UNIVERSITÀ DEGLI STUDI DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Informatica

Percrso di studio
Scienze e Tecnologie Informatiche

Elaborato finale

Titolo?

Sottotitolo (alcune volte lungo - opzionale)?

Supervisore
Alberto Montresor

Laureando
Stefano Leonardi

Anno accademico 2017/2018

Ringraziamenti

...thanks to...

Indice

1 Sommario

Questa tesi va a ripercorrere il lavoro svolto da Stefano Leonardi durante il tirocinio svolto all'interno dell'università, con la supervisione del dottorando Nasrullah Sheikh, per il professore Alberto Montresor.

1.1 Introduzione

L'intero tirocinio è basato sull'articolo intitolato "Community-enhanced Network Representation Learning for Network Analysis"[17] che può essere abbreviato con la sigla CNRL . Questo articolo propone un innovativo metodo per individuare delle comunità all'interno dei grafi. Lo scopo di questo tirocinio è stato quindi quello di replicare i risultati ivi proposti, una volta compreso il codice alla base, e, dove possibile, cercare di migliorare le prestazioni dell'algoritmo proposto, anche sfruttando elementi non originariamente considerati.
L'algoritmo proposto nell'articolo sfrutta diverse tecniche, alcune delle quali dettagliatamente spiegate negli articoli:

- "DeepWalk: Online Learning of Social Representations"[13]
- "node2vec: Scalable Feature Learning for Networks"[4]
- "How exactly does word2vec work?"[11]

Di seguito la spiegazione del perché l'individuazione di comunità è una tecnica così importante.

1.2 Cosa sono i grafi e cosa le comunità

Alla base di tutto vi sono i grafi, una particolare struttura dati composta da nodi/vertici V (dove $n = |V|$) e archi/bordi/lati E (dove $m = |E|$). I primi possono rappresentare un entità eventualmente anche con l'aiuto di attributi. Diversamente gli archi sono dei collegamenti che legano due nodi, questi possono essere interamente orientati o non orientati. Questa struttura dati permette di rappresentare tantissime situazioni per tale motivo è così importante.
Lo scopo dell'individuazione di comunità è, come dice il nome, scovare delle comunità, ossia dei gruppi di nodi simili fra loro.
È importante notare che i nodi possono essere simili per molte diverse ragioni, possono avere le stesse caratteristiche strutturali in quanto sono dei cardini che legano molti altri elementi, magari dei punti isolati da tutto il resto, o più semplicemente hanno i valori di alcuni attributi analoghi.
Trovare nodi simili permette di gestirli in maniera omogenea e di sfruttarne le peculiarità. Visto che con i grafi possono rappresentare molte situazioni anche le comunità risultano essere molto versatili, per tale motivo sono tanto importanti.

1.3 Come si valutano le comunità

È stato spiegato cosa sono i grafi e cosa sono le comunità che vi si possono costruire sopra, ma non è stato spiegato come si può decidere, date due partizioni differenti su un grafo, quale di queste due è la migliore. Intuitivamente se il grafo è molto piccolo tanto da poterlo comprendere guardando unicamente una sua rappresentazione grafica, e le comunità individuate sono poche allora si potrebbe decidere anche ad occhio qual è la migliore partizione.
Se però le comunità individuate sono tante e il grafo non è più piccolo, allora non è più così facile effettuare una scelta e tanto meno riuscire a giustificarla. Esistono perciò dei metodi formali per analizzare e decidere cosa andare a preferire, questi metodi prendono il nome di modularità.

Prima di parlare di ciò è però necessario formalizzare il concetto di partizione per comprendere di cosa si sta parlando:

- Una partizione è un insieme di comunità
- Non tutte le comunità hanno le stesse dimensioni, ossia lo stesso numero di nodi che vi appartengono
- Le comunità non hanno né una dimensione massima né una dimensione minima.
  In realtà si può considerare che la loro dimensione cada nell'intervallo $[1, n]$, dove il limite inferiore 1 è dato dal fatto che devono contenere almeno un elemento, mentre il limite superiore n è dato dal fatto che non può contenere più nodi di quelli esistenti nel grafo (che sono in numero di n)
- È possibile che un nodo non appartenga a nessuna comunità
- Un nodo del grafo può appartenere a più di una comunità, potenzialmente anche a tutte quelle presenti, questo significa che c'è una sovrapposizione di comunità

Come accennato la metrica di valutazione di una partizione è la modularità (il cui simbolo è Q). Questo nome può essere associato a diverse metriche infatti a seconda di cosa cerchiamo avremo interesse a valorizzare certi aspetti e penalizzarne altri, per tale motivo la modularità è solo il nome usato per indicare la valutazione di una partizione.
Durante il tirocinio sono stati adottati tre metodi differenti: mod_withMax , mod_overlap e modularità modificata . I primi due nomi non sono ufficiali in quanto sono stati scelti al solo scopo di riconoscerle, mentre l'ultimo modularità modificata è quello utilizzato nell'articolo su CNRL [**?**].
I dettagli sul funzionamento e del perché le abbiamo scelte sono all'interno dell'apposito capitolo sulle metriche di valutazione.

