Coupling force-based and graph-based models for multi-agent wayfinding in complex environments

Gregor Lämmel

Transport Systems Planning and Transport Telematics - TU Berlin

Salzufer 17-19

10587 Berlin, Germany

Tel. (+49)30 314 21376 Fax. (+49)30 314 26269

email: laemmel@vsp.tu-berlin.de

Abstract

In this paper, we propose a new 2D simulation model for large-scale pedestrian crowds. The agents in the model navigate along a navigation graph that represents all possible paths in the environment. Therefore the agents compute a route in the navigation graph from their origin to the desired destination. While moving along the graph, the agents keep track on the other agents and obstacles in the environment and explicitly avoiding collision. The collision avoiding behavior is modeled by a force model,

where obstacles and other agents emit repelling forces. The simulation model also comprises a learning framework, which let the agents optimize their individual paths. Simulation results based on a simple evacuation scenario show the model's validity and applicability to large scenarios.

Keywords: pedestrian crowds, navigation graph, complex wayfinding, multi-agent simulation

Introduction

The microscopic 2D simulation of pedestrian crowds is a challenging task. In the pedestrian simulation community there are different approaches to simulate pedestrian crowds on a microscopic level. One approach uses Cellular Automata (CA) to simulate the pedestrians movement. CA models represent the environment on grid based structure, where each cell on the grid can hold only one agent at a time. CA models have often been used for evacuation simulations (see, e.g., [1, 2]). The main problem with existing CA models is that they do not provide an approach for complex wayfinding, since usually all agents are controlled by one global potential field. In the simulation the agents are moving down the potential. In theory, it would be possible to assign an individual potential field to each agent. This approach, however, would be to complex in terms of computational costs for large scenarios. Other simulation approaches are based on discretized differential equations ("molecular dynamics (MD)") [3, 4]. The best known model using the MD analogy is the social force model introduced by [5]. In the social force model approach each agent has a desired velocity towards a desired destination. During the simulation the agents adapting their velocity to the desired one. However, a real adaption to the desired velocity is not always possible since the agents have to avoid other agents and obstacles. This is modeled by repelling forces emitted by obstacles and agents. Force based models are well understood and have reasonable computational costs. This means those models are well suited to simulate even large scenarios. However, the general model development of force based models does not

focus on complex wayfinding but rather on small scale pedestrian movement.

One approach that extends the force based model with the capability of complex wayfinding has been introduced in [6]. The basic idea of this model is that the agents are navigating on a complex navigation graph that represents all possible paths in the environment. The movement of the agents is modeled by forces that drive the agent along the graph combined with repelling forces known from the social force model. This work extends the graph based force model by a simpler, but still adequate, navigation graph approach and a learning framework that let agents optimize their individual paths through the environment.

This paper introduces a simple approach for navigation graph generation, integrates explicit collision avoidance in the force based model, discusses a learning framework that let the agents individually optimize their paths in order to avoid congestions, and finally the model is tested on simple evacuation scenario to show its validity and applicability to larger scenarios.

Graph generation

The agents in the simulation are moving through the 2D space of environments like the one depicted in Figure 1. The environment can be any spatial extent like buildings, parks, airports, and so on. The environment contains geometric entities like rooms in an office building, benches in a park, or the gates at an airport. Usually the environment is given by architectural drawings or maps and constitutes a geometric model of the reality, where the

arrangement of its entities is given by their spatial configuration. This makes it hard for agents to find appropriate paths through the environment. Therefore it is desirable to have the spatial relations of the environment and its entities captured in a graph based structure. Throughout this work we will refer to this graph as the navigation graph. There are efficient methods to make complex route planning on a graph, since for graphs one can rely on simple and fast least cost past algorithms like Dijkstra's algorithm [7] for route computation. This kind of route computing can be seen as the agents' high level route planning. The high level route planning does not deal with low level planning like obstacle avoidance but guides the agents through the environment along the navigation graph.

It is important that all relevant entities in the environment are interconnected by the navigation graph. The actual definition which entity is relevant is given by the purpose of the simulation and constrained by the geometric detail of the environment.

The navigation graph generation is a challenging task, since it has to be guarantied that all entities are reachable and that no edge of the graph intersects any wall or other obstacles in the environment. Furthermore, it must be guarantied that an appropriate path exist in the navigation graph between any two entities.

One type of graphs that fulfill the requirements are the so called visibility graphs [8]. The visibility graph is a graph where all mutually visible locations are connected by arcs. For the visibility graph construction the environment needs to be given by a set of geometries representing the obstacles. The visibility graph is generated by connecting all mutually visible vertices of the obstacles. In the current context the geometric entities would be added

to the set of geometries in order to include them into the visibility graph. While a visibility graph can be used to find a least cost path[1] between any two entities it can become very complex even for environments with a few simple obstacles. If the geometries representing the obstacles and entities have altogether n vertices, than the correspond visibility graph can have n^2 edges in the worst case (all vertices are mutual visible). In most situations the space complexity will be much less than $O(n^2)$. Still, even simple environments lead to complex visibility graphs as it is illustrated in Figure 2. Visibility graphs have been applied to agent-based pedestrian simulations in the past (see, e.g. [6, 2]).

However, in the current context it is necessary to perceive the travel conditions (i.e. achievable walking speed or congestion level) on the links of the navigation graph. This information is needed by the agents to make their route (re-)planning (e.g. if there is congestion in a certain area of the environment, than it could be faster to take a detour and avoid that congestion). Details on the agents route (re-)planning is given in the next Section. The problem that exist with visibility graphs is that there are many (almost) parallel edges. If some agents travel along a certain edge in the visibility graph then they would also influence the travel conditions on neighboring edges. The specific influence depends on the actual distance between the neighboring edges. Speaking qualitatively, one can say that the influence between those neighboring edges is high since the distance is small.

[1]We refer here to the more general term of "least cost path" instead of the common used term "shortest path". The reason is that in general the shortest path is not alway the best path, meaning the cost of a path may depend on an arbitrary cost function.

