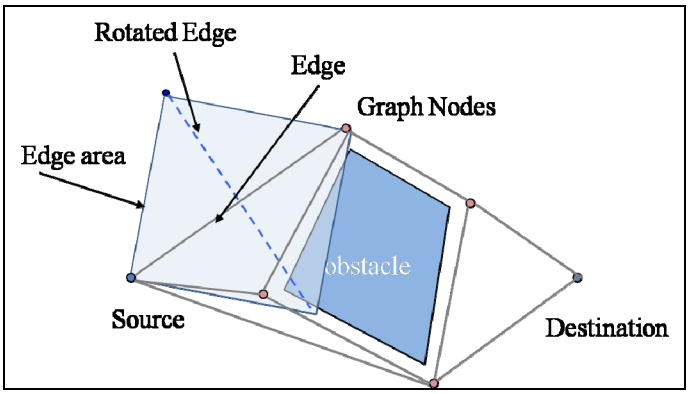

Fig. 5. Construction of an area to measure density on an edge: The edge is rotated by 90° to define the corner points of the area.

According to [Weidmann 1993], there exists a relation between density and velocity for pedestrians (see Fig. 6). We take this proposed relation to obtain a deceleration factor for the pedestrians.

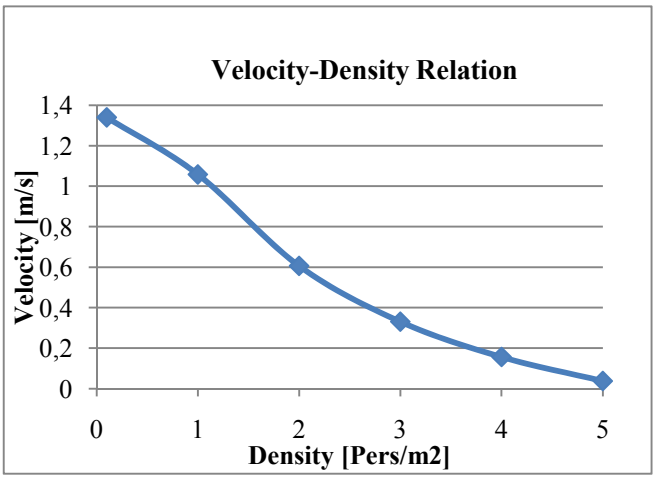

Fig. 6. Velocity-Density Relation for pedestrians, as proposed in [Weidmann 1993].

More precisely, we take the density factor of an area and derive the corresponding velocity. Multiplying this velocity with the Euclidean distance of the edge, the expected travel time is computed as:

$$travel\_time_{edge} = veloc_{edge}^{density} * dist_{edge} \quad (2)$$

As travel times may vary over time depending on the density on each edge, we have to re-compute it with each pedestrian that enters a specific edge area.

Consequently, a shortest path is time-dependent, so that pedestrians entering the scenario later may get assigned a different path than earlier ones.

In case of more than one route leading to the target, this technique even allows us to avoid congestions by dynamically performing checks for each pedestrian entering an edge. If this assigned edge has a density factor that exceeds a certain threshold, we search for a different route.

4.2 *A* algorithm with heuristic node weights*

To minimize runtime for the re-computation of shortest ways, we introduce the A* algorithm, which may not need to inspect each graph edge (like a conventional Dijkstra algorithm) by applying an additional heuristic.

The A* algorithm uses therefore edge weights as well as node weights calculated with a node related function $F(x) = G(x) + a \cdot H(x)$. $G(x)$ represents the real length of a detected shortest path between a source and the (intermediate) destination x. The real length equals the sum of the path edge weights, e.g. the Euclidian distance or the travel time. $H(x)$ represents the approximated length of the path between x and the destination, scaled with a factor a. For $a = 0$, the A* algorithm analyzes all graph nodes – like the Dijkstra algorithm. For $a \neq 0$, the heuristic path search ranges over a part of the network only, so that the time involved gets reduced.

A pseudo code of the A* algorithm is illustrated in Fig. 7. The algorithm uses a list L_O for analyzed graph nodes and a closed list L_C. At the beginning of an individual routing process, both lists are empty. L_O is initialized with the source and weights it with $F(x)$. Next, we loop until the destination is reached or if L_O does not contain a graph node any more.

The loop starts with checking L_O and removing the element x with the lowest node weight. For this element x all successors x_N will be analyzed, which are not already contained in the closed list L_C.

