Name: Ryan Patton 50 points possible

CS 6402 – Advanced Data Mining
Spring 2020
Exam #3

This take-home exam is due by noon on Monday, May 11, 2020.

Everyone should submit a [redacted] file of their exam via Canvas.

This exam is open-book. However, you may not communicate with any person about any aspect of the exam until after the due date.

If you have questions about any of the problems, contact Dr. Leopold via email (leopoldj@mst.edu).

Answers to all problems on this exam MUST be very neat and legible. Pages containing your solutions must be in the same order as the problems are listed in the exam. Add extra pages as necessary.

NOTE: There is a better chance that you will get partial credit for an incorrect solution to a problem if YOU HAVE SHOWN YOUR WORK! Therefore, you should show how you arrived at your solutions, regardless of whether the problems explicitly ask you to do so!!!

Good luck!!!

Name: _____ **50 points possible**

Lecture(s) 4/15/20, 4/13/20, 4/10/20

1. For this problem you will need to download and install the **Neo4j** graph database software on your own computer; it is available at https://neo4j.com. You are to build a **recommender graph** that has a relationship named **RATED** (like we did in lecture) using the following files, which are posted on Canvas along with this exam: **songs.csv, users.csv, userRatedSong.csv**. Note: The setup of these files is slightly different from the ones used in lecture; namely, these files use IDs for songs and users.

   a. Show the **Neo4j commands** you used to create the graph database and a screenshot of the **Neo4j graph** with the **RATED** relationships between the user and song nodes. Before you make the screenshot of the graph, arrange the nodes so that your picture is **reasonably readable!!!** **(2.5 pts.)**

```
LOAD CSV WITH HEADERS FROM 'file:/songs.csv' AS row
CREATE (:songs{songID: row.songID, title: row.title, year: row.year})

LOAD CSV WITH HEADERS FROM 'file:/users.csv' AS row
CREATE (:users {userID: row.userID, name: row.name})

LOAD CSV WITH HEADERS FROM 'file:/userRatedSong.csv' AS row
MATCH (s:songs {songID: row.songID})
MATCH (u:users {userID: row.userID})
MERGE (u)-[r:RATED]->(s)
ON CREATE SET r.userRating = toFloat(row.userRating)

MATCH(u1:users)-[x:RATED]->(s:songs)<-[y:RATED]-(u2:users)
WITH ABS(SUM(REDUCE(x.userRating * y.userRating))) AS Aggregate,
    ABS(REDUCE(xDot = 0, i IN COLLECT(x.userRating) | xDot + toInteger(i))) AS xLength,
    ABS(REDUCE(yDot = 0, j IN COLLECT(y.userRating) | yDot + toInteger(j))) AS yLength,
    u1, u2
MERGE (u1)-[s:SIMILARITY]-(u2)
SET s.value = ((ABS(Aggregate))/((ABS(xLength))+(ABS(yLength))-(ABS(Aggregate)))+2)

MATCH(u1:users {name: 'Steve'})-[s:SIMILARITY]-(u2:users)
WITH u2,s.value AS sim
ORDER BY sim DESC
RETURN u2.name AS Neighbor, sim AS Similarity

MATCH(u1:users)-[x:RATED]->(s:songs)<-[y:RATED]-(u2:users)
WITH u1, u2
MERGE (u1)-[:IS_LIKE]->(u2)
```