1.4 I cambiamenti apportati

Per calcolare la partizione il codice di CNRL utilizza un sistema di visite random e grazie a questo ricostruisce le possibili somiglianze fra i nodi. Non volendo toccare questa sezione le modifiche da noi riportate si inseriscono all'interno della creazione grazie a tali visite, dei cammini.
Una cammino è definito come una sequenza di nodi, preso un elemento qualsiasi fra questi deve esser possibile arrivare all'elemento successivo attraversando un solo arco, ossia i due elementi devono essere direttamente connessi, almeno nel senso di percorrenza.
L'algoritmo genera x cammini, ognuno di lunghezza massima l, sul grafo, dove $x = n \cdot w$:

- n: è il numero di nodi del grafo
- w: è il numero di cammini che si fanno partire da ogni singolo nodo
- l: è la lunghezza massima di ogni cammino

Con questi dati l'algoritmo di CNRL calcola la suddivisione in comunità, si può notare come tutti questi dati vengono estratti esclusivamente dalla struttura del grafo, ossia da come nodi ed archi sono interconnessi.
L'intuizione sfruttata per modificare l'algoritmo sta nel fatto che esistono altri dati non utilizzati, ossia gli attributi dei nodi. Tramite la creazione di nuovi particolari cammini è possibile far risultare due nodi in precedenza lontani, in quanto non direttamente connessi, vicini, in quanto questi due nodi condividono uno o più attributi. È facile intuire che l'introduzione di tutti questi nuovi dati porta ad un radicale cambiamento della suddivisione in comunità.

1.5 Risultati

Parte temporaneamente mancante...

2 Implementazione

In questo capitolo vengono spiegati i metodi utilizzati per cercare di migliorare le prestazioni dell'algoritmo di CNRL .

2.1 Come funziona il codice di CNRL

Prima di tutto, è necessario spiegare come il codice di CNRL funziona.
In prima istanza vengono presi in input tutti i dati necessari, tra cui anche il grafo su cui si andrà a lavorare, questo vien caricato dal formato edgelist, ossia una lista d'archi identificati da nodo di partenza più nodo d'arrivo. Il grafo vien ora preprocessato per andare ad utilizzare l'algoritmo di Node2Vec per effettuare le visite. Node2Vec è un algoritmo che regola le visite sulla base di due parametri p e q, questi due valori possono far variare l'esplorazione del grafo da una visita similare a quella in ampiezza fino ad una visita similare a quella in profondità, permettendo anche delle varianti intermedie.
I cammini individuati grazie alle visite effettuate con questo algoritmo devono sottostare a ben definite caratteristiche. Infatti da ogni nodo (in tutto n) sono fatti partire w cammini, per un totale di $w \cdot n$ cammini generati tutti di lunghezza al più l. Tali cammini sono lunghi al più l e non esattamente l in quanto quest'algoritmo gestisce sia grafi orientati che non. Di conseguenza se si visita un grafo orientato è possibile imbattersi in un pozzo, ce lo si può immaginare come un vicolo cieco, in tali situazioni la visita si ferma in quanto non è più possibile procedere, pertanto avrà una lunghezza inferiore a l. Diversamente se si parla di grafi non orientati tale problema non si pone in quanto, anche se si arrivasse ad un pozzo, sarebbe sempre possibile tornare indietro, infatti Node2Vec permette di ritornare su nodi già visitati.

Durante la creazione di queste cammini tramite le visite ci sono alcuni casi limite, perfettamente gestiti dall'algoritmo, ma comunque interessanti da riportare.

- **nodo isolato su grafo orientato:** si avrà un cammino di lunghezza 1, che comprende unicamente quel nodo
- **nodo isolato su grafo non orientato:** si avrà un cammino lungo l contenente l'ID di quel nodo in continuazione
- **loop su grafo orientato:** questa è una sequenza di nodi che si ripete costante attraverso il cammino, è possibile iniziare con un nodo già presente nel loop o arrivarci dopo svariati passaggi
- **sequenze fisse finali su grafo orientato:** è possibile che tanti cammini terminino con la stessa sequenza di nodi, questo è dovuto al fatto che sul grafo vi è un percorso senza possibilità di scelta che porta ad un pozzo, quindi una volta imboccato tale percorso il cammino andrà a terminare sempre con la stessa sequenza

Una volta che Node2Vec ha generato i cammini è compito di un'altra sezione ricavarne le comunità, questo può essere fatto mediante due metodi differenti. Il primo, il più noto, è Word2Vec che va a considerare gli ID dei nodi come se fossero parole di conseguenza un cammino ossia un array di nodi vien considerato come un array di parole interpretabili come una frase. I dettagli del funzionamento si possono trovare nell'articolo "How exactly does word2vec work?"[**?**].
Diversamente il codice di CNRL utilizza, di default, un algoritmo creato appositamente per questo contesto, non ne riportiamo i funzionamenti esatti in quando non sono stati investigati in maniera approfondita e non esiste una documentazione dettagliata che li spieghi.
Che il codice usi uno o l'altro dei due metodi previsti, entrambi ritornano un insieme di comunità

che formano la partizione che stiamo cercando. È ora possibile calcolare la modularità[1] per dare una misurazione al lavoro svolto.