Successors which are not even contained in the list L_O, are added to it and weighted with $F(x)$; furthermore, a backward-reference between each successor x_N and x will be stored.

If a successor x_N is already contained in L_O, the node weight and the backward-reference of the successor x_N will be updated, if the calculated weight is lower than the stored node weight.

At the end of each loop, the element x will be stored in the closed list L_C.

In case of removing the destination from L_O for examination, the algorithm terminates and reconstructs the shortest path between the source and the destination via the stored backward-links.

Algorithm: Path-finding
Parameters: Graph G:=(X;E), source s in X, destination d in X, node weight function $F(x)$

1. init node lists L_O und L_C
2. store s with $F(s)$ in L_O
3. while L_O is not empty
4. select node x | x in L_O, stored $F(x)$ = min
5. remove x from L_O
6. if $x == d$
7. init path P from s to d using the stored backward-links
8. **Result**: P
9. end if
10. for all successors x_N of x in G
11. if x_N not element of L_C
12. if x_N element of L_O
13. if $F(x_N)$ < stored $F(x_N)$
14. stored $F(x_N) = F(x_N)$
15. set the stored backward-link from x_N to x
16. end if
17. else
18. store x_N with $F(x_N)$ in L_O
19. store backward-link from x_N to x
20. end if
21. end if
22. end for
23. store x in L_C
24. end while
25. **Result**: No path between s and d found

Fig. 7. Pseudo code of the A* algorithm.

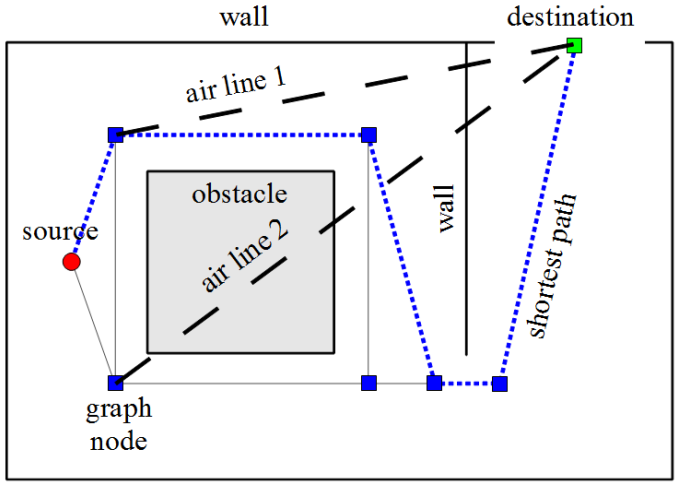

Fig. 8. Heuristic path search with the air line distance.

A commonly used definition of the heuristic function $H(x)$ is the Euclidian distance of the air line between the current graph node and the destination. The air line distance heuristic describes individual map knowledge about the surrounding space and the related navigation behavior realistically.

Fig. 8 illustrates such a heuristic path search with a simple scenario. The approximated shortest path is depicted by a dashed line leading from the source (left in the figure) to the destination (right in the figure). It runs permanently along the lowest air line distances. In this manner, it differs partly from the real shortest path, which runs along the lower side of the obstacle. In this case, the heuristic term $H(x)$ dominates the node weight function $F(x)$.

5 APPLICATION EXAMPLE

By means of the presented model approaches and routing algorithms, pedestrian flow inside an office building was simulated.

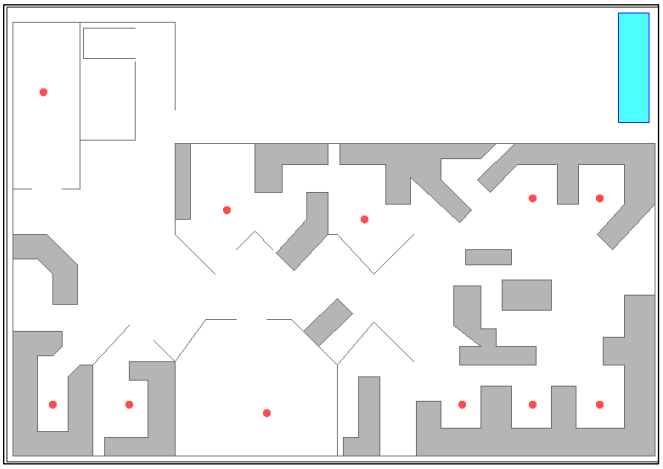

Fig. 9. Topography of the tested scenario: The grey areas and lines define obstacles and walls of the building, the dots refer to sources, and the rectangle on the upper right corner defines the destination of all pedestrians.

