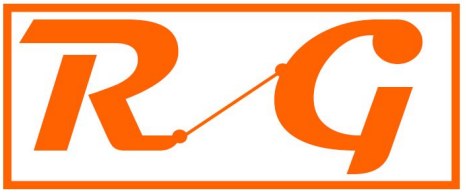
RepGraph



User Manual

---

1. Introduction

This tool will aid in the visualisation and analysis of meaning representation graphs (MRG) produced according to the linguistic framework known as Dependency Minimal Recursion Semantics (DMRS).

There are four visualisation formats:

- Hierarchical - The MRG is constructed focusing on node spans.
- Tree - The MRG is constructed focusing on directed edges.
- Flat - The MRG is constructed focusing on token order.
- Planar - The MRG is constructed in a similar manner as Flat but highlights the planarity of a graph.

Furthermore the analysis tools available include:

- Displaying subsets based on adjacent or descendent nodes.
- Searching for graphs that contain a specific sub-graph pattern.
- Searching for graphs that contain a specific set of nodes.
- Comparing two graphs based on their nodes and edges.
- Finding the longest directed or undirected path.
- Determining if the graph is connected.
- Determining whether the graph is planar.

This user manual will give you the necessary knowledge to efficiently and correctly use this tool.

2. Getting Started

The User Interface has been created to allow for a quick and efficient way to make use of the visualisations and analysis tools offered.

The following is a step by step guide on how to start using the tool and a general overview of the many components present.

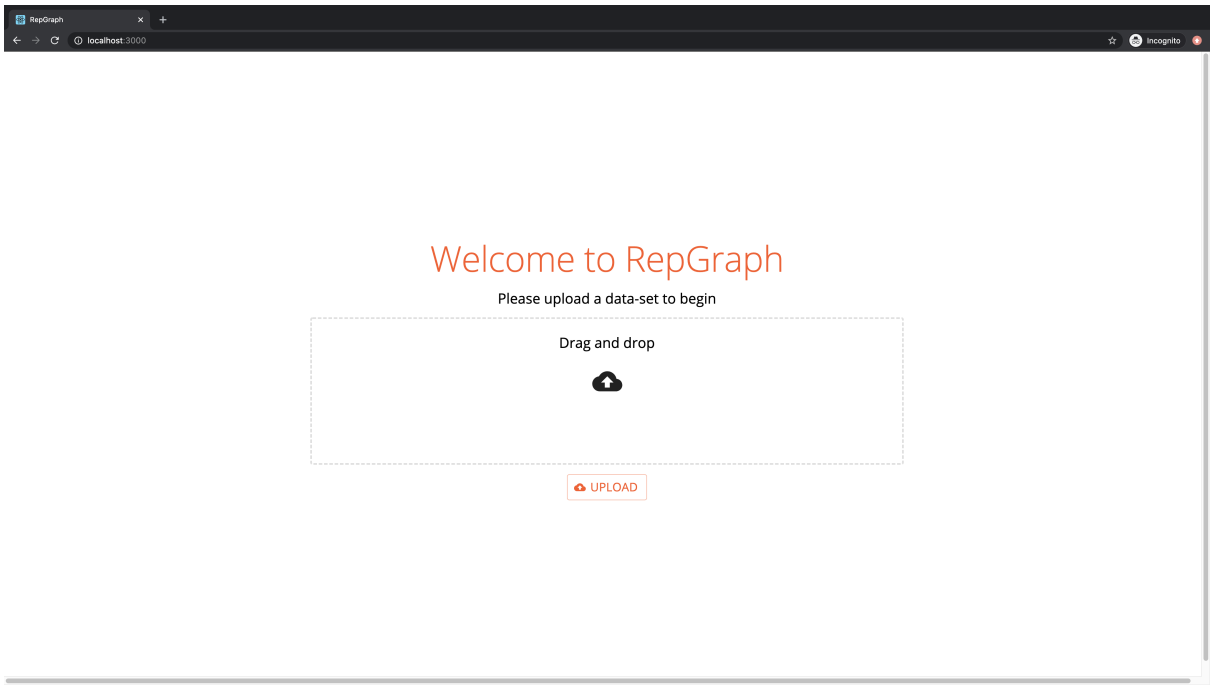
2.1. Upload a File

The first step to using the tool is to upload a data-set of your choosing. This data-set needs to follow a DMRS format and each meaning representation graph in the data file needs to be on a new line.

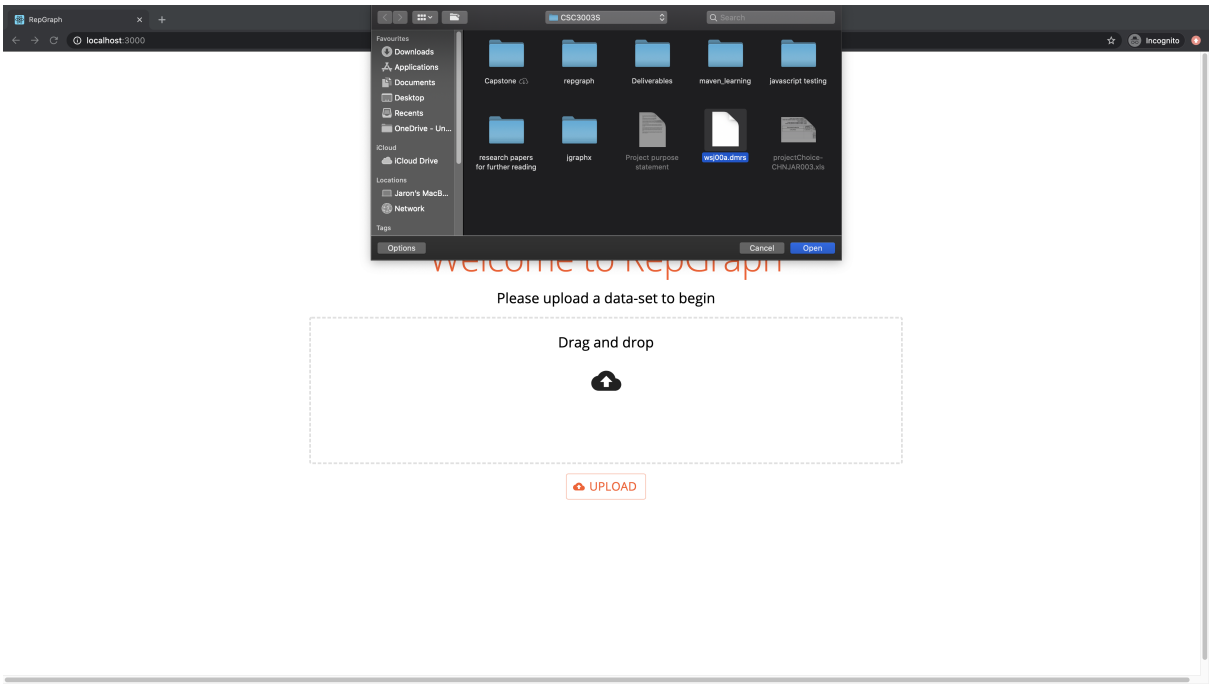
An example of the parsed DMRS format data is as follows:

{"id": "20001001", "source": "wsj00a", "input": "Pierre Vinken, 61 years old, will join the board as a nonexecutive director Nov. 29.", "tokens": [{"index": 0, "form": "pierre", "lemma": "Pierre", "carg": "Pierre"}, {"index": 1, "form": "Vinken,", "lemma": "Vinken", "carg": "Vinken"}, {"index": 2, "form": "61", "lemma": "61", "carg": "61"}, {"index": 3, "form": "years", "lemma": "year"}], "nodes": [{"id": 0, "label": "proper_q", "anchors": [{"from": 0, "end": 0}]}, {"id": 1, "label": "named", "anchors": [{"from": 0, "end": 0}]}, {"id": 2, "label": "named", "anchors": [{"from": 1, "end": 1}]}], "edges": [{"source": 9, "target": 2, "label": "RSTR", "post-label": "H"}, {"source": 3, "target": 2, "label": "ARG1", "post-label": "EQ"}], "tops": [10]}

The user will be shown an upload page as follows:

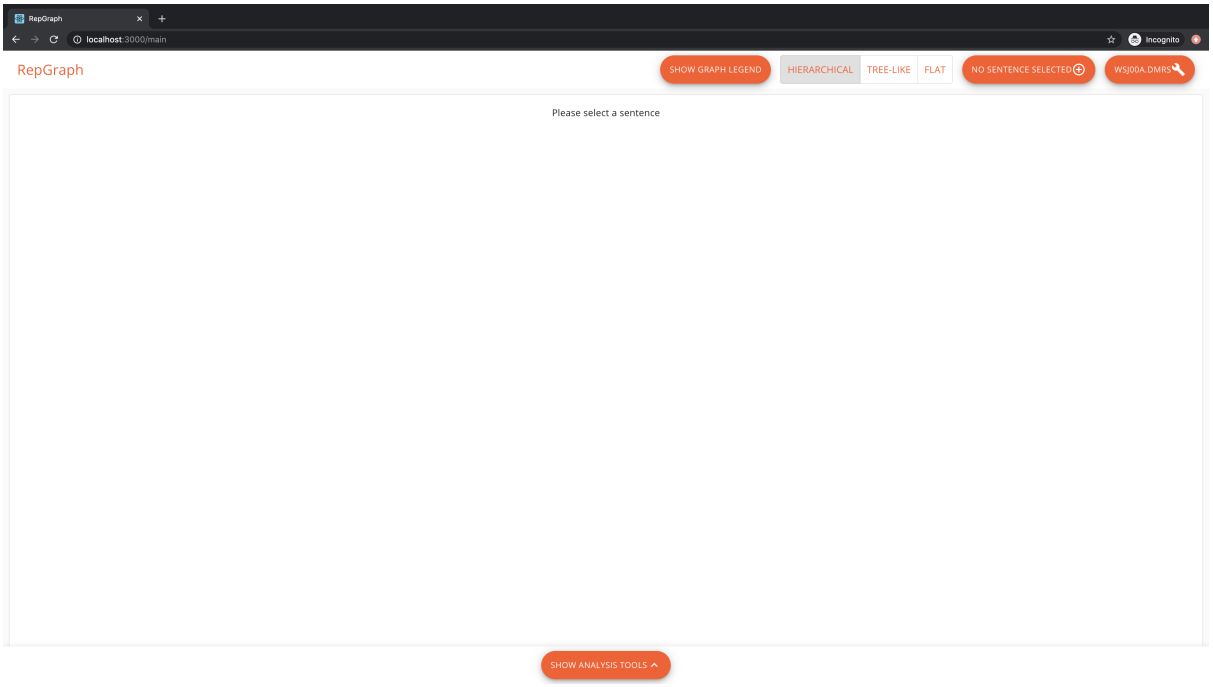


Once on this page, the user can simply drag and drop their data file into the box or make use of the upload file dialog to select their file from their local directory. The file dialog only looks for .dmrs extensions.

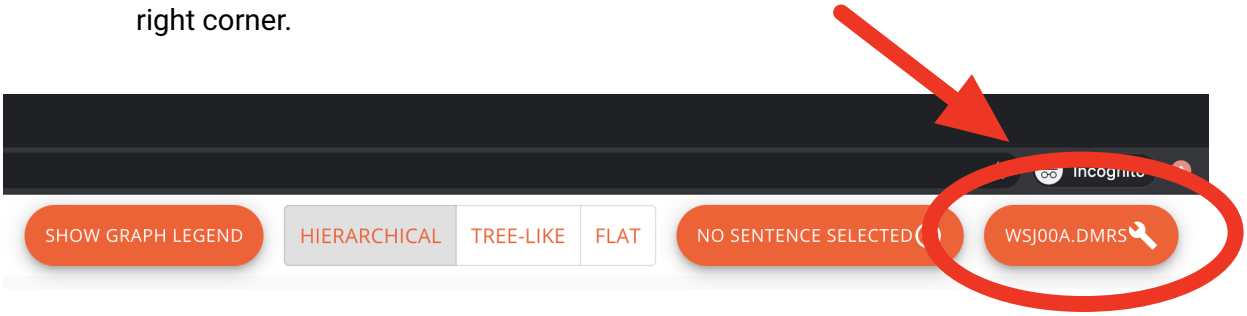


When the file is present in the upload box, the user must click the upload button which will then send the file to the server to be processed and stored.

Once a file is uploaded, the user will be taken to the main page.

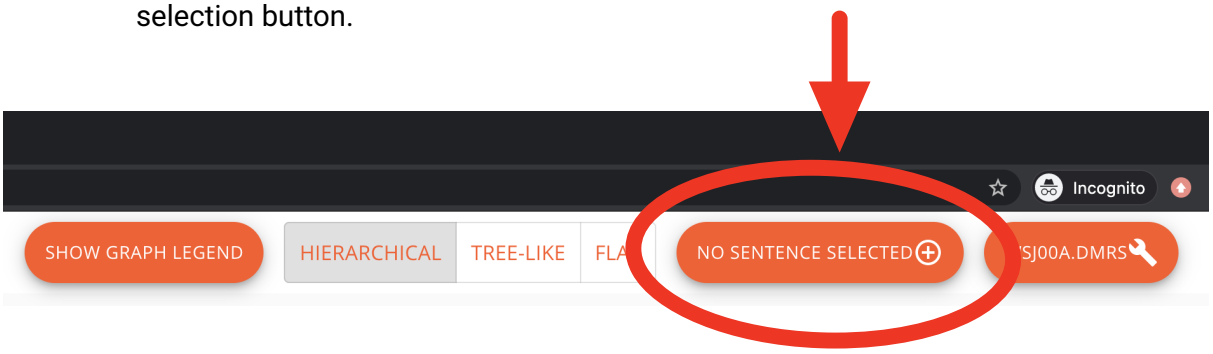


To reupload a new file, the user must click on the data-set button in the top right corner.

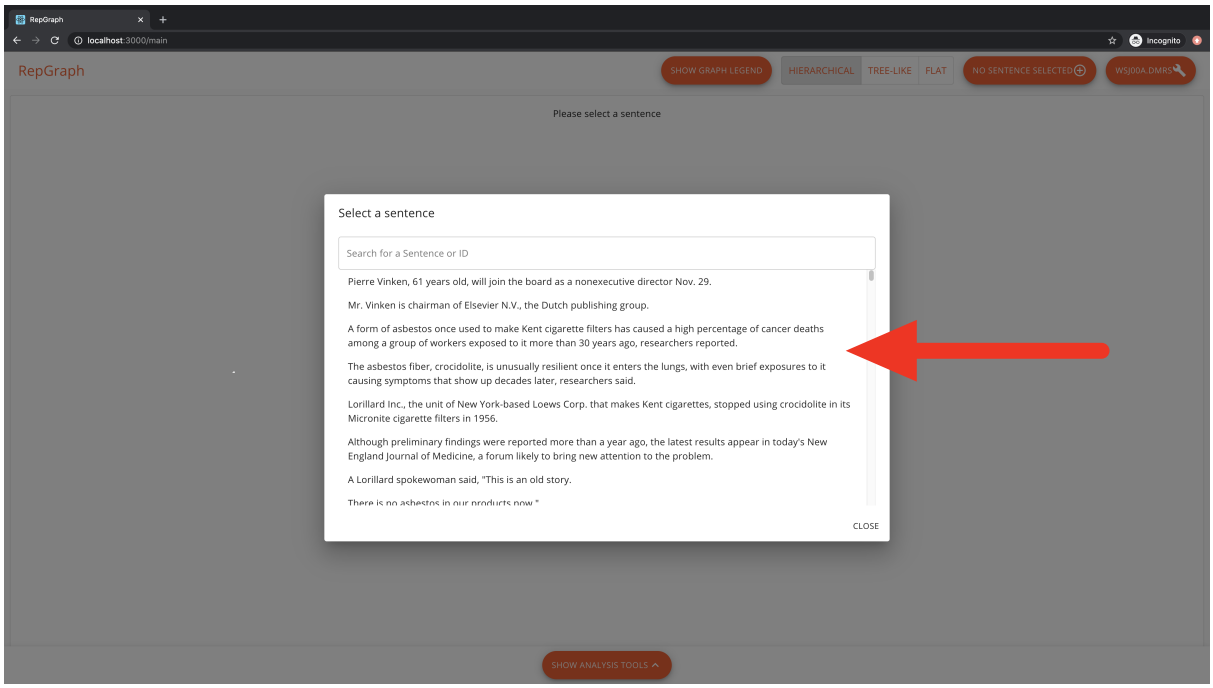


2.2. Select a Sentence

To select a sentence for visualisation, the user must click the sentence selection button.

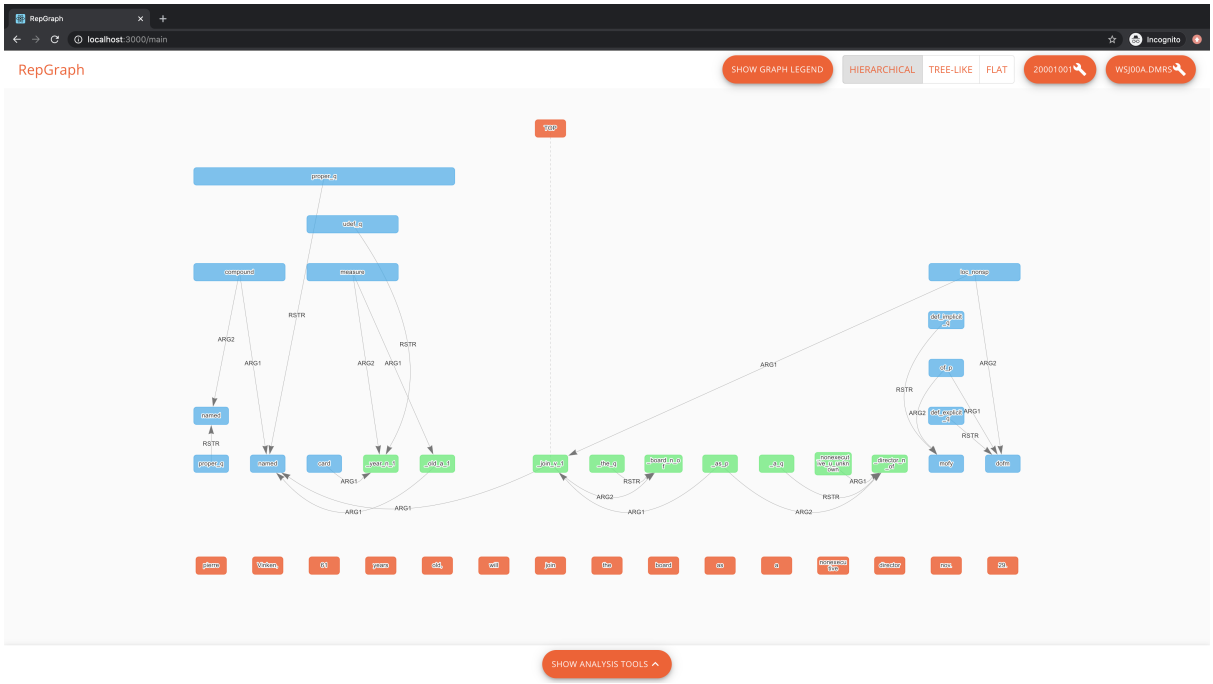


When the sentence selection button is clicked, the user will be shown a list of sentences from the dataset.



From here, the user can scroll to find or search for (either by words or ID) the sentence they want visualised.

To visualise a sentence, the user must simply click on the sentence in the list. Once a sentence has been clicked on, the list dialog will disappear and the visualisation will appear.



When a sentence has been chosen, all visualisations and analysis tools will act upon the chosen sentence.

2.3. Visualisation Formats

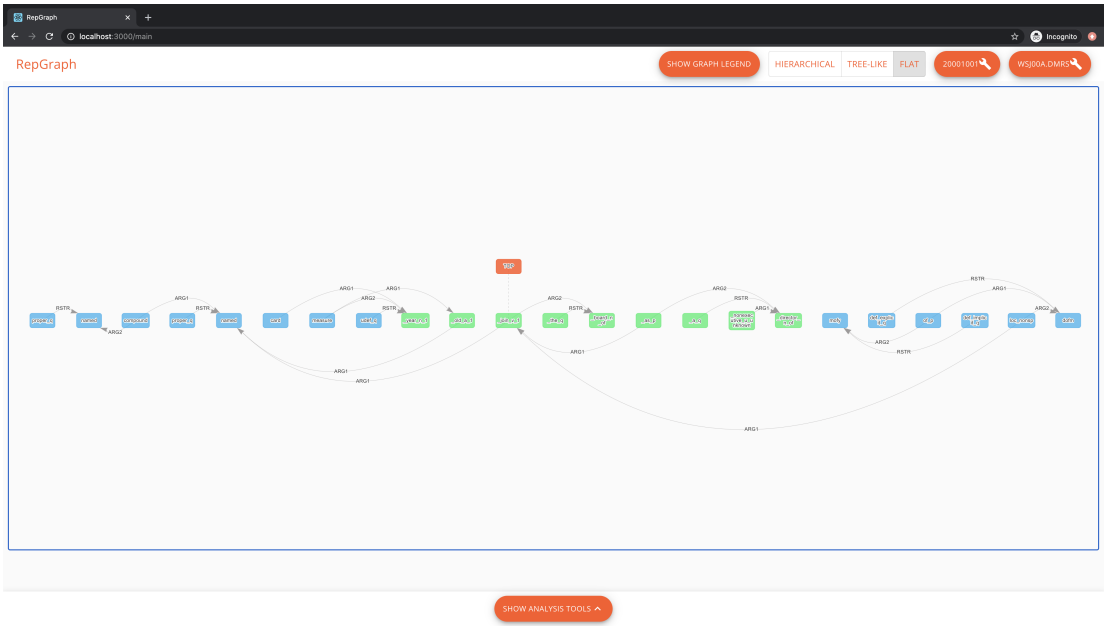
The tool offers 4 different visualisation formats to view meaning representation graphs.

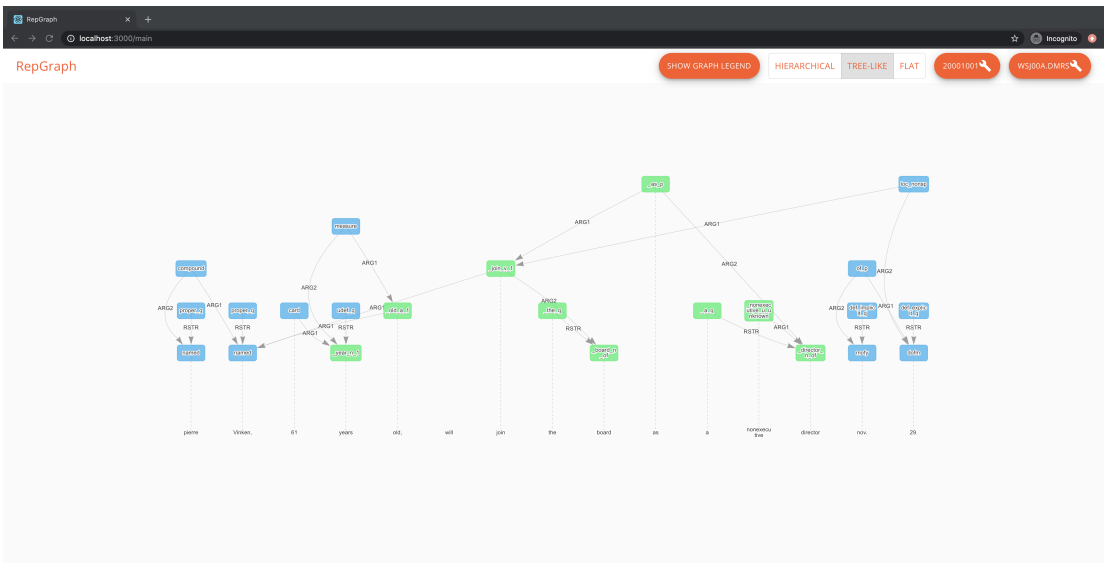
- Hierarchical
- Tree
- Flat
- Planar - *Special Case*

The last visualisation “Planar” is a special case and is not viewed in the same way as the others but that will be explained in the ***Formal Tests (4.5.2)*** section.

The three main visualisations are selected on the Top bar of the tool

Whichever visualisation is clicked on, decides the visualisation of all graphs throughout the application, including within the analysis tools. This means users can choose whichever format they wish to view all graphs in consistently.

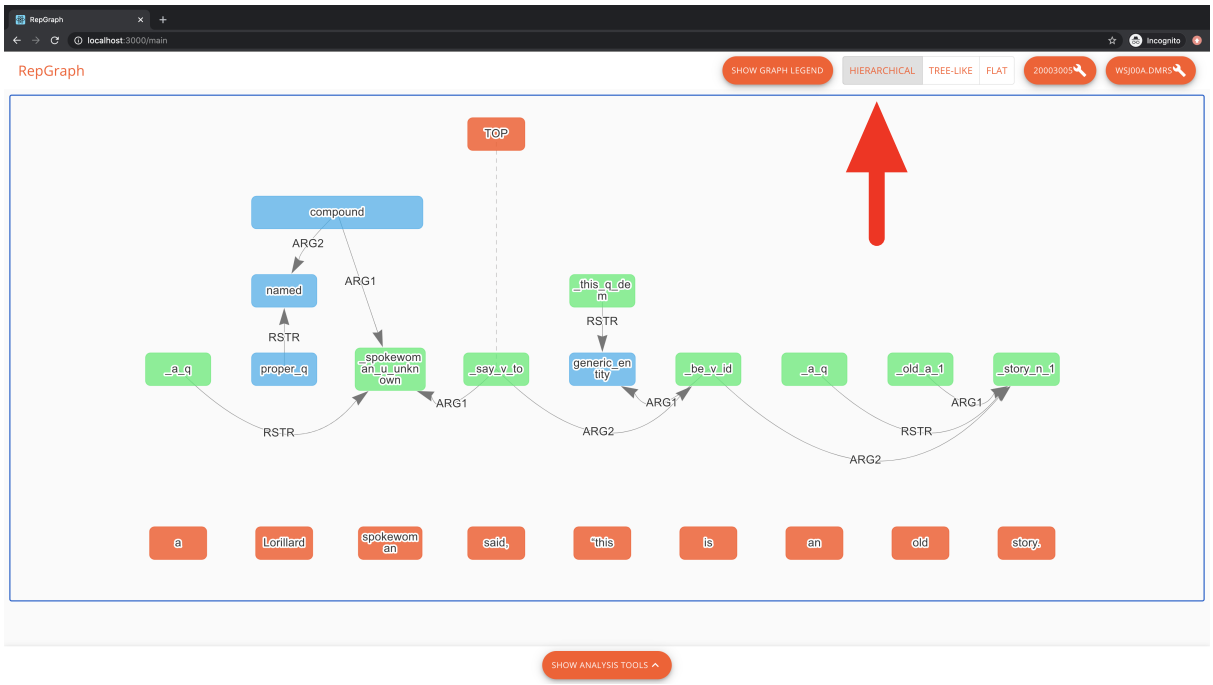




3. Visualisations

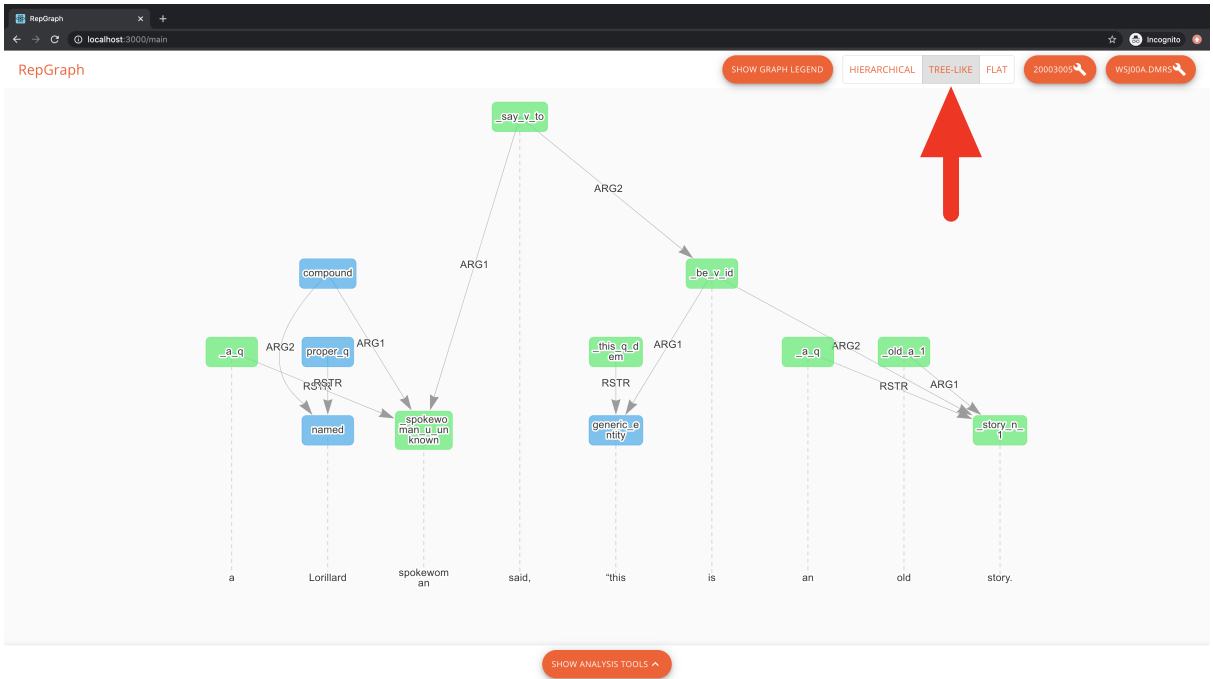
3.1. Hierarchical

The hierarchical visualisation format focuses on the spans of nodes. The width of nodes corresponds with the number of tokens they span.



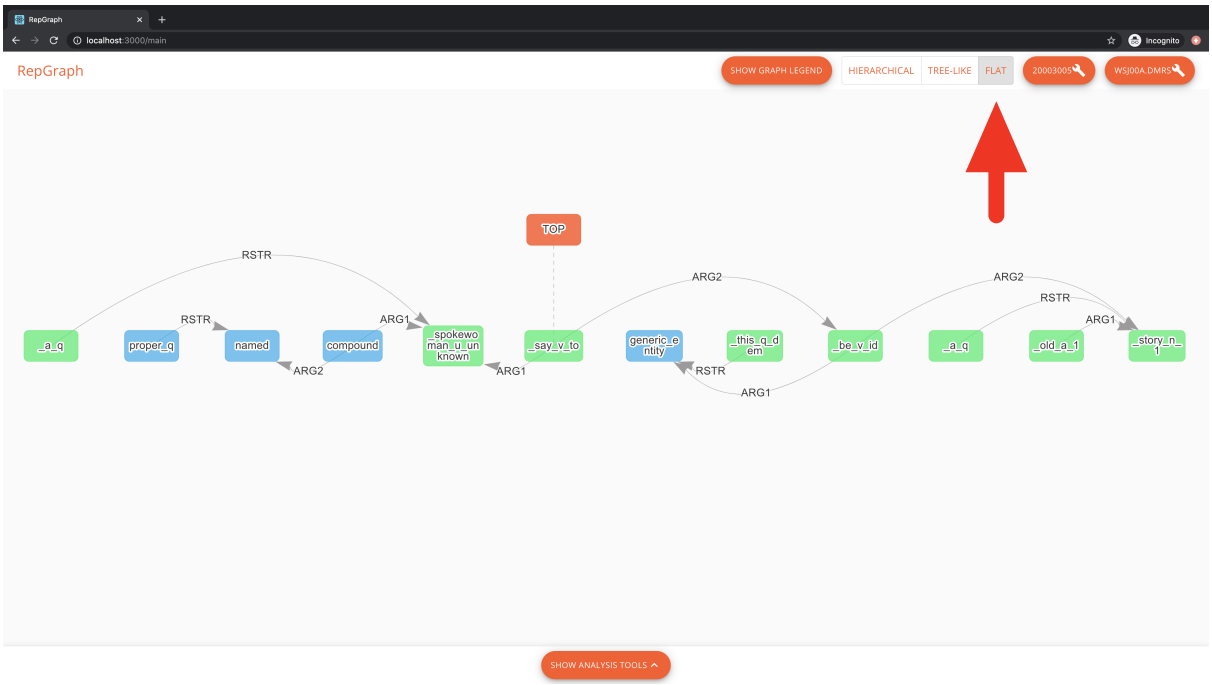
3.2. Tree

The Tree visualisation focuses on nodes with the most descendents i.e it visualises the graphs in a tree like manner with all nodes that are below other nodes have less descendants.



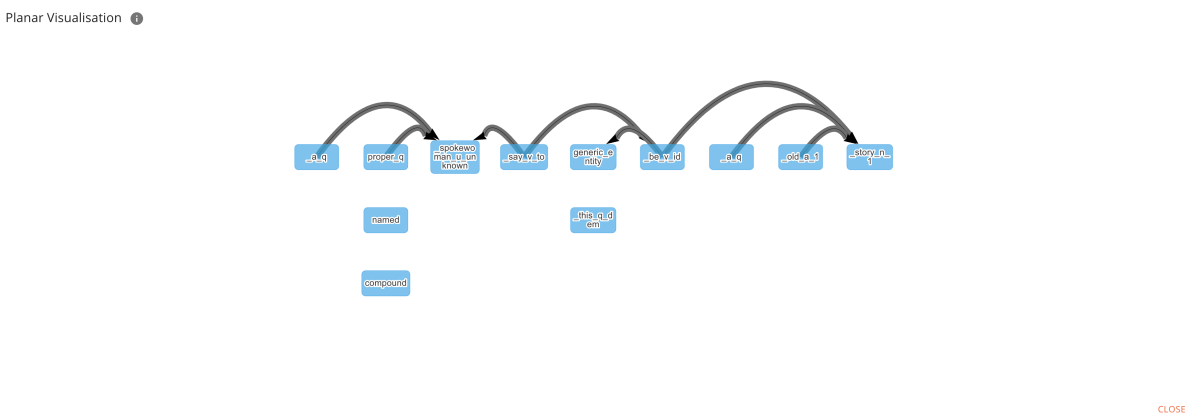
3.3. Flat

The Flat visualisation is the simplest visualisation which just shows the nodes on a flat plane with their edges.



3.4. Planar

The planar visualisation is a special visualisation showing all nodes in their token reference position and showing all edges in the top plane whilst highlighting crossing edges.



4. Analysis Tools

The analysis tools are used to analyse the selected graph or other graphs in the model. To access the analysis tools, the user must click on the “Show Analysis Tools” button.

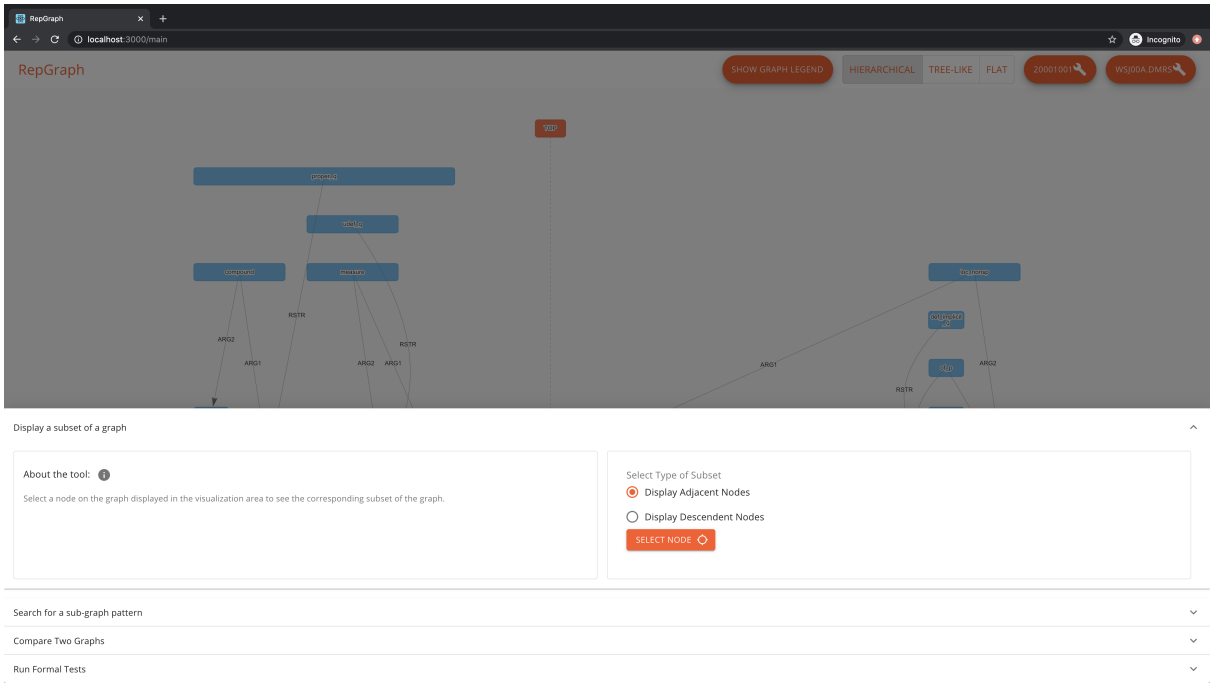
which will then proceed to display a list of the tools available.

To use any of the tools, the user must click on it in the list.

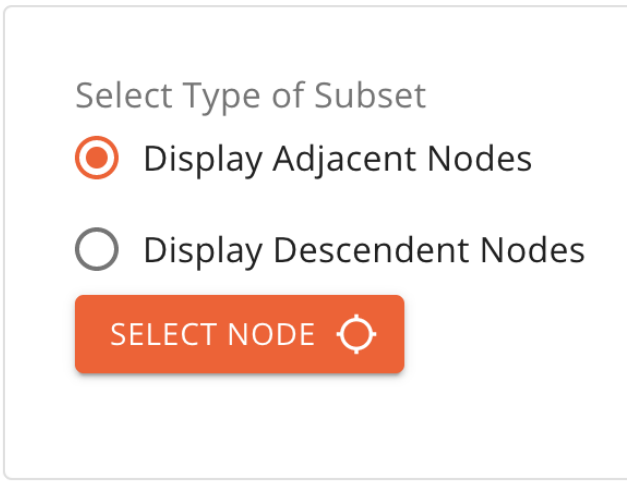
4.1. Display Subset

The display subset tool is used whenever a user wants to only see a specific portion of a graph or if they want to see how a specific node is connected and/or relates to other nodes.

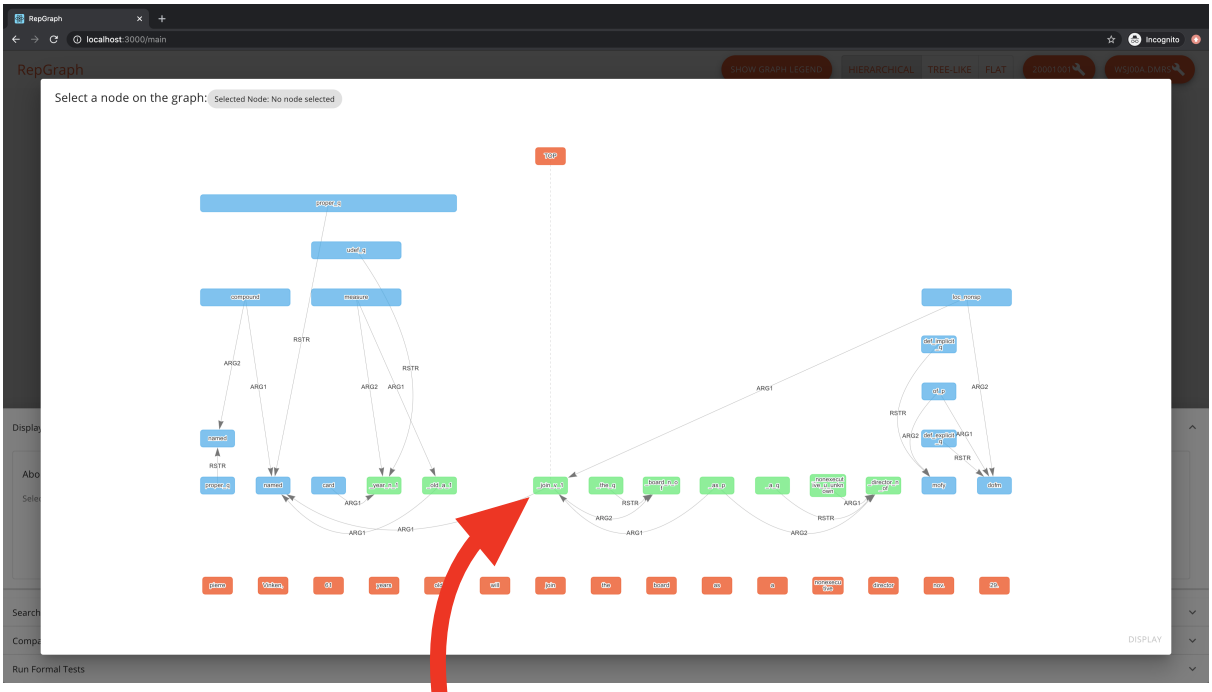
Once the tool is clicked, the list will expand to show the user the tool's options.



The user can then decide whether they want an adjacent subset (meaning the subset is created based on the selected node's adjacent nodes i.e all nodes that are connected to that node regardless of edge direction) or a descendent subset (meaning the subset is created based on the selected nodes descendents).

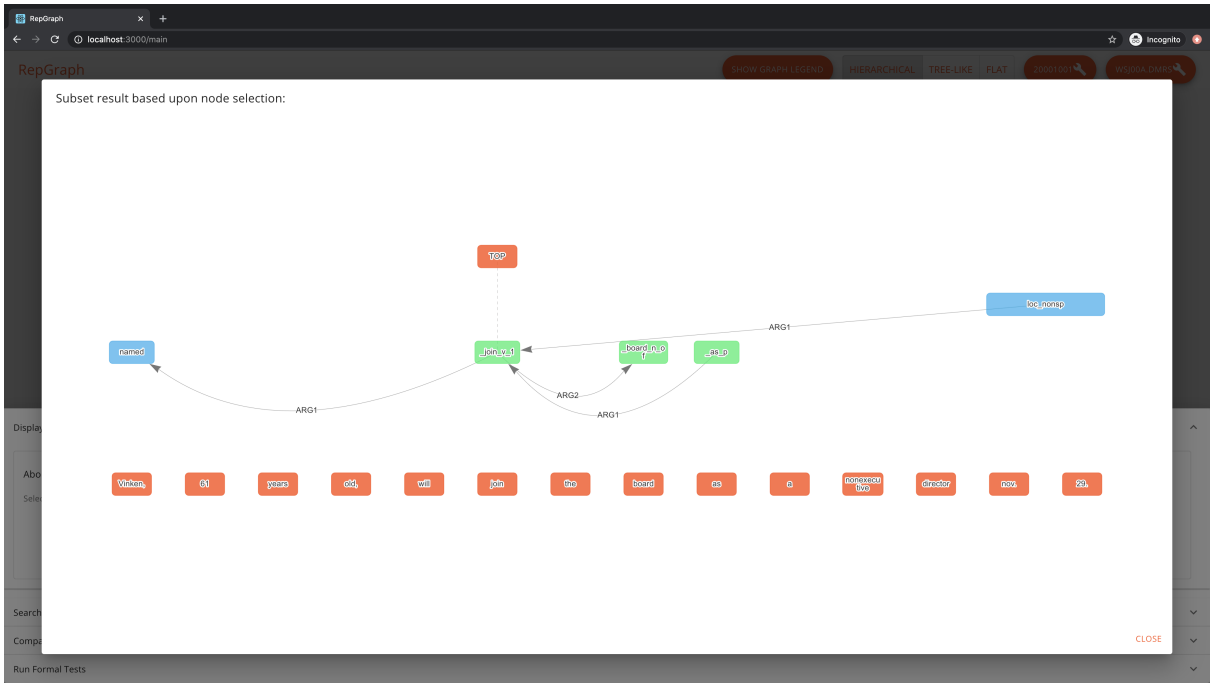


Once the user has selected which type of subset they want displayed, the user must click the “Select Node” button. This will open a dialog of the current graph selected.



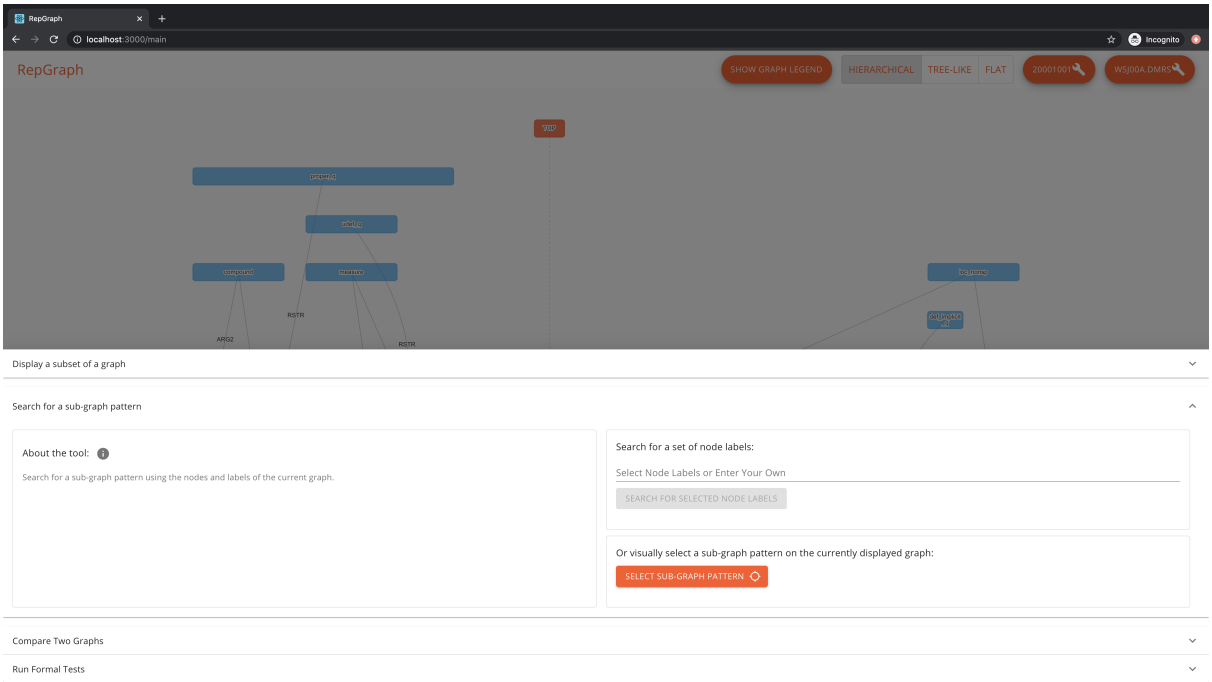
The user must then select the node they want the subset to be created from. After the user has selected the node they want, they must press the “Display” button to display the subset.

This will result in a dialog with the subset visualisation appearing.



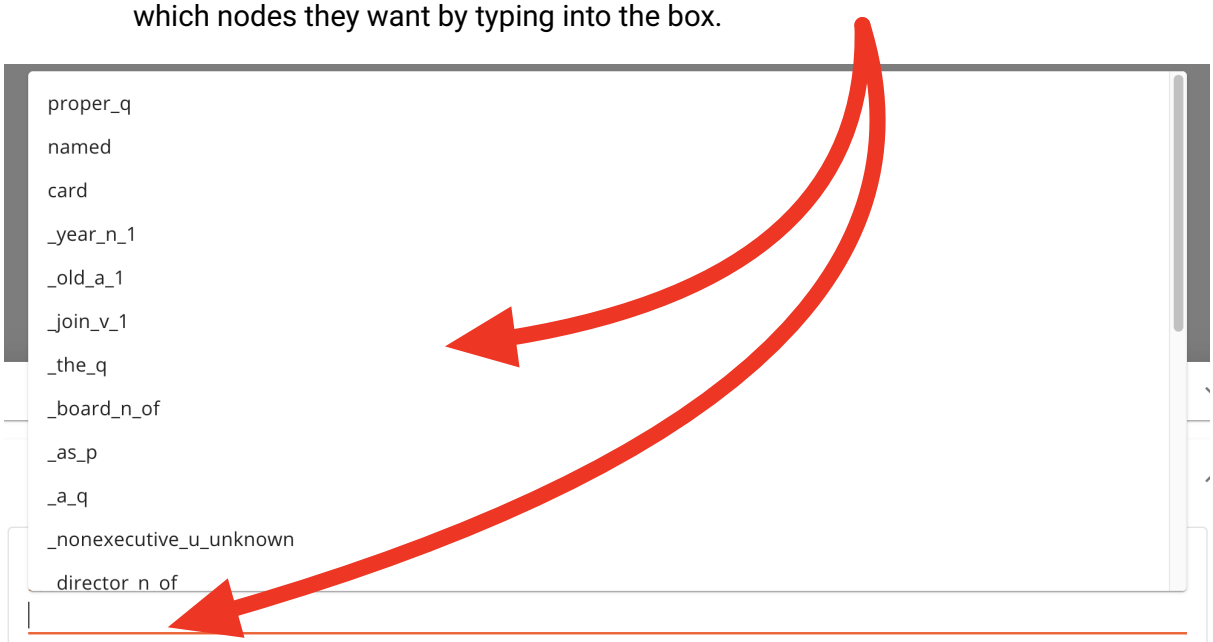
4.2. Search For Set of Nodes

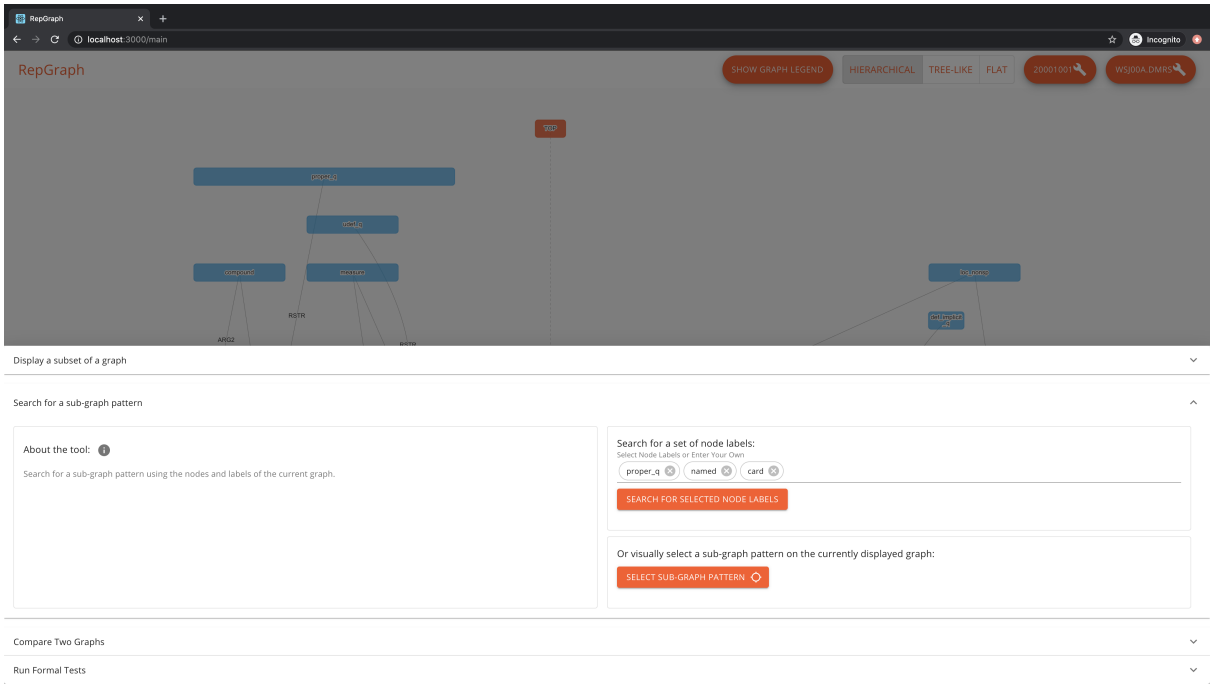
The search for a set of nodes tool is used when the user wants to know which graphs in the data-set have a certain set of nodes.



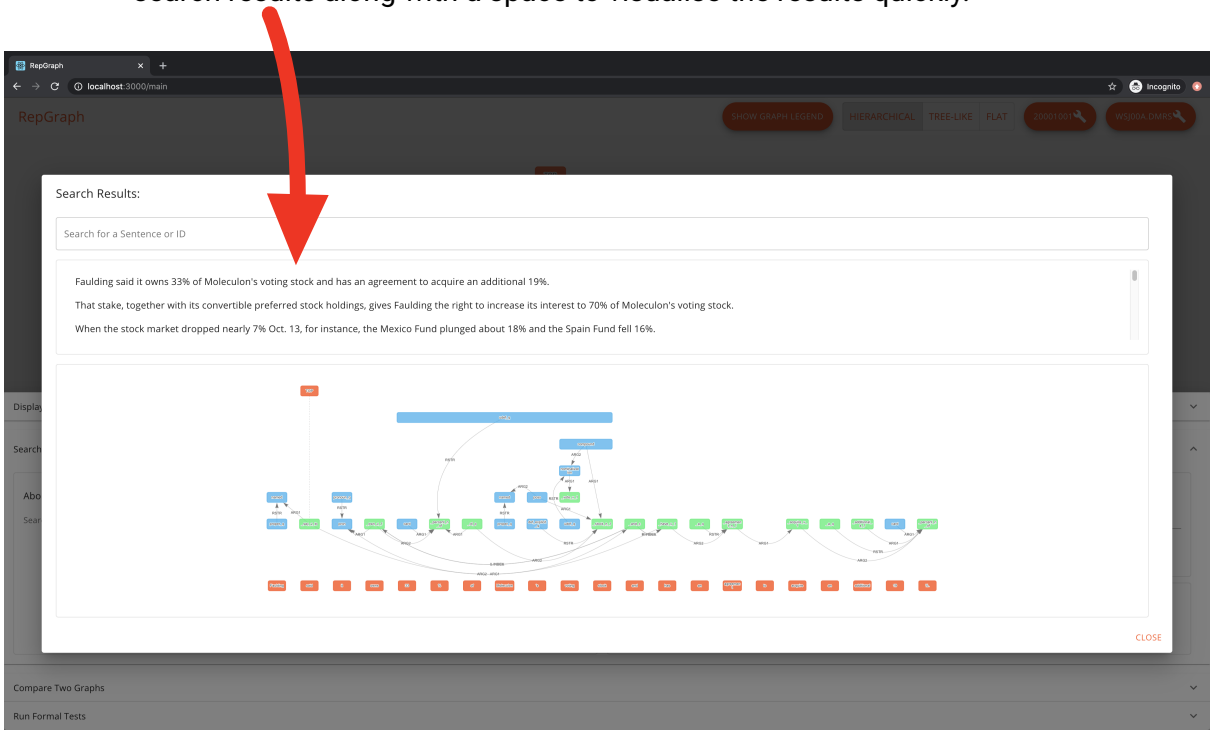
To search for which graphs have a set of nodes, the user must enter which nodes they want searched for.

They can do this either by clicking the dropdown menu and using node labels that are present on the current selected graph or they can manually enter which nodes they want by typing into the box.



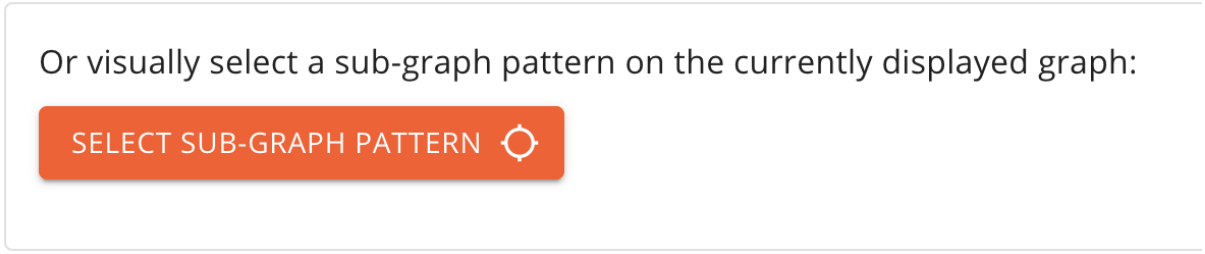


After the button has been clicked, a dialog will appear showing the user the search results along with a space to visualise the results quickly.



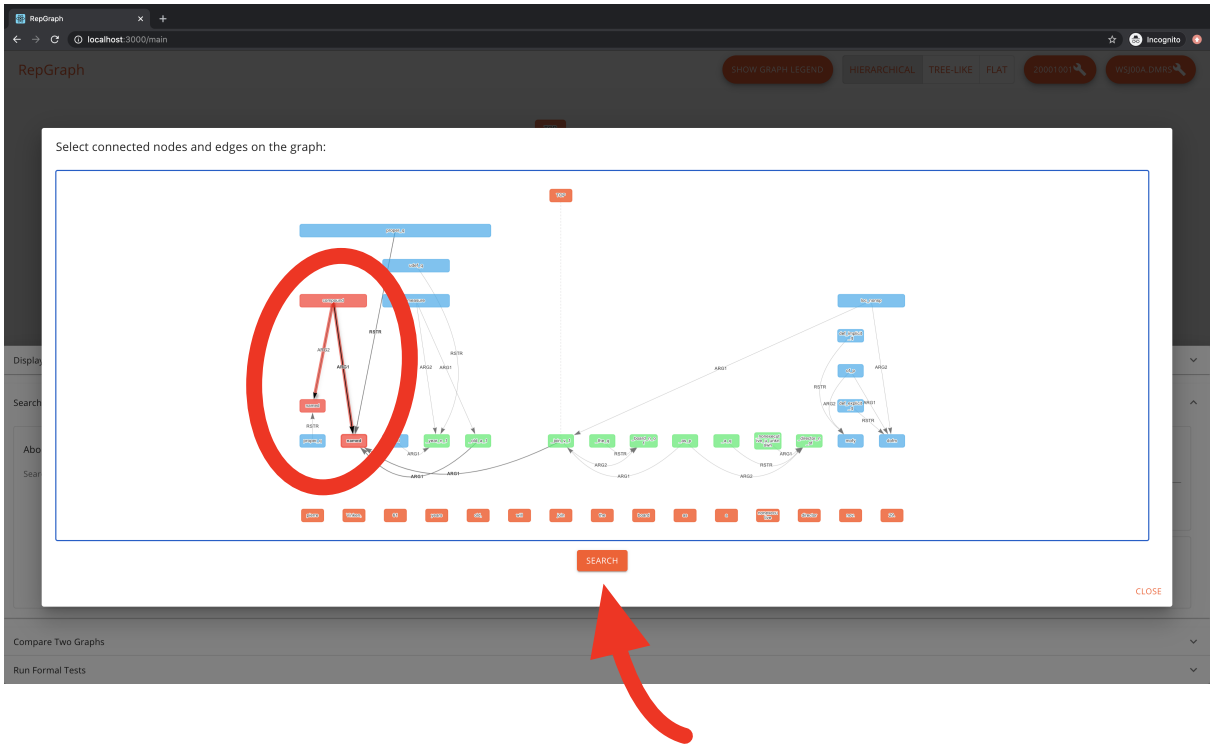
4.3. Search For Subgraph Pattern

The search for subgraph pattern tool is used when the user wants a specific pattern in a graph searched for in other graphs.



To use this tool, the user must click on the “Select Sub-Graph Pattern” button. This will open up the current selected graph in a dialog to allow the user to select the pattern visually.