2.2 Introduzione degli attributi

Nei passaggi precedenti è stato evidenziato come il codice di CNRL utilizza unicamente la struttura del grafo, ossia come nodi ed archi sono interconnessi fra di loro, per individuare le comunità esistenti. Tuttavia entrambi i dataset su cui si è lavorato durante il tirocinio, denominati Cora e Citeseer (meglio spiegati nel capitolo sugli esperimenti) presentano oltre al file che delinea la struttura del grafo anche un secondo file, con estensione .content, che contiene per ogni nodo una lista di valori binari, il cui significato è:

- 0 il nodo non ha l'attributo identificato da questa colonna
- 1 il nodo ha l'attributo identificato da questa colonna

Tutte le informazioni contenute in questo file non sono considerate dal codice di CNRL ed è per questo motivo che si è deciso di introdurle. La finestra di codice in cui è possibile apportare modifiche si apre dopo che i dati sono stati caricati dall'input, e si chiude una volta che inizia il processo d'estrazione delle comunità, ciò significa che la sola sezione in cui è possibile intervenire apportando delle modifiche è la generazione dei cammini lungo il grafo.
I cammini che si vanno a generare allo stato attuale non danno informazioni sugli attributi, pertanto è necessario oltre a questi aggiungerne altri che considerino tali informazioni. In particolare due nodi sempre allo stato attuale sono legati unicamente se vi è un arco che va da uno all'altro, intuitivamente ora possiamo dire che due nodi sono vicini e quindi legati se sono simili.
Questa somiglianza, se riportata sugli attributi, la possiamo definire come: due nodi sono simili se condividono uno o più attributi, contrariamente non sono simili se non condividono nessun attributo. Per come abbiamo definito la somiglianza di due nodi attraverso gli attributi, questa non presenta una direzione pertanto la rappresenteremo attraverso un arco non orientato.
È quindi ora possibile creare un nuovo grafo non orientato che tenga conto di tutte le somiglianze derivanti dagli attributi, e su questo richiamare gli algoritmi di Node2Vec o Word2Vec per generare i nuovi cammini.

2.3 Grafo degli attributi

In questa sezione verrà formalizzato il concetto di grafo degli attributi ed una volta descritte le due alternative se ne faranno i paragoni.
Prima di tutto è necessario rimuovere gli archi originari del grafo, in quanto se li tenessimo rischieremmo di generare una seconda volta gli stessi cammini. Diversamente i nodi vanno conservati, in quanto sono questi i punti che vogliamo nuovamente collegare, da ora in avanti questo insieme di nodi prende il nome di N1.
Come visto, se due nodi condividono lo stesso attributo, sono simili e quindi connessi, esistono tuttavia due metodologie per connettere i due nodi, a e b, in maniera diretta o indiretta.

- diretta consiste nel creare un arco non orientato (l'arco (a, b)) fra i due nodi, è il metodo semplice
- indiretta consiste nel creare un nuovo nodo c, che verrà inserito nell'insieme N2, e creare due archi che collegano ogni nodo originario a c, quindi si creano i due archi (a, c) e (b, c)
  questo metodo andrà a formare un nodo da ogni attributo esistente sul grafo, e li inserirà in N2

Nelle figure 2.1 e 2.2 sono mostrati i due possibili metodi di collegamento dei nodi attraverso gli attributi. In ambo le figure possiamo osservare tre scene:

1. ci sono unicamente i nodi appartenenti all'insieme N1 a cui vanno aggiunti gli archi

[1] Spiegata in dettaglio nell'apposito capitolo sui metodi di valutazione

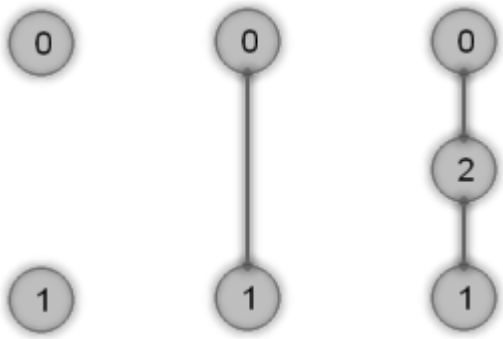

Figura 2.1: Due nodi condividono un attributo, ecco come possiamo legarli, 1-nessun arco, 2-ipergrafo (completo), 3-grafo bipartito (a stella)