Database Information

Use database

neo4j - default

Node Labels

*(22) songs users

Relationship Types

*(47) RATED

Property Keys

name songID title userID userRating year

Connected as

Username: neo4j
Roles: admin
Admin: :server user list
:server user add

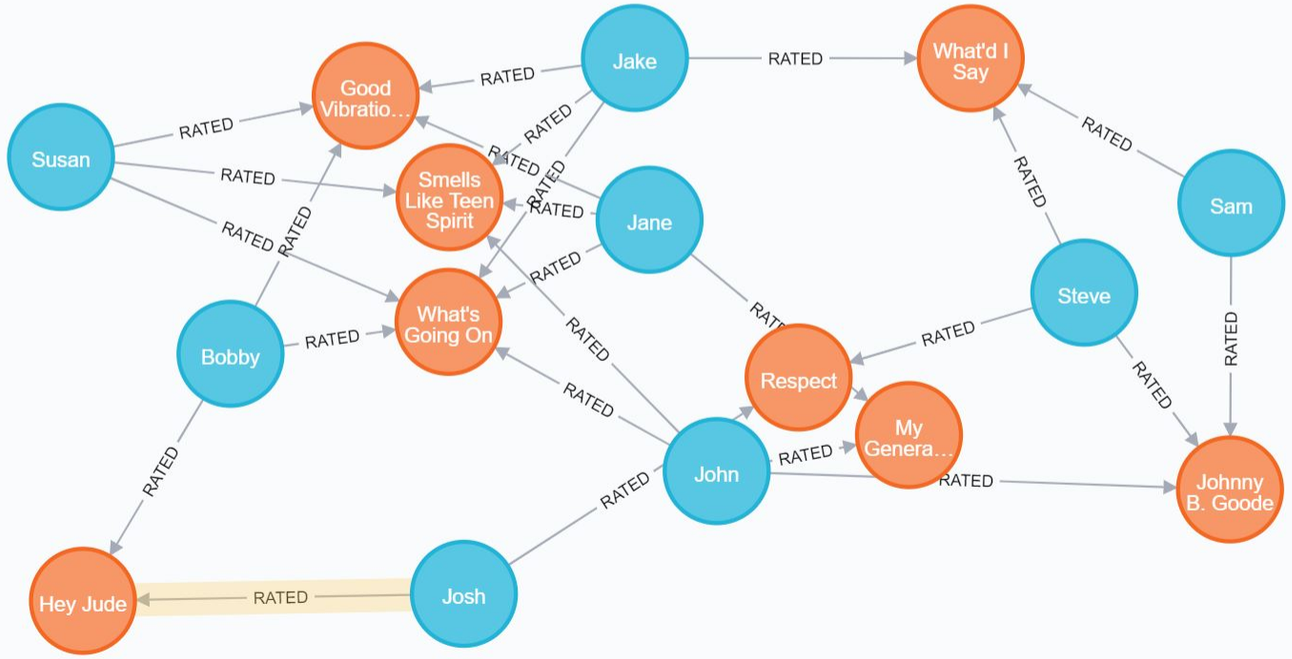
neo4j$

neo4j$ MATCH p=()-[r:RATED]→() RETURN p LIMIT 25

Graph Table Text Code

*(16) users(8) songs(8)

*(25) RATED(25)



RATED <id>: 86 userRating: 4.0

b. In class we defined a similarity relationship between users based on cosine similarity. Write a Cypher (i.e., Neo4j) query that instead defines a relationship named **SIMILARITY** computed using **Jaccard similarity**, which has the following formula:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Here A and B are vectors associated with two different users, respectively. The values in the vectors A and B would be the ratings those users have given to songs that they have rated (i.e., A[x] would be user A's rating for some song x). It should be the case that $0 \leq J(A, B) \leq 1$.

Show your query to create this relationship. Note: Your query needs to "protect" against division by zero!

Then give a query that will display how similar every user is to the user named 'Steve'; show those values listed in descending order of similarity value (i.e., actually run the query in Neo4j and include a screenshot of the results). **(7 pts.)**

```
LOAD CSV WITH HEADERS FROM 'file:/songs.csv' AS row
CREATE (:songs{songID: row.songID, title: row.title, year: row.year})

LOAD CSV WITH HEADERS FROM 'file:/users.csv' AS row
CREATE (:users {userID: row.userID, name: row.name})

LOAD CSV WITH HEADERS FROM 'file:/userRatedSong.csv' AS row
MATCH (s:songs {songID: row.songID})
MATCH (u:users {userID: row.userID})
MERGE (u)-[r:RATED]->(s)
ON CREATE SET r.userRating = toFloat(row.userRating)

MATCH(u1:users)-[x:RATED]->(s:songs)<-[y:RATED]-(u2:users)
WITH u1, u2, COUNT(s) AS intersection, COLLECT(s.songID) AS i
MATCH (u1)-[:RATED]->(u1s:RATED)
WITH u1,u2, intersection,i, COLLECT(u1s.songID) AS xLength
MATCH (u2)-[:RATED]->(u2s:RATED)
WITH u1,u2,intersection,i,xLength, COLLECT(u2s.songID) AS yLength

WITH u1,u2,intersection,xLength,yLength

WITH u1,u2,intersection,xLength+((x IN yLength) WHERE NOT (x in xLength)) AS union, xLength, yLength

RETURN u1.title, u2.title, xLength,yLength((1.0*intersection)/SIZE(union)) AS jaccard ORDER BY jaccard DESC
LIMIT 100

MATCH(u1:users {name: 'Steve'})-[s:SIMILARITY]-(u2:users)
WITH u2,s.value AS sim
ORDER BY sim DESC
RETURN u2.name AS Neighbor, sim AS Similarity

MATCH(u1:users)-[x:RATED]->(s:songs)
WITH u1, algo.similarity.asVector(songs, x.rating) AS u1Vector
MATCH(u2:users)-[x:RATED]->(s:songs) WHERE u2 <> u1

WITH u1, u2, u1Vector, algo.similarity.asVector(songs, x2.rating) AS u2Vector
WHEREsize(apoc.coll.intersection([v in u1Vector | v.category], [v in u2Vector | v.category])) > 4

WITH u1, u2, algo.similarity.pearson(u1Vector, u2Vector, {vectorType: "maps"}) AS similarity
ORDER BY similarity DESC
LIMIT 4

MATCH (u2)-[r:RATED]->(s:songs) WHERE NOT EXISTS( (u1)-[:RATED]->(m) )
RETURN m.songID, SUM( similarity * r.userRating) AS score
ORDER BY score DESC LIMIT 25
```

“neo4j query better attempts” was my attempt at implementing what I thought would be more correct Jaccard similarity and hammock jump”. I got stalled out towards the end of Jaccard similarity on the “AS” part of the query. The results are from “neo4j query” not “neo4j query better attempts”.