There is another type of graph that suited representing the spatial relations in 2D environments. This graph is called skeleton or medial axis. Skeletons have been used for navigation in virtual environments before, e.g. [9]. The skeleton forms a tree that can be seen as a thin version of the corresponding shape. Skeletons have an important characteristic. If one draws a circle around any vertex with an radius equal to vertex's minimum distance to the shape, than this circle touches the shape in at least two points. An intuitive model for constructing a skeleton of a shape is given by [10]. In this article the analogy of a grass fire propagating as wave fronts is used to construct the skelton. In the analogy the 2D space is seen as grass field. The shape for which the skeleton shall be calculated describes locations where a grass fire simultaneously starts. The skeleton is defined as the locations where two or more fire fronts propagating through the 2D space meet. The algorithm introduced by [10] is rather complex in terms of computational and implementation costs. An approximate solution by which the the 2D space is discretized into a grid like structure is given by [9]. An interesting observation is that the skeleton if a polygon is totally contained in its set of Voronoi edges [11].

A similar procedure is used in this work. However, the real skeleton of an 2D environment often results in a graph where edges are parabolas. In the underlying 2D simulation model it is required that edges are straight line segments. Certainly, any parabola can be approximated with arbitrarily small error by a set of straight line segments. However, such an approach would lead to rather complex graphs. Therefore some heuristic procedures are used to simplify the skeleton. The graph generation works as follows.

- Construction of the skeleton using a procedure similar to [11]. An example of such a skeleton is given in Figure 3.

- Replacing the parabola edges between the vertices in the graph with straight line segments, but only for those edges where the resulting straight line segment does not intersect any geometry in the 2D environment. An example for a resulting intermediate graph is given in Figure 4.

- Contraction of edges under a certain length, again only if the resulting edges do not intersect any geometry in the 2D environment. An example for a resulting graph is given in Figure 5.

Force model

This sections discusses the force based agent movement model. We distinguish between two stages in the movement model. The first stage deals with the low level movement of the agents, meaning collision avoidance, velocity adaptation and so on. The second stage deals with the more high level movement of the agents, meaning moving along a given route. Both stages are modeled based on additive attracting and repelling forces, which are "pushing" the agents through the environment. The general force model is defined according to Newton's law ($F = \mathrm{m} \cdot \mathrm{a}$).

$$\mathrm{m}_i \cdot \mathbf{a}_i(t) = \frac{\mathrm{m}}{\tau}(\mathbf{v}_i^0(t) - \mathbf{v}_i(t)) + \sum_{j \neq i} \mathbf{f}_{i,j}(t), \tag{1}$$

with $\mathbf{v}_i^0$ is the desired velocity vector for agent i at time t. The term $\mathbf{v}_i(t)$ denotes the agent's actual velocity at time t. The time constant τ describes the time that is needed to adjust the actual velocity to the desired velocity. The second term of the equation builds the sum over all influential entities j in the environment (i.e. other agents, walls, and obstacles). Each of those entities emit a repelling (or attracting) force to agent i.

Many different force models have been discussed in recent years (see, e.g. [12] for an overview), most of them are build on the so called social force model introduced by [5]. The basic social force model implicitly reproduces collision avoiding behavior as it can be observed in real-world situations. At has been shown that the model works particularly well in high density conditions, such as one can observe in evacuation situations [13]. In this work we adapted an extension to the social force model where collisions are explicitly avoided to the current context. Collisions are avoided by explicitly predicting potential collision points [14].

In the model each agent i computes for each other agent j in the environment the angle $\theta_{i,j}$ between $\mathbf{d}_{i,j}$ and $\mathbf{v}_{i,j}$. Where $\mathbf{d}_{i,j}$ denotes the vector pointing from agent i's position to agent j's position and $\mathbf{v}_{i,j}$ denotes the relative velocity between both agents. If $|\theta_{i,j}| > \pi/4$ then the minimal distance time $t_{i,j} = \infty$. Otherwise $t_{i,j}$ reflects the true minimal distance time which is the time when the distance of both agents is minimal by assuming that neither agent i nor agent j changes her velocity or direction of movement. The agent than takes the minimum of these times $t_i = \min_j(t_{i,j})$. Afterwards the agent computes the configuration of the environment for t_i by again assuming that non of the pedestrians changes her velocity

or direction of movement. Let $\mathbf{d}'_{i,j}(t_i)$ denote the predicted vector pointing form agent i to agent j at time t_i. The agent j's influence on agent i (second term in Equation 1) is

$$\mathbf{f}_{i,j}(t) = \mathtt{A}_{env} \frac{v_i(t)}{t_i} \mathtt{e}^{-\mathbf{d}_{i,j(t)}/\mathtt{B}_{env}} \frac{\mathbf{d}'_{i,j}(t_i)}{d'_{i,j}(t_i)}. \tag{2}$$

The constants $\mathtt{A}_{env}$ and $\mathtt{B}_{env}$ are free parameters in the model. Equation 2 is not only applicable for other agents in the environment but also for any non moving object like walls or obstacles. The collision avoiding model describes how the environment influences the agents' movement. However, in order to move through the environment along a given route a "driving force", like the velocity adaption term in Equation 1, is needed.

In the current setup every agents start at a link in the navigation graph. A simple approach would be to let $\mathbf{v}_i^0$ point towards the to-node of the start link at the beginning. As soon as the node is reach $\mathbf{v}_i^0$ points to the to-node of the next link and so on until the agent reaches her destination. However, in [15] it has been shown that such an approach leads to an unrealistic behavior in situations when agents are moving next to each other. The reason is that those agents are pulled together at close range to a node and after the node is passed their trajectories diverge again. The authors proposed a force system that follows the route in the navigation graph. The basic idea is that each agent keeps a shadow tag on the navigation graph, which moves along the graph as the agents move forward. Furthermore, the agents are connected by a virtual rubber strap to their corresponding shadow tag. The agents are driven by a driving force that works parallel to the link which has the shadow tag on it. The virtual rubber strap pulls the agents towards the shadow tag if they diverge to

much from the link. This leads to the following force:

$$\frac{\mathrm{m}_i}{\tau}(\mathbf{v}_i^0(t) - \mathbf{v}_i(t)) + \mathtt{A}_{path}\mathrm{e}^{d_i^p(t)/\mathtt{B}_{path}}\mathbf{d}_i^p(t), \tag{3}$$

where $d_i^p(t)$ is the agent i's distance to the current link, and $\mathbf{d}_i^p(t)$ is the perpendicular unit vector pointing from the agent to the current link.

