UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DESIGN AND IMPLEMENTATION OF AN AGENT-BASED SOCIAL MODEL OF TERRORIST NETWORK BASED ON SOCIAL NETWORK ANALYSIS TECHNIQUES

DAVID GARCÍA MARTÍN

Junio 2017

TRABAJO FIN DE GRADO

Título:	Diseño e Implementación de un Modelo Social de Redes Terroristas basado en Agentes y Técnicas de Análisis de Redes Sociales.
Título (inglés):	Design and Implementation of an Agent-based Social Model of Terrorist Networks based on Social Network Analysis Techniques
Autor:	David García Martín
Tutor:	Carlos A. Iglesias Fernández
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:

Vocal:

Secretario:

Suplente:

FECHA DE LECTURA:

CALIFICACIÓN:

Resumen

Esta memoria es el resultado de un proyecto cuyos objetivos son el diseño e implementación de una Red Terrorista, la visualización y análisis de la misma aplicando distintas técnicas de análisis. Este modelo puede ser usado por las fuerzas y cuerpos de seguridad del estado para prevenir futuros atentados y anticiparse a sus movimientos.

Para llevar a cabo este modelo, nos hemos basado en el estudio del comportamiento en Redes Sociales y los distintos tipos de redes que podemos encontrarnos, poniendo especial atención en el cátalogo de simuladores y funcionalidades del software de análisis de redes NetworkX. Tanto para la realización del modelo, como de la simulación y la visualización de la Red, hemos usado el simulador social basado en agentes Soil, gracias a sus ventajas frente a otros simuladores sociales.

Con el fin de realizar el diseño y desarrollo de los modelos de comportamiento terrorista se han aplicado distintas técnicas usando el lenguaje de programación Python. Para alcanzar nuestros objetivos, se ha evaluando y simulando información útil como puede ser información propagada en la red, tipos de agentes y otras características importantes. Ayudándonos de distintas topologías, métodos de manipulación estructural de redes y diferentes técnicas de estudio de centralidad de nodos en redes.

Por último se visualizará el grafo generado con ayuda de la herramienta de visualización de redes Gephi, donde se podrán identificar los distintos tipos de agentes, sus estados y además observar cómo se expande la red de forma dinámica. Todo ello facilitará el análisis, la comprensión del modelo y el estudio de la variación de sus parámetros. Adicionalmente, se generan gráficas de línea para observar la evolución de la red a lo largo del tiempo.

Palabras clave: Modelo, Analisis, Python, Implementación, Gephi, NetworkX, Redes Sociales, Soil.

Abstract

This thesis is the result of a project whose objectives have been to design and implement a Terrorist Network model and its' visualisation. This model can be used by security forces with the aim of preventing other attacks and being one step ahead from terrorist intentions and movements.

As we have mentioned, the project will be focused on the design and implementation of a Terrorist Network model.

To do so, we have based our model in the study of Social Networks behaviour and network types, paying attention to disparate simulators and functions of network analysis software Networkx. Soil, a social simulator based in agents, has been used to help us in this project due to its advantages as for example: the ability to provide capacities for modelling, simulating and visualizing Social Networks.

Different techniques have been applied to design, develop and study terrorist behaviour models on Python, evaluating and simulating useful information such as information spread, agent types and other characteristics of terrorist networks using a variety of network topologies and methods of manipulating networks' structure. In addition, we have used some functions to study and manipulate this structure, for example nodes centrality.

To continue, a Terrorist Network will be rendered being able to identify different agents and visualising network expansion. This interactive visualisation will be generated using Gephi network visualization tool; making easy to understand and analyse it. Besides, a line-graph will be also generated to easily understand our model.

As a result, this project will allow an extensive analysis of Social Networks structure and conduct patterns, pointing out how organizations expand affecting all type of agents. With all these informations we are going to be able to understand terrorist behaviour and how these organizations expand and work in a new easily way.

Keywords: Model, Analysis, Python, Implementation, Gephi, NetworkX, Social Network, Soil.

Agradecimientos

Me gustaría en estas líneas agradecer a la gente que me ha ayudado y apoyado no solo en este trabajo de fin de grado, también en toda la carrera.

A mi familia, por su apoyo, paciencia y comprension durante todos estos años.

A Mica y Sonia, por todos vuestros ánimos en todo momento y vuestra insistencia para mantenerme al pie del cañón.

A Diego e Ismael, por todos nuestros momentos de desconexión que siempre me han alentado a continuar con una gran sonrisa.

También a todas las personas que he conocido en la Universidad durante estos años que en todo momento me han prestado su ayuda y han contribuido a que el día a día en esta etapa fuera mejor.

A todos los compañeros del Grupo de Sistemas Inteligentes por ayudar en todo lo posible y en particular a Carlos, que además de llevar el seguimiento de mi trabajo, siempre ha estado dispuesto a concertar reuniones para resolver cualquier duda y revisar avances.

Contents

List of Figures

Listings

CHAPTER 1

Introduction

1.1 Motivation

Nowadays with the recent terrorist attacks, prevention and surveillance have become increasingly important consideration within society and security forces.

Radicalism reflects the political and ideological dimension of threat [18]. While terrorism is a serious and often lethal challenge, radicalism has several social, economic and behavioural components. When relative inequality, collective frustration and deprivation stands out in a society, terrorist easily exploits this social radicalised habit. Therefore, radicalism is sustained by multiples causes; including broad grievances pushing individuals toward a radical ideology or more specific, factors that attract them. Armed groups build stronger bonds with population by investing in social services, they build schools or even hospitals, also offering something that the state is not providing: safety and security, filling the gap left by the government. [7]

Anti-radicalism refers to the activity of developing defensive measures to lessen the vulnerability of populations and infrastructure to the growth of radicalism. There is a need to comprehend terrorist modus operandi, how they work, expand or recruit new members. So that, understanding non-state armed groups like ISIS is key to solving most conflicts,

but war has changed. It used to be a contest between states, but now it is a conflict between states and non-states actors.

As we have said before, there is a need to know what makes these organizations change. Usually, the spotlight is put on how terrorists fight, why they fight, but no one looks at what they are doing when they are not fighting. Terrorist groups do more than just shoot, or suicide attacks, they are multi-task. [3]

In the past few years, we can see that the way how people communicate have changed, becoming more relevant Social Networks, where everyone can exchange messages, images and videos. Terrorist organizations also move forward by setting up radio stations, TV channels, Internet websites, etc. An example is ISIS magazine, printed in English and published with the objective of recruit new members. These activities allow them to increase their strength, their funds and better recruit new people.

Thus, there is a need for developing new ways of analysing terrorist behaviour in order to anticipate and comprehend their "modus operandi" with the main idea of neutralize future terrorist attacks.

With regard to all these considerations, the project will be based on Social Network Analysis Techniques [10]. Terrorist organizations continuously expand their networks through real-time information exchange, enabling operatives to organize, spread information and recruit new members into the organization. Hence, there is a relationship between Social Networks in which we can study how information is shared and how people become members of groups or even new friend relationships. A model is created by taking into account different levels of radicalism, information spread parameters, type of agents and crucial positions as training environments; where they become more radicalised. By running the simulation of the model, we have the resources to comprehend the expansion of these organizations through geographical spaces and the connections between the whole population.

1.2 Project goals

The main purpose of this project is to create a useful model for the analysing and understanding processes of terrorism network behaviours. This model must be able to show the connection between population and terrorism, giving an insight look of the conversion into a radicalised person or terrorist, also being able to identify how leaders recruit new members while they expand through geographical spaces.

To sum up, this project will allow an extensive analysis of Social Networks structure

and behaviour, pointing out how terrorist organizations expand affecting all type of agents.

Among the main goals of this project, we can highlight some tasks such as:

- Study of Social Networks behaviour and network types.
- Investigation of Network Simulators.
- Design of an Agent-based Social Model taking into account how information is spread and how networks expand.
- Design and Implementation of a Terrorist Network model.
- SNA techniques appliance.
- Build a Graph showing the results and created by the Simulation process.

1.3 Structure of this document

In this section we provide a brief overview of the chapters included in this document. The project is divided into the following sections:

Chapter 1 presents an introduction of the project. Besides, it is explained the main goals of the project and the context in which this project is developed.

Chapter 2 describes the main technologies on which this project relies. In this chapter some technologies will be analysed providing a technological background of the technologies used in the final work and special considerations.

Chapter 3 provides a description of the complete architecture of this project. It includes a global overview followed by the explanation of all the modules that make-up the project describing models generated.

Chapter 4 explains the results obtained in this project, analysing all experimentations made and showing the results obtained.

Chapter 5 joins the conclusions drawn from this project, problems faced and a brief future perspective.

CHAPTER 2

Enabling Technologies

Before designing the system environment and the visualisation of the model, this chapter is going to give a background of the project and also focus on techniques used. First of all, in Sec. 2.1 we are presenting a global vision about this thesis, giving an introduction of social simulators and identifying and explaining the considerations that we have made to design the model. Secondly, in Sec. 2.2 we are going to give an insight into Soil [15] , the technology that has made possible to run this model. To continue, NetworkX is explained in more detail in Sec. 2.3. Finally,in Sec. 2.4 Gephi, which is the technology in charge of the visualisation's simulation.

2.1 Background

Social simulators are complicated to design due to the complexity of human societies. The first problem we came with when we analyse human societies is that they are not lineal systems. This is why it is not possible to use classics models for the analysis and study of them.

Agent-based Social Simulation (ABSS) is one of the most common techniques used for analysing and simulating social networks. It merges computer simulation, social science

and agent-based technologies. Within this context, simulation consists in the design and implementation of a model based on a real system trying to simplify reality. Simulation is useful if we want to change some parameters and see the influence in the model, evaluating the results induced. Besides, social science is referred to sociology, psychology, politics, etc. Finally, agent-based is used for modelling social organizations, conducts or social behaviours.

There has been developed multiple models focused on radicalism and extremism using ABSS. Most of them shape the evolution process; steps followed to become radical and recruitment process [11]. Nowadays with new technologies based on Internet connexion it is necessary to comprehend the reasons and evolution process of becoming radical. Some models like [2] try to study this process by analysing social media, yihadist propagandist videos, etc. Others have based their ABSS models in modelling dynamic terrorist cells [13], individual radicalization [5] or extreme idea's evolution [12].

It is hard to create a complete terrorist network due to the difficulties to obtain information, they do not publish any detail about their members. First of all we need to collect as much information as we can of radical behaviour, social influence techniques, etc.

The novelty of this work consists in applying Social Network Analysis (SNA) techniques to calculate metrics(e.g. centrality, betweenness) as well as algorithms for community detection. From the point of view of visualising results and the methodology of this project, we have created our model using Geometric Graph Generators. They provide geographical positions to Agents, which is important to approximate as much as we can to real radicalism evolution, also being able to manage real environments and agents connections.

Finally, we are going to present the main information that we need to create our Terrorist Model, giving an overview of special details and how we have made our designing decisions.

In order to develop our model, we have based in some theories about how radical populations arise and endure [6]. Some methods have been also studied for modelling radicalised networks in today's work:

2.1.1 Geographical Simulators

As we have mentioned before, is really important to consider Geographical data when we want to simulate and model Terrorist Networks. When people are close they tend to be socially similar, so influence will be bigger. In this way, it is important for our model to consider real geographical connexions between people by generating an approximate Geographical model. Physical space aims to produce more insightful results when considering

the spread of terrorism.

In Sec 2.3 an insight look of nx-Geographical Simulators is given.

2.1.2 Social Network Analysis

Terrorism is based on influence as culture, relative inequality, age or residence. These network memberships typically shares similar attitudes and behaviour to non-radicalized networks. SNA is developed to highlight and analyse formal or informal relationships by collecting data from diverse organizations. It characterises network structures in terms of nodes and edges, focusing in relationships between actors instead of individual attributes.

Complex networks as terrorists, can be categorized into three types:

- ***Random Clustered:*** Networks are created by proximity between nodes.
- ***Small-world:*** Nodes are not neighbours of others, but most nodes can be reached from every other by a few hops or steps.
- ***Scale-free:*** In which network members prefer to make connection to the more popular existing members.

Considering these three networks types and Geographical Simulators, we can create our model trying to simulate real social connections and evaluating distances between nodes. With all of these, connections within nodes can be approximated to real behaviour.

In addition, we have used Social Network Analysis (SNA) techniques to calculate metrics(e.g. centrality, betweenness) as well as algorithms for community detection. In Chap. 4 we will explain how we use these Networkx facilities.

- ***degree_centrality:*** Centrality identify the most important vertices within a graph, including the most influential persons in a network. Degree centrality computes this degree for each node: the degree centrality for a node "v" is the fraction of nodes it is connected to. Degree centrality values are normalized by dividing the maximum possible degree in a simple-graph "nodes-1".

Listing 2.1: Networkx degree_centrality

```
def degree_centrality(G):
    centrality={}
    s=1.0/(len(G)-1.0)
```

```
    centrality=dict((n,d*s) for n,d in G.degree_iter())
    return centrality
Parameters
----------
G : A networkx graph
Returns
-------
nodes : Dictionary of nodes with degree centrality as the value.
Examples
--------
>>> G = nx.random_geometric_graph(20,0.1)
```

- ***betweeness_centrality:*** This algorithm computes the shortest-path betweenness centrality for nodes. If we have a node v, its betweenness centrality is the sum of the fraction of all-pairs the shortest path that pass through it:

$$c_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)} \tag{2.1}$$

Eq. (2.1) shows the algorithm for calculating betweenness centrality, where v is the set of nodes, where V is the set of nodes, $\sigma(s,t)$ is the number of shortest (s,t)-paths, and $\sigma(s,t|v)$ is the number of those paths that passing through some node v other than s,t:

Listing 2.2: Networkx betweeness_centrality

```
def betweenness_centrality(G, k=None, normalized=True, weight=None,
endpoints=False,
seed=None):
  Parameters
  ----------
  G : A NetworkX graph
  k : int, optional (default=None)
  normalized : bool, optional
  weight : None or string, optional
  endpoints : bool, optional
  Returns
  -------
  nodes : Dictionary with betweenness centrality as node value.
```

- ***community detection:***[1] This module implements useful algorithms for breaking down a social network into different potentially overlapping communities. The criteria for finding communities is based on the desire of maximize intra-community edges while minimizing inter-community edges. Formally, the algorithm tries to maximize the modularity of network, or the fraction of edges that fall within the community minus the expected fraction of edges if the edges were distributed by random. Communities should have a high number of intra-community edges, so by maximizing the modularity, we detect dense communities that have a high fraction of intra-

[1] https://bitbucket.org/taynaud/python-louvain

community edges. In our project we have used, as we will see in the following chapters, "best_partition".

Modularity is a measure of the structure of networks of graphs. It measures the strength of division of a network into modules (also called groups, clusters or communities). Modularity is the fraction of the edges that fall within the given groups minus the expected fraction if edges were distributed at random. It can be defined as shown in Eq. (2.2).

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{2.2}$$

- Aij edge weight between nodes i and j.
- ki and kj are the sum of weights edges' attached to nodes i and j.
- m is the sum of edge weights .
- ci and cj are the communities of nodes i and j.
- δ function.

The Louvain Method [17] for community detection is a method to extract communities from large networks and provides the highest modularity.

Listing 2.3: community.best_partition

```
community.best_partition(graph, partition=None, weight='weight', resolution=1.0)
Parameters
----------
G : A NetworkX graph to descompose
partition : dict, optional
weight : str, optional
resolution : double, optional
Returns
-------
partition : dictionnary. Communities numbered from 0 to number of communities.
```

- ***Louvain algorithm:*** This method consists in the repetition of two steps. The first step is a "greedy" assignment of nodes to communities, favouring local optimizations of modularity. Modularity is a scale value between -1 and 1 that measures the density of edges inside communities to edges outside of them. The second step is the definition of a new coarse-grained network in terms of the communities found in the first step. These two steps are repeated until no further modularity-increasing reassignments of communities are possible.

They are all functions of Networkx, which their use in our project is detailed in Sec. 2.3.

2.1.3 Dynamic Network Analysis

DNA [1] focuses in information spread. It can handle large dynamic, multi-mode and multi-link networks with varying levels of uncertainty. As opposed of SNA, DNA takes interactions of social features conditioning structure and behaviour of networks into account, highlighting and analysing formal and informal relationships.

2.1.4 Radicalization theories

Based on Cummings research [6], there are some theories of radicalism. Most of them are based on economic isolation, social marginalization and the perception of social disparity and injustice. As we have mentioned in the introduction, terrorists exploit this social habits by investing in social services, including health, education and even housing. Terrorists also offer something that the state is not providing as safety, security and giving access to opportunities for improving quality life. The improvement motivation is to find characteristics these models that can be mitigated in order to avoid extremist propensities due to all these gaps left by the government.

2.1.5 Agent Based Models

Agent Based Models have been used to study radicaliation in the past few years. They are an important tool for generating hypotheses about the behaviour of these Models that can be tested and analysed in a lab. Regarding these systems, all of them begin with a slow build-up of instability, a near inflection and often a random perturbation unexpected. All of these factors are really important when we are trying to look for variations that may cause a radical group to become active and violent. The behaviour of these systems might appear random but they have an internal change, adaptation and evolutionary mechanisms that reach the desire behaviour.

2.2 Soil

As we have seen, ABSS is one of the most common techniques used for analysing and simulating social networks with the aim of understanding and even forecasting their dynamics. In this chapter we are going to introduce an ABSS platform specially designed for modelling social networks. As we are going to see, this platform has numerous advantages comparing to others like NetLogo [19], Repast [16], Mesa, MASON [14] or other ABSS platforms.

Without considering Mesa, which is based in Python, most of these platforms are based in Java language. The main differences between Soil and these platforms is that they do not provide support for the analysis of social networks or the visualization and simulation of them.

Soil [15] is an Agent-based Social Simulator (ABSS) in Python that includes the characteristics of ABSS platforms and takes into account their deficiencies. Despite most ABSS platforms do not provide specific facilities for modelling, simulating and visualizing Social Networks (SNs), Soil is specifically designed for modelling them.

This platform is basically designed in Python, it provides interactive analysis, which is really helpful for simulation processes in which we want to see the evolution of all the parameters involved on them. Thanks to IPython[2] interface we can provide interactive analysis. Assisted by IPython we can get easily and at a real time the results of the model: graphs, parameters, etc.

2.2.1 Simulation Model for Social Networks

Soil platform is based on users represented by agents and a network that represents the social link between them. An example architecture of an implemented model is shown in Fig. 2.1 and consists of four main components[3]:

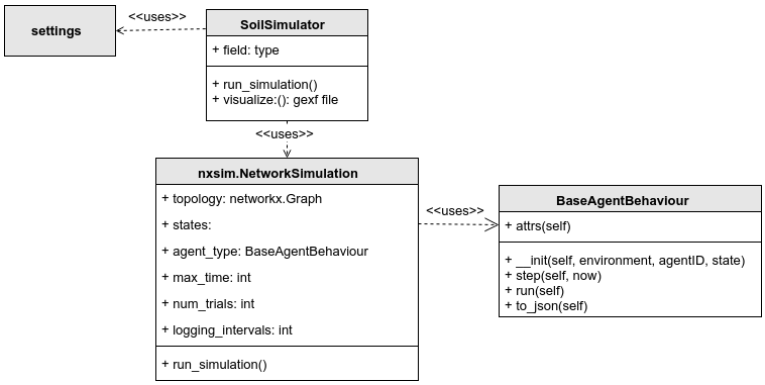


Figure 2.1: Simulation components

- ***NetworkSimulation class:*** is in charge of the network simulator engine. It provides forward-time simulation of events in a network based on nxsim and Simpy[4]. A graph

[2]https://ipython.org/

[3]*In the following chapters we will focus on our model, giving an insight in more detail of these components and their improvements.*

[4]https://pypi.python.org/pypi/simpy

is generated with NetworkX[5] based on configuration parameters. In next section we are going to focus on nxsim and NetworkX packages.

- ***BaseAgentBehaviour class:*** this class should be extended for each social network simulation model including the basic agent behaviour. It generates a JSON file that includes relevant information of agents for its analysis.
- ***Soil Simulator class:*** it is the class in charge of running the simulation pipeline described in Sect. 2.2.2. The result of this workflow consists in running the simulation and generating a visualisation file in GEXF which can be visualised in Gephi, it also provides interactive analysis using IPython.
- ***Settings:*** in this file we have the general settings for simulations and the settings of different models available in Soil.

2.2.2 Simulation Workflow

In this Fig. 2.2 we can see the Soil system's flow. Configuration, simulation and visualization are the steps of simulation's pipeline.

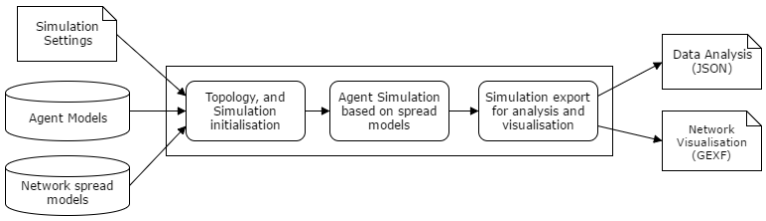


Figure 2.2: Social simulator's workflow

1. Main parameters of the simulation are configured in settings.py file.
2. Next step is the simulation, it can be done step by step or a number of steps. A SON file is generated once the simulation is finished collecting all relevant information and the GEXF file.
3. Finally, users can expand the analysis with the JSON file as well as visualized the .gexf file.

Considering all this information, we have opted to use Soil for the design and implementation of our model because of its facilities for modelling SNs and also for Social Network

[5]https://networkx.github.io

Analysis (SNA). We find useful this platform due to the possibility of creating static or dynamic graphs in which we can visualize the results of our studio and network's evolution through time and also geographical visualization. Regarding these considerations, NetworkX Python package is used to make easier these visualization requirements.

2.3 NetworkX

NetworkX [8] is a Python language software package for SNA analysis of small to medium networks. It provides capacity for creating, studying and manipulating graphs. It also includes graph algorithms for analysing graph properties. Moreover, NetworkX is interoperable with a wide range of graph formats': GML, GraphML, JSON and GEXF. Thus, further analysis can be made using disparate tools as Gephi.

Due to the interest of situate agents into a geographical space and try to make the model as much as realistic as possible, we have studied some NetworkX simulation generators for generating geometric graphs [6]:

- ***Random Geometric Graph:*** This simulator generator returns a random geometric graph in the unit cube. It places n nodes uniformly at random in the unit cube, two nodes are connected with an edge if their distances is below a radius threshold.

Listing 2.4: Random Geometric Graph

```
def random_geometric_graph(n, radius, dim=2, pos=None):
    Parameters
    ----------
      n : int. Number of nodes
      radius: float. Distance threshold value
      dim : int, optional. Dimension of graph
      pos : dict, optional
    Examples
    --------
      >>> G = nx.random_geometric_graph(20,0.1)
```

- ***Geographical Threshold Graph:*** The geographical threshold graph places n nodes uniformly at a rectangular domain. For each node u, a weight w is assigned. If weights are not specified they are assigned to nodes by drawing randomly from an the exponential distribution with rate parameter *lambda=1* and also if node positions are not specified they are randomly assigned from the uniform distribution.

- ***Geographical Threshold Edges:*** Given a threshold geographical graph generates edges with position and weights assigned as node attributes *"pos"* and *"weight"*.

[6] https://networkx.github.io/documentation/

- ***Waxman Graph:*** this simulation random graph model places n nodes uniformly at random in a rectangular domain. Two nodes *u,v* are connected with an edge with probability *P.*

- ***Navigable Small World Graph:*** This graph simulator generates a directed grid with additional low-range connections that are chosen randomly.

Once the study and analysis of disparate network simulator is made, we have opted to use these kinds of simulators. Despite problems faced, we have chosen these simulators due to their facilities to generate a geographical graph. Our interest lies in obtaining a graph that simulates as good as possible reality, making geographical connexions between nodes considering their proximity, friends of friends tend to talk to each other. Because of this, we can make a connection between social networks and terrorist networks, so these graph simulators fits our necessities.

2.3.1 nxsim

nxsim[7] is a Python package for simulating agents. It is based on SimPy[8] and NetworkX[9] that provides a basic ABSS framework for doing forward-time simulations of events occurring in a network. It requires a graph generated by NetworkX and an "agent" class to populate the network.

1. Create a graph using NetworkX as we have seen before:

Listing 2.5: NetworkX

```
    import networkx as nx
    number_of_nodes = 10
>>>G = nx.complete_graph(number_of_nodes)
```

2. Subclass BaseNetworkAgent to create an agent based on our needs. It will be described in more detail in the following chapters.

Listing 2.6: BaseNetworkAgent

```
from nxsim import BaseNetworkAgent
  # Just like subclassing a process in SimPy
```

[7]https://pypi.python.org/pypi/nxsim
[8]https://pypi.python.org/pypi/simpy
[9]https://networkx.github.io/

```
class MyAgent(BaseNetworkAgent):
def __init__(self, environment=None, agent_id=0, state=()):
  #Make sure to have these three keyword arguments
super().__init__(environment=environment, agent_id=agent_id, state=state)
  # Add your own attributes here
def run(self):
  # Add your behaviors here
```

3. Finally, we can set up our simulation by creating a NetworkSimulation instance and starting it up:

Listing 2.7: Network Simulation

```
from nxsim import NetworkSimulation
  # Initialize agent states. Let's assume everyone is normal.
  # Add keys as as necessary, but "id" must always refer to that state category
init_states = [{'id': 0, } for _ in range(number_of_nodes)]
  # Seed a zombie
init_states[5] = {'id': 1}
sim = NetworkSimulation(topology=G, states=init_states, agent_type=ZombieOutbreak,
max_time=30, dir_path='sim_01', num_trials=1, logging_interval=1.0)
```

2.4 Gephi

Gephi [4] is an open source tool to visualise and explore interactive networks. It is specially designed for graph and network analysis, displaying large networks in real time with a 3D render engine. One of the great advantages of this platform is that allow us to access into network data, filtering, navigating, manipulating, clustering and displaying dynamic graphs.

Before rendering a graph there are some steps and considerations that we have to follow. In Fig. 2.3 we can see Gephi work flow in which by introducing input data we can explore and export our graph and also analyse relevant parameters.

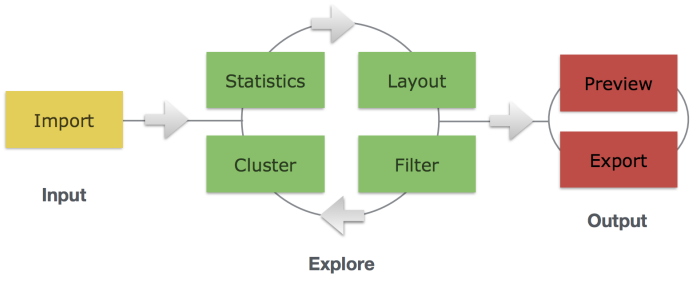


Figure 2.3: Gephi Steps

2.4.1 Input Data

In order to render a graph, Gephi supports most common file formats which can store node and edges attributes. If we want to give more details about our nodes, this platform also supports file formats in which we can set layouts and presentation information as position, size, colour, etc.

Native Gephi format is .gephi file. It includes multiple workspaces and supports diverse open formats that can be used to exchange data with other tools:

- ***GEXF*** [10]***:*** Graf Exchange XML Format is the language that we are going to use in this project. Is useful language for describing complex networks based on XML. GEXF is designed to represent network's elements and data associated to them, providing useful resources to support dynamic networks, change presentation information and managing all render parameters and basic graph data. Here we have an example of a GEXF file defying a static graph:

Listing 2.8: GEXF

```
<?xml version="1.0" encoding="UTF-8"?>
<gexf xmlns="http://www.gexf.net/1.2draft" version="1.2">
  <meta lastmodifieddate="2009-03-20">
    <creator>Gexf.net</creator>
    <description>A hello world! file</description>
    </meta>
    <graph mode="static" defaultedgetype="directed">
      <nodes>
        <node id="0" label="Hello" />
        <node id="1" label="Word" />
      </nodes>
      <edges>
        <edge id="0" source="0" target="1" />
      </edges>
  </graph>
</gexf>
```

- ***GraphML***[11]***:*** XML Graph Markup Language, this language is based also in XML but it does lacks the dynamic network drawing capability.

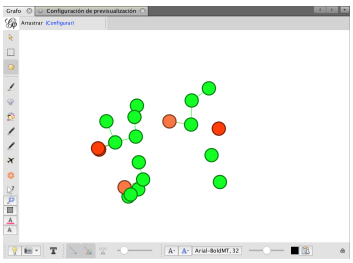
- ***GML***[12]***:*** Graph Modelling Language is a hierarchical ASCII-based file format for describing graphs. We are not going to use this format because of is a text-based language and it doesn't provide as much as resources as the others.

[10] https://gephi.org/gexf/format/
[11] http://graphml.graphdrawing.org/
[12] http://www.fim.uni-passau.de/index.php

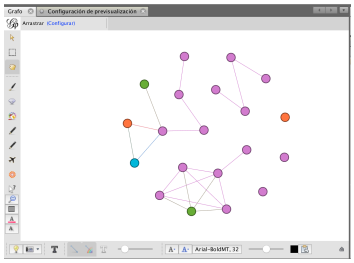
2.4.2 Explore

Once we have import our graph file we can explore it by applying layouts, filtering relevant information or seeing statistics of our model.

- ***Graph Layouts*** Gephi provides resources for applying layouts in order to visualize better our graphs and also change node appearance depending on nodes attributes as we can see in Fig. 2.4a and Fig. 2.4b



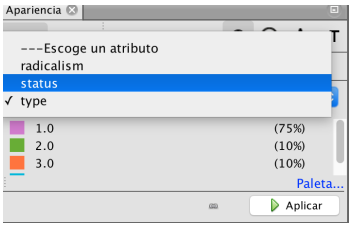
(a) Gephi Layouts. Filtering by status and Geo Layout



(b) Gephi Layouts. Filtering by type and Fruchterman Reingold Layout

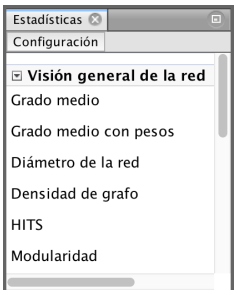
- ***Ranking Nodes*** Nodes can be re-sized and coloured based on their statistics or their attribute values, we have an example of "Apariencia" tab in Fig. 2.4a.

- ***Filtering Nodes*** The "Filtros" tab supports complex methods to temporarily highlight or hide subsets of nodes and edges in the graph. Nodes can be filtered by attribute value or based on node statistics ("Estadísticas" tab). Fig. 2.4b and Fig. 2.4c.



(a) Appearance

(b) Filter



(c) Statistics

- ***Data Laboratory Screen*** In this tab we can see an alternative tabular view of the same graph data. Here apart from being able to see every node and attribute, we can choose a node and see it highlighted in the graph.Fig. 2.4.

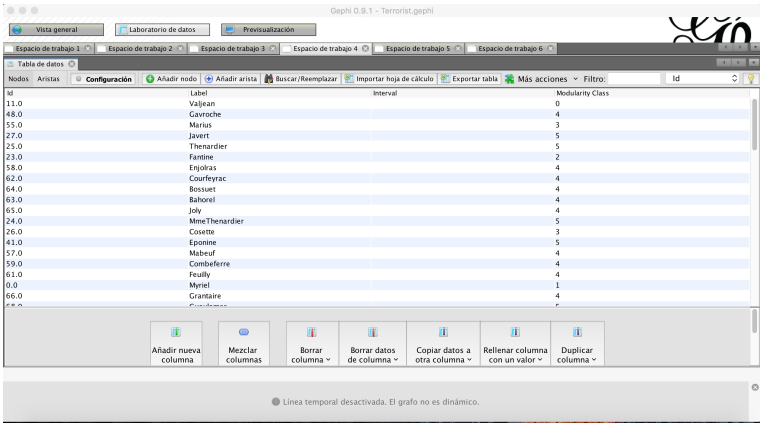


Figure 2.4: Data Laboratory

2.4.3 Output

After selecting our Layout, ranked our nodes, analysed nodes behaviour by seeing their attributes or filtered them, we can see a preview of the graph and export it in SVG, PNG, PDF or graph files. We can see an example of "Les Miserables" example graph in Fig. 2.5.

In conclusion, we use Gephi due to it is the only platform that allows us to filter nodes and see also the evolution of dynamic graphs by enabling "Temporal Line". Regarding all of these considerations, we consider that Gephi is really useful to see Terrorist evolution through time and filtering nodes by their type or status(Radical, Non radical or Neutral), identifying which nodes are changing their status and the connection between them.

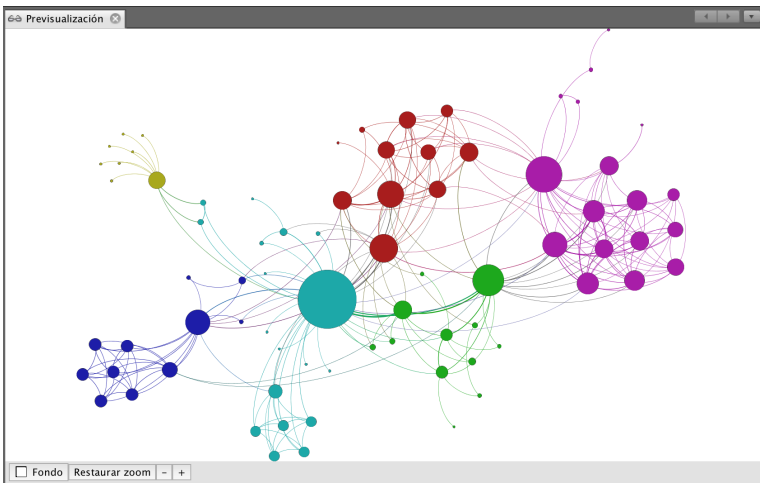


Figure 2.5: Pre visualisation

CHAPTER 3

Architecture

3.1 Introduction

In this chapter, we will explain the architecture of this project. We are going to provide a general overview about the connection between the different elements that composed it, showing how they have been implemented. First of all, we will present a global vision about the project architecture, identifying and explaining the considerations that we have made to design the general model, also giving an insight look of the agents and classes that our project is composed of.

3.2 General Model

In this section we are presenting the global architecture, defining the different modules that participate in the simulation.

Firs of all, in Fig.3.1 we present the agents involved in our model, which is based in the work by Paul Cummings [6]. This models consider four types of actors. It also distinguishes among people and places. People can play to roles: leaders (terrorists active on recruiting and propaganda) or population (people that could become radical). Regarding

places, havens (places where terrorists cannot be caught) and training areas (places where terrorists learn to be more effective). The reason to model places as agents is the radicalism of the places depending on the people located in them.

ACTORS	REAL SPACE	VIRTUAL SPACE
HAVENS	Places where terrorist are able to live and operate without taking any risk	Terrorist seek access to safe havens, adding influence to that specific area
TRAINING ENVIRONMENT	Terrorist learn and train, becoming more radical	Population and leaders have access to instruction becoming more radical
POPULATION	People living in an specific area	Neighbors that are connected and influenced by every agent
LEADERS	Terrorist people leading a group	Terrorist people that add influence to any agent connected with them

Figure 3.1: Actors

Each actor of our model is identified with an *id* or *status*, that is setted depending on their radicalism level. This level is based on their own parameter *rad* or *radicalism*. As we will see later, at the moment that a "NON-RADICAL" actor, for example, has a "RADICAL" or "NEUTRAL" neighbour connected to him, its level of radicalism grows, being able to cause a change in their status. Another attribute of actors is *type* that allow us to distinguish between "HAVENS", "POPULATION", "LEADERS" or "TRAINING" environments. In Sec. 3.2.1 we will see every detail of these actors, explaining how they interact and the influence they make to each other. Due to our interest of rendering a Geographical Graph, each agent has its own position in the space, connected to each other depending on their distance. Finally, attribute *fstatus* is just used to render the Graph in Gephi, due to Gephi's limitations. Thus, we can simulate and visualize "status" and "type" at once.

In the following Fig.3.2 we can see the simulation components and their connection. As we have mentioned before, we use SOIL simulator [15], for the simulation and the visualization processes. Within the simulation work flow, we are capable of identify the attributes of agents mentioned before.

As we can see in this figure, we use SOIL for simulating our Models. In Chap. 2 we have given an insight look into SOIL. This simulator is based on NetworkX (Sec. 2.3) and nxsim (Sec. 2.3.1). These packages give an allowance for simulating models and rendering graphs.

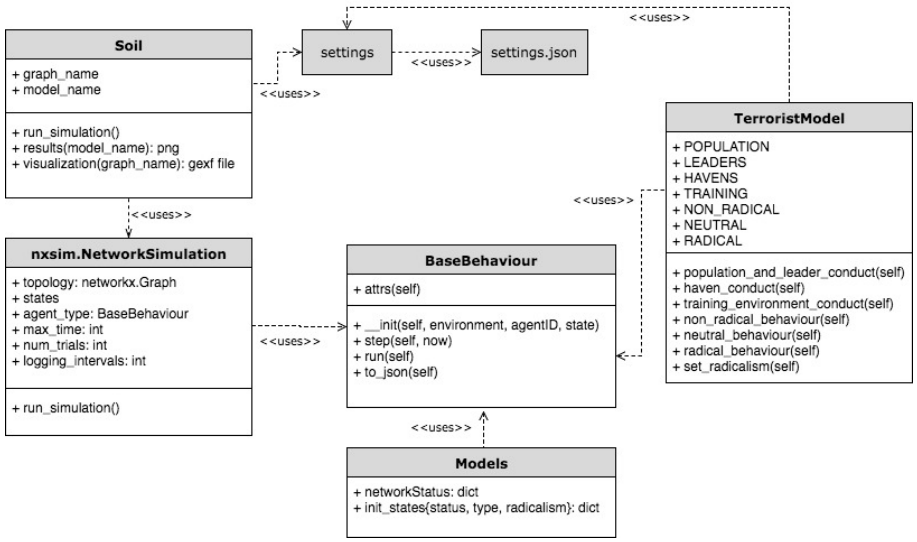


Figure 3.2: Simulation Process

SOIL generates a GEXF graph by calling to the methods *runsimulation()* and *visualization()*, in this one, a graph name is needed as parameter. After generating this file, we are capable of visualizing it using Gephi tool. In conjunction, SOIL platform generates line-graphs that allow us to study the evolution of designed models, representing how fast all agents change their status, type, level of radicalism or any further information we want to analyse. In the event of calling the method *results()*(giving a model name as parameter) a .png graph is generated representing these useful information. To summarize, we can export a GEXF file for rendering a dynamic graph and visualizing it using Gephi. At the same time, we can export a .png line-graph to contrast and expand the information. In addition, a .txt file is generated collecting all the attributes and information from every id, status, type, radicalism, etc.)

At last, in order to simulate any model desired, there are some considerations we have to deal with. It is necessary to extend "BaseBehaviour" class, in which standard agent's' behaviour is defined. So that, our "TerroristModel" Python class uses "BaseBehaviour" and also "settings" file. We will give a deeper view into these python classes in Chap. 4. At this moment we just want to mention the attributes of every agent *id* or *status*, *type* and *radicalism*, as well as the methods used in "TerroristModel" class. We also want to point up that for every agent involved in the model, there is a customized behaviour depending on their "status" or "type" (i.e *havenconduct()*).

Finally in Fig.3.3 we can see the attributes of every agent of the network and their connections. Each agent is connected to each other, adding influence. It is important to emphasise the fact that this Fig.3.3 only explains the relationships within actors filtering by

type. In Chap. 4 we will explain how connections and relationships are made considering all parameters. In this case, we can observe that physical places like "Havens" add influence into population and "Leaders". However, "Population" just add influence to "Leaders" but is not as strong as the influence that "Leaders" make to them.

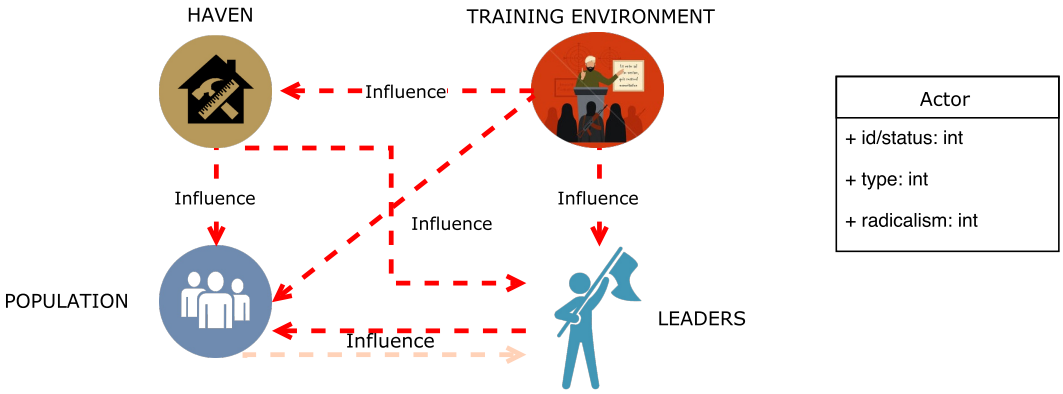


Figure 3.3: Actors or Agents

3.2.1 Agents

In this section we are going to describe every agent involved in our model, we will be giving an insight look on their behaviour and own attributes. One of the novelties of our project is that every agent is connected with each other considering geographical spaces. So that, for each agent (node) it is setted a latitude and longitude using, as we have mentioned in Sec 2.3, "Random Geometric Graph". Thanks to this graph generator, we achieved this goal of considering geographical spaces. It takes into account distance between nodes for generating their connections; two nodes are connected with an edge if their distances is below a radius threshold.

3.2.1.1 HAVENS

As we have said before, these are the places where terrorist seek access to safe refuges. In real life, these places are where terrorist live and operate without taking any risk of being discovered, also using these "Havens" to hide relevant information, weapons or sensitive material.

Havens' behaviour is defined in the following diagram3.4:

When simulation is initialized, every agent of the model has status: "NON-RADICAL", type: "POPULATION" and radicalism: "0". Havens do not change their type so when we

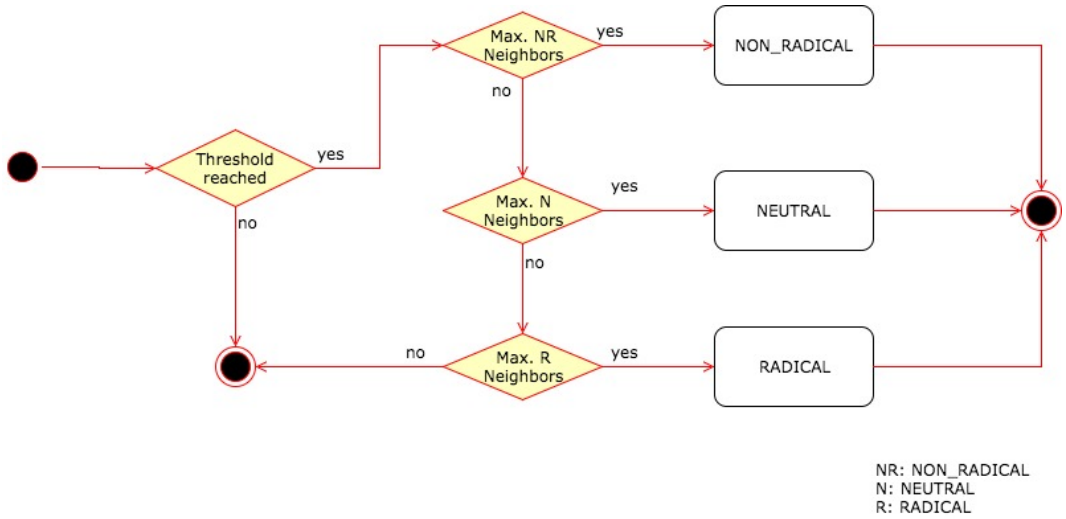


Figure 3.4: Havens' behaviour

set the number of nodes, we also set the quantity of them.

Havens change their status by getting neighbouring agents. When a haven is surrounded by "NEUTRAL" or "RADICAL" neighbours and the number of them reaches a threshold, havens' state will change to neighbour' status when its particular radicalism level is reached.

3.2.1.2 TRAINING ENVIRONMENT

In our models, Training Environments are places were radicalism has the maximum level(100%). This is why if any agent reached them, agent radicalism level will grow dangerously. In real spaces and life, Training Environments are places where terrorist learn new skills and train. They are also places where influence is really high due to relative inequality, social differences, etc.

3.2.1.3 POPULATION

Population's agents represent general people living in a specific area. Regarding our model, we achieve this goal by using Geographical Graphs. Each agent of this type has also a different status: "NON-RADICAL", "NEUTRAL" or "RADICAL".

At first, every agent is "NON-RADICAL" with a concrete level of radicalism. This is setted at the start, depending on the agent status, there will be an interval of radicalism that is detailed in the next chapter.

If we belong to the Population type and our status is “NON-RADICAL”, we will be influenced by our neighbours. In case of having a “NEUTRAL” neighbour we will be influenced with a certain level of influence defined in model’s settings. When we reach a concrete level of radicalism influenced by this neighbour, we will change our status to “NEUTRAL”. In the event of being exposed to a “RADICAL” neighbour, we contemplate the probability of increasing our radicalism but not to change our status, at least in one step, not becoming a “RADICAL” person. This is due to the difficulty for a “NON-RADICAL” neighbour to listen, trust or believe terrorist propaganda by their first encounter. However, because of the risk of having as a neighbour a “RADICAL” person, we will be influenced. As the example that we have seen before, when we reach a concrete level of radicalism, our status will change to “NEUTRAL”.

We study other possibilities as being near to a “Training Environment” where influence grows by adding additional influence to the usual level. This is why, in the case of having as a neighbour these places, our radicalism will grow dangerously changing our status more fast.

To this point, we have seen different scenarios of changing the status depending on neighbours’ behaviour or finding “Training Environments”. However, as we have seen before, there are also different types of People and places with their own status. In the case that our neighbour type is “Haven”, additional influenced will be added to the usual level, the same happens in the case that we meet a “Leader”, as we will see in the next subsection.

In Fig.3.5, we can observe this Populations’ behaviour explained before:

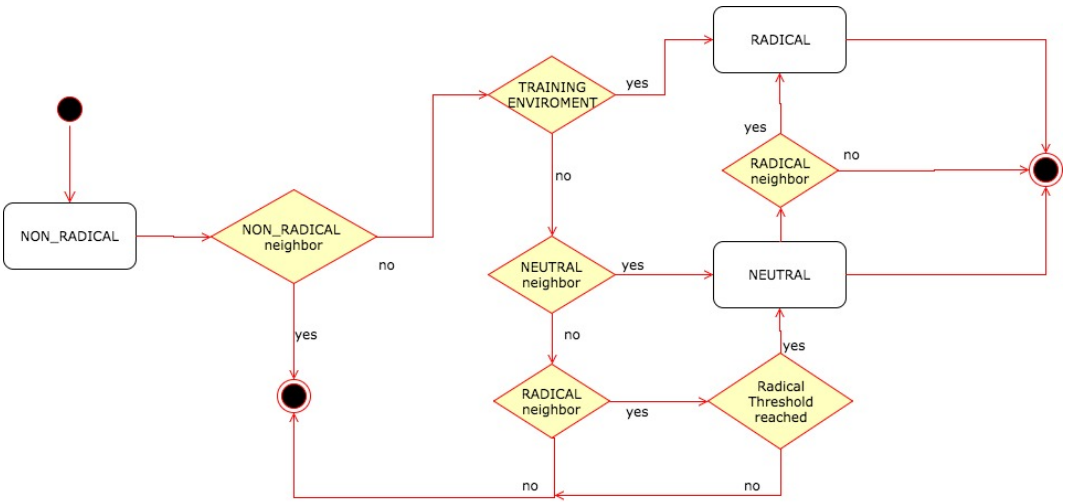


Figure 3.5: Populations’ behaviour

3.2.1.4 LEADERS

This type of nodes can be defined as neighbours that add additional influence to any agent connected with it. Thanks to SNA facilities, we analyse our network for getting the centrality of our nodes. At the moment that any agent radicalism level reaches a specific "RADICAL" level, it could change into "LEADER" type. To do so, we have used some functions that are explained in Chap. 4 in more detail. We need to consider that our network is divided into *communities*. Basically, when a node reaches this radicalism level, if in its community there is a Leader, they compare their *betweenness centrality*. If node's centrality is bigger than the centrality from Leader, the node will change its type into "LEADER" . If there is not a Leader in node's area, automatically it will become a "LEADER" of this area.

In Fig.3.5, we can observe this Leaders' behaviour:

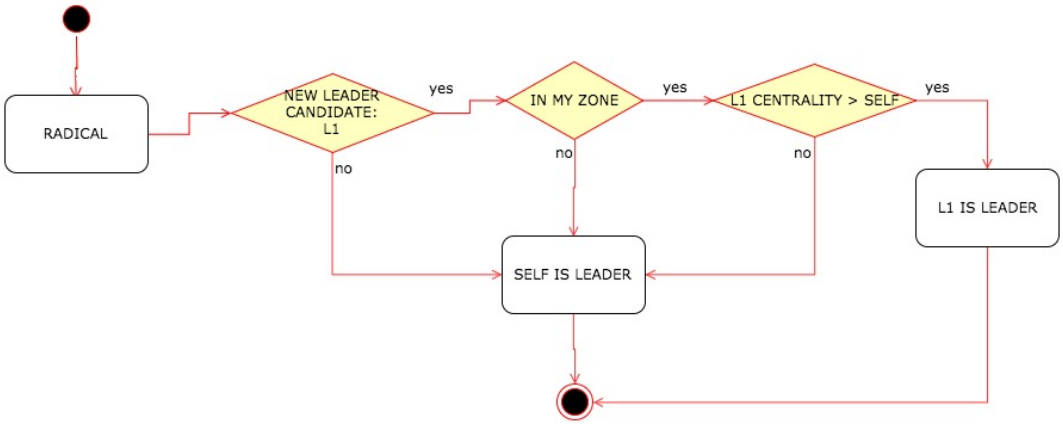


Figure 3.6: Leaders' behaviour

Leaders' behaviour is similar to Populations', the only difference is that when a "NON-RADICAL" Leader meets a "RADICAL" neighbour, he becomes "RADICAL". We have made this design decision because Leaders have more influence and they are also more radicalised than Population. They are also affected by Havens which give additional influence to them.

CHAPTER 4

Implementation

In this chapter we are going to describe the implementation of the terrorist model developed in this work explaining its base behaviour. First of all in Sec. 4.1 we will see the differences between this Model comparing to the implementation of other models that does not use Soil simulator. After that, in Sec. 4.2 we will give an insight look into the main functions that composed our model. Finally, in Sec. 4.3 we will explain the problems faced during the implementation of our model.

4.1 Background

The novelty of our project is that we consider geographical location to connect and place the agents or nodes of our model. Besides, we use Soil [15] platform for the design and implementation of our models and SNA metrics. As we have seen in Sect. 2.2, this simulator is based on Python language, giving the advantages of using the wide numbers of libraries for network analysis and SNA, for example Networkx or nxsim. Thus, the main difference between "Cummings Model" [6] and our models is that we use Soil, Python language and SNA metrics for the design and implementation of our model. As we have mentioned, our simulator and "Mesa" platform are the only ones based on Python, fulfilling

our requirements.

It is also interesting to point out that by using Soil, our model is the only one providing network facilities in case of needing to expand the analysis made.

To sum up, by using Soil ABSS platform and consequently Python, our terrorist model exploits the advantages that Python provides: popularity, gradual learning curve, clear syntax and availability of libraries for network processing and machine learning.

In the following sections we will see disparate Python functions used, detailing how visualisation has been made and the problems we have faced in order to achieve our goals.

4.2 Terrorist Model

In this section we are detailing Python functions used to run our model and simulation. First of all, we need to understand the parameters involved in the simulation. Next Fig.4.1 shows the parameters related to the network.

PARAMETER	INITIAL VALUE	OPTIONS	DESCRIPTION
NETWORK TYPE	0	Network type 0 is Random Geometric Graph	Soil gives the posibility of rendering a graph using disparate network generators
NUMBER OF NODES	80	As desired	Number of total nodes or agents in our Model
MAX TIME	50	As desired	Maximum time of the simulation
NUM TRIALS	1	-	Number of simulation trials
TIMEOUT	2	-	Time between steps

Figure 4.1: General Simulation Parameters

Regarding our terrorist model we can identify the following environment parameters.

All of these parameters are collected in settings.json and accessed by Soil from settins.py file as we have seen before in Fig. 3.2. The methodology followed to define these parameters and probabilities is explained in Chap. (5) Experimentation.

In order to apply SNA techniques to calculate metrics as betweenness centrality" and "community detection" we need to prepare some dictionaries to collect these data. In setting.py file we have add the following dictionaries, that will be used in model's functions.

PARAMETER	INITIAL VALUE	OPTIONS	DESCRIPTION
AGENT	Terrorist Model	-	This parameter allows to change between designed models
INITIAL POPULATION	85%	0-100%	With these parameters we can control the amount of agents depending on the number of nodes
INITIAL HAVENS	10%		
INITIAL TRAINING ENVIRONMENTS	5%		
INITIAL RADICALISM	10%	0-100%	Initial radicalism when model is initialised. Changes initial havens' state.
RELATIVE INEQUALITY	8%		When we increase the value, relative deprivation grows Increasing radicalism
INFORMATION SPREAD INTENSITY	20%		How much information is exchanged between agents
INFLUENCE	5%		Influence between agents
ADDITIONAL INFLUENCE	10%		Agents are more influenced by leaders and havens

Figure 4.2: Terrorist Model Parameters

Listing 4.1: settings.py

```
centrality_param = {}
partition_param={}
leaders={}
```

Once we have seen the parameters involved in the simulation and the attributes of the agents, we can move on to the functions designed. The following functions are in charge of the model's behaviour, we are going to explain the most important functions.

4.2.1 SNA functions

As previously mention, with the objective of change node's type to create new leaders, we have applied SNA metrics. Betweenness centrality and community detection, gives us the opportunity of creating new Leaders in a community and create it taking into account it's network connection. It is really important for our model both, geographical generators and SNA metrics like these. They let us to create a unique model approximating reality as much as possible. In the following code, we can observe how we prepare dictionaries created before, to use them in other functions:

Listing 4.2: SNA soil.py

```
    settings.partition_param = community.best_partition(G)
    settings.centrality_param = nx.betweenness_centrality(G).copy()
```

G is the Graph generated using "Random Geometric Graph" generator.

Finally, we can design a function for the creation of a new leader:

Listing 4.3: create_leader

```
def create_leader(agent):
  my_partition = get_partition(agent)
  old_leader = get_leader(my_partition)
  if old_leader == None:
    set_leader(my_partition, agent)
    return True
  else:
    my_centrality = get_centrality(agent)
    old_leader_centrality = get_centrality_given_id(old_leader)
    if my_centrality > old_leader_centrality:
      set_leader(my_partition, agent)
      return True
    return False
```

This function will be called during the evolution of the model. In particular, when an agent with "POPULATION" type reaches a certain level of radicalism. We consider the case where a leader is managing an area and another leader tries to appear. In this case, just the leader with the biggest centrality will become the new leader.

4.2.2 Soil init:

In this function we prepare the parameters stored in settings.json. We also set the number of agents of each type by applying the percentage declared and their status. In the following extract from the code we can see an example of population type.

Listing 4.4: Init

```
def __init__(self, environment=None, agent_id=0, state=()):
   super().__init__(environment=environment, agent_id=agent_id, state=state)
   self.population = settings.network_params["number_of_nodes"] * settings.
        environment_params['initial_population']
   if TerroristModel.num_agents < self.population:
     self.state['type'] = POPULATION
     TerroristModel.num_agents = TerroristModel.num_agents + 1
     random1 = random.random()
     if random1 < 0.7:
       self.state['id'] = NON_RADICAL
       self.state['fstatus'] = POPNON
     elif random1 >= 0.7 and random1 < 0.9:
```

```
            self.state['id'] = NEUTRAL
            self.state['fstatus'] = POPNE
        elif random1 >= 0.9:
        self.state['id'] = RADICAL
        self.state['fstatus'] = POPRAD
```

4.2.3 Step:

This function is called in every "step" to filter the behaviour of each agent and also saving their attributes in a dictionary. Once the simulation is completed, this dictionary is used to generate the GEXF file. We are just considering population type in order to summarize the code.

Listing 4.5: Step

```
def step(self, now):
  if self.state['type'] == POPULATION:
    self.population_and_leader_conduct()
  self.attrs['status'] = self.state['id']
  self.attrs['type'] = self.state['type']
  self.attrs['radicalism'] = self.state['rad']
  self.attrs['fstatus'] = self.state['fstatus']
  super().step(now)
```

4.2.4 Population_and_leader_conduct:

When an agent's type is "Population" or "Leader" this function will be called. As we can see, in this moment we filter by their *id* or *status* assigning them a concrete value of radicalism if necessary (the first time every agent call this function).

Listing 4.6: Population_and_leader_conduct

```
def population_and_leader_conduct(self):
  if self.state['id'] == NON_RADICAL:
    if (self.state['rad'] == 0.000):
      self.state['rad'] = self.set_radicalism()
    self.non_radical_behaviour()
  if self.state['id'] == NEUTRAL:
    if (self.state['rad'] == 0.000):
      self.state['type'] = LEADERS
      self.state['rad'] = self.set_radicalism()
    while self.state['id'] == RADICAL:
      self.radical_behaviour()
      break
    self.neutral_behaviour()
  if self.state['id'] == RADICAL:
    if (self.state['rad'] == 0.000):
      self.state['rad'] = self.set_radicalism()
   self.radical_behaviour()
```

4.2.5 Set_radicalism:

This function is called when we initialize the simulation. Based on Cummings work [6] , in the following code we distinguish every agent by their level of radicalism. Considering it, we can filter every agent into "NON RADICAL", "NEUTRAL" or "RADICAL". In Chapter 5 is detailed the methodology followed.

Listing 4.7: set_radicalism

```
def set_radicalism(self):
  if self.state['id'] == NON_RADICAL:
    radicalism = random.uniform(0.0, 0.29) * self.relative_inequality
    return radicalism
  elif self.state['id'] == NEUTRAL:
    radicalism = 0.3 + random.uniform(0.3, 0.59) * self.relative_inequality
    if radicalism >= 0.6:
      self.state['id'] = RADICAL
    return radicalism
  elif self.state['id'] == RADICAL:
    radicalism = 0.6 + random.uniform(0.6, 0.1) * self.relative_inequality
    return radicalism
```

4.2.6 Neutral_behaviour:

In this point, we are explaining how we manage the behaviour of each node. We are presenting the code of a "NEUTRAL" agent, but every one with other status will work in a very similar way. Every agent is going to look around to see the neighbours connected to them. Thanks to *get_neighboring_agents()* function, we can extract a list of connected neighbours. As we can observe in the following code, depending on the agent neighbour it will be influenced or not, causing a change of its status. We are giving just the part of the code where a leader is created when an agent radicalism reaches a particular radicalism level:

Listing 4.8: neutral_behaviour

```
    def neutral_behaviour(self):
    neighbors = self.get_neighboring_agents()
    for neighbor in neighbors:
    if neighbor.state['type'] == POPULATION and neighbor.state['id']==RADICAL:
      self.state['rad'] = self.state['rad'] + self.influence *                    self.
          information_spread_intensity
      if self.state['rad'] >= 0.62:
        if create_leader(self):
          self.state['type'] = LEADERS
          self.state['fstatus'] = LEADER
        elif self.state['type'] == LEADERS:
          self.state['type'] = POPULATION
          self.state['fstatus'] = POPRAD
```

Using these functions we are capable of simulating a Terrorist Network. The next step after declaring these functions and using Network Status dictionary, is running the simulation using soil.py file. In this Python file, we go over into every step (time-out) of the model. In each one of these steps, we take the agent Status (radicalism, status, fstatus and type), so that we can create a dynamic graph. In this file we also store the position of every node. With all these considerations we can create a GEXF file and render it using Gephi.

4.3 Problems Faced

Reaching this point of the project was not an easy task. In the following points we can observe the problems we had to deal with, some of them we were not able to fix them.

4.3.1 Geographical Graph Generators

First of all, we wanted to use each geographic generators we have research, but it was not possible to implement them using Soil platform and the way that we manage nodes. Some problems appeared using "Geographical Threshold Edge", "Waxman Graph" and "Navigable Small World Graph". The last one we wanted to use it in the first place, but it was not possible to adapt data for generating this type of graph.

Finally, we were able to adapt data for positions supported in "Random Geometric Graph". We dive into every node adding and attribute "position" with the positions generated by each Graph Generator. Using "spells" for time line and GEXF tag "viz',' we can easily assign to each node the position desired.

Listing 4.9: Graph Generators in soil.py file

```
for node in range(settings.network_params["number_of_nodes"]):
  G.node[node]['x'] = G.node[node]['pos'][0]
  G.node[node]['y'] = G.node[node]['pos'][1]
  G.node[node]['viz'] = {"position": {"x": G.node[node]['pos'][0], "y": G.node[node]['pos
      '][1], "z": 0.0}}
  del (G.node[node]['pos'])
```

4.3.2 Dynamic Graphs

Another problem we have come with is that we were not able to represent a dynamic graph using Gephi last version, 0.9.1. At first, we were storing time information linked to every

attribute of agents. Gephi was trying to extract this information without any success. After deep research, we have found that in this new version of Gephi "time intervals" declared by a "start" - "end" were not working. Considering this, we decided to use the other alternative supported in this version: "spells". So that, it has been possible to create a dynamic graph.

4.3.3 Gephi

Gephi is an open graph platform released in earlies 2000. We have come out with problems that affected, as we have seen in the previous section, to Dynamic Graphs and also to the visualization process.

In order to render geographical graphs successfully we have installed some plugins as "Geo Layout" or "Map of Countries" that allows us to represent geographical data created by geometric graph generators.

One of our desires was the assignment of an image to every node instead of a circular shape. Our interest is due to we want to difference between agent types, assigning an image to "HAVENS", "TRAINING ENVIRONMENTS", "POPULATION" or "LEADERS". We tried to install "Image Preview" plugin layout in order to be able to do so, but it was not supported in version 0.9.1. We tried to install the previous version, Gephi 0.8.2 but it was not possible to use some other important facilities. Trying to do so without using a plugin layout, we have tried to change GEXF archive adding in tag "viz" an attribute "image". Unfortunately changing shape of nodes to add an image, is not supported in any version of Gephi.

Consequently, we thought about changing nodes' shape by installing "Polygon Nodes" plugin that supposedly allows changing nodes' shape. We were thinking about to give different polygons to the disparate types of agent. Unfortunately changing shape of nodes is not supported in this version of Gephi.

In conclusion, despite Gephi is the best platform for visualising and study data from graphs, there is a need to improve its range and abilities. It would have been better if we could assign to each node an image for identifying their type. Even so, we have created "fstatus" attribute for visualizing both attributes at once.

4.3.4 Summary

We have faced some troubles while designing our model. For example, some problems programming in Python, specially the connexion between every attribute to GEXF file. In

addition, managing dynamic and geographical data or using Networkx package facilities.

As we have seen, the main problems we have had to deal with has been related with Gephi platform, but we have been capable to implement some basic features.

CHAPTER 5

Experimentation

Finally, in this chapter we are going to explain the methodology followed to create our model, Sec. 5.1. After that in Sec. 5.2, we are presenting disparate analysis of our terrorist model behaviour depending on different scenarios.

5.1 Methodology

This terrorist model explore the rise of radicalism based on influence made between agents and considering disparate environment parameters. Using SNA facilities we create communities with N number of nodes defined in "settings.json" file. As we have seen, each agent has their own attributes, behaviour, geographical location and connection with other agents. In next Fig.5.1 we can observe these parameters and how have been verified.

The verification process have been based in Cummings paper [6], Gini coefficient[1] and for "INFLUENCE" parameter on IEEE article [9].

In order to analyse *"Relative inequality"* we have studied *Gini coefficient*. It is a measure of statistical dispersion intended to represent the income or wealth distribution of a nation's

[1] http://www.ine.es/jaxiT3/Datos.htm?t=9966

PARAMETER	INITIAL VALUE	OPTIONS	VERIFICATION METHOD	DESCRIPTION
INITIAL RADICALISM	5%	0-100%	Examined considering "Paul Cummings" paper.	By increasing this parameter, havens will increase their radicalism level.
RELATIVE INEQUALITY	34%	0-100%	Based on Gini coefficient for 2016 in Spain	Increasing generates disparity in economic opportunity for agents, increasing radicalism.
INFORMATION SPREAD INTENSITY	10%	0-100%	Examined considering "Paul Cummings" paper.	How much information is necessary to accurately attack network.
INFLUENCE	2%	0-100%	Based on "Instituto Español de Estudios Estratégicos"	Increasing influence values per group generated linear increases in radicalism.
ADDITIONAL INFLUENCE	10%	0-100%		

Figure 5.1: Methodology

residents. This coefficient interval is between 0 and 1, if its 0 equality is reached but in the case its 1 inequality is reached. Radicalism is directly proportional affected by this parameter as followed.

$$radicalism = radicalism * relative\_inequality \tag{5.1}$$

This equation is applied in "set_radicalism" function, which is shown in Sect. 4.2.1.

Each node radicalism is affected depending on neighbours' type, in the case that a "Haven" or "Training environment" is affecting that node.

$$rad = rad + (influence + additional\_influence) * information\_spread\_intensity$$

5.2 Scenarios

In this section, we are presenting the results of different scenarios where we vary environment parameters. In all of them we are presenting the communities created. We need to know that the time is represented as an interval between $t = 0$ and $t = 50$.

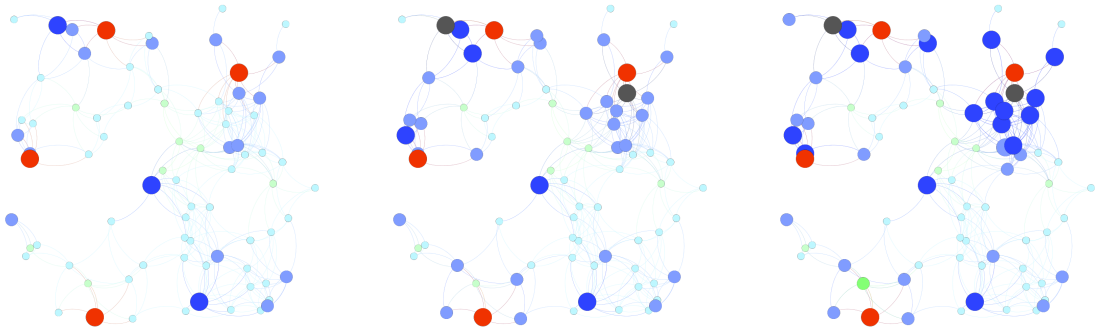
In Fig.5.2 we can see the relationships between colors, status and type of nodes. The more radicalised a node becomes, the darker its colour appear. The same happens with the size of the node. Thus, we will have three colours and sizes depending on the status: Non radical, Neutral and Radical.

Figure 5.2: Colour-node relationship

In order to understand the following graphs, we need to know that X-axis represents time and Y-axis represents agents' quantity depending on their status or type.

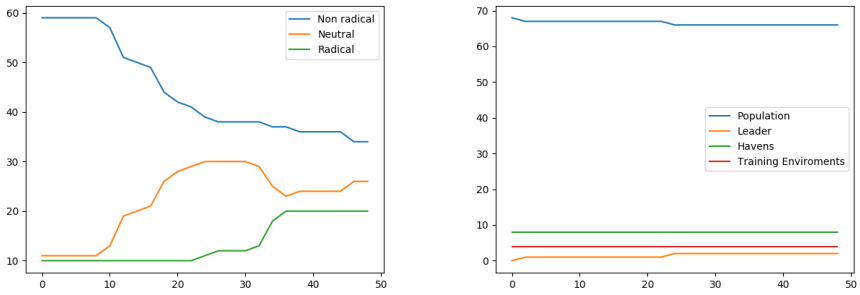
5.2.1 Low radicalism country model

In this scenario we have used initial value parameters, with a similar as simulating Spain country, taking into account that in our model we can not get back from "RADICAL" status.



(a) t=0 (b) t=25 (c) t=50

Figure 5.3: Low radicalism country model Gephi graph



(a) Status' evolution (b) Evolution of type

Figure 5.4: Low radicalism country model Line graph

In these graphs, we can observe the evolution of our model considering initial time, medium and final time intervals. We can easily distinguish between communities created,

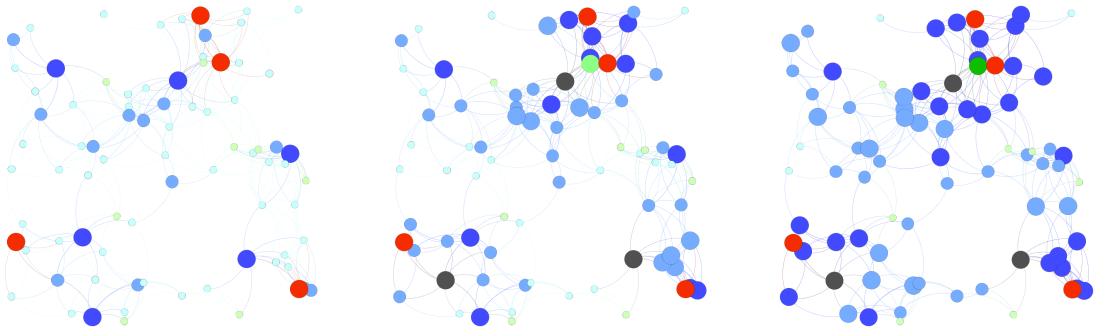
in most of them a training environment is situated. Leader emergence can easily be seen (Gray dots), we can check that a leader is created considering SNA centrality and also how radicalism becomes higher near them. We need to consider that due to the radicalism is really low, there are not a lot of leaders (radicalism need to grow to a certain level).

Finally, in Fig.5.4a we can see the evolution of nodes' status and in Fig.5.4b nodes' type evolution through time.

5.2.2 Increasing initial radicalism and influence scenario

In this case we are trying to simulate a semi radical country. In which initial radicalism and influence has been raised:

- ***Initial radicalism:*** 20%. The number of "RADICAL" havens will be increased.
- ***Influence:*** 5%. The influence between agents we will be more relevant in radicalism growth.

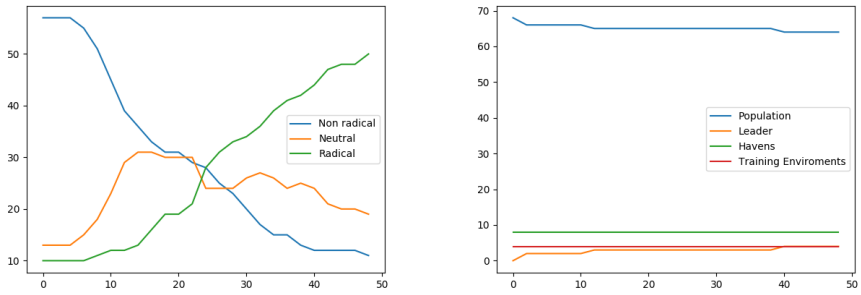


(a) t=0 (b) t=25 (c) t=50

Figure 5.5: Increasing initial radicalism and influence scenario Gephi graph

As well as previous graphs, in Fig.5.6 we can analyse the evolution of this scenario considering all agents and behaviours. We can easily observe the difference between previous graphs. In this scenario, "Non radical" agents decrease faster, while "Neutral" and "Radical" agents grows dangerously. "Radical" agents specially grows faster due to the increment of the parameters studied in this scenario. It is also important to mention that "Leaders" appear also in more quantity and faster, due to the global level of radicalism.

Finally, in the following line-graphs we can observe this environment evolution. In Fig.5.6a we can see the evolution of nodes' status and in Fig.5.6b nodes' type evolution.



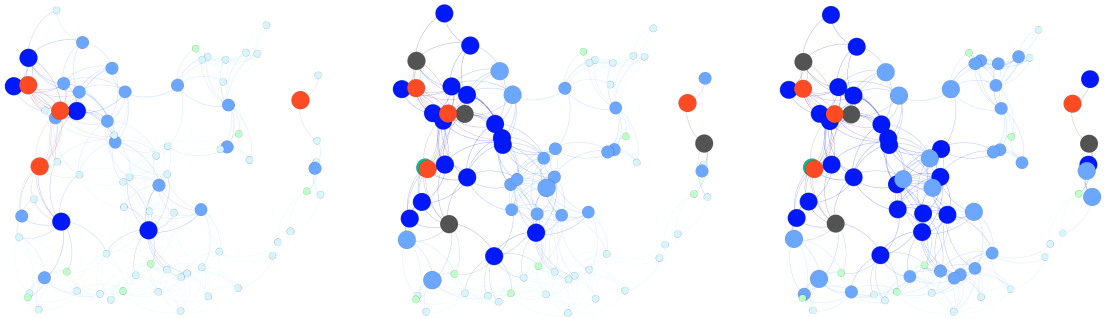
(a) Status' evolution (b) Evolution of type

Figure 5.6: Increasing initial radicalism and influence scenario Line graph

5.2.3 Increasing information spread intensity

In this scenario we are increasing information spread intensity while the others maintain their level.

- ***Information spread intensity:*** 30%. As we have seen by increasing this parameter, due to it is multiplying to influence, general radicalism grows dangerously.

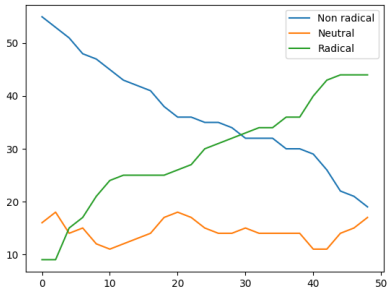


(a) t=0 (b) t=25 (c) t=50

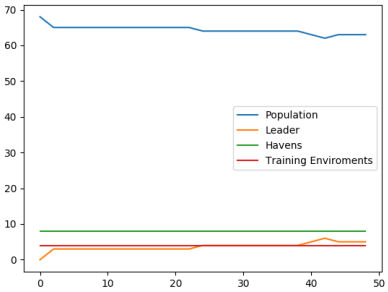
Figure 5.7: Increasing initial radicalism and influence scenario Gephi graph

As well as previous graphs, in Fig.5.8 we can analyse the evolution of this scenario considering all agents and behaviours. We can easily observe the difference between previous graphs. In this scenario, "Non radical" agents decrease faster, while "Neutral" agents increase slowly and "Radical" agents grows really fast. "Radical" agents specially grows faster due to the increment of this parameter because it is proportionally direct to influence and additional influence. So that, radicalism of each agent will increase really fast.

Finally, in the following line-graphs we can observe this environment evolution. In Fig.5.8a we can see the evolution of nodes' status and in Fig.5.8b nodes' type evolution.



(a) Status' evolution



(b) Evolution of type

Figure 5.8: Increasing initial radicalism and influence scenario Line graph

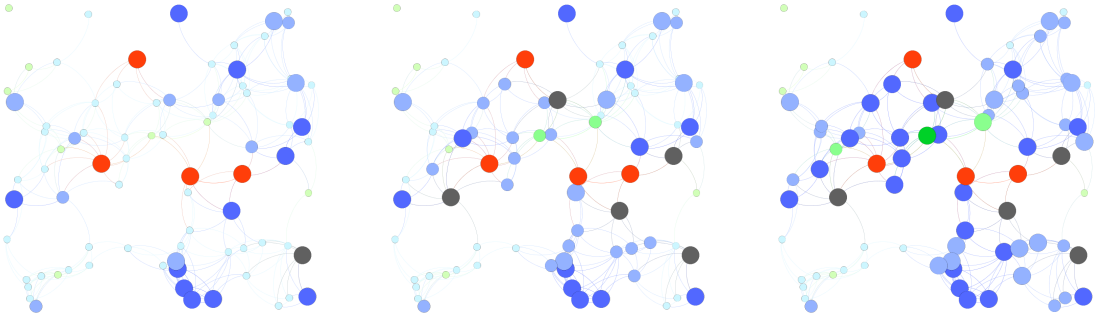
5.2.4 Increasing relative inequality and training environments

Finally, in this scenario we are visualizing the variation of our model when we increase relative inequality level and the number of training environments.

- ***Relative inequality:*** 60%. As we have seen by increasing this parameter, we generate disparity in economic opportunity for agents. As we have seen, we use this parameter to set the initial radicalism of every agent, when relative inequality grows, the initial radicalism of each agent also grows.
- ***Number of training environments:*** 10%. Due to these places where terrorist hide, increasing this parameter makes also grow the number of "Radical" agents in our model.

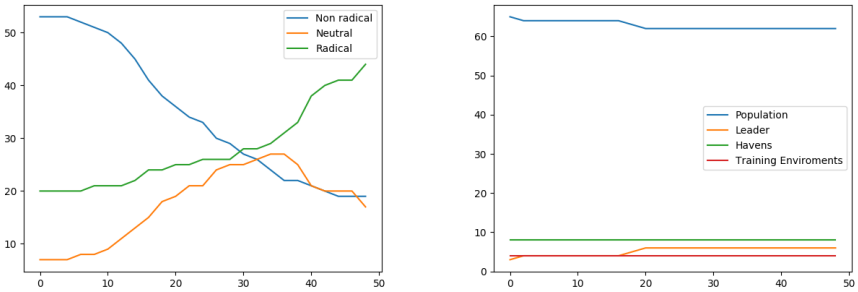
In Fig.5.10 we can analyse the evolution of this scenario considering all agents and behaviours. We can easily observe the difference between previous graphs. In this scenario, we can see that initial number of "Radical" agents starts in a really high level. This makes "Radical" people grow fast and also the appearance of "Leaders", due to the high increase of radicalism level.

Finally, in the following line-graphs we can observe this environment evolution. In Fig.5.10a we can see the evolution of nodes' status and in Fig.5.10b nodes' type evolution:



(a) t=0 (b) t=25 (c) t=50

Figure 5.9: Increasing relative inequality and training environments Gephi graph



(a) Status' evolution (b) Evolution of type

Figure 5.10: Increasing relative inequality and training environments Line graph

CHAPTER 6

Conclusions and future work

6.1 Introduction

In this chapter we will describe the conclusions extracted from this work, achievements in Sec. 6.3 and finally in Sec. 6.4 suggestions about future work.

6.2 Conclusions

To conclude this project we are going to resume the principal concepts that has been explained in this document. We have designed and implemented a Terrorist Model which can be used to simulate terrorists' behaviour. It is also possible to create new models based on a General Terrorist Model to study new cases.

We have made a study of Social Network Analysis in order to generate a realistic Terrorist model, taking into account connections made between nodes, like "betweenness centrality". This concept can be associated as "friends of friends" in a social network. Using this Networkx algorithm in addition to community detection and ABSS techniques, we have succeeded develop a "Terrorist model" using Soil simulator platform.

In order to visualise our results, we have used Gephi and Python functions (in which is based Soil) for generating graphs and lineal graphs. Using Gephi has been a tedious chore due to its' limitations but we have achieved all goals we have planed to do.

To sum up, the novelty of our project relies on the application of SNA and ABSS techniques for rendering a dynamic graph. With this graph we are able to study the evolution of terrorism in a concrete area and understand better how some social parameters affect terrorism evolution.

6.3 Achieved goals

To continue, in this section we will explain achieved features and goals during the evolution of this project.

Network simulators and study of Social Networks Before designing any model, we needed to gather information about Social Networks and how they expand and work. This is due to the relationship between them and how terrorist network evolves. Finally, we needed to investigate about disparate types of network simulators to generate a dynamic graph and see the evolution in time.

Design and Implementation of a Terrorist Network model Once the study of networks simulators and social networks is completed we are able to generate a Terrorist Model. We have studied which parameters are involved in Terrorist Network expansion. After that, we have all the information needed to generate our model. Finally, we have accomplished our goal using this parameters and generating the desired behaviour, storing parameters in .json files.

SNA techniques appliance To continue, after generating our model, we have improved it using these techniques. In order to consider real life connections, in the case of becoming a "Leader" agent, this node applies SNA techniques. These techniques are betweenness centrality" and "Community detection". The first one help us to create a leader basing on its connection in the network and the second one allow us to create different communities. This communities creation fills both necessities, creating a leader in a specific area and creating nodes in a local area.

Graphs Finally, we can present our results by generating a Gephi Graph from a GEXF file. We also generate a .txt file with all information of nodes and a lineal-graph. Gephi graph is used to see the evolution of the dynamic and geographical graph, being able

to see different geographical areas generated with SNA facilities and geographical graphs. We are also able to see the evolution through time as well as in lineal-graphs.

6.4 Future work

In the following section we are explaining the possible new features or improvements that could be done to the project.

Gephi Improvements Once Gephi update its functions, it will be interesting to generate a graph with a background desired image. We consider that is also necessary to change nodes shape for identifying nodes more easily. For example, by adding an image to each node or changing their shape into polygonal.

Infiltrators Model This is the main point for future development. It will be interesting to add infiltrators agents in order to slow down terrorism evolution. An infiltrator agent can be added and when is connected with a neighbour, it can reduce its radicalism level, radicalism spread will be slowed down.

Possible uses In this project we have provided a general model for the study and approximation of terrorism behaviour in real life. It will be stimulating to improve in a more realistic way this project to predict with a high exit percentage radicalism behaviour. With all of these, we want to approximate to this goal. These models will be useful to being one step ahead of terrorist intentions.

Graphic interface Finally, we have thought about the possibility of generating a web interface. In that interface we can manage the parameters using sliders or even having the possibility of add icons of each agent type.

Bibliography

[1] Charu Aggarwal and Karthik Subbian. Evolutionary network analysis: A survey. *ACM Computing Surveys (CSUR)*, 47(1):10, 2014.

[2] Jamie Bartlett and Carl Miller. The edge of violence: Towards telling the difference between violent and non-violent radicalization. *Terrorism and Political Violence*, 24(1):1–21, 2012.

[3] Benedetta Berti. *Armed Political Organizations: from conflict to integration.* JHU Press, 2013.

[4] Ken Cherven. *Network graph analysis and visualization with Gephi.* Packt Publishing Ltd, 2013.

[5] Claudio Cioffi-Revilla and Joseph F Harrison. Pandemonium in silico: Individual radicalization for agent-based modeling. 2011.

[6] Paul Cummings and Chalinda Weerasinghe. Paper title: Modeling the characteristics of radical ideological growth using an agent based model methodology.

[7] Oliver Gruebner, Martin Sykora, Sarah R Lowe, Ketan Shankardass, Ludovic Trinquart, Tom Jackson, SV Subramanian, and Sandro Galea. Mental health surveillance after the terrorist attacks in paris. *The Lancet*, 387(10034):2195–2196, 2016.

[8] Aric Hagberg, Dan Schult, Pieter Swart, D Conway, L Séguin-Charbonneau, C Ellison, B Edwards, and J Torrents. Networkx. high productivity software for complex networks. *Webová strá nka https://networkx. lanl. gov/wiki*, 2013.

[9] Eguskine Lejarza Illaro. Terrorismo islamista en las redes–la yihad electrónica. *IEEE. Instituto Español de Estudios Estratégicos.(100/2015)*, page 4, 2015.

[10] Mitchell Joblin and Wolfgang Mauerer. An interactive survey application for validating social network analysis techniques. *R Journal*, 8(1), 2016.

[11] Michael King and Donald M Taylor. The radicalization of homegrown jihadists: A review of theoretical models and social psychological evidence. *Terrorism and Political Violence*, 23(4):602–622, 2011.

[12] Bo Li, Duoyong Sun, Shuquan Guo, and Zihan Lin. Agent based simulation of group emotions evolution and strategy intervention in extreme events. *Discrete Dynamics in Nature and Society*, 2014, 2014.

[13] Bo Li, Duoyong Sun, Renqi Zhu, and Ze Li. Agent based modeling on organizational dynamics of terrorist network. *Discrete Dynamics in Nature and Society*, 2015, 2015.

[14] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527, 2005.

[15] Eduardo Merino, Jesús M. Sánchez, David García Martín, J. Fernando Sánchez-Rada, and Carlos A. Iglesias. *Modeling Social Influence in Social Networks with SOIL, a Python Agent-based Social Simulator*. 2017 2017.

[16] Jonathan Ozik, Nicholson Collier, Todd Combs, Charles M Macal, and Michael North. Repast simphony statecharts. *Journal of Artificial Societies and Social Simulation*, 18(3):11, 2015.

[17] A Scherrer and V Blondel. The louvain method for community detection in large networks. université catholique de louvain, louvain-la-neuve, belgium. 2011, 2014.

[18] Bart Schuurman and Quirine Eijkman. Indicators of terrorist intent and capability: Tools for threat assessment. *Dynamics of Asymmetric Conflict*, 8(3):215–231, 2015.

[19] Uri Wilensky and William Rand. *An introduction to agent-based modeling: modeling natural, social, and engineered complex systems with NetLogo*. MIT Press, 2015.