Stable Growing Neural Gas: a Topology Learning Algorithm based on Player Tracking in Video Games

F. Tence[b], L. Gaubert[a], J. Soler[a,b], P. De Loor[a], C. Buche[a,*]

[a]UEB / ENIB / LAB-STICC / CERV - 25 rue Claude Chappe, 29280 Plouzané, FRANCE
[b]Virtualys, 41 rue Yves Collet, 29220 Brest, FRANCE

Abstract

Characters in video games usually use a manually-defined topology of the environment to navigate. To evolve in an open, unknown and dynamic world, characters should not have pre-existing representations of their environment. In this paper, characters learn this representation by imitating human players. We here put forward a modified version of the Growing Neural Gas model (*GNG*) called Stable Growing Neural Gas (*SGNG*). The algorithm is able to learn how to use the environment from one or more teachers (players) by representing it with a graph. Unlike *GNG*, *SGNG* learning is in-line, reflecting the dynamic nature of the environment. The evaluation of the quality of the learned representations are detailed.

Keywords: Growing Neural Gas, imitation learning, topology representation, video games, agent.

*Corresponding author: buche@enib.fr
Email addresses: tence@enib.fr (F. Tence), gaubert@enib.fr (L. Gaubert), soler@enib.fr (J. Soler), deloor@enib.fr (P. De Loor), buche@enib.fr (C. Buche)

1. Introduction

This work contributes to designing a behavioral model for controlling believable characters [1, 2] in video games [3]. Characters are controlled by computer programs we call agents. We define a believable agent as a computer program able to control a virtual body in a virtual environment in such a way that other human users in the environment believe that the virtual body is controlled by a human user [4, 5].

Contrary to what is done in most video games [6, 7], agents' perception abilities must be similar to those of a player. Agents should be able to handle the flow of time, remembering information from the past and thinking ahead, enabling it to make plans. Finally, agents have to be able to evolve, changing their behavior to make them more efficient and believable [8, 9]. This evolution must be fast enough for the players not to notice it, making them feel as if they are playing against an evolved being.

In order to achieve behavior believability, the best method is imitation learning by which agents learn their behavior by observing one or more players [10, 8]. According to our definition of believability, this is the best way for agents to resemble players. Indeed, the aim of imitation learning is to make agents act like human players.

For agents to adapt to unknown environments, we enhanced the Growing Neural Gas model (*GNG*) [11]. *GNG* is an algorithm usually used to perform unsupervised tasks like clustering, interpolation or vector quantization. It represents data with a graph and automatically adapts the graph topology to the data. In our case, the data concerns one or several players' positions. For a given player, the data can be seen as a temporal series. Indeed, the position of the player at time t is dependent on its position at time $t-1$. The model can then learn a representation of the volume of the environment which also corresponds to how the players use that environment. The final representation used by the behavior model is composed of the graph nodes alone: each node represents a place where the agent can go but the edges do not provide any information about how to access one node from another [12].

GNG appears to be an interesting algorithm for learning maps by imitation because of its ability to learn topology. We argue that this aspect will make agents use the environment in a more human-like fashion. It also removes the burden from the map designers of manually defining the navigation graph. Its settings do not change over time. This is particularly useful as we want the algorithm to be able to adapt to changes in the map (for example a wall that collapses).

However, *GNG* was not designed to work with temporal series but with samples. Its node insertion policy does not suit our needs as it inserts nodes at constant iterations and in the area with the maximum error. The constant iteration insertion policy causes the network to grow infinitely over time and the insertion policy in the area containing the maximum error is not appropriate when working with temporal series. By proceeding in this way, we may oversample an area of the map not frequently visited by players. The primary goal of

this article is to provide a node insertion policy that overcomes these problems.

Moreover, although *GNG* has already been used in a video game [12], its characteristics and parameters have not been studied for this kind of application. The following questions have not been answered: What influence do the parameters have on learning? How should we choose the parameters? How can we accelerate the learning process? Answering these questions is the second objective of this article.

In the following sections we will present a modified version of *GNG*, called *SGNG*. Unlike *GNG*, *SGNG* learning is continuous, thus reflecting the dynamic nature of the environment. In addition, we will analyze the algorithm's characteristics, the quality of the learned topology and the influence of each parameter. Finally, we compare *GNG* with *SGNG*. To be fair, this comparison will be made with both temporal series and samples. We will see how our changes improve performance over temporal series and the impact of results on data samples. The paper is organized as follows: first we will present the *SGNG* (section 2). Then, we will present the results (section 3). Finally, we conclude (section 4).

2. *SGNG*

First we will explain the reasons for choosing the *GNG* algorithm to learn the representation of the environment and describe its principles in section 2.2. Then we will introduce some modifications for the algorithm to suit our needs in section 2.3.

2.1. Positioning

Models which control virtual humanoids use different kinds of representations to determine paths to go from one point to an other. All the meshes used to render the environment are too complex for agents to handle. As a consequence, classic approaches use a graph to represent accessible places, with nodes and paths between each place by edges (see figure 1). Current solutions tend to use a simple mesh, with different degrees of complexity to represent the accessible zones (see figure 2). The problem with the latter solution is that it requires an algorithm to find the optimal path between two points, a path which may not be natural or believable. Moreover, a graph solution is more suitable: each node of the graph can be used by the model to attract or repel the character.

In video games, classical bots are designed to follow pre-defined waypoints determined by the map designer. These bots need to have a waypoint file for each map, or a pathnode system embedded in the map. For example, *Quake 3 Arena* bots use an area awareness system file to move around the map, while *Counter-Strike* bots use a waypoint file. *Unreal Tournament*'s series bots use a pathnode system embedded in the map to navigate. To support the many community-created maps, some games include an automatic mesh generation system (for example *Valve* bots). The first time users attempt to play a custom map with bots, the generation system will build a navigation file for that map. Starting at a player spawn point, walkable space is sampled by "flood-filling" outwards from that spot, searching for adjacent walkable points. Finally,

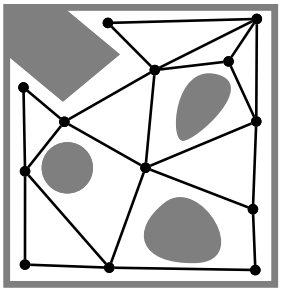

Figure 1: A simple environment (obstacles are in grey) represented by a graph. Nodes are represented by circles and edges by black lines. An avatar can move from one node to another only if the nodes are connected by an edge. Usually, an A* is used to find the path between two nodes.

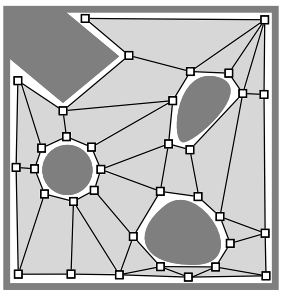

Figure 2: A simple environment (obstacles are in dark grey) represented by a mesh. The avatar can navigate in the zone defined by the mesh (in grey) because it knows there are no obstacles in this zone. Different algorithms can be used to find optimal paths.

dynamic bots are able to dynamically learn levels and maps as they play. RealBot, for *Counter-Strike*, is an example of this. However, this learning is not guided by human behavior. Navigation points obtained will therefore not produce believable behavior. The paths the bots use to go from one point in the environment to another do not resemble those human players would take. This problem comes not from the decision-making process itself, but from the representation of the environment it uses. Indeed, the bots use navigation points in the environment which may not accurately or naturally represent how players use the same environment.

In order to achieve optimum believability, we want the nodes of the graph to be learned by imitating human players (tracking human navigation). This work was done in [13] where nodes and a potential field were learned from humans playing a video game. Agents could then use this representation to navigate the game environment, following the field defined at each node. In order to learn the positions of the nodes, Thurau used an algorithm called *GNG*.

2.2. GNG Principle

GNG [11] is a graph model capable of incremental learning. Each node has a position (x, y, z) in the environment and has a cumulated error which measures how well the node represents its surroundings. The fewer errors a node has, the better it represents its surroundings. Each edge links two nodes and has an age informing us when it was last activated. This algorithm needs to be omniscient because the position of the human teacher needs to be known at any given time.

The principle of *GNG* is to modify its graph for each input of the teacher's position in order to alter the graph to match the teacher's position. The model can add or remove nodes and edges if they do not fit the behavior and change the position of the nodes to better represent the teacher's position. Figure 3 provides pseudo code of the algorithm and its corresponding steps in figure 4. In figure 4, we assume that the iteration number is a multiple of η.

while Number of nodes $\leq N_{max}$ **do**
  Get input position (4 (a))
  Pick the closest (n_1) and the second closest nodes (n_2) (4 (b))
  Create edge between n_1 and n_2 (4 (c)).
  If an edge already existed, reset its age to 0.
  Increase the error of n_1 (4 (d))
  Move n_1 and its neighbors toward the input (4 (e))
  Increase the age of all the edges emanating from n_1 by 1 (4 (f))
  Delete edges exceeding a certain age (4 (g))
  if Iteration number is a multiple of η **then**
    Find the maximum error node n_{max}
    Find the maximum error node n_{max2} among the neighbor of n_{max} (4 (h))
    Insert node between n_{max} and n_{max2} (4 (i))
    Decrease the error of n_{max} and n_{max2}
  end if
  Decrease each node's error by a small amount (4 (j))
end while

Figure 3: The *GNG* algorithm as defined in [11]. Comment (4 (X)) corresponds to step X from figure 4.

2.3. Modification of the GNG: SGNG

The version of *GNG* we use is modified as shown in figure 5. The two main problems with *GNG* is that it fails to handle temporal series appropriately, and it continues to grow over time.

The constant iteration insertion policy causes the network to grow infinitely over time. This problem can easily be overcome by increasing the number of nodes in the network, but how can we define this maximum number of nodes? Obviously, a large map probably needs more nodes than a smaller one. We

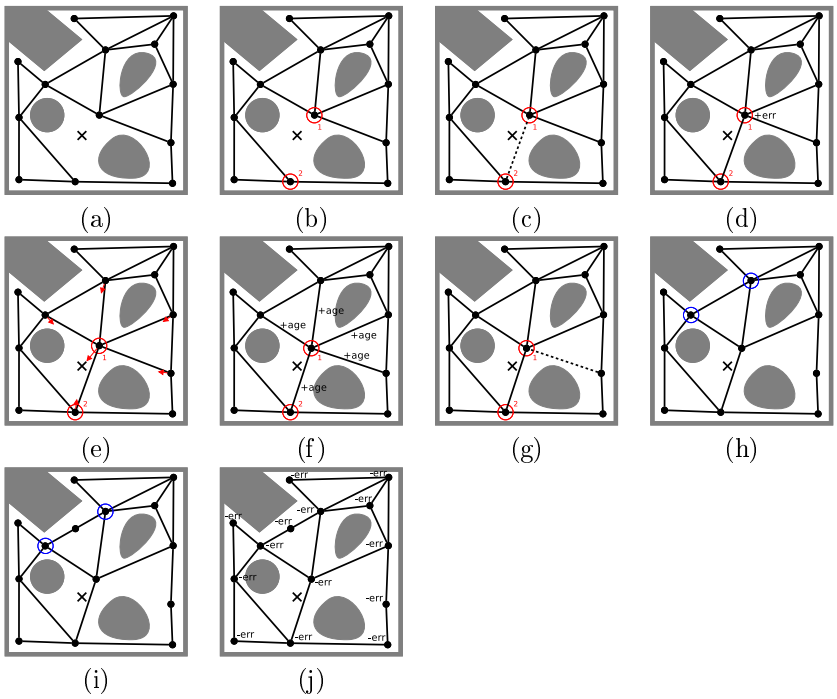

Figure 4: Steps of the *GNG* algorithm. The black cross is the input, black circles are the nodes of the *GNG* and black lines are the edges of the *GNG*. Gray shapes represent the obstacles in the environment.

could also stop the learning process when a suitable number of nodes is reached but we also need the *GNG* to adapt to variations in the use of the map. If the teacher suddenly uses part of the map which he/she has not yet explored, the *GNG* should be able to learn this new part even if it has been learning for a long time.

The insertion policy in the area containing the maximum error may oversample an area of the map not frequently visited by players. Inserting a node in the current area of the temporal series seems to be more appropriate.

We propose that instead of inserting a new node each η input, a node should be inserted when the error of a node is superior to a parameter $\overline{Err}$. In this way, we add nodes only in the area just visited by the player (avoiding maximum error area search) and we add nodes only when the network fails to fit the data suitably. As the error of each node is reduced by a small amount Err for each input, the modified *GNG* algorithm does not need a stopping criterion. Indeed, if there are many nodes which accurately represent the environment, the error added for the input will be small and for a set of inputs the total added error will be distributed among several nodes. Decreasing the error will avoid new nodes being added to the *GNG*, thus resulting in a stable state. However, if the teacher goes to a place in the environment he/she has never been before, the

added error will be high enough to counter the decay of the error, thus creating new nodes.

This algorithm has five parameters which influence the density of nodes, the quality of the representation, adaptability and convergence time:

- The attraction $\overrightarrow{attract_1}$ applied to n_1 toward (x, y, z)
- The attraction $\overrightarrow{attract_2}$ applied to the neighbors of n_1 toward (x, y, z)
- The error decay for nodes, Err
- The maximum error for the nodes, $\overline{Err}$
- The maximum age for the edges, $\overline{Age}$

SGNG has a complexity of $O(n)$ where n is the number of nodes in the graph. This should leave plenty of computing power for other algorithms. The algorithm is entirely capable of handling data from several teachers. By giving input from several teachers we should greatly increase learning speed. The only drawback with this technique is that different teachers can use the environment very differently. As a consequence, the learned *SGNG* may reflect neither of the individual teachers' behaviors.

3. Results

In section 3.1 we will first present the results and explain how we will measure their characteristics and quality. Then we will study the influence of each parameter in section 3.2. As learning speed is very important, we will explain some possible ways of increasing this speed in section 3.4. Finally, we will compare *GNG* and *SGNG* in section 3.5.

3.1. Measures and Representation of the Results

3.1.1. Application to UT2004

Tracking the position of the teacher in the game Unreal Tournament 2004 (UT2004) is easy using Pogamut [14]. *SGNG* can be given the position for learning. The difficulty is in finding parameters (see section 2.3) for *SGNG* to generate a representation with enough nodes for the agent to be able to move in the environment but not so many as to overload the agents with information. We have to choose these parameters empirically as they cannot be found analytically, nor can we use an optimization algorithm. The parameters giving representations similar to those usually found in video games are as follows:

- $\overrightarrow{attract_1}$: Attraction force applied to the closest node $first$ from the $input$ is 0.03 times the vector $input - first$
- $\overrightarrow{attract_2}$: Attraction force applied to $first$'s neighbors is 0.0006 times the vector $input - first$
- Err: The error decay for the nodes is 6 UU (Unreal Unit)

```
nodes ← {}
edges ← {}
while teacher plays do
   (x,y,z) ← teacher's position
   if |nodes| = 0 or 1 then
      nodes ← nodes ∪ {(x,y,z,error=0)}
   end if
   if |nodes| = 2 then
      edges ← {(nodes,age=0)}
   end if
   n1 ← closest((x,y,z),nodes)
   n2 ← secondClosest((x,y,z),nodes)
   edge ← edges ∪ {{n1,n2},age=0)}

   n1.error+=||(x,y,z)-n1||
   Attract n1 toward (x,y,z)
   ∀ edge ∈ edgesFrom(n1), edge.age++
   Delete edges older than Age
   Attract neighbors(n1) toward (x,y,z)
   ∀ node ∈ nodes, node.error-=Err

   if n1.error > Err then
      maxErrNei ← maxErrorNeighbour(n1)
      newNode ← between(n1,maxErrNei)
      n1.error/=2
      maxErrNei.error/=2
      newError ← n1.error+maxErrNei.error
      nodes ← nodes ∪ {(newNode,newError)}
   end if
end while
```

Figure 5: Algorithm used to learn the topology of the environment.

- $\overline{Err}$: The maximum error for the nodes is 16000 UU
- $\overline{Age}$: The maximum age for the edges is 75

In order to be able to make comparisons with other environments, the position in Unreal Tournament is given in UU (1 meter is roughly equal to 50 UU).

3.1.2. Representation of the Environment

With those parameters we trained two *SGNG* on two different environments. The first is a simple environment called *Training Day*; it is small and flat which is interesting in order to visualize the data in two dimensions. The second, called *Mixer*, is much bigger and more complex with stairs, elevators and slopes which make it interesting to see how *SGNG* behaves in three dimensions. The results are given in figure 6 for the simple environment and in figure 7 for the complex environment.

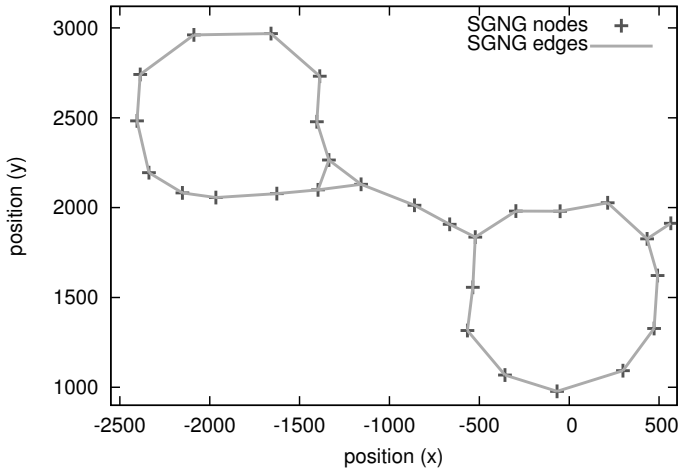

Figure 6: Result of a *SGNG* learned from a player for a simple environment after 30 minutes, top view.

As the complex environment is hard to visualize we focus mainly on the simple one in this section to make the explanation easier to understand. However, *SGNG* is able to represent complex environments just as well as simple environments.

3.1.3. Measures of Time Evolution

In order to study the quality of the learned topology, we first choose to compare the nodes of the *SGNG* with the navigation points placed manually by the environment's creators. Here, the objective is not to show that the behavior is more believable, but to ensure that it is acceptable from the point of view of a human expert. In the following, *node* will always refer to the *SGNG* and

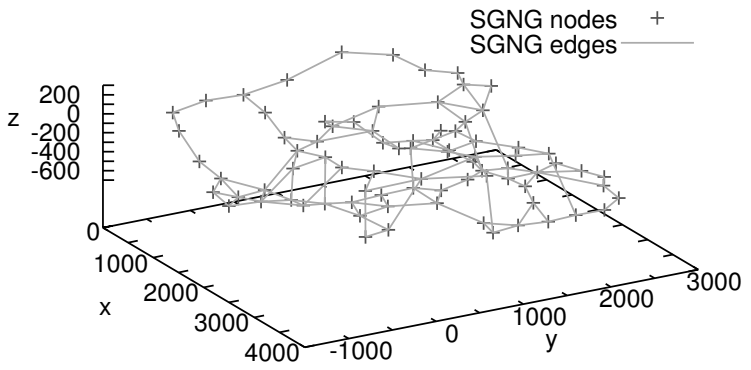

Figure 7: Result of a *SGNG* learned from a player for a complex environment after 1 hour.

navigation point to the representation made by the environment creators. We do not want the *SGNG* to exactly fit the navigation points, but they can help as an initial evaluation of the learned representation. In the game UT2004, we have those navigation points but our goal is for them to no longer be necessary for an agent to move around a new environment. The representations learned by the *SGNG* should also facilitate more believable behaviors as they already provide information on how players use the environment.

Figure 8 shows both the navigation points and the nodes of the *SGNG* for the simple environment. As we can see, the two representations look alike, indicating that the model is very effective at learning the layout of the environment. However, there are zones where the *SGNG*'s nodes are more concentrated than the navigation points and others where they are less so. We cannot tell now if it is good behavior or not as we need to evaluate an agent actually using this representation to see if it navigates well. Even in the less concentrated zones, the nodes are always close enough to be seen from their neighboring nodes, which at least makes node-to-node navigation possible.

As the qualitative evaluation of the representation is not sufficient, we introduce two measures, the principles of which are borrowed from statistics: *sensitivity* and *specificity*.

Sensitivity. Sensitivity measures how successfully the *SGNG* represents the part of the environment the teacher used, which can be seen as true positives. $mindist(a, B)$ is the minimum distance between point a and the points in set B. We computed this measure using the following formula:

$$Sensitivity \propto \frac{1}{\sum_i mindist(NP_i, nodes)} \tag{1}$$

Where NP_i is the i^{th} navigation point. The higher the value, the more sensitive the *SGNG*.

Specificity. Specificity measures how much the *SGNG* did not represent the part of the environment the teacher did not use, which can be seen as true negatives.

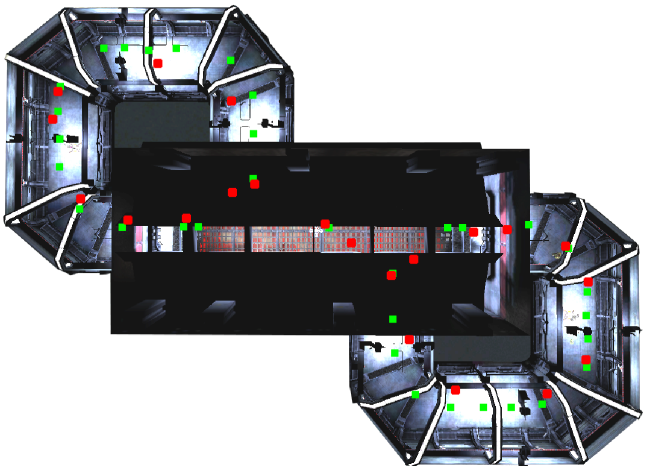

Figure 8: Comparison of nodes learned by the *SGNG* (in red) with the navigation points placed manually by the game developers (in green). The environment viewed from above is visible in the background.

We computed this measure using the following formula:

$$Specificity \propto \frac{Number\,of\,Nodes}{\sum_i mindist(Node_i, NPs)} \tag{2}$$

The higher the value, the more specific the *SGNG*.

Number of Nodes. In the following, we will also study the number of nodes the *SGNG* has, as we do not want the *SGNG* to have too many or too few nodes.

As the attraction applied to the nodes for each input is constant, the *SGNG* does not converge to a totally stable state. The small variations in the distance in figure 9 shows that the *SGNG* nodes still move. This is intentional as it enables the *SGNG* to adapt to a variation in the use of the environment: if the teacher suddenly uses part of the environment which he/she has not yet explored, the *SGNG* will still be able to learn this new part even if the *SGNG* has been learning for a long time.

Even if the *SGNG* does not converge, we do not want it to grow indefinitely. We also want the model to give similar results for similar behaviors. Figure 10 shows that even after 10 hours, the number of nodes is not higher than after 30 minutes and even that the solutions represent the environment better. It also shows that running the *SGNG* twice on similar behaviors gives similar results in terms of shape and number of nodes. Two *SGNG* learnings on the exact same data therefore give the exact same representations.

Models learned from different teachers do not give the same results and the time evolution is also quite different (see figure 11). Depending on the behavior, the *SGNG* can reach a stable state much more rapidly. For this reason we will only compare *SGNG* which learned on the exact same data in the following experiments.

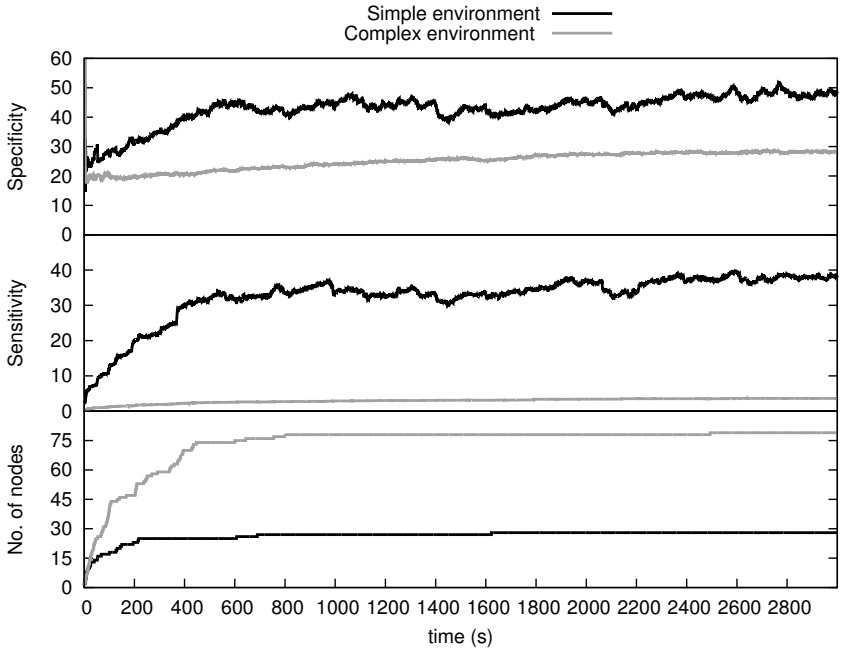

Figure 9: Time evolution of the *SGNG* number of nodes and the cumulated distance between the *SGNG* nodes and the navigation points.

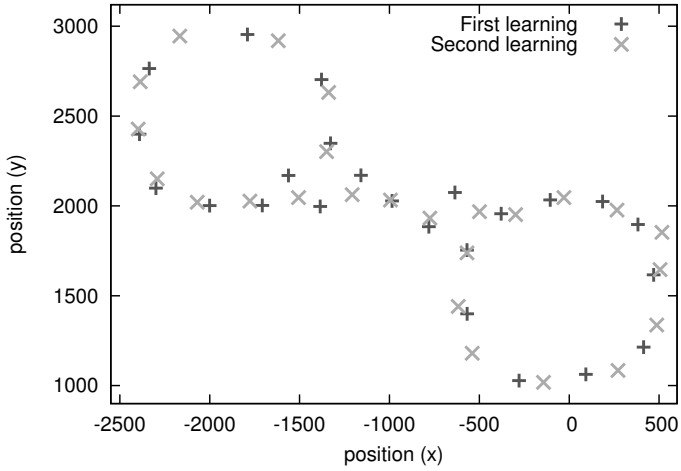

Figure 10: Comparison of two *SGNG* learned on the same environment after a very long training period of 10 hours.

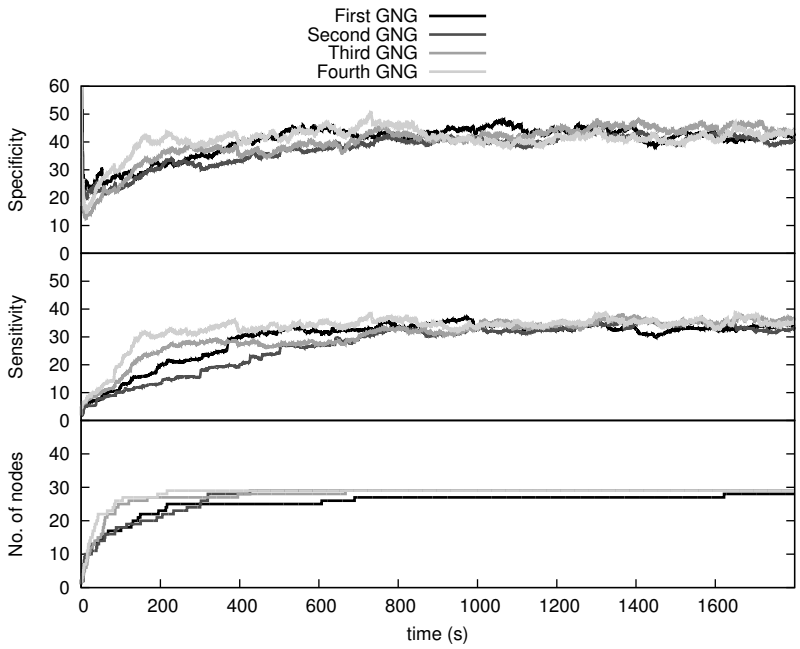

Figure 11: Comparison of four *SGNG* learned from different players.

3.2. Influence of the Parameters on Learning

We listed five parameters for the *SGNG* and gave values for our implementation to give similar results to representations used in UT2004. In order to be able to identify those parameters we will now describe the influence on the results for each of the parameters and on the time taken to generate an accurate representation. As each of the parameters influences several characteristics of the *SGNG*, we will explain how to choose the parameters at the end of each analysis.

3.2.1. Attraction of the Winning Node

When an input is given to the *SGNG*, the closest node, the winner, is moved by a certain amount toward this input. We will analyze the impact of the force of attraction applied to the winning node with the results in figure 12.

Number of Nodes. A high force makes the *SGNG* produce less nodes because, as the nodes move closer to the input, their error is lower. Nodes are thus less likely to reach the maximum error causing a new node to be added. Similarly, a low force makes the *SGNG* produce more nodes.

Sensitivity. A high force of attraction makes the nodes move more. As a result, the *SGNG* is less stable causing variations in the representation shown by less stable sensitivity. There is no big difference in final sensitivity, the greater sensitivity for the low force must come from the higher number of nodes.

Specificity. As is the case for sensitivity, a high force of attraction makes the value fluctuate showing that the *SGNG* is not very stable. However, the attraction must be strong enough to attract nodes which do not represent the environment: a low force of attraction generates too many useless nodes resulting in low specificity.

Time to Stability. A strong force of attraction makes the model converge to a stable state more rapidly because the nodes are more rapidly distributed through the whole environment. This can be seen with the stabilization of the three measures, which occurs earlier for a high value of the parameter than for a low value.

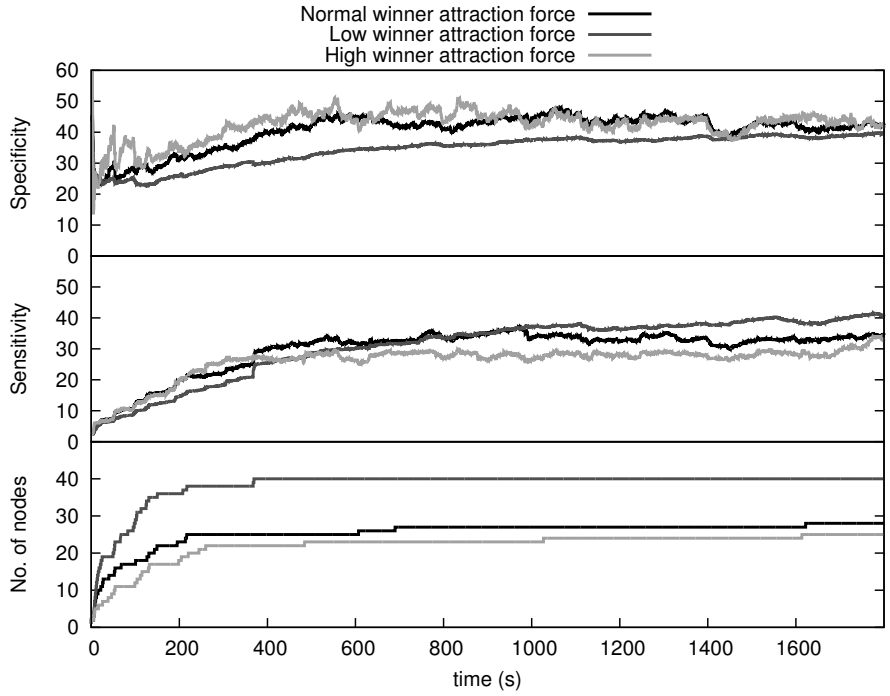

Figure 12: Comparison of the time evolution of *SGNG* which learned on the same data but have different values for the adjustment of the winning node toward the input. The higher the attraction, the faster the *SGNG* converges, the less nodes the *SGNG* has and the less stable the representation.

3.2.2. Attraction of the Nodes Neighbouring the Winning Node

Once the winning node has been attracted toward the input, the same occurs, with a lesser force, to all the nodes neighboring the winner. The time evolution for the three measures is given in the figure 13.

Number of Nodes. The applied force of the neighboring nodes has little or no impact on the number of nodes the *SGNG* creates.

Sensitivity. As is the case for the force applied to the winner, the higher the force applied to the neighboring nodes, the less stable the *SGNG*. With a low force, the model is more stable and has better sensitivity because the *SGNG* is more able to generalize.

Specificity. Conversely a low force of attraction does not enable the *SGNG* to move foreign nodes closer to the teacher. Consequently, specificity is better with a high force, able to move "false positive" nodes toward a more suitable location.

Time to Stability. The force of attraction of the neighboring nodes does not seem to influence the time the *SGNG* takes to reach a stable state. However, a high force makes the *SGNG* so unstable that it is difficult to determine when the structure of the *SGNG* has finished evolving.

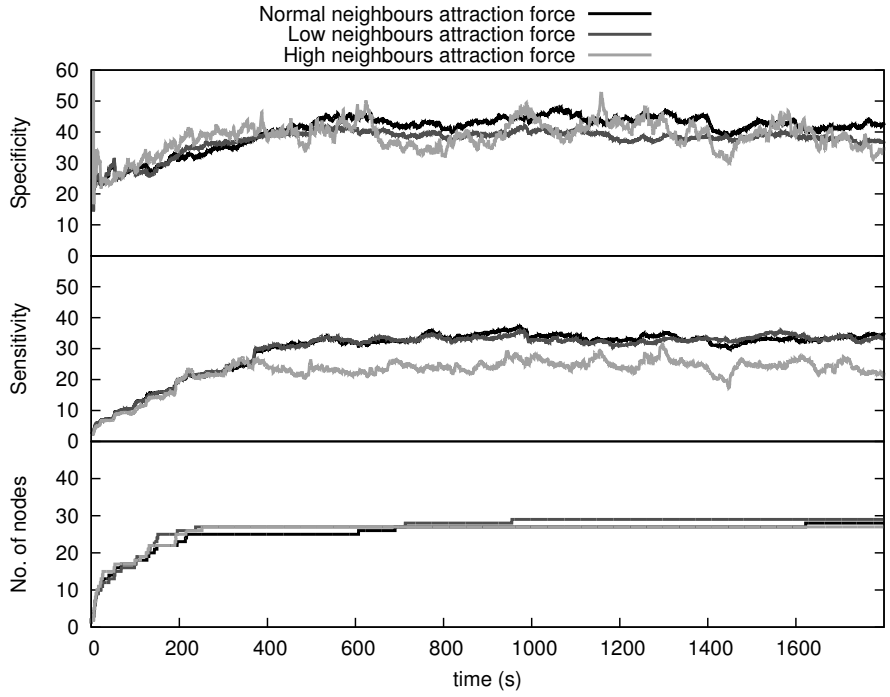

Figure 13: Comparison of the time evolution of *SGNG* which learned on the same data but which have different values for the adjustment of the neighbors of the winning node toward the input. The higher the attraction, the less stable the representation.

3.2.3. Maximum Error for Nodes

When the nodes have been moved, if the winning node exceeds $\overline{Err}$, a new node is created. We will now study the influence of this value, the maximum error for a node with the results given in figure 14.

Number of Nodes. This parameter has a big impact on the number of nodes. The lower the error, the more nodes are created because the more likely they are to exceed the value.

Sensitivity. Because a low error makes the *SGNG* produce more nodes, sensitivity is higher. Indeed, the higher the number of nodes, the more likely it is that a node will be close to a navigation point. Therefore the maximum error does not seem to directly influence sensitivity.

Specificity. As we have already seen, a low error makes the *SGNG* create nodes more often, even if they are not useful for the accuracy of the representation of the environment. Therefore, a low error increases *SGNG* specificity by reducing the number of useless nodes.

Time to Stability. The maximum error does not have a very significant impact on the time the *SGNG* needs to reach a stable state. It seems, however, that a low error makes the *SGNG* converge slightly faster. This may be due to rapid creation of nodes across the environment.

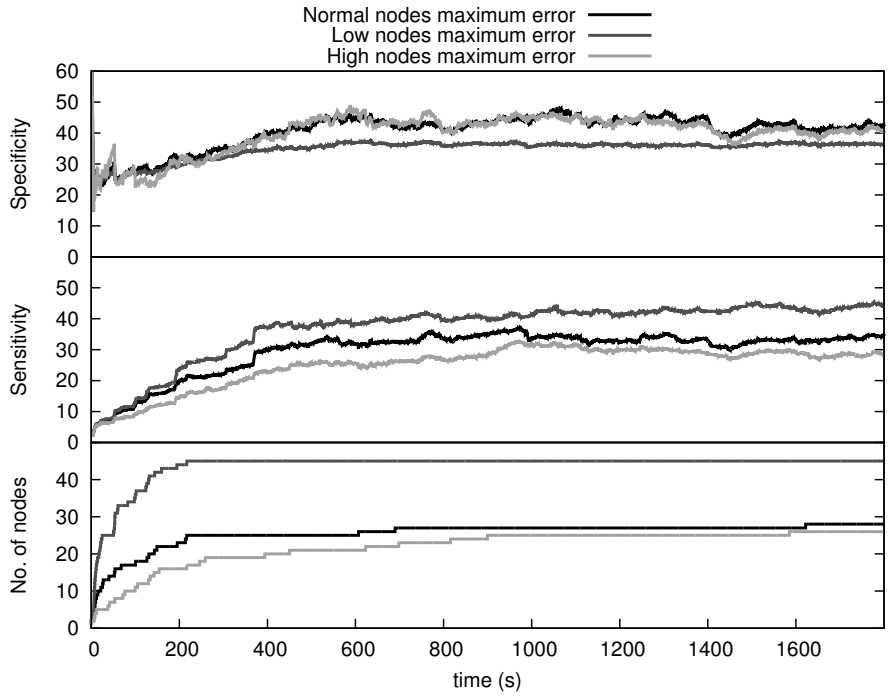

Figure 14: Comparison of the time evolution of *SGNG* which learned on the same data but which have different values for the maximum error admitted for nodes before the creation of another node in the *SGNG*. The higher the maximum error allowed for a node, the less nodes the *SGNG* has.

3.2.4. Maximum Age for Edges

Each time a winning node is selected, each edge connected to this node sees its age incremented by one. If the age exceeds $\overline{Age}$, the edge is deleted. The influence of this parameter will be studied in figure 15.

Number of Nodes. Maximum age has no significant influence on the number of nodes.

Sensitivity. Maximum age has no significant influence on sensitivity. Indeed, the slight difference comes from the difference in number of nodes.

Specificity. Surprisingly, this parameter has no influence on specificity. We could have expected edges lasting only a few time steps to leave nodes alone where they do not represent the environment correctly. A more detailed study needs to be done to validate this result.

Time to Stability. Maximum age has no influence on time the *SGNG* needs to be stable.

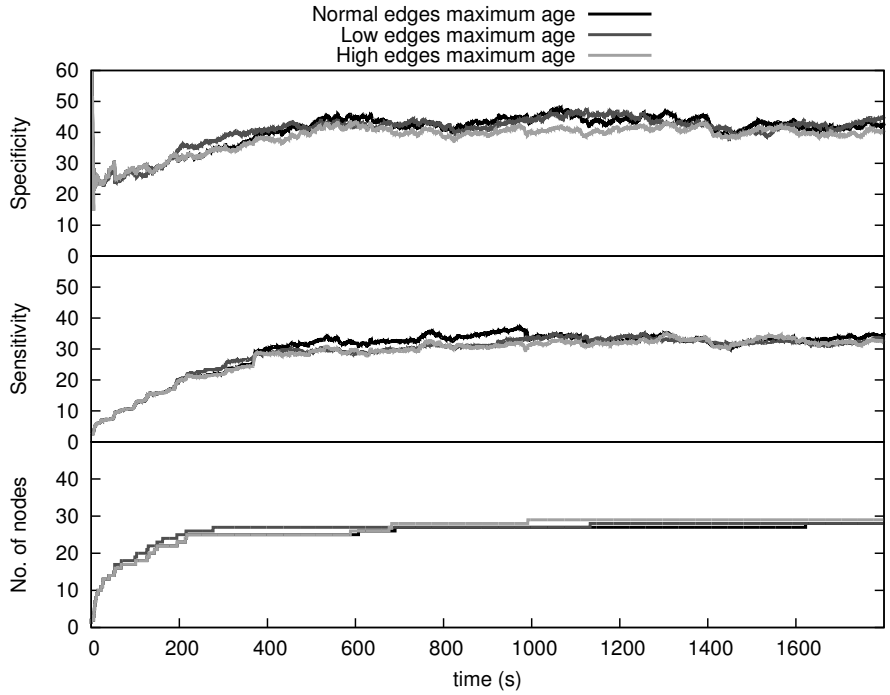

Figure 15: Comparison of the time evolution of *SGNG* which learned on the same data but which have different values for the maximum age admitted for edges before they are deleted. No change in the results was observed.

3.2.5. Error Decay

With each time step, the *SGNG* reduces the error for each node by Err. The time evolution of the three measures depending on the value of this parameters is given in figure 16.

Number of Nodes. A low error decay makes the nodes more likely to reach the $\overline{Err}$, thus making the *SGNG* create more nodes. Similarly, a higher decay makes the *SGNG* create less nodes.

Sensitivity. The difference in sensitivity mainly arises from the difference in the number of nodes, thus the parameter does not directly affect sensitivity.

Specificity. A high decay increases the specificity of the model. We are not sure of the reasons for this behavior. It may be because the model reaches its maximum number of nodes more quickly, thus enabling the *SGNG* to improve the representation without the disturbance of new nodes.

Time to Stability. The *SGNG* reaches a stable state faster with a high decay because the error shared between all the nodes is rapidly compensated by the decay. Similarly, a low decay makes the *SGNG* very slow to converge because new nodes are added long after learning begins.

3.3. How to Choose the Parameters

The $\overline{Age}$ parameter can be chosen easily as it does not influence the results. A value of 0 should be avoided as it would impede the attraction of neighboring nodes, which would alter the results.

A high Err can be then used to reduce the convergence time of the model. However this value is linked to the $\overline{Err}$, so if the decay is very high, the maximum error must be too.

A rather high force of attraction should be applied to the winner, keeping in mind that an excessively high value will make the *SGNG* less stable. If the teacher often discovers new areas, the force should be high in order to allow the *SGNG* to adapt more quickly.

Similarly, the force of attraction applied to the neighboring nodes should be high enough to avoid having nodes which would be useless or even harmful for believability. However, a high force will also make the *SGNG* less stable.

Finally, the $\overline{Err}$ can be chosen according to the value of Err so that the representation gives enough nodes for the agent to be able to navigate the environment but not so many as to overload it with information. The notion of "overloading" depends mainly on the behavioral model: if it is capable of handling a great deal of information, $\overline{Err}$ can be set to a low value.

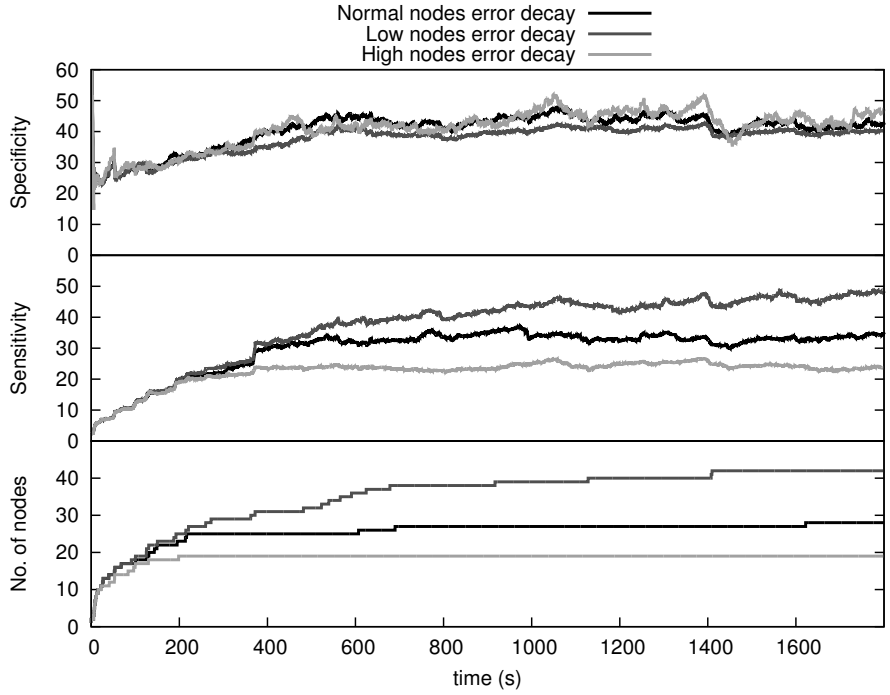

Figure 16: Comparison of the time evolution of *SGNG* which learned on the same data but which have different values for node error decay. The higher the error decay, the less nodes the *SGNG* has and the higher the specificity.

3.4. Increasing Representation Learning Speed

The speed with which the agents learn the representation of the environment is very important. If the agents do not have correct representation of the environment, they cannot navigate it and worse, they cannot learn any behavior. Indeed, the decision-making algorithm cannot associate actions to stimuli if the stimuli are not yet defined. We will see how to increase learning speed without compromising the quality of the representation.

3.4.1. Learning from Several Teachers

SGNG can handle input from several teachers. Figure 17 shows our three measures for *SGNG* trained by one, two, three and four teachers. Learning from multiple teachers is slightly faster. Moreover, learning with multiple teachers generates *SGNG* with better results in sensitivity and specificity, and yet the number of nodes is the same. As a consequence, *SGNG* should be learned from as many players as possible in order to optimize results. In addition, there should be more teachers in larger environments to accelerate learning.

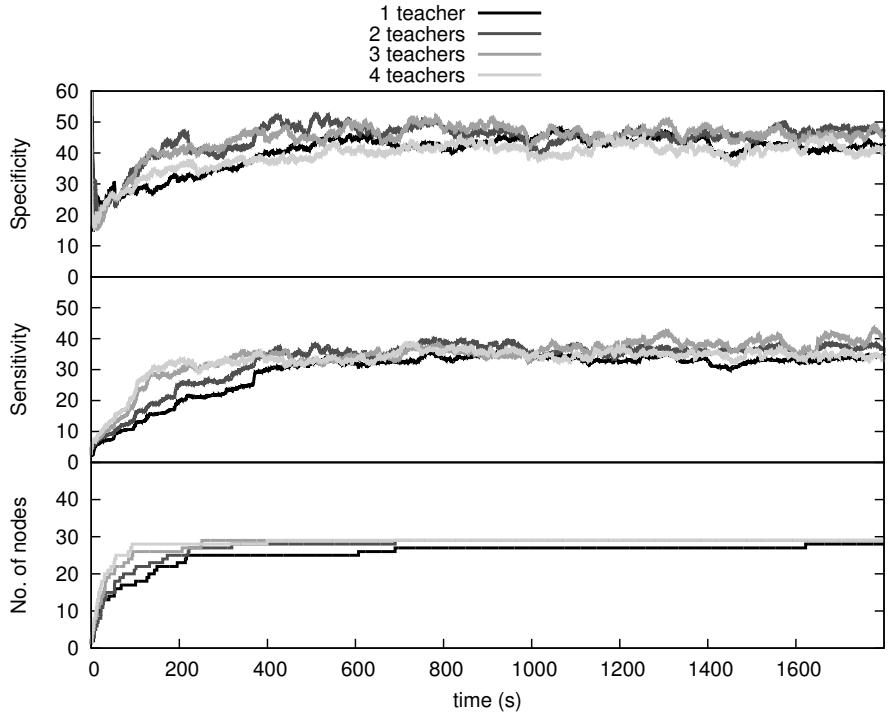

Figure 17: Comparison of the time evolution of *SGNG* which learned from 1, 2, 3 and 4 different teachers simultaneously. The more teachers there are the faster the learning, but also the more stable the representation.

3.4.2. Input Frequency

The last evaluation assesses the impact of the frequency at which the demonstrator's position is given to the *SGNG*. For the previous experiments, the frequency was set at 10Hz. Figure 18 shows the differences for 1, 5, 10 and 20 Hz (Pogamut does not allow higher than 20Hz). Results indicate that the higher the frequency, the faster the *SGNG* stabilizes and the better the results in terms of sensitivity and specificity. With high frequencies the number of nodes is a bit higher but the main problem is the stability of the representation. Sensitivity and specificity vary greatly at 20Hz. A variable frequency could be used to accelerate learning at the beginning and to avoid instability when most of the learning has been achieved.

SGNG proves to be very efficient at learning the topology of an environment though imitation. Learning is quite fast, even for complex environments. Although the model does not converge toward a unique and totally stable solution, the representations are similar in shape and accuracy. The advantage of this constant evolution is that the model can adapt quickly to changes in the use of the environment.

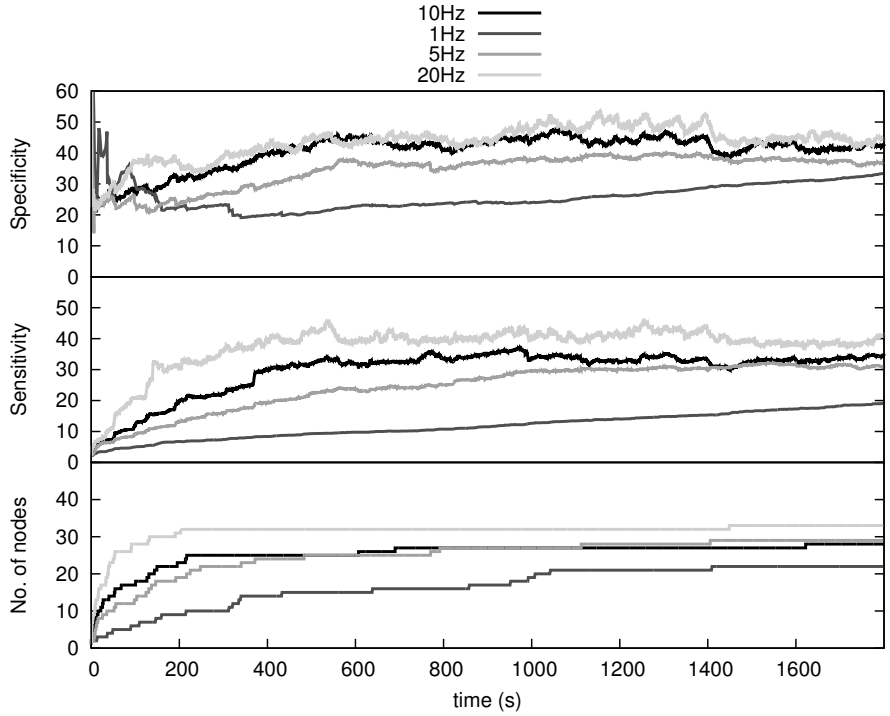

Figure 18: Comparison of the time evolution of *SGNG* learning at 1, 5, 10 and 20Hz. The higher the frequency, the faster the learning but the less stable the representation.

3.5. Comparison GNG/SGNG

3.5.1. Synthetic Data

To compare *GNG* and *SGNG*, it is more convenient to use synthetic data. In this way, we have a controlled environment in which the indicators described above will be more accurate. The environment used is a parametric butterfly curve for which we want to discretize the trajectory:

$$\rho = 100e^{cos(\theta)} - 2 * cos(4\theta) \tag{3}$$

We will use an additional indicator representing how the nodes are dispersed. Indeed, as we want to learn a navigation graph, we want the nodes to be well distributed thoughout the area. In other words, we want the density of nodes to be as constant as possible. We propose a new measure:

$$DensityHomogeneity \propto \frac{NumberofNodes}{\sum_i (mindist(Node_i, Nodes) - \overline{NearNeiDist})^2} \tag{4}$$

where $\overline{NearNeiDist}$ is the mean of $mindist(Node_i, Nodes)$.

As a first comparison of this synthetic environment, we empirically found *GNG* settings to yield the highest scores possible on indicators for a number of nodes arbitrarily set to 100. Settings common to both *SGNG* and *GNG* were chosen identically and the $\overline{Err}$ parameter was chosen so that the number of nodes stabilized at around 100. Figures 19 and 20 illustrate the results. We can see that sensitivity is better even when there are twice as many nodes in *GNG* than in *SGNG*. As the number of nodes in *GNG* became higher than for the *SGNG*, *GNG* specificity overtook that of *SGNG*. However, for the same number of nodes, *SGNG* performed better. The density variation indicator is always better in *SGNG* than in *GNG*.

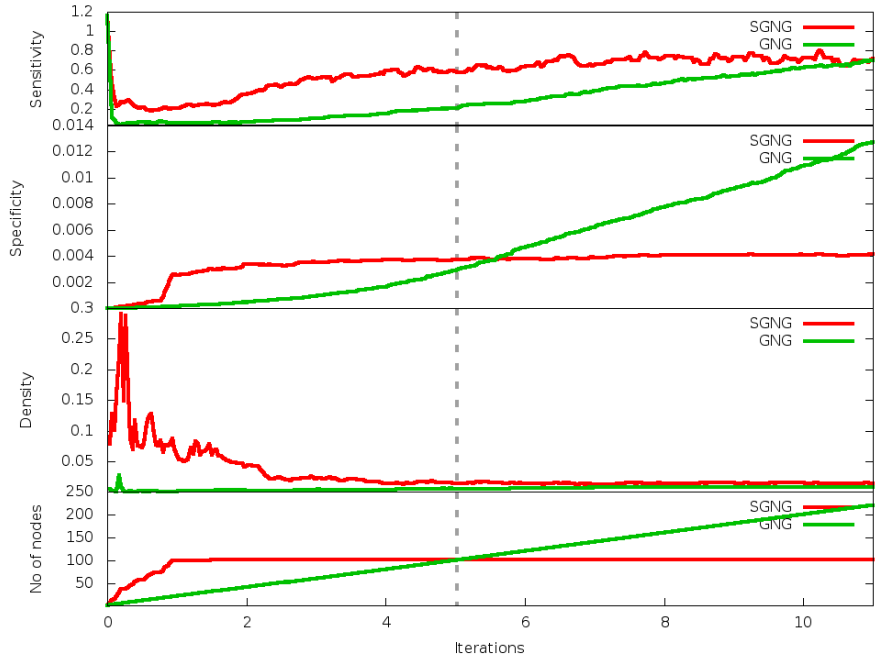

Figure 19: Evolution of Indicators for *SGNG* and *GNG* on synthetic temporal data

Figure 21 provides performance indicators for different values of parameters η for *GNG* and $\overline{Err}$ for *SGNG*. The performances are compared when the number of nodes of *GNG* is equal to the performance of *SGNG*. *SGNG* almost always outperforms *GNG*.

Next, we did the same experiment with the same parameters but with a sample taken from the butterfly curve. The results rea presented in figures 22 and 23. As expected, this time, *GNG* outperforms *SGNG*. But both sensitivity and specificity are very close when *GNG* and *SGNG* have the same number

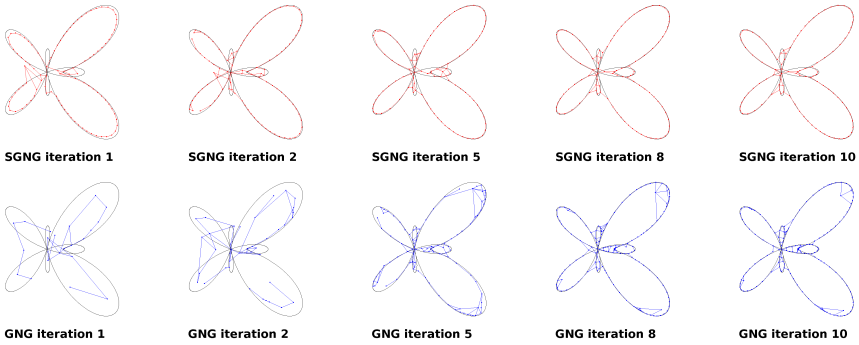

Figure 20: Evolution of *SGNG* (top) and *GNG* (bottom) on synthetic temporal data

of nodes. Convergence is faster with *SGNG* than with *GNG*. We can see in figure 23 that the rapid growth of *SGNG* generates nodes far from the data distribution, thus decreasing specificity. This could be avoided by adding a simple removal policy. We expect that by making a few changes to *SGNG*, we might be able to provide an algorithm effective for both temporal series and data samples with better convergence speed than *GNG*.

3.5.2. UT2004 Environment

Figure 24 illustrates a comparison between *GNG* and *SGNG*, using three measures during learning in the simple environment.

The problem with *GNG* is that the number of nodes increases steadily so that the representation quickly becomes too large to be managed by the decision-making model. Unlike *GNG*, *SGNG* stabilizes quickly, which is what we are aiming for. Over time, the graph shows that *SGNG* provides better sensitivity faster. However, as the number of nodes *GNG* increases continuously, the sensitivity of *GNG* eventually exceeds that of *SGNG*. Note that for the same number of nodes (vertical dotted line), the sensitivity of *SGNG* is greater. The last measurement is specificity, and again the *SGNG* converges faster than the *GNG*. After a while the two representations become identical.

Overall, *SGNG* performs better than *GNG*, giving good results faster and generating a more stable solution. These two advantages, speed and stability, allow the model to quickly generate a reliable representation.

3.6. Discussion

SGNG behaves very well using input positions from players. It is possible to adjust the graph's level of detail making the nodes much closer and numerous or, on the contrary, reducing their number. The model quickly converges to a solution; in about 10 minutes for a simple environment and 25 minutes for a complex one. It is possible to increase convergence speed by imitating more than one player, halving learning time in the better cases. Finally, the model

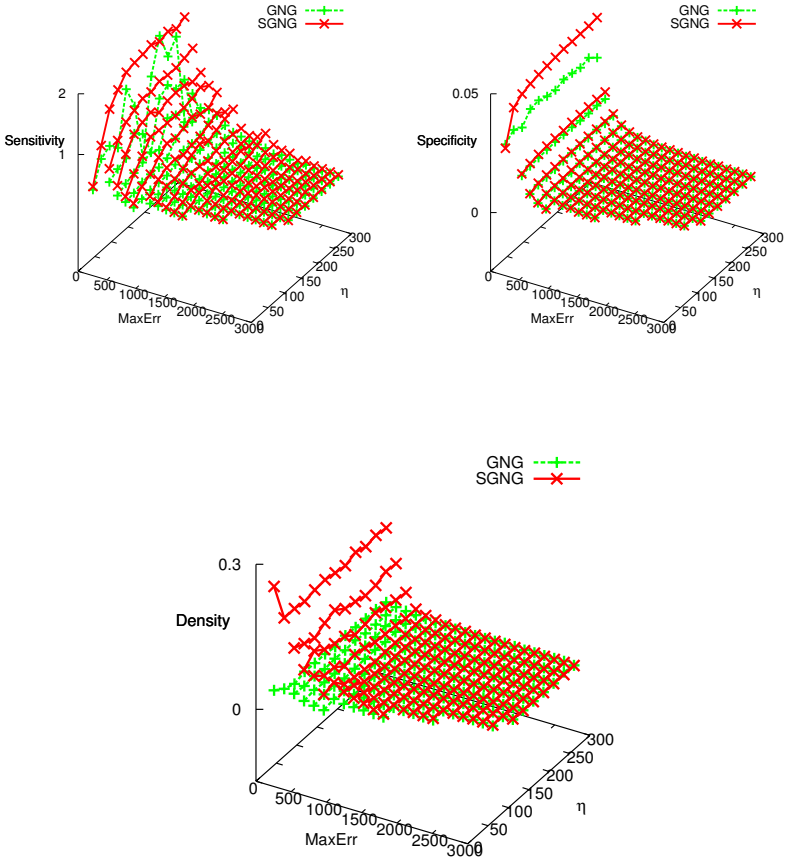

Figure 21: *SGNG* and *GNG* performance using different parameters.

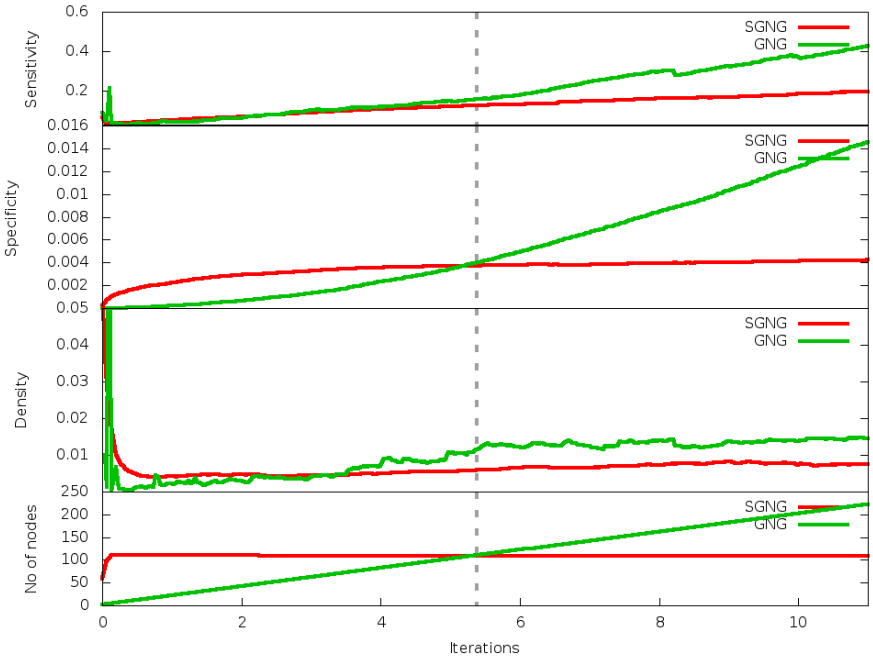

Figure 22: Evolution of indicators in $SGNG$ and GNG on a synthetic sample

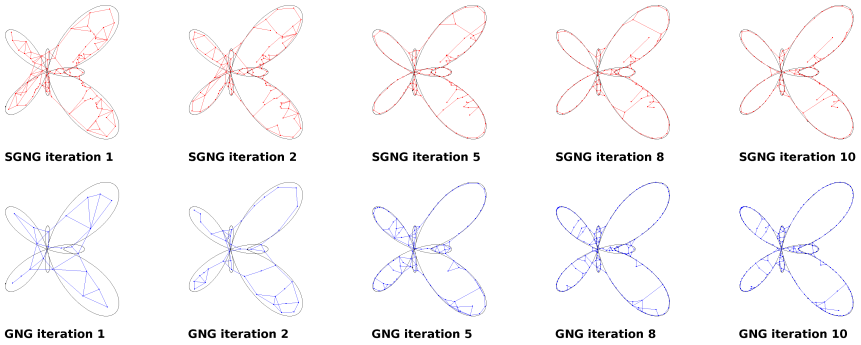

Figure 23: Evolution of $SGNG$ (top) and GNG (below) on a synthetic sample

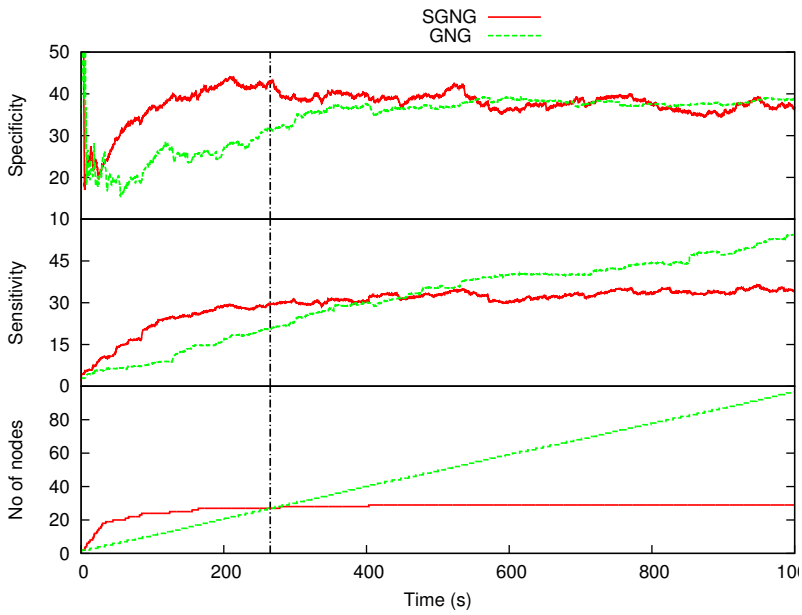

Figure 24: Comparison *GNG*/*SGNG*

can be left to learn for a very long period of time without fear of it growing indefinitely. It is even recommended to let the model learn for it to be able to learn new places in the environment if players begin visiting them only later in the game. However, the information gathered and given to the agent is limited to the nodes of the graph.

4. Conclusion

In order to make the agents adapt to unknown environments without help from programmers, we wanted them to learn the layout of the environment by imitation. As our main objective is believability, we also want the representation to reflect how players use the environment, to make it easier for the agents to reproduce these behaviors.

In order to represent and to learn the environment, we modified a model named *GNG* which updates a graph according to input coordinates. The graph stretches and grows to cover the entire space where the player has been observed to go. The model was modified to be able to learn continuously from players without growing indefinitely. However, it is important for the graph to be able to grow later in the game if the teacher begins to use a new part of the environment.

SGNG behaves very well with the input positions from players. We studied of each of the *SGNG* parameters and explained their influence. The representation

learned is very similar to usual representations found in video games which are used by the agents to navigate the environment. Agents are thus able to adapt to totally unknown environments.

The next step is to choose or design a decision-making model using the results of *SGNG*. The central notion will continue to be believability. To be able to achieve optimum believability, we want agents to behave like human-controlled virtual characters. Indeed, there is no better example of believable behavior than human behavior itself. Thus, learning and particularly imitation learning seems the most appropriate solution. We will put forward a two-step method to develop new models for believable agents. First we must find the criteria for believability and define how it can be evaluated. We will then design the decision-making model and the learning algorithm.

The final step will be to evaluate our work by gathering a pool of players from whom the agents can lean their behavior. When learning is complete, we will try to assess the believability of our agent. As believability is subjective, it is very difficult to evaluate [4]. An overview of evaluation methods has already been compiled in [15]. Working with psychologists, we will carefully study how to evaluate believability. Indeed, believability experiments must involve people expressing their feelings about the test subjects. We will do a test based on [16, 17], using humans as judges.

References

[1] A. B. Loyall, Believable agents: building interactive personalities, Ph.D. thesis, Carnegie Mellon University (1997).

[2] J. Bates, The nature of characters in interactive worlds and the Oz project, Tech. Rep. CMU-CS-92-200, School of Computer Science, Carnegie Mellon University (1992).

[3] A. McMahan, Immersion, engagement and presence, The video game theory reader (2003) 67–86.

[4] D. Livingstone, Turing's test and believable AI in games, Computers in Entertainment 4 (1) (2006) 6.

[5] F. Tence, C. Buche, P. De Loor, O. Marc, The challenge of believability in video games: Definitions, agents models and imitation learning, in: W. Mao, L. Vermeersch (Eds.), 2nd Asian Conference on Simulation and AI in Computer Games (GAMEON-ASIA'10), Eurosis, 2010, pp. 38–45.

[6] S. Cass, Mind games, IEEE Spectrum 39 (12) (2002) 40–44.

[7] B. Mac Namee, Proactive persistent agents: using situational intelligence to create support characters in character-centric computer games, Ph.D. thesis, University of Dublin (2004).

[8] C. Thurau, T. Paczian, C. Bauckhage, Is bayesian imitation learning the route to believable gamebots?, in: GAMEON-NA'2005, 2005, pp. 3–9.

[9] B. Gorman, M. Humphrys, Imitative learning of combat behaviours in first-person computer games, in: Proceedings of CGAMES 2007, the 11th International Conference on Computer Games: AI, Animation, Mobile, Educational & Serious Games, 2007.

[10] S. Schaal, Is imitation learning the route to humanoid robots?, Trends in Cognitive Sciences 3 (6) (1999) 233–242.

[11] B. Fritzke, A growing neural gas network learns topologies, in: Advances in Neural Information Processing Systems 7, MIT Press, 1995, pp. 625–632.

[12] C. Thurau, C. Bauckhage, G. Sagerer, Imitation learning at all levels of game-AI, in: Proceedings of the International Conference on Computer Games, Artificial Intelligence, Design and Education, 2004, pp. 402–408.

[13] C. Thurau, C. Bauckhage, G. Sagerer, Learning human-like movement behavior for computer games, in: Proceedings of the 8th International Conference on the Simulation of Adaptive Behavior (SAB'04), 2004.

[14] O. Burkert, R. Kadlec, J. Gemrot, M. Bida, J. Havlicek, M. Dorfler, C. Brom, Towards Fast Prototyping of IVAs Behavior: Pogamut 2, in: Intelligent Virtual Agents, Vol. 4722, Springer, 2007, pp. 362–363.

[15] F. Tence, C. Buche, P. De Loor, O. Marc, The challenge of believability in video games: Definitions, agents models and imitation learning, in: W. Mao, L. Vermeersch (Eds.), 2nd Asian Conference on Simulation and AI in Computer Games (GAMEON-ASIA'10), Eurosis, 2010, pp. 38–45.

[16] A. Turing, Computing machinery and intelligence, Mind 59 (236) (1950) 433–460.

[17] B. Gorman, C. Thurau, C. Bauckhage, M. Humphrys, Believability testing and bayesian imitation in interactive computer games, in: From Animals to Animats 9, Vol. 4095, Springer, 2006, pp. 655–666.