Fig. 9 illustrates the scenario topography of our example scenario. On this scenario we first ran the conventional Dijkstra algorithm with static and dynamic edge weights and compared the results. Additionally we applied the A* algorithm on this scenario and show that the number of visited nodes can be sufficiently smaller than with a conventional Dijkstra algorithm.

5.1 *Comparison of the Dijkstra algorithm with static and dynamic edge weights*

For the comparison between static and dynamic path finding, we used the first, space discrete model approach, as described in [Klein et al. 2010]. Pedestrians are placed inside the offices (dots); their destination is situated outside the building. In order to observe density effects, we generated pedestrians periodically (six pedestrians per source and second) until a sufficient number (750 pedestrians) was reached.

Fig. 10. Mainstream of pedestrians shown for the static path algorithm (left) and dynamic path algorithm (right).

Fig. 10 shows the mainstream (the counted number of pedestrians that passed on a rectangular grid cell throughout the simulation) - in the upper picture for the static shortest path where we assigned the Euclidean distance as edge weights and in the lower picture for the dynamic fastest path.

Applying the static path algorithm, we can observe that all pedestrians walk the same route as there is no dynamical adaptation of new routes. In contrast, the mainstream resulting from the dynamic path algorithm shows that there were pedestrians who changed their route according to the densities on the conventional shortest path. We get a better distribution of the pedestrians throughout the right part of the office.

The drawback concerning runtime is – that in the dynamic version – a new fastest path has to be calculated as soon as an edge exceeds a certain density.

5.2 *Results for the A* algorithm*

To resolve these runtime issues we can calculate these individual dynamic paths – depending on the travel time – more efficiently by using the heuristic A* algorithm instead of the conventional Dijkstra algorithm. The behavior of the A* algorithm was tested with the simulation tool JWalkerS [JWalkerS 2010], which is an object-oriented implementation of the second, space-continuous model approach (see section 2).

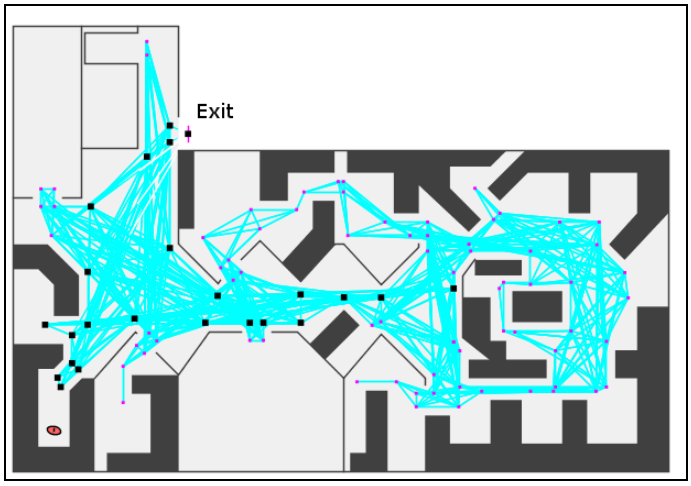

Fig. 11. Searched graph nodes (black filled quadrangles) of an individual routing process with the A* algorithm ($a = 1$). A conventional Dijkstra algorithm would have visited all nodes of the visibility graph.

Fehler! Verweisquelle konnte nicht gefunden werden. illustrates a visibility graph calculated with JWalkerS: Orientation nodes are represented by small-sized circles. The simulation is restricted to the routing process of a single pedestrian from an office room towards the exit which is located in the upper part of the figure. The pedestrian is depicted by an elliptical space in the lower left room of the office building. The approximated shortest path is located between the heuristically analyzed graph nodes represented by black filled quadrangles. It is clearly visible that the path search ranges only over a small part of the network whereas the conventional Dijkstra algorithm visits every single node of the graph. As a consequence, the performance of the A* algorithm is much better than the one of the Dijkstra algorithm for this specific case.

More generally spoken, one can observe, that in buildings with a complex geometry (e.g. many offices or rooms), the A* algorithm does not search within rooms that are neither positioned in the same direction as the air line to the destination nor close to the real shortest path.

Configuring the A* algorithm, it must be considered that the probability for differences between approximated and real shortest path increases with the heuristic factor. From our experience we can state, that for $a \leq 1$, we could not observe any differences between the real shortest path and the one found by the A* algorithm.

6 OUTLOOK

In this contribution, we introduced new methods for a graph-based description of paths in buildings as well as individual routing processes.

