Trace generator for Opportunistic Networks routing protocols through heuristic techniques

Marc Amin Abou-Hamdan Chamorro

Abstract– This report is concerning about the execution of the project to build an algorithm capable of generating *The ONE simulator* Scenarios in order to test delay tolerant networks (DTN) algorithms, specifically opportunistic networks, as well as the methodology employed to carry it out, special mention to the Simulated Annealing technique, which will be the key to the impartial *The ONE simulator* Scenarios Generator. However, it is also explained what kind of Scenario is going to be built. Furthermore, the report will show the state of the art, specifying other methods to obtain these Scenarios, such as by getting prebuild derived from human mobility ones or real-based ones. Moreover, the report will include the planning of the project, updates on the development of the project, including the explanation of the structure of the code and a guide to use it, and whether it has followed the planfication or not and why. Finally it will include the sources consulted in addition to some acknowledgments.

Keywords– DTN, Store and Forward, The ONE Simulator (The ONE), Opportunistic Networks (OppNets), Simulated Annealing, Scenario, Scene, Matrix, Gantt diagram, Nodes, class diagram, Python, Numpy, generator, absolute network density, sparsity distribution, centrality, quadrant.

1 Introduction

Opportunistic networks are wireless connection networks where the topology of the nodes and the connection between them change continuously and is a priori undefinable. The connection between nodes happens only when they are in range and can disappear whenever because nodes get far from each other, so end to end principle between source and destination nodes is not accomplished.

Opportunistic Networks are characterized by having nodes which can be disconnected or isolated but if they are holding a message then they will keep running the communication protocol alive until they reconnect and pass the message to another suitable node to make the message reach its destination. This is thanks to the store and forward technique, which holds that a node will store the message in its memory at least until it complete the cicle of the communication algorithm. In other words that is when it passes a message to another suitable node, the message is dropped when the TTL has run out or the algorithm runs a mechanism to make the node know that the message is deprecated.

• E-mail: marcamin.abouhamdan@e-campus.uab.cat
• University Mention made: Information Technology
• Project tutorized by: Diego Mauricio Freire Bastidas (DEIC)
• Year 2019/20

The task of building a routing algorithm for Opportunistic Networks is a hard job due to its main issues: security, privacy and its indeterminable and continous changing topology. This is why there is not a standard routing algorithm for opporunistic networks yet.

However a large variety of routing protocols have emerged as options to become the standard protocol for routing the opportunistic network messages. They can be classified by different ways and one classification could be the one in Figure 1.

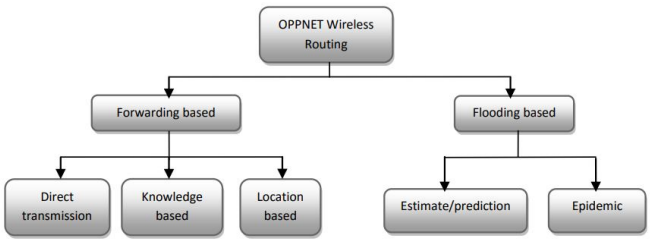

Fig. 1: Classification of Opportunistic Networks routing protocols [1].

We can find many different routing protocols inside every type of them, each one with its particularities, with its pros ans cons. And it is at this point where this Treball de Fi de Grau starts to get involved. As said before, all of the routing protocols have their pros and cons and the best method to

prove their efficiency is by running a simulation in which each routing protocol can be tested.

As a consequence, the author of every routing protocol has run its own simulation with its particular results with different metrics such as dropped messages, unreached destination or network congestion among others. The problem comes when every routing protocol creator builds its own simulaion Scenario, which is the nodes location in a specified area or map where the simulation will be run and at a concrete time. With this concept, the Scenario represents the sequence of nodes' locations through time inside the map, what resembles a movie. Very frequently, the nodes location is determining in the results obtained from the simulation, as they can be located specially to make a specific routing protocol to excel in its simulation results.

For this reason we believe that it is important to be able to build an 'impartial' Scenario, in order to get more fair results in the testing of routing protocols. This Scenario will need some specific characteristics to be the most impartial as possible.

2 OBJECTIVE

It is logical that the authors of the different routing protocols build their optimal Scenario to make their protocol look more efficient given that there is not a single Standard testing Scenario. Due to the need of a Scenario to test their routing protocol, they must build one, so it is obvious that they will not make their protocol's task hard by building a challenging Scenario, but building a proper Scenario to make their routing protocols shine.

In consequence, our task is to try to build a Scenario generator capable of building impartial Scenarios with a very specified characteristics that will give it balance.

In order to get an impartial Scenario we will build an algorithm able to generate a Scenario with the characteristics that will be introduced in it. One of the most widespread Opportunistic Networks simulators is *The ONE*, so we will build a Scenario to be run in the *The ONE* software.

The corresponding characteristics were the relative density of the nodes, which are relative to the quadrants in which we divide our area of action (map); the centrality of the nodes, that shows the mean number of connections of every node, meaning connections with those nodes in a predetermined range to establish a connection, in other words, the number of neighbour nodes; relative dispersion, relative to the quadrants in which the map is divided, meaning the number of quadrants without nodes; the number of clusters per *scene* (this term will be explained later on, in section 3.2), meaning the number of agruppations of nodes divided per the number of *scenes* in our Scenario; the mean distance between clusters per *scene* and the mean variance of the size of the clusters between *scenes*.

However, these characteristics have changed following the decisions made in the meeting. Currently, the characteristics are:

- Absolute Network Density, that is a portion between the actual connections over the total connections, where total connections refers to the connections that could potentially exist between nodes and actual connections are the ones that actually exist. For unweighted networks of N nodes without multiple connections, the network structure can be represented as an NxN adjacency matrix A.

- Quadrant, which consists of two characteristics at the same time. One of them is *Quadrant* itself, which is a sector of the Scenario that represents the physical division of the area into q blocks. where q is an integer in a 1 to 6 range. The other one is *Nodes : Quadrant*, that is the number of nodes within a quadrant. This relationship is defined as two integers, $n : q$, which means that the *Scenario* has a mean distribution of n nodes in q quadrants.

- Centrality, which consists of three chracteristics at the same time. One of them is *Closeness Centrality*. Another one is *Betweenness Centrality*. Finally, the last one is called *Clustering Centrality*.

- Sparsity Distribution. This characteristics shows the physical distribution over the area of the *Scenario*, taking into account how many areas have low presence of nodes.

3 METHODOLOGY

3.1 Simulated Annealing

In order to build this impartial Scenario with the optimal values for the characteristics we want to achieve, we will use a method called *Simulated annealing*.

Simulated annealing is a probabilistic technique for aproximating the global optimum of a given function. Simulated annealing is a specially used technique when trying to find the global maximum, because, although the problem may be very complex, it can overcome the complexities with its performance.

The term *annealing* comes from metallurgy and is a techinque where metal is heated and then controlledly cooled to make the size of their crystals and reduce defects.

In our case we will use the simulated annealing technique to find the best Scenario, *heating* heavily the nodes at the beginning and iteration after iteration of improvement, the range of movement of the node will be reduced since it makes the Scenario get the characteristics it has been required previously.

3.2 Our Scenario

In order to develop our task efficiently we take a different concept of the Scenario from the conventional one. Although the conventional Scenario concept inlcudes the messages that nodes exchange, we will not include them in our Scenario concept because it does not contribute in any way.

Furthermore, we divide the *Scenario* in different *scenes*, which are different location of nodes in the map corresponding to their position at a concrete instant, making the agruppation of *scenes* an equivalence of the movement of the different nodes through time inside the area of analysis (the map).

3.3 Work methodology changes

After a meeting with the head of the project, the decision of acquire a new methodology was taken. The methodology under consideration was the prototype based methodology, in which the project progress will be reflected in periodically improved prototypes.

Thus the project will be divided in different phases, which will include a prototype each, every one of them being an improved version of the previous one.

At the beginning, in the First Phase, the prototype includes a Scenario generator with random initial positions for the nodes, which will have a restricted mobility through time.

Afterwards, in the Second Phase, the prototype includes the Scenario generator with random initial positions for the nodes, with a varying restricted mobility through time, by executing a certain amount of iterations, which, at the end of each one, the characteristics of each iteration Scenario will be calculated and evaluated in order to vary this movement restriction, to achieve a better nodes location through the map that will end up improving the Scenario characteristics values. With this, we will be able to hypothetically get a Scenario built with the required set of characteristics.

4 Planning

At the time of executing the project we proceeded in an organized way, following the planning of Figure 2.

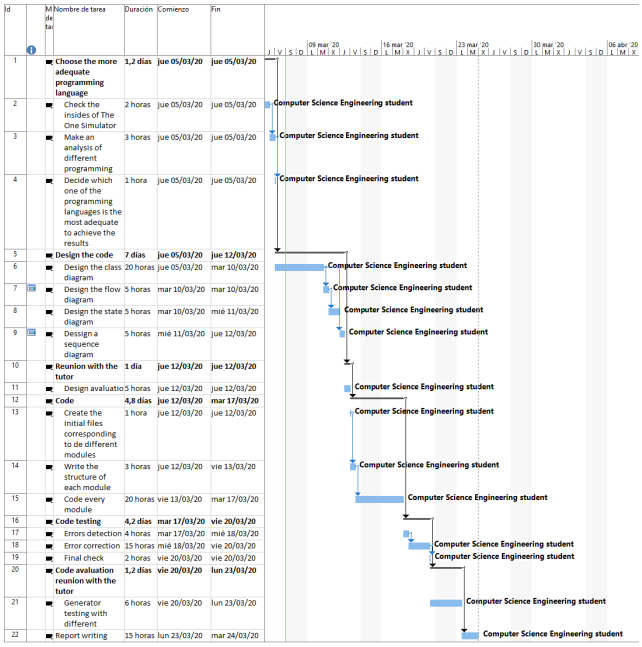

Fig. 2: Gantt diagram of the project.

4.1 Setbacks and planning changes or updates

Due to a viral pandemic in the beginning of 2020, the whole world has entered a state of global quarantine, having to stay locked at our homes. This situation has made it difficult to celebrate meetings within the research group, delaying the project features decisions. Thus the project has had to be paused during this time until the meeting finally was celebrated. However, the majority of the information obtained has not been able to be included in the project completely yet, although it will be presented as extra data in this report.

The planing after that has got the appearance shown in Figure 3

In the First phase we have achieved to build the base structure of the program, which generates a whole Scenario based on random initial location for the nodes and random restricted movement of them through time.

As far as we achieved this first objective, phase one of the project had been concluded and from then on, the next objective has been to develop the simulated annealing behaviour of the program.

Unfortunately, due to the previously commented situation, a more advanced state of the project has not been yet able to achieve, although it is possible to recover from the drawback as far as they do not appear oftentimes.

In the second phase it is planned to build a Scenario generator with random initial positions for the nodes, as in the First phase, with a **varying** restricted mobility through time, by executing a certain amount of iterations, which, at the end of each one, one characteristic of each iteration Scenario will be calculated and evaluated in order to vary this movement restriction, to achieve a better nodes location through the map that will end up improving the Scenario characteristic value. With this, we will be able to hypothetically get a Scenario built with the required characteristic. In this Second phase, the characteristic will keep being the Density of the nodes, now with a better calculation algorithm extracted from a page in the *rosettacode* website [2].

5 Development

5.1 First phase

First of all, we created the concept of the structure that will hold the data of the whole *Scenario*, which consisted of a matrix of SxNx3, being N the number of nodes participating and S the number of *scenes* that our *Scenario* consists of. The last dimension refers to a triple of coordinates+time. The coordinates are 2D, so we see them as X and Y. The last member of the triple is the time. Time refers to the time associated to the scene in which the node has the concrete coordinates.

In this structure, every row corresponds to a node and the different columns refers to each *scene*. As an example we show a case in which we have 5 nodes and 3 *scenes*:

As we can see in Figure 4, each variable has different colours, implying that when colour changes, the value of the variable may have changed. On the one hand, as we expect at every column, time has the same color(value) because they correspond to the same *scene*. On the other hand, we can see that the tuple of coordinates change of colour at every column, as the nodes may have moved from one *scene* to another. Also between rows, colour of the tuple of coordinates changes as two nodes may not be in the same place.

However, throughout the execution of the project, the need of adding more elements to the last dimension had emerged. For example, in order to calculate the density of the Scenario, it was needed to take into account the **range**

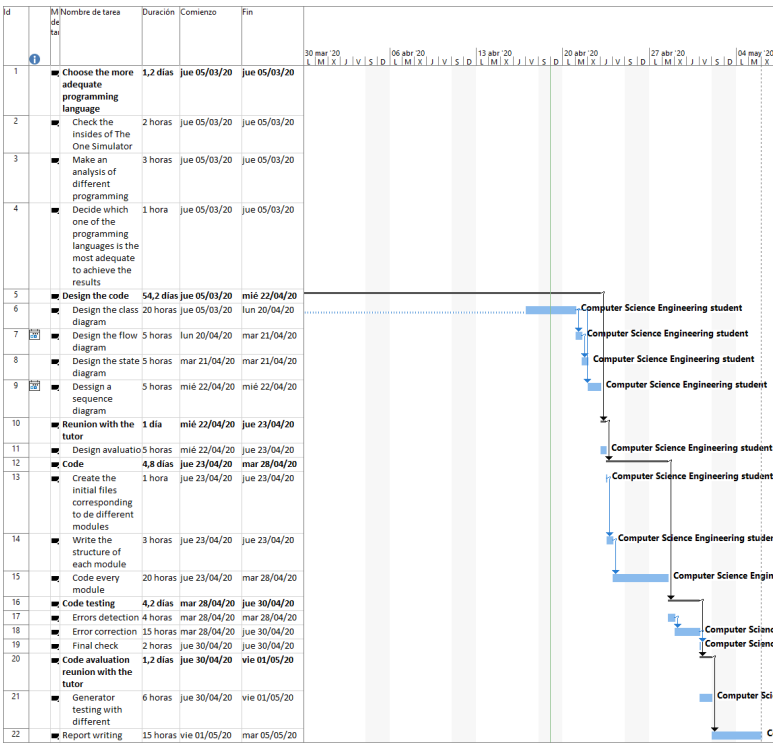

Fig. 3: Current Gantt diagram of the project.

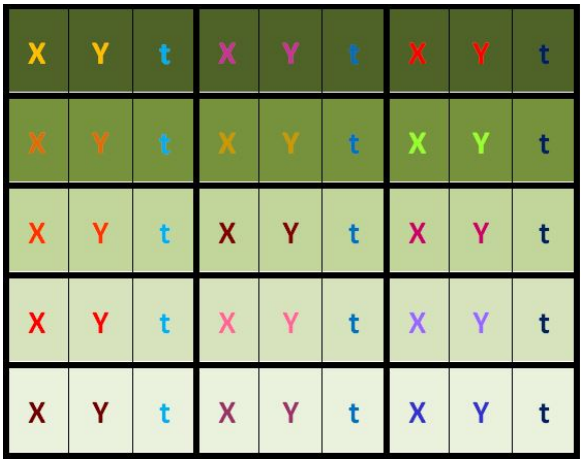

Fig. 4: Matrix of 3x5x3.

of communications of each node, data that had had a fixed value assigned for all of them in order to simplify it for testing.

Furthermore, we wanted to add another specification to each node in order to take the Scenario closer to the reality, which is the speed of every node. For this specification, we planned to assign three differents speeds that will correspond to the node mobility method. These specific methods are, nodes moving from one point to the other on foot, nodes moving from one point to the other by car and nodes moving from one point to the other running. Thus we are specifying a probability ratio for the nodes to be using each method, so that at the time of creating the first scene of every iteration Scenario, it will relatively randomly be assigned a speed to each node corresponding to the method.

Because of these new specifications, the matrix that will hold the Scenario would have the appearance shown in Figure 5:

Second of all we designed a preliminary class diagram updated with the new characteristics.

As we can see later in Figure 6, we have as our main class *Scenario*, which runs the task of nexus of the majority of the classes. It has the Matrix object as an attribute and builds up the *Scenario* with *scenes*. Once the *Scenario* has been built, it has connections to a path of classes that will apply the simulated annealing to the built *Scenario*.

However, throughout the execution of the project, it has been seen that the interface should be a class that unifies the simulated annealing method classes, so that each iteration of simulated annealing would have its own Scenario. The class may be named SimulatedAnnealing and it works as the "user-interface" class.

Afterwards we first designed a preliminary sequence diagram that tried to show the process of the Scenario creation and the later annealing. Nonetheless, the code differ from the diagram, but it still represents the basic idea of how the program was going to initially work.

5.2 Second phase

For the second phase of the project, as we said previously, the main class has become the Simulated Annealing one, making it the new user-interface class, although it is still possible to access directly to the Scenario class to build a Scenario without requiring any characteristic, as this parameter has been deleted from the Scenario class because its usage is implemented in the Simulated Annealing class.

Regardless what was previously said of adding the speed specification for the nodes as an add-up, it has not been implemented as was qualified as not needed for the project. As a consequence, the column in the matrix referred to the speed will not be added and the resultant matrix is finally as

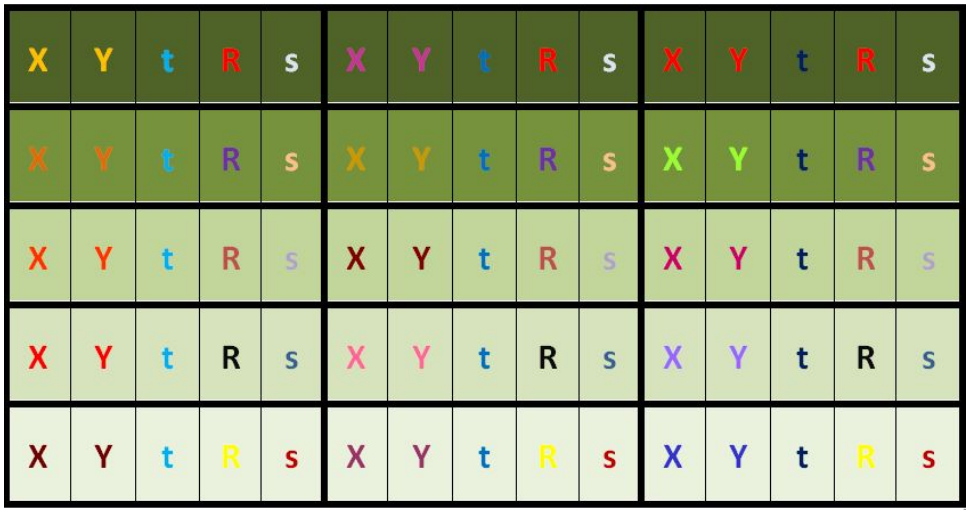

Fig. 5: Matrix of 3x5x5.

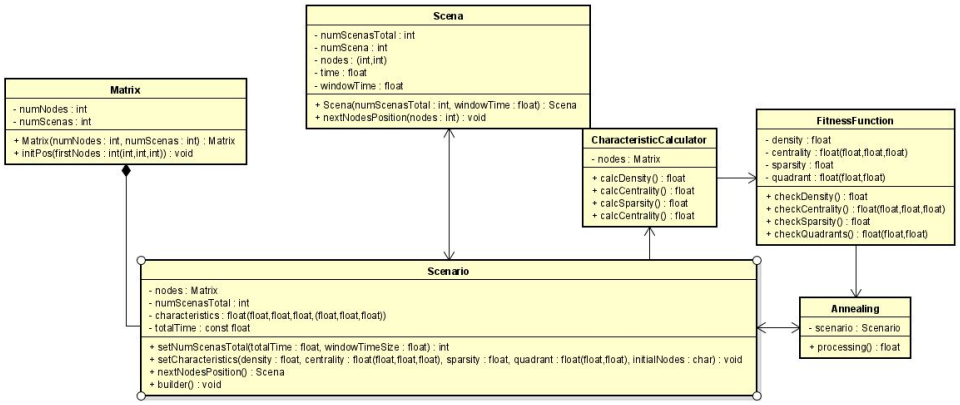

Fig. 6: Class diagram.

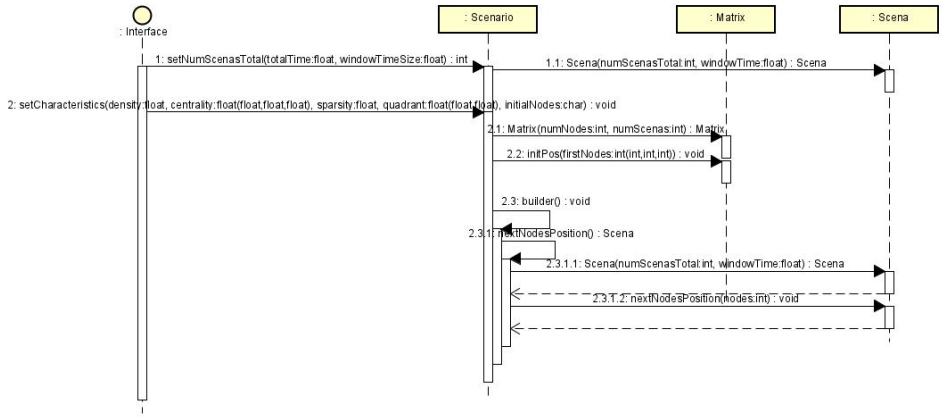

Fig. 7: Sequence diagram.

shown in Figure 8.

In this new class, the parameter characteristics will be used as the required characteristic. Although the format of the parameter is a list of at least two values, it will only use the first one, as the required value for the density. As it was an intermediate test phase, it was not needed to make it more easily scalable to add more characteristics, but it could be done by simply adding new methods to calculate the required characteristic and a line which would call it at every iteration.

The way it works is assigning a weight to “each characteristic”, in this case a weight of 1 for the density (as it is the only processed characteristic) to ponder the contributions of the different characteristics. Besides this weight, we calculate another weight, which is particulr for every characteristic. This one relates to a value by what will be multiplied the moveRestriction applied to the next iteration.

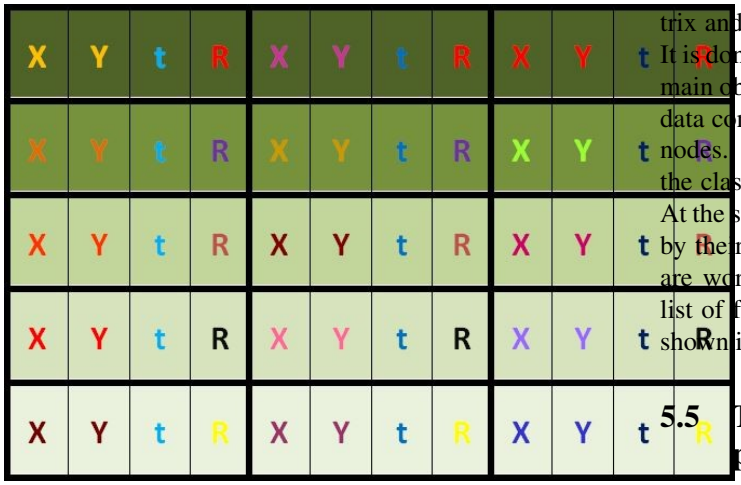

Fig. 8: Final Matrix scheme.

5.3 Third phase

In this final phase we have implemented a scalable characteristics system that allows to add as many different required characteristics as wanted, as far as the parameter is included as an object of a child class of the Characteristic class, which will be added in the Annex of this report. Each characteristic must have its own calculating method, which accepts a Scenario object as a parameter to calculate its characteristic value. The object gets the required value as a parameter when it is created.

Moreover, the "Temperature" parameter has finally been implemented in the SimulatedAnnealing class, becoming more close to the real simulated annealing concept. The temperature is set to a higher value when the obtained characteristic from the recently build Scenario is far beyond the required value, either lower or higher. When the temperature is set to 'success' means that all the obtained values of the characteristics are within the previoulsy set tolerance value. Temperature is multiplied by the movementRestriction of the previous iteration. In this third phase we decided to start with a value of moveRestriction matching the value of the shortest side of the Scenario map, to help the simulated annealing algorithm to converge by increasing its value.

We have tested the project with two characteristics, density and sparsity, classes that are included within the files of the project. Beside them, we have created a program as a more "user-friendly" UI for testing, which runs different Scenario cases tests to check the results of the project. This program is used to test the SimulatedAnnealing with the density and sparsity required values specifically, although the SimulatedAnnealing class is capable to handle more different characteristics.

5.4 The structure

The main class is SimulatedAnnealing, which receives the list of required characteristics and the rest of Scenario parameters and handle to run different iterations until the requiresd Scenario is built or until the 100 maximum iterations are achieved.

The main class, after the SimulatedAnnealing one is the Scenario class, in which converges classes Scene and Matrix and carries out the task of generating a Scenario itself. It is done with the key help of the Matrix class, of which the main object of the Scenario class is made out. It is the main data container as it holds the previous mentioned matrix of nodes. This Matrix object will be made up of elements of the class Scene, as much as scenes the Scenario will have. At the same time, the Scene object is made up of nodes that, by their side, they are made up of namedtuple lists, as we are working with Python. Thus the nodes are finally the list of four elements that were described antecedently and shown in Figure 8.

5.5 The usage of the Scenario generator at phase 1

Once reached this point is important to be in knowledge of how to generate our Scenario with the given prototype code. Thus the usage of the first prototype will be described as follows:

- First of all we have to open our command table and go to the project directory
- Second of all we open the python environment by executing the command `Python`
- Afterwards we have to import the main class Scenario by executing the command `import Scenario`
- Finally we can execute the program by calling the Scenario constructor and specifying our requirements for the Scenario in the proper order, namely: numNodes as the number of nodes we want the Scenario to have, characteristics as a list of the characteristics that our Scenario should accomplish (although this part is not implemented yet, it is important to write any value in, preferably a list or array), totalTime as the time of duration of the whole Scenario, dimensions as a tuple of Scenario dimensions i.e [100, 100], comRange = 10 as a range of communication of each node (as you can see it is established to 10 by default so it is not a required parameter to send), windowTimeSize = 1 as the time within scenes or duration of each scene and, as it is shown it is established to 1 by default, so there is no need to be sent if do not want to change it. This call can be made as the one that follows as an example: `Scenario.Scenario(10, [5], 5, [50, 50])` As it is shown, the unnecessary parameter fields are not filled and the default value will be used.

By the execution of this command, the result will be the Scenario printed and the mean density specified, as it is seeable in Figure 9 and 10

5.6 The usage of the Scenario generator at phase 2

In this improved prototype, the user-interface (UI) is the *SimulatedAnnealing* class, which establishes the parameters of the Scenario through the ones specified in the time of its creation, and afterwards executes up to 20 iterations of Scenarios, trying to transform the Scenario in order to achieve

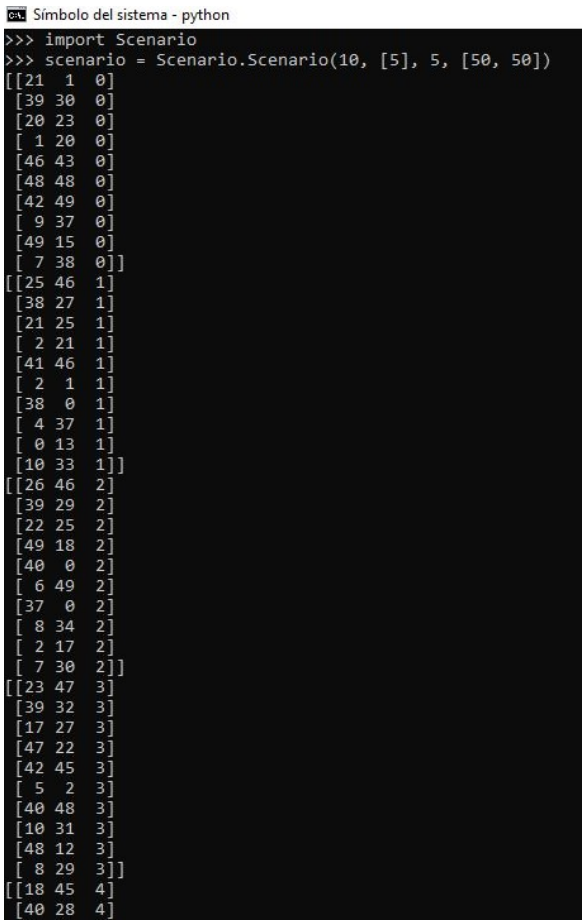

Fig. 9: First part of the result of the execution of the program in phase 1.

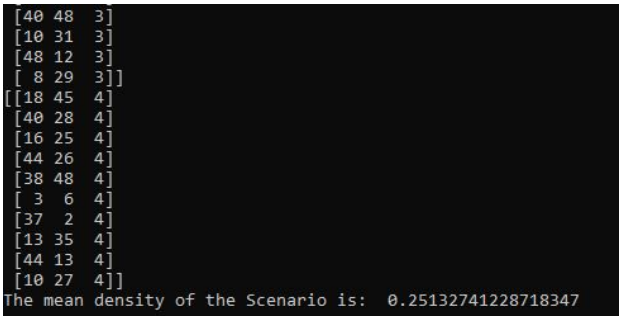

Fig. 10: Second part of the result of the execution of the program in phase 1.

the characteristics to be required through its parameters for the Scenario.

In pursuance for obtaining a Scenario with the required characteristics, the *SimulatedAnnealing* class will execute the simulated annealing method varying the movement of the nodes within scenes until it obtains a Scenario that fits the required characteristics (satisfying a given tolerance of 0.1 of difference between the obtained characteristics and he required ones) or until it reaches 100 iterations, in other words, 20 tries made up of Scenarios.

Thus this is the first contact with the final result, although still not complete. So that we can use it, we need to be in touch with its usage tips. These are described as as follows:

- First of all we have to open our command table and go to the project directory
- Second of all we open the python environment by executing the command `Python`
- Afterwards we have to import the main class SimulatedAnnealing by executing the command `import SimulatedAnnealing as sa`
- Finally we can execute the program by calling the SimulatedAnnealing constructor and specifying our requirements for the Scenario in the proper order, namely: characteristics as a list of the required characteristics the Scenario should accomplish, in a specified order, which in this phase will be as the first one the required density and as the second one, a random number which, in a future would be another characteristic, but in this testing phase it was done this way to make it improvable but not yet functional. Then, numNodes as the number of nodes we want the Scenario to have, totalTime as the time of duration of the whole Scenario, dimensions as a tuple of Scenario dimensions i.e [100, 100], comRange = 10 as a range of communication of each node (as you can see it is established to 10 by default so it is not a required parameter to send), windowTimeSize = 1 as the time within scenes or duration of each scene and, as it is shown it is established to 1 by default, so there is no need to be sent if do not want to change it. This call can be made as the one that follows as an example: `sa.SimulatedAnnealing( (0.7, 0.8), 10, 5, [50, 50]).`

  As it is shown, the unnecessary parameter fields are not filled and the default value will be used and the required density value is 0.7, and the next value (0.8) will not be used.

By the execution of this command, the result is besides some state-checking data for each Scenario created (iteration), a final Scenario which is printed at the end of the execution. I can be seen in the following Figures 11 12:

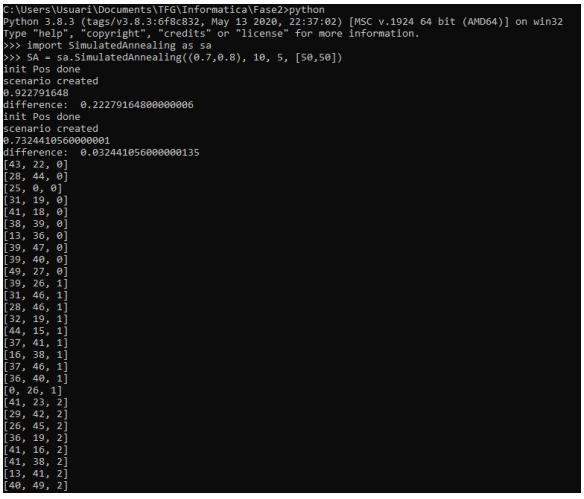

Fig. 11: First part of the result of the execution of the program in phase 2.

```
[40, 49, 2]
[33, 36, 2]
[47, 28, 2]
[40, 21, 3]
[25, 42, 3]
[29, 49, 3]
[39, 21, 3]
[45, 12, 3]
[38, 40, 3]
[12, 45, 3]
[43, 45, 3]
[30, 33, 3]
[45, 29, 3]
[44, 24, 4]
[28, 46, 4]
[26, 3, 4]
[35, 24, 4]
[41, 9, 4]
[41, 44, 4]
[12, 49, 4]
[43, 43, 4]
[33, 30, 4]
[41, 28, 4]
>>>
```

Fig. 12: Second part of the result of the execution of the program in phase 2.

5.7 The usage of the Scenario generator at phase 3

In this phase, despite the previous ones, it is important that the required characteristics are objects of the concrete required characteristic class, so it is important to have in mind to import these classes in order to use them.

In our case, we have used a separate program, by creating a new class file that will handle with this easier. The results of the execution of this program are included in another section, so they will not be shown in this one.

Here it will be explained the plain steps to create one Scenario with the SimulatedAnnealing class by the commmand table.

- First of all we have to open our command table and go to the project directory
- Second of all we open the python environment by executing the command `Python`
- Afterwards we have to import the main class SimulatedAnnealing by executing the command `import SimulatedAnnealing as sa`
- Besides it is important to import the characteristics classes, as an example, the Density and Sparsity classes, which are both child classes of the Characteristic class: `from Density import Density` and also `from Sparsity import Sparsity`
- Finally we are able to call the constructor of the SimulatedAnnealing class adding the parameters we want to send, as seen previously, but with the new characteristics list, which instead of raw numbers, they have to be objects of the concretes characteristics classes, which can be directly done by writing in the call of the characteristic class constructor with the required characteristic value as a parameter. As an example it could be: `sa.SimulatedAnnealing( [Density(0.4), Sparsity(0.1)], 50, 10, [100,100] )`. As said previously, the unnecessary parameter fields are not filled and the default value will be used. In this case, the required density value is 0.4 and the required sparsity value is 0.1

6 Results presentation and discussion

At the beginnings of the project, the density value was calculated without having into account the overlap between the ranges of communication of the different nodes, so it was not reliable data.

Nonetheless this issue has been solved at the beginning of the Second phase with the decision of using the *rosettacode* algorithm [2].

This algorithm calculates the area of the range of communication of all nodes, counting only once the overlapped areas, and selecting the scene with the higher area value, for later calculating the density with this value, by dividing it by the dimensions of the map.

$$DensityPerScene = \frac{CalculatedArea}{ScenarioArea}$$

$$MeanDensity = \frac{\sum_{n=1}^{TotalNumberOfScenes} DensityPerScene}{TotalNumberOfScenes}$$

At the end of the second phase, we have achieved the results shown in Figures 15, 16 of the Appendix. The parameters of the Scenario are shown in Figure 16. In Figure 13 we show an example of what is displayed in the Appendix.

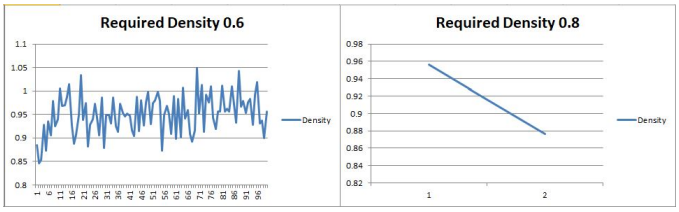

Fig. 13: Two charts of a set of testing in phase 2.

As we can see, the obtained density values suffered unsignificant changes within the different iterations, making the simulated annealing algorithm unable to converge in the majority of the cases.

From this we can extract that the determining factor in the values obtained for the density are the parameters of the Scenario.

At the Third phase, we included the sparsity calculations to request a sparsity value for the Scenario. Moreover, we increased the initial value for the movementRestriction of the nodes, until the maximum available for the map, in other words, matching the shortest side of the map which, in our case, is any of them as our map is squared. In addition

to this measure, we have have decided to take the maximum density value within scenes instead of the mean value within them. The purpose of these measures was to help the algorithm converge.

$$MaximumDensity = \frac{max(DensityPerScene)}{MapOfTheScenarioArea}$$

We have completed two tests for this final phase, with the results shown in Figures 17, 18, 19 and 20 of the Appendix. The Scenario parameters for each test are specified in Figures 18 and 20. In the Figure 14 we show an example of what is displayed in the Appendix:

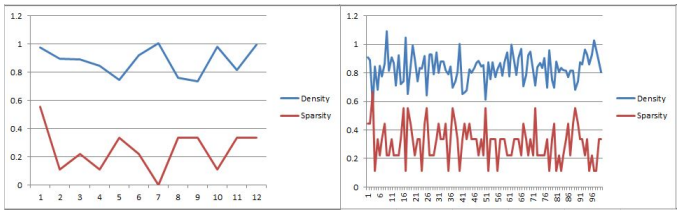

Fig. 14: Two charts of a set of testing in phase 3.

As we can see in Figure 17 although we are able to obtain a suitable Scenario in 3 cases out of 7, the variation of the characteristics is not as significant as expected, even with the measures taken, making it too reliant of the Scenario paremeters introduced.

Afterwards we tried with different parameters for the Scenario in order to see the behaviour of the algorithm with greater parameters. In this case, as we can see in Figure 19, we have obtained even worse results than before, with only one convergence. From this results we can extract that, again, the density values are very reliant of the Scenario paremeters and that the situation is worsened when these parameters increase their values, in other words, when the Scenario is bigger.

7 State of the art

Currently there is not a Scenario generator open code, nor any published one. Nevertheless the authors of routing protocols use their designed Scenarios for the testing.

However, it exists real-based Scenarios or generated Scenarios derived from real human mobility, such as CAHM [3] or Map-Based Movement [4]. To give some examples, we are going to describe seven of them, extracted from the paper *Fair Comparative Analysis of Opportunistic Routing Protocols: An Empirical Study* by Jay Gandhi and Zunnun Narmawala [5].

- CAHM: Community Aware Heterogeneous Human Mobility (CAHM) model creates the initial nodes location map by overlapping community structure and give the nodes a movement pattern based on human mobility characteristics taken from real-world mobility traces analysis and derived from human social behaviour [3].

- Map-based Movement: In this mobility model, the nodes location is based on a predefined real map, hence its particularity comes from the movement of the nodes. It consists of three movement models, which are Random Map-based Movement, Routed Map-based Movement, and Shortest Path Mapbased Movement [4].

- Infocom05: This is a real-world Scenario generated by 50 students attending to the 2005 Infocom conference, from Infocom student workshop, who were given Bluetooth devices and carried them for 4 days in the conference. Every 2 minutes, each Bluetooth device (iMotes) performed a neighbourhood scan [6].

- Infocom06: It is the same experiment as Infocom05 but on a larger scale as the iMote was carried by 80 students this time, for 5 days in the Infocom 2006 conference. On this occasion the conference area included three floors, and 34 out of the 80 participants were divided in four groups based on their academic affiliation [6].

- Reality: In this experiment, carried out at MIT, 100 smartphones were given to students and staff for 9 months, collecting approximately 5,000,000 h of data on the user's communication, location and device usage. The smartphones' Bluetooth was enabled and performed device discovery every 5 minutes [7].

- Cambridge: In this experiment the participants were 70 students and researchers as well as every other Bluetooth enabled device carrier, as it was possible to record a Bluetooth contact with externals devices, calling it an external contact. Hence, it existed two types of contacts, internal contact, which was referred to those Bluetooths conctacts between two iMotes (the device the participants were carrying with them) and the external contact already explained. The experiment took 11 days of collecting information [6].

- Sassy: In this experiments there were 27 participants who were staff members of St. Andrews University. They were given T-mote devices. The participants carried the devices whenever possible for 79 days, with the device detecting others in a range of approximately 10 meters. To collect information from the devices, every encounter events were stored for being uploaded to a central database via base stations [8].

8 Conclusions

In the First phase of the Scenario generator project we achieved the prototype of the Scenario generator that allows to build a Scenario with initially randomly placed nodes and random restricted movement (by default, restricted to 5 positions) for the nodes, as well as calculate the mean Density of the nodes in the Scenario.

However, this density is not fully reliable as commented previously, but this issue had been fixed, by using the *rosettacode* algorithm [2], avoiding the overlapping. In addition, in order to help the algorithm converge, we have changed the characteristics calculations by taking always the maximum value within scenes instead of calculating a mean value, which would normalize the value, decreasing the impact of the temperature on it.

At the second phase of the project we did still use the mean value of the density, although with the new algorithm,

as we were not able to check its lack of convergence potential with the mean values, as it was one of the first testing cases of the SimulatedAnnealing class.

From this testing case we can conclude that the major influence for the density value obtained from the generator program comes mainly from the Scenario parameters, such as the number of nodes (*numNodes*) of the Scenario, which in the case of the density makes sense, because it is closely related to the density, as, by definition, the density is the division of the number of nodes (each node with its range of communications) per the area covered by the Scenario map, which comes along with another Scenario parameter, the Scenario dimensions. As well as the number of nodes, the parameter *dimensions* has also a too big impact in the density of the Scenario, hindering the contribution of the moveRestriction to be determining in the final density value.

In this case, the number of scenes, directly related to the parameters *totalTime* and windowTimeSize, that are key to ots calculation, also has some impact to the density calculation, because the higher the number of scenes, the higher the normalization due to the mean value from the density calculations.

This last issue could be fixed by using the maximum value within scenes for the characteristics calculations, but the other issues could not be fixed this way. So we decided to be more strict with the weights of the characteristics and set the initial *moveRestriction* value to the shortest Scenario map side, in other words, increasing the randomness, to exagerate the initial temperature.

Therefore for the last phase we obtained better results with low values for the Scenario parameters, showing that the improvements worked.

Nevertheless, once we increase the value of the Scenario parameters, the results worsened again, making evident that the dependence to the Scenario parameters was strong yet.

We can conclude that the simulated annealing method by itself is not capable of obtaining a Scenario fitting some given required characteristics in any case, because they highly depend on the Scenario parameters.

As a hypothesis, it seems that this method, in combination with others could manage to obtain the required Scenario in any condition, in case of setting some indicative values for the Scenario parameters instead of just setting the exact value, letting some range of action for the algorithm to work with.

Independently from the reults of the project, we are open to add new features such as other modes of initially fill the Scenario with nodes, for instance, by taking the initial nodes position from a file, and including movement patterns to the nodes, as well as movement speed.

ACKNOWLEDGMENTS

In first place I want to thank the project tutor Diego Freire for being always in touch with me while elaborating the project and for all the help that handed to me, specially in the moments where the situation was stucked and the advance of the project was not possible. He helped to set an initial path to the project in order to being able to begin developping the project from an initial point.

In second place I want to thank Sergi Robles, who is the responsible for the investigation and is in charge of the investigation team. He initially helped me to understand what was required from me for the project and explained it in a precise and clear way, what helped me a lot in order to understand the task and consequently, develop the project. Not only he first explained me the project requirements, but also provided us from a new project methodology, which helped a lot at the time of advancing with the project, as it simplified the load of work in time by spreading the burden of work over time and letting us have different prototypes to evaluate better the job carried out through time.

Finally I want to thank all of my information sources that helped me to develop the project, in one way or another, either by providing of knowledge of the art or either by providing of a direct response to a project requirement.

REFERENCES

[1] N. Kaur and G. Mathur, "Opportunistic networks: A review," *IOSR Journal of Computer Engineering (IOSR-JCE)*, vol. 18, no. 2, pp. 20–26, Mar. 2016.

[2] Rosettacode community. (2020, Mar.) Total circles area, grid sampling version (python). [Online]. Available: https://rosettacode.org/wiki/Total_circles_areaPython

[3] Z. Narmawala and S. Srivastava, "Community aware heterogeneous human mobility (cahm): Model and analysis," *Pervasive and Mobile Computing*, vol. 21, pp. 119–132, 2015.

[4] A. Keränen, J. Ott, and T. Kärkkäinen, "The one simulator for dtn protocol evaluation," in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, ser. Simutools '09. Brussels, BEL: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009. [Online]. Available: https://doi.org/10.4108/ICST.SIMUTOOLS2009.5674

[5] J. Gandhi and Z. Narmawala, "Fair comparative analysis of opportunistic routing protocols: An empirical study," in *Data Communication and Networks*, L. C. Jain, G. A. Tsihrintzis, V. E. Balas, and D. K. Sharma, Eds. Singapore: Springer Singapore, 2020, pp. 285–294.

[6] J. Scott, R. Gass, J. Crowcroft, P. Hui, C. Diot, and A. Chaintreau, "Crawdad dataset cambridge/haggle (v. 2006-09-15)," *CRAWDAD wireless network data archive*, 2006.

[7] N. Eagle and A. S. Pentland, "Reality mining: sensing complex social systems," *Personal and ubiquitous computing*, vol. 10, no. 4, pp. 255–268, 2006.

[8] G. Bigwood, D. Rehunathan, M. Bateman, T. Henderson, and S. Bhatti, "Crawdad dataset st_andrews/sassy," *Retrieved from http:/crawdad. cs. dartmouth. edu/st_andrews/sassy*, 2011.

APPENDIX

A.1 Results of testing in phase 2

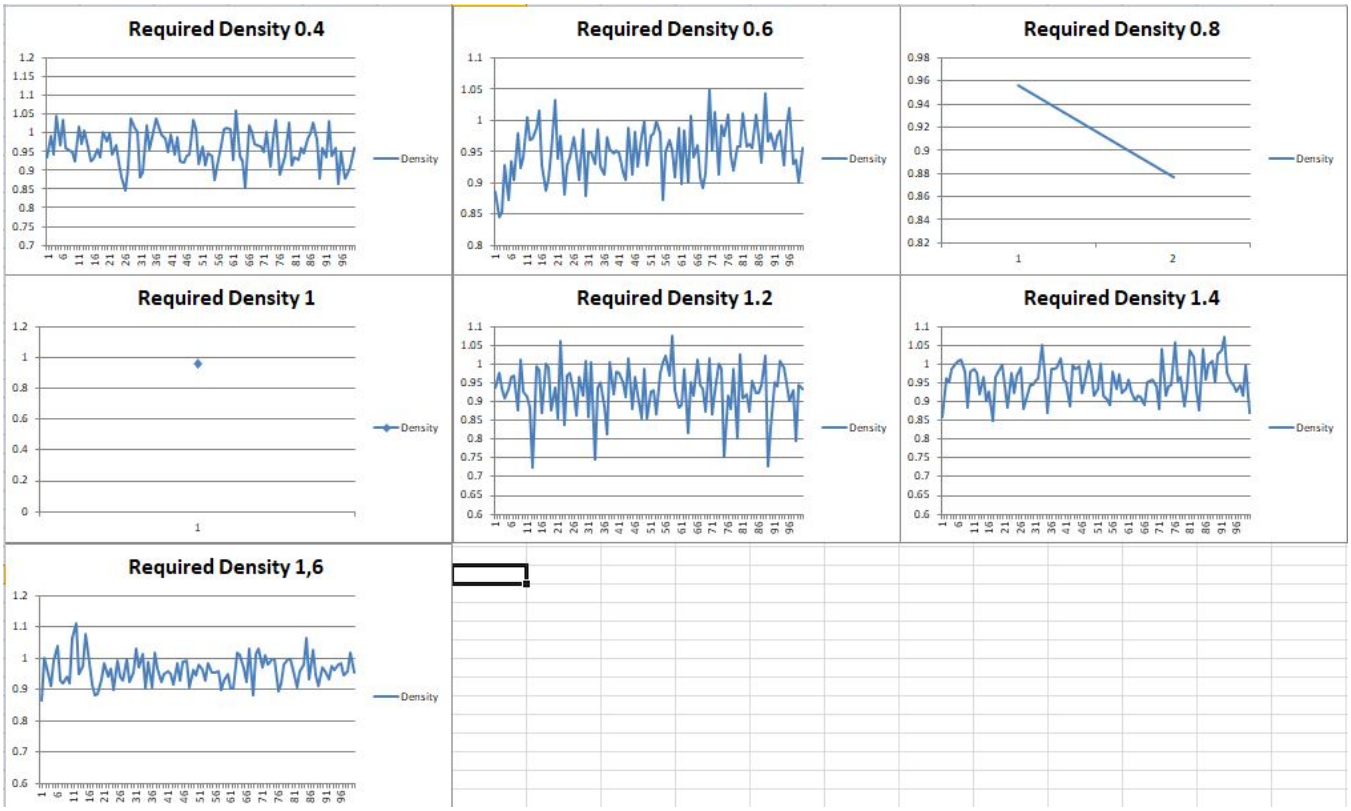

Fig. 15: Track of the density value variation between iterations for each density requirement in phase 2.

```
For density 0.400000 required we have obtained a Scenario with:
 density: 0.958647
 success: FAILED
 With the following parameters for our Scenario:
 Number of nodes: 10
 Total time: 10
 Dimensions: [50, 50]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 0.600000 required we have obtained a Scenario with:
 density: 0.955638
 success: FAILED
 With the following parameters for our Scenario:
 Number of nodes: 10
 Total time: 10
 Dimensions: [50, 50]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 0.800000 required we have obtained a Scenario with:
 density: 0.876682
 success: ACHIEVED
 With the following parameters for our Scenario:
 Number of nodes: 10
 Total time: 10
 Dimensions: [50, 50]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 1.000000 required we have obtained a Scenario with:
 density: 0.953825
 success: ACHIEVED
 With the following parameters for our Scenario:
 Number of nodes: 10
 Total time: 10
 Dimensions: [50, 50]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 1.200000 required we have obtained a Scenario with:
 density: 0.932938
 success: FAILED
 With the following parameters for our Scenario:
 Number of nodes: 10
 Total time: 10
 Dimensions: [50, 50]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 1.400000 required we have obtained a Scenario with:
 density: 0.870732
 success: FAILED
 With the following parameters for our Scenario:
 Number of nodes: 10
 Total time: 10
 Dimensions: [50, 50]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 1.600000 required we have obtained a Scenario with:
 density: 0.952842
 success: FAILED
 With the following parameters for our Scenario:
 Number of nodes: 10
 Total time: 10
 Dimensions: [50, 50]
 Range of communications: 10
 Windowtime size: 1
 ----------------
```

Fig. 16: Results of the phase 2 test with density only requirement.

A.2 Results of testing in phase 3

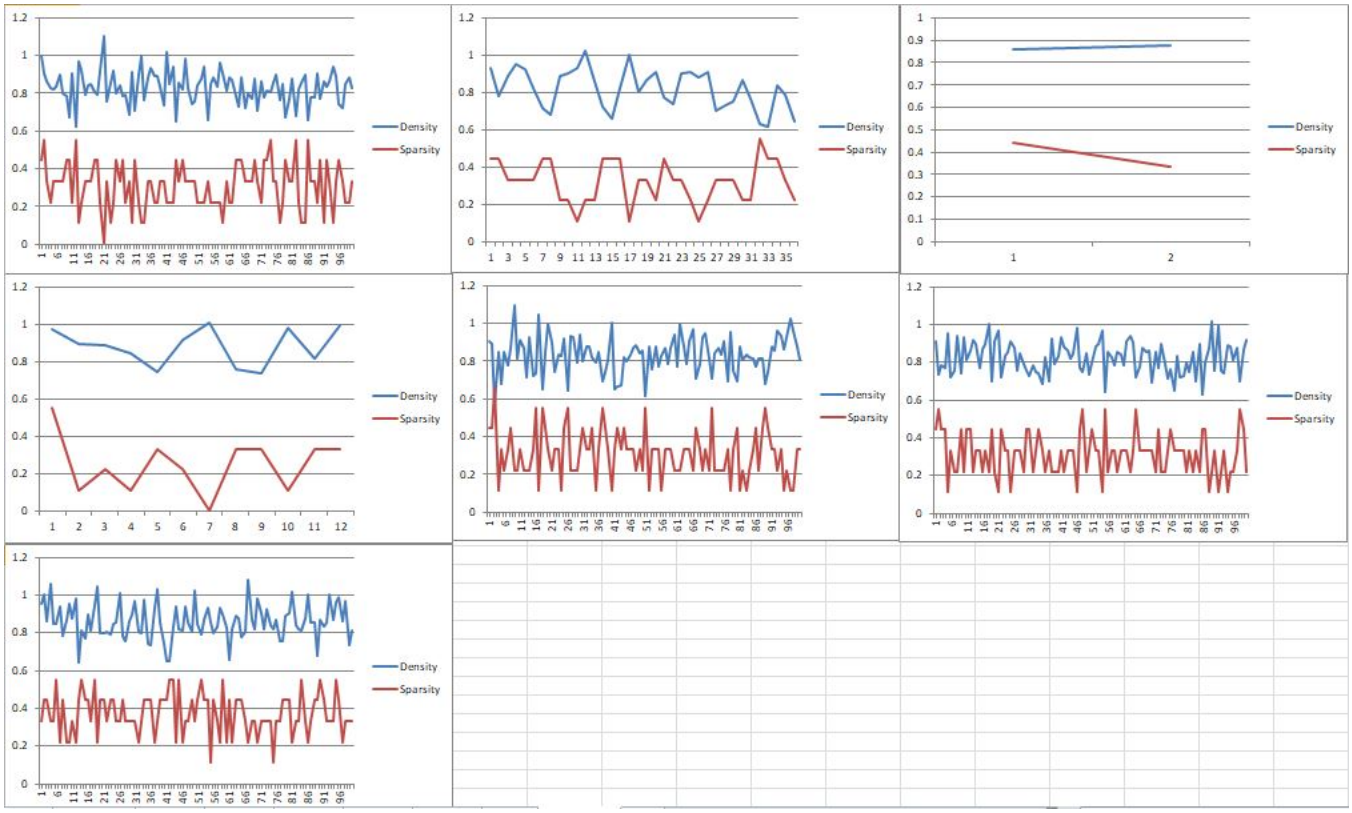

Fig. 17: Track of the density and sparsity values variation between iterations for each density requirement in phase 3 for the first test.

```
For density 0.400000 required and sparsity 0.100000 required we have obtained a Scenario with:
 density: 0.829162
 sparsity: 0.333333
 success: FAILED
 With the following parameters for our Scenario:
 Number of nodes: 10
 Total time: 10
 Dimensions: [50, 50]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 0.600000 required and sparsity 0.200000 required we have obtained a Scenario with:
 density: 0.644106
 sparsity: 0.222222
 success: ACHIEVED
 With the following parameters for our Scenario:
 Number of nodes: 10
 Total time: 10
 Dimensions: [50, 50]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 0.800000 required and sparsity 0.300000 required we have obtained a Scenario with:
 density: 0.876234
 sparsity: 0.333333
 success: ACHIEVED
 With the following parameters for our Scenario:
 Number of nodes: 10
 Total time: 10
 Dimensions: [50, 50]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 1.000000 required and sparsity 0.400000 required we have obtained a Scenario with:
 density: 0.995072
 sparsity: 0.333333
 success: ACHIEVED
 With the following parameters for our Scenario:
 Number of nodes: 10
 Total time: 10
 Dimensions: [50, 50]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 1.200000 required and sparsity 0.500000 required we have obtained a Scenario with:
 density: 0.803820
 sparsity: 0.333333
 success: FAILED
 With the following parameters for our Scenario:
 Number of nodes: 10
 Total time: 10
 Dimensions: [50, 50]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 1.400000 required and sparsity 0.600000 required we have obtained a Scenario with:
 density: 0.920621
 sparsity: 0.222222
 success: FAILED
 With the following parameters for our Scenario:
 Number of nodes: 10
 Total time: 10
 Dimensions: [50, 50]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 1.600000 required and sparsity 0.700000 required we have obtained a Scenario with:
 density: 0.810762
 sparsity: 0.333333
 success: FAILED
 With the following parameters for our Scenario:
 Number of nodes: 10
 Total time: 10
 Dimensions: [50, 50]
 Range of communications: 10
 Windowtime size: 1
 ----------------
```

Fig. 18: Results of the first test of phase 3.

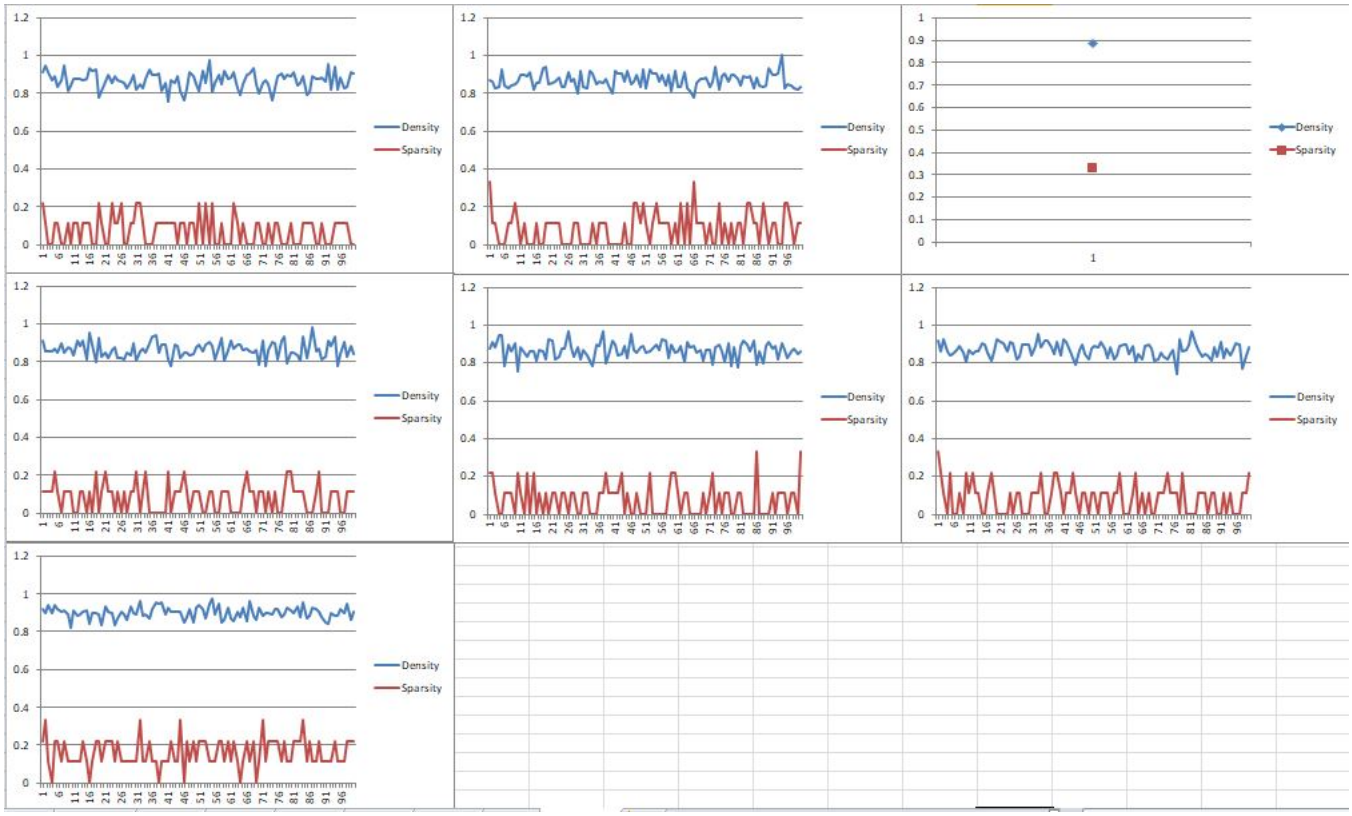

Fig. 19: Track of the density and sparsity values variation between iterations for each density requirement in phase 3 for the second test.

```
For density 0.400000 required and sparsity 0.100000 required we have obtained a Scenario with:
 density: 0.903355
 sparsity: 0.000000
 success: FAILED
 With the following parameters for our Scenario:
 Number of nodes: 50
 Total time: 10
 Dimensions: [100, 100]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 0.600000 required and sparsity 0.200000 required we have obtained a Scenario with:
 density: 0.836983
 sparsity: 0.111111
 success: FAILED
 With the following parameters for our Scenario:
 Number of nodes: 50
 Total time: 10
 Dimensions: [100, 100]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 0.800000 required and sparsity 0.300000 required we have obtained a Scenario with:
 density: 0.884274
 sparsity: 0.333333
 success: ACHIEVED
 With the following parameters for our Scenario:
 Number of nodes: 50
 Total time: 10
 Dimensions: [100, 100]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 1.000000 required and sparsity 0.400000 required we have obtained a Scenario with:
 density: 0.840243
 sparsity: 0.111111
 success: FAILED
 With the following parameters for our Scenario:
 Number of nodes: 50
 Total time: 10
 Dimensions: [100, 100]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 1.200000 required and sparsity 0.500000 required we have obtained a Scenario with:
```

```
 ----------------
For density 1.200000 required and sparsity 0.500000 required we have obtained a
Scenario with:
 density: 0.865850
 sparsity: 0.333333
 success: FAILED
 With the following parameters for our Scenario:
 Number of nodes: 50
 Total time: 10
 Dimensions: [100, 100]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 1.400000 required and sparsity 0.600000 required we have obtained a
Scenario with:
 density: 0.885181
 sparsity: 0.222222
 success: FAILED
 With the following parameters for our Scenario:
 Number of nodes: 50
 Total time: 10
 Dimensions: [100, 100]
 Range of communications: 10
 Windowtime size: 1
 ----------------
For density 1.600000 required and sparsity 0.700000 required we have obtained a
Scenario with:
 density: 0.904305
 sparsity: 0.222222
 success: FAILED
 With the following parameters for our Scenario:
 Number of nodes: 50
 Total time: 10
 Dimensions: [100, 100]
 Range of communications: 10
 Windowtime size: 1
 ----------------
```

Fig. 20: Results of the second test of phase 3.