Model testing and running

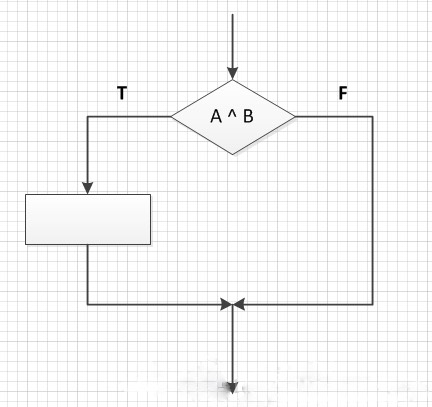
Introduction

This section is the run and test section of the model. The main objective is to run and test the built model to check if the pre-designed functionality is implemented and to adjust the variables to run the model multiple times to ensure the accuracy, stability and scalability of the model. This is in addition to verifying the proposed hypothesis that different levels of protection can affect the data sharing and collaboration process while protecting the data source. By running the model multiple times many sample data can be obtained, which can be reasoned and analyzed and the results obtained can be used to validate the hypothesis. Finally, the results of the runs and data analysis are used to determine whether the model can be presented as the final outcome of the project.

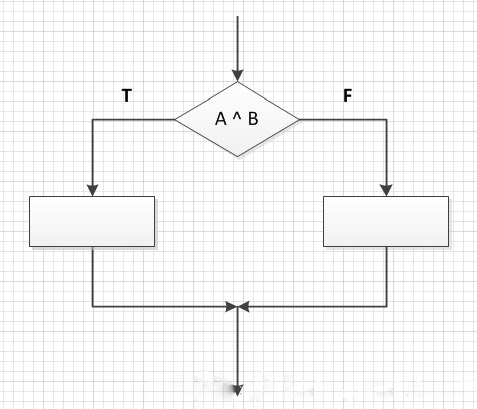
5.1unit test

This section is a unit test, the main purpose of which is to test the model against the various components such as code, information labels, sliders, buttons, etc. to ensure that there are no errors or defects and that the model runs smoothly.

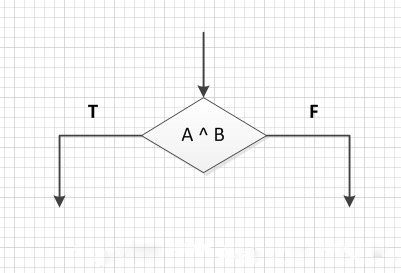
The first part is the testing of code modules, this part uses the conventional code testing method, i.e. white box testing, to divide the code by functional modules and test them one by one to ensure that they can run independently. Although NetLogo's language style and statements are different from those of a normal object-oriented programming language, the basic data structures such as conditions, judgments, loops, arrays, lists, etc. still exist, so they conform to the testing methods of a normal programming language. The methods used in this test include statement override, which creates test cases so that each executable statement in the program can be executed once.



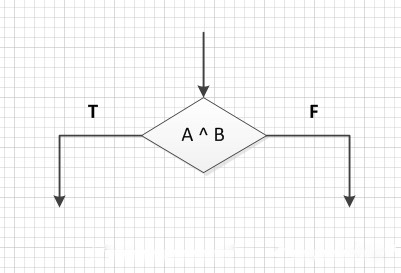
Judgment coverage method, that is, for judgment statements, the use cases are designed so that the result of the judgment statement is both True and False.



Conditional override, that is, each conditional expression true and false should be taken once when designing the use case without considering the calculation result of the judgment statement.



Judgment condition override, that is, when designing test cases, make all possible results of each conditional expression in the judgment statement appear at least once, and all possible results of the judgment statement itself appear at least once.



As a result of these tests, none of the code modules showed any functional problems. Therefore, we believe that the code part of the model is capable of implementing the pre-designed functions. The next step is to test the sliders, buttons, switches, etc. of the interactive interface. The unit test phase is to test the interactive functionality of these interactive tools, the functionality will be tested in the next phase. After confirming that the slider can be slid freely, the button can be clicked and the switch can be turned off, the next phase of testing is carried out.