Sticking all together the agents' acceleration at time t is given by:

$$\mathbf{a}_i(t) = \frac{1}{\tau}(\mathbf{v}_i^0(t) - \mathbf{v}_i(t)) + \frac{1}{\mathrm{m}_i}(\mathtt{A}_{path}\mathrm{e}^{d_i^p(t)/\mathtt{B}_{path}}\mathbf{d}_i^p(t) + \sum_{j\neq i}\mathtt{A}_{env}\frac{v_i(t)}{t_i}\mathrm{e}^{-\mathbf{d}_{i,j(t)}/\mathtt{B}_{env}}\frac{\mathbf{d}'_{i,j}(t_i)}{d'_{i,j}(t_i)}). \tag{4}$$

This leads to a model with four free parameters ($\mathtt{A}_{env}$, $\mathtt{B}_{env}$, $\mathtt{A}_{path}$, and $\mathtt{B}_{path}$).

In the rubber strap analogy discussed above the agent would switch to the next link as soon as the shadow tag moves over the node. This means, at the end of each link is virtual perpendicular finish line and as soon as the agent crosses that finish line, the shadow tag is assigned to the next link. However, as it has been pointed out in [16], this approach results in an artifact if the angle between two consecutive links is smaller than $\pi/2$. In those cases the agents will move toward the inner curve of the path, which is not plausible. The proposed solution that is implemented her is that the finish line is not be perpendicular to the link but it corresponds to the bisector of the angle between both links.

The model for the agent movement defined in Equation 4 defines the agents' acceleration in a continues time model. However, in order to integrate the model into a computer simulation the time has to be discretized. This means an agent's velocity will only be updated at discrete time steps. There are different ways how to update velocities at discrete

time steps. A common used way is the Euler-Cromer method (see, e.g. [17]), which ends up in the following equations:

$$\mathbf{v}_i(t+h) = \mathbf{v}_i(t) + h\mathbf{a}_i(t) \tag{5}$$

$$\mathbf{r}_i(t+h) = \mathbf{r}_i(t) + h\mathbf{v}_i(t+h) \tag{6}$$

Where $\mathbf{r}_i(t)$ is the agent i's position at time t and h is the time step size.

Agent learning

The previous section discussed the agent movement model, which allows the agents to move along a given route. However, the route generation has not been discussed so far. In this section a method to generate reasonable routes will be discussed briefly. A detailed discussion on this matter is given in [18].

A straight forward solution is the shortest path solution, where every agent choses the shortest path from her respective origin to the desired destination. However, the shortest path solution does not necessarily lead to a good or realistic solution. The reason is that the shortest path solution does not take congestion into consideration and does therefore lead to rather long travel times in the simulation. Therefore an iterative learning algorithm has been implemented to find better solutions. Iterative learning means that the simulation is repeated many times while the agents are trying to find better routes (e.g. in terms of travel time). This is a common approach in transport science (see, e.g., [19] for an application). In later work the agent model has been extended in a way that each agent can hold several plans in her

memory in order to revise older plans [20]. The basic iterative learning algorithm works as follows:

1 Initialize $\tau_l(k)$ with free speed travel time for all links l and time steps k.

2 Calculate least cost paths for all agents i according to $\tau_l(k)$.

3 Repeat for many iterations:

  a Run one simulation iteration.

  b Extract time-dependent link travel time $\tau_l(k)$ for all l and k

  c For every agent $i = 1 \ldots N$, do:

    - With $P_{reroute}$: Compute a new route from $s(i)$ to $t(i)$ based on the experienced travel time $\tau_l(k)$ from the previous iteration

    - With P_{select}: Select an already tested route out of i's memory

The free speed travel time $\tau_l(k)$ is the link travel time that would be reached if a single agent moves along the link with her desired velocity. This means the free speed travel time is agent dependent unless all agents have the same desired velocity. An important aspect is that the travel time is time-dependent, i.e. the travel time can vary for a link depending on its time-dependent utilization. However, even small changes in the agents' behavior can lead to strong fluctuations in the time-dependent travel times from one iteration to the next. This would lead to a bad learning behavior. For that reason the experienced travel times are smoothed using the method of successive averages (MSA, see [21]). MSA is an inductive

definition of the mean value for a set of measurements. Let $\tau_{k,i}(k)$ denote the experienced time-dependent link link travel time for link l, time step k in learning iteration i that has been smoothed with MSA. The MSA smoothed time-dependent travel times are calculated by

if $i == 0$ **then** $\tau_{l,i}(k) = \tau_l(k)$

if $i > 0$ **then** $\tau_{l,i} = i/(i+1) * \tau_{l,i-1}(k) + 1/(i+1) * \tau_l(k)$.

For the sake of simplicity the iteration index i is omitted and the term $\tau_l(k)$ referrers to the MSA smoothed time-dependent link travel time for link l and time step k in the corresponding iteration.

At the end of each iteration the agents perform a re-planning procedure in order to find better routes. Every agent chooses between two re-planning strategies. With probability of $P_{reroute}$ an agent chooses the `ReRoute` strategy. The `ReRoute` strategy generates new routes based on the information of the experienced MSA smoothed time-dependent travel times from the last iteration.

The other strategy is called `ChangeExpBeta`, which is chosen with the probability of P_{select}. This strategy decides if the just performed plan should be used again, or if a random plan out of the memory should be selected for the next iteration. The probability to change the selected plan is calculated by:

$$P_{change} = min(1, \alpha * e^{\beta * (s(p_{random}) - s(p_{current}))/2}). \tag{7}$$

With:

- α: The probability to change if both plans have the same experienced travel time
- β: A sensitivity parameter
- $s(p_{\{random,current\}})$: The negated travel time of the current/random plan as it was experienced during its last execution.

If the system is "well-behaved", this set-up converges to a steady state where the probability that an agent uses plan p_i is

$$P(p_i) = \frac{e^{\beta * s(p_i)}}{\sum_j e^{\beta * s(p_j)}} \ , \tag{8}$$

i.e. the standard multinomial logit model (e.g. [22]).

Each strategy is selected with a certain probability. These probabilities are assigned before the simulation starts. Typically, `ReRoute` is called with a relatively small probability, say 10%, and `ChangeExpBeta` is called in the remaining cases. After re-planning every agent has a selected plan that will be executed in the next iteration.

Experiments

The simulation frame work is tested on a hypothetical evacuation scenario, where the office building depicted in Figure 1 has to be evacuated. For the evacuation three different exit doors can be used. In from of the exit doors there are meeting points where the evacuated

persons can gather. At the moment of the evacuation there are 164 persons (agents) in the building. The distribution of the agents and the locations of the meeting points are shown in Figure 6. The iterative learning algorithm described in the previous section has been performed for 20 iterations. In the underlying simple scenario with only 164 agents 20 iterations are sufficient to reach a relaxed state[2]. What is more, for every agent there exist only three different routes to reach one of the meeting points. For that reason the `ReRoute` strategy is not used in this scenario. Instead, each agent starts with three different evacuation routes in her memory and uses `ChangeExpBeta` to switch between them.

This setup converges to an average evacuation time of 42 seconds. A plot that shows the change in the average evacuation time over the iteration numbers is shown in Figure 7. The final meeting point selection for the individual agents is given in Figure 8. The results are as expected besides some artifacts, e.g. a yellow and a green agent in the right wing of the building. This artifacts are caused by the `ChangExpBeta` strategy, which for each agents chooses a random plan with a small probability. The simulation frame work is implemented in Java and the simulation run has been executed on a laptop computer with a 2.66 GHz Intel Core i7 processor. The overall runtime for 20 iterations was 3:29 minutes, where the pure simulation time was about 3 minutes and all other operations (i.e. data loading, re-planning, and out writing) took about 30 seconds. The memory consumption during the simulation was about 120 MB. These results show that the runtime and memory consumption is reasonable and leaves space to run large and more complex simulations.

[2]In general a much higher number of iterations is needed until the system is relaxed.

Conclusion

A 2D multi-agent based simulation framework has been introduced. Concepts of the framework are based on well known models like the social force model first introduced by [5] and transport simulation models as discussed in [20]. The novelty of the proposed model is the combination of common transport simulation models, which are not capable to do highly resolved 2D simulation, with social force models, which in turn are not designed to simulate complex wayfinding. The agents in the simulation navigate along a navigation graph. In earlier work it has been proposed to use the visibility graph as a navigation graph [6]. The visibility graph, however, is very complex even for simple environment and is, therefore, not suited for the current approach. The navigation graph in this work is a simplification of the skeleton which describes the shape of area. The skeleton has been automatically generated using a procedure similar to [11] and simplified afterwards using some heuristic approaches. This procedure leads to a much simpler navigation graph while being well suited to model wayfinding in complex environments.

The simulation framework has been tested on a simple evacuation scenario, where the individual agents try to optimize their evacuation routes. The results are feasible and the computing costs are reasonable. Therefore, the simulation framework is well suited for larger and more complex scenarios. In future work the simulation frame work will be applied to large real-world scenarios. Before those complex scenarios will be simulated, the model has to be calibrated and validated. It is planned to do this using data, such as pedes-

trian trajectories, from real-world experiments.

There are some other future research topics. First of all comes the extension of the re-planning module to allow the agents constructing plans that are more complex than the plans demonstrated in the simple evacuation scenario. Furthermore, through the flexibility of the simulation model other than pure force models can be used. An interesting approach for collision avoidance is so called reciprocal velocity obstacles approach [23].

Acknowledgements

This project was funded in part by the German Ministry for Education and Research (BMBF) under grant 13N11382 ("GRIPS") and by the German Research Fundation (DFG) under grant NA 682/5-1 ("Methods for modeling and large-scale simulation of multi-destination pedestrian crowds").

References

[1] H. Klüpfel, T. Meyer-König, A. Keßel, and M. Schreckenberg. Simulating evacuation processes and comparison to empirical results. In M. Fukui et al, editor, *Traffic and granular flow '01*, pages 449–454. Springer, Berlin Heidelberg New York, 2003.

[2] K. Nishinari, A. Kirchner, A. Nazami, and A. Schadschneider. Extended floor field CA model for evacuation dynamics. *IEICE Transactions on Information and Systems*,

E87-D(3):726–732, 2004.

[3] D. Helbing, I.J. Farkas, P. Molnar, and T. Vicsek. Simulation of pedestrian crowds in normal and evacuation situations. In M. Schreckenberg and S. D. Sharma, editors, *Pedestrian and Evacation Dynamics*, Proceedings of the 1st international conference, Duisburg, 2001, pages 21–58. Springer, 2002.

[4] D. Helbing, L. Buzna, A. Johansson, and T. Werner. Self-organized pedestrian crowd dynamics: experiments, simulations and design solutions. *Transportation Science*, 39:1–24, 2005.

[5] D. Helbing and P. Molnár. Social force model for pedestrian dynamics. *PRE*, 51(5):4282–4286, May 1995.

[6] C. Gloor, P. Stucki, and K. Nagel. Hybrid techniques for pedestrian simulations. In P. Sloot, B. Chopard, and A. Hoekstra, editors, *Cellular automata, Proceedings*, number 3305 in Lecture Notes in Computer Science, pages 581–590. Springer, 2004.

[7] E. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[8] M. de Berg, M. van Kreveld, M. Overmars, and M. Schwarzkopf. *Computational Geometry*. Spinger, 2000.

[9] P. Chaudhuri, R. Khandekar, D. Sethi, and P. Kalra. An efficient central path algorithm for virtual navigation. In *Proceedings of the Computer Graphics International*, pages 188–195, Washington, DC, USA, 2004. IEEE Computer Society.

[10] Harry Blum. A transformation for extracting new descriptors of shape. *Models for the Perception of Speech and Visual Form*, pages 362–380, 1967.

[11] D.T. Lee. Medial axis transformation of a planar shape. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-4(4):363 –369, july 1982.

[12] R. Oleson, D.J. Kaup, T.L. Clark, L.C. Malone, and L. Boloni. Social potential models for traffic and transportation. In A.L.C. Bazzan and F. Klügl, editors, *Multi-agent systems for traffic and transportation engineering*, chapter VII, pages 155–175. Information Science Reference (IGI Global), 2008.

[13] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, 407:487–490, 2000.

[14] F. Zanlungo, T. Ikeda, and T. Kanda. Social force model with explicit collision prediction. *EPL (Europhysics Letters)*, 93(6):68005, 2011.

[15] C. Gloor, L. Mauron, and K. Nagel. A pedestrian simulation for hiking in the alps. In *Proceedings of the Swiss Transport Research Conference (STRC)*, Monte Verita, CH, 2003. See www.strc.ch.

[16] C. D. Gloor. *Distributed Intelligence in Real World Mobility Simulations*. PhD thesis, ETH - SWISS FEDERAL INSTITUTE OF TECHNOLOGY ZURICH, 2005. thesisType=phd; department=Technical Sciences;.

[17] W. Cheney and D. Kincaid. *Numerical Mathematics and Computing*. Thomson, fifth edition, 2004.

[18] G. Lämmel. *Escaping the Tsunami: Evacuation Strategies for Large Urban Areas. Concepts and Implementation of a Multi-Agent Based Approach*. PhD thesis, TU Berlin, 2011.

[19] C. Gawron. An iterative algorithm to determine the dynamic user equilibrium in a traffic simulation model. *International Journal of Modern Physics C*, 9(3):393–407, 1998.

[20] B. Raney and K. Nagel. Iterative route planning for large-scale modular transportation simulations. *Future Generation Computer Systems*, 20(7):1101–1118, 2004.

[21] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[22] M. Ben-Akiva and S. R. Lerman. *Discrete choice analysis*. The MIT Press, Cambridge, MA, 1985.

[23] J. van den Berg, Ming Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1928 –1935, may 2008.

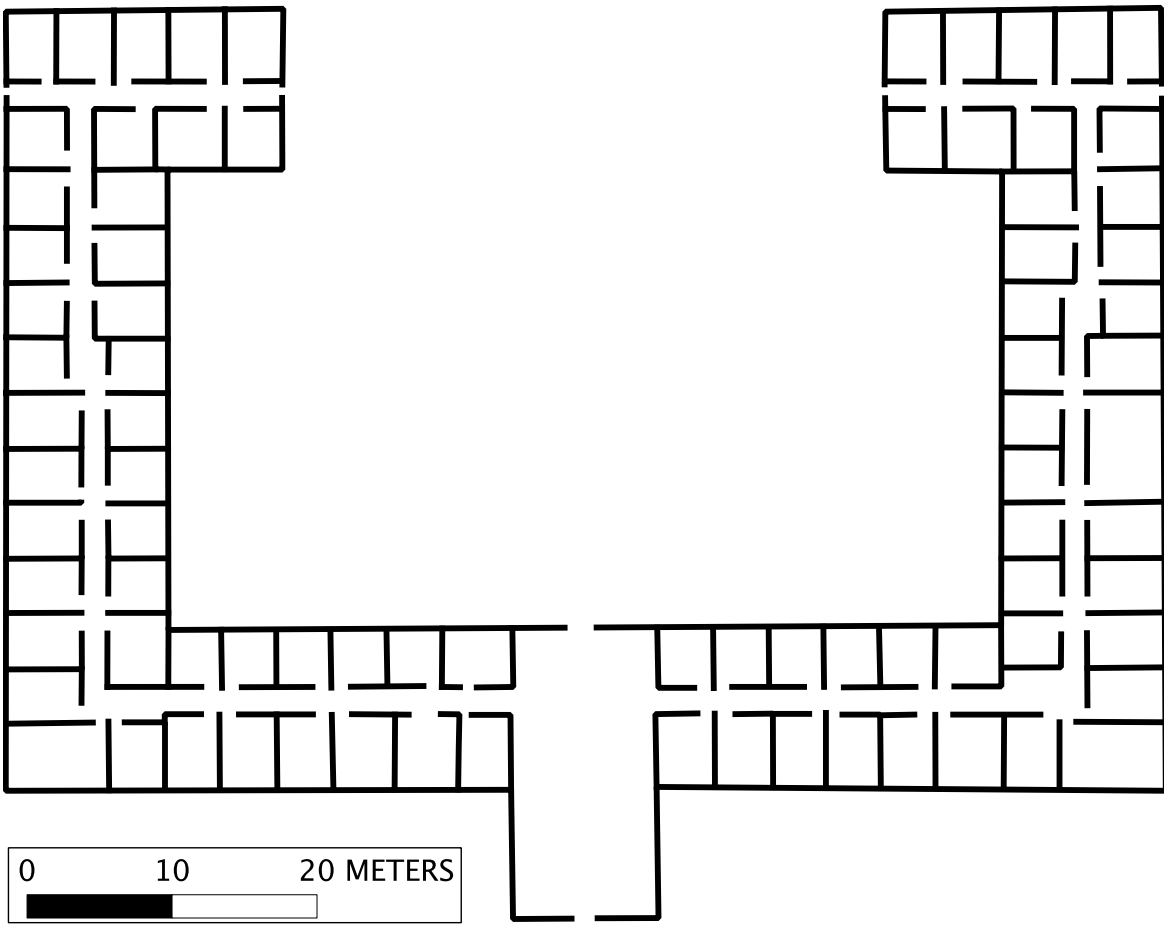

Figure 1: Floor plan of a hypothetical office buildings ground floor used in the simulation.

