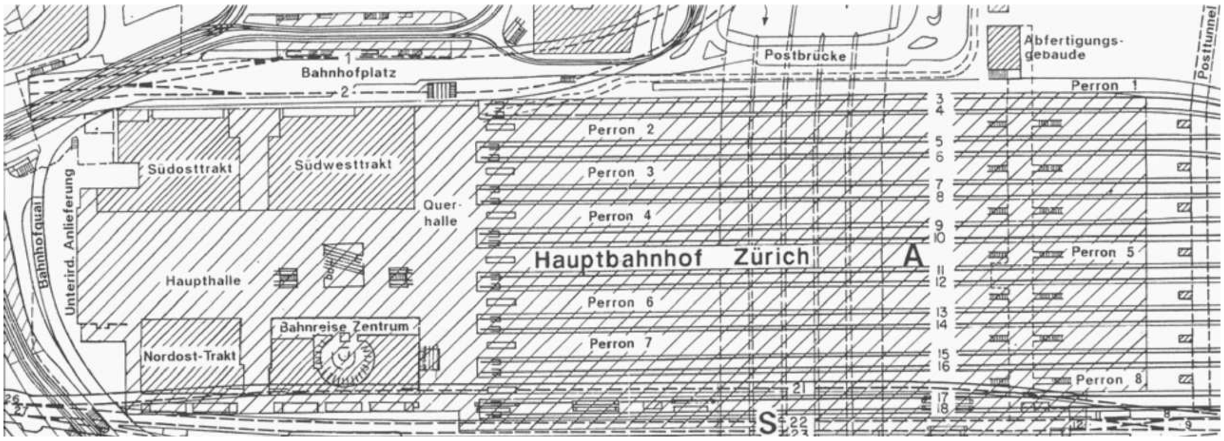
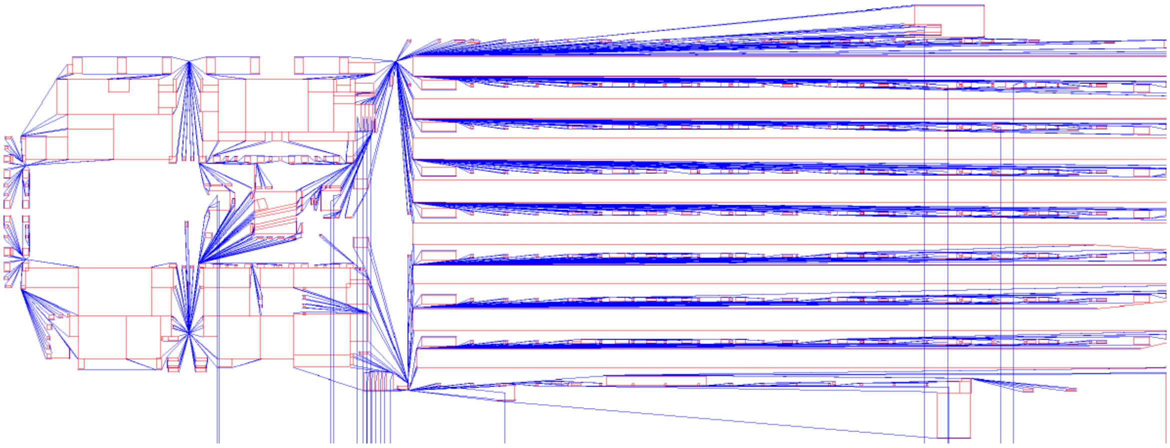




Hybrid techniques for pedestrian simulations

Christian Gloor, Dept. of Computer Science, ETH Zürich
Pascal Stucki, Dept. of Computer Science, ETH Zürich
Kai Nagel, Dept. of Computer Science, ETH Zürich

STRC 04 Conference paper

STRC | **4th Swiss Transport Research Conference**
Monte Verità / Ascona, March 25-26, 2004

Hybrid techniques for pedestrian simulations

Christian Gloor
Dept. of Computer Science
ETH Zürich
8092 Zürich, Switzerland

Phone: 01-632 04 32
Fax: 01-632 13 74
eMail: chgloor@inf.ethz.ch

Pascal Stucki
Dept. of Computer Science
ETH Zürich
8092 Zürich, Switzerland

eMail: stuckip@student.ethz.ch

Kai Nagel
Dept. of Computer Science
ETH Zürich
8092 Zürich, Switzerland

Phone: 01-632 54 27
Fax: 01-632 13 74
eMail: nagel@inf.ethz.ch

Abstract

There exist multiple models for pedestrian simulations. Cell based models are easy to understand, fast, but consume a lot of memory once the scenario becomes larger. In models based on continuous space, which need almost no memory at all, however, the CPU becomes the bottleneck soon.

In our project "Planning with Virtual Alpine Landscapes and Autonomous Agents", we simulate an area of 150 square kilometers, with more than thousand agents for one week. Every agent is able to move freely, adapt to the environment and make decisions during run time. This decisions are based on perception and communication with other agents.

This requires a simulation model that is fast and still fits into main memory of a typical workstation. We combined the advantages of both approaches into a hybrid model. This model exploits some of the special properties of the area.

- Hikers tend to walk on trails. It is possible to fit a coordinate system on a graph of these trails. Using this coordinate system, a continuous simulation is possible.
- Obstacles like houses, trees, or rivers influence the route choice of hikers. We developed an algorithm which adds additional nodes to the exiting graph for each obstacle. The hiker is not only able to walk around the obstacles. but also to take the path length into account during trip planning.
- Paths in the Alps are not like streets, their walkability differs a lot. The speed of the hikers is influenced by the quality of the trail. We added a grid known from cell based simulations, which allows us to control the speed of the hikers. Parts of this grid are dynamically loaded as needed.

This paper will present an overview over this hybrid system, and some performance results.

Keywords

Pedestrian Dynamics – Multi Agent Simulation – Parallel Computing – 4th Swiss Transport Research Conference – STRC 04 – Monte Verità

1. Introduction

The project "Planning with Virtual Alpine Landscapes and Autonomous Agents" (ALPSIM www page, accessed 2004) uses a multi-agent simulation to model the activities of tourists (primarily hikers). The goal is to have these agents populate a virtual world, where they are able to evaluate different development scenarios. Such scenarios include the question of re-forestation of meadows, or the summer use of chair lifts and the like. Left to themselves, many areas in the Swiss Alps would be covered by dense forest; it seems however that most hikers would prefer a more variable landscape. Many people, in particular families with children or people with health limitations, like mechanical aids to bring them nearer to the top of mountains.

The aim of this project is to implement a multi-agent simulation of tourists hiking in the Alps in order to investigate the achievable level of realism. At the same time, the project is used to explore general computational implementations of mobility simulations.

Such a simulation generically consists of two components: The physical mobility simulation, which moves the hikers through the system and computes their interactions; and the strategy generation module(s), which compute(s) strategic decisions of the agents such as destination or route choice. For the simulation, care needs to be taken that the agents explicitly react to visual stimuli; for example, they cannot look through a mountain.

Our approach is adapted from one used in traffic microsimulations. A synthetic population of tourists is created that reflect current (and/or projected) visitor demographics. These tourists are given goals and expectations that reflect existing literature, on-site studies, and, in some cases where sufficient data is not available, are based on experts' estimates. These expectations are individual, meaning that each agent could potentially be given different goals and expectations.

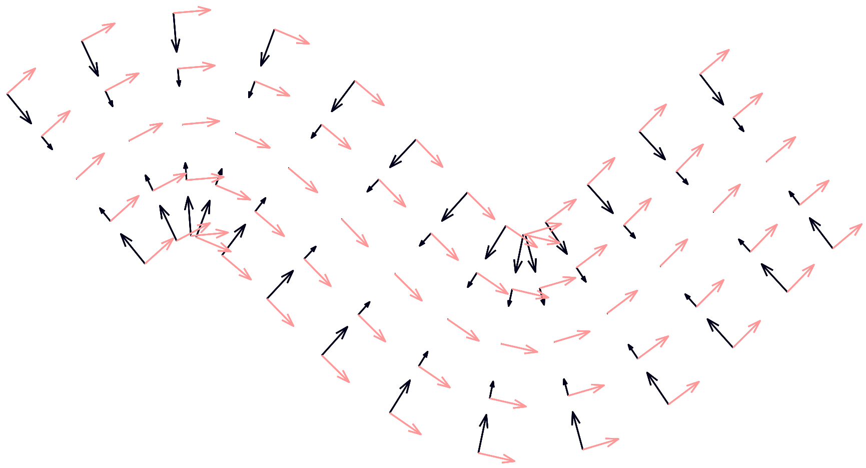
These agents are given "plans", and they are introduced into the simulation with no "knowledge" of the the environment. The agents execute their plans, receiving feedback from the environment as they move throughout the landscape. At the end of each run, the agents' actions are compared to their expectations. If the results of a particular plan do not meet their expectations, on subsequent runs the agents try different alternatives, learning both from their own direct experience, and, depending on the learning model used, from the experiences of other agents in the system.

A "plan" can refer to an arbitrary period, such as a day or a complete vacation period. As a first approximation, a plan is a completely specified "control program" for the agent. It is, however, also possible to change parts of the plan during the run, or to have incomplete plans, which are completed as the system goes.

After numerous runs, the goal is to have a system that, in the case of a status quo scenario, reflects observed patterns in the real world. In this case, this could, for example, be the observed distribution of hikers across the study site over time.

An introduction to possible techniques for pedestrian simulation can be found in (Schreckenberg and Sharma, 2001; Galea, 2003). For microscopic simulations, there are essentially two techniques: methods based on coupled differential equations, and cellular automata (CA) models. In

Figure 1: Path-oriented coordinate system for the computation of the desired velocity and the path forces. The light arrows show the desired velocity, which drives the agent forward along the path. The dark arrows show the path force, which pull the agent toward the middle of the path.



our situation, it is important that agents can move in arbitrary directions without artifacts caused by the modeling technique, which essentially rules out CA techniques. A generic coupled differential equation model for pedestrian movement is the social force model (Helbing *et al.*, 2000)

$$m_i \frac{d\mathbf{v}_i}{dt} = m_i \frac{\mathbf{v}_i^0 - \mathbf{v}_i}{\tau_i} + \sum_{j \neq i} \mathbf{f}_{ij} + \sum_W \mathbf{f}_{iW} \tag{1}$$

where m_i is the mass of the pedestrian and $\mathbf{v}_i$ its velocity. $\mathbf{v}_i^0$ is its desired velocity; in consequence, the first term on the RHS models exponential approach to that desired velocity, with a time constant τ_i. The second term on the RHS models pedestrian interaction, and the third models interaction of the pedestrian with the environment.

The specific mathematical form of the interaction term does not seem to be critical for our applications as long as it decays fast enough. Fast decay is important in order to cut off the interaction at relatively short distances. This is important for efficient computing, but it is also plausible with respect to the real world: Other pedestrians at, say, a distance of several hundred meters will not affect a pedestrian, even if those other pedestrians are at a very high density. We use an exponential force decay of

$$\mathbf{f}_{ij} = \exp\left(\frac{|\mathbf{r}_i - \mathbf{r}_j|}{B_p}\right) \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|} , \tag{2}$$

which seems to work well in practice. $\mathbf{f}_{ij}$ is the force contribution of agent j to agent i; $\mathbf{r}_i$ is the position of agent i. Alternative more sophisticated formations are described by Helbing *et al.* (2000). For the environmental forces, $\mathbf{f}_{iW}$, the same mathematical form as for the pedestrian-pedestrian interaction is used.

We introduced (Gloor *et al.*, 2003b) a model that uses only sparse information which fits into computer memory, runs efficiently on our scenarios, and has agents follow paths without major artifacts. Our model uses a path-oriented coordinate system (see Fig. 1) for the computation of the desired velocity. This model also uses a so-called **path-force**, which pulls the agents back on the path when he moves away from its center (e.g. due to interaction with other agents or obstacles).

Using these equations, a pedestrian without an own intention can be modelled. Simple intentions, like walking into a certain direction, can be described by simple expressions for v_i^0. In a more complex scenario, however, pedestrians tend to follow a path or avoid obstacles that where predictable. It is even possible to think of an simulation of pedestrian that moves inside a city or inside a building.

Usually the curent position $\mathbf{r}_i$ and the eventual destination of the pedestrian is known. The v_i^0 is given by the optimal trajectory.

In this paper, we look at two completely different methods to calculate v_i^0. The resulting trajectory, however, is not always the one the pedestrian will walk on eventually, since the other term of the RHS of equation (1) have an influence as well. This paper is concluded by a look at an example simulation, an evacuation of Zürich Main Station.

2. Potential Field Model

The first approach looked at in this paper is based on the **utility maximization** model by Hoogendoorn *et al.* (2002). For this model, a potential field is generated for the simulated area, which allows the pedestrians to find their destinations by walking towards the minimal potential.

Instead of using partial differential equations, the potential field can be created by calculationg the distance to a given destination for each point of the walking area. If the influence of possible obstacles is not neglected, a potential field as shown in Figure 8 emerges. Note that for each possible destination a separate potential fiels needs to be created. However, this field can be used for all agents heading to that destination, regardless of their starting position.

The potential fields are calculated *before* the actual simulation is started. For this, the simulated area has to be divided into **cells**, each of them describing the average value all positions covered by this cell. The actual values for each cell are calculated using a simple **flooding algorithm**, starting at the destination. This algorithm only updates cells that are not blocked by an obstacle. The results generated by this algorithm are close but not exactly similar to the ones generated by the utility maximization model.

Stucki (2003) descibes three different ways to implement this flooding algorithm. The most promising implementation, using a *priority queue*, is about 2000 times faster than an unoptimized, recursive implementation. An area of 50×50 cells is calculated in approximately $\frac{1}{10}$ seconds.

To deduce an agents' desired velocity, or direction, from this potential field is not as easy as expected. The first idea is to just walk into the direction of the neighbour cell with has a lower

Figure 2: Different algorithms to calculate the desired velocity (walking direction) from a given potential field.

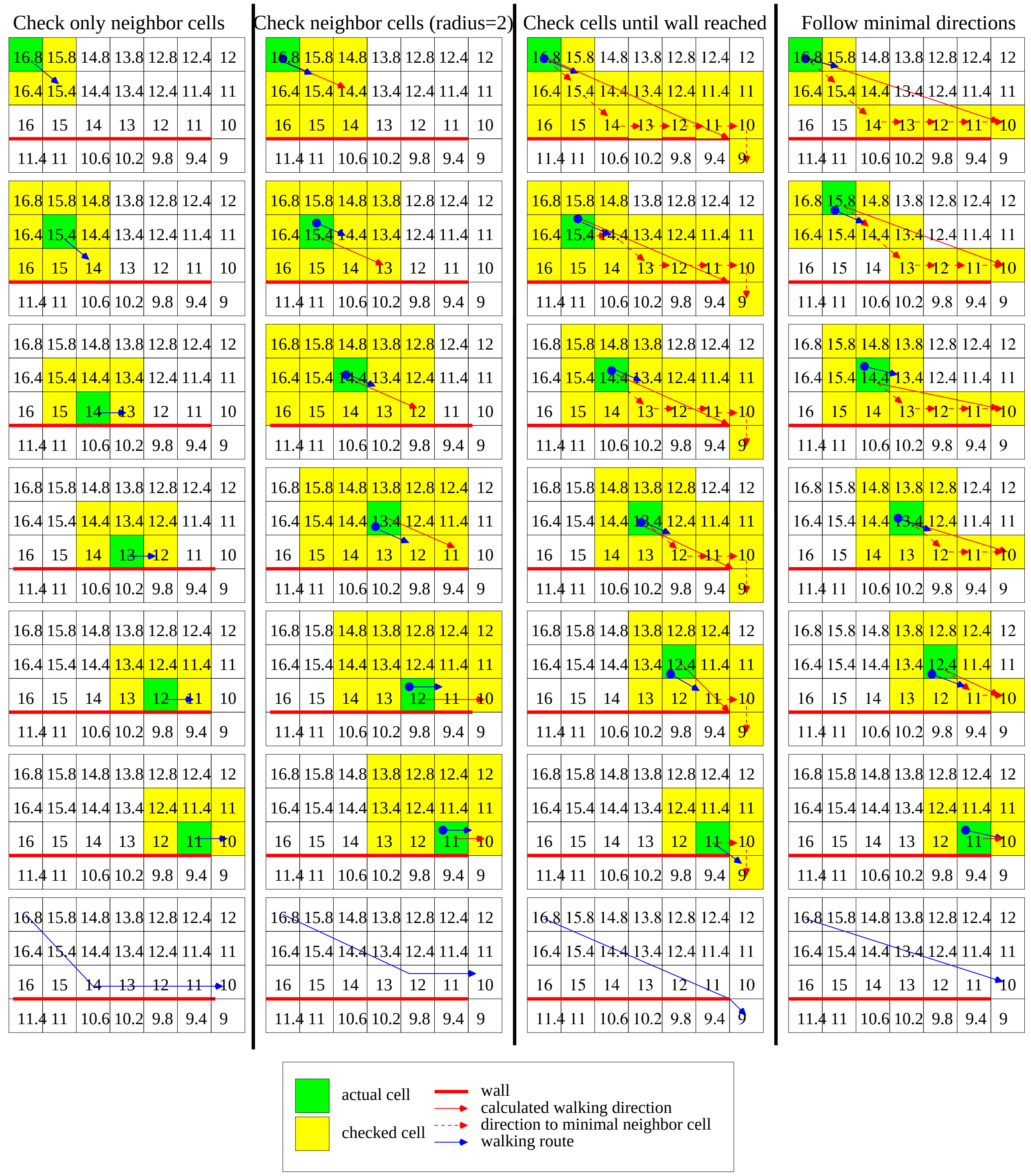
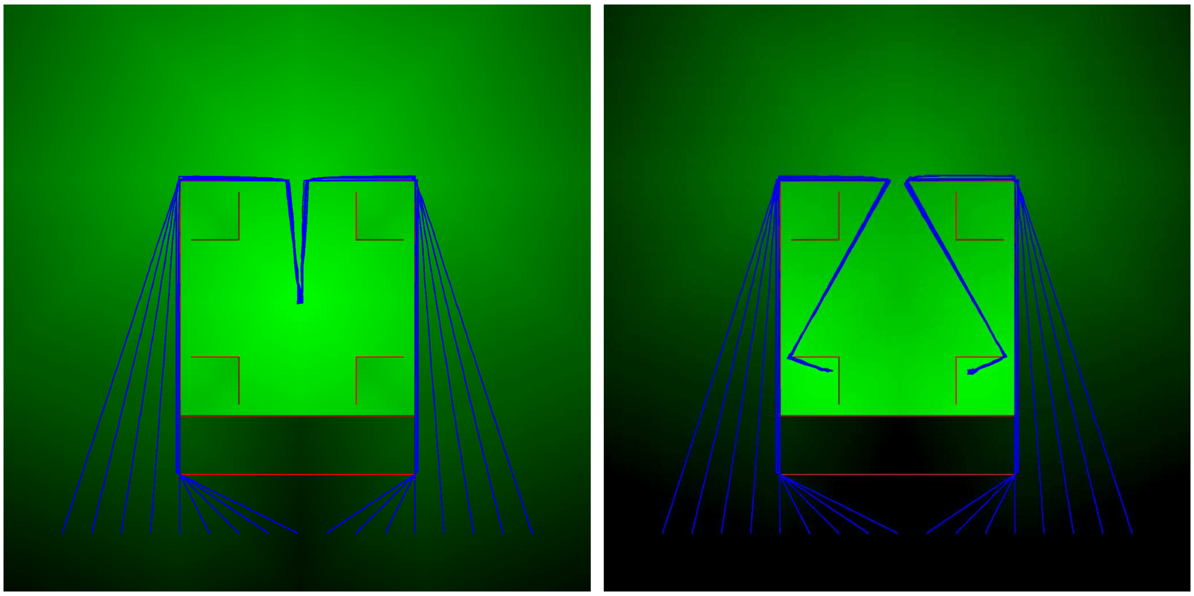


Figure 3: A potential field generated for one destination in the center of the formation (left) and for two destinations (right). A lighter color means a lower potential (close to the center, the potential is lowest). Some example walking directions derived from this potential field are shown in blue.



potential, and which is therefore closer to the destination. Using this method, however, yields in a zigzaging path, because the possible walking directions are limited by the number of neibour cells, which is eight (1^{st} column of Figure 2).

However, even by using the cell-based direction information, the agents are able to walk in any arbitrary direction. The pedestian dynamics are still computed according equation (1), but we use a v_i^0 that is deduced from the precomputed potential field.

One possibility to increase the number of possible walking directions is to increase the number of cells looked at. If cells not just one but two steps away are considered as well, 16 directions are possible (2^{nd} column of Figure 2).

An even more realistic result can be achieved if the algorithm continues checking cells in one direction until it reaches a wall. Then, it follows the minimal potential until it reaches the corner of that wall. This corner is in the direction the agent should walk (3^{rd} column of Figure 2). Alternatively, one can follow the minimal potential until the cell can no longer be reached directly from the starting cell (4^{th} column of Figure 2). This last two algorithms require knowledge of the position of objects in the simulated area.

It is possible to solve the problem of finding the optimal walking direction using the correct distances to the destination instead of using Manhattan distances (Nishinari *et al.*, 2001). This basically yields in a graph connecting all corners of every object. The pedestrians appear to walk along the edges.

3. Graph Model

For the ALPSIM project, we use a network of hiking paths in the Alps. This network has a resolution of approximately 2m, which is accurate enough for path through forests or meadows. However, inside villages or close to obstacles, a resolution of 25cm is needed to model a realistic behaviour of the pedestrians.

One possibility would be to switch to the just presented potential field model where needed. Since only v_i^0 is calculated by the potential field, all other aspects of the pedestrian dynamics, as forces between pedestrians or from obstacles, would remain the same.

However, a more intuitive solution is to keep the existing graph, but to add more details where needed. We were looking for a method that generated a graph around a given set of obstacles. This path can be merged with the global hiking path graph eventually.

In a first step, we have to decide where the **nodes** of the new graph should be. There should be enough nodes that a agent is able to circumvent obstacles on a naturally looking path, but not too much. Each additional node has to be considered in route choices and adds to path lenghts. Paths, which are list of nodes, are stored in agents or transmitted over the network.

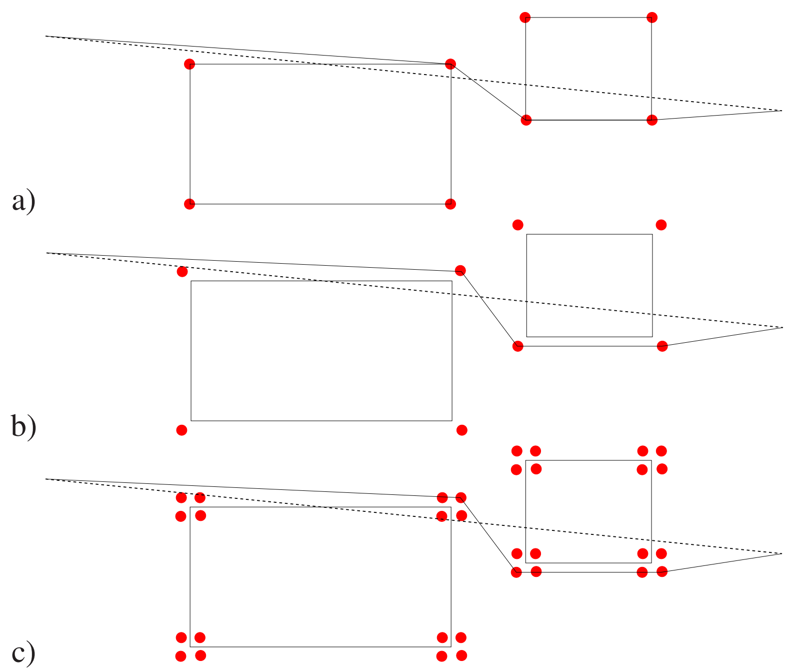
It is not that easy to find a simple yet realistic looking position of a node. The simplest solution would be to add a node to each corner of all objects (see Figure 4a). However, people tend to keep a certain distance to objects. Weidmann (1993) outlines that pedestrians keep an distance of 0.25 m (inside buildings) and 0.45 m (outdoors) to walls, even more to obstacles like fences. The paths would be to close to the objects. This would not be a problem, since agents as well keep a distance to obstacles due to environmental forces, but it would be better to avoid the problem in the graph directly.

The pedestrians will observe a distance if the nodes have a certain distance to the object corners as well. We decided to use multiple nodes for each corner of an object. Each of them in a distance of 0.25 m to the object corner (see Figure 4c). It does not matter in which direction the pedestrian approaches the corner, the distance to the corner is more or less equal. In order to reduce the numbers of nodes, we run an algorithm that eliminates nodes lying inside an object.

Based on these nodes an intial **graph** is constructed. At first, every node is connected to each other node, and each such edge has an weight of the distance between the nodes. We use then an algorithm to reduce the edges generated by fully connecting all the nodes geneated. We run an visibility check algorithm, known from e.g. the area of 3D computer graphics, to determine all the edges that intersect with obstacles. These edges are deleted.

In order to find the **shortest path** through this network, the Dijkstra (1959) shortest path algorithm is used. The implementation of this algoritm is *time dependant*. For traffic simulations, time dependant routes are primarily because of traffic jams, which is a feature implied by the agents

Figure 4: a) The simplest solution to place nodes would be to add a node to each corner of all objects. b) since people tend to keep a certain distance to objects, the nodes should be placed at a distance of 25cm to the corner of obstacles. c) is is, however, easier to place 4 or more nodes close to each corner and remove the ones inside an object in a later step.



themselfs. For pedestrian simulations, where the density is not too high (e.g. outdoors), time dependancy in route algorithms can be used to model external influences (e.g. weather, closed chairlifts). Note that the cost function used here is not only dependant of the travel time, but is influenced by other factors as well (e.g. steepness of the path, how nice the view is, diversity of environment, see Gloor *et al.*, 2003a).

4. Precomputed Forces

The two presented models have in common that they both use precomputed force fields, stored in a cell-based grid. Different is the way they calculate the desired velocity v_i^0 of a pedestrian. Table 1 gives an overview of the forces used.

Table 1: A comparison of the way the forces that influence the movement of an agent are computed in the two models.

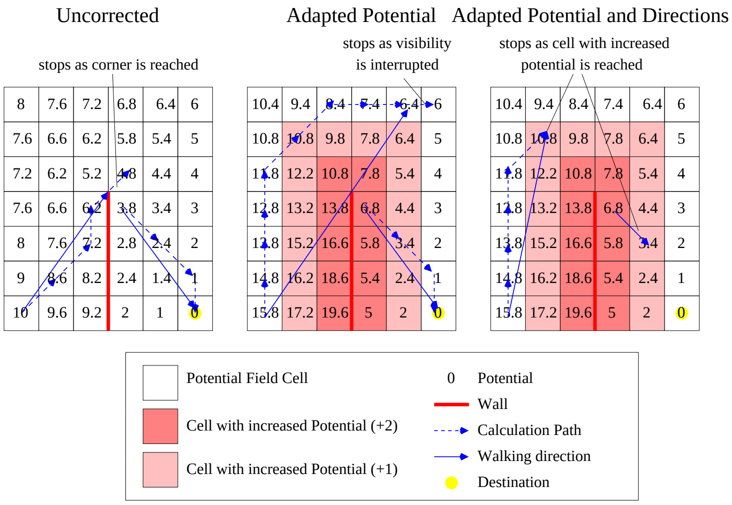
Force	Description	Potential Model	Graph Model
Desired velocity	The force which pulls the agents along the path	Pre-computed	During runtime
Social forces	Influence of the other agents	During runtime	During runtime
Path force	The force which pulls the agents back on the path	N/A	During runtime
Walkability	How fast an agent can walk in a given position	Pre-computed	Pre-computed

Note that there is no **path force** in the potential model. This force is used in the graph model to keep the pedestrians near the middle of an walking path. In the potential model, however, the pedestrians figure out where to walk by themselves, provided that walking on a path is faster.

To encode this information, a further grid is needed. It stores how fast a pedestrian is able to walk in a given location. The value of the **walkability parameter** $w(\mathbf{r}_i)$ is between 0 (obstacle, no walking possible) and 1 (flat street). This actually reduces the desired velocity from v_i^0 to $w(\mathbf{r}_i)v_i^0$. In the Alpsim project, walkability can also be used to model swamps, snow fields, or dense forests, where walking is in principle possible, but cumbersome.

This parameter is used by both models. The graph model uses it to slow down the pedestrians that are not exactly on the path. A reason to leave the path could be because of other pedestrians

Figure 5: The walkability parameter can be used to prevent agents from walking too close to an obstacle. Walking directions without any corrections (left), with adapted potential field (middle) and with adapted walking direction algorithm (right)



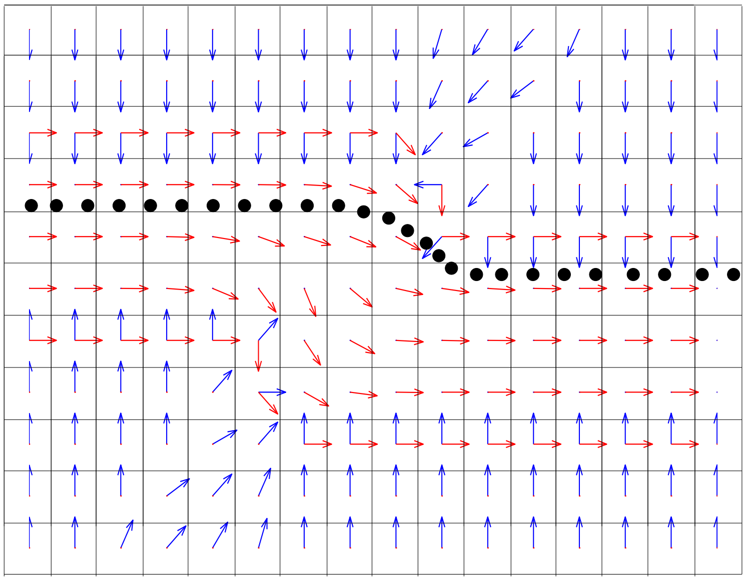
which apply the force defined in equation (2). It is also possible to assign a lower walkability to paths of lesser quality, e.g. narrow hiking paths. This does not affect the route choice of the pedestrians directly. However, the modules of the strategical layer (Gloor *et al.*, 2003a) notice this delay eventually and will take this into account for the next iteration.

In the potential model, the walkability is used directly by the algorithm that calculates the optimal walking direction for an agent (Figure 5). This yields in trajectories that circumvent regions with lower walkability, if the detour is small enough. As a side effect, the walkability can also be abused to prevent the agents from walking too close to a wall or an obstacle. This is done by setting the walkability parameter to a lower values where an agent should not walk.

These forces are relatively expensive to calculate, since one needs to enumerate through all possible objects that could influence a given location. Yet, since those forces do not depend on time, they can be pre-computed before the simulation starts. In order for this to be successful, some coarse-graining of space is necessary. For this, we use cells of size $25cm \times 25cm$, and assume that all time-independent forces are constant inside a cell. The resulting force field (Fig. 6) becomes non-continuous in space, but this is not a problem in practice since this only influences the acceleration of pedestrians. That is, the acceleration contribution from the environmental forces can jump from one time step to the next, but since time is not continuous, this is not noticeable.

Pre-computing the values for all cells in a hiking region of, say, $50km \times 50km$, does not fit into regular computer memory. To avoid this problem, we implemented two methods: lazy initialization, and disk caching. By **lazy initialization**, we mean that the values are computed only when an agent really needs them, also knows as *Virtual Proxy Pattern* (Gamma *et al.*, 2001, pages 207–217).

Figure 6: The hybrid simulation technique. The forces (arrows) are valid for the whole cell; a pedestrian's trajectory (dots) can follow arbitrary positions.



In practice, the simulation area is divided into blocks of size $200m \times 200m$. Every time an agent enters one of these blocks, the values for *all* cells inside that block are computed. Since hiking paths cross only a small fraction of those blocks, the cell values for many blocks in our hiking area will never be calculated.

In addition, the cell values, once computed, are stored on disk (**disk caching**). Every time when an agent encounters a block for which the cell values are not in memory, the simulation first checks if they are maybe on disk. Computation of the cell values is only started when those values are not found on disk. In consequence, a simulation started for the first time will run more slowly, because the disk cache is not yet filled.

If the simulation runs out of memory, then blocks which are no longer needed, i.e. which have not been crossed by an agent for a long time, are unloaded from memory. If they are needed again, they are just re-loaded from disk. This corresponds to the *Least Recently Used (LRU) Page Replacement Algorithm* described by Tanenbaum (2001, pages 218–222).

An additional advantage of the blocks, well known from molecular dynamics simulations, is that one can use them to cut off the short-range interaction between the pedestrians. Agents which are not in the same or one of the eight adjacent blocks are ignored. This implies that there needs to be some data structure where agents are registered to the block. Agents that move from one block to another need to unregister in the first block and register in the next one. In this way, an agent searching for its neighbors only needs to go through the registered agents in the relevant blocks. This brings the computation complexity from O(N^2) down to O(NM), where N is the number of all agents in the simulation, and M is the number of agents in a single block. M is a reasonably small number when compared to the number N of all agents in a real-world scenario.

5. Simulation of Zürich Main Station

As a real-world scenario for testing our models, we chose a simulation of an evacuation of Zürich Main Station. We simulated an area of 700m $\times$ 200m with more than 3000 obstacles. Agents were placed randomly within the simulated area (neighter inside buildings, nor on the railways). We considered eight different exit locations, every pedestrian chooses the one closest to him. Note that we did just one iteration, which means that congestion at a certain exit can occur and is not avoided by the pedestrians. It would be possible, however, to run multiple iterations of the scenario to enable the agents lo learn from such a situation (see e.g. Gloor *et al.*, 2003a; Raney and Nagel, 2004).

To handle the different levels of the Main Station, we had to introduce the concept of stairs and elevators. Different levels were placed beside each other. Elevators were implemented like teleportation (with a time delay of some seconds), stairs are divided into two halves, each simulated in a level, with teleportation in the middle. However, forces between pedestrian on different sides of this boundary still contribute to pedestrian movements.

Since Zürich Mainstation has more than one exit, we had reflect this in our models. For the potential field model, this is simple: starting the flooding algorithm from all exits simultaneously is sufficient. All the different exits are stored in the same precomputed map, since for every given point in the simulated area, there is exactly one closest exit. However, for the graph model, this was more complicated: we had to generate multiple graphs, one for each possible exit.

For a pedestrian simulation, two measurements are inportant: i) how realistic the results are, and ii) how fast the computation is.

The size of the cells that store the potential field does affect the time pre-computation takes. For the Zürich Main Station scenario, we measured on a 700MHz Pentium III equipped with 256 Mbytes of RAM:

Table 2: Influence of the cell size to the time required for pre-computing the potential field

Cell Size (m)	Cell Checks	Calculation Time (s)
0.25	1’325’459	1554
0.5	325’331	386
1	78’324	102

Before the potential of a new cell can be calculated, a visibility check has to be performed to ensure that the cell is accessible (visible) from the neighbour cell. As the check has to iterate

Figure 7: An evacuation of Zürich Main Station using the potential field model. The pedestrians hurry to the closest exit avaliable. This is a capture of the first of multiple iterations, which means that congestion at a certain exit can occur and is not avoided by the pedestrians

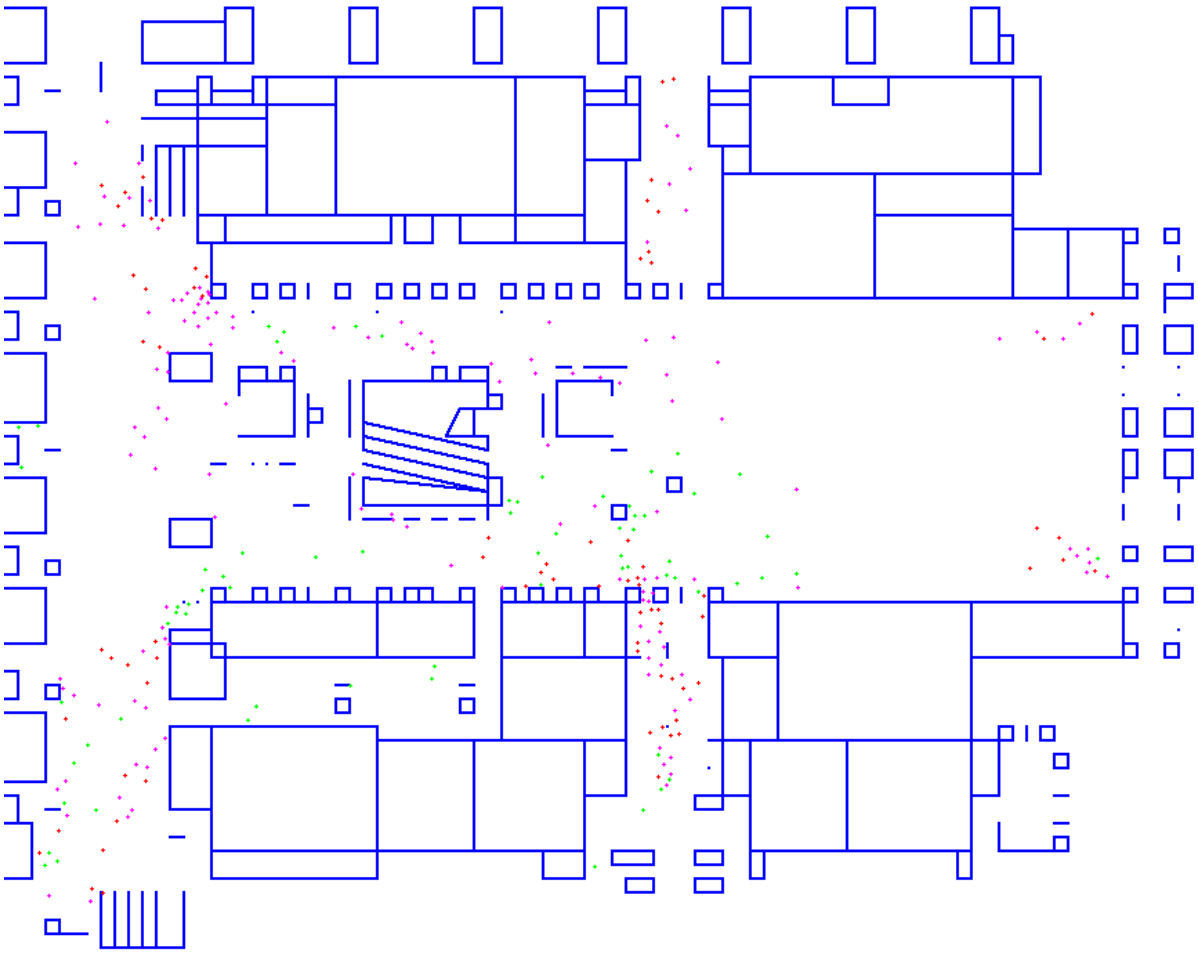


Figure 8: Potential field for Zürich Main Station, generated for the 8 potential exits. This potential field has to be generated each time the destination changes.

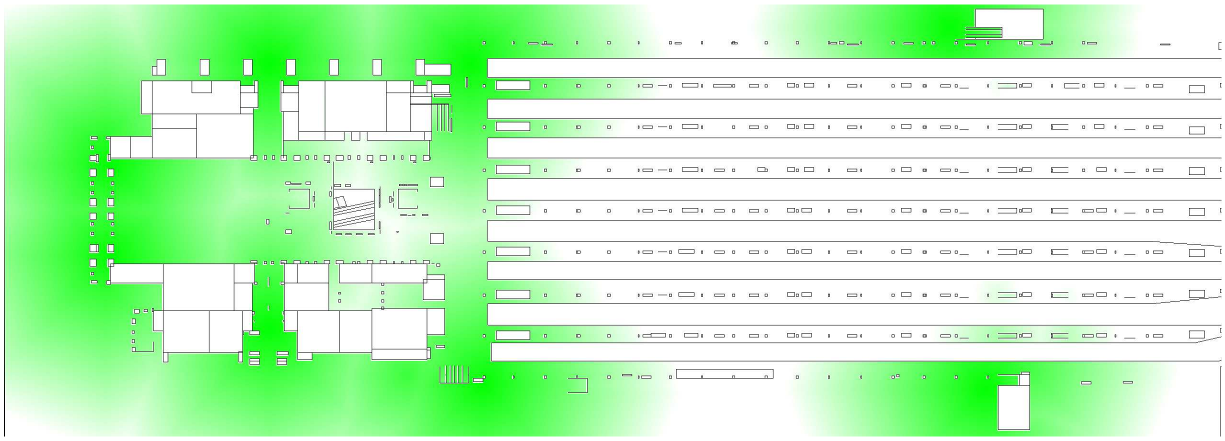
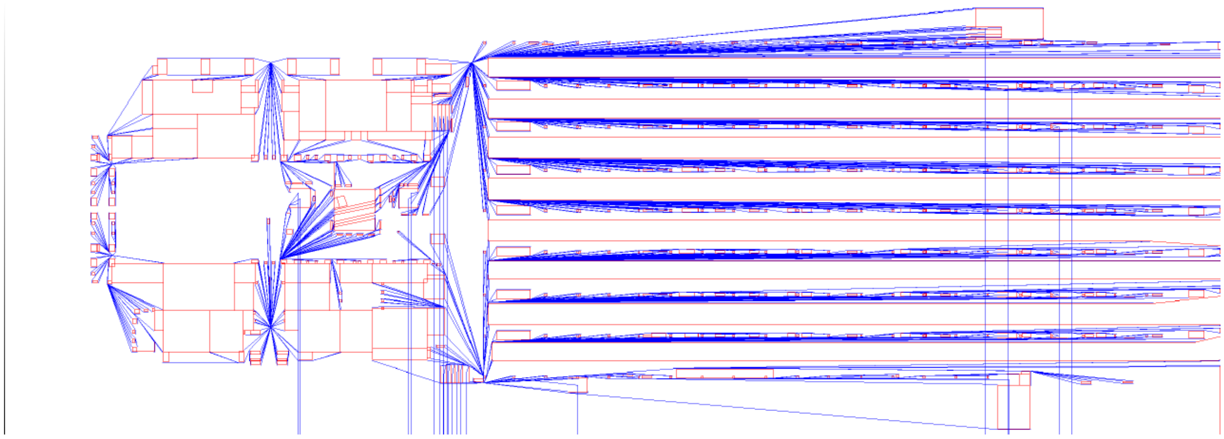


Figure 9: Spanning tree for Zürich Main Station, generated for the 8 potential exits. This spanning tree has to be generated each time the destination changes, but the underlying nodes do not change.



over all obstacles in the simulated area, the calculation time increases linearly with the number of obstacles.

Table 3: Influence of the number of obstacles in the simulated area

#Objects	Calculation Time (s)
100	65
500	219
1000	420
1500	643
2000	866

The graph models requires that the graph is pre-computed. This takes 175 minutes for the full scenario, containing all of th 300 obstacles. If small obstacles like pillars or benches are removed for the graph generation, the time can be reduced to 35 minutes. Small objects hardly influence the path of a pedestrian chooses. However, the pedestrians still do not walk through these obstacles, since the 3^{rd} term of the RHS of equation (1) still pushes them away from obstacles.

An comparison of the presented models is shown in Table 4. The Zürich Main Station scenario was run for 10, 100 and 500 agents using each model. Note that in this numbers the time used to pre-calculate the potential field and to generate the graph is *not* included.

For the graph model (column d) and e)), the time to simulate the first steps takes longer than the average because the agents have to find an optimal node on the graph to walk to their destination.

The potential field model needs about one minute to load the pre-calculated cells into memory.

References

ALPSIM www page (accessed 2004) www.sim.inf.ethz.ch/projects/alpsim/. Planning with Virtual Alpine Landscapes and Autonomous Agents.

Dijkstra, E. (1959) A note on two problems in connexion with graphs, *Numerische Mathematik 1*, (1) 269 – 271.

Galea, E. R. (Ed.) (2003) *Pedestrian and Evacation Dynamics 2003*, Proceedings of the 2nd international conference, CMS Press, University of Greenwich.

Gamma, E., R. Helm, R. Johnson and J. Vlissides (2001) *Design Pattern: Elements of Reusable Object-Oriented Software*, Addison-Wesley professional computing series, Addison-Wesley.

Gloor, C., D. Cavens, E. Lange, K. Nagel and W. Schmid (2003a) A pedestrian simulation for very large scale applications, in A. Koch and P. Mandl (Eds.), *Multi-Agenten-Systeme in der Geographie*, no. 23 in Klagenfurter Geographische Schriften, 167–188.

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

Helbing, D., I. Farkas and T. Vicsek (2000) Simulating dynamical features of escape panic, *Nature*, (407) 487–490.

Hoogendoorn, S., P. Bovy and W. Daamen (2002) Microscopic pedestrian wayfinding and dynamic modelling, in M. Schreckenberg and S. Sharma (Eds.), *Pedestrian and Evacuation Dynamics*, 123–154.

Nishinari, K., A. Kirchner, A. Nazami and A. Schadschneider (2001) Extendes floor field CA model for evacuation dynamics, in *Special Issue on Cellular Automata of IEICE Transactions on Information and Systems*, vol. E84-D, January 2001.

Raney, B. and K. Nagel (2004) An improved framework for large-scale multi-agent simulations of travel behavior, in *Proceedings of Swiss Transport Research Conference (STRC)*, Monte Verita, CH. See www.strc.ch.

Schreckenberg, M. and S. D. Sharma (Eds.) (2001) *Pedestrian and Evacation Dynamics*, Springer.

Stucki, P. (2003) Obstacles in pedestrian simulations, Diploma thesis, Swiss Federal Institute of Technology ETH.

Tanenbaum, A. S. (2001) *Modern Operating Systems*, Prentice-Hall, Inc., second, international edn.

Weidmann, U. (1993) *Transporttechnik der Fussgänger*, vol. 90 of *Schriftenreihe des IVT*, Institute for Transport Planning and Systems ETH Zürich, 2 edn. In German.

Table 4: Time to simulate an evacuation of Zürich Main Station using a) potential field model with minimal neighbor approach, b) potential field with minimal distances approach, c) potential field with precalculated walking directions, d) graph model ignoring small obstacles and e) graph model. Further the average/maximum time and distance needed to leave the station are shown. This shows that the compared models generate similar results.

	a)	b)	c)	d)	e)
10 walking agents:					
Total Time	81s	76s	81s	38s	84s
Time First Round	<1s	<1s	<1s	16s	66s
Time per Round	<1s	<1s	<1s	<1s	<1s
Walking Time	72s	116s	57s	88.6s	80s
Max. Walking Time	132s	203s	154s	199s	199s
Walked Distance	75m	98m	91m	99m	96m
Maximal Walking Distance	132m	213m	240m	237m	227m
100 walking agents:					
Total Time	139s	75s	252s	290s	525s
Time First Round	<1s	<1s	1s	131s	341s
Time per Round	<1s	<1s	1s	<1s	1s
Walking Time	67.1s	50.9s	55s	70.6s	74s
Max. Walking Time	239s	180s	230s	182s	182s
Walked Distance	73m	51m	66m	80m	77m
Maximal Walking Distance	239m	217m	214m	227m	229m
500 walking agents:					
Total Time	282s	240s	446s	1726s	4617s
Time First Round	2s	1s	4s	664s	1459s
Time per Round	2s	1s	4s	12s	12s
Walking Time	60.3s	67s	53s	69.5s	76.6s
Max. Walking Time	234s	189s	195s	262s	271s
Walked Distance	64m	79.4m	92m	79m	78m
Maximal Walking Distance	236m	245m	189m	247m	233m