Environmental abstraction and path planning techniques for realistic crowd simulation

Sébastien Paris, PhD student

Stéphane Donikian, CNRS Researcher

IRISA - Campus de Beaulieu - 35042 Rennes cedex, France

Tel. (+33) 02 99 84 74 18 Fax. (+33) 02 99 84 71 71

email: sgparis@irisa.fr ; donikian@irisa.fr

Nicolas Bonvalet, AREP Research Manager

AREP - 163 bis avenue de Clichy - 75847 Paris Cedex 17, France

Tel. (+33) 01 56 33 44 14 Fax. (+33) 01 56 33 05 67

email: nicolas.bonvalet@arep.fr

Abstract

This paper treats two linked subjects underlying behavioural simulation. First, the way to describe a virtual environment through an informed hierarchical abstract graph.

This graph stores some pre-computations such as potential visibility sets, oriented grids, or densities of people, which can be used individually by simulated entities. Second, the way to use this abstract graph to perform a realistic and efficient path planning, which takes care of individual preferences as well as individual knowledge of the environment. Moreover, the path planning method we propose is reactive to some events, reflecting the perceived modifications of the environment, which allows the entity to adapt its behaviour in consequence.

Keywords: Environment Description, Path Planning, Simulation Involving Virtual Humans

Introduction

Simulating crowds of people is a complex topic, implying to manage many entities at the same time, possibly thousands, while making them sufficiently realistic to be analysed. For these entities to be realistic, they must handle some behavioural procedures which will guide them during the simulation. One of these necessary behaviours is unquestionably the capacity to move, and so the capacity to plan a path. Our study is undertaken within the framework of microscopic crowd simulation inside constrained environments, and particularly exchange areas like train stations. We want to provide maximal entity autonomy and realism to achieve two main goals. First, an entity must be able to adapt its behaviour to take into account environments with low population as well as overcrowded ones, making it possible to produce the macroscopic emergent behaviour called *crowd*. Second, an entity must

achieve a sufficiently realistic behaviour in order to make the interpretation of data from the simulation possible. Moreover, as previously stated, crowd simulation implies to simultaneously manage many entities, making computation cost a major constraint in our model. The point we are focusing on in this study is the way an agent can plan a realistic path inside a more or less known environment. We start, in the related work section, by presenting the spatial subdivision, environmental abstraction, and path planning techniques, including our previous work which is used as a basis for this paper. We also talk about crowd studies in that section. Then, we present the improvements done to our previous environment abstraction, making it more adapted to realistic path planning. We continue with our path planning method, which is hierarchical, handles many criteria (including subjective ones), and can react to some adaptation events. After that, we present some applications of our model with our industrial partner, AREP, and technical benchmarks. Finally, we conclude by giving the main advantages of this work, and address future work.

Related work

Spatial subdivision converts complex geometric data, made up of a great number of polygons, in a more or less informed database. The three principal approaches, which have been reused in behavioural animation, come from the field of robotics [1]. *Potential fields* [2] associate repulsive powers with the environment's obstacles, and an attractive one with the agent's destination. *Roadmaps* [3] discretise the navigation space in a network of paths

made up of lines and curves. *Cell decomposition* computes a connexity graph in two different ways: an approximate decomposition [4], using grids or quadtrees, covers a subspace included in the environment's free space; an exact decomposition [5], generally using convex cells, covers the whole of the environment's free space. Our previous model's starting data is obtained with such an exact decomposition, provided in part by F. Lamarche *et al.* [6] algorithm. This algorithm is applied to a 2D graphical representation of the environment, and extracts a set of convex cells by computing a constrained Delaunay triangulation while keeping the bottlenecks of the environment. **Topological abstraction** is a complementary process, used to better organise the information obtained at the time of spatial subdivision, which generally consists in producing a hierarchical graph representation of the environment. The unification process is addressed principally in two ways: a *pure topological* unification [6] associates the subdivision cells according to their number of connexions; a *more conceptual* unification introduces a semantical definition of the environment, like with the IHT-graph structure [7], producing an informed environment [8]. **Path planning** is the agent behaviour which produces a path in order to reach a destination. This process is generally performed by a graph crossing algorithm, which can exploit the hierarchical aspect given by a topological abstraction [9]. Even if travelled distance impacts path planning, some experimental research have shown that a certain number of other factors must be taken into account [10]. The *least-angle strategy* [11] minimises the angle between the agent's direction and the destination, allows to take into account the agent's knowledge about the environment, and introduces the notion of preference in the decision process. The

simplest path strategy [12] introduces a cognitive cost minimisation for path evaluation. The congestion of an area of the environment [13] is an important factor, especially for crowd simulation. A *stress factor* [14] can also be taken into account, based on the number of surrounding people, or the difference between the current and the shortest path. Our previous model did not directly address path planning, but was defining the way to manage an individual knowledge of the environment, and the processes to update (observation) and access (remember) this knowledge. Even if J.J. Fruin [15] has been a pioneer in the seventies, the development of **crowd simulation** is still recent and limited. *Macroscopic* simulation [16] has been historically the first approach, globally managing all the entities of the simulation. Based on a large number of observations, *statistical* models [17] have been proposed to express the evacuation delay of a building through equations. *Particle* systems [18] are based on physical laws that describe attractive and repulsive forces that can be associated to obstacles and moving entities. Some studies have also focused on crowd simulation and its levels of autonomy [19] in order to provide realistic crowd behaviour inside virtual environments.

Topological abstraction

Introduction

The first necessary step to perform path planning is to describe the simulated environment. Our previous model [20] was based on a pure topological unification. A hierarchical

graph, composed by three interconnected layers (as shown in figure 1), is computed: the first layer corresponds to the output of the spatial subdivision process, while layers two and three are successive groupings. The type of the nodes of the graph is set according to their number of connections c : dead end ($c = 1$), corridor ($c = 2$), and crossroad ($c \geq 3$). Moreover, two flags can be set for some specific nodes: the nodes connected to the subdivision bounding limits are *virtual*; the nodes connected to at least one virtual and one non virtual node are *entry/exit*. Through the rest of this paper, we will call the nodes of the first layer *cells*, of the second layer *groups*, and of the third layer *zones*. In order to improve the expressivity of the informed hierarchical graph, we proposed to pre-compute some static data. The *surface area* of each node is stored, enabling to compute the node's density. A *potential visibility set* (PVS) is computed for each inter-group connection. We also associate a *grid representation* to each group, which is used to compute the density values corresponding to the inner paths of the group. Finally, a set of data is associated to each oriented pair of external connections of a group or a zone, which gives $c \times (c - 1)$ sets per node. Each set of data contains the information corresponding to the crossing of the node through the corresponding connections. This information is composed of : the *shortest path length*; the *minimal path width*; the *path direction*; a *flows of people* counter, refreshed by the entities currently moving through these connections; and a *density of people* marker, which is filled by a dynamic global process using grids. We will present in the following how the topological abstraction of our previous model can be improved in order to make it more suited to realistic path planning.

Improved groupings

In order to improve the overall aspect of the abstraction, we have introduced a geometrical heuristic in the algorithm. This heuristic is concurrent with the topological heuristic, with the need to validate the topological property for both layers two and three of the abstraction graph. The geometrical heuristic has a greater priority than the topological heuristic for the first abstraction grouping, and a lower one for the second abstraction. The goal of the geometrical heuristic is to obtain groups with the best convex aspect. To do so, oriented bounding boxes are computed for every node of the working layer. Then the coverage ratio of the bounding box is evaluated as the surface of the box divided by the actual surface of the node. The geometrical heuristic finally performs groupings of a set of connected nodes if and only if the group ratio is better, i.e. closer to one, than every base node ratio. The direct application of this geometrical grouping method is to keep subdivision bottlenecks in groups (as shown in figure 2.c), as well as to highlight large areas, thus allowing efficient pre-computations and graph traversal. The topological heuristic has also been modified concerning the computation of the last abstraction layer. Indeed, in the previous heuristic, dead end groups were absorbed by their adjacent crossroad group to form a resulting zone, which was a terminating condition for groupings. In the modified algorithm, if the resulting zone is also a dead end connected to a crossroad zone, they are merged, and so on until the adjacent crossroad zone is connected to at least two non dead end zones. This last improvement highlights zones which are much more conceptual (as shown in figure 2.e),

and prepares the environment for a fast discriminant path planning heuristic.

Improved information

To improve the information stored in the environment abstraction, we have introduced new types of nodes in addition to previous *dead end*, *corridor*, *crossroad*, and *virtual* nodes. These nodes are called **conceptual** because they do not correspond to any physical area, i.e. they have a null surface. All of these nodes are ungroupable: a conceptual cell, group or zone is represented by the same node, present in the three layers of the graph. Moreover, these conceptual nodes are named to be easily targeted by a path planning procedure, and thus can be used in a rational behaviour process. **Stoppage** nodes have no outgoing connection, and an unlimited number of incoming connections. They are used to stop any graph crossing algorithm, while preserving the standard nodes connection abilities for future graph modifications. So, if a connection must be temporarily broken in the graph, it is redirected to a stoppage node, thus respecting the integrity of the abstract graph, i.e. relieving the constraint to recompute the abstraction. **Oriented** nodes give a direction for graph exploration algorithms. They are specific *corridor* nodes, whose orientation can be changed dynamically thanks to their automatic management of stoppage connections (as shown in figures 3.a-d). **Access** nodes are a refinement of oriented nodes. They are used as connections between a virtual cell and a non virtual one, thus representing possible entries and exits of the environment. **Link** nodes are also a refinement of oriented nodes. They are

used as gates between two hierarchical graphs, allowing to manage more than one abstracted graph. For example, to represent a train station composed of two floors, a hierarchical graph is created for each floor. Then, a link node is introduced everywhere a connection between the floors is necessary, like for stairs (an illustration is shown on figure 3.e). But, as opposed to access nodes, a link node does not directly connect two standard nodes. Instead, it is connected to a standard node of its corresponding graph, and to another specific conceptual node called connected links. **Connected links** nodes are the only nodes which are not associated to a specific hierarchical graph. They are used to associate a set of link nodes between each other. To take again the example of the two story train station, a connected link will represent a stairwell or an elevator, and thus be connected to all the link nodes associated to this stairwell or elevator. By introducing these new types of nodes, we have made the hierarchical graph oriented, re-orientable, and able to be associated to any number of other graphs. Moreover, the introduction of connected links nodes between graphs enables us to describe environments containing as many abstracted graphs as needed. The direct application is to simulate buildings containing more than one floor, but another application can be to divide a huge environment into interconnected conceptual hierarchical graphs ; for example, a city can be represented by a graph, and some buildings of the city by other graphs connected to the first one. The most interesting property of this approach is that finally, all interconnected hierarchical graphs can be used as one global hierarchical graph, and thus, be used by graph exploration algorithms (including path planning) without any specific management.

Path planning

The path planning method we propose has two major constraints. First, it must attain minimal calculation costs, in order to allow the simulation of a great number of agents, to achieve crowd simulation. Second, it should maintain a high level of realism, in order to allow data extraction for further analysis by specialists. So, we propose to base our path planning method on our informed environment description, using a hierarchical approach with a multi-criteria heuristic. Moreover, we propose to take care of the changes that might occur inside the environment since the last planned path by triggering events, which are handled by the reactive part of our algorithm.

Hierarchical path planning

Since we provide a topological description of the environment as three hierarchical graphs, we propose to take advantage of it by using hierarchical path planning. The algorithm is divided into three main steps (*figure 4.a*), one for each level of the graph:

1. Plan in the **highest abstraction graph**, from current to destination zones. If no path is found, then the ending condition cannot be satisfied. Otherwise, proceed to step 2.

2. Plan in the **first abstraction sub graph** included in the current zone, and the next one if any. If the zone path only contains one node, plan from current to destination groups both located in the current zone. Otherwise, plan from the current group to the first encountered group in the next zone. Proceed to step 3.

3. Plan in the **informed subdivision sub graph** included in the current group, and the next one if any. If the group path only contains one node, plan from current to destination cells both located in the current group (*the entire path is computed*). Otherwise, plan from the current cell to the first encountered cell in the next group (*this sub part of the path is computed and can be used for navigation*).

The best property of such a hierarchical method is the smoothing of calculation costs over time. Indeed, the whole path is only computed for the most abstracted graph, which contains a small amount of nodes compared to the informed subdivision. Then, both sub paths are computed only when needed, as the entity moves. Consequently, lost computation time when a path is invalidated and must be recomputed is reduced compared to a complete path evaluation on a graph which is neither abstracted nor hierarchical. As for any graph crossing algorithm, this hierarchical path planning minimises the cost to find the best path. The cost evaluation method, which is based on multiple criteria, will be detailed in the next section. In addition, the algorithm only takes into account the connections of the graph which are known by the entity, simply ignoring the others. Finally, the algorithm can be specialised in three ways according to the type of ending condition used, allowing to handle specific cost evaluations. The **reaching** specialisation has a unique known node of the graph as a goal. In this case, the subjacent algorithm for graph exploration is the well known A^*. The **choosing** specialisation has a set of known nodes of the graph as a goal. This procedure can be used by a higher behaviour to choose the best location to go to, for outgoing accesses for

example. In this case, the subjacent algorithm for graph exploration is a *flood fill*. Indeed, an A^* is difficult to use because a predictive cost estimation of the ending conditions from a given place cannot be easily done. Even if the *flood fill* method has a greater complexity than the A^* algorithm, it remains acceptable since the most abstracted graph only contains a small number of nodes. The **exploration** specialisation describes a more conceptual path planning. Here, the goal is not to reach a specific location, but to improve the agent's topological knowledge. So, the ending condition of that procedure is when a partially known node has been reached. For the same reasons as for the choosing procedure, the subjacent algorithm for graph exploration is a *flood fill*.

Multiple criteria path planning

We propose to take into account many criteria for our cost evaluation, in order to make the decision process as realistic as possible, and more specific to each simulated entity. The cost we evaluate is divided into two parts. First, a **time to travel** cost represents the amount of time needed by the entity to travel through the path. Second, a **preference** cost increases or decreases the first cost by filtering it using individual affinities. One can notice that if an infinite cost is produced, then the corresponding part of the path is considered uncrossable by the entity.

The ***time to travel*** cost is the first to be evaluated, and corresponds to the time needed by an entity to travel through a node. So, the initial cost is computed as $cost = \frac{distance}{desired\ speed}$

where *desired speed* is the travel speed chosen by the entity. The *distance* is retrieved as shown on figure *4.b*. The density over the path can then be taken into account for the zones which have been seen by the entity a little while ago. It allows to evaluate the maximum speed (S_{max}) available for an entity, according to J.J. Fruin [15] study on levels of services:

$$S_{max} = \begin{cases} \textit{desired speed} & \text{if } \textit{density} < 0.3 \\ (3.79 - \textit{density}) * 0.37 & \text{if } 0.3 \leq \textit{density} \leq 3 \\ 0 & \text{if } \textit{density} > 3 \end{cases}$$

Then, if this maximal speed is smaller than the entity's desired speed, but not null, the cost is modified: $cost = cost \times \frac{desired\ speed}{S_{max}}$. For the case where the maximum speed is null, the cost is infinite. One can notice that the cost could be under evaluated for the zones of the environment that have not been seen by the entity for a long time. The solution to this problem will be addressed in the section describing the reactive part of the algorithm.

The ***preference*** cost is obtained by filtering the time to travel cost according to individual affinities. These affinities are represented by preference levels P_F : the higher a preference level is, the more a filtered cost is impacted; a null preference simply cancels the filter. In fact, this preference level corresponds to the time that an entity is ready to spend in order to avoid a zone which really does not satisfy the corresponding criterion (a negative preference signifies that the filter is attractive). One can notice that the preference level can be dynamically changed to reflect the entity's current state of mind (in a hurry, lazy, etc.), or its current goal (reaching, choosing, exploration). The filter function is applied thus : $cost_F = cost + P_F \times V_F$ where V_F is the computed value of that filter, with $V_F \in$

$\{[0;1], \textit{infinity}\}$ in order to make the filters comparable between each other.

The first filters are based on static data, and thus can be computed without any preliminary test. The **passage width** filter reflects the need to travel through large areas :

$$V_{wid} = \begin{cases} \textit{infinity} & \text{if } \textit{passage width} < \textit{entity's width} \\ 0 & \text{if } \textit{passage width} > \textit{entity width} \times 11 \\ 1.1 - \frac{\textit{passage width}}{\textit{entity width} \times 10} & \text{otherwise} \end{cases}$$

where the *passage width* is a precomputed value obtained as shown in figure *4.c*. The **current direction** filter reflects the need for the entity to minimise its direction changes through the path : $V_{cdir} = \frac{1}{2} \times (1 - \overrightarrow{\text{dir}_{crt}} \cdot \overrightarrow{\text{dir}_{prev}})$ where both current and previous directions are normalised vectors obtained as shown in figure *4.d*. The **destination direction** filter reflects the need for the entity to find the most direct path to its goal : $V_{ddir} = \frac{1}{2} \times (1 - \overrightarrow{\text{dir}_{crt}} \cdot \overrightarrow{\text{dir}_{dest}})$ where both current and destination directions are normalised vectors obtained as shown in figure *4.e*. The **discovering potential** filter reflects the need of the entity to increase its topological knowledge : $V_{dp} = \frac{\textit{known node connections}}{\textit{supposed node connections}}$ where *known node connections* is extracted from the entity's individual knowledge, and *supposed node connections* is taken, for now, as the real number of connections of the node in the abstract graph.

The other filters are based on dynamic data, and thus can only be evaluated for the zones seen by the entity a little while ago. Moreover, as the dynamic data are only computed for groups and zones, these filters are not evaluated for cells. The **flows of people** filter is composed of two sub-filters: one for the flows of people going in the same direction than the entity (F_{sf}) and one for all other directions (F_{of}). Both of these sub-filters have their

own preference and value: $V_{sf} = \frac{\textit{same direction flows}}{\textit{population of the node}}$ and $V_{of} = \frac{\textit{other directions flows}}{\textit{population of the node}}$. One can notice that $V_{sf} \neq 1 - V_{of}$ because it is possible that some people of a node are not moving. The preference of the *same direction flow* sub-filter is negative in order to express the desire of the entity to travel through nodes with same direction flows. The **density of people** filter reflects the desire of the entity to avoid overcrowded areas : $V_{den} = \frac{\textit{density}}{3}$.

Reactive path planning

As seen before, the cost evaluations that depend on dynamic data can only be done if the entity has seen the corresponding node a little while ago. So, a path could be computed at a time, then become invalid since the entity can observe new areas. That is why our model needs an adaptation method which can correct the current path when new data are available. A possibility is to check the validity of the path at a regular frequency, and to recompute the invalid part of the path. Such a process is unnecessarily costly as it would certainly find a valid path most of the time. Moreover, the validity evaluation is difficult to calculate, having to be sensitive enough to allow decision changes, but permissive enough not to lead to behaviour oscillations.

So, we propose in our model to handle specific events which request the recomputation of a part of the path, depending on the modified data. The reactive aspect is well shown by the recomputation decision heuristic: 1) If the event may decrease the weight of a node not belonging to the path, recompute the whole path. 2) Else, if the event may increase the

weight of a zone of the path, recompute the whole path. 3) Else, if the event may increase the weight of a group of the current zone, recompute the group then the cell part of the path.

The path recomputation we are talking about is a little specific, because it takes care of the previously computed path. Indeed, the new path is computed with a part of the nodes of the previous path as additional ending conditions. The part of the previous path which is taken starts from the further invalidated node and finishes with the destination node. Then, when the algorithm validates an ending condition, and if that condition corresponds to a node of the previous path, the corresponding reminder of the previous path is appended to the found path. In addition, the events are not taken into account immediately, but are stored and processed collectively at a low frequency (once a second is largely sufficient). The events we manage are classified in two categories. The **observation** events are triggered by the navigation process of the entity. Such events occur when new nodes are discovered by the entity, and recorded in its topological knowledge, allowing it to find a better path. These events also occur when an already known node is seen again after a while, allowing to take into account the dynamic data that may have changed. The **rational** events are more conceptual, and can occur when the destination is modified (here the events are automatically triggered), or when the current path is too old (here the events are triggered by a low frequency individual control process). This last case occurs when a dynamic value on which the path depends becomes out-of-date, i.e. when the time passed since the last evaluation of a dynamic criterion is larger than the time of validity of the information.

Results

Our model has been integrated in a larger application, to be used by architects specialised in flows of people study inside exchange areas. The simulation environment is described in a simple way, through *AutoCAD*. Then, an automated process extracts the environment's information, including walls and any specific orientation data, directly from the *AutoCAD* file. These simulation data can then be saved in *XML* for future use. The last part of the application automatically performs the subdivision of the environment, and finally computes the informed abstraction. This application has been used by our partners, the architects of *AREP*, to study the flows of people inside the St Lazare train station in France. To configure the simulated flows of people, some investigations have been done inside the real train station: people distribution at the entries and exits and time spent in the environment. This experiment is the first in a series, which will allow us to fine tune our model's parameters (essentially the path planning costs), and to validate our results.

The computation performances of our model are obtained using one core of a *Xeon 3.8GHz* processor. For the two story train station, whose definition map is presented in figure *3.e*, the whole abstraction process, including all pre-computed data (such as PVS, shortest path lengths, oriented grids, etc.), is obtained in 5 seconds. The abstraction memory cost is 20 MB, while an entity's knowledge takes 1.5 KB. The environment refreshing, to update the densities of people markers, is performed with an average time of 11 ms, independently of the number of simulated entities. Finally, an entity average planning cost, including the

initial planning as well as the stages of hierarchical plannings and the reactions to the events, is 0.2 ms (also independently of the number of simulated entities). Such performances do not make it possible to perform a simulation with thousands of people in interactive time, but allows all the same to obtain it in an exploitable time. We have already performed large simulations, with two thousand people, obtaining a 3 Hz simulation with all entities animated in 3D, and a 7 Hz simulation with non animated impostors for the entities. Since a simulated entity's main frequency is 20 Hz, the simulation of 2,000 people is performed at a third of real time.

Conclusion and future work

The approach presented in this model has two major advantages. First, it allows the description of virtual environments in an automated way, and renders information concerning this environment available at low costs: PVS, fine densities of people, shortest paths, etc. Second, our model describes a complete path planning method: efficient in computation time thanks to the hierarchical aspect; realistic thanks to the number of managed criteria; individual to each entity thanks to the preference costs and the environment knowledge; and automatically adaptive thanks to the managed events.

Our future work will treat two topics. First, the informed environment abstraction will be updated to integrate specific information about the interactive objects of the simulation. Thus, the path planning method will be able to select the best object to use, for example

while performing a specific behaviour, with only minor changes. Moreover, some specific objects will be managed to give more conceptual data to the entities. These objects could be maps or signs offering a visual interaction to the entities, which could update their environment knowledge or be taken into account in an additional preference cost. Second, we plan on continuing our experiments, by comparing them with real cases, to fine tune our model.

References

[1] J.-C. Latombe. *Robot Motion Planning*. Boston: Kluwer Academic Publishers, Boston, 1991.

[2] Christian Gloor, Pascal Stucki, and Kai Nagel. Hybrid techniques for pedestrian simulations. In *4^th Swiss Transport Research Conference*, Monte Verit, Ascona, 2004.

[3] Mankyu Sung, Lucas Kovar, and Michael Gleicher. Fast and accurate goal-directed motion synthesis for crowds. In K. Anjyo and P. Faloutsos, editors, *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, pages 291–300, 2005.

[4] J. J. Kuffner. Goal-directed navigation for animated characters using real-time path planning and control. *Lecture Notes in Computer Science*, 1537:171–179, 1998.

[5] C. Andújar, P. Vázquez, and M. Fairén. Way-finder: guided tours through complex walkthrough models. *Computer Graphics Forum, Eurographics'04*, 2004.

[6] F. Lamarche and S. Donikian. Crowds of virtual humans : a new approach for real time navigation in complex and structured environments. *Computer Graphics Forum, Eurographics'04*, 2004.

[7] Romain Thomas and Stéphane Donikian. A model of hierarchical cognitive map and human memory designed for reactive and planned navigation. In *4th International Space Syntax Symposium*, Londres, June 2003.

[8] Nathalie Farenc, Ronan Boulic, and Daniel Thalmann. An informed environment dedicated to the simulation of virtual humans in urban context. In P. Brunet and R. Scopigno, editors, *Computer Graphics Forum (Eurographics '99)*, volume 18(3), pages 309–318. The Eurographics Association and Blackwell Publishers, 1999.

[9] Juan-Antonio Fernández-Madrigal and Javier González. Multihierarchical graph search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(1):103–113, 2002.

[10] R. Golledge. Path selection and route preference in human navigation: A progress report. In A. Frank and W. Kuhn, editors, *Spatial Information Theory: A Theoetical Basis for GIS*, volume 988 of *Lecture Notes in Computer Science*, pages 207–222, Berlin: Springer, 1995.

[11] Hartwig H. Hochmair and Victoria Karlsson. Investigation of preference between the least-angle strategy and the initial segment strategy for route selection in unknown environments. *Lecture Notes in Computer Science*, 3343:79 – 97, 2005.

[12] M. Duckham and L. Kulik. "Simplest Paths": Automated route selection for navigation. In W. Kuhn, M. F. Worboys, and S. Timpf, editors, *Spatial Information Theory: Foundations of Geographic Information Science*, volume 2825 of *Lecture Notes in Computer Science*, pages 182–199, Berlin: Springer, 2003.

[13] Wei Shao and Demetri Terzopoulos. Environmental modeling for autonomous virtual pedestrians. *Digital Human Modeling for Design and Engineering Symposium*, 2005.

[14] Toshihiro Osaragi. Modeling of pedestrian behavior and its applications to spatial evaluation. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 836–843, Washington, USA, 2004. IEEE Computer Society.

[15] John J. Fruin. Pedestrian planning and design. New-York : Metropolitan association of urban designers and environmental planners, Inc., 1971.

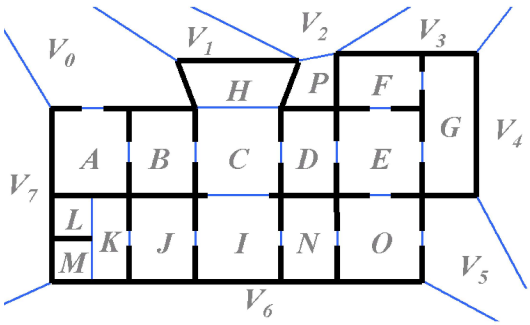
[16] V. M. Predtechenskii and A. I. Milinskii. *Planning for foot traffic flow in buildings*. Amerind Publishing CO Pvt Ltd, New Dehli, Inde, National Bureau of Standards edition, 1978.

[17] Jake Pauls. The movement of people in buildings and design solutions for means of egress. In *Fire technology*, volume 20, chapter 1, pages 27–40. February 1984.

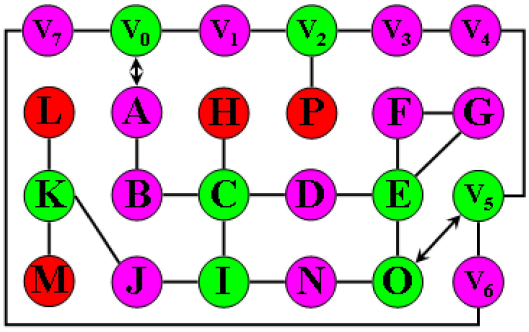
[18] Dirk Helbing, Illes J. Farkas, and Tamas Vicsek. Simulating dynamical features of escape panic. *Nature*, 407:487–490, 2000.

[19] S.Raupp Musse and D. Thalmann. A model of human crowd behavior : Group inter-relationship and collision detection analysis. In *Computer Animation and Simulation '97*, pages 39–51. Springer Verlag, 1997.

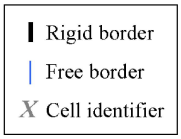
[20] Sébastien Paris, Stéphane Donikian, and Nicolas Bonvalet. Towards more realistic and efficient virtual environment description and usage. In *First International Workshop on Crowd Simulation (V-Crowds'05)*. VRlab, EPFL, November 2005.

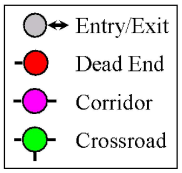


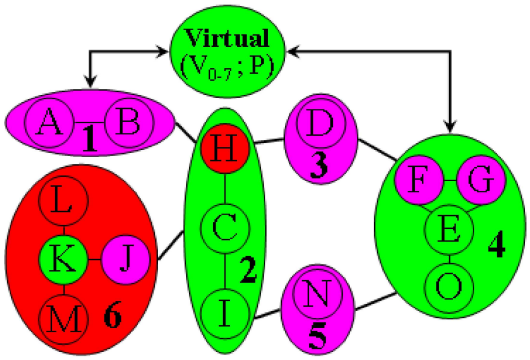
(a) Spatial subdivision



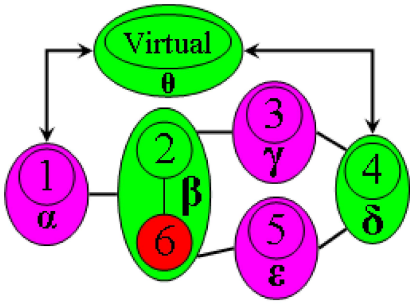
(b) Informed subdivision (first layer)





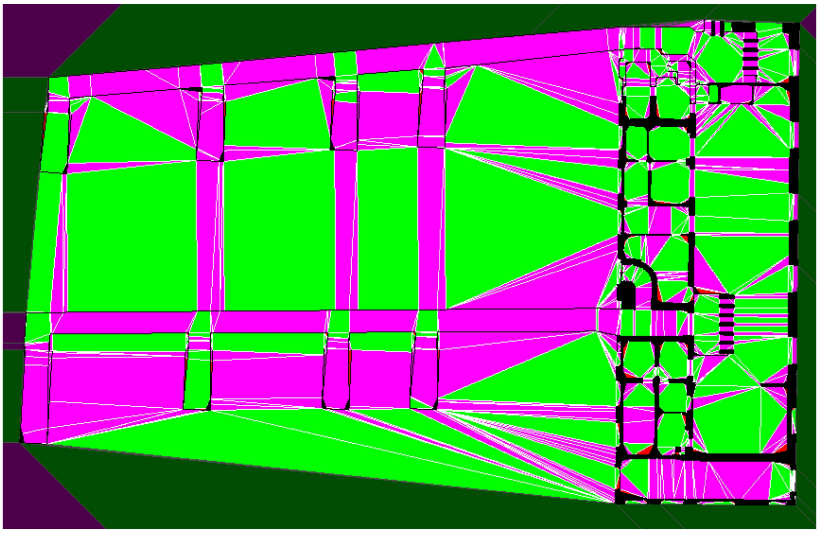


(c) First abstraction (second layer)

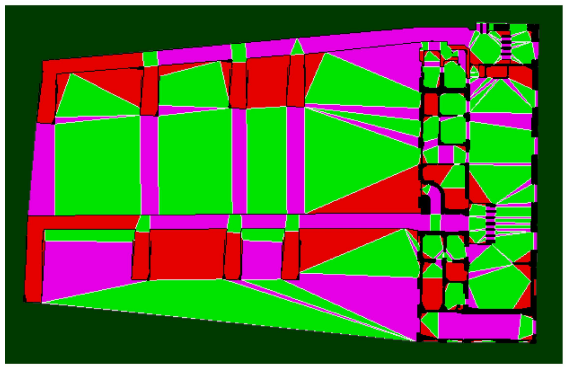


(d) Second abstraction (third layer)

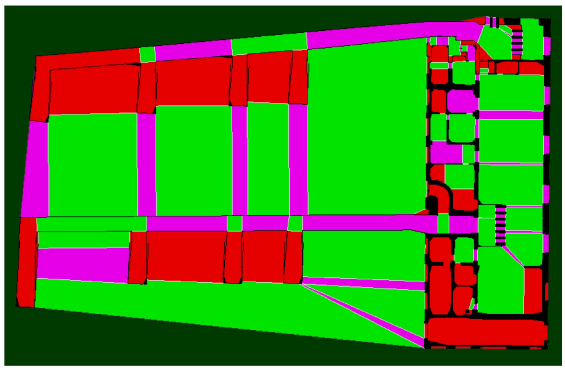
Figure 1: Informed hierarchical graph.



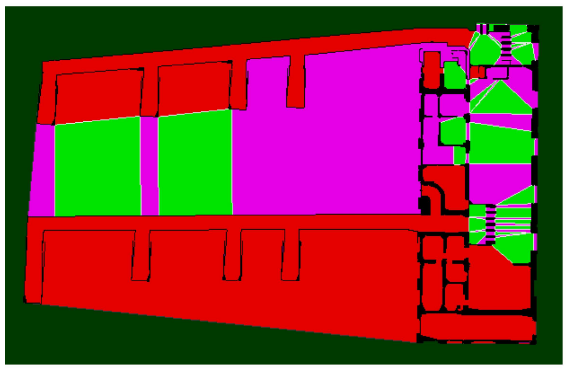
(a) Informed subdivision: *1526 cells*



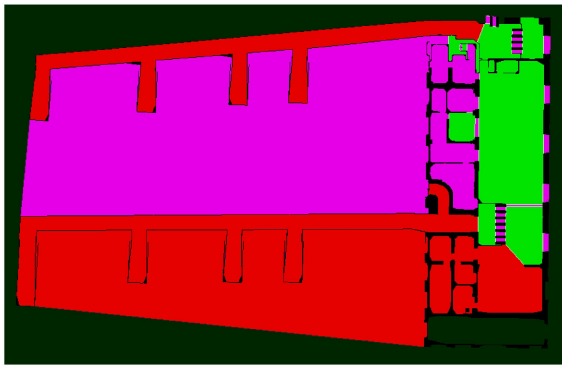
(b) First original abstraction: *255 groups*



(c) First improved abstraction: *153 groups*

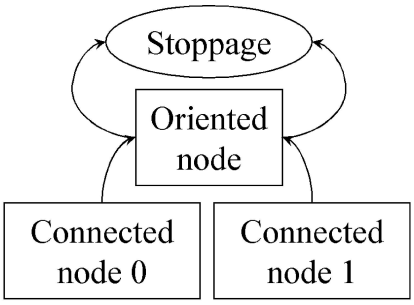


(d) Second original abstraction: *79 zones*

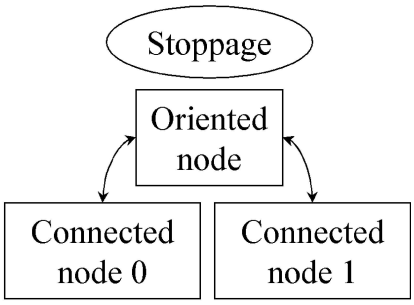


(e) Second improved abstraction: *40 zones*

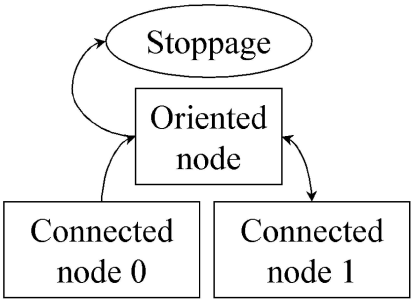
Figure 2: Three levels of abstraction of a train station environment. The left column corresponds to our original model, while our improved groupings are on the right. The same colour keys as for figure 1 are used (virtual areas are darker).



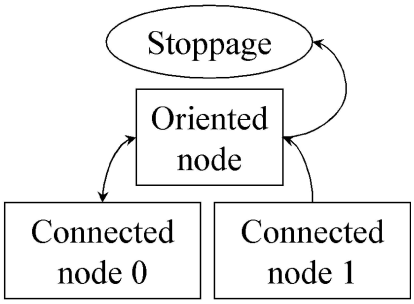
(a) Blocked oriented node



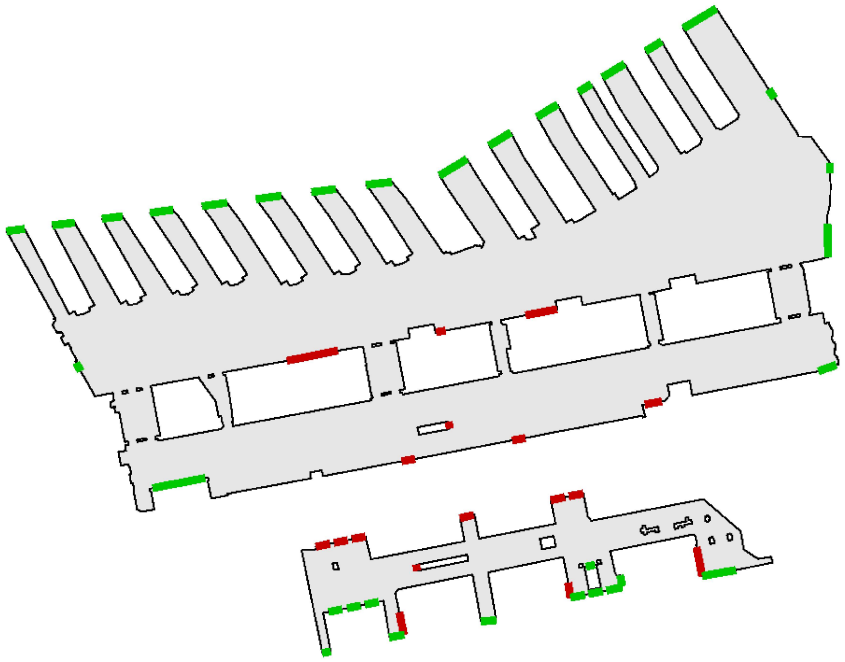
(b) Two way oriented node



(c) One way : *node 0 to node 1*

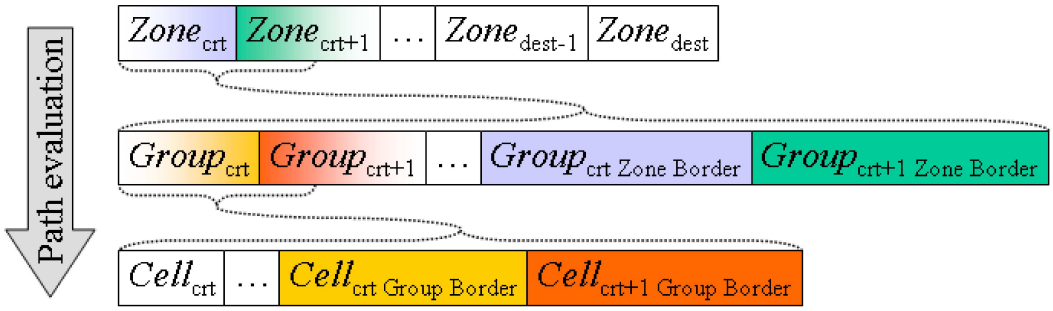


(d) One way : *node 1 to node 0*

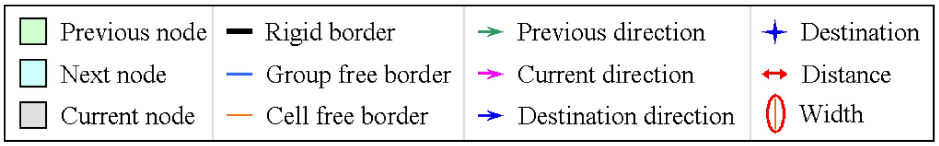


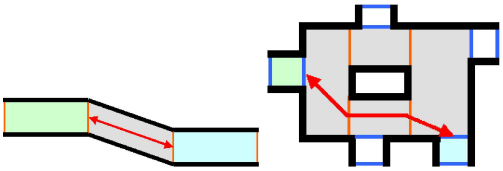
(e) Link (red) and access (green) nodes on the map of a two floor train station (St Lazare in France)

Figure 3: Principle (a-d) and example (e) of the oriented nodes.

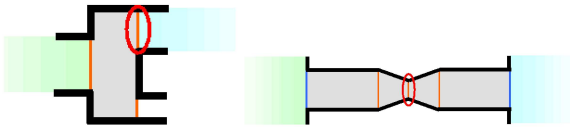


(a) Hierarchical path planning diagram

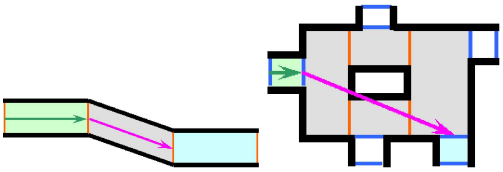




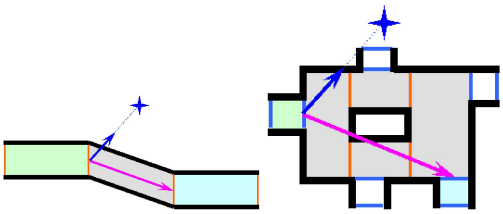
(b) Path length evaluation



(c) Path width evaluation



(d) Current direction evaluation



(e) Destination direction evaluation

Figure 4: Hierarchical path planning principle (a), and data evaluations (b-e) used for some criteria computation (*for each pair of figures, the cells evaluation is represented on the left, while the groups and zones evaluations are on the right*).