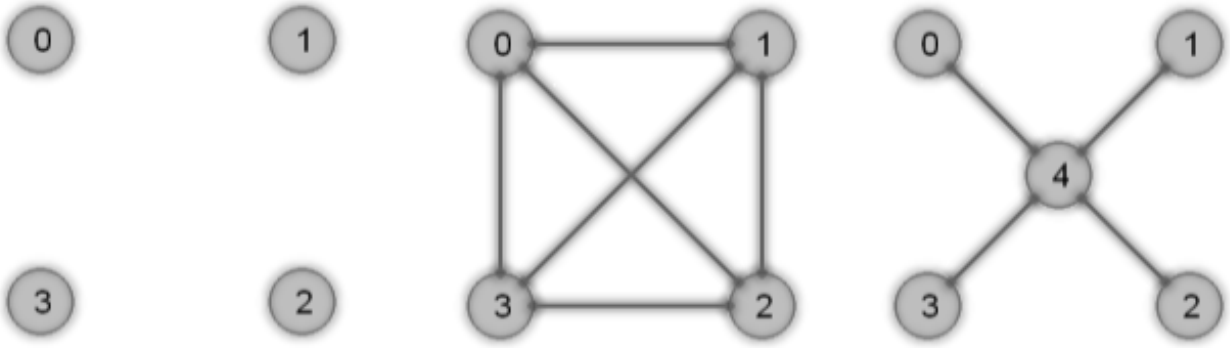

Figura 2.2: Quattro nodi condividono un attributo, ecco come possiamo legarli, 1-nessun arco, 2-ipergrafo (completo), 3-grafo bipartito (a stella)

2. è mostrato il collegamento diretto, da ogni punto raggiungo qualsiasi altro elemento attraversando esattamente un arco, ossia devo fare un singolo passo
   Questa nuova struttura prende ora il nome di ipergrafo

3. il collegamento è indiretto in quanto si passa attraverso il nodo 2 in figura 2.1 e attraverso il nodo 4 in figura 2.2, questo nodo di collegamento è quello generato dagli attributi e pertanto apparterrà ad N2 è necessario utilizzare esattamente due passi per raggiungere un altro nodo
   Questa nuova struttura prende ora il nome di grafo bipartito

Si può notare come i due modelli siano molto differenti nella seconda scena si va a creare un grafo completo, ciò vuol dire che il numero di archi presenti è altissimo infatti segue la legge binomiale $\binom{n}{2}$ dove n rappresenta il numero di nodi coinvolti.
Per il caso in figura 2.2 vengono aggiunti $\binom{4}{2} = 6$ archi però il numero di nodi rimane invariato.
Diversamente nella terza scena viene costruito un grafo a stella, si ha ossia un nodo centrale che fa da fulcro per collegare tutti gli altri nodi che sono legati esclusivamente a lui. Questo comporta che sempre in figura 2.2 vengano aggiunti n archi e un nodo.

Dipendentemente dalla situazione andremo a preferire uno piuttosto che l'altro, ecco le motivazioni che hanno spinto noi alla scelta.
Se vi sono tanti attributi e ognuno di questi appartiene ad un estremamente ristretto numero di nodi allora ci si può permettere di creare degli ipergrafi. Diversamente se ci sono anche pochi attributi che sono condivisi fra un alto numero di nodi la scelta ricade sul grafo bipartito.
Un esempio di quest'ultimo caso è facile da ideare, se esiste un attributo condiviso fra 100 nodi questo significa che con il grafo bipartito andremo ad aggiungere un nodo e 100 archi, mentre con l'ipergrafo non si aggiungerà nessun nodo ma in compenso bisogna creare $\binom{100}{2} = 4950$ nuovi archi.
Nei casi che abbiamo dovuto gestire situazioni come quella presentata in quest'esempio sono presenti e quindi per limitare il numero di archi da gestire abbiamo optato per il grafo bipartito.

È importante far presente che quando si effettuano visite sul grafo bipartito si incontrano tutti i nuovi nodi dell'insieme N2, che sono sconosciuti alle altre parti dell'algoritmo di CNRL , per questo motivo una volta che Node2Vec ha generato i cammini su tale grafo bisogna rimuovere da ognuno di questi gli ID dei nuovi nodi generati, altrimenti i futuri passaggi potrebbero ritornare un errore. Questo non sarebbe necessario se avessimo scelto l'ipergrafo in quanto non va a creare nessun nuovo nodo.

2.3.1 Grafo bipartito e ipergrafo, origine dei nomi

Poco sopra sono stati introdotti questi due nuovi nomi, ma senza un adeguata spiegazione delle loro origini. Le figure 2.1 e 2.2 mostrano la situazione di alcuni nodi prendendo in considerazione un unico attributo. Tuttavia nei casi reali vi sono molti attributi di conseguenza entrambe le nuove strutture sono un insieme di tante piccole sezioni, il grafo bipartito è un insieme di grafi a stella, mentre l'ipergrafo è un insieme di grafi completi.
Nel dettaglio:

Grafo bipartito

Il grafo è detto bipartito in quando vi sono due insiemi di nodi, quelli che in precedenza son stati chiamati N1 (nodi originali) e N2 (nodi derivanti dagli attributi). Tutti gli archi del grafo vanno da un nodo di N1 ad un nodo di N2 e viceversa visto che non sono orientati. Non esistono archi che vanno da N1 a N1 o da N2 a N2.
Questo è esattamente quello che ci si aspetterebbe dalla situazione, un arco da N1 ad N1 è un arco originario, che quindi non necessita dell'attributo per passare da un punto ad un altro, ma questi sono stati rimossi in principio. Mentre un arco da N2 a N2 semplicemente non avrebbe significato, due attributi legati non hanno motivo d'esistere.

Ipergrafo

Gli ipergrafi nella loro definizione formale non presentano semplici archi ma dispongono di iperarchi. I normali archi possono essere rappresentati mediante una linea in quanto presentano due estremità entrambe legate ad un nodo, diversamente gli iperarchi non possono essere rappresentati da una linea in quanto potrebbero arrivare ad avere infinite estremità, o meglio n estremità dove n è sempre il numero di nodi nel grafo. Tutti i nodi che sono estremità di un iperarco possono raggiungere una qualsiasi delle altre estremità con un passo di lunghezza uno. Se si volesse rappresentare ciò su un grafo normale è necessario costruire un grafo completo per ogni iperarco.
È ora chiaro il collegamento con la situazione presentata, ogni attributo altro non è che un iperarco. Nel nostro caso è stato necessario rappresentare gli iperarchi come archi normali in quanto l'algoritmo di Node2Vec non è pensato per gestire la reale sintassi degli ipergrafi.

2.4 Creazione del grafo - esempio

In questa sezione viene mostrato come un grafo viene trasformato partendo dalla struttura di base fino a diventare un grafo bipartito o ipergrafo.

node	ID attributi	
1	11	
2	11	12
3	10	12
4	11	13
5	11	12
6	12	13
7	13	14

Tabella 2.1: Ad ogni nodo è associata la lista degli attributi che possiede

att	ID nodi			
10	3			
11	1	2	4	5
12	2	3	5	6
13	4	6	7	
14	7			

Tabella 2.2: Ad ogni attributo è associata la lista dei nodi a cui appartiene

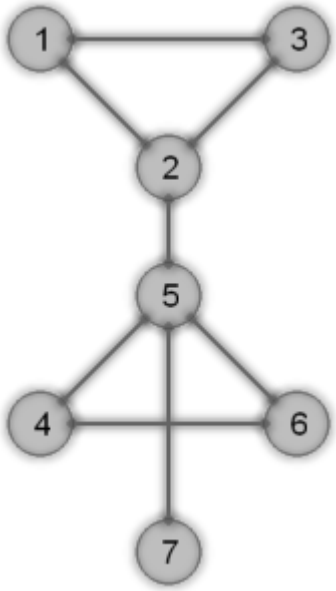

Figura 2.3: Rappresentazione della struttura originaria del grafo

Nella figura 2.3 si vede la struttura del grafo di partenza.
Nella tabella 2.1 si la lista di attributi che ogni nodo possiede.
Mentre la tabella 2.2 è l'inversa della precedente in quanto per ogni attributo sono mostrati tutti i nodi a cui appartiene.

Sono riportate entrambe le tabelle per comodità di comprensione, tipicamente solo una è sufficiente.
È ora possibile mostrare come i due tipi di grafi vengono generati.

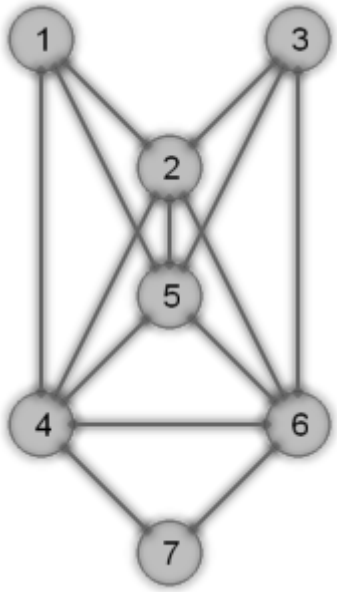

Figura 2.4: Rappresentazione dell'ipergrafo generato

Osserviamo per primo l'ipergrafo mostrato in figura 2.4.

- si può vedere come alcuni archi non sono più presenti, due esempi sono $(1, 3)$ e $(5, 7)$, si ricorda che non sono questi due archi che son stati rimossi in quanto tutti sono stati rimossi semplicemente tutti gli altri sono stati nuovamente aggiunti al contrario di questi
- i nodi che condividono gli stessi attributi vanno a formare un grafo completo, due esempi sono $[1, 2, 4, 5]$ con l'attributo 11 e $[4, 6, 7]$ con l'attributo 13
- se solo un nodo dispone di un particolare attributo, nessun arco sarà creato, questo è il caso dei due attributi 10 e 14.
  Questo può essere spiegato intuitivamente, in quanto si farebbe partire un arco da un nodo ma questo non avrà mai una seconda estremità e quindi non ha senso di esistere, ma anche matematicamente in quanto è questo un caso limite infatti $\binom{1}{2} = error$ la linea numero 1 del triangolo di tartaglia è "1 1" non esiste il terzo elemento

Osserviamo ora la più complessa situazione del grafo bipartito mostrato in figura 2.5. Non è facile capire cosa succede.
I nodi dall'1 al 7, sulla sinistra, sono i nodi originali, mentre i nodi dal 10 al 14 (maggiori o uguali a 10), sulla destra, sono i nuovi nodi generati dagli attributi.

- i nodi sulla sinistra e sulla destra sono separati da un immaginaria linea verticale, e rappresentano i due insiemi di cui si parlava in precedenza ossia N1 e N2
- tutti gli archi attraversano la linea immaginaria, andando da sinistra a destra e viceversa
- tutti gli archi presenti nella struttura originaria di figura 2.3 non sono più presenti in quanto son stati rimossi e mai più rimpiazzati, in quanto non esistono archi da N1 a N1
- i nodi che condividono un attributo creano un grafo a stella, due esempi sono $[1, 2, 4, 5]$ con l'attributo 11 e $[4, 6, 7]$ con l'attributo 13
- se solo un nodo possiede un particolare attributo questo non fa differenza, infatti i due attributi 10 e 14 sono comunque legati alla parte sinistra come ci si aspetterebbe dalle tabelle

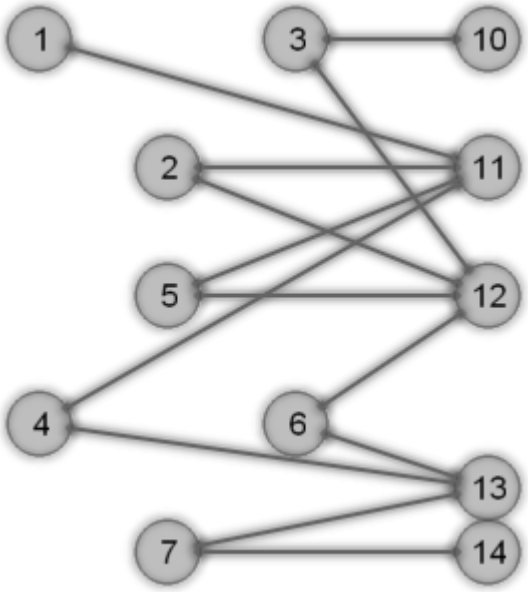

Figura 2.5: Rappresentazione del grafo bipartito generato

3 Metodi di valutazione

Questo capitolo è dedicato alla spiegazione di come si va a valutare una partizione creata dagli algoritmi d'individuazione delle comunità, com'è stato introdotto nel sommario, il metodo scelto nel corso del tirocinio è stato la modularità il cui simbolo è Q.
La modularità altro non è che un valore solitamente compreso all'interno dell'intervallo $[-1, 1]$, anche se questo non è sempre vero in quanto dipende strettamente da che metodo d'implementazione si va ad utilizzare. Le prime persone che hanno lavorato sulla modularità sono state Newman e Girvan, in quanto necessitavano di un metodo formale per decidere fra due partizioni di nodi, quale fosse la migliore.
Newman e Girvan quando hanno scritto le prime applicazioni della modularità sono partiti da una semplice assunzione, un nodo non può appartenere a più di una comunità, questo elimina tutti i problemi dovuti alla sovrapposizione delle comunità. Per i casi studiati durante il tirocinio quest'assunzione non è sempre valida, ma inizialmente è molto comoda in quanto permette di ridurre notevolmente la complessità del calcolo della modularità.
Durante tutto il lavoro svolto si è passati attraverso tre differenti metodi, tutti e tre verranno ora mostrati ed ove possibile spiegati, si parte dal più semplice fino ad arrivare al più complesso.
Durante la spiegazione di ogni metodo viene assunto di lavorare con grafi non orientati, alla fine di ogni sezione sono indicati gli eventuali cambiamenti da applicare in caso si abbia invece a che fare con grafi orientati.

3.1 Modularità con massimi (mod_withMax)

È stato scelto questo particolare nome in quanto questo metodo gestisce al massimo una comunità per ogni nodo, ricalca l'assunzione di base di Newman e Girvan.
L'idea di base è che questo metodo va a calcolare un valore di modularità per ogni comunità, tutti questi elementi sono poi sommati fra loro per andare a creare il valore di modularità del grafo in se. È necessario far notare che questo metodo non permette la sovrapposizione delle comunità, per l'assunzione fatta, di conseguenza è impossibile che la sommatoria finale vada a considerare più di una

volta i valori raccolti, da un qualsiasi elemento del grafo.
Questo metodo si basa sull'osservazione che è possibile identificare una comunità sulla base della frequenza degli archi contenuti al suo interno. Di fatto se un insieme di nodi dispone di una grande quantità di archi che li collegano direttamente allora questi nodi dovrebbero essere una comunità a sé stante o almeno far parte della stessa, se così non è allora la valutazione della partizione sul grafo deve essere penalizzata in quanto non ha considerato questi nodi come parte dello stesso gruppo.
Il motivo per cui abbiamo scelto questo metodo, oltre al fatto che è semplice da calcolare, è proprio perché va a penalizzare non poco le partizioni che non considerano una comunità molto evidente.
Di seguito la formula completa:

$$Q = \sum_{c}^{C_r} \left(\frac{l_c}{L} - \left(\frac{d_c}{2L} \right)^2 \right) \tag{3.1}$$

Dove gli elementi dell'equazione sono:

- Q è il simbolo di modularità
- C_r è l'insieme delle comunità calcolate
- c è l'iteratore sulle comunità
- L è il numero totale di archi all'interno del grafo, solitamente indicato con m
- l_c è il numero totale di archi interni alla comunità c, questo significa che ognuno degli archi qui contati ha ambo i nodi alle estremità appartenenti alla comunità c
- d_c è il grado della comunità c, definito come la sommatoria dei gradi dei nodi che vi appartengono

Si può notare che $\frac{l_c}{L}$ è il peso della comunità c rispetto al resto del grafo, in quanto si calcola la percentuale di archi di cui la comunità c dispone. Mentre $\frac{d_c}{2L}$ è la densità di archi con cui la comunità c ha a che fare, ossia quanto, rispetto al totale, sono importanti i nodi che appartengono a questa comunità.

Quando si parla di grafo orientato l'unica parte da modificare è $\frac{d_c}{2L}$ che diventa $\frac{d_c}{L}$, in quanto in un grafo non orientato un arco incrementa di uno il grado di due nodi quello di partenza e quello d'arrivo, mentre se l'arco fosse orientato questo non accadrebbe in quanto aumenterebbe solamente il grado del nodo di partenza, da qui quel fattore moltiplicativo 2 di differenza.

3.2 Modularità con sovrapposizione (mod_overlap)

L'algoritmo qui descritto è preso dall'articolo intitolato "Modularity measure of networks with overlapping communities"[6].
Come si può intuire dal nome questo metodo permette a due comunità di condividere uno o più nodi e quindi d'aver una sovrapposizione. In linea con diversi altri metodi la modularità generata con questo algoritmo cade nell'intervallo $[-1, 1]$.
Se nel metodo precedente si calcolava un valore di modularità per ogni comunità e poi li si sommava in quanto non era prevista la sovrapposizione, qui la gestione è analoga si calcola un valore per ogni comunità e poi invece di sommarli se ne ricava una media in quanto la sovrapposizione è concessa e quindi uno stesso nodo può portare il suo contributo a diversi elementi della sommatoria.
Di seguito la formula completa:

$$Q = M^{ov} = \frac{1}{K} \sum_{r=1}^{K} \left[\frac{\sum_{i \in c_r} \left(\frac{\sum_{j \in c_r, i \neq j} (a_{ij}) - \sum_{j \notin c_r} (a_{ij})}{d_i \cdot s_i} \right)}{n_{c_r}} \cdot \frac{n^e_{c_r}}{\binom{n_{c_r}}{2}} \right] \tag{3.2}$$

Dove gli elementi dell'equazione sono:

- C_r è l'insieme delle comunità calcolate
- K è il numero di comunità calcolate, pari a $|C_r|$
- c_r è la comunità attuale
- n_{c_r} rappresenta il numero di nodi interni alla comunità c_r
- $n^e_{c_r}$ rappresenta il numero di archi interni alla comunità c_r
- $\binom{n_{c_r}}{2}$ è il numero massimo di archi potenzialmente presenti nella comunità c_r
- d_i è il grado del nodo i
- s_i è il numero di comunità a cui il nodo i appartiene
- a_{ij} vale 1 se l'arco dal nodo i al nodo j esiste, altrimenti prende il valore di 0

È ora il momento di spiegare i dettagli presenti all'interno di quest'equazione (eq: 3.2).
Come per il metodo precedente si va a calcolare la densità della comunità (attraverso la formula 3.3), tuttavia ora si confronta il numero di archi interni presenti rispetto al potenziale numero massimo, ossia il numero di archi presenti in un grafo completo con lo stesso numero di nodi.

$$\frac{n^e_{c_r}}{\binom{n_{c_r}}{2}} \tag{3.3}$$

In aggiunta, si considera la relazione fra il numero di archi interni ed quelli uscenti (tramite la formula 3.4), di conseguenza sembrerebbe che se il numero di archi uscenti è in numero maggiore rispetto al numero di archi interni il valore di modularità vada ad assumere un valore negativo, in caso contrario positivo.

$$\sum_{i \in c_r} \left(\frac{\sum\limits_{j \in c_r, i \neq j} (a_{ij}) - \sum\limits_{j \notin c_r} (a_{ij})}{d_i \cdot s_i} \right) \tag{3.4}$$

In realtà non è sempre così, in quanto si deve tener conto anche del peso degli elementi d_i e s_i, questi due parametri hanno il compito di assegnare ad ogni arco un valore rappresentativo della sua importanza, che sia questo interno o uscente è completamente indifferente. Si può notare come il valore di un arco possa variare all'interno dell'intervallo $]0, 1]$ in quanto un arco ha inizialmente valore 1 e viene poi diviso per due interi positivi moltiplicati fra loro, in seguito viene considerato con peso positivo se favorisce la comunità ossia se è interno, negativo se così non è.
d_i è importante perché va a premiare gli archi che partono da un nodo con un basso grado, il nodo avendo quindi pochi archi, fa si che ognuno di questi sia molto importante, in quanto la somma totale dei pesi degli archi di un nodo è un valore costante uguale per tutti i nodi. Inoltre si premiano i nodi che appartengono a poche comunità, in quanto tali nodi meglio rappresentano la loro comunità d'appartenenza. Se un nodo è comune a tutte le comunità del grafo non è per nulla rappresentativo e quindi ogni volta avrà un influenza estremamente bassa.

È stato scelto questo metodo in quanto prima di tutto permette la sovrapposizione di comunità, ma anche perché andando a calcolare un valore di modularità per ogni comunità individuata nel grafo si può facilmente capire quale di queste dà un maggior contributo al risultato finale e quale invece tende solo a penalizzare la partizione a causa della sua bassa coesione.
Compresa la formula generatrice, si può valutare che un particolare valore di modularità può rivelare due importanti dettagli del grafo:

- un valore negativo sta a significare che i gruppi di nodi non sono densamente connessi al loro interno ma piuttosto sono legati a punti esterni. Pur con un valore negativo è possibile che gli archi interni siano in numero maggiore rispetto agli archi esterni, ma se ciò accade è perché i parametri di d_i e s_i hanno dato maggior rilevanza agli archi esterni

- se la modularità ha un valore che in valore assoluto si avvicina molto allo zero, sta a significare che il grafo valutato è estremamente sparso infatti il fattore moltiplicativo mostrato nell'equazione 3.3 va a collassare i vari elementi della formula vicino allo zero, di conseguenza anche la sommatoria non si allontanerà di molto da quel valore

Quando si parla di grafo orientato l'unica differenza di cui si deve tener conto la si applica alla formula 3.3, si dà il caso che il numero massimo di archi in un grafo orientato sia il doppio del numero di archi per lo stesso grafo non orientato, questo per il semplice motivo che si possono creare due archi leganti gli stessi due nodi se presentano versi opposti. Pertanto il numero massimo cambia da $\binom{n_{c_r}}{2}$ per diventare $2\binom{n_{c_r}}{2}$.

3.3 Modularità modificata

Questo algoritmo[20] è preso dall'articolo intitolato "Incorporating Implicit Link Preference Into Overlapping Community Detection"[19].
È giunto il momento di spiegare la modularità modificata , questo è l'algoritmo che è stato scelto dall'articolo di CNRL ed è per questa ragione che è stato adottato anche da noi. È estremamente importante perché permette di replicare i risultati illustrati nel documento cui facciamo riferimento, si ha quindi un punto fisso da cui partire e con cui all'occorrenza confrontarsi.
Sfortunatamente questo è il più lento dei tre metodi che vengono illustrati in questo capitolo, per il semplice motivo che è il più complesso sia logicamente che computazionalmente.
Di seguito vi sono due versioni della stessa formula (senza e con sistema):

$$Q = \frac{1}{M} \cdot \sum_{u,v \in V} \left[\left(A\left[u,v\right] - \frac{d_{in}\left(u\right) \cdot d_{out}\left(v\right)}{M} \right) \cdot \left| C_u \cap C_v \right| \right] \tag{3.5}$$

$$Q = \frac{1}{M} \cdot \sum_{u,v \in V} \begin{cases} 0 & \left| C_u \cap C_v \right| = 0 \\ \left[\left(A\left[u,v\right] - \frac{d_{in}(u) \cdot d_{out}(v)}{M} \right) \cdot \left| C_u \cap C_v \right| \right] & altrimenti \end{cases} \tag{3.6}$$

Dove gli elementi dell'equazione sono:

- M può assumere due valori a seconda che il grafo sia orientato o no, ossia misura m se indiretto assume invece il valore $2m$ se diretto.
  In tutto questo m è il numero di archi del grafo, ossia come di norma $m = |E|$
- u, v sono gli iteratori sui nodi del grafo, chiamati come di consueto V
- $A\left[u,v\right]$ rappresenta il peso assunto dall'arco (u,v), se tale arco non esiste il valore è 0, mentre se l'arco esiste e non è specificato diversamente il valore considerato è 1. Altri valori maggiori possono essere accettati, ma non quelli negativi
- $d_{in}\left(u\right)$ e $d_{out}\left(u\right)$ sono i gradi entranti e uscenti del nodo u, questi rispettivamente contano il numero di archi entranti e uscenti dal nodo.
  È facile intuire che se il grafo è indiretto, questi due valori per un qualsiasi nodo saranno perfettamente uguali
- C_u è l'insieme delle comunità a cui il nodo u appartiene
- $|C_u \cap C_v|$ è il numero totale di comunità condivise fra i nodi u e v

Come per i metodi precedenti segue una spiegazione delle varie sezioni di tale formula (eq: 3.5).
Cominciando dalla fine si può notare l'elemento $|C_u \cap C_v|$, questo fattore moltiplicativo ci permette d'ignorare gli archi che legano nodi che non condividono nessuna comunità. Per capire il suo scopo si può pensare ad una partizione che non presenta sovrapposizioni, in questo caso tale componente permetterà di ignorare tutