GRAPH ANALYTICS – CRIME INVESTIGATION

Simone Richetti, mat. 129180, attività 6.

TRACCIA

L'obiettivo è analizzare una Sandbox di Neo4j (esclusa quella vista a lezione) e implementare un'attività di graph analytics attraverso l'esecuzione di *almeno 3 tecniche distinte e una loro interpretazione nel contesto della Sandbox scelta*.

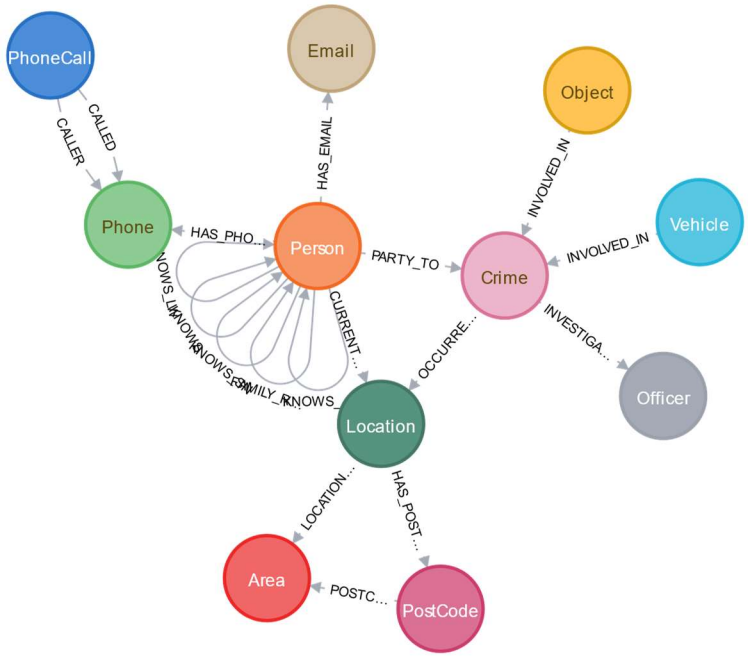
Alcune precisazioni riguardo l'attività richiesta:

- Neo4J presenta diverse Sandbox utili a questo scopo, ad esempio la recentissima Crime Investigation, Russian Twitter Trolls, Paradise Papers by ICIJ, Legis-Graph, Trumpworld, Network and IT Management;
- Le tecniche potranno essere implementate sia usando le procedure di *graph analytics* messe a disposizione da Neo4J sia attraverso l'esecuzione di query *Cypher*. *Almeno una tecnica* dovrà prevedere una *Cypher Projection*;
- Il risultato dell'attività sarà un documento contenente una breve descrizione della Sandbox scelta e dei risultati ottenuti dall'esecuzione delle query proposte dalla Sandbox stessa seguita da una descrizione degli obiettivi delle tecniche di *graph analytics* implementate, il codice eseguito sulla Sandbox e un'interpretazione dei risultati ottenuti.

SANDBOX NEO4J: CRIME INVESTIGATION

Per lo svolgimento di questa traccia si è scelto di analizzare la Sandbox Neo4j *Crime Investigation*. Questa Sandbox contiene dati relativi ai crimini a Manchester seguendo il *POLE data model*, ovvero un grafo in cui entità e relazioni sono relative a *Persons*, *Objects*, *Locations* ed *Events*. Questo modello dei dati è uno standard nei settori della polizia, dell'investigazione e della sicurezza. I dati sono stati manipolati per ragioni di privacy e sicurezza (i crimini riportano solo il mese, le persone sono fittizie, ecc.).

Andando maggiormente nel dettaglio, lo *schema* del database è il seguente:



Nel database sono presenti numerose entità. Nella nostra analisi non siamo intenzionati ad analizzare l'intero grafo, bensì ci concentreremo sulle persone (nodi *Person*) e sui crimini (nodi *Crime*). Ogni persona è definita da un identificativo univoco, rappresentato nel database con l'attributo *nhs_no* del relativo nodo *Person*.

Passando alle relazioni, osserviamo innanzitutto che due persone possono essere collegate da diversi tipi di relazione:

- *KNOWS*: relazione generica di conoscenza, tipicamente accompagnata da un'altra relazione più specifica tra le seguenti;
- *FAMILY_REL*: relazione di parentela;
- *KNOWS_LW*: le persone vivono insieme;
- *KNOWS_PHONE*: è stata registrata una telefonata tra i due;
- *KNOWS_SN*: relazione di amicizia sui social network.