Here, the paths are automatically derived from scenario topography by placing orientation points on the bisectors of convex obstacle corners and calculating sight connections. By using criterions for melting adjacent orientation points and for reducing sight connections on crossings, we developed an efficient graph structure by reducing the number of nodes and edges as far as possible.

Moreover, we presented routing algorithms which can dynamically assign new individual paths depending on pedestrian densities. Along with an application example we demonstrated that this dynamic assignment leads to a more realistic prognosis of pedestrian route choice and thus of pedestrian distribution. To minimize the path-finding time involved, we introduced an A* algorithm using the airline distance between an orientation point and a destination as heuristic value. The A* algorithm is able to the individual paths faster than the conventional Dijkstra algorithm, for specific cases as we described in Section 5.2.

Nevertheless, there are still some aspects of the graph structure as well as the routing algorithms to be investigated.

In order to simulate multi-level scenarios, graph-based models for level connections like stairs or escalators must be taken into account. For a multi-level navigation process, a further stage of the air line heuristic is necessary, e.g. with a temporary orientation to the level connections. Besides, another heuristic representing individual route decision criterions must be developed.

To be more efficient in deriving the edge weights, we will improve the algorithm in defining non-overlapping edge areas. Another step will be to implement a detection of pedestrian cluster which can be associated with the corresponding edges.

To reduce runtime and start re-calculation of fastest path only when necessary, further work has to focus on defining precise criteria, e.g. slow down factors on edges or distance ratios of *k*-shortest paths etc.

Given this dynamical path-assignment, a next step will be to dynamically vary a scenario during runtime (with occurring events like door closure or fire).

REFERENCES

Burstedde, C., Klauck, K., Schadschneider, A., Zittartz, J., 2001. Simulation of Pedestrian Dynamics Using a 2-dimensional Cellular Automaton, *Physica A* 295, pp. 507-525.

de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O., 2000: "Chapter 15: Visibility Graphs", *Computational Geometry* (2nd ed.), Springer-Verlag, pp. 307–317, ISBN 3-540-65620-0.

Dijkstra, E. W., 1959. A note on two problems in connection with graphs. *Numerische Mathematik* 1, pp. 269-271.

Emmerich H., Rank, E., 1997. An Improved Cellular Automaton Model for Traffic Flow Simulation. Physica A, 234: pp. 676-686, 199.

Kinkeldey, C., 2003. Fussgängersimulation auf der Basis zellulärer Automaten. Studienarbeit im Fach Bauinformatik. Universität Hannover.

Klein, W., Köster, G., Meister, A., 2010. Towards the Calibration of Pedestrian Stream Models. *Lecture Notes in Computer Science: PPAM 2009*. Springer, Berlin Heidelberg.

Klüpfel, H., 2003. A Cellular Automaton Model for Crowd Movement and Egress Simulation. *PhD thesis*, Universität Duisburg–Essen.

Kretz, T., 2007. Pedestrian Traffic. Simulation and Experiments. *PhD thesis*, Universität Duisburg-Essen.

Helbing, D., Farkas, I., Molnár, P., Viscek, T., 2001. Simulation of Pedestrian Crowds in Normal and Evacuation Situations. In: Schreckenberg, M., Sharma, S. D., *Pedestrian and Evacuation Dynamics*, Springer, Berlin, 2002, pp. 21-58.

Höcker, M., Milbradt, P., Seyfried, A., 2009. Simulation of Pedestrian Dynamics and Model Adjustments: A Reality-Based Approach. *Proceedings of the Conference on Traffic and Granular Flow*, Shanghai University, Shanghai.

JWalkerS, 2010. Institut für Bauinformatik, Leibniz Universität Hannover, http://www.bauinf.uni-hannover.de/427.html.

Russel, S. J., Norvig, P., 2003. Artificial Intelligence: A Modern Approach. *International Conference on Conceptual Structures* (*ICCS*), *2nd edition*, Prentice Hall.

Schadschneider, A., Klingsch, W., Kluepfel, H., Kretz, T., Rogsch, C., Seyfried, A., 2009. Evacuation Dynamics: Empirical results, Modeling and Applications. *R.A. Meyers (Ed.), Encyclopaedia of Complexity and System Science Vol. 3*, Berlin Heidelberg: Springer, pp. 3142-3176.

Weidmann, U., 1993. Transporttechnik der Fußgänger. *Schriftenreihe des IVT*, *Vol. 90, 2nd edition*, Institute for Transport Planning and Systems, ETH Zürich.