

Navigation *IML*

Cédric Buche

ENIB

11 septembre 2019

Page 1 :

Goal

Navigation

- ▷ determine paths to go from one point to another.



└ Goal

Page 2 :

Goal

Navigation

- > determine paths to go from one point to another.



1 Classical approaches

- Mesh
- Graph

2 Video games

- Quake 3 Arena bots
- Counter-Strike bots
- Unreal Tournament's series bots
- Valve bots

3 GNG

4 SGNG

2019-09-11

Page 3 :

1 Classical approaches

- Mesh
- Graph

2 Video games

- Quake 3 Arena bots
- Counter-Strike bots
- Unreal Tournament's series bots
- Valve bots

3 GNG

4 SGNG

2019-09-11

Page 4 :

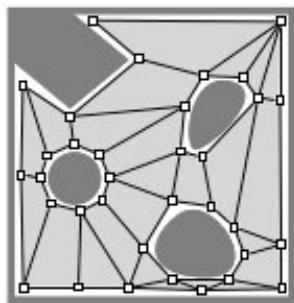


FIGURE – A simple environment (obstacles are in dark grey) represented by a mesh. The avatar can navigate in the zone defined by the mesh (in grey) because it knows there are no obstacles in this zone. Different algorithms can be used to find optimal paths.

Problems

- ▷ requires an algorithm to find the optimal path between 2 points
- ▷ a path which may not be natural or believable
- ▷ All the meshes used to render the environment are too complex for agents to handle.



FIGURE – A simple environment (obstacles are in dark grey) represented by a mesh. An 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.

Problems
▷ requires an algorithm to find the optimal path between 2 points
▷ a path which may not be natural or believable
▷ All the meshes used to render the environment are too complex for agents to handle.

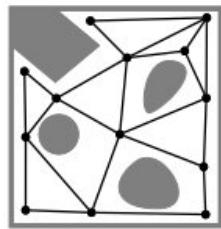


FIGURE – A simple environment (obstacles are in grey) represented by a graph. Nodes are represented by circles and edges by black lines. An avatar can move from one node to another only if the nodes are connected by an edge. Usually, an A* is used to find the path between two nodes.

FIGURE – 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.

Page 7 :

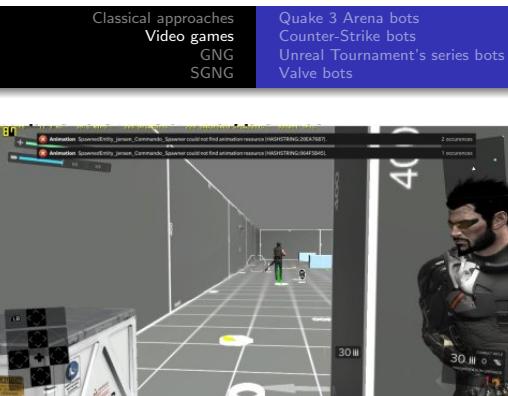


FIGURE – Classical bots are designed to follow pre-defined waypoints determined by the map designer.

Problems

- ▷ These bots need to have a waypoint file for each map, or a pathnode system embedded in the map.



FIGURE – Classical bots are designed to follow pre-defined waypoints determined by the map designer.

Problems

- ▷ These bots need to have a waypoint file for each map, or a pathnode system embedded in the map.

Page 8 :

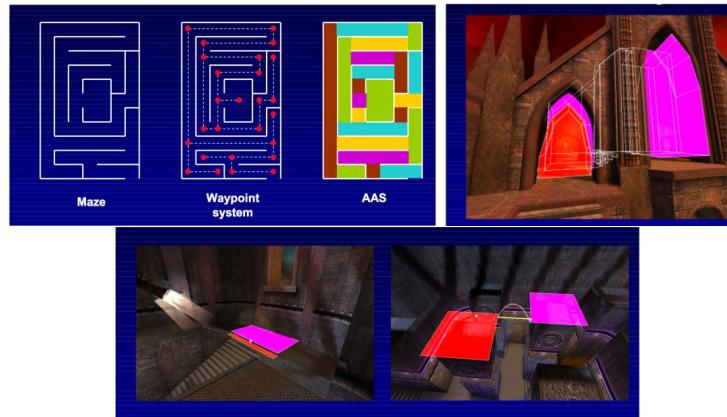


FIGURE – Quake 3 Arena bots use an area awareness system file to move around the map

- ▷ The structure of the levels is analyzed, which is used to extract volumes of space that are navigable (e.g. walkable surfaces, swimming areas).
- ▷ A process then analyzes these areas for connections by simulating potential movement from each of them, and then clusters the areas together to allow hierarchical path-finding
- ▷ Quake 3 uses a form of pathfinding within the AAS graph that relies on the clusters of areas.
- ▷ Within these clusters, the paths to common destinations are pre-calculated and stored for later use.

2019-09-11 | IML | Video games | Quake 3 Arena bots

FIGURE – Quake 3 Arena bots use an area awareness system file to move around the map

Page 9 :

2019-09-11 | IML | Video games | Quake 3 Arena bots

- ▷ The structure of the levels is analyzed, which is used to extract volumes of space that are navigable (e.g. walkable surfaces, swimming areas)
- ▷ A process then analyzes these areas for connections by simulating potential movement from each of them, and then clusters the areas together to allow hierarchical path-finding
- ▷ Quake 3 uses a form of pathfinding within the AAS graph that relies on the clusters of areas.
- ▷ Within these clusters, the paths to common destinations are pre-calculated and stored for later use.

Page 10 :

Classical approaches
Video games
GNG
SGNG

Quake 3 Arena bots
Counter-Strike bots
Unreal Tournament's series bots
Valve bots

Counter-Strike bots use a waypoint file.

C. BUCHE - buche@enib.fr

IML

11 / 46



FIGURE – Unreal Tournament's series bots use a pathnode system embedded in the map to navigate

2019-09-11 IML
└ Video games
 └ Counter-Strike bots

Counter-Strike bots use a waypoint file.

Page 11 :

2019-09-11 IML
└ Video games
 └ Unreal Tournament's series bots



FIGURE – Unreal Tournament's series bots use a pathnode system embedded in the map to navigate

Page 12 :

C. BUCHE - buche@enib.fr

IML

12 / 46

- ▷ To support the many community-created maps, some games include an automatic mesh generation system
- ▷ 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.

- ▷ To support the many community-created maps, some games include an automatic mesh generation system
- ▷ 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, 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.

- ▷ Finally, 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.

1 Classical approaches

- Mesh
- Graph

2 Video games

- Quake 3 Arena bots
- Counter-Strike bots
- Unreal Tournament's series bots
- Valve bots

3 GNG

4 SGNG

Page 15 :

Graph model

- ▷ incremental learning
- ▷ node
 - ◊ position (x, y, z)
 - ◊ cumulated error (measures how well the node represents its surroundings)
- ▷ edge
 - ◊ links two nodes
 - ◊ an age informing us when it was last activated.
- ▷ Principle : 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.

Page 16 :

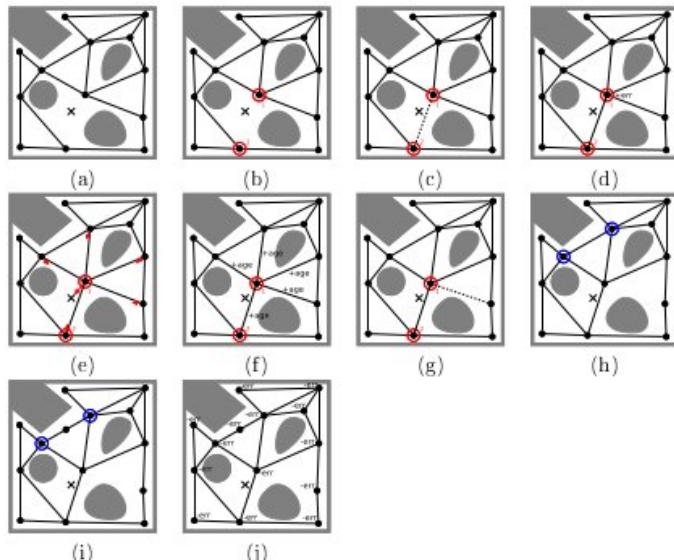
Video : GNG.mp4
Video : moons.mov

GNG - algorithm

```

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

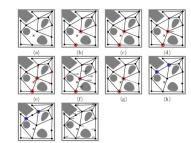
```



```

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

```



```

import numpy as np
from scipy import spatial
import networkx as nx
import matplotlib.pyplot as plt
from sklearn import decomposition

class GrowingNeuralGas:

    def __init__(self, input_data):
        self.network = None
        self.data = input_data
        self.units_created = 0

    def find_nearest_units(self, observation):
        distance = []
        for u, attributes in self.network.nodes(data=True):
            vector = attributes['vector']
            dist = spatial.distance.euclidean(vector, observation)
            distance.append((u, dist))
        distance.sort(key=lambda x: x[1])
        ranking = [for u, dist in distance]
        return ranking

    def prune_connections(self, a_max):
        for u, v, attributes in self.network.edges(data=True):
            if attributes['age'] > a_max:
                self.network.remove_edge(u, v)
        for u in self.network.nodes():
            if self.network.degree(u) == 0:
                self.network.remove_node(u)

```

C. BUCHE - buche@enib.fr | IML

19 / 46

Page 19 :

```

import numpy as np
from scipy import spatial
import networkx as nx
import matplotlib.pyplot as plt
from sklearn import decomposition

class GrowingNeuralGas:

    def __init__(self, input_data):
        self.network = None
        self.data = input_data
        self.units_created = 0

    def find_nearest_units(self, observation):
        distance = []
        for u, attributes in self.network.nodes(data=True):
            vector = attributes['vector']
            dist = spatial.distance.euclidean(vector, observation)
            distance.append((u, dist))
        distance.sort(key=lambda x: x[1])
        ranking = [for u, dist in distance]
        return ranking

    def prune_connections(self, a_max):
        for u, v, attributes in self.network.edges(data=True):
            if attributes['age'] > a_max:
                self.network.remove_edge(u, v)
        for u in self.network.nodes():
            if self.network.degree(u) == 0:
                self.network.remove_node(u)

```

```

def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=False):
    # logging variables
    accumulated_local_error = []
    global_error = []
    network_order = []
    network_size = []
    total_units = []
    self.units_created = 0
    # 0. start with two units a and b at random position w_a and w_b
    w_a = [np.random.uniform(-2, 2) for _ in range(np.shape(self.data)[1])]
    w_b = [np.random.uniform(-2, 2) for _ in range(np.shape(self.data)[1])]
    self.network = nx.Graph()
    self.network.add_node(self.units_created, vector=w_a, error=0)
    self.units_created += 1
    self.network.add_node(self.units_created, vector=w_b, error=0)
    self.units_created += 1

```

C. BUCHE - buche@enib.fr | IML

20 / 46

Page 20 :

```

def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=False):
    # logging variables
    accumulated_local_error = []
    global_error = []
    network_order = []
    network_size = []
    total_units = []
    self.units_created = 0
    # 0. start with two units a and b at random position w_a and w_b
    w_a = [np.random.uniform(-2, 2) for _ in range(np.shape(self.data)[1])]
    w_b = [np.random.uniform(-2, 2) for _ in range(np.shape(self.data)[1])]
    self.network = nx.Graph()
    self.network.add_node(self.units_created, vector=w_a, error=0)
    self.units_created += 1
    self.network.add_node(self.units_created, vector=w_b, error=0)
    self.units_created += 1

```

```
def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=False):
...
# 1. iterate through the data
sequence = 0
for p in range(passes):
    print('Pass %d' % (p + 1))
    np.random.shuffle(self.data)
    steps = 0
    for observation in self.data:
        # 2. find the nearest unit s_1 and the second nearest unit s_2
        nearest_units = self.find_nearest_units(observation)
        s_1 = nearest_units[0]
        s_2 = nearest_units[1]
        # 3. increment the age of all edges emanating from s_1
        for u, v, attributes in self.network.edges_iter(data=True,
            nbunch=[s_1]):
            self.network.add_edge(u, v, age=attributes['age']+1)
        # 4. add the squared distance between the observation and the
        # nearest unit in input space
        self.network.node[s_1]['error'] += spatial.distance.euclidean(
            observation, self.network.node[s_1]['vector'])**2
```

```
def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=False):
    ...
    # 1. iterate through the data
    sequence = 0
    for p in range(passes):
        print('Pass %d' % (p + 1))
        np.random.shuffle(self.data)
        steps = 0
        for observation in self.data:
            # observation is unit s_1
            # s_1 is the first nearest neighbor of s_2 and the second nearest unit s_2
            # s_2 is the second nearest neighbor of s_1
            nearest_units = self.find_nearest_units(observation)
            s_1 = nearest_units[0]
            s_2 = nearest_units[1]
            # 3. increment the age of all edges emanating from s_1
            for u, v, attributes in self.network.edges_iter(data=True,
                nbunch=[s_1]):
                self.network.add_edge(u, v, age=attributes['age']+1)
            # 4. add the squared distance between the observation and the
            # nearest unit in input space
            self.network.node[s_1]['error'] += spatial.distance.euclidean(
                observation, self.network.node[s_1]['vector'])**2
```

```
def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=False):
...
# 5 . move s_1 and its direct topological neighbors towards the
#      observation by the fractions
#      e_b and e_n, respectively, of the total distance
update_w_s_1 = e_b * (np.subtract(observation, self.network.node
    [s_1]['vector']))
self.network.node[s_1]['vector'] = np.add(self.network.node[s_1]
    ['vector'], update_w_s_1)
update_w_s_n = e_n * (np.subtract(observation, self.network.node
    [s_1]['vector']))
for neighbor in self.network.neighbors(s_1):
    self.network.node[neighbor]['vector'] = np.add(self.network.
        node[neighbor]['vector'], update_w_s_n)

# 6. if s_1 and s_2 are connected by an edge, set the age of
#     this edge to zero
#     if such an edge does not exist create it
self.network.add_edge(s_1, s_2, age=0)
# 7. remove edges with an age larger than a_max
#     if this results in units having no emanating edges, remove
#     them as well
self.prune_connections(a_max)
```

```
def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=False):
    ...
    # 1. iterate through the data
    sequence = 0
    for p in range(passes):
        print('Pass %d' % (p + 1))
        np.random.shuffle(self.data)
        steps = 0
        for observation in self.data:
            # observation is unit s_1
            # s_1 is the first nearest neighbor of s_2 and the second nearest unit s_2
            # s_2 is the second nearest neighbor of s_1
            nearest_units = self.find_nearest_units(observation)
            s_1 = nearest_units[0]
            s_2 = nearest_units[1]
            # 3. move s_1 and its direct topological neighbors towards the
            #      observation by the fractions
            #      e_b and e_n, respectively, of the total distance
            update_w_s_1 = e_b * (np.subtract(observation, self.network.node
                [s_1]['vector']))
            self.network.node[s_1]['vector'] = np.add(self.network.node[s_1]
                ['vector'], update_w_s_1)
            update_w_s_n = e_n * (np.subtract(observation, self.network.node
                [s_1]['vector']))
            for neighbor in self.network.neighbors(s_1):
                self.network.node[neighbor]['vector'] = np.add(self.network.
                    node[neighbor]['vector'], update_w_s_n)

            # 6. if s_1 and s_2 are connected by an edge, set the age of
            #     this edge to zero
            #     if such an edge does not exist create it
            self.network.add_edge(s_1, s_2, age=0)
            # 7. remove edges with an age larger than a_max
            #     if this results in units having no emanating edges, remove
            #     them as well
            self.prune_connections(a_max)
```



```

def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=False):
    ...

    # 8. if the number of steps so far is an integer multiple of
    # parameter l, insert a new unit
    steps += 1
    if steps % l == 0:
        if plot_evolution:
            self.plot_network('visualization/sequence/' + str(
                sequence) + '.png')
    sequence += 1
    # 8.a determine the unit q with the maximum accumulated
    # error
    q = 0
    error_max = 0
    for u in self.network.nodes_iter():
        if self.network.node[u]['error'] > error_max:
            error_max = self.network.node[u]['error']
            q = u
    # 8.b insert a new unit r halfway between q and its neighbor
    # f with the largest error variable
    f = -1
    largest_error = -1
    for u in self.network.neighbors(q):
        if self.network.node[u]['error'] > largest_error:
            largest_error = self.network.node[u]['error']
            f = u
    w_r = 0.5 * (np.add(self.network.node[q]['vector'], self.
        network.node[f]['vector']))

```

C. BUCHE - buche@enib.fr | IML

23 / 46

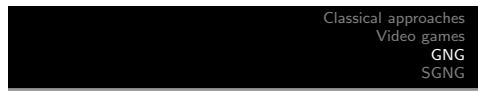
2019-09-11
IML
└ GNG

Page 23 :

```

def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=False):
    ...
    # 8. if the number of steps so far is an integer multiple of
    # parameter l, insert a new unit
    steps += 1
    if steps % l == 0:
        if plot_evolution:
            self.plot_network('visualization/sequence/' + str(
                sequence) + '.png')
    sequence += 1
    # 8.a determine the unit q with the maximum accumulated
    # error
    q = 0
    error_max = 0
    for u in self.network.nodes_iter():
        if self.network.node[u]['error'] > error_max:
            error_max = self.network.node[u]['error']
            q = u
    # 8.b insert a new unit r halfway between q and its neighbor
    # f with the largest error variable
    f = -1
    largest_error = -1
    for u in self.network.neighbors(q):
        if self.network.node[u]['error'] > largest_error:
            largest_error = self.network.node[u]['error']
            f = u
    w_r = 0.5 * (np.add(self.network.node[q]['vector'], self.
        network.node[f]['vector']))

```



```

def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=False):
    ...

    # 8.c insert edges connecting the new unit r with q and f
    # remove the original edge between q and f
    self.network.add_node(r, vector=w_r, error=0)
    self.network.add_edge(r, q, age=0)
    self.network.add_edge(r, f, age=0)
    self.network.remove_edge(q, f)
    # 8.d decrease the error variables of q and f by multiplying
    # them with a
    # initialize the error variable of r with the new value
    # of the error variable of q
    self.network.node[q]['error'] *= a
    self.network.node[f]['error'] *= a
    self.network.node[r]['error'] = self.network.node[q]['error',
    ]
    # 9. decrease all error variables by multiplying them with a
    # constant d
    error = 0
    for u in self.network.nodes_iter():
        error += self.network.node[u]['error']
    accumulated_local_error.append(error)
    network_order.append(self.network.order())
    network_size.append(self.network.size())
    total_units.append(self.units_created)
    for u in self.network.nodes_iter():
        self.network.node[u]['error'] *= d
        if self.network.degree(nbunch=[u]) == 0:
            print(u)

```

C. BUCHE - buche@enib.fr | IML

24 / 46

2019-09-11
IML
└ GNG

Page 24 :

```

def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=False):
    ...
    # 8.c insert edges connecting the new unit r with q and f
    # remove the original edge between q and f
    self.network.add_node(r, vector=w_r, error=0)
    self.network.add_edge(r, q, age=0)
    self.network.add_edge(r, f, age=0)
    self.network.remove_edge(q, f)
    # 8.d decrease the error variables of q and f by multiplying
    # them with a
    # initialize the error variable of r with the new value
    # of the error variable of q
    self.network.node[q]['error'] *= a
    self.network.node[f]['error'] *= a
    self.network.node[r]['error'] = self.network.node[q]['error',
    ]
    # 9. decrease all error variables by multiplying them with a
    # constant d
    error = 0
    for u in self.network.nodes_iter():
        error += self.network.node[u]['error']
    accumulated_local_error.append(error)
    network_order.append(self.network.order())
    network_size.append(self.network.size())
    total_units.append(self.units_created)
    for u in self.network.nodes_iter():
        self.network.node[u]['error'] *= d
        if self.network.degree(nbunch=[u]) == 0:
            print(u)

```

```
def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=False):
...
    plt.clf()
    plt.title('Accumulated_local_error')
    plt.xlabel('iterations')
    plt.plot(range(len(accumulated_local_error)), accumulated_local_error)
    plt.savefig('visualization/accumulated_local_error.png')
    plt.clf()
    plt.title('Global_error')
    plt.xlabel('passes')
    plt.plot(range(len(global_error)), global_error)
    plt.savefig('visualization/global_error.png')
    plt.clf()
    plt.title('Neural_network_properties')
    plt.plot(range(len(network_order)), network_order, label='Network_order',
            )
    plt.plot(range(len(network_size)), network_size, label='Network_size')
    plt.legend()
    plt.savefig('visualization/network_properties.png')
```

```
def fit_network(self, e_b, e_n, a_max, l, a, d, passes=1, plot_evolution=False):
...
    plt.clf()
    plt.title('Accumulated_local_error')
    plt.xlabel('iterations')
    plt.plot(range(len(accumulated_local_error)), accumulated_local_error)
    plt.savefig('visualization/accumulated_local_error.png')
    plt.clf()
    plt.title('Global_error')
    plt.xlabel('passes')
    plt.plot(range(len(global_error)), global_error)
    plt.savefig('visualization/global_error.png')
    plt.clf()
    plt.title('Neural_network_properties')
    plt.plot(range(len(network_order)), network_order, label='Network_order',
            )
    plt.plot(range(len(network_size)), network_size, label='Network_size')
    plt.legend()
    plt.savefig('visualization/network_properties.png')
```

```
def plot_network(self, file_path):
    plt.clf()
    plt.scatter(self.data[:, 0], self.data[:, 1])
    node_pos = {}
    for u in self.network.nodes_iter():
        vector = self.network.node[u]['vector']
        node_pos[u] = (vector[0], vector[1])
    nx.draw(self.network, pos=node_pos)
    plt.draw()
    plt.savefig(file_path)

def number_of_clusters(self):
    return nx.number_connected_components(self.network)
```

```
def plot_network(self, file_path):
    plt.clf()
    plt.scatter(self.data[:, 0], self.data[:, 1])
    node_pos = {}
    for u in self.network.nodes_iter():
        vector = self.network.node[u]['vector']
        node_pos[u] = (vector[0], vector[1])
    nx.draw(self.network, pos=node_pos)
    plt.draw()
    plt.savefig(file_path)

def number_of_clusters(self):
    return nx.number_connected_components(self.network)
```

```
def cluster_data(self):
    unit_to_cluster = np.zeros(self.units_created)
    cluster = 0
    for c in nx.connected_components(self.network):
        for unit in c:
            unit_to_cluster[unit] = cluster
        cluster += 1
    clustered_data = []
    for observation in self.data:
        nearest_units = self.find_nearest_units(observation)
        s = nearest_units[0]
        clustered_data.append((observation, unit_to_cluster[s]))
    return clustered_data

def reduce_dimension(self, clustered_data):
    transformed_clustered_data = []
    svd = decomposition.PCA(n_components=2)
    transformed_observations = svd.fit_transform(self.data)
    for i in range(len(clustered_data)):
        transformed_clustered_data.append((transformed_observations[i],
                                            clustered_data[i][1]))
    return transformed_clustered_data
```

```
def cluster_data(self):
    unit_to_cluster = np.zeros(self.units_created)
    cluster = 0
    for c in nx.connected_components(self.network):
        for unit in c:
            unit_to_cluster[unit] = cluster
        cluster += 1
    clustered_data = []
    for observation in self.data:
        nearest_units = self.find_nearest_units(observation)
        s = nearest_units[0]
        clustered_data.append((observation, unit_to_cluster[s]))
    return clustered_data

def reduce_dimension(self, clustered_data):
    transformed_clustered_data = []
    svd = decomposition.PCA(n_components=2)
    transformed_observations = svd.fit_transform(self.data)
    for i in range(len(clustered_data)):
        transformed_clustered_data.append((transformed_observations[i],
                                            clustered_data[i][1]))
    return transformed_clustered_data
```

```
def plot_clusters(self, clustered_data):
    number_of_clusters = nx.number_connected_components(self.network)
    plt.clf()
    plt.title('Cluster affectation')
    color = ['r', 'b', 'g', 'k', 'm', 'r', 'b', 'g', 'k', 'm']
    for i in range(number_of_clusters):
        observations = [observation for observation, s in clustered_data if
                        s == i]
        if len(observations) > 0:
            observations = np.array(observations)
            plt.scatter(observations[:, 0], observations[:, 1], color=color[i],
                        label='cluster #' + str(i))
    plt.legend()
    plt.savefig('visualization/clusters.png')

def compute_global_error(self):
    global_error = 0
    for observation in self.data:
        nearest_units = self.find_nearest_units(observation)
        s_1 = nearest_units[0]
        global_error += spatial.distance.euclidean(observation, self.network
                                                    .node[s_1]['vector'])**2
    return global_error
```

```
def plot_clusters(self, clustered_data):
    number_of_clusters = nx.number_connected_components(self.network)
    plt.title('Cluster affectation')
    color = ['r', 'b', 'g', 'k', 'm', 'r', 'b', 'g', 'k', 'm']
    for i in range(number_of_clusters):
        observations = [observation for observation, s in clustered_data if
                        s == i]
        if len(observations) > 0:
            observations = np.array(observations)
            plt.scatter(observations[:, 0], observations[:, 1], color=color[i],
                        label='cluster #' + str(i))
    plt.legend()
    plt.savefig('visualization/clusters.png')

def compute_global_error(self):
    global_error = 0
    for observation in self.data:
        nearest_units = self.find_nearest_units(observation)
        s_1 = nearest_units[0]
        global_error += spatial.distance.euclidean(observation, self.network
                                                    .node[s_1]['vector'])**2
    return global_error
```

example

```
from gng import GrowingNeuralGas
from sklearn import datasets
from sklearn.preprocessing import StandardScaler

print('Generating data...')
data = datasets.make_moons(n_samples=2000, noise=.05)
data = StandardScaler().fit_transform(data[0])
print('Done.')
print('Fitting neural network...')
gng = GrowingNeuralGas(data)
gng.fit_network(e_b=0.1, e_n=0.006, a_max=10, l=200, a=0.5, d=0.995, passes=8,
                 plot_evolution=True)
print('Found %d clusters.' % gng.number_of_clusters())
gng.plot_clusters(gng.cluster_data())
```

```
Preparing data...
Done.
Fitting neural network...
    Pass #1
    Pass #2
    Pass #3
    Pass #4
    Pass #5
    Pass #6
    Pass #7
    Pass #8
Found 2 clusters.
```

example

Page 29

Demo : generateDataGNG.py
Demo : exampleGNG.py (use gng.py)

exam

```

from pg import GreenbergModel
from pg import Data
from sklearn.preprocessing import StandardScaler

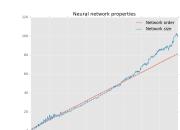
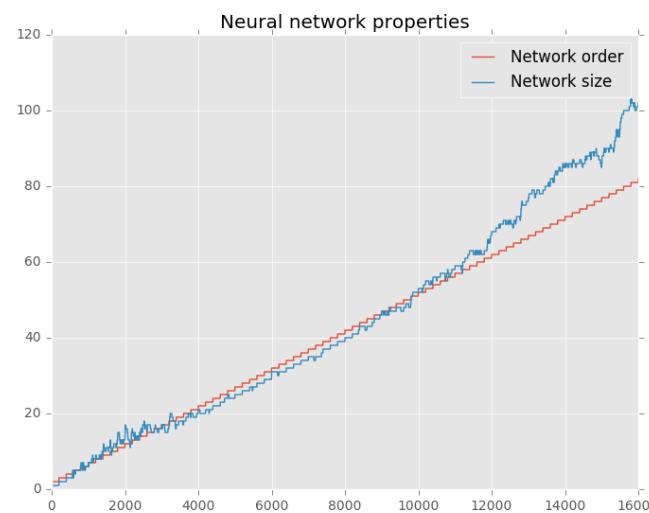
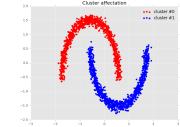
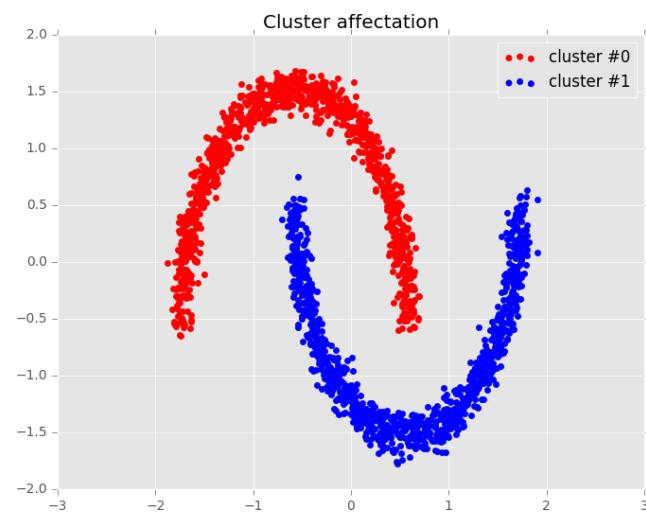
print("Generating data...")
data = Data()
data.read_csv("data/greenberg0000.csv", maxsize=100)
print("Data loaded (%d rows, %d features)" % (len(data), len(data[0])))
print("Data shape: %s" % str(data.shape))
print("Scaling numerical features...")
gng = GreenbergModel()
gng.fit(data)
data = gng.transform(data)
print("Data scaled (%d rows, %d features)" % (len(data), len(data[0])))
print("Plotting first two dimensions...")
gng.plot_2d(data)
print("Plotting first dimension vs. L1 norm of difference...")
gng.plot_1d(data)
print("Plotting number of clusters vs. L1 norm of difference...")
gng.plot_nclusters(data)

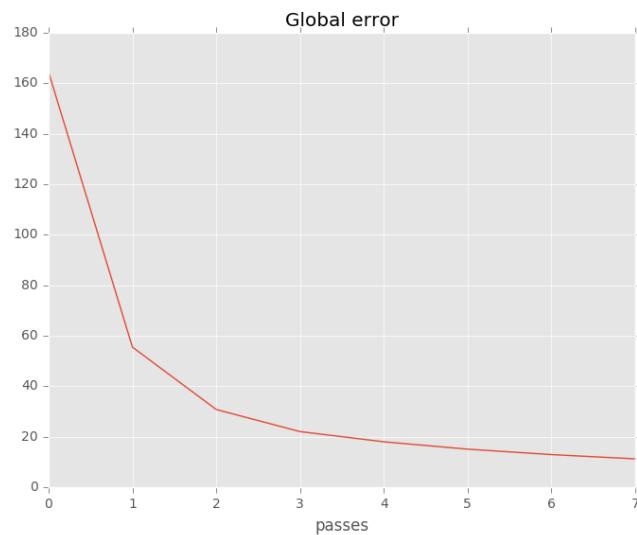
```

Page 29

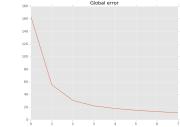
Demo : generateDataGNG.py
Demo : exampleGNG.py (use gng.py)

Preparing
 Bone.
 Fixing same.
 Pass #1
 Pass #2
 Pass #3
 Pass #4
 Pass #5
 Pass #6
 Pass #7
 Pass #8
 Pass #9
 Pass #10





Page 33 :



Problems

- ▷ Fails to handle temporal series
- ▷ Grow up over the time (constant iteration insertion policy)

Page 34 :

Problems

- Fails to handle temporal series
- Grow up over the time (constant iteration insertion policy)

1 Classical approaches

- Mesh
- Graph

2 Video games

- Quake 3 Arena bots
- Counter-Strike bots
- Unreal Tournament's series bots
- Valve bots

3 GNG

4 SGNG

Page 35 :

SGNG : Pro

- ▷ Grow up over the time : a node should be inserted when the error of a node is superior to a parameter \overline{Err} (add nodes only in the area just visited and add nodes only when the network fails to fit the data suitably).

Page 36 :

SGNG - algorithm

```

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

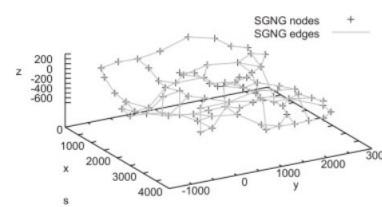
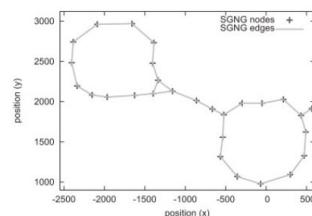
```

C. BUCHE - buche@enib.fr

IML

37 / 46

SGNG : Application to video games



<https://www.youtube.com/watch?v=HSKkr4CQYr8>

2019-09-11

└ SGNG - algorithm

Page 37 :

```

SGNG - algorithm
nodes = []
edges = []
edgeAge = 0
err = 0.001
age = 100
maxAge = 1000
s = 0.001
s0 = 0.001
s1 = 0.001
s2 = 0.001
s3 = 0.001
s4 = 0.001
s5 = 0.001
s6 = 0.001
s7 = 0.001
s8 = 0.001
s9 = 0.001
s10 = 0.001
s11 = 0.001
s12 = 0.001
s13 = 0.001
s14 = 0.001
s15 = 0.001
s16 = 0.001
s17 = 0.001
s18 = 0.001
s19 = 0.001
s20 = 0.001
s21 = 0.001
s22 = 0.001
s23 = 0.001
s24 = 0.001
s25 = 0.001
s26 = 0.001
s27 = 0.001
s28 = 0.001
s29 = 0.001
s30 = 0.001
s31 = 0.001
s32 = 0.001
s33 = 0.001
s34 = 0.001
s35 = 0.001
s36 = 0.001
s37 = 0.001
s38 = 0.001
s39 = 0.001
s40 = 0.001
s41 = 0.001
s42 = 0.001
s43 = 0.001
s44 = 0.001
s45 = 0.001
s46 = 0.001
s47 = 0.001
s48 = 0.001
s49 = 0.001
s50 = 0.001
s51 = 0.001
s52 = 0.001
s53 = 0.001
s54 = 0.001
s55 = 0.001
s56 = 0.001
s57 = 0.001
s58 = 0.001
s59 = 0.001
s60 = 0.001
s61 = 0.001
s62 = 0.001
s63 = 0.001
s64 = 0.001
s65 = 0.001
s66 = 0.001
s67 = 0.001
s68 = 0.001
s69 = 0.001
s70 = 0.001
s71 = 0.001
s72 = 0.001
s73 = 0.001
s74 = 0.001
s75 = 0.001
s76 = 0.001
s77 = 0.001
s78 = 0.001
s79 = 0.001
s80 = 0.001
s81 = 0.001
s82 = 0.001
s83 = 0.001
s84 = 0.001
s85 = 0.001
s86 = 0.001
s87 = 0.001
s88 = 0.001
s89 = 0.001
s90 = 0.001
s91 = 0.001
s92 = 0.001
s93 = 0.001
s94 = 0.001
s95 = 0.001
s96 = 0.001
s97 = 0.001
s98 = 0.001
s99 = 0.001
s100 = 0.001
s101 = 0.001
s102 = 0.001
s103 = 0.001
s104 = 0.001
s105 = 0.001
s106 = 0.001
s107 = 0.001
s108 = 0.001
s109 = 0.001
s110 = 0.001
s111 = 0.001
s112 = 0.001
s113 = 0.001
s114 = 0.001
s115 = 0.001
s116 = 0.001
s117 = 0.001
s118 = 0.001
s119 = 0.001
s120 = 0.001
s121 = 0.001
s122 = 0.001
s123 = 0.001
s124 = 0.001
s125 = 0.001
s126 = 0.001
s127 = 0.001
s128 = 0.001
s129 = 0.001
s130 = 0.001
s131 = 0.001
s132 = 0.001
s133 = 0.001
s134 = 0.001
s135 = 0.001
s136 = 0.001
s137 = 0.001
s138 = 0.001
s139 = 0.001
s140 = 0.001
s141 = 0.001
s142 = 0.001
s143 = 0.001
s144 = 0.001
s145 = 0.001
s146 = 0.001
s147 = 0.001
s148 = 0.001
s149 = 0.001
s150 = 0.001
s151 = 0.001
s152 = 0.001
s153 = 0.001
s154 = 0.001
s155 = 0.001
s156 = 0.001
s157 = 0.001
s158 = 0.001
s159 = 0.001
s160 = 0.001
s161 = 0.001
s162 = 0.001
s163 = 0.001
s164 = 0.001
s165 = 0.001
s166 = 0.001
s167 = 0.001
s168 = 0.001
s169 = 0.001
s170 = 0.001
s171 = 0.001
s172 = 0.001
s173 = 0.001
s174 = 0.001
s175 = 0.001
s176 = 0.001
s177 = 0.001
s178 = 0.001
s179 = 0.001
s180 = 0.001
s181 = 0.001
s182 = 0.001
s183 = 0.001
s184 = 0.001
s185 = 0.001
s186 = 0.001
s187 = 0.001
s188 = 0.001
s189 = 0.001
s190 = 0.001
s191 = 0.001
s192 = 0.001
s193 = 0.001
s194 = 0.001
s195 = 0.001
s196 = 0.001
s197 = 0.001
s198 = 0.001
s199 = 0.001
s200 = 0.001
s201 = 0.001
s202 = 0.001
s203 = 0.001
s204 = 0.001
s205 = 0.001
s206 = 0.001
s207 = 0.001
s208 = 0.001
s209 = 0.001
s210 = 0.001
s211 = 0.001
s212 = 0.001
s213 = 0.001
s214 = 0.001
s215 = 0.001
s216 = 0.001
s217 = 0.001
s218 = 0.001
s219 = 0.001
s220 = 0.001
s221 = 0.001
s222 = 0.001
s223 = 0.001
s224 = 0.001
s225 = 0.001
s226 = 0.001
s227 = 0.001
s228 = 0.001
s229 = 0.001
s230 = 0.001
s231 = 0.001
s232 = 0.001
s233 = 0.001
s234 = 0.001
s235 = 0.001
s236 = 0.001
s237 = 0.001
s238 = 0.001
s239 = 0.001
s240 = 0.001
s241 = 0.001
s242 = 0.001
s243 = 0.001
s244 = 0.001
s245 = 0.001
s246 = 0.001
s247 = 0.001
s248 = 0.001
s249 = 0.001
s250 = 0.001
s251 = 0.001
s252 = 0.001
s253 = 0.001
s254 = 0.001
s255 = 0.001
s256 = 0.001
s257 = 0.001
s258 = 0.001
s259 = 0.001
s260 = 0.001
s261 = 0.001
s262 = 0.001
s263 = 0.001
s264 = 0.001
s265 = 0.001
s266 = 0.001
s267 = 0.001
s268 = 0.001
s269 = 0.001
s270 = 0.001
s271 = 0.001
s272 = 0.001
s273 = 0.001
s274 = 0.001
s275 = 0.001
s276 = 0.001
s277 = 0.001
s278 = 0.001
s279 = 0.001
s280 = 0.001
s281 = 0.001
s282 = 0.001
s283 = 0.001
s284 = 0.001
s285 = 0.001
s286 = 0.001
s287 = 0.001
s288 = 0.001
s289 = 0.001
s290 = 0.001
s291 = 0.001
s292 = 0.001
s293 = 0.001
s294 = 0.001
s295 = 0.001
s296 = 0.001
s297 = 0.001
s298 = 0.001
s299 = 0.001
s300 = 0.001
s301 = 0.001
s302 = 0.001
s303 = 0.001
s304 = 0.001
s305 = 0.001
s306 = 0.001
s307 = 0.001
s308 = 0.001
s309 = 0.001
s310 = 0.001
s311 = 0.001
s312 = 0.001
s313 = 0.001
s314 = 0.001
s315 = 0.001
s316 = 0.001
s317 = 0.001
s318 = 0.001
s319 = 0.001
s320 = 0.001
s321 = 0.001
s322 = 0.001
s323 = 0.001
s324 = 0.001
s325 = 0.001
s326 = 0.001
s327 = 0.001
s328 = 0.001
s329 = 0.001
s330 = 0.001
s331 = 0.001
s332 = 0.001
s333 = 0.001
s334 = 0.001
s335 = 0.001
s336 = 0.001
s337 = 0.001
s338 = 0.001
s339 = 0.001
s340 = 0.001
s341 = 0.001
s342 = 0.001
s343 = 0.001
s344 = 0.001
s345 = 0.001
s346 = 0.001
s347 = 0.001
s348 = 0.001
s349 = 0.001
s350 = 0.001
s351 = 0.001
s352 = 0.001
s353 = 0.001
s354 = 0.001
s355 = 0.001
s356 = 0.001
s357 = 0.001
s358 = 0.001
s359 = 0.001
s360 = 0.001
s361 = 0.001
s362 = 0.001
s363 = 0.001
s364 = 0.001
s365 = 0.001
s366 = 0.001
s367 = 0.001
s368 = 0.001
s369 = 0.001
s370 = 0.001
s371 = 0.001
s372 = 0.001
s373 = 0.001
s374 = 0.001
s375 = 0.001
s376 = 0.001
s377 = 0.001
s378 = 0.001
s379 = 0.001
s380 = 0.001
s381 = 0.001
s382 = 0.001
s383 = 0.001
s384 = 0.001
s385 = 0.001
s386 = 0.001
s387 = 0.001
s388 = 0.001
s389 = 0.001
s390 = 0.001
s391 = 0.001
s392 = 0.001
s393 = 0.001
s394 = 0.001
s395 = 0.001
s396 = 0.001
s397 = 0.001
s398 = 0.001
s399 = 0.001
s400 = 0.001
s401 = 0.001
s402 = 0.001
s403 = 0.001
s404 = 0.001
s405 = 0.001
s406 = 0.001
s407 = 0.001
s408 = 0.001
s409 = 0.001
s410 = 0.001
s411 = 0.001
s412 = 0.001
s413 = 0.001
s414 = 0.001
s415 = 0.001
s416 = 0.001
s417 = 0.001
s418 = 0.001
s419 = 0.001
s420 = 0.001
s421 = 0.001
s422 = 0.001
s423 = 0.001
s424 = 0.001
s425 = 0.001
s426 = 0.001
s427 = 0.001
s428 = 0.001
s429 = 0.001
s430 = 0.001
s431 = 0.001
s432 = 0.001
s433 = 0.001
s434 = 0.001
s435 = 0.001
s436 = 0.001
s437 = 0.001
s438 = 0.001
s439 = 0.001
s440 = 0.001
s441 = 0.001
s442 = 0.001
s443 = 0.001
s444 = 0.001
s445 = 0.001
s446 = 0.001
s447 = 0.001
s448 = 0.001
s449 = 0.001
s450 = 0.001
s451 = 0.001
s452 = 0.001
s453 = 0.001
s454 = 0.001
s455 = 0.001
s456 = 0.001
s457 = 0.001
s458 = 0.001
s459 = 0.001
s460 = 0.001
s461 = 0.001
s462 = 0.001
s463 = 0.001
s464 = 0.001
s465 = 0.001
s466 = 0.001
s467 = 0.001
s468 = 0.001
s469 = 0.001
s470 = 0.001
s471 = 0.001
s472 = 0.001
s473 = 0.001
s474 = 0.001
s475 = 0.001
s476 = 0.001
s477 = 0.001
s478 = 0.001
s479 = 0.001
s480 = 0.001
s481 = 0.001
s482 = 0.001
s483 = 0.001
s484 = 0.001
s485 = 0.001
s486 = 0.001
s487 = 0.001
s488 = 0.001
s489 = 0.001
s490 = 0.001
s491 = 0.001
s492 = 0.001
s493 = 0.001
s494 = 0.001
s495 = 0.001
s496 = 0.001
s497 = 0.001
s498 = 0.001
s499 = 0.001
s500 = 0.001
s501 = 0.001
s502 = 0.001
s503 = 0.001
s504 = 0.001
s505 = 0.001
s506 = 0.001
s507 = 0.001
s508 = 0.001
s509 = 0.001
s510 = 0.001
s511 = 0.001
s512 = 0.001
s513 = 0.001
s514 = 0.001
s515 = 0.001
s516 = 0.001
s517 = 0.001
s518 = 0.001
s519 = 0.001
s520 = 0.001
s521 = 0.001
s522 = 0.001
s523 = 0.001
s524 = 0.001
s525 = 0.001
s526 = 0.001
s527 = 0.001
s528 = 0.001
s529 = 0.001
s530 = 0.001
s531 = 0.001
s532 = 0.001
s533 = 0.001
s534 = 0.001
s535 = 0.001
s536 = 0.001
s537 = 0.001
s538 = 0.001
s539 = 0.001
s540 = 0.001
s541 = 0.001
s542 = 0.001
s543 = 0.001
s544 = 0.001
s545 = 0.001
s546 = 0.001
s547 = 0.001
s548 = 0.001
s549 = 0.001
s550 = 0.001
s551 = 0.001
s552 = 0.001
s553 = 0.001
s554 = 0.001
s555 = 0.001
s556 = 0.001
s557 = 0.001
s558 = 0.001
s559 = 0.001
s560 = 0.001
s561 = 0.001
s562 = 0.001
s563 = 0.001
s564 = 0.001
s565 = 0.001
s566 = 0.001
s567 = 0.001
s568 = 0.001
s569 = 0.001
s570 = 0.001
s571 = 0.001
s572 = 0.001
s573 = 0.001
s574 = 0.001
s575 = 0.001
s576 = 0.001
s577 = 0.001
s578 = 0.001
s579 = 0.001
s580 = 0.001
s581 = 0.001
s582 = 0.001
s583 = 0.001
s584 = 0.001
s585 = 0.001
s586 = 0.001
s587 = 0.001
s588 = 0.001
s589 = 0.001
s590 = 0.001
s591 = 0.001
s592 = 0.001
s593 = 0.001
s594 = 0.001
s595 = 0.001
s596 = 0.001
s597 = 0.001
s598 = 0.001
s599 = 0.001
s600 = 0.001
s601 = 0.001
s602 = 0.001
s603 = 0.001
s604 = 0.001
s605 = 0.001
s606 = 0.001
s607 = 0.001
s608 = 0.001
s609 = 0.001
s610 = 0.001
s611 = 0.001
s612 = 0.001
s613 = 0.001
s614 = 0.001
s615 = 0.001
s616 = 0.001
s617 = 0.001
s618 = 0.001
s619 = 0.001
s620 = 0.001
s621 = 0.001
s622 = 0.001
s623 = 0.001
s624 = 0.001
s625 = 0.001
s626 = 0.001
s627 = 0.001
s628 = 0.001
s629 = 0.001
s630 = 0.001
s631 = 0.001
s632 = 0.001
s633 = 0.001
s634 = 0.001
s635 = 0.001
s636 = 0.001
s637 = 0.001
s638 = 0.001
s639 = 0.001
s640 = 0.001
s641 = 0.001
s642 = 0.001
s643 = 0.001
s644 = 0.001
s645 = 0.001
s646 = 0.001
s647 = 0.001
s648 = 0.001
s649 = 0.001
s650 = 0.001
s651 = 0.001
s652 = 0.001
s653 = 0.001
s654 = 0.001
s655 = 0.001
s656 = 0.001
s657 = 0.001
s658 = 0.001
s659 = 0.001
s660 = 0.001
s661 = 0.001
s662 = 0.001
s663 = 0.001
s664 = 0.001
s665 = 0.001
s666 = 0.001
s667 = 0.001
s668 = 0.001
s669 = 0.001
s670 = 0.001
s671 = 0.001
s672 = 0.001
s673 = 0.001
s674 = 0.001
s675 = 0.001
s676 = 0.001
s677 = 0.001
s678 = 0.001
s679 = 0.001
s680 = 0.001
s681 = 0.001
s682 = 0.001
s683 = 0.001
s684 = 0.001
s685 = 0.001
s686 = 0.001
s687 = 0.001
s688 = 0.001
s689 = 0.001
s690 = 0.001
s691 = 0.001
s692 = 0.001
s693 = 0.001
s694 = 0.001
s695 = 0.001
s696 = 0.001
s697 = 0.001
s698 = 0.001
s699 = 0.001
s700 = 0.001
s701 = 0.001
s702 = 0.001
s703 = 0.001
s704 = 0.001
s705 = 0.001
s706 = 0.001
s707 = 0.001
s708 = 0.001
s709 = 0.001
s710 = 0.001
s711 = 0.001
s712 = 0.001
s713 = 0.001
s714 = 0.001
s715 = 0.001
s716 = 0.001
s717 = 0.001
s718 = 0.001
s719 = 0.001
s720 = 0.001
s721 = 0.001
s722 = 0.001
s723 = 0.001
s724 = 0.001
s725 = 0.001
s726 = 0.001
s727 = 0.001
s728 = 0.001
s729 = 0.001
s730 = 0.001
s731 = 0.001
s732 = 0.001
s733 = 0.001
s734 = 0.001
s735 = 0.001
s736 = 0.001
s737 = 0.001
s738 = 0.001
s739 = 0.001
s740 = 0.001
s741 = 0.001
s742 = 0.001
s743 = 0.001
s744 = 0.001
s745 = 0.001
s746 = 0.001
s747 = 0.001
s748 = 0.001
s749 = 0.001
s750 = 0.001
s751 = 0.001
s752 = 0.001
s753 = 0.001
s754 = 0.001
s755 = 0.001
s756 = 0.001
s757 = 0.001
s758 = 0.001
s759 = 0.001
s760 = 0.001
s761 = 0.001
s762 = 0.001
s763 = 0.001
s764 = 0.001
s765 = 0.001
s766 = 0.001
s767 = 0.001
s768 = 0.001
s769 = 0.001
s770 = 0.001
s771 = 0.001
s772 = 0.001
s773 = 0.001
s774 = 0.001
s775 = 0.001
s776 = 0.001
s777 = 0.001
s778 = 0.001
s779 = 0.001
s780 = 0.001
s781 = 0.001
s782 = 0.001
s783 = 0.001
s784 = 0.001
s785 = 0.001
s786 = 0.001
s787 = 0.001
s788 = 0.001
s789 = 0.001
s790 = 0.001
s791 = 0.001
s792 = 0.001
s793 = 0.001
s794 = 0.001
s795 = 0.001
s796 = 0.001
s797 = 0.001
s798 = 0.001
s799 = 0.001
s800 = 0.001
s801 = 0.001
s802 = 0.001
s803 = 0.001
s804 = 0.001
s805 = 0.001
s806 = 0.001
s807 = 0.001
s808 = 0.001
s809 = 0.001
s810 = 0.001
s811 = 0.001
s812 = 0.001
s813 = 0.001
s814 = 0.001
s815 = 0.001
s816 = 0.001
s817 = 0.001
s818 = 0.001
s819 = 0.001
s820 = 0.001
s821 = 0.001
s822 = 0.001
s823 = 0.001
s824 = 0.001
s825 = 0.001
s826 = 0.001
s827 = 0.001
s828 = 0.001
s829 = 0.001
s830 = 0.001
s831 = 0.001
s832 = 0.001
s833 = 0.001
s834 = 0.001
s835 = 0.001
s836 = 0.001
s837 = 0.001
s838 = 0.001
s839 = 0.001
s840 = 0
```

SGNG : Application to video games

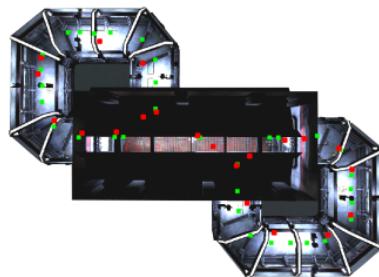


FIGURE – Comparison of nodes learned by the SGNG (in red) with the navigation points placed manually by the game developers (in green). The environment viewed from above is visible in the background.

└ SGNG : Application to video games



SGNG : Evaluation

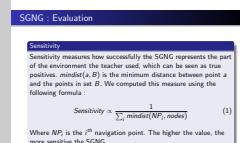
Sensitivity

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 :

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

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

└ SGNG : Evaluation



SGNG : Evaluation

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. We computed this measure using the following formula :

$$\text{Specificity} \propto \frac{\text{Number of Nodes}}{\sum_i \text{mindist}(\text{Node}_i, \text{NPs})} \quad (2)$$

The higher the value, the more specific the SGNG.

Specificity measures how much the SGNG did not represent a part of the environment the teacher did not use and can be seen as true negatives. We computed this measure using the following formula :

$$\text{Specificity} = \frac{\text{Number of Nodes}}{\sum_i \text{mindist}(\text{Node}_i, \text{NPs})} \quad (2)$$

The higher the value, the more specific the SGNG.

Page 41 :

SGNG : Evaluation

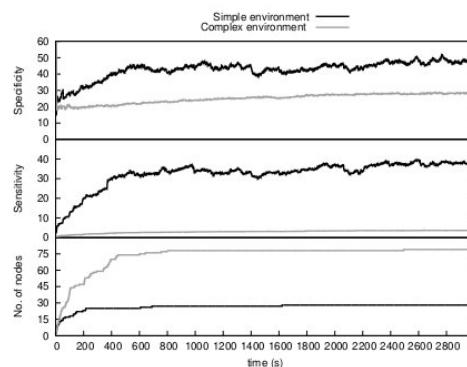
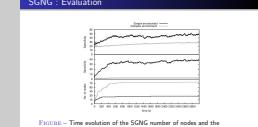


FIGURE – Time evolution of the SGNG number of nodes and the cumulated distance between the SGNG nodes and the navigation points.



Page 42 :

SGNG : Evaluation

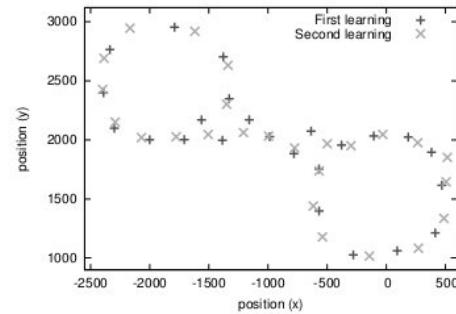
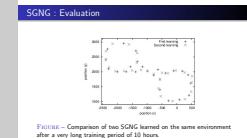
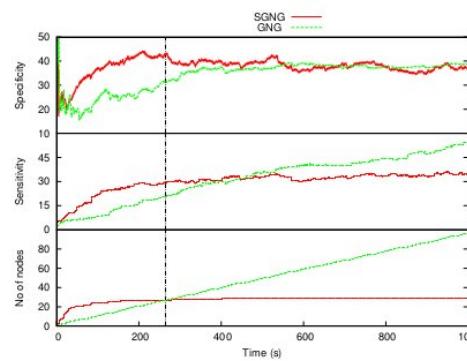


FIGURE – Comparison of two SGNG learned on the same environment after a very long training period of 10 hours.



Page 43 :

SGNG : Evaluation



Page 44 :

SGNG : Référence

F. Tence, L. Gaubert, J. Soler, P. De Loor, and C. Buche.
Stable Growing Neural Gas : a Topology Learning Algorithm based on Player
Tracking in Video Games.
Applied Soft Computing (ASC),
13(10) :4174-4184, 2013.

https://www.enib.fr/~buche/article/ASC_13.pdf

Navigation IML

Cédric Buche

ENIB

11 septembre 2019