Database Information

Use database

neo4j - default

Node Labels

*(22) songs users

Relationship Types

*(72) RATED SIMILARITY

Property Keys

name songID title userID userRating value year

Connected as

Username: neo4j
Roles: admin
Admin: :server user list

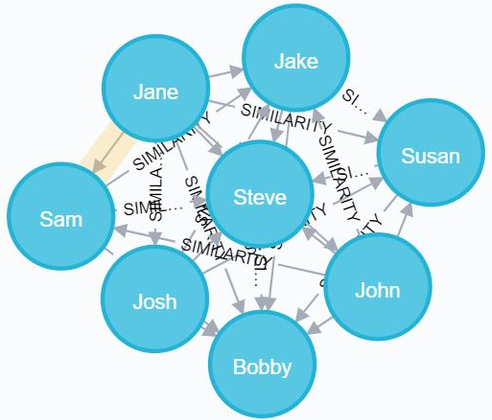
neo4j$

neo4j$ MATCH p=()-[r:SIMILARITY]→() RETURN p LIMIT 25

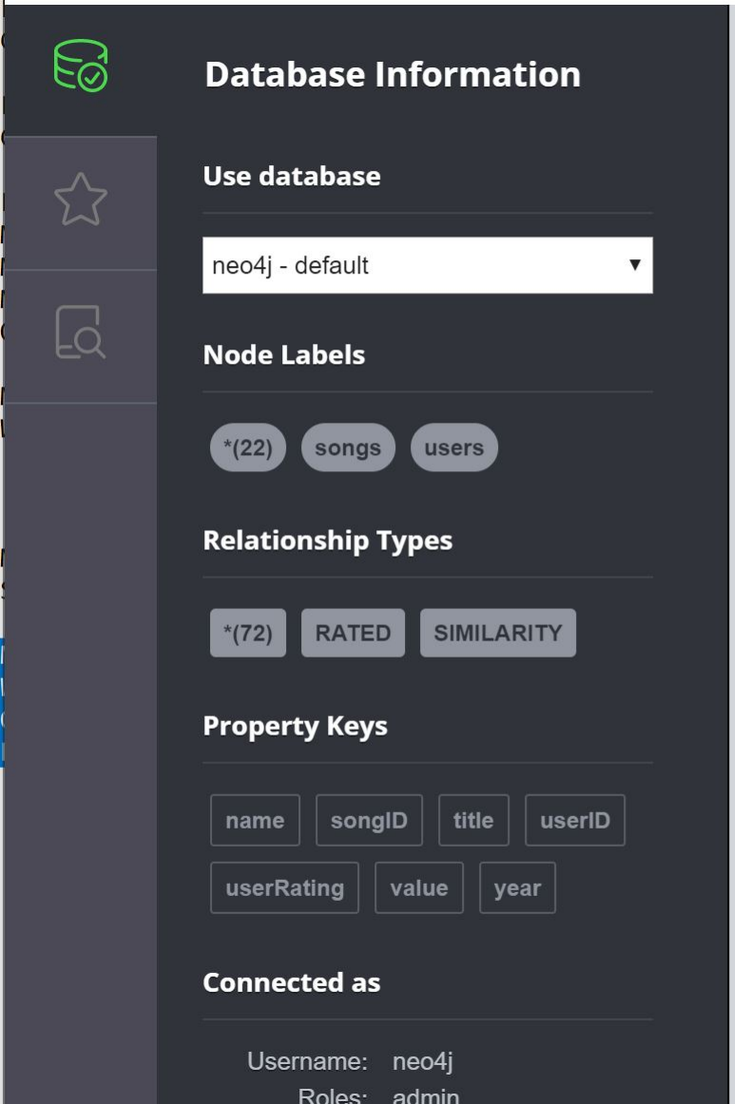
Graph Table Text Code

*(8) users(8)

*(25) SIMILARITY(25)



SIMILARITY <id>: 22 value: 0.6000000000000001



neo4j$

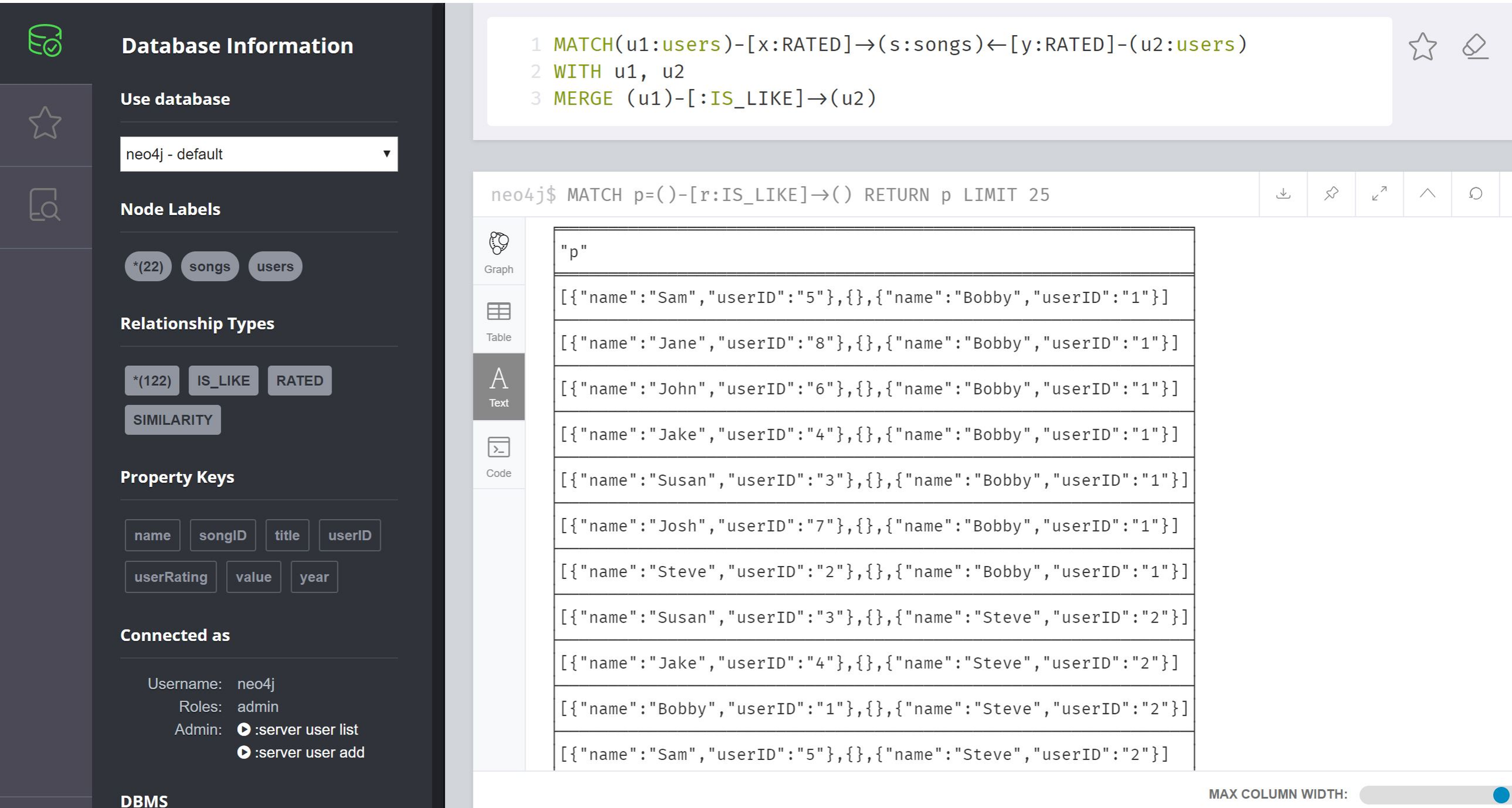
neo4j$ MATCH(u1:users {name: 'Steve'})-[s:SIMILARITY]-(u2:users) WIT…

Table
Text
Code

Neighbor	Similarity
"Jane"	0.6000000000000001
"Jake"	0.6000000000000001
"Bobby"	0.5438596491228069
"John"	0.5384615384615385
"Josh"	0.5192307692307692
"Sam"	0.4893617021276595
"Susan"	0.3529411764705883

Started streaming 7 records after 1 ms and completed after 4 ms.

c. In class we created a recommender graph induced by skips, creating a relationship called **IS_LIKE** between users. Write a query that will create a relationship called **IS_LIKE** in your Neo4j graph database, creating a **recommender graph induced by hammock jumps of size 4**. The relationship also should have a **property** that contains a value equal to the number of songs the pair of users has (rated) in common. Show **BOTH** your query to create this relationship **as well as** a screenshot of the Neo4j **text** (**NOT** the graph!!!) showing the resulting **IS_LIKE** relationship values between all users in the database. **(8 pts.)**


Database Information
Use database
neo4j - default
Node Labels
*(22)
songs
users
Relationship Types
*(122)
IS_LIKE
RATED
SIMILARITY
Property Keys
name
songID
title
userID
userRating
value
year
Connected as
Username: neo4j
Roles: admin
Admin: :server user list
:server user add
DBMS
1 MATCH(u1:users)-[x:RATED]→(s:songs)←[y:RATED]-(u2:users)
2 WITH u1, u2
3 MERGE (u1)-[:IS_LIKE]→(u2)
neo4j$ MATCH p=()-[r:IS_LIKE]→() RETURN p LIMIT 25
Graph
Table
Text
Code
"p"
[{"name":"Sam","userID":"5"},{},{"name":"Bobby","userID":"1"}]
[{"name":"Jane","userID":"8"},{},{"name":"Bobby","userID":"1"}]
[{"name":"John","userID":"6"},{},{"name":"Bobby","userID":"1"}]
[{"name":"Jake","userID":"4"},{},{"name":"Bobby","userID":"1"}]
[{"name":"Susan","userID":"3"},{},{"name":"Bobby","userID":"1"}]
[{"name":"Josh","userID":"7"},{},{"name":"Bobby","userID":"1"}]
[{"name":"Steve","userID":"2"},{},{"name":"Bobby","userID":"1"}]
[{"name":"Susan","userID":"3"},{},{"name":"Steve","userID":"2"}]
[{"name":"Jake","userID":"4"},{},{"name":"Steve","userID":"2"}]
[{"name":"Bobby","userID":"1"},{},{"name":"Steve","userID":"2"}]
[{"name":"Sam","userID":"5"},{},{"name":"Steve","userID":"2"}]
MAX COLUMN WIDTH:

Name: _____ **50 points possible**

Lecture 4/20/2020 34 minutes in

2. Consider a directed graph of 6 vertices where the **degree** (i.e., indegree + outdegree) and **PageRank** of each vertex is shown below:

vertex #	degree	pageRank
0	3	0.1
1	4	0.15
2	5	0.5
3	7	0.5
4	3	0.2
5	0	0

a. Calculate the **Local Outlier Factor (LOF)** for each pair ***(degree of vertex v, pageRank of vertex v)***, for each vertex *v* in the graph. Use k = 1 (for the k-nearest neighbors to consider). **SHOW YOUR WORK** in calculating these values; if you write a program or set up a spreadsheet to do the calculations, include a listing of that. Use **Manhattan distance** for calculating the distance between 2 points. **Put the resulting values in the table below. (9 pts.)**

See Excel Sheet

A = (degree, PageRank)	LOF(A)
3, 0.1	1
4, 0.15	0.5
5, 0.5	0.5
7, 0.5	$1.3\overline{3}$
3, 0.2	0
0, 0	0

$$lrd(A) := 1/\left(\frac{\sum_{B \in N_K(A)} reach\text{-}dist_K(A,B)}{|N_K(A)|}\right)$$

$$LOF_K(A) = \frac{\sum_{B \in N_K(A)} \frac{lrd(B)}{lrd(A)}}{|N_K(A)|} = \frac{\sum_{B \in N_K(A)} lrd(B)/lrd(A)}{|N_K(A)|}$$

$N_K(A) = 1$

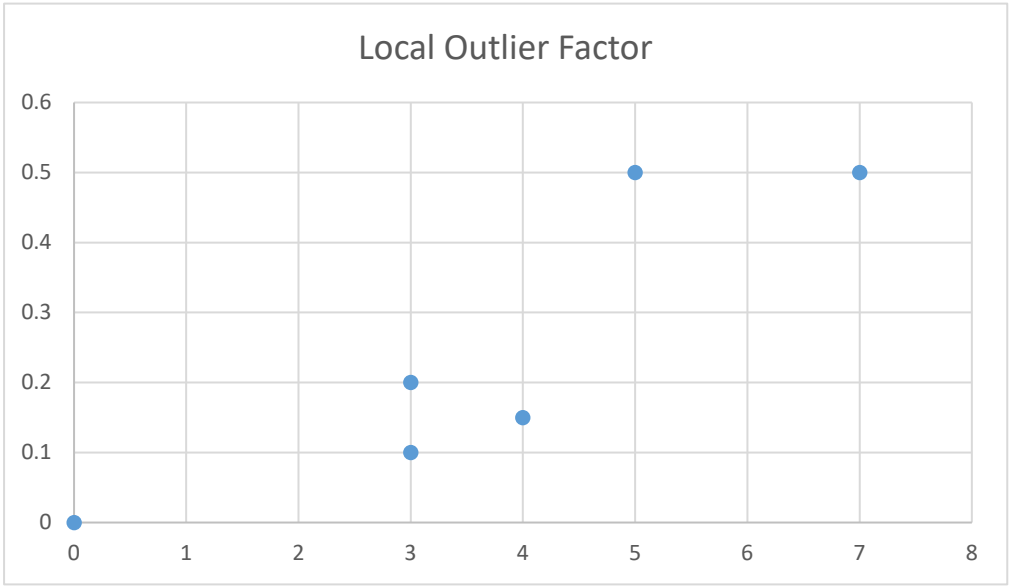
$$\text{Manhattan distance} = \sum_{i=1}^{n} |x_i - y_i|$$

b. Identify which (if any) **vertices** in the graph might be **anomalies** based on the values you calculated in part a. **Justify** your selection of these vertices. **(1 pt.)**

if value is ~1, then A is comparable to its neighbors
if value is <1, then A is part of a dense region (i.e., an inlier)
if value is >1, then A is an outlier/anomaly

vertex 3 would be an outlier because its LOF is 1.33 (>1) and it's the furthest point on my fixed graph

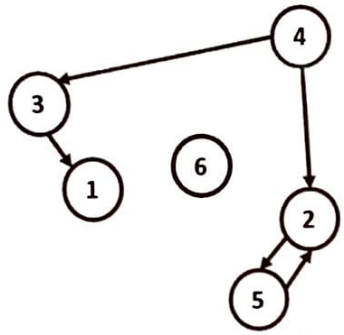
vertex #	degree	pageRank	Manhattan Distance	NkA	lrd	LOF
0	3	0.1		1	1	1
1	4	0.15		1	1	0.5
2	5	0.5		2	1	0.5
3	7	0.5		4	1	0.25	1.333333
4	3	0.2		3	1	0.333333	0
5	0	0		0	1	0	0



Name: _____ **50 points possible**

Lecture 04/22/20

3. Consider the graph shown below, where each vertex depicts a person, and each edge represents a person somehow associated with another person in a social network:



Let c_{ij} represent the **suspiciousness of the association between vertex i and vertex j**, $1 \le i, j \le 6$; these values will be the same as the values in the adjacency matrix for the graph. Let a_i represent the **suspiciousness of each vertex i**, $1 \le i \le 6$; these values are [0.1 0.3 0.4 0.5 0.3 0]. Use the **FRAUDAR** algorithm to find the fraudulent network (i.e., subgraph). **SHOW YOUR WORK** in how you applied the algorithm (i.e., the calculations). **(5 pts.)**

For computing $f(S)$ here, let c_{ij} be the adjacency matrix for the graph and let a_i's be $[1\ 2\ 3\ 4\ 5\ 6] = [0.1\ 0.3\ 0.4\ 0.5\ 0.3\ 0]$

$S = [1\ 2\ 3\ 4\ 5\ 6]$

$g(S) = f(S)/|S|$

$f(S) = f_V(S) + f_E(S)$

$$= \sum_{i \in S} a_i + \sum_{i,j \in S \wedge (i,j) \in \mathcal{E}} c_{i,j}$$

$f(\{1,2,3,4,5\}) = (0.1+0.3+0.4+0.5+0.3) + 5 = 6.8$

$f(\{1,2,3,4,5\})/|\{1,2,3,4,5\}| = 6.8/5 = 1.36$

$f(\{1,2,3,4,6\}) = (0.1+0.3+0.4+0.5) + 3 = 4.3$

$f(\{1,2,3,4,6\})/|\{1,2,3,4,6\}| = 4.3/3 = 1.4\overline{3}$

$f(\{1,2,3,5,6\}) = (0.1+0.3+0.4+0.3) + 3 = 4.1$

$f(\{1,2,3,5,6\})/|\{1,2,3,5,6\}| = 4.1/3 = 1.3\overline{6}$

$f(\{1,2,4,5,6\}) = (0.1+0.3+0.5+0.3) + 3 = 4.2$

$f(\{1,2,4,5,6\})/|\{1,2,4,5,6\}| = 4.2/3 = 1.4$

$f(\{1,3,4,5,6\}) = (0.1+0.4+0.5+0.3) + 2 = 3.3$

$f(\{1,3,4,5,6\})/|\{1,3,4,5,6\}| = 3.3/2 = 1.65$

3. (continued)

$f(\{2,3,4,5,6\}) = (0.3+0.4+0.5+0.3)+4 = 5.5$

$f(\{2,3,4,5,6\})/|\{2,3,4,5,6\}| = 5.5/4 = 1.375$

Highest g if 2 is removed, $S = [1\ 3\ 4\ 5\ 6]$

$f(\{1,3,4,5\}) = (0.1+0.4+0.5+0.3)+2 = 3.3$

$f(\{1,3,4,5\})/|\{1,3,4,5\}| = 3.3/2 = 1.65$

$f(\{1,3,4,6\}) = (0.1+0.4+0.5)+2 = 3$

$f(\{1,3,4,6\})/|\{1,3,4,6\}| = 3/2 = 1.5$

$f(\{1,3,5,6\}) = (0.1+0.4+0.3)+1 = 1.8$

$f(\{1,3,5,6\})/|\{1,3,5,6\}| = 1.8/1 = 1.8$

$f(\{1,4,5,6\}) = (0.1+0.5+0.3)+0 = 0.9$

$f(\{1,4,5,6\})/|\{1,4,5,6\}| = 0.9/1 = 0.9$

$f(\{3,4,5,6\}) = (0.4+0.5+0.3)+1 = 2.2$

$f(\{3,4,5,6\})/|\{3,4,5,6\}| = 2.2/1 = 2.2$

Highest g if 1 is removed, $S = [3, 4, 5, 6]$

$f(\{3,4,5\}) = (0.4+0.5+0.3)+1 = 2.2$

$f(\{3,4,5\})/|\{3,4,5\}| = 2.2/1 = 2.2$

$f(\{3,4,6\}) = (0.4+0.5)+1 = 1.9$

$f(\{3,4,6\})/|\{3,4,6\}| = 1.9/1 = 1.9$

$f(\{3,5,6\}) = (0.4+0.3)+0 = 0.7$

$f(\{3,5,6\})/|\{3,5,6\}| = 0.7/1 = 0.7$ ×

$f(\{4,5,6\}) = (0.5+0.3)+0 = 0.8$

$f(\{4,5,6\})/|\{4,5,6\}| = 0.8/1 = 0.8$ ×

3. (continued)

Highest g if 6 is removed, S = [3, 4, 5]

$f(\{3,4\}) = (0.4 + 0.5) + 1 = 1.9$

$f(\{3,4\})/|\{3,4\}| = 1.9/1 = 1.9$

$f(\{3,5\}) = (0.4 + 0.3) + 0 = 0.7$

$f(\{3,5\})/|\{3,5\}| = 0.7/1 = 0.7$

$f(\{4,5\}) = (0.5 + 0.3) + 0 = 0.8$

$f(\{4,5\})/|\{4,5\}| = 0.8/1 = 0.8$

Highest g if 5 is removed, S = [3, 4]

$f(\{3\}) = (0.4) + 0 = 0.4$

$f(\{3\})/|\{3\}| = 0.4/1 = 0.4$

$f(\{4\}) = (0.5) + 0 = 0.5$

$f(\{4\})/|\{4\}| = 0.5/1 = 0.5$

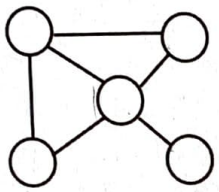
Highest g if 4 is removed, S = [3]

Of all the scenarios, the highest g value was for $\{3, 4, 5, 6\}$ and $\{3\ 4\ 5\}$ so these are the fraudulent graphs

Name: _____ **50 points possible**

4/27/20 Lecture - continues to 4/29/20?

4. Consider the subgraph shown below. Using the **VoG** strategy, determine whether it would **best** be labeled as **a full-clique, a star, or a chain**. Assume that this subgraph exists in a graph G = (V, E) where |V| = 7 (we're keeping it small so your calculations are easy!). Also, assume a binary encoding scheme when determining description length (e.g., $L_N(5) = 3$ because it takes 3 bits to encode 5 in binary). Use base 2 for your log calculations[1]. **NO PARTIAL CREDIT WILL BE GIVEN UNLESS YOU SHOW YOUR WORK FOR ALL OF THE VoG CALCULATIONS! (8 pts.)**



Note: Since this subgraph is not a "perfect" match for a full-clique, star, or chain, you will have to consider $\mathbf{E^+}$ (edges that are there but shouldn't be there) and $\mathbf{E^-}$ (edges that are "missing" in the sense that they should be there).

E^+ = incorrectly modeled edges (i.e., edges that are there, but shouldn't be there)

E^- = incorrectly modeled edges (i.e., edges that are missing, should be there)

Calculate cost $L(x) + L(E_x^+) + L(E_x^-)$ where:

- $L(x)$ is the value for encoding the subgraphs as type x,
- $L(E_x^+)$ is what it would "cost" (i.e. # bits) to encode the edges that the subgraph has that shouldn't be there for type x and
- $L(E_x^-)$ is what it would "cost" (i.e., # bits) to encode the edges that the subgraph is missing to be type x (x = full-clique, star, bipartite core, chain)

Chain? It has 2 edges that shouldn't be there

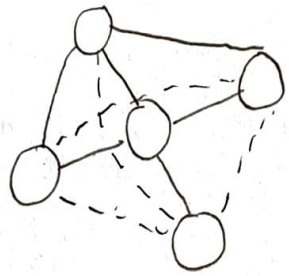
Star? 2 edges that shouldn't be there

Full-clique? It is missing 4 edges

Bipartite Core?

[1] There's a log calculator available at http://www.rapidtables.com/calc/math/Log_Calculator.htm#

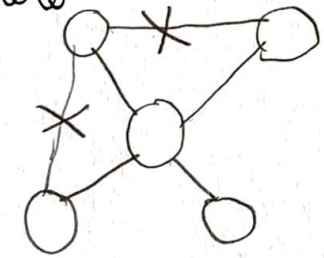
Full Clique:



$L(fc) + L_N(4)$

$L_2(4) = 2$

Star? 2 edges that shouldn't be there



$L(st) + L_N(2)$

$L_2(2) = 1$

Assign type to each subgraph, add each subgraph to the set C of candidate structures (where C_x = structures of type $x \in \Omega$)

$$C = \bigcup_{x \in \Omega} C_x$$

Plain, Top-K, Greedy 'n' Forget

$$L(ch) = L_N(|ch| - 1) + \sum_{i=0}^{|ch|} \log(n-i)$$

$$L(fc) = L_N(|fc|) + \log\binom{n}{|fc|} = L_N(|nc|) + \log\binom{n}{|nc|} + \log(\text{area}|nc|) + \|nc\|_1 + \|nc\|_0$$

$$L(st) = L_N(|st| - 1) + \log n + \log\binom{n-1}{|st|-1}, \quad |st| - 1 = \#\text{ of spokes} = 4$$

4. (Continued)

$$L_{(FC)} = L_2(5) + \log\binom{10}{5}$$
$$= 2.321928094\,89 + 0.3010299957$$
$$= 2.622958091$$
$$L_{(FC)} + L_2(4) = 4.622958091$$

$$L_{(ch)} = L_2(4) + \log(4-1)$$
$$= 2 + 0.4771$$
$$= 2.477121255$$
$$L_{(ch)} + L_N(2) = 3.477121255$$

same

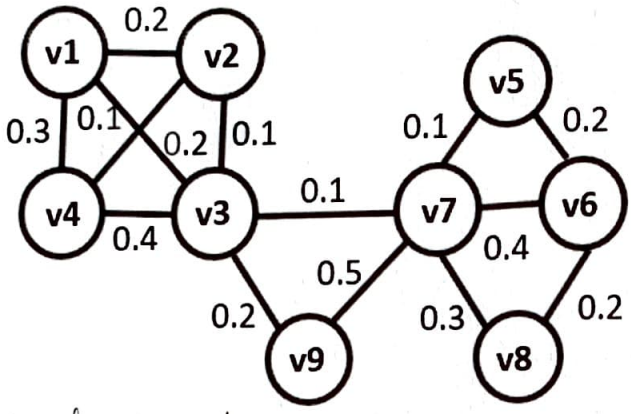
$$L_{(st)} = L_2(4) + \log(4) + \log\binom{3}{4}$$
$$= 2 + 0.6021 + (-0.1249)$$
$$= 2.477121255$$
$$L_{(st)} + L_N(2) = 3.477121255$$

∴ Could be labeled star or chain

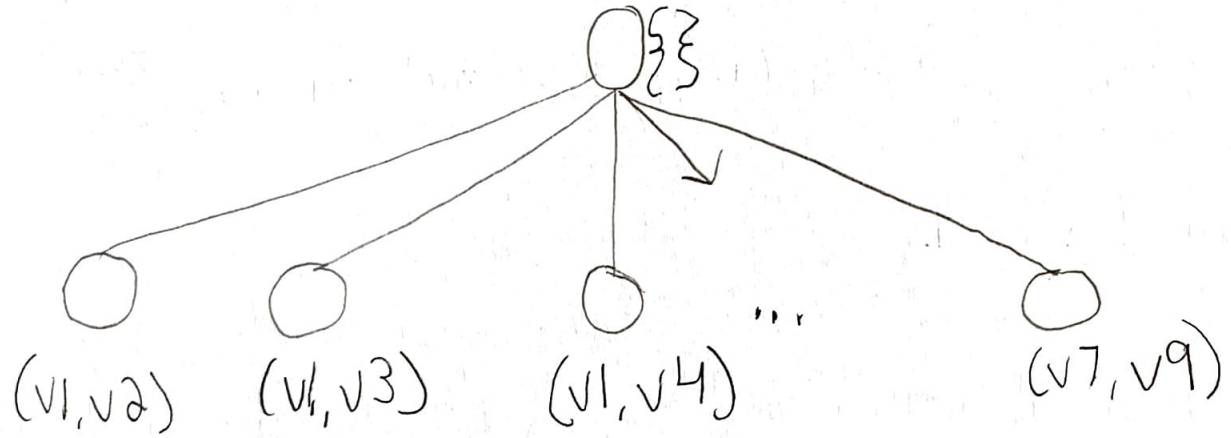
Name: _____ **50 points possible**

5/21/20 lecture

5. Consider the **uncertain graph** shown below. Using the method discussed in class, find the **top 3 maximal cliques. SHOW YOUR WORK! (6 pts.)**



First two levels of search tree will end up looking something like this (i.e. all single edge subgraphs are cliques):



If processed breadth-first, mini-heap would contain

$\{v2, v3\}, \{v2, v4\}, \{v3, v7\}, \{v5, v7\}$

Depth-wise progression from the $\{v1, v2\}$ node would look like this:

$\{v1, v2\}$

$\{v1, v2, v3\}$ $0.2 \times 0.1 = 0.02$

$\{v1, v2, v3, v4\}$ $0.2 \times 0.1 \times 0.4$

5. (continued)

Note that $\{v1, v2\}$ has value 0.2, $\{v1, v2, v3\}$ has value 0.02, and $\{v1, v2, v3, v4\}$ has value 0.008

Adding more vertices makes values smaller

Want max cliques → keep min-heap

If $\{v1, v2\}$ (0.2) is in the heap, it will get replaced by $\{v1, v2, v3\}$ (0.02)

If $\{v1, v2, v3\}$ (0.02) is in the heap, it will get replaced by $\{v1, v2, v3, v4\}$ (0.008)

∴ The top 3 maximal cliques are $\{v1, v2\}$ w/ a value of 0.2, $\{v1, v2, v3\}$ w/ a value of 0.02, and $\{v1, v2, v3, v4\}$ w/ a value of 0.008.

Name: _____ 50 points possible

6. The paper you were assigned to read on **GMiner** [Chen et al. 2018] discussed **vertex-centric programming** as a strategy to solve graph mining problems for large graphs. This strategy requires a graph mining algorithm to specify **logic for each vertex** to perform so that those computations can be executed in parallel. Briefly discuss how a **vertex-centric approach** could be used to find all **k-trusses** in a single graph. **(3.5 pts.)**

See Word doc/PDF

“The vertex-centric graph systems do not consider the characteristics of graph mining algorithms in their design. In particular, the computational model should be more coarse-grained, subgraph-centric instead of vertex-centric, since each subgraph now plays the role of a processing unit that maintains a local state and accordingly decides how to involve more vertices for updating. Taking a vertex-centric programming approach means users must specify the algorithm logic for each individual vertex so that the runtime may execute in parallel. This may generate a lot of subgraphs and algorithm logic is more attached with each subgraph instead of any single vertex.”

This benefits finding all k-trusses in a single graph because finding all k-trusses is mostly linear complexity for each iteration, resulting in light computation and communication on the vertices. Finding all k-trusses is an elimination process between the iterations, and with a vertex-centric approach the algorithm logic may be specified for finding the 3-trusses, then the 4-trusses, and on.