Oltre a queste relazioni un'altra relazione importante che utilizzeremo è la partecipazione di una persona ad un crimine: *PARTY_TO*.

INFORMAZIONI GLOBALI SUL GRAFO

Iniziamo ad analizzare il grafo cercando di ottenere informazioni che lo descrivano nella sua globalità. Iniziamo cercando il numero di persone e di crimini registrati nel database:

```
1 MATCH (:Person)
2 RETURN count(*)
```

count(*)
369

```
1 MATCH (:Crime)
2 RETURN count(*)
```

count(*)
28762

Notiamo come il numero di crimini registrati sia sproporzionato rispetto al numero di persone che il database contiene. Quanti di essi sono stati effettivamente collegati ai relativi colpevoli?

```
1 MATCH (:Person)-->(:Crime)
2 RETURN count(*)
```

count(*)
55

Una minima percentuale di essi: esistono solo 55 relazioni tra persone e crimini. Quante relazioni sono presenti tra le persone?

```
1 MATCH (:Person)-->(:Person)
2 RETURN count(*)
```

count(*)
1180

Abbiamo 369 persone e 1180 relazioni tra di esse, una media di 3.2 relazioni per ogni persona. Concludiamo questa parte calcolando il diametro del grafo e rapportandolo alle informazioni che abbiamo ricavato finora. Il diametro del grafo è calcolato come il più lungo tra tutti gli *shortest path* tra due possibili nodi. Nel nostro caso siamo interessati a calcolarlo per quanto riguarda le *Person* e le possibili relazioni tra loro: il valore del diametro dovrebbe essere un indicatore per aiutarci a capire quanto sono effettivamente connesse tra di loro le diverse persone nel grafo.

```
CALL algo.allShortestPaths.stream(null,{nodeQuery:'Person',defaultValue:1.0})
YIELD sourceNodeId, targetNodeId, distance
RETURN sourceNodeId, targetNodeId, distance
ORDER BY distance DESC
```

sourceNodeId	targetNodeId	distance
57	456	11.0
57	1126	11.0
57	1128	11.0
456	57	11.0
1126	57	11.0
1128	57	11.0
66	57	10.0
86	57	10.0
166	57	10.0
57	66	10.0
57	86	10.0

Il numero massimo di connessioni che separano due persone è 11. Questo, insieme ad un discreto rapporto tra il numero di persone e il numero di relazioni tra di esse, ci fa intuire che le persone in questo grafo siano abbastanza connesse tra di loro.

COMMUNITY ANALYSIS

Possiamo considerare questo grafo come una forma particolare di rete sociale, essendo presenti persone e relazioni tra di esse, e la tipologia di analisi che vogliamo operare sul grafo è per molti aspetti simile alle analisi che si svolgono su reti sociali. In questo tipo di reti, le analisi di *community detection* forniscono spesso informazioni preziose, identificando gruppi di persone fortemente connesse tra di loro: cerchiamo quindi di applicare tecniche di analisi di questa tipologia per ottenere informazioni più approfondite sulle comunità presenti nel grafo.

TRIANGLE COUNTS

Una prima semplice tecnica che possiamo utilizzare per individuare delle comunità è quella dei **Triangle Counts**: contiamo ogni persona a quanti triangoli appartiene, identificando in questo modo delle comunità fortemente connesse tra di loro. Raffiniamo ulteriormente la nostra ricerca: non consideriamo tutte le relazioni di conoscenza di tutte le persone, bensì consideriamo solo le persone che hanno partecipato ad un crimine o che hanno diretti conoscenti criminali. In questo modo, potremmo identificare gruppi criminali.

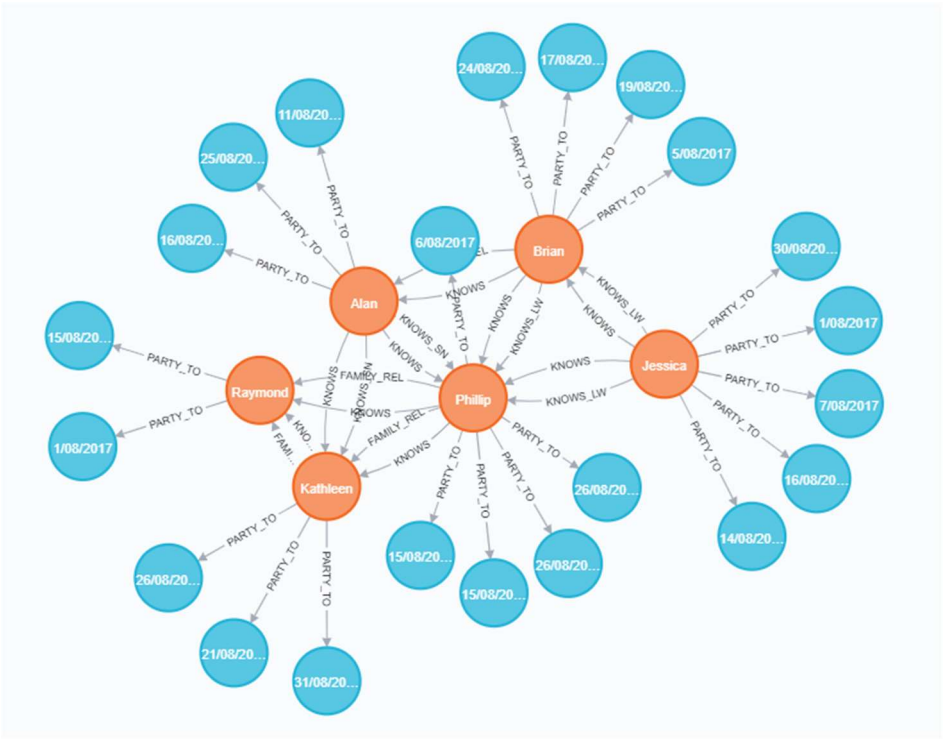
Iniziamo identificando le persone che appartengono al maggior numero di triangoli:

```
CALL algo.triangleCount.stream(
'MATCH (p:Person)-[:KNOWS*0..1]-(:Person)-[:PARTY_TO]->(c:Crime) RETURN id(p) AS id',
'MATCH (p1:Person)-[:KNOWS]-(p2:Person) RETURN id(p1) AS source, id(p2) AS target',
{graph:'cypher'})
YIELD nodeId, triangles
MATCH (p:Person)
WHERE ID(p) = nodeId AND
triangles > 0
RETURN p.name AS name, p.surname AS surname, p.nhs_no AS id, triangles
ORDER BY triangles DESC
LIMIT 5;
```

name	surname	id	triangles
"Phillip"	"Williamson"	"337-28-4424"	4
"Brian"	"Morales"	"335-71-7747"	3
"Alan"	"Ward"	"881-20-2396"	3
"Kathleen"	"Peters"	"250-75-5238"	2
"Jessica"	"Kelly"	"311-75-6483"	2

Possiamo osservare come *Philip Williamson* sia la persona che appartiene al maggior numero di triangoli, 4. Per capire se Philip faccia effettivamente parte di una banda criminale, proviamo a cercare quante persone appartenenti ad uno dei triangoli a cui lui stesso appartiene hanno partecipato a crimini:

```
MATCH (p1:Person {nhs_no: '337-28-4424', surname: 'Williamson'})-[k1:KNOWS]-(p2)-[k2:KNOWS]-(p3)-[k3:KNOWS]-(p1)
WITH *
MATCH (person)-[pt:PARTY_TO]->(crime) WHERE person IN[p1, p2, p3]
RETURN *
```



Sembra che abbiamo identificato un gruppo criminale. Approfondiamo le loro attività controllando a quali tipologie appartengono i crimini commessi:

```
MATCH (p1:Person {nhs_no: '337-28-4424', surname:
'Williamson'})-[k1:KNOWS]-(p2)-[k2:KNOWS]-(p3)-
[k3:KNOWS]-(p1)
WITH *
MATCH (person)-[pt:PARTY_TO]->(c:Crime) WHERE person
IN[p1, p2, p3]
WITH c
RETURN c.type AS Tipologia, count(*) AS No
```

Tipologia	No
"Vehicle crime"	76
"Drugs"	14
"Robbery"	4

Il fatto che i crimini commessi siano numerosi e quasi sempre dello stesso tipo è un'ulteriore conferma che la rete relazionale di Philip Williamson forma una banda criminale, specializzata in furti d'auto.

Un altro aspetto interessante che si può approfondire è quello legato al rapporto tra bande criminali e tecnologia: esistono bande criminali che si organizzano sui social o via telefono, proprio per evitare di essere collegati l'uno all'altro? Utilizziamo nuovamente il *Triangle Counts* nella modalità appena vista, ma considerando solo le relazioni *KNOWS_PHONE* e *KNOWS_SN*:

```
CALL algo.triangleCount.stream(
'MATCH (p:Person)-[*0..1]-(:Person)-[:PARTY_TO]->(c:Crime) RETURN id(p) AS id',
'MATCH (p1:Person)-[:KNOWS_PHONE|:KNOWS_SN]-(p2:Person) RETURN id(p1) AS source, id(p2) AS target',
{graph:'cypher'})
YIELD nodeId, triangles
MATCH (p:Person)
WHERE ID(p) = nodeId AND
triangles > 0
RETURN p.name AS name, p.surname AS surname, p.nhs_no AS id, triangles
ORDER BY triangles DESC
LIMIT 5;
```

Questa query non fornisce alcun risultato. Una possibile ragione potrebbe essere un basso numero di relazioni di questo tipo, ipotesi supportata dal fatto che considerando tutte le relazioni tra persone il massimo numero di triangoli a cui appartiene una persona è 4. Per aver conferma di questo, facciamo alcune queries sul database per vedere quante relazioni di questo tipo ci sono e quanti triangoli si formano, senza filtrare per i criminali:

```
MATCH (:Person)-[:KNOWS_SN|:KNOWS_PHONE]-(:Person)
RETURN count(*)
```

count(*)
718

```
MATCH (p1)-[:KNOWS_SN|:KNOWS_PHONE]-(p2)-[:KNOWS_SN|:KNOWS_PHONE]-(p3)-[:KNOWS_SN|:KNOWS_PHONE]-(p1)
RETURN count(*)
```

count(*)
6

In tutto il database sono presenti 718 relazioni di conoscenza telefonica o legate ai social network, ma solo 6 triangoli. È dunque impossibile ottenere informazioni relative all'uso dei social da parte dei criminali per organizzarsi: questo è sicuramente un aspetto prioritario su cui ci si potrebbe concentrare per arricchire il grafo, data l'alta importanza strategica e l'alta accessibilità delle informazioni legate ai *social media.*

LOUVAIN METHOD

Un altro metodo per individuare comunità all'interno di un grafo è il metodo di *Louvain*. *Louvain* ha un approccio completamente opposto rispetto alla tecnica *Triangle Counts*:

- *Triangle Counts* ha un approccio locale: preso il singolo nodo, esplora le relazioni intorno ad esso per identificare triangoli che coinvolgono il nodo considerato. Identifica piccoli cluster senza alcuna nozione legata alla globalità del grafo;
- *Louvain* è un metodo globale che sfrutta la misura della modularità: viene inizializzato assegnando ad ogni nodo una comunità casuale, quindi cerca di trovare il partizionamento che massimizza la modularità di tutto il grafo.

Può essere interessante identificare le comunità nel nostro sotto grafo di nodi *Person* sfruttando il metodo di *Louvain*, salvare la comunità di ogni persona come attributo del relativo nodo e osservare se il gruppo che abbiamo identificato precedentemente con il metodo *Triangle Counts* viene inserito da *Louvain* nella stessa *community*:

```
CALL algo.louvain('Person', null, {write:true, writeProperty:'community'})
YIELD nodes, communityCount
```

nodes	communityCount
369	41

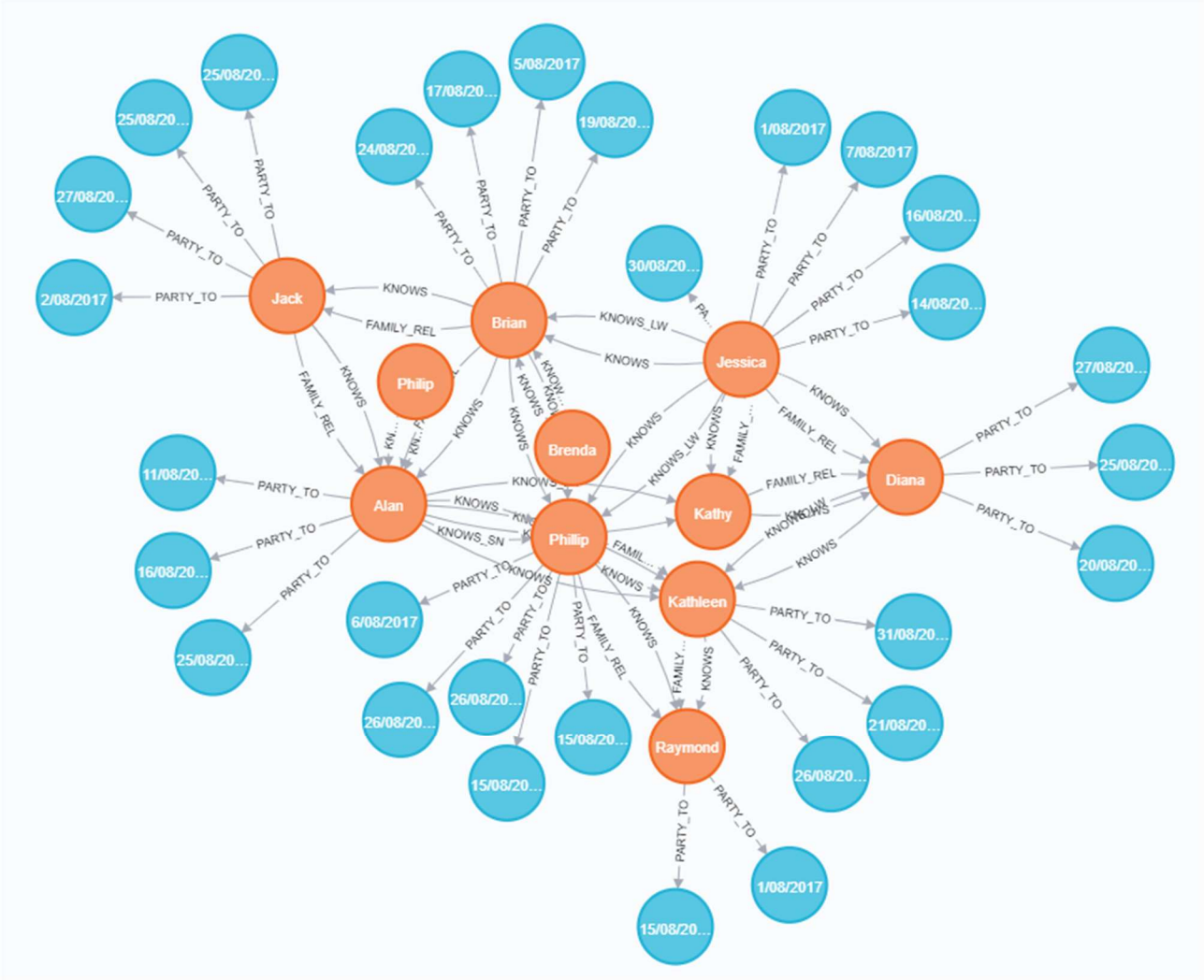
La comunità calcolata dall'algoritmo di *Louvain* per ogni nodo *Person* è stata salvata nella proprietà *community*. Visualizziamo quindi il valore di questo attributo per il gruppo di Philip Williamson:

```
MATCH (p1:Person {nhs_no: '337-28-4424', surname: 'Williamson'})-[k1:KNOWS]-(p2)-[k2:KNOWS]-(p3)-[k3:KNOWS]-(p1)
WITH *
MATCH (person)-[pt:PARTY_TO]->(crime) WHERE person IN[p1, p2, p3]
RETURN DISTINCT person.name, person.surname, person.community
```

person.name	person.surname	person.community
"Phillip"	"Williamson"	6
"Jessica"	"Kelly"	6
"Brian"	"Morales"	6
"Alan"	"Ward"	6
"Raymond"	"Walker"	6
"Kathleen"	"Peters"	6

Il metodo di *Louvain* aggrega tutti le persone correlate a crimini e facenti parte di un triangolo relazionale con Philip nello stesso gruppo. A questo punto possiamo controllare quali siano le persone della comunità 6 che hanno partecipato a crimini oppure che conoscono qualcuno della stessa comunità che lo ha fatto, per vedere se possiamo identificare un gruppo ancora più ampio:

```
MATCH (p1:Person)-[*0..1]->(p2:Person)-[:PARTY_TO]->(c:Crime)
WHERE p1.community = 6 AND p2.community = 6
RETURN *
```



Possiamo osservare come ci siano dei potenziali *outliers*, ovvero persone che hanno un solo collegamento con una persona del gruppo e quindi non è detto che siano effettivamente criminali, ma allo stesso tempo abbiamo individuato nuove persone direttamente correlate a crimini e con diverse relazioni con persone del gruppo di Philip che prima non avevamo individuato. Guardiamo ai loro nomi:

```
MATCH (p1:Person)-[*0..1]->(p2:Person)-[:PARTY_TO]->(c:Crime)
WHERE p1.community = 6 AND p2.community = 6
WITH *
MATCH (p:Person)
WHERE p IN [p1, p2]
RETURN DISTINCT p.name, p.surname
```

p.name	p.surname
"Raymond"	"Walker"
"Phillip"	"Williamson"
"Kathleen"	"Peters"
"Alan"	"Ward"
"Diana"	"Murray"
"Kathy"	"Wheeler"
"Jessica"	"Kelly"
"Philip"	"Gardner"
"Brian"	"Morales"
"Jack"	"Powell"
"Brenda"	"Edwards"

Diana Murray e *Jack Powell* non erano stati identificati dalla nostra analisi con il *Triangle Counts*, ma sono entrambi legati a più membri del gruppo e correlati a numerosi crimini, rispettivamente 3 e 4. Peraltro, i loro crimini sono tutti legati alla droga, tipologia di reati in cui sono frequentemente coinvolti anche altri individui della banda.

CENTRALITY ANALYSIS

Nell'ultima parte di questo progetto ci spostiamo dall'analisi dei gruppi a quella dei singoli individui. Utilizziamo due tecniche di analisi di centralità dei grafi per individuare nodi importanti all'interno della rete.

BETWEENNESS CENTRALITY

La ***betweenness centrality*** è una delle due grandezze che utilizziamo per misurare la centralità e l'importanza di un nodo in un grafo. Essa è calcolata misurando per ogni nodo quanti degli *shortest path* che collegano ogni coppia di nodi passano attraverso di esso. I nodi con *betweenness* più alta sono, quindi, i nodi che fanno da collegamento tra più comunità e clusters: filtrando il grafo considerando solo persone legate a crimini, i nodi con *betweenness* più alta sono persone che hanno contatti con diversi gruppi criminali. Un'analisi di questo tipo potrebbe essere utile, ad esempio, ad un distretto di polizia che deve individuare soggetti da monitorare per intercettare flussi di informazioni o traffici di merci.

Utilizziamo quindi le funzioni di *Neo4j* per calcolare la *betweenness centrality* delle persone legate a crimini registrati nel grafo. A differenza della sezione precedente, saranno considerate le persone fino a due gradi di separazione di relazioni da una persona coinvolta in un crimine: questo per avere una panoramica più chiara delle comunità che vengono connesse da un singolo individuo.