5.2 functional test

This section is the functional testing section, which focuses on the basic functionality of the model, the functionality of the interactive components of the user interface. The established functions of the model are: setting up different kinds of nodes (data sources, common agents, spy nodes), setting up data types and quantities, building data sharing networks, simulating data collaboration, and applying restrictive protection measures to data sources. This part of the test is mainly through repeatedly running the model to test, because the construction of the data sharing network is random, each node every time to obtain the data is also random, so through a lot of repeated runs of the model can verify the model to see if there are functional omissions and defects. As for the interactive components, since the interactive components in NetLogo need to be bound to the code, the testing of the interactive components, such as the step size of the slider, whether the button works, etc., is carried out in the same way as the unit testing in the previous section, i.e., syntax checking and running tests on the code. After the tests, the model completes its intended functionality and the interactive component is ready to be used.

In addition, since we have added some variables (restrictions, protections) to the model during the model implementation phase, we have also tested the functionality and interaction of these variables to ensure that the model can be further investigated based on the original functionality. Finally, the line graphs were also tested, as the data from the line graphs are needed for the later validation of the hypothesis, so testing the functionality of the line graphs can ensure the accuracy of the data and make the conclusions obtained from the analysis more realistic. The test method is the same as unit testing, because the line graph is also bound to the code, as the program runs in real time changes.

5.3 Integration Test

This section is integration testing, which, as the name implies, is the complete testing of the entire model at the system level to ensure that the model is running as expected with accurate results and data that can be used for inferential analysis. The 'setup' button is first tested to see if it can initialize the model. Since the model is dynamic, i.e. it changes over time, we need to initialize the model at the end of the run to make it easier for the next run. The 'go' button is then tested, as it is divided into single and repeated executions (until the program runs to a pre-set end condition).



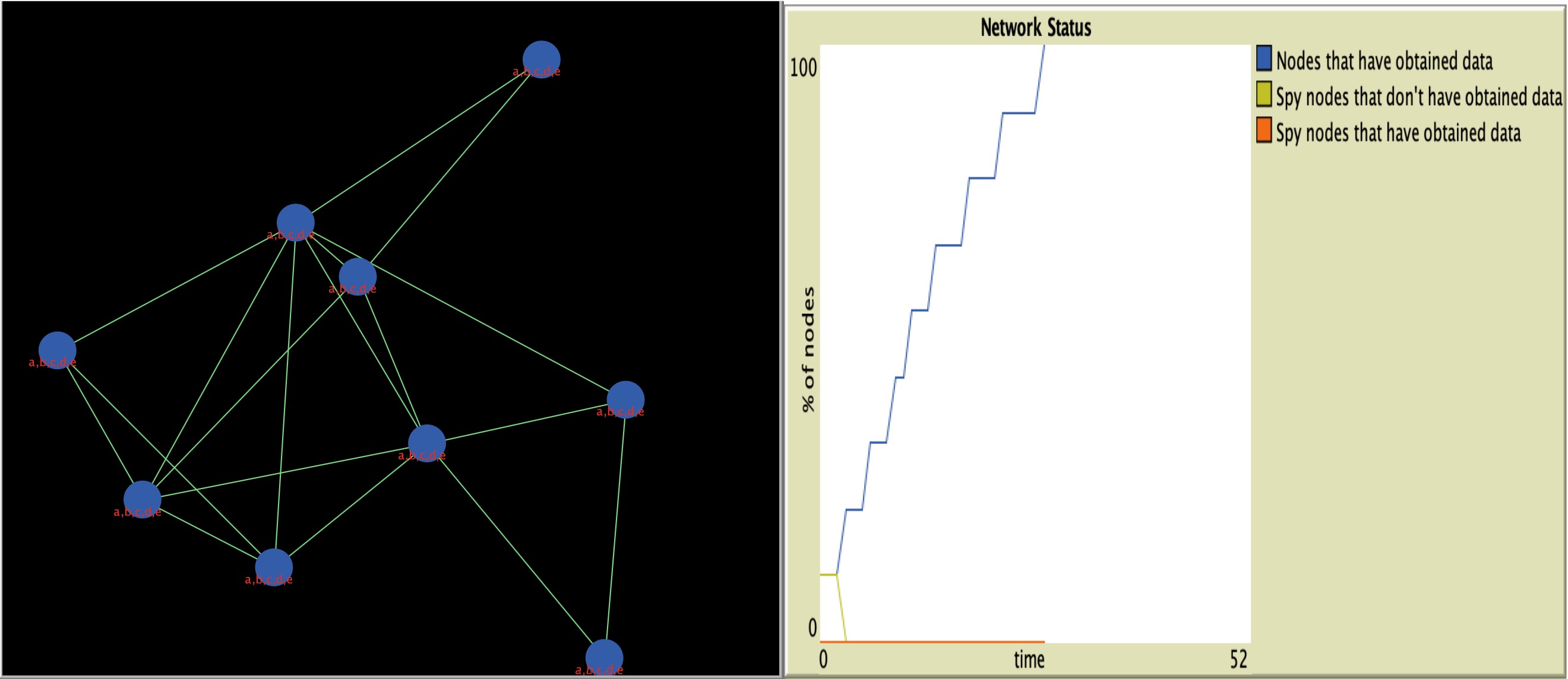
Normally we would set the 'go' button to repeat when building a model, as this gives us a quicker view of the model and a clear trend of the variables over time on a line graph. But if you want to see the model changes frame by frame or want to record the run data, you need to select the 'go' button for a single run. This test phase therefore tests both cases and ensures that both buttons are available.

Next the model was run multiple times, as the networks and nodes generated by the model are random each time, so the purpose of this step was to check if there were any extremes or conditions that were not taken into account during the programming phase that could lead to distortion of the model. With the assurance that the model will not be distorted or has a very low probability of occurring, the model can now be run multiple times and the data recorded for analysis and hypothesis validation.

5.4 Results of model runs

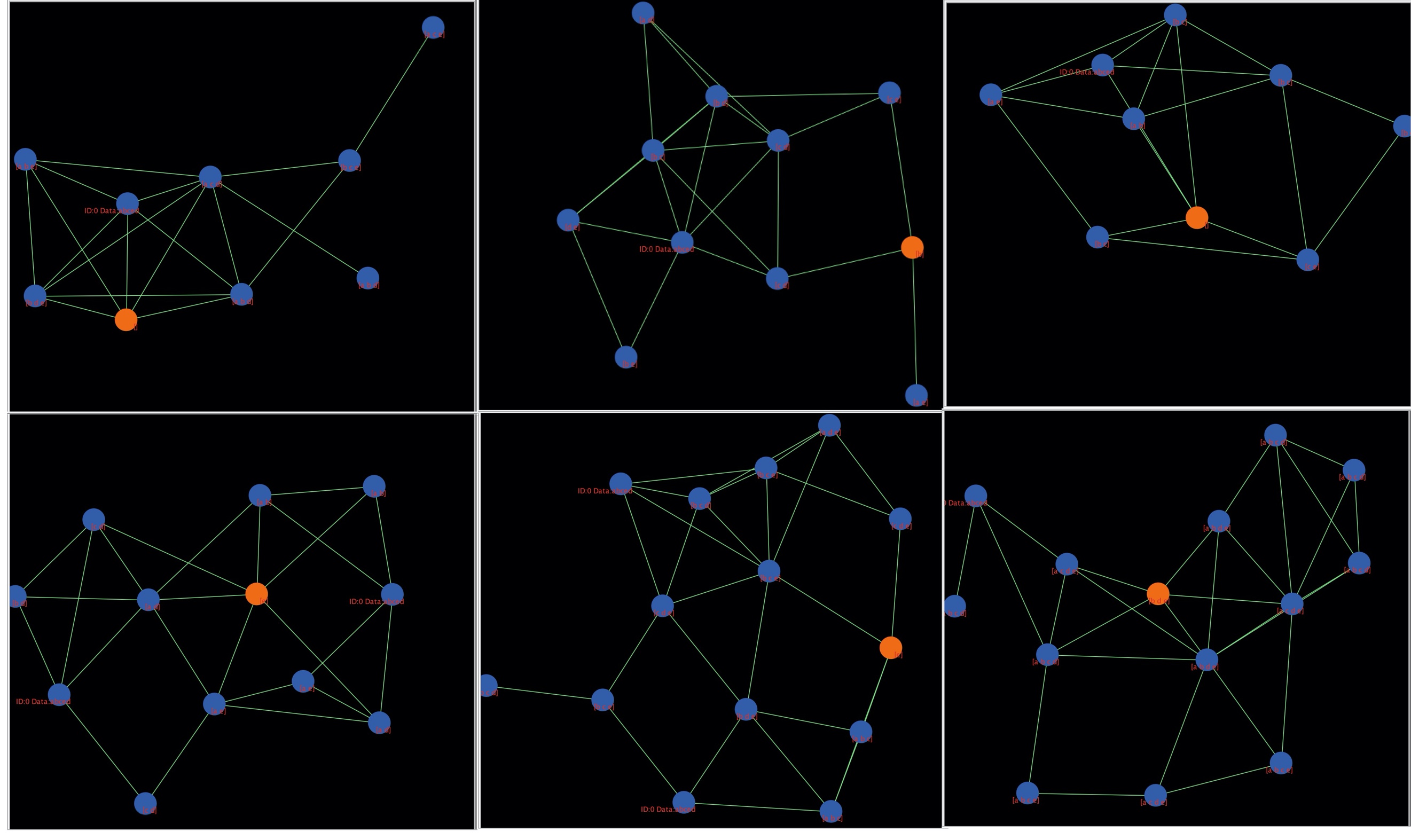
After the model has been tested and ensured that there are no problems affecting the operation, the model can be run multiple times to check that the results are as expected and objective, and to ensure the stability and scalability of the model. Since the model is divided into two versions, we run them separately and record the results, then compare them.

The unrestricted version is first run and tested, unrestricted means that the data source does not have any firewalls or protections, so any agent can access the data at any time and any place, i.e., a fully open source state.



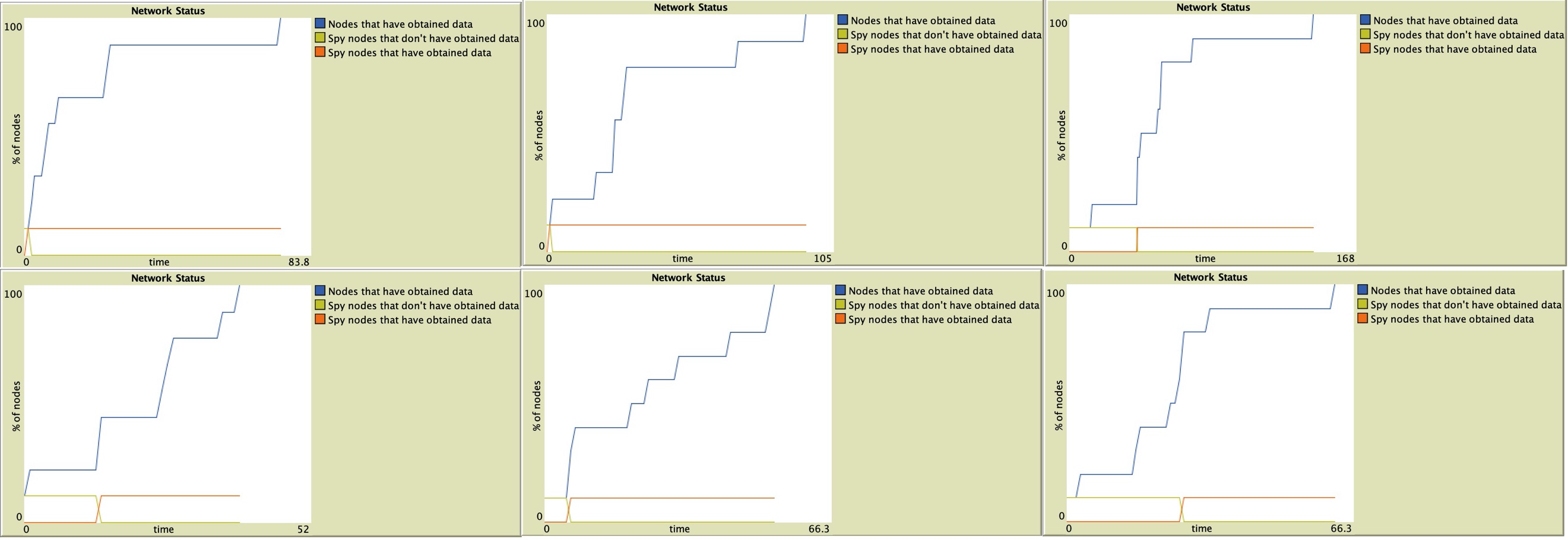
As can be seen from the diagram, in the fully open source state, all the agents can get all the data as long as they request from the data source, and then marking the spy node is meaningless because everyone can get the data they want and the existence of the spy is unnecessary. So we can see that each node ends up being the same color as the data source node and has the same data as the data source. In addition to this, we can also see that the time it took for all nodes to get data was very short, taking only 26 ticks (number of model runs) to achieve full coverage, as there were no restrictions and agents did not have to request access and wait for feedback. The unrestricted version of the model also achieves the goal of data sharing and collaboration.

