Section 1:

For my graph I wanted to consider college courses in a university. The dropout rate among first year students in Irish colleges fluctuates at around 15% depending on the year. For many STEM courses, dropout rates are hitting as high as 80%.

I considered a new model for a college/ university which allowed students to switch courses in first year. This would hopefully allow students who are not satisfied with their initial course, instead of completely dropping out, they could instead switch to a similar course, but hopefully more engaging to them.

After thinking about this further, I felt it would not make sense, or be practical, to allow students to switch to whatever course they wanted. They would have to switch to a course which had some form of similarity to their current course.

I developed a dataset roughly based on undergraduate courses in DCU. Not every course DCU offers is in my dataset, but it does give rough estimates as to

1. A variety of different course in the different schools (computing, business etc)
2. The number of students in those courses

The idea of this is it can be scaled up or down depending on the size of the college, while the proportion of students in each domain (e.g business, computing, education etc.) stays roughly the same.

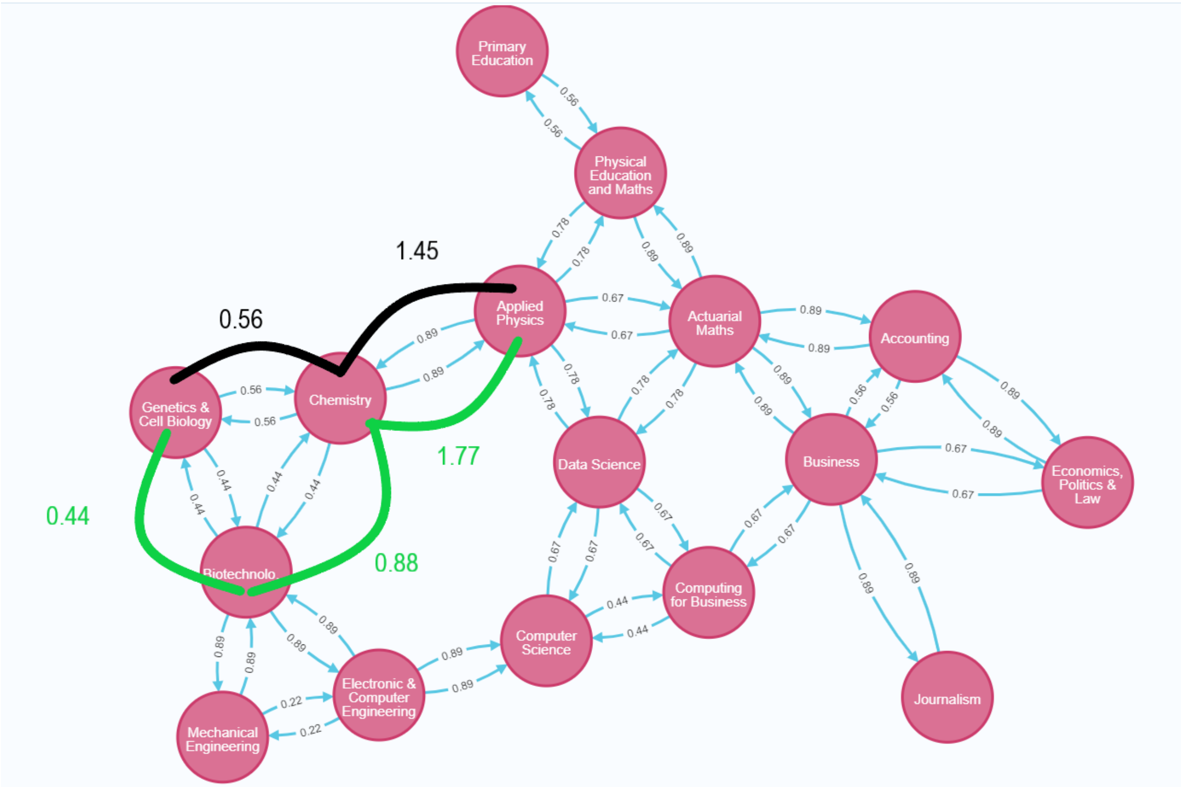
For my idea, I needed some measure of the similarity between courses. I wanted courses with a lot of modules in common to have a low value for their relationship, while courses with very few modules in common to have a high value for their relationship. I decided to use **dissimilarity** for the relationships, with my reasoning explained in an example below.

Computer Science	Data Science
Programming 1	Programming 1
Programming 2	Programming 2
Web Design	Calculus
IT Maths 1	Probability
DIME	DIME
Computer Systems	Collaboration and Innovation
IT Maths 2	Linear Maths 1
Networks & Internet	Linear Maths 2
Introduction to Operating Systems	Data Science & Databases

These courses have 3 modules out of 9 in common. I wanted courses that had a lot of modules in common to have a low value for their relationship and courses that do not have a lot of modules in common to have a high value for their relationship. That is why I used dissimilarity as opposed to similarity, so when applying algorithms like shortest path, it will find the “most similar courses”. (ie low relationship -> more similar, high relationship -> less similar).

If courses do not have any modules in common, there is no relationship between those courses.

The context of my graph is a college which allows students to change courses in first year, but only to courses which have at least one module in common with their current course. They could then, theoretically, “hop” between courses to get to their desired course, even if their current course does not have any modules in common with their preferred course.



For example, a student in “Genetics & Cell Biology” decides they want to do “Applied Physics”. These two courses do not have any modules in common, so they have to go through other courses to get there.

The first path is “Genetics & Cell Biology” -> “Chemistry” -> “Applied Physics”, with a sum of the weights of 1.45.

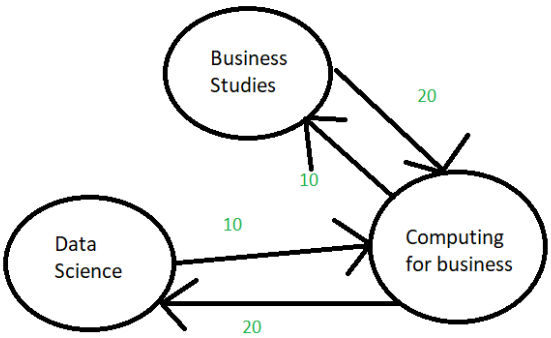
Alternatively, they could go “Genetics & Cell Biology” -> “Biotechnology” -> “Chemistry” -> “Applied Physics”, with a sum of the weights of 1.77.

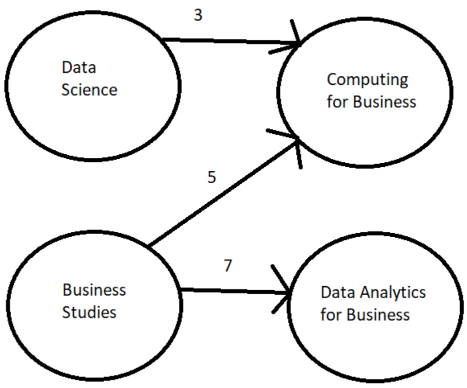
Because the first path has “less dissimilarity”, i.e. more similar, it would make more sense for the student to go through the first path rather than the second.

The problems which I plan to address with this graph are;

1. Are there “bottleneck” courses? There may be a few courses that are on the only path between two parts of the graph. This may indicate that there is demand for a new course.

In the figure below, the relationship represents the number of students passing between “Data Science” and “Business Studies”. The fact that there are a lot of students switching between those courses may indicate that there is an opportunity for another course, for example, “Data Analytics for Business”, “Fintech” or something similar. This could reduce the number of people changing course.

<table>
<tr><td>Before adding new course - Total number of course transfers = 60</td><td>After adding new course - Total number of course transfers = 15</td></tr>
<tr><td>


</td><td>


</td></tr>
</table>

2. Should a maximum limit on course changes be introduced? Ie, should there be a maximum dissimilarity measure, ie, a student can change course as many times as they like, but the total dissimilarity sum must be less than 2. What effect would these limits have?
3. Running community detection algorithms should indicate communities for the different faculties and schools. (ie, a community may be the school of computing). Is this the case? Are there courses which are close to 2 different communities? Does that make sense in context? (e.g. computing for business should be close to the community for computing courses and the community for business courses)
4. Running simulations to test how many people are switching between the different courses taking into account
    a) The number of people in each course
    b) The dissimilarity between the courses

    to measure a real world application.

Section 2

To create the nodes, I used the following cypher query. Each node is a course, with attribute “students”, the number of students in the course.

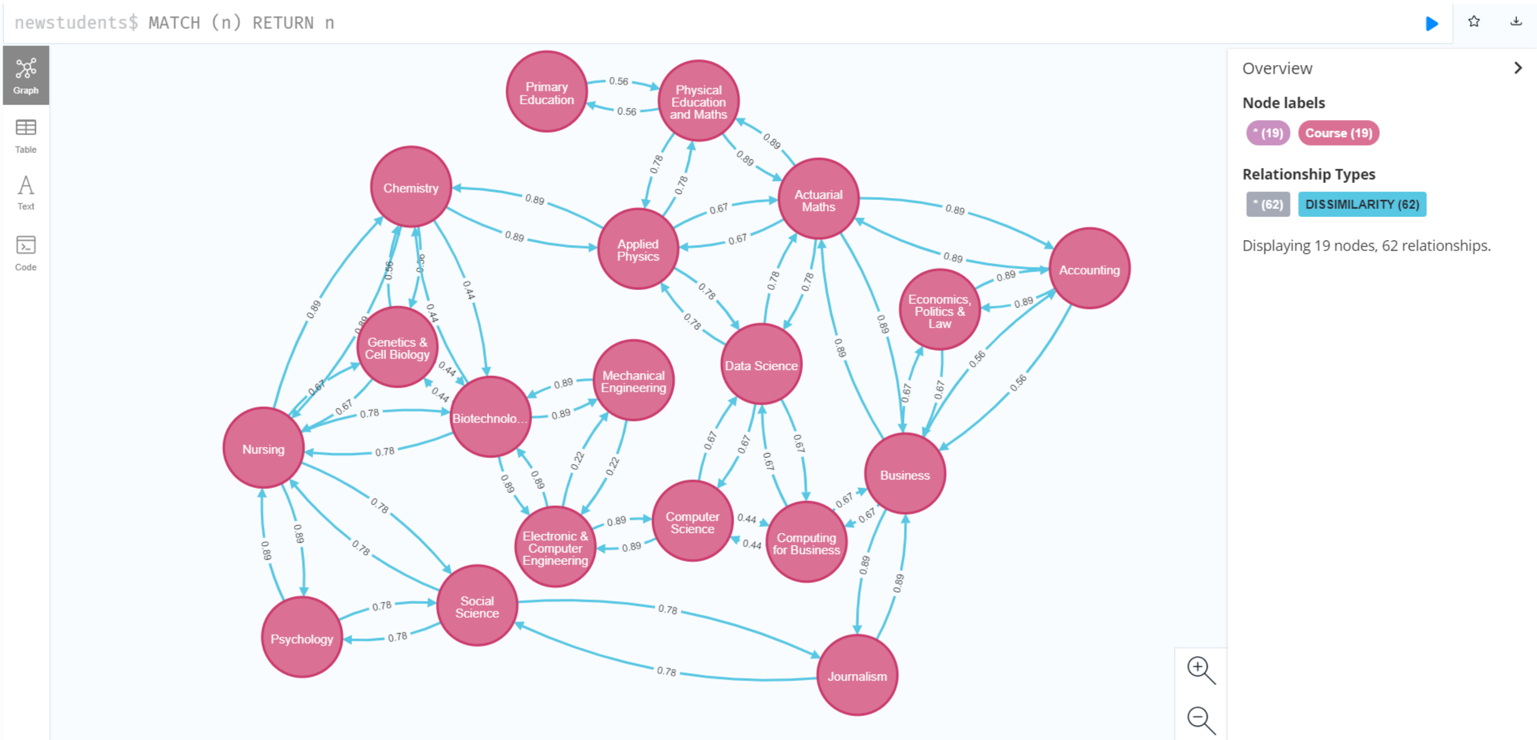
```
CREATE (cs:Course {name: "Computer Science", students: 120}), (ds:Course {name: "Data Science",
students: 20}), (cb:Course {name: "Computing for Business", students: 30}),
(a:Course {name: "Electronic & Computer Engineering", students: 50}), (b:Course {name: "Mechanical
Engineering", students: 30}), (c:Course {name: "Business", students: 100}),
(d:Course {name: "Accounting", students: 30}), (e:Course {name: "Applied Physics", students: 20}), (f:Course
{name: "Actuarial Maths", students: 50}),
(g:Course {name: "Economics, Politics & Law", students: 60}), (h:Course {name: "Primary Education",
students: 120}), (i:Course {name: "Physical Education and Maths", students: 30}),
(j:Course {name: "Chemistry", students: 40}), (k:Course {name: "Biotechnology", students: 30}), (l:Course
{name: "Genetics & Cell Biology", students:30}),
(m:Course {name: "Psychology", students: 50}), (n:Course {name: "Nursing", students:80}), (o:Course {name:
"Social Science", students: 40}), (p:Course {name: "Journalism", students: 50})
```

The following cypher query creates the dissimilarity relationships between the different courses.

```
MATCH (ds:Course {name:"Data Science"}), (ec:Course {name:"Computing for Business"}), (cs:Course
{name:"Computer Science"}),
(a:Course {name: "Electronic & Computer Engineering", students: 50}), (b:Course {name: "Mechanical
Engineering", students: 30}), (c:Course {name: "Business", students: 100}),
(d:Course {name: "Accounting", students: 30}), (e:Course {name: "Applied Physics", students: 20}), (f:Course
{name: "Actuarial Maths", students: 50}),
(g:Course {name: "Economics, Politics & Law", students: 60}), (h:Course {name: "Primary Education",
students: 120}), (i:Course {name: "Physical Education and Maths", students: 30}),
(j:Course {name: "Chemistry", students: 40}), (k:Course {name: "Biotechnology", students: 30}), (l:Course
{name: "Genetics & Cell Biology", students:30}),
(m:Course {name: "Psychology", students: 50}), (n:Course {name: "Nursing", students:80}), (o:Course {name:
"Social Science", students: 40}), (p:Course {name: "Journalism", students: 50})
CREATE
(cs)-[:DISSIMILARITY {percent:0.67}]->(ds)
(cs)-[:DISSIMILARITY {percent:0.44}]->(ec)
(ds)-[:DISSIMIILARITY{percent:0.67}]->(ec)
(a)-[:DISSIMILARITY {percent:0.89}]->(cs),
(ds)-[:DISSIMILARITY {percent:0.78}]->(e),
(ds)-[:DISSIMILARITY {percent:0.78}]->(f),
(ec)-[:DISSIMILARITY {percent:0.67}]->(c),
(a)-[:DISSIMILARITY {percent:0.22}]->(b),
(a)-[:DISSIMILARITY {percent:0.89}]->(k),
(b)-[:DISSIMILARITY {percent:0.89}]->(k),
(c)-[:DISSIMILARITY {percent:0.56}]->(d),
(c)-[:DISSIMILARITY {percent:0.89}]->(f),
(c)-[:DISSIMILARITY {percent:0.67}]->(g),
(c)-[:DISSIMILARITY {percent:0.89}]->(p),
(d)-[:DISSIMILARITY {percent:0.89}]->(f),
(d)-[:DISSIMILARITY {percent:0.89}]->(g),
(c)-[:DISSIMILARITY {percent:0.89}]->(o),
(e)-[:DISSIMILARITY {percent:0.67}]->(f),
(e)-[:DISSIMILARITY {percent:0.78}]->(i),
(f)-[:DISSIMILARITY {percent:0.89}]->(i),
(g)-[:DISSIMILARITY {percent:0.89}]->(o),
(h)-[:DISSIMILARITY {percent:0.56}]->(i),
(h)-[:DISSIMILARITY {percent:0.89}]->(o),
(j)-[:DISSIMILARITY {percent:0.44}]->(k),
(j)-[:DISSIMILARITY {percent:0.56}]->(l),
(j)-[:DISSIMILARITY {percent:0.89}]->(n),
(j)-[:DISSIMILARITY {percent:0.89}]->(e),
(k)-[:DISSIMILARITY {percent:0.44}]->(l),
(k)-[:DISSIMILARITY {percent:0.78}]->(n),
(l)-[:DISSIMILARITY {percent:0.67}]->(n),
(m)-[:DISSIMILARITY {percent:0.78}]->(o),
(m)-[:DISSIMILARITY {percent:0.89}]->(n),
(n)-[:DISSIMILARITY {percent:0.78}]->(o),
(o)-[:DISSIMILARITY {percent:0.78}]->(p)
```

This command was repeated, but with the courses reversed.

The graph I created is shown below,



The graph is connected, so without any restrictions on the number of hops or dissimilarity, students could in theory go to any course.

Section 3

1 - Implementing pagerank Courses

Trying to determine important nodes usually involves applying centrality algorithms. It is important to consider what is ‘central’ in the context of a graph, when choosing a centrality algorithm. In my graph, important courses are those which have a lot of incoming relationships, but also, the importance of the courses to which they are related. This is because you can hop between multiple courses, so being connected to one course which has a lot of relationships is more important than being connected to a course with very few relationships.

Therefore, pagerank is the best algorithm to use, as it takes into account the number of incoming relationships and the importance of those courses. The code is shown below

```
CALL gds.graph.create(
  'myGraph1',
  'Course',
  'DISSIMILARITY',
  {
    relationshipProperties: 'percent'
```

```
  }
)
CALL gds.pageRank.stream('myGraph1')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC, name ASC
```

Pagerank

name	score
Business	1.435055
Actuarial Maths	1.397468
Nursing	1.384527
Biotechnology	1.371848
Applied Physics	1.129528
Data Science	1.120123
Chemistry	1.090938
Physical Education and Maths	0.985578
Social Science	0.932901
Accounting	0.900518
Electronic & Computer Engineering	0.898425
Computer Science	0.885894
Computing for Business	0.877632
Genetics & Cell Biology	0.84508
Journalism	0.654715
Psychology	0.646138
Economics, Politics & Law	0.645529
Mechanical Engineering	0.634215
Primary Education	0.427458

2 - Max limit on course swaps

I found the shortest path from one each node, to every other node in the graph. I then set three values for the maximum shortest path, 1.5, 2 and 2.5. The cypher code is shown below, for less than 2 dissimilarity away.

```
// Calls shortest path on all nodes, and shows relationships nodes less than
// 2 away
CALL gds.alpha.allShortestPaths.stream({
```

```
  nodeProjection: 'Course',
  relationshipProjection: {
    ROAD: {
      type: 'DISSIMILARITY',
      properties: 'percent'
    }
  },
  relationshipWeightProperty: 'percent'
})
YIELD sourceNodeId, targetNodeId, distance
WITH sourceNodeId, targetNodeId, distance
WHERE gds.util.isFinite(distance) = true

MATCH (source:Course) WHERE id(source) = sourceNodeId
MATCH (target:Course) WHERE id(target) = targetNodeId
WITH source, target, distance WHERE source <> target AND distance < 2

RETURN source.name AS source, target.name AS target, distance
ORDER BY source, target ASC
```

The resulting table is shown below

source	target	distance
Accounting	Actuarial Maths	0.89
Accounting	Applied Physics	1.56
Accounting	Business	0.56
Accounting	Computer Science	1.67
Accounting	Computing for Business	1.23
Accounting	Data Science	1.67
Accounting	Economics, Politics & Law	0.89
Accounting	Journalism	1.45
Accounting	Physical Education and Maths	1.78
Actuarial Maths	Accounting	0.89
. . . .		

I stored this as a pandas dataframe, and ran the following python code.

```
df = pd.read_csv("less_than_2.csv")

courses = list(df.loc[:,"source"])

# Inclusive
d = {}
```

```
i = 0
while i < len(courses):
  if courses[i] not in d.keys():
    if i != 0:
      d[courses[i - 1]].append(i - 1)
    d[courses[i]] = [i]
  i += 1

d[courses[i - 1]].append(len(courses) - 1)

df_v_1 = pd.DataFrame(columns=["Course","Possible Courses","Number of Possible Courses", "Average
Distance to Possible Courses"])

for course in d.keys():
  start, end = d[course][0], d[course][1]
  a = df.loc[start:end,"target"]
  b = df.loc[start:end, "distance"]
  tmp = pd.DataFrame(data = [[course, ", ".join(list(a)), len(a), b.mean()]],columns=["Course","Possible
Courses","Number of Possible Courses", "Average Distance to Possible Courses"])
  df_v_1 = df_v_1.append(tmp,ignore_index=True)
```

```
df_v_1
```

	Course	Possible Courses	Number of Possible Courses	Average Distance to Possible Courses
0	Accounting	Actuarial Maths, Applied Physics, Business, Co...	9	1.300000
1	Actuarial Maths	Accounting, Applied Physics, Business, Chemist...	11	1.215455
2	Applied Physics	Accounting, Actuarial Maths, Biotechnology, Bu...	12	1.253333
3	Biotechnology	Applied Physics, Chemistry, Computer Science, ...	9	1.086667
4	Business	Accounting, Actuarial Maths, Applied Physics, ...	10	1.114000
5	Chemistry	Actuarial Maths, Applied Physics, Biotechnolog...	11	1.253636
6	Computer Science	Accounting, Actuarial Maths, Applied Physics, ...	10	1.235000
7	Computing for Business	Accounting, Actuarial Maths, Applied Physics, ...	10	1.169000
8	Data Science	Accounting, Actuarial Maths, Applied Physics, ...	10	1.248000
9	Economics, Politics & Law	Accounting, Actuarial Maths, Business, Compute...	6	1.300000
10	Electronic & Computer Engineering	Biotechnology, Chemistry, Computer Science, Co...	8	1.152500
11	Genetics & Cell Biology	Applied Physics, Biotechnology, Chemistry, Ele...	8	1.098750
12	Journalism	Accounting, Actuarial Maths, Business, Computi...	8	1.392500
13	Mechanical Engineering	Biotechnology, Chemistry, Computer Science, Co...	8	1.235000
14	Nursing	Applied Physics, Biotechnology, Chemistry, Ele...	9	1.187778
15	Physical Education and Maths	Accounting, Actuarial Maths, Applied Physics, ...	7	1.288571
16	Primary Education	Actuarial Maths, Applied Physics, Physical Edu...	3	1.116667
17	Psychology	Biotechnology, Chemistry, Genetics & Cell Biol...	6	1.373333
18	Social Science	Biotechnology, Business, Chemistry, Genetics &...	7	1.241429

I repeated the steps above, changing the maximum values to 1.5 and 2. The overall analysis of the results is given below

Maximum dissimilarity	Minimum possible courses	Maximum possible courses	Average possible courses	Average distance
1.5	2	9	5.368	0.956
2	3	12	8.526	1.224
2.5	7	17	12.737	1.574

3 - Simulating with < 1.5 dissimilarity

Based on the max possible limit results, I decided to try and simulate a year, assuming 10% of students in each course decide to switch to a different course. It is randomly decided which course the outgoing students will go to, implementing python's random.choice() function. The code is shown below.

```
import random

df_v_2["Start Students"] = [30,50,20,30,100,40,120,30,20,60,50,30,50,30,80,30,120,50,40]
df_v_2["Outgoing Students"] = (df_v_2["Start Students"] * 0.1).astype(int) # Assuming 10% leave
df_v_2["Incoming Students"] = 0

outgoing_courses = {}

for index, row in df_v_2.iterrows():
  outgoing = row['Outgoing Students']
  outgoing_courses[row['Course']] = {}

  start, end = d2[row['Course']][0], d2[row['Course']][1]
  possible_courses = list(df2.loc[start:end,"target"])
  for course in possible_courses:
    outgoing_courses[row['Course']][course] = 0

  i = 0
  while i < outgoing:
    student_new_course = random.choice(possible_courses)
    df_v_2.loc[df_v_2["Course"] == student_new_course,"Incoming Students"] = df_v_2.loc[df_v_2["Course"] ==
student_new_course,"Incoming Students"] + 1
    outgoing_courses[row['Course']][student_new_course] += 1
    i += 1

df_v_2['End Students'] = df_v_2['Start Students'] - df_v_2['Outgoing Students'] + df_v_2['Incoming Students']
```

	Course	Possible Courses	Number of Possible Courses	Average Distance to Possible Courses	Start Students	Outgoing Students	Incoming Students	End Students
0	Accounting	Actuarial Maths, Business, Computing for Busin...	5	1.004000	30	3	5	32
1	Actuarial Maths	Accounting, Applied Physics, Business, Compute...	8	1.058750	50	5	8	53
2	Applied Physics	Actuarial Maths, Biotechnology, Chemistry, Com...	9	1.126667	20	2	10	28
3	Biotechnology	Applied Physics, Chemistry, Electronic & Compu...	6	0.795000	30	3	5	32
4	Business	Accounting, Actuarial Maths, Computer Science,...	7	0.875714	100	10	6	96
5	Chemistry	Applied Physics, Biotechnology, Electronic & C...	6	0.906667	40	4	4	40
6	Computer Science	Actuarial Maths, Applied Physics, Business, Co...	7	1.017143	120	12	1	109
7	Computing for Business	Accounting, Actuarial Maths, Applied Physics, ...	8	1.072500	30	3	10	37
8	Data Science	Actuarial Maths, Applied Physics, Business, Co...	5	0.848000	20	2	5	23
9	Economics, Politics & Law	Accounting, Business, Computing for Business	3	0.966667	60	6	2	56
10	Electronic & Computer Engineering	Biotechnology, Chemistry, Computer Science, Co...	6	0.998333	50	5	4	49
11	Genetics & Cell Biology	Applied Physics, Biotechnology, Chemistry, Ele...	7	1.032857	30	3	4	31
12	Journalism	Accounting, Business, Social Science	3	1.040000	50	5	6	51
13	Mechanical Engineering	Biotechnology, Chemistry, Computer Science, El...	5	0.976000	30	3	2	29
14	Nursing	Biotechnology, Chemistry, Genetics & Cell Biol...	5	0.802000	80	8	7	79
15	Physical Education and Maths	Actuarial Maths, Applied Physics, Primary Educ...	3	0.743333	30	3	5	32
16	Primary Education	Actuarial Maths, Applied Physics, Physical Edu...	3	1.116667	120	12	3	111
17	Psychology	Nursing, Social Science	2	0.835000	50	5	4	49
18	Social Science	Genetics & Cell Biology, Journalism, Nursing, ...	4	0.947500	40	4	7	43

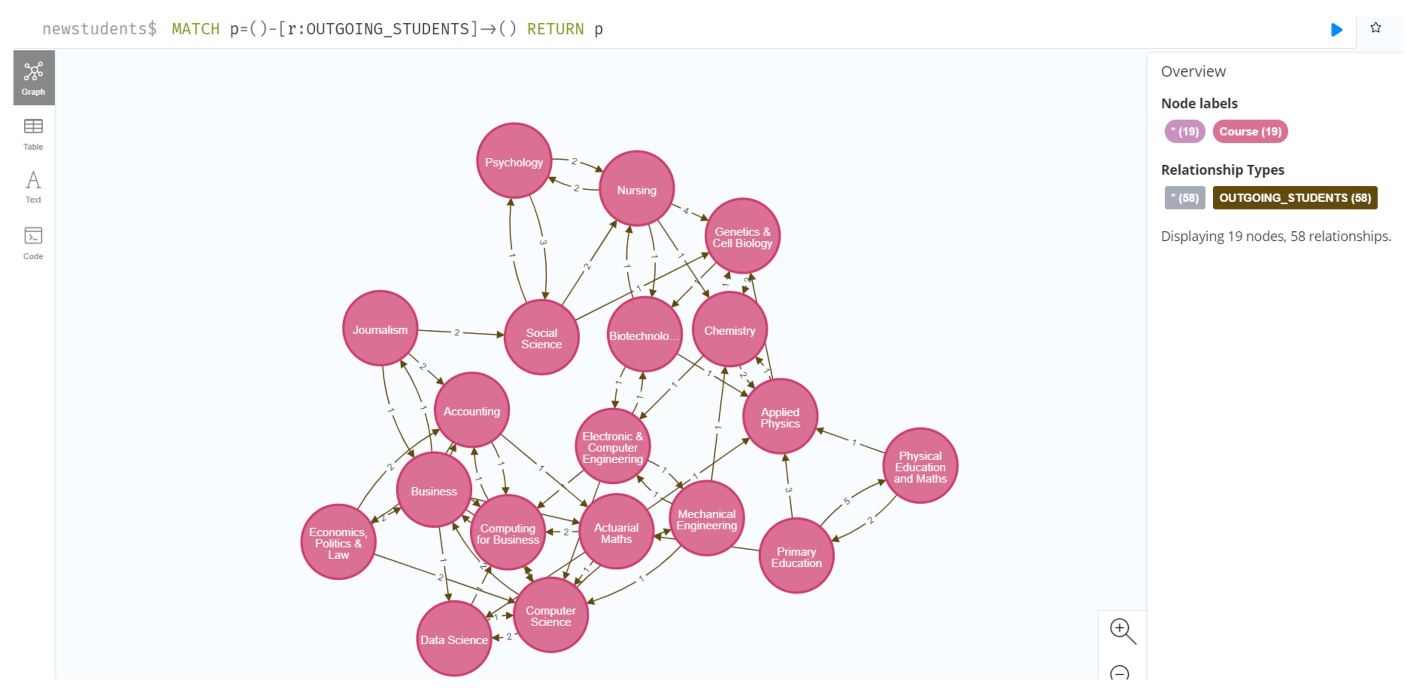
The outgoing students' destination courses are stored in a dictionary.

```
{'Accounting': {'Actuarial Maths': 1,
  'Business': 0,
  'Computing for Business': 1,
```

```
 'Economics, Politics & Law': 1,
 'Journalism': 0},
'Actuarial Maths': {'Accounting': 0,
 'Applied Physics': 1,
 'Business': 0,
 'Computer Science': 1,
 'Computing for Business': 2,
 'Data Science': 1,
 'Physical Education and Maths': 0,
 'Primary Education': 0},
          .
          .
          .
}
```

I used these values to create a new relationship, showing the actual changes in the number of students in the first year of college.

```
// Creating new students outgoing relationships
MATCH (ds:Course {name:"Data Science"}), (ec:Course {name:"Computing for Business"}), (cs:Course {name:"Computer Science"}),
(a:Course {name: "Electronic & Computer Engineering", students: 50}), (b:Course {name: "Mechanical Engineering", students: 30}), (c:Course {name: "Business", students: 100}),
(d:Course {name: "Accounting", students: 30}), (e:Course {name: "Applied Physics", students: 20}), (f:Course {name: "Actuarial Maths", students: 50}),
(g:Course {name: "Economics, Politics & Law", students: 60}), (h:Course {name: "Primary Education", students: 120}), (i:Course {name: "Physical Education and Maths", students: 30}),
(j:Course {name: "Chemistry", students: 40}), (k:Course {name: "Biotechnology", students: 30}), (l:Course {name: "Genetics & Cell Biology", students:30}),
(m:Course {name: "Psychology", students: 50}), (n:Course {name: "Nursing", students:80}), (o:Course {name: "Social Science", students: 40}), (p:Course {name: "Journalism", students: 50})
CREATE
(d)-[:OUTGOING_STUDENTS{no_of_students:1}]->(f),
(d)-[:OUTGOING_STUDENTS{no_of_students:1}]->(ec),
(d)-[:OUTGOING_STUDENTS{no_of_students:1}]->(g),
(f)-[:OUTGOING_STUDENTS{no_of_students:1}]->(e),
(f)-[:OUTGOING_STUDENTS{no_of_students:1}]->(cs),
(f)-[:OUTGOING_STUDENTS{no_of_students:2}]->(ec),
          .
          .
          .
```



4 - Implementing degree centrality on the new graph

I believe the structure of this graph provides a new, more accurate measure of centrality. Unlike in 1, this graph shows a real world example of students swapping courses, instead of the theoretical courses they could potentially swap to.

One key difference between the graphs is, unlike with the first graph, the destination course of the students is known, so using the courses as links is not necessary. Therefore, pagerank is no longer suited, as the idea of a ‘random surfer’ traversing the courses is not a useful measure of centrality.

Degree centrality is the best measure of centrality for this graph, as it implements the number of incoming and outgoing students to find the most central courses, which people typically swap to. I’m using weighted degree centrality.

```
CALL gds.graph.create(
  'myGraph2',
  'Course',
  {
    OUTGOING_STUDENTS: {
      orientation:'UNDIRECTED',
      properties: ['no_of_students']
    }
  }
)


CALL gds.degree.stream(
   'myGraph2',
   { relationshipWeightProperty: 'no_of_students' }
)
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC, name DESC
```

Degree Centrality

name	score
Computer Science	20
Business	17
Actuarial Maths	15
Primary Education	14
Nursing	13
Computing for Business	12

Genetics & Cell Biology	10
Applied Physics	10
Social Science	9
Mechanical Engineering	9
Chemistry	9
Accounting	9
Psychology	8
Physical Education and Maths	8
Electronic & Computer Engineering	8
Economics, Politics & Law	7
Journalism	6
Data Science	6
Biotechnology	6

5 - Community Detection

I ran Louvain algorithm on the initial graph, to try and find communities. This should show communities of schools of related courses.

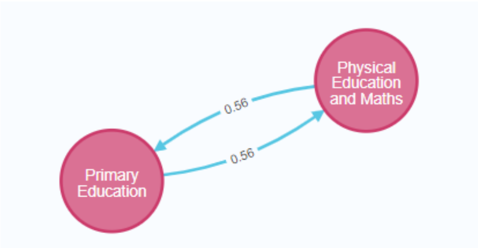
```
CALL gds.graph.create(
    'myGraph',
    'Course',
    {
        DISSIMILARITY: {
            orientation: 'UNDIRECTED'
        }
    },
    {
        nodeProperties: 'students',
        relationshipProperties: 'percent'
    }
)

CALL gds.louvain.stream('myGraph')
YIELD nodeId, communityId, intermediateCommunityIds
RETURN gds.util.asNode(nodeId).name AS name, communityId, intermediateCommunityIds
ORDER BY name ASC
```

name	communityId	intermediateCommunityIds
Accounting	9	null

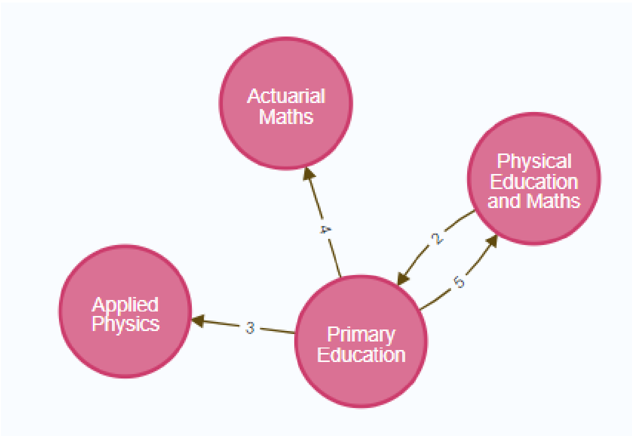
Actuarial Maths	11	null
Applied Physics	11	null
Biotechnology	12	null
Business	9	null
Chemistry	12	null
Computer Science	2	null
Computing for Business	2	null
Data Science	2	null
. . .		

Section 4

Comparing pagerank from first graph to degree centrality of second graph



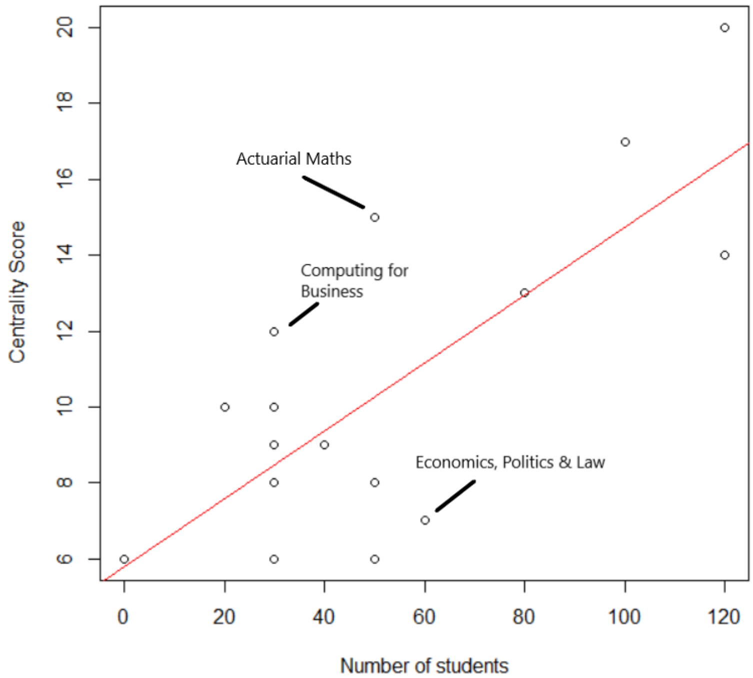
The most interesting point to note when comparing the pagerank scores and the degree scores of the courses, is that "Primary Education" is the lowest ranked course using pagerank, while it is the 4th highest ranked score using degree centrality.



The reason primary education is the lowest rank in pagerank is because it only takes into account the number and quality of links going into the course. In the original graph using dissimilarity, primary education had only one direct link, with "Physical Education and Maths". This also did not take into account the **number of students** participating in those courses. The only factor it considered was the dissimilarity between the courses

However, when applying weighted degree centrality, the measure was simply the sum of the weights from incoming and outgoing relationships. This does take into account the **number of students** in the course, as the number of outgoing students is proportional to the number of students in the course (10%, in my example simulation). The course also held more connections with other courses, via physical education and maths path (“Primary Education -> “Physical Education and Maths” -> “Actuarial Maths”/”Applied Physics”).

In this case, there are 12 outgoing students, and 2 incoming students. It might be argued that because most students are leaving, it is not a central course. However, I don’t think this is accurate, as people could use a course as an entry point, to see which subjects they like, and then switch to the related course which is better suited to them.



With degree centrality, one thing to note is that it uses the number of incoming and outgoing students as a measure of centrality. As a result, the largest courses would tend to have a larger number of outgoing students, while the smaller courses would have fewer outgoing students. This creates a positive linear relationship between the number of students in a course, and its centrality score.

The red line shows the linear regression model between the number of students in a course, and its centrality score. The biggest positive outliers are “Actuarial Maths” and “Computing for business”, while the biggest negative outlier is “Economics, Politics & Law”.

This shows that Actuarial Maths and Computing for Business have a lot of students entering and leaving the course. Looking at a subgraph of these courses.

This may cause problems in the future, as there may be capacity issues within the courses. This could be an indicator for the college to increase capacity for these courses.

There is a high number of students transferring from primary education to actuarial maths, as well as business to actuarial maths. This could indicate there is demand for new courses, such as “Maths Education” or “Maths for business”.

Choosing max dissimilarity value & Simulation

The average number of possible courses a student can go to with the max dissimilarity at 1.5 is 5.368, 2 is 8.526 and 2.5 is 12.737. I thought that 5 courses seemed reasonable, and chose 1.5 as the max dissimilarity value for the simulation.

One problem I faced when trying to simulate the first year change of courses is how to decide which courses the students would choose out of the possibilities. I decided that a student is equally likely to choose any of the possible courses. In reality, this might not be the case, as there may be limits on the total number of people in certain courses, or a course may be popular to switch to that year.

I chose a value of 10% for students switching courses based on the fact that around 15% of students dropout of college in first year. I predict, given the option of switching courses, 2 thirds of these students would decide to switch.

A general trend of the simulation was that larger courses would end up with less students than when they started, and smaller courses would end up with more students. This is not necessarily a problem with the graph, but in a real world scenario, the larger proportion of outgoing students might be made up for by a larger number of incoming students

Community Detection

I decided louvain method is an appropriate measure of community detection. It evaluates how more densely connected the nodes within a community are, compared to how connected they would be in a random network. It should show dense connections between courses in the same schools and faculties. The final communities are shown below.

Business	Actuarial Maths	Biotechnolog y	Computer Science	Electronic & Computer Engineering	Journalism
Accounting	Applied Physics	Chemistry	Computing for Business	Mechanical Engineering	Psychology
Economics, Politics & Law		Genetics & Cell Biology	Data Science		Social Science

Physical Education& Maths		Nursing			
Primary Education					

I think most of these courses make sense, the only surprise being there not being a separate community for Physical Education & Maths and Primary Education