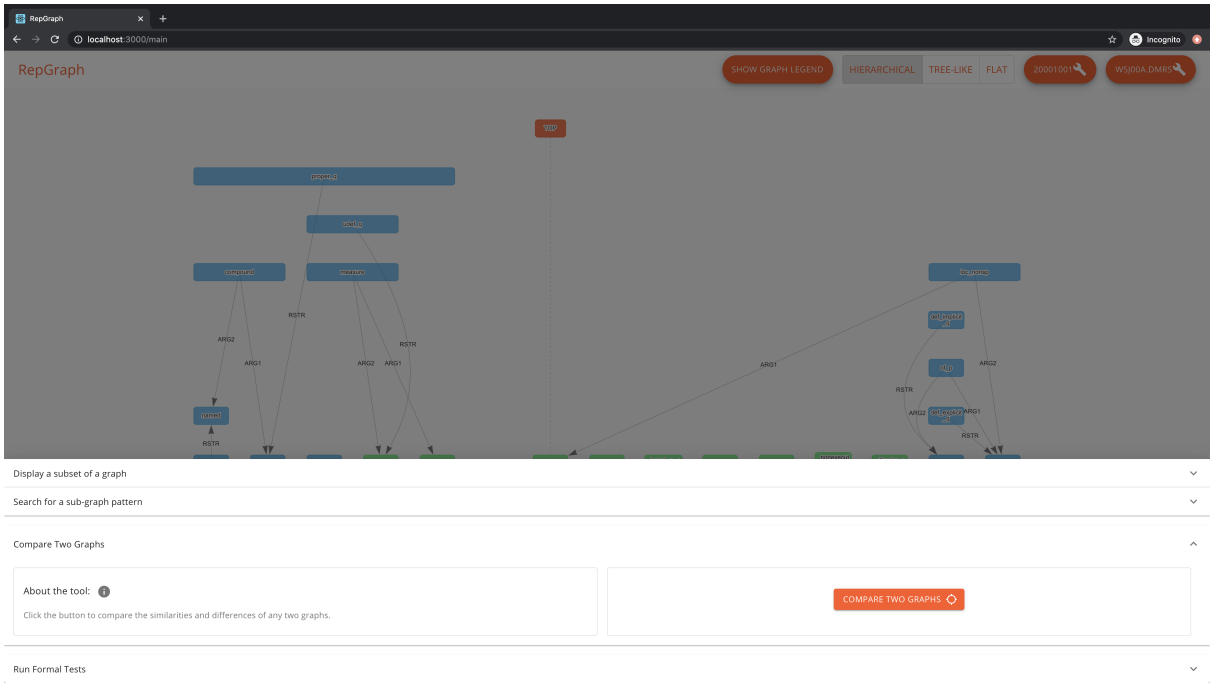
Once the dialog is shown, the user must click on the nodes and edges of the pattern they want to be searched.



After the user selects the pattern they want, they must click the search button which will then display a dialog with the search results along with a space to visualise the results quickly.

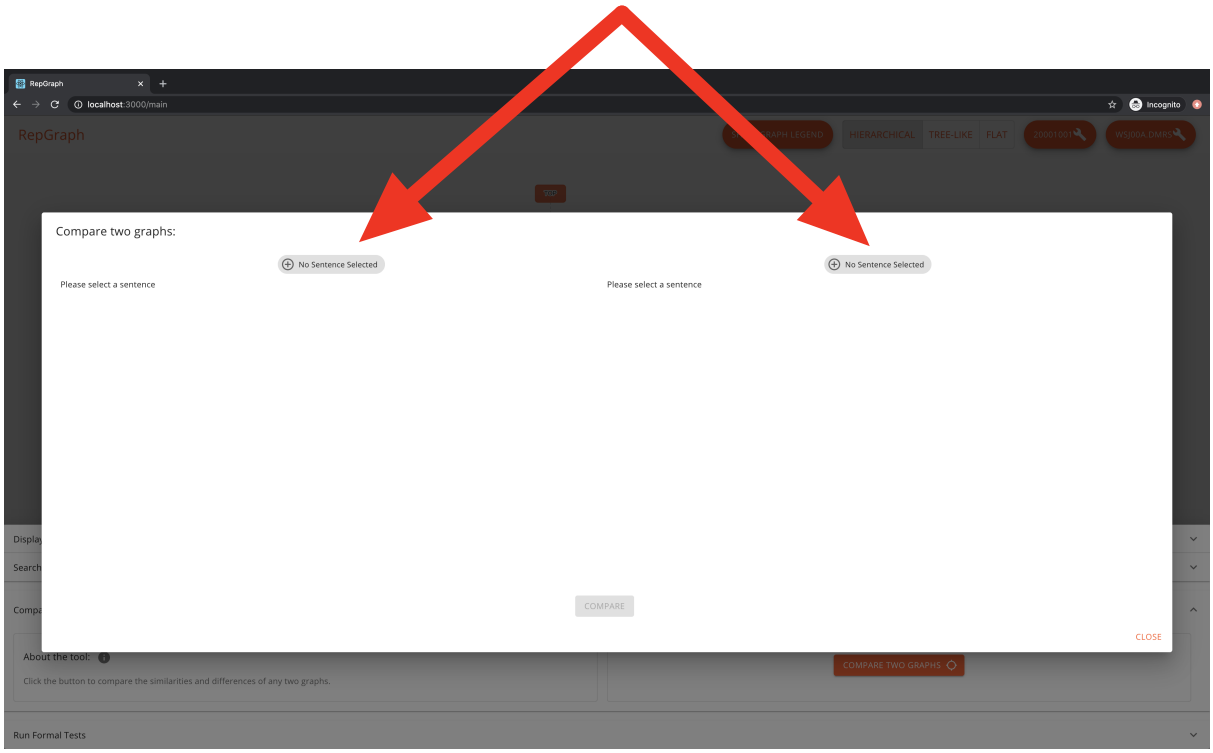
4.4. Compare Two Graphs

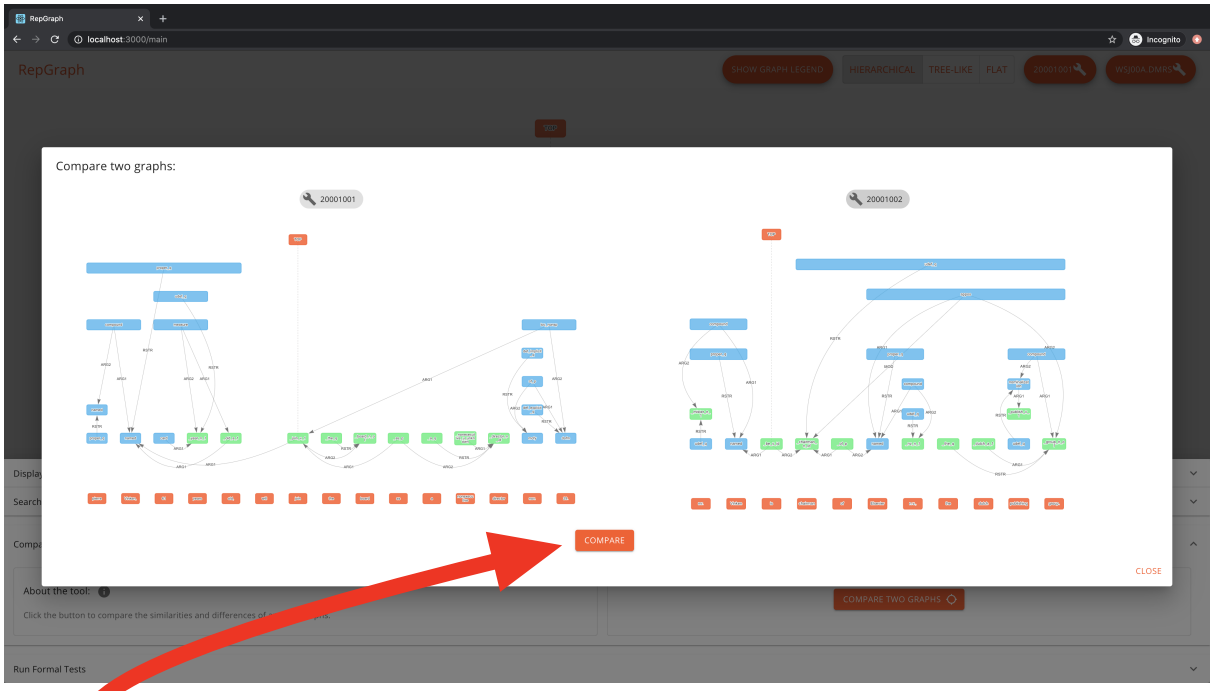
The compare two graphs tool is used when a user wants to compare two different graphs based on their nodes and edges.



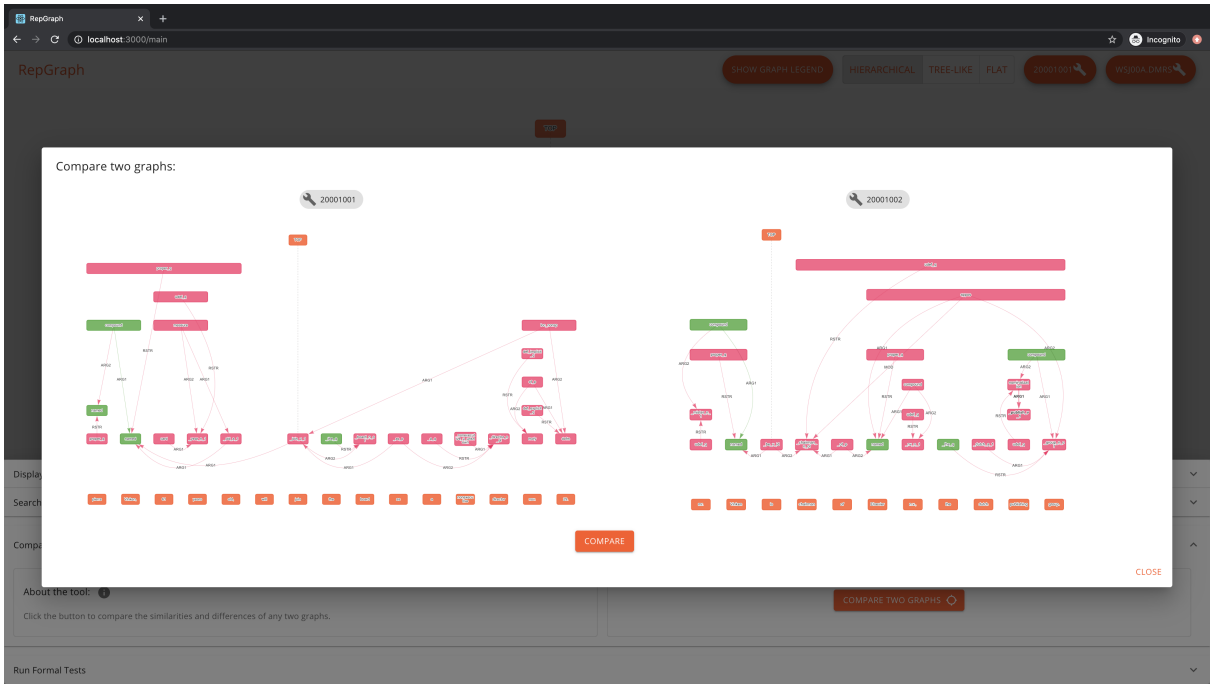
To use this tool the user must click on the “Compare Graphs” button which will open a dialog showing two spaces to select a graph.

The user must then select two graphs by independently choosing them from the list by clicking the to sentence selection buttons





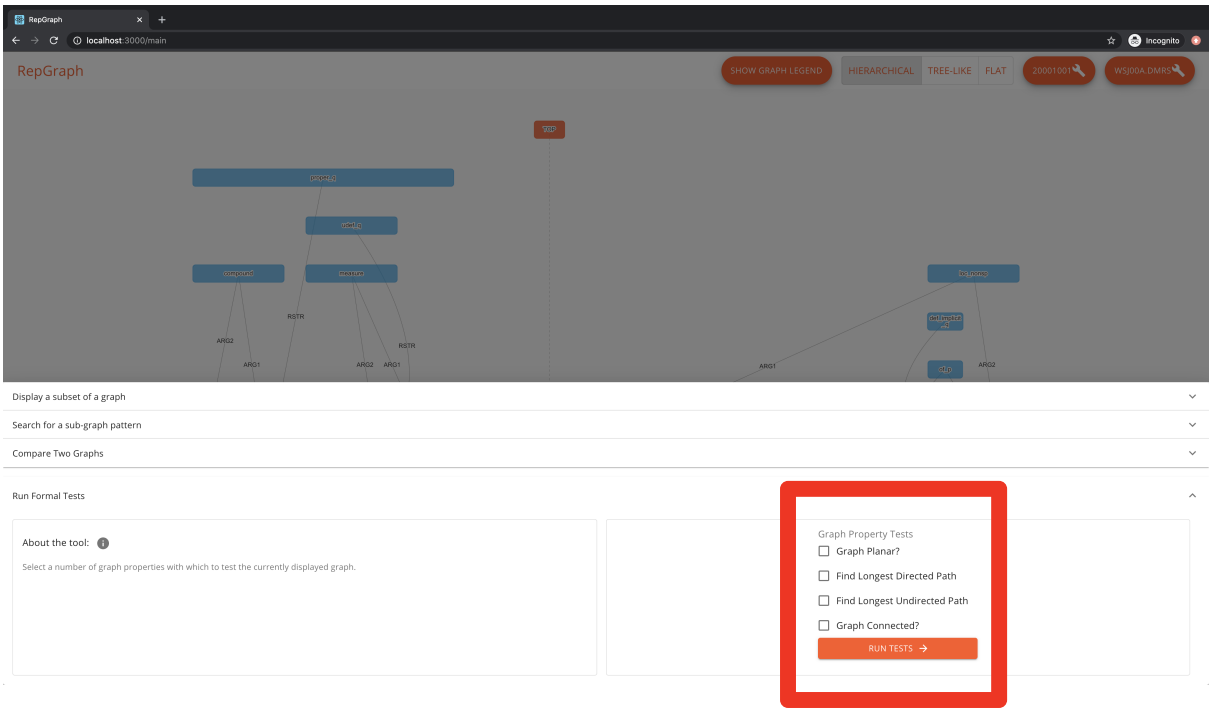
Once two graphs have been visualised on the dialog, the user must click the “Compare” button to compare the graphs. The graphs will then change colour indicating the similar nodes and edges on each with dark green colour and non similar nodes and edges with a red colour.



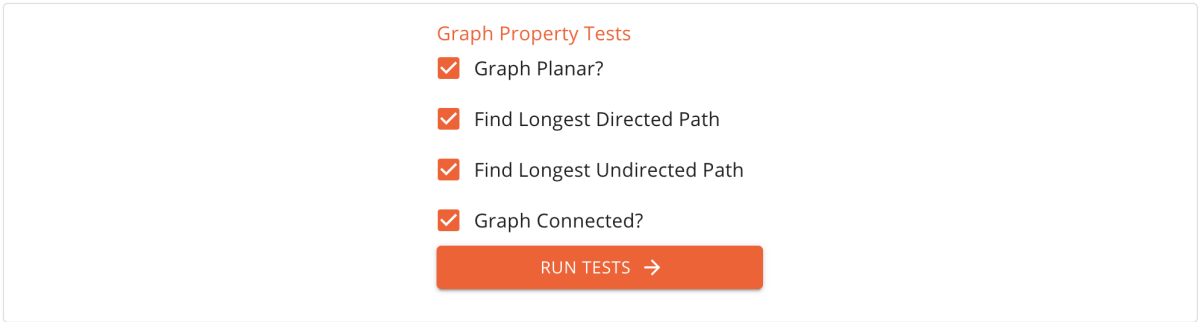
4.5. Formal Tests

The formal tests tool consists of a variety of tests the user can perform on the selected graph. The tests available are:

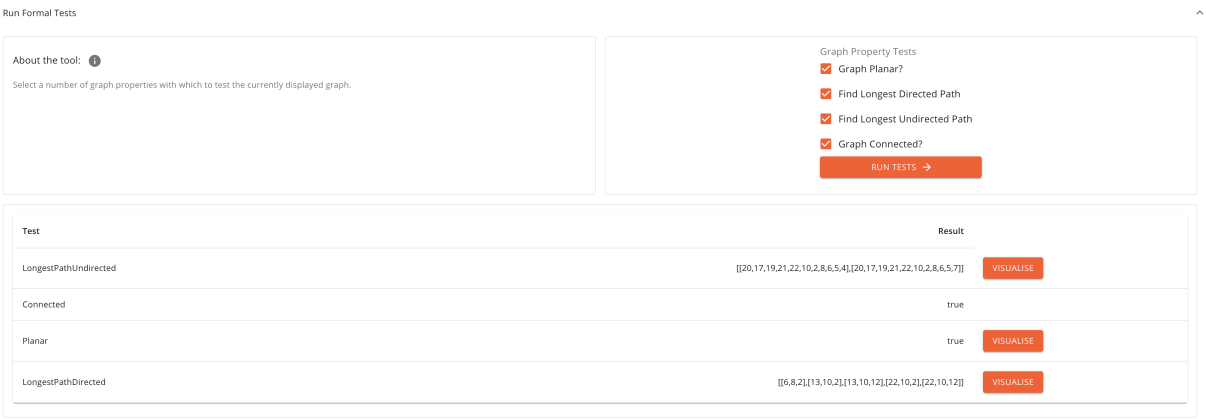
- Is the graph connected?
- Is the graph planar?
- What is the longest directed path?
- What is the longest undirected path?
- Directed vs Undirected Cycle Checker

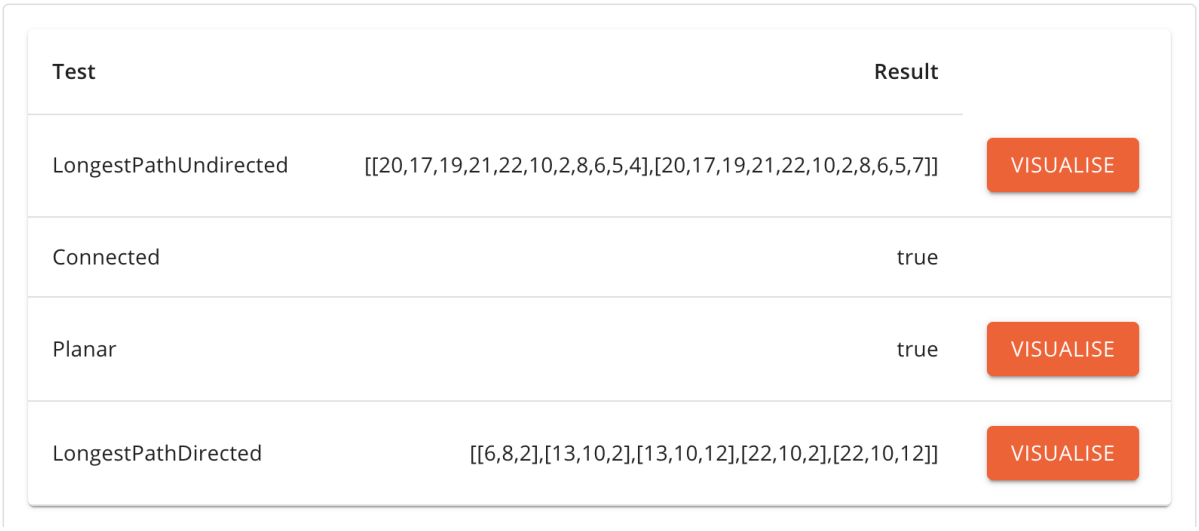


To run any of the tests, the user must select the test they want performed by ticking the checkbox and clicking run tests.



The results are displayed in a list below.





4.5.1. Connected

The connected test simply tells the user if the selected graph is connected or not. It will return true if it is connected and false if it is disconnected. The result will be displayed on the list.

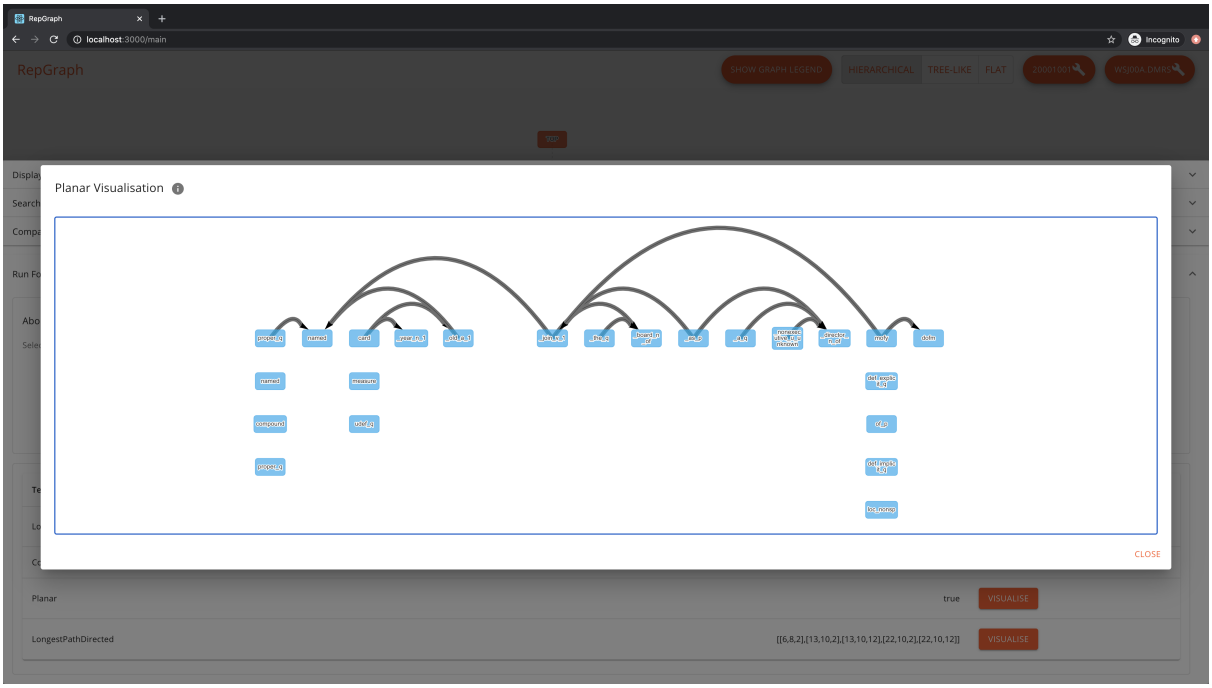
4.5.2. Planar

The planar test tells the user if the selected graph is planar or not. This test also provides the 4th visualisation format.

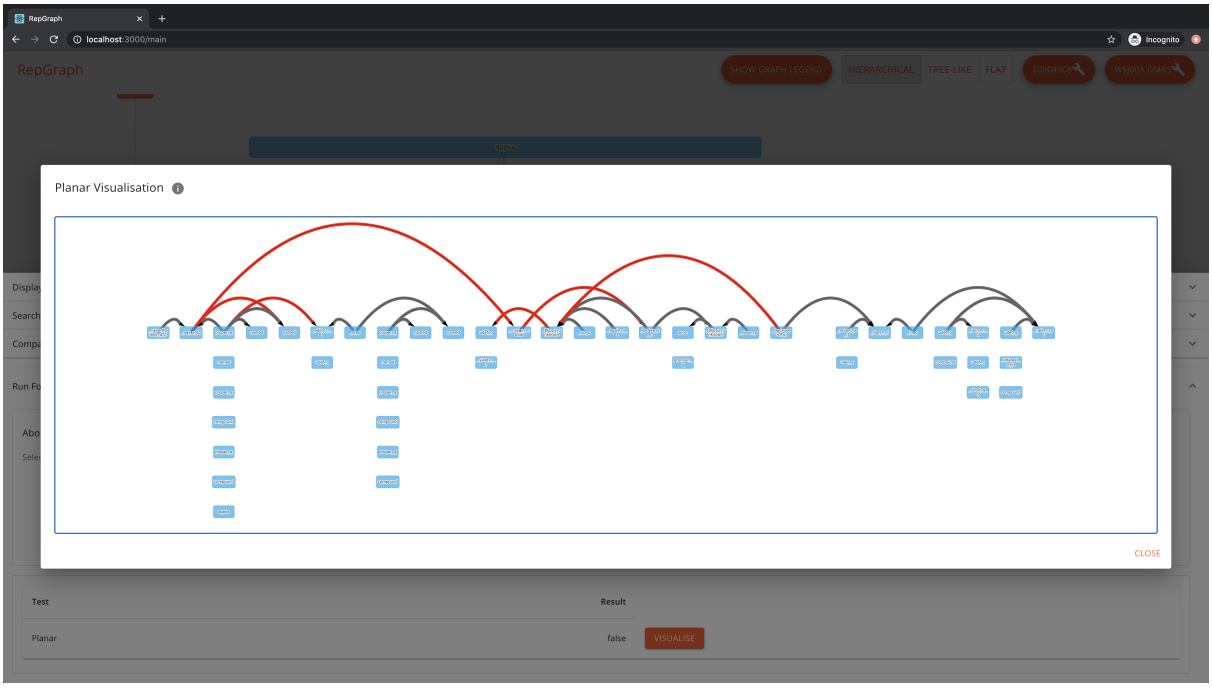
The test returns true if the selected graph is planar and false if it is not.

To see the 4th visualisation illustrating the planarity of the graph. The user must select the visualise button.

Below is an example of the planar visualisation for a graph that is planar. As can be seen below, all edges that obey the planarity requirements and are shown in black.



Below is an example of the planar visualisation for a graph that is not planar. As can be seen below, the edges that break the planarity property are highlighted in red.

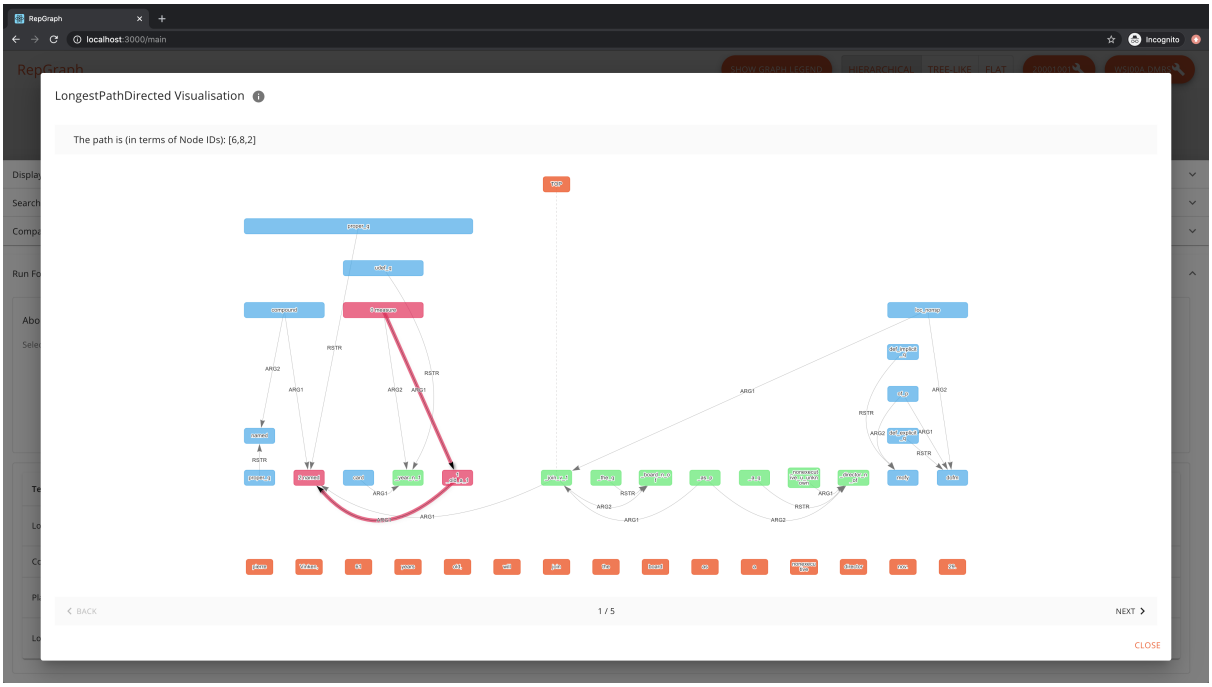


4.5.3. Longest Path

4.5.3.1. Directed

The directed longest path test shows the user the longest directed path either by text which has the node id's or by visually highlighting the nodes and edges of the path in a dialog when the user clicks the "visualise" button.

Once the button is clicked, a dialog will appear with the graph having it's nodes and edges highlighted to illustrate the longest directed path.



The user can also go through the multiple longest paths by making use of the “Next” and “Back” buttons

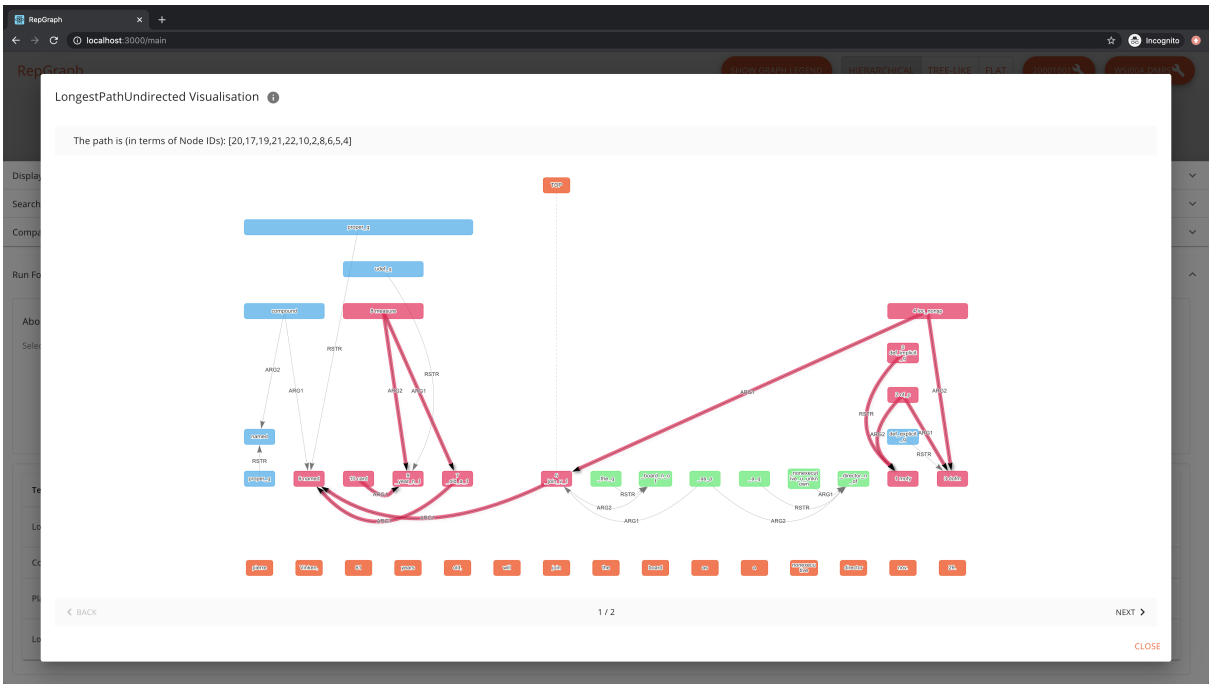
This test also acts as a directed cycle checker as if there is a cycle the longest path cannot be viewed and instead of a result the longest path value will say “Cycle Detected”.

LongestPathUndirected	"Cycle Detected"	VISUALISE

4.5.3.2. Undirected

The undirected longest path test acts in the same way as the directed path test except the edges directions do not matter.

LongestPathUndirected	[[20,17,19,21,22,10,2,8,6,5,4],[20,17,19,21,22,10,2,8,6,5,7]]	VISUALISE



The user can also go through the multiple longest paths by making use of the “Next” and “Back” buttons

This test also acts as an undirected cycle checker and will inform the user in the same way as the directed test.

5. API

The API has many methods that can be used in any front-end for analysing a MRG according to the DMRS format. The following are the Mappings, Parameters and Return objects of all API methods.

POST:

Request:	**Description:**
(“/UploadData”)	Request to send file to back-end and automatically parse the file sent line by line into graph objects to be stored in the model.

Parameters:	**Description:**
String - FileName	The name that the file will be saved as.

data	The multipart file data.

Return Type:	**Description:**
HashMap<String, Object>	Returns a hashmap of String keys and objects. The “response” key returns a value with a message of whether or not duplicates were found in the dataset and the “data” key returns a hashmap with the graph id being the key and the input of the graph being the value.

Request:	**Description:**
(“/UploadSingle”)	Request to send a single JSON graph object to the back-end to be stored in the model.

Parameters:	**Description:**
Graph - data	JSON of the graph object to be added to the model.

Return Type:	**Description:**
HashMap<String,String>	Returns a hashmap with the graph’s id and input.

GET:

Request:	**Description:**
(“/Visualise”)	Request the visualisation data of a certain graph in the model based on the graphID and format.

Parameters:	**Description:**
String - graphID	The ID of the graph requested.
Int - format	The format of the visualisation of the

	requested graph: ● Hierarchical - 1 ● Tree - 2 ● Flat - 3 ● Planar - 4

Return Type:	**Description:**
HashMap<String,Object>	Returns a JSON object of the visualisation data.

Request:	**Description:**
(“/DisplaySubset”)	Request to construct and display a subset of a graph based on a node’s adjacent nodes or its descendent nodes.

Parameters:	**Description:**
String - graphID	The ID of the graph requested.
Int - NodeID	The node ID to create the subset from
String - SubsetType	The way to construct the subset: ● Adjacent - construct the subset based on the adjacent nodes ● Descendent - construct the subset based on the node’s descendent nodes
Int - format	The format of the visualisation of the requested subset: ● Hierarchical - 1 ● Tree - 2 ● Flat - 3 ● Planar - 4

Return Type:	**Description:**
HashMap<String,Object>	Returns the JSON object data of the

	created subsets visualisation data

Request:	**Description:**
(“/SearchSubgraphNodeSet”)	Request a search of all the graphs in the model for the graphs containing a set of node labels.

Parameters:	**Description:**
List - labels	List of node labels to be searched for.

Return Type:	**Description:**
HashMap<String,Object>	Returns a JSON object of a method response and data of the search.

Request:	**Description:**
(“/SearchSubgraphPattern”)	Request a search of all the graphs in the model for the graphs containing a set of node labels.

Parameters:	**Description:**
String - graphID	The id of the graph containing the selected pattern.
int[] - NodeId	The nodes ids in the selected pattern.
int[] - EdgeIndices	The indices in the edge list of the selected edges.

Return Type:	**Description:**
HashMap<String,Object>	Returns a JSON object of a method response and data of the search.

Request:	**Description:**
(“/CompareGraphs”)	Request a comparison analysis of two graphs in the model.

Parameters:	**Description:**
String - graphID1	The first ID of the graph in the model requested to be compared.
String - graphID2	The second ID of the graph in the model requested to be compared.

Return Type:	**Description:**
HashMap<String, Object>	Returns a hashmap of String keys and Object values. The “SimilarNodes1” key gives a value of a list of node IDs that are similar in the first graph. The “SimilarNodes2” key gives a value of a list of node IDs that are similar in the second graph. The “SimilarEdges1” key gives a value of a list of edge indices that are similar in the first graph. The “SimilarEdges2” key gives a value of a list of edge indices that are similar in the second graph.

Request:	**Description:**
(“/TestGraph”)	Request formal tests to be conducted on a graph

Parameters:	**Description:**
String - graphID	The ID of the graph in the model requested to be tested.
Boolean - planar	Boolean to decide whether or not to run a planarity check on the requested graph

Boolean - longestPathDirected	Boolean to decide whether or not to run a the longest path directed algorithm and find the directed longest paths in the graph
Boolean - longestPathUndirected	Boolean to decide whether or not to run a the longest path undirected algorithm and find the undirected longest paths in the graph
Boolean - connected	Boolean to decide whether or not to run a connection check on the requested graph

Return Type:	**Description:**
HashMap<String, Object>	Returns a hashmap of String keys and Object values. Depending on which tests were run the return object can change. If all tests are run then these are the following keys: ● Planar - boolean ● PlanarVis - HashMap<String,Object> ● Connected - boolean ● LongestPathDirected - ArrayList<ArrayList<Integer>> ● LongestPathUndirected - ArrayList<ArrayList<Integer>>

Request:	**Description:**
(“/GetGraph”)	Request graph data of a graph in the model.

Parameters:	**Description:**
String - graphID	The ID of the graph in the model requested to be tested.

Return Type:	**Description:**
graph	Returns graph data of requested graph.

Request:	**Description:**
(“/GetSubset”)	Request graph data of a graph in the model.

Parameters:	**Description:**
String - graphID	The ID of the graph in the model requested to be tested.
int - NodeID	The node id of the head node of the subset.
String - SubsetType	The type of subset being created i.e “adjacent” or “descendent”

Return Type:	**Description:**
graph	Returns subset graph data of requested subset.