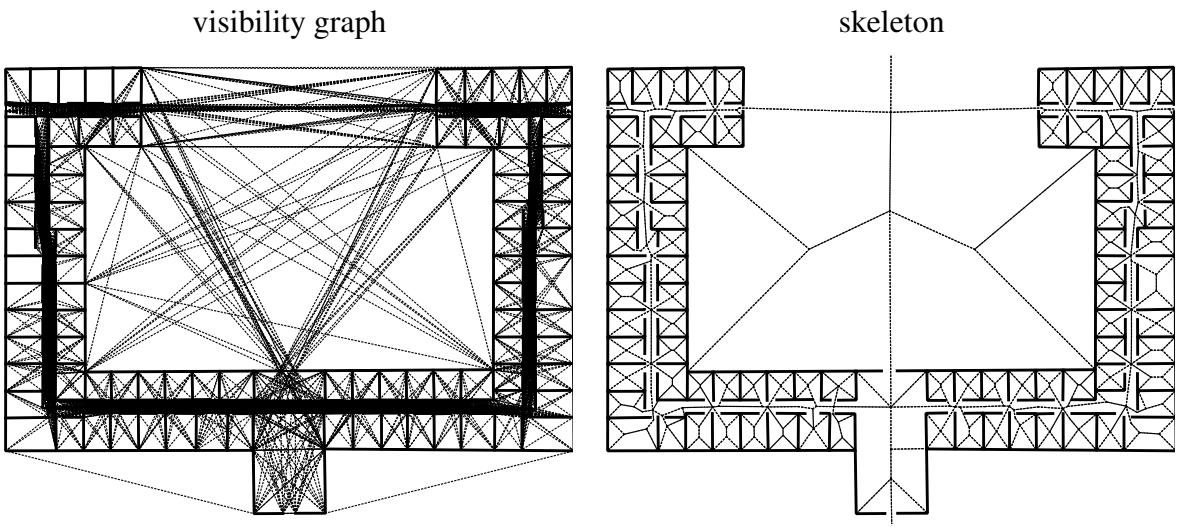

Figure 2: Comparison of the visibility graph approach and the skeleton approach. The visibility graph shown on the left side consists of almost 8000 edges, whereas the skeleton approach on the right side only needs about 900 edges.

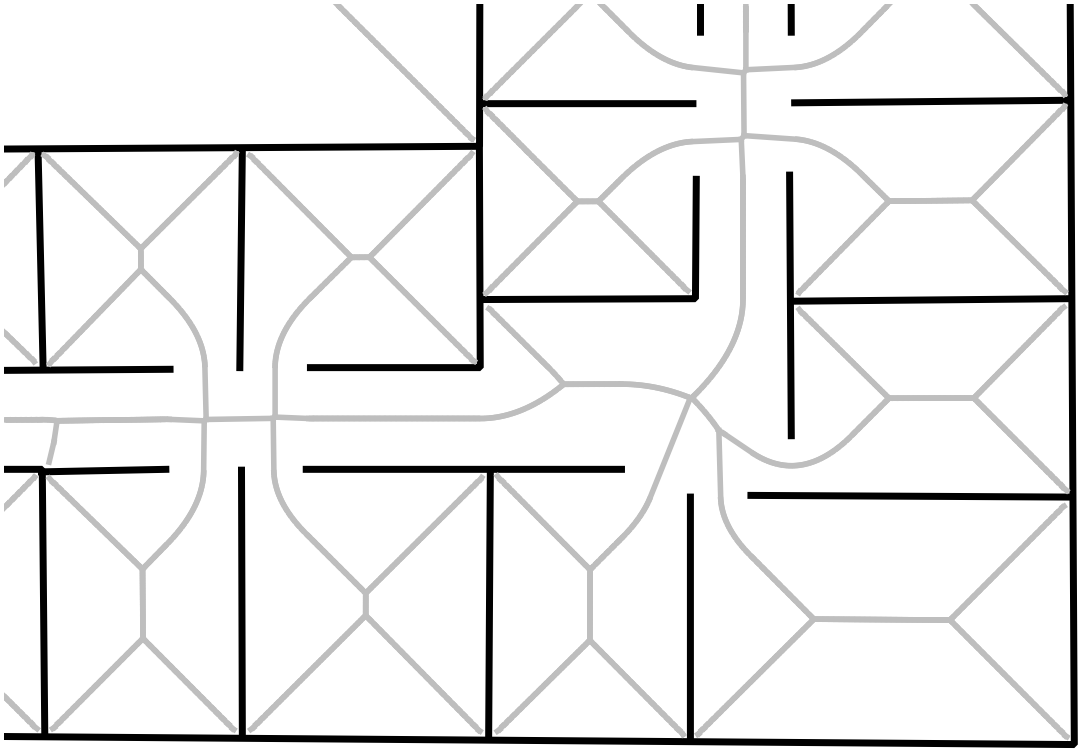

Figure 3: Skeleton of the buildings floor plan generated from a Voronoi diagram using a procedure similar to [11].

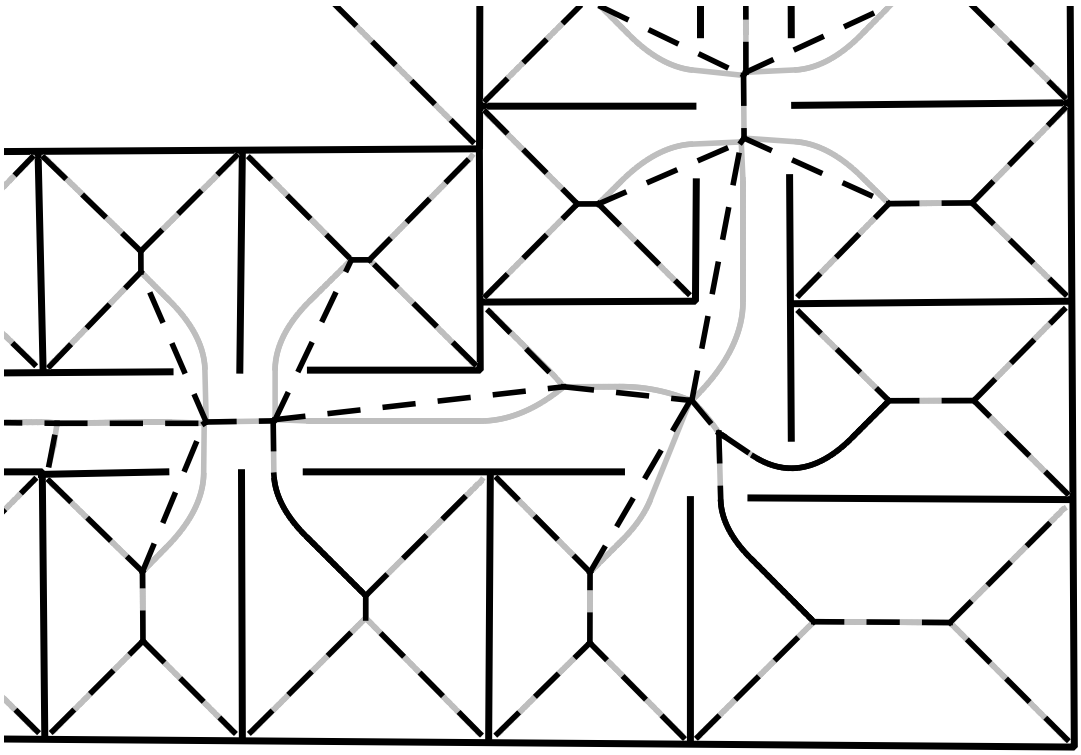

Figure 4: Simplified skeleton where parabolas are replaced with straight lines if possible.

