An iterative approach to finding correlation between different classes using Neo4j Cypher queries

Problem Statement: Given an ontology, with different classes linked to each other. Build a Cypher query to find **correlation** between different classes having various numerical instances, such that the **query iterates between two classes automatically, without manual user injection of class names**, and **without explicitly knowing path between any two of the classes**.

In my case, the ontology was structured like this:

:LaserSpeed :ns0__has_value :Speedvalue^^xsd:float.

:LaserPower :ns0__has_value :Powervalue^^sd:float.

:LaserSpeed :ns0__prescribes :Process.

:LaserPower :ns0__prescribes :Process.

:Process :ns0__has_output :Part.

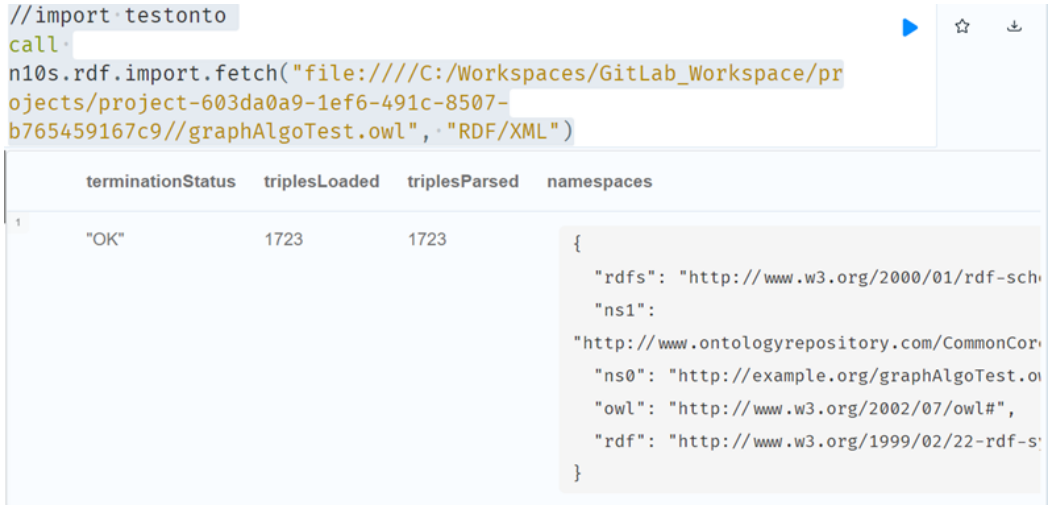
:Part :ns0__has_quality :Density.

:Density :ns0__has_value :densityvalue^^xsd:float.

1. **First, import the ontology:**

//import testonto

call n10s.rdf.import.fetch("file:////C:/Workspaces/GitLab_Workspace/projects/project-603da0a9-1ef6-491c-8507-b765459167c9//graphAlgoTest.owl", "RDF/XML")



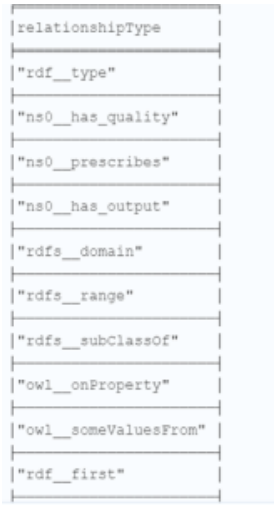
2. Next, find out the attribute which connects the classes having numerical values to the numerical value. In our case, it's ns0__has_value.

3. **Find out all the relationships existing in the given graph.**

CALL db.relationshipTypes()

YIELD relationshipType

RETURN relationshipType



relationshipType
"rdf__type"
"ns0__has_quality"
"ns0__prescribes"
"ns0__has_output"
"rdfs__domain"
"rdfs__range"
"rdfs__subClassOf"
"owl__onProperty"
"owl__someValuesFrom"
"rdf__first"

Select the relationships which contribute meaningfully to paths between classes. In our case, they are :ns0__prescribes, ns0__has_output, ns0__has_quality, ns0__has_value.

4. Run the query:

```
// Step 1: Query all instances with ns0__has_value

MATCH (inst)

WHERE inst.ns0__has_value IS NOT NULL

WITH collect(inst) AS instances


// Step 2: Build all possible combinations of instances

UNWIND instances AS inst_start

UNWIND instances AS inst_end

WITH inst_start, inst_end

WHERE id(inst_start) < id(inst_end) // Only unique pairs


// Step 3: Search the shortest path for each pair of instances

MATCH path = shortestPath((inst_start)-[:ns0__prescribes|ns0__has_output|ns0__has_quality|ns0__has_value*]-(inst_end))

WITH inst_start, inst_end, nodes(path) AS pathNodes, relationships(path) AS pathRels


// Step 4: Query the class of each instance in the path

UNWIND pathNodes AS instance

MATCH (instance)-[:rdf__type]->(class)

WITH inst_start, inst_end, pathNodes, pathRels, collect(DISTINCT class) AS classPath


// Step 5: Remove duplicated rows

WITH DISTINCT inst_start, inst_end, classPath, pathRels


// Step 6: Calculate Pearson correlation between classes by aggregating values from all instances of each class

// Aggregate all the values of instances for each class in the start and end

WITH classPath, collect(inst_start.ns0__has_value) AS startValues, collect(inst_end.ns0__has_value) AS endValues
```

// Calculate Pearson correlation based on the aggregated class values

RETURN classPath, gds.similarity.pearson(startValues, endValues) AS pearsonCorrelation

```
┌──────────────────────────────────────────────────────────────────────┬──────────────────┐
│classPath                                                             │pearsonCorrelation│
╞══════════════════════════════════════════════════════════════════════╪══════════════════╡
│[(:Resource {uri: "http://www.w3.org/2002/07/owl#NamedIndividual"}), (:│1.0000000000000002│
│Resource:owl__Class {uri: "http://example.org/graphAlgoTest.owl#LaserSp│                  │
│eed"}), (:Resource:owl__Class {uri: "http://example.org/graphAlgoTest.o│                  │
│wl#Process"}), (:Resource:owl__Class {uri: "http://example.org/graphAlg│                  │
│oTest.owl#Part"}), (:Resource:owl__Class {uri: "http://example.org/grap│                  │
│hAlgoTest.owl#Density"})]                                             │                  │
├──────────────────────────────────────────────────────────────────────┼──────────────────┤
│[(:Resource {uri: "http://www.w3.org/2002/07/owl#NamedIndividual"}), (:│-0.9593486455851578│
│Resource:owl__Class {uri: "http://example.org/graphAlgoTest.owl#LaserSp│                  │
│eed"}), (:Resource:owl__Class {uri: "http://example.org/graphAlgoTest.o│                  │
│wl#Process"}), (:Resource:owl__Class {uri: "http://example.org/graphAlg│                  │
│oTest.owl#LaserPower"})]                                              │                  │
├──────────────────────────────────────────────────────────────────────┼──────────────────┤
│[(:Resource:owl__Class {uri: "http://example.org/graphAlgoTest.owl#Lase│1.0000000000000002│
│rSpeed"}), (:Resource {uri: "http://www.w3.org/2002/07/owl#NamedIndivid│                  │
│ual"}), (:Resource:owl__Class {uri: "http://example.org/graphAlgoTest.o│                  │
│wl#Process"}), (:Resource:owl__Class {uri: "http://example.org/graphAlg│                  │
│oTest.owl#Part"}), (:Resource:owl__Class {uri: "http://example.org/grap│                  │
│hAlgoTest.owl#Density"})]                                             │                  │
└──────────────────────────────────────────────────────────────────────┴──────────────────┘
```

You will get duplicated values. You can eliminate the duplicated values by exporting the results and identifying the unique results.

Now, each step is described in detail:

Step1: **MATCH (inst)**

WHERE inst.ns0__has_value IS NOT NULL

WITH collect(inst) AS instances

RETURN instances

```
┌──────────────────────────────────────────────────────────────────────┐
│instances                                                             │
╞══════════════════════════════════════════════════════════════════════╡
│[(:Resource:owl__NamedIndividual:ns0__LaserSpeed {ns0__has_value: 142.│
│3353881422547,uri: "http://example.org/graphAlgoTest.owl#LaserSpeed_12│
│"}), (:Resource:owl__NamedIndividual:ns0__LaserSpeed {ns0__has_value: │
│133.03189848191548,uri: "http://example.org/graphAlgoTest.owl#LaserSpe│
│ed_11"}), (:Resource:owl__NamedIndividual:ns0__LaserSpeed {ns0__has_va│
│lue: 146.66439338954993,uri: "http://example.org/graphAlgoTest.owl#Las│
│erSpeed_14"}), (:Resource:owl__NamedIndividual:ns0__LaserSpeed {ns0__h│
│as_value: 187.43126261565476,uri: "http://example.org/graphAlgoTest.ow│
│l#LaserSpeed_13"}), (:Resource:owl__NamedIndividual:ns0__LaserSpeed {n│
│s0__has_value: 140.86114507799186,uri: "http://example.org/graphAlgoTe│
│st.owl#LaserSpeed_16"}), (:Resource:owl__NamedIndividual:ns0__LaserSpe│
│ed {ns0__has_value: 181.1256529454162,uri: "http://example.org/graphAl│
│goTest.owl#LaserSpeed_15"}), (:Resource:owl__NamedIndividual:ns0__Lase│
│rSpeed {ns0__has_value: 185.03953726732038,uri: "http://example.org/gr│
```

This step collects all instances of all classes where ns0__has_value **is not null**.

Step 2: Build all possible combinations of instances

UNWIND instances AS inst_start

UNWIND instances AS inst_end

WITH inst_start, inst_end

WHERE id(inst_start) < id(inst_end) // Only unique pairs

return id(inst_start),id(inst_end)

id(inst_start)	id(inst_end)
14	15
14	16
14	17
14	18
14	19
14	20
14	21

Here, we **unwind the collection in step 1 twice as as two lists, and combine each pair of instances, taking one from each each list, such that the two instances are not the same, e.g.,(A,A) , or transitive, for e.g, (A,B) and (B,A**).

Step 3: **Search the shortest path for each pair of instances**

MATCH path = shortestPath((inst_start)-[:ns0__prescribes|ns0__has_output|ns0__has_quality|ns0__has_value*]-(inst_end))

WITH inst_start, inst_end, nodes(path) AS pathNodes, relationships(path) AS pathRels

return inst_start, inst_end, pathNodes, pathRels

inst_start	inst_end	pathNodes	pathRels

(:Resource:owl__NamedIn
dividual:ns0__LaserSpee
d {ns0__has_value: 142.
3353881422547,uri: "htt
p://example.org/graphAl
goTest.owl#LaserSpeed_1
2"})
	(:Resource:owl__NamedIn
	dividual:ns0__Density {
	ns0__has_value: 3.84670
	77628450936,uri: "http:
	//example.org/graphAlgo
	Test.owl#Density_12"})
		[(:Resource:owl__NamedI
		ndividual:ns0__LaserSpe
		ed {ns0__has_value: 142
		.3353881422547,uri: "ht
		tp://example.org/graphA
		lgoTest.owl#LaserSpeed_
		12"}), (:Resource:owl__
		NamedIndividual:ns0__Pr
		ocess {uri: "http://exa
		mple.org/graphAlgoTest.
		owl#Process_12"}), (:Re
		source:owl__NamedIndivi
			[[:ns0__prescribes], [:
			ns0__has_output], [:ns0
			__has_quality]]

Here, for each of the pairs of instances built in step 2, we t**raverse the shortest path from instance 1 to instance 2, constrained by the condition that they travel through only one or more of the meaningful relationships: ns0__prescribes,ns0__has_output,ns0__has_quality,ns0__has_value**. The returned values are the starting instance, the ending instance, the nodes traversed in the shortest route between the instances through the relationships, and the relationships traversed.

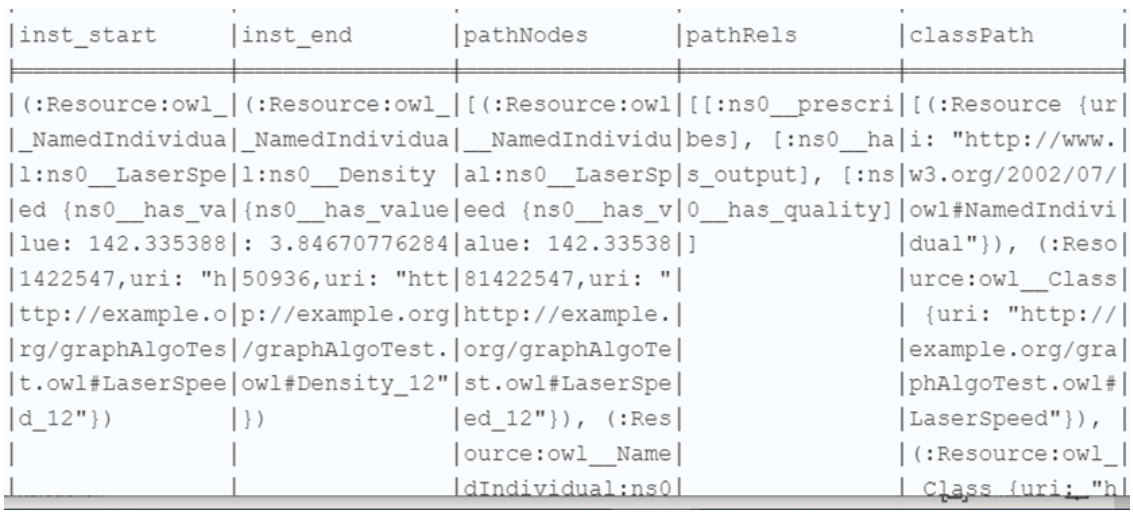
Step 4: Query the class of each instance in the path

UNWIND pathNodes AS instance

MATCH (instance)-[:rdf__type]->(class)

WITH inst_start, inst_end, pathNodes, pathRels, collect(DISTINCT class) AS classPath

return inst_start, inst_end, pathNodes,pathRels, classPath



inst_start	inst_end	pathNodes	pathRels	classPath
(:Resource:owl_ _NamedIndividual:ns0__LaserSpeed {ns0__has_value: 142.335388,uri: "http://example.org/graphAlgoTest.owl#LaserSpeed_12"})	(:Resource:owl_ _NamedIndividual:ns0__Density {ns0__has_value: 3.84670776284,uri: "http://example.org/graphAlgoTest.owl#Density_12"})	[(:Resource:owl__NamedIndividual:ns0__LaserSpeed {ns0__has_value: 142.33538,uri: "http://example.org/graphAlgoTest.owl#LaserSpeed_12"}), (:Resource:owl__Name dIndividual:ns0	[[:ns0__prescribes], [:ns0__has_output], [:ns0__has_quality]]	[(:Resource {uri: "http://www.w3.org/2002/07/owl#NamedIndividual"}), (:Resource:owl__Class {uri: "http://example.org/graphAlgoTest.owl#LaserSpeed"}), (:Resource:owl_ Class {uri: "h

Here, we **map the instances of the nodes through which the shortest path is traversed for each pair of starting and ending instances to their classes**.

Step 5: Remove duplicated rows

return DISTINCT classPath

classPath
[(:Resource {uri: "http://www.w3.org/2002/07/owl#NamedIndividual"}), (:Resource:owl__Class {uri: "http://example.org/graphAlgoTest.owl#LaserSpeed"}), (:Resource:owl__Class {uri: "http://example.org/graphAlgoTest.owl#Process"}), (:Resource:owl__Class {uri: "http://example.org/graphAlgoTest.owl#Part"}), (:Resource:owl__Class {uri: "http://example.org/graphAlgoTest.owl#Density"})]
[(:Resource {uri: "http://www.w3.org/2002/07/owl#NamedIndividual"}), (:Resource:owl__Class {uri: "http://example.org/graphAlgoTest.owl#LaserSpeed"}), (:Resource:owl__Class {uri: "http://example.org/graphAlgoTest.owl#Process"}), (:Resource:owl__Class {uri: "http://example.org/graphAlgoTest.owl#LaserPower"})]

Step 5 returns the distinct paths, i.e., **the distinct classes that need to be travelled starting from one class to another, in order to find the Pearson coefficient for correlation**.

return **DISTINCT pathRels**

We also look for **distinct relationships traversed via the shortest paths**:

pathRels
[[:ns0__prescribes], [:ns0__has_output], [:ns0__has_quality]]
[[:ns0__prescribes], [:ns0__prescribes]]
[[:ns0__prescribes], [:ns0__has_output], [:ns0__has_quality]]
[[:ns0__prescribes], [:ns0__prescribes]]
[[:ns0__prescribes], [:ns0__has_output], [:ns0__has_quality]]

Step 6: **Calculate Pearson correlation between classes by aggregating values from all instances of each class**

// Aggregate all the values of instances for each class in the start and end

WITH classPath, collect(inst_start.ns0__has_value) AS startValues, collect(inst_end.ns0__has_value) AS endValues

// Calculate Pearson correlation based on the aggregated class values

RETURN classPath, gds.similarity.pearson(startValues, endValues) AS pearsonCorrelation

The last step collects the values of **all instances of one class as startValues, and the corresponding values of the second class as endValues, for each pair of classes, and finds the Pearson coefficient between them using GDS's Pearson similarity function**.