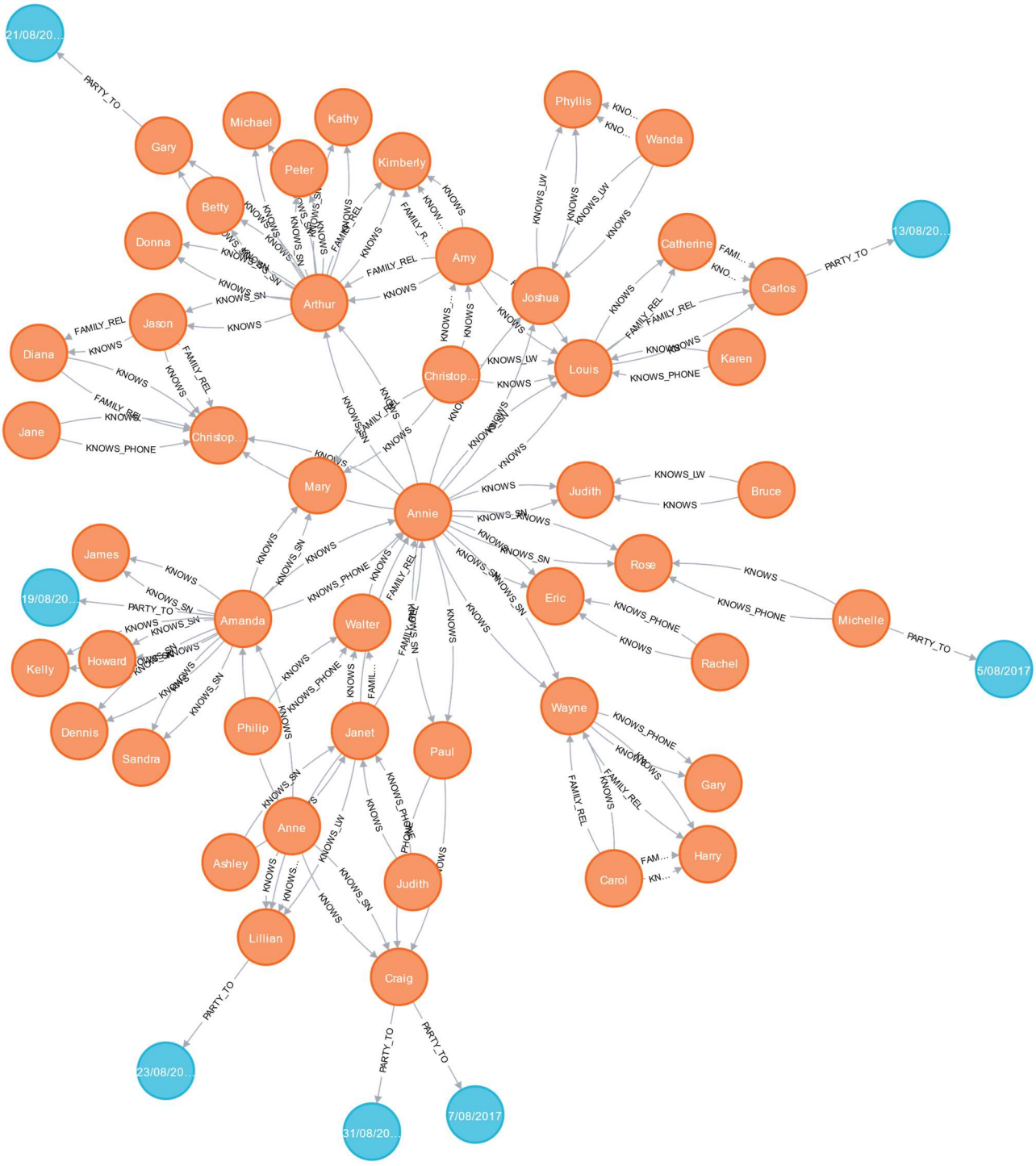
```
CALL algo.betweenness.stream(
'MATCH (p:Person)-[:KNOWS*0..2]-(:Person)-[:PARTY_TO]->(c:Crime) RETURN id(p) AS id',
'MATCH (p1:Person)-[:KNOWS]-(p2:Person) RETURN id(p1) AS source, id(p2) AS target',
{graph:'cypher', direction: 'both'})
YIELD nodeId, centrality
MATCH (p:Person)
WHERE ID(p) = nodeId
RETURN p.name AS name, p.surname AS surname, p.nhs_no AS id, toInteger(centrality) AS score
ORDER BY centrality DESC
LIMIT 10;
```

name	surname	id	score
"Annie"	"Duncan"	"863-96-9468"	2810
"Amanda"	"Robertson"	"455-19-0708"	2421
"Andrea"	"Montgomery"	"351-83-4608"	2348
"Anne"	"Freeman"	"804-54-6976"	2305
"Arthur"	"Willis"	"271-78-8919"	1957
"Ann"	"Fox"	"576-99-9244"	1876
"Bonnie"	"Gilbert"	"622-53-3302"	1837
"Andrea"	"George"	"800-46-2184"	1826
"Annie"	"George"	"575-05-6519"	1696
"Amy"	"Murphy"	"367-54-3328"	1590

Annie Duncan risulta essere la persona con maggiore *betweenness centrality* nel sotto grafo considerato. Esploriamo il grafo delle conoscenze di Annie, visualizzando anche i crimini commessi dalle persone collegate:

```
MATCH (:Person {nhs_no: '863-96-9468', surname: 'Duncan'})-[:KNOWS*0..2]-(p2:Person)
OPTIONAL MATCH (p2)-[:PARTY_TO]->(c:Crime)
RETURN *
```

Dai risultati della query si può vedere come *Annie* faccia da punto di congiunzione di numerose comunità di individui strettamente connessi tra loro, facendone quindi un individuo di alta rilevanza.



DEGREE CENTRALITY

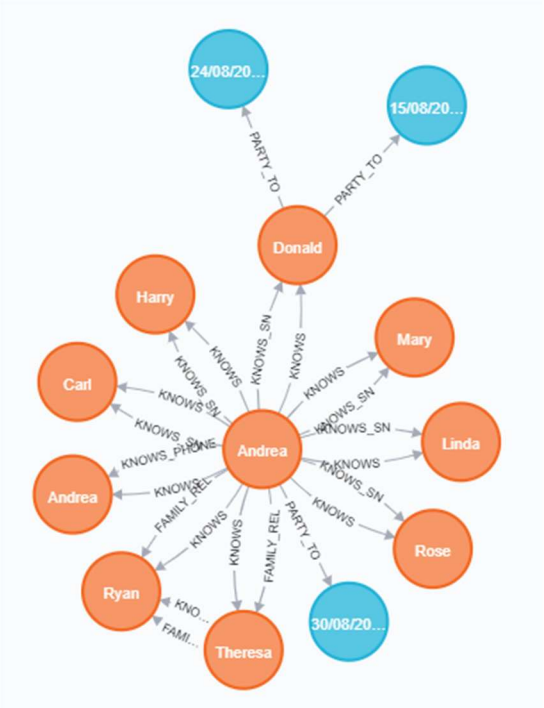
La *degree centrality* è una misura molto semplice di centralità e importanza di un nodo: essa è proporzionale al numero di archi entranti e/o uscenti da un nodo. Ne consegue che un nodo con alta *degree centrality* è un nodo molto influente, che si connette a molti altri nodi. Se un nodo con alta *betweenness centrality* è un punto di contatto tra diverse comunità, probabilmente un nodo con alta *degree* è il leader di una comunità numerosa, a cui tutti i membri di quel gruppo si collegano.

Ancora una volta, considerando il sotto grafo su cui abbiamo fatto analisi finora, può essere utile fare un'analisi di centralità con questa misura per identificare leader e persone influenti nella comunità criminale:

```
CALL algo.degree.stream(
'MATCH (p:Person)-[:KNOWS*0..1]-(:Person)-[:PARTY_TO]->(c:Crime) RETURN id(p) AS id',
'MATCH (p1:Person)-[:KNOWS]-(p2:Person) RETURN id(p1) AS source, id(p2) AS target',
{graph:'cypher', direction: 'both'})
YIELD nodeId, score
MATCH (p:Person)
WHERE ID(p) = nodeId
RETURN p.name AS name, p.surname AS surname, p.nhs_no AS id, toInteger(score) AS score
ORDER BY score DESC
LIMIT 10;
```

name	surname	id	score
"Annie"	"George"	"575-05-6519"	9
"Andrea"	"Montgomery"	"351-83-4608"	9
"Anne"	"Freeman"	"804-54-6976"	8
"Alan"	"Ward"	"881-20-2396"	8
"Amanda"	"Robertson"	"455-19-0708"	8
"Amy"	"Bailey"	"276-19-9235"	8
"Bonnie"	"Gilbert"	"622-53-3302"	7
"Jessica"	"Kelly"	"311-75-6483"	6
"Brian"	"Morales"	"335-71-7747"	6
"Annie"	"Duncan"	"863-96-9468"	6

Abbiamo due persone a parimerito in quanto a *degree centrality* più alta: *Annie George* e *Andrea Montgomery*. Possiamo vedere, inoltre, che tra le persone con centralità più alta compare ancora *Annie Duncan*. Osserviamo quindi la rete di contatti di Montgomery:



Vediamo che lo stesso *Andrea Montgomery* ha commesso un crimine, ma solo uno dei suoi contatti ha commesso un crimine a sua volta: potrebbe non essere il leader di un gruppo criminale. Proviamo a restringere il sotto grafo su cui calcoliamo la *degree centrality* considerando solo le relazioni tra persone che hanno commesso crimini:

```
CALL algo.degree.stream(
'MATCH (p:Person)-[:PARTY_TO]->(c:Crime) RETURN id(p) AS id',
'MATCH (p1:Person)-[:KNOWS]-(p2:Person) RETURN id(p1) AS source, id(p2) AS target',
{graph:'cypher', direction: 'both'})
YIELD nodeId, score
MATCH (p:Person)
WHERE ID(p) = nodeId
RETURN p.name AS name, p.surname AS surname, toInteger(score) AS score
ORDER BY score DESC
LIMIT 10;
```

name	surname	score
"Phillip"	"Williamson"	5
"Brian"	"Morales"	4
"Alan"	"Ward"	4
"Kathleen"	"Peters"	4
"Jessica"	"Kelly"	3
"Diana"	"Murray"	2
"Jack"	"Powell"	2
"Raymond"	"Walker"	2
"Lillian"	"Martinez"	1
"Donald"	"Robinson"	1

Le persone con *degree centrality* più alta sono tutti membri dello stesso gruppo di *Philip Williamson*, emerso nell'analisi delle comunità con *Triangle Counts* e *Louvain Methods*: deve essere senz'altro il gruppo più numeroso e maggiormente connesso del nostro database.