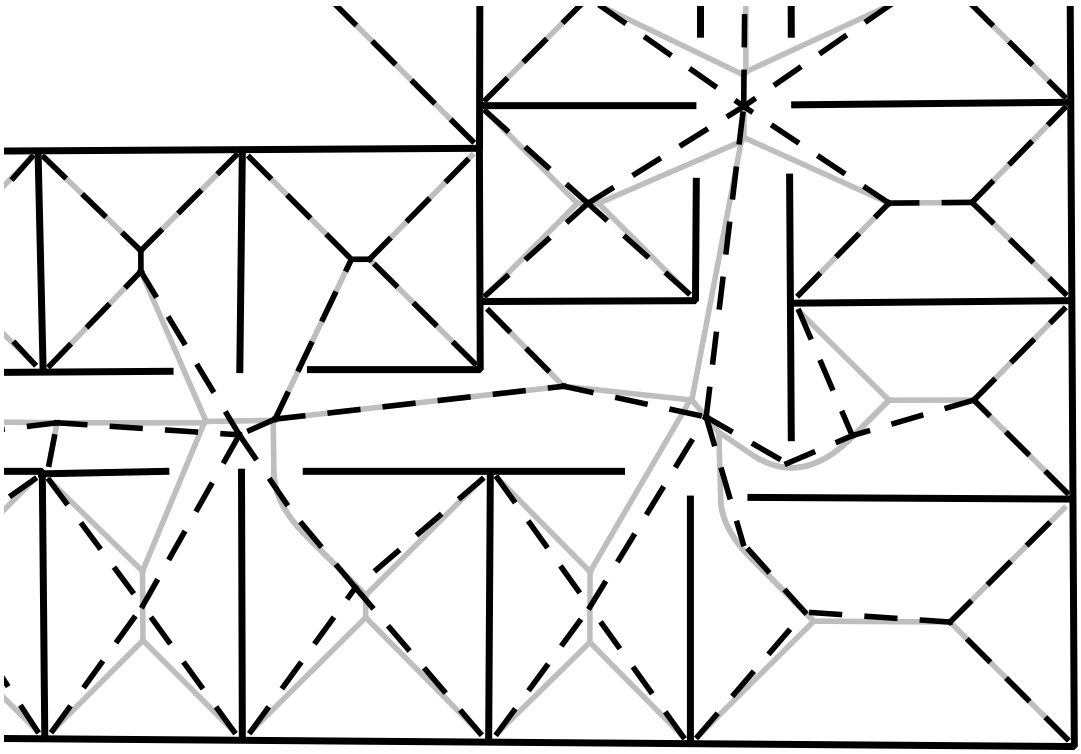

Figure 5: Final result of the skeleton simplification. In the last step short edges a contracted if the resulting graph does not intersect any geometry in the 2D environment.

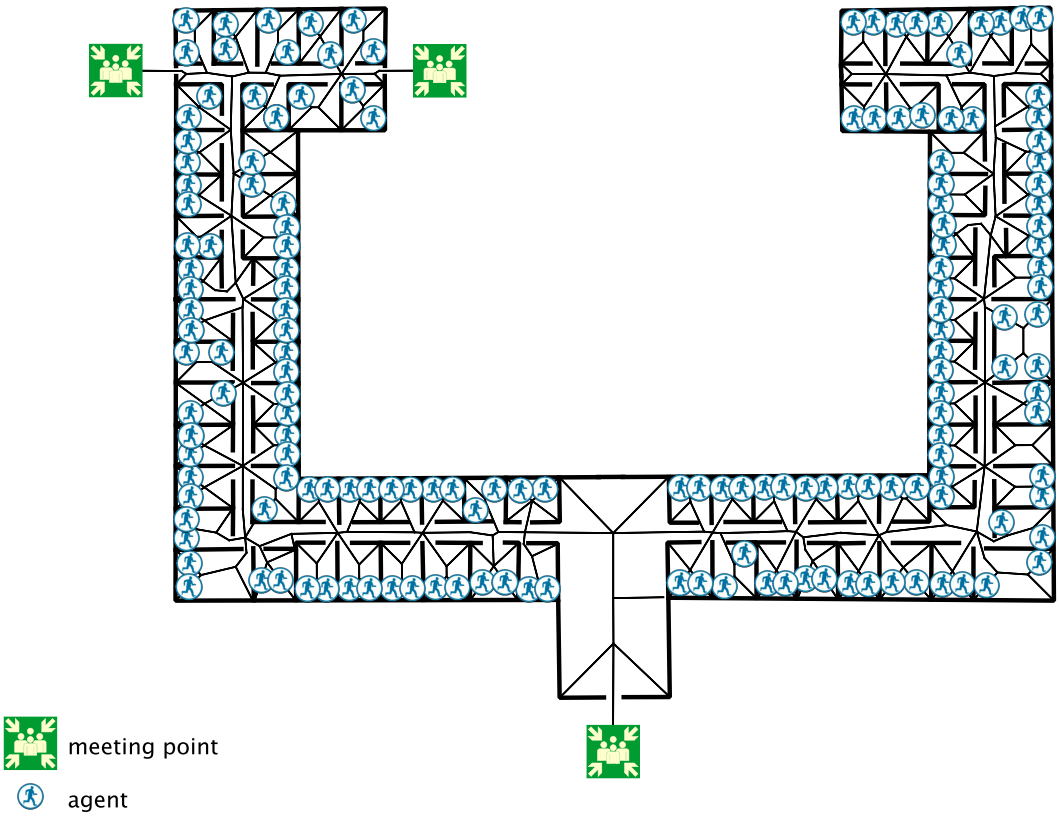

Figure 6: Illustration of the evacuation scenario. The agents can choose between three different meeting points.

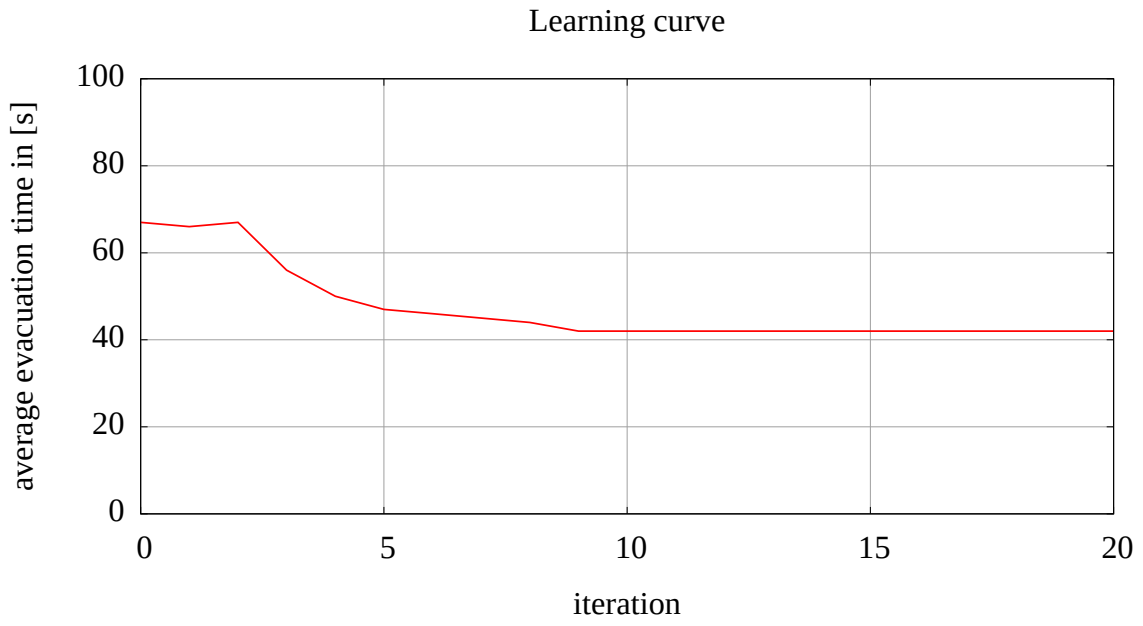

Figure 7: Plot of the average evacuation time over the iteration number. The average evacuation time converges to 42 seconds. Since the change in evacuation time over the iterations corresponds to the learning success of the agents we call this also learning curve.

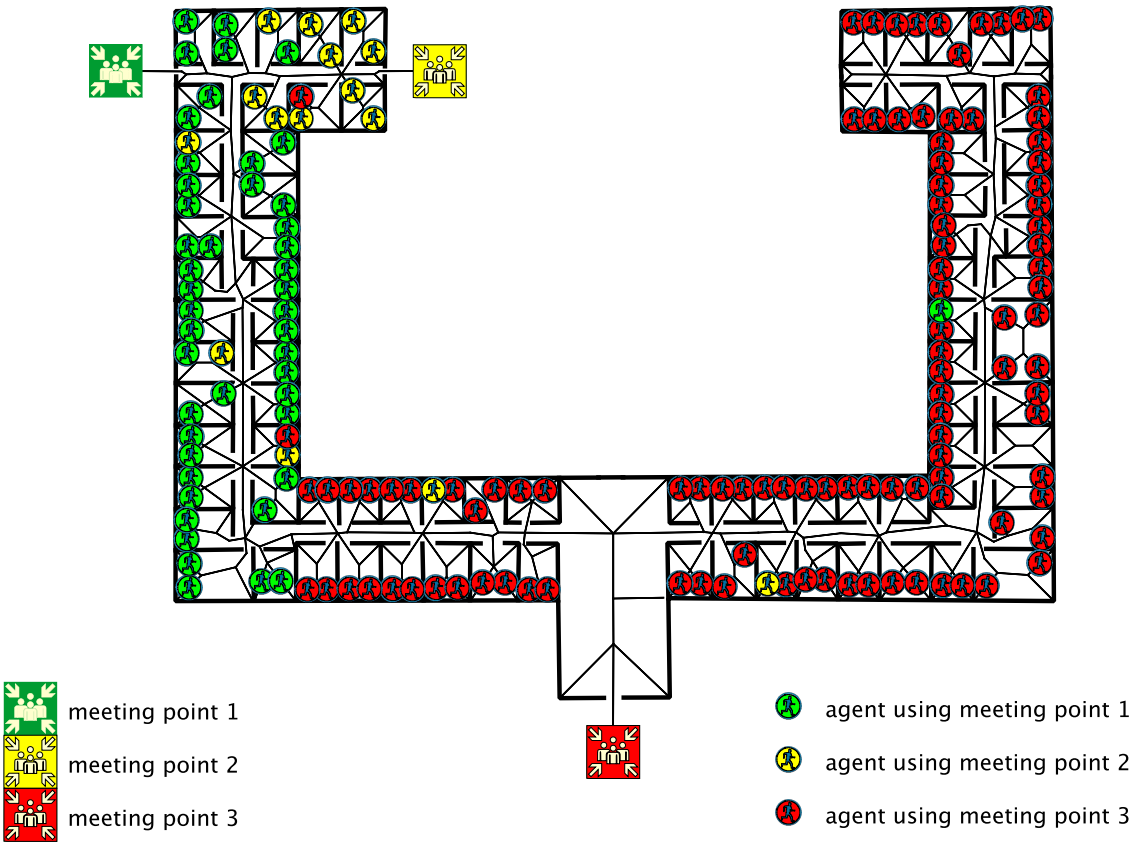

Figure 8: Figure showing the meeting point choice for each agent after 20 iterations of learning. Same color means same meeting point. The results are as expected besides some artifacts, e.g. a yellow and a green agent in the right wing of the building. This artifacts are caused by the `ChangExpBeta` strategy, which for each agents chooses a random plan with a small probability.