The unrestricted version only achieves data sharing under ideal conditions without considering potential dangers and is therefore imperfect; it is the model with restrictions that is the focus of the study. Through multiple runs of the model we can see that after the introduction of restrictions and spy nodes, the model is still able to achieve data sharing, and the results of each complete run are random, in line with the characteristics of data sharing in a network.

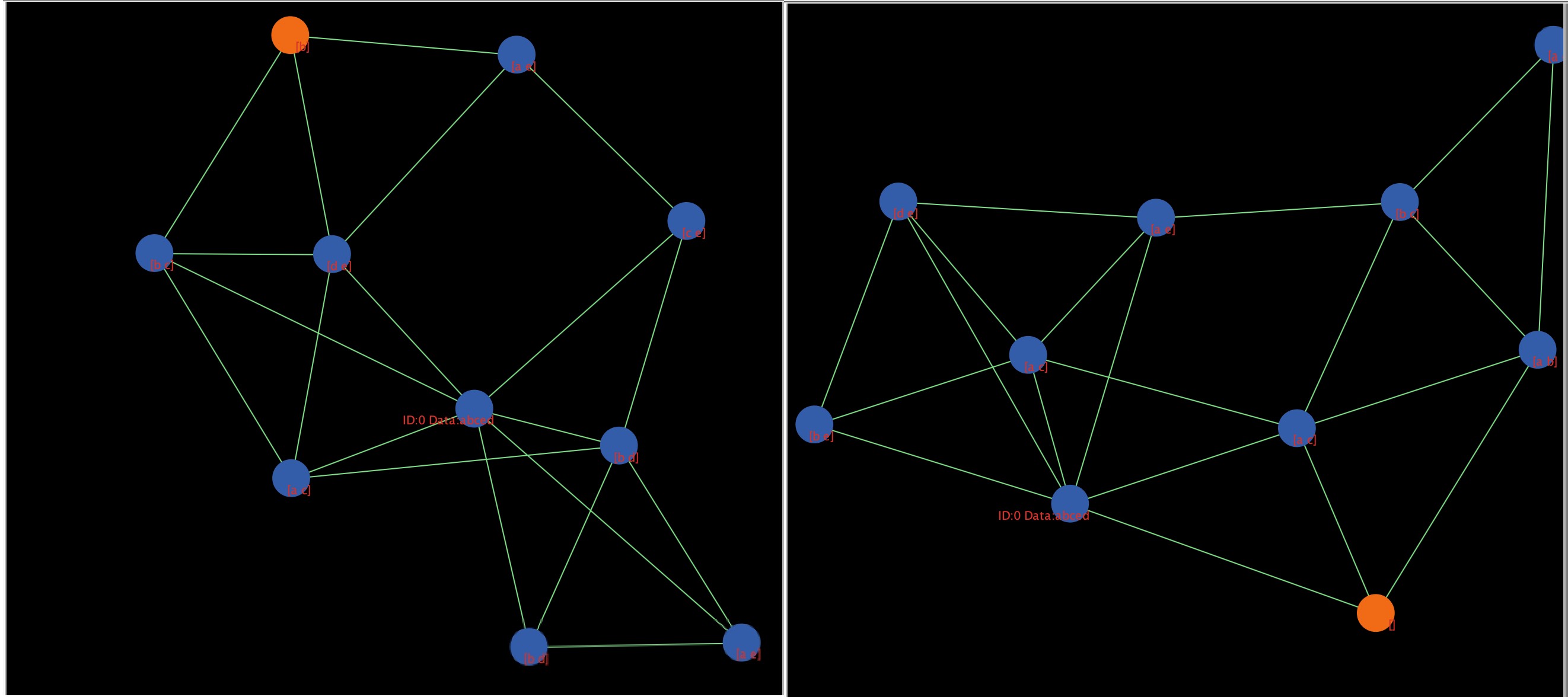


It is worth noting that the restricted version of the model has several variables that can be adjusted, such as the total number of nodes, the amount of data each node can respond to, the level of restrictions, etc. We can adjust some or all of these variables before each run in order to increase the randomness of the model and obtain different results for each run. As shown in the figure above, each model is different, some models only have access to two data per node (randomly), some have a higher total number of nodes, and some have stricter restrictions on data sharing, the effects of these different variables can be seen in the subsequent line graphs. It is worth noting that because of the restrictions introduced, we want to protect the security and privacy of the data, which is why the spy node exists; it will try to steal the data from the data source because it is not authorized to access it. Therefore, we designed the spy node to be a different color than all other nodes, so that it is easy to distinguish (in practice, the spy will try to disguise itself to prevent detection) and so that its rules for accessing data are independent of those of other nodes (the protection-tolerance slider does the job), which makes it more representative of the uncertainty of 'spies'.

The model interface can be used to study the effect of restrictions on the extent of data sharing, while the line graph can be used to study the effect of restrictions on the speed of data sharing.



As we can see from the above figure, the time taken by all nodes to obtain the data increases significantly after the introduction of restrictions and the impact of different levels of restrictions is different. For example, with the same 10 nodes, when the limit-tolerance is 2, the average time taken by all nodes to obtain data is 148 ticks, but when the limit-tolerance is 4, the average time taken by all nodes to obtain data is 65 ticks. , the greater the probability that each node gets data on each tick run, and the easier it is to get data. The tighter the restriction means that each data point may take several or even dozens of ticks to obtain the data, even if some data nodes are common and have no malicious agents. For spy nodes, we can see that under the unrestricted version, the spy node gets the data almost as soon as it starts propagating, spending on average only 5 ticks and all. However, under the restricted version, the time spent by the spy node to steal the data is significantly higher. Also in the case of 10 nodes, the average time spent by the spy node to acquire data is 58 ticks at protection-tolerance 4 and 133 ticks at protection-tolerance 2. The protections are very effective. Also, the amount and content of data that the spy node is able to retrieve each time is completely random and is not controlled by the data-of-number-received variable. In other words, even if a spy node gets by with a high level of protection, it may return empty-handed, or with only a small amount of data, making its efforts futile or unproductive.



This is also to simulate a real world situation where a data source or database may have more layers of protection, and simply breaking through one layer may still not help.

In summary, we believe that the model largely fulfills its stated function and can be used to model data sharing and collaboration, and to study the impact of data protection measures. Likewise, our hypothesis that imposing different levels of protection measures on data sources is effective in protecting data security and privacy, but also has an impact on the time spent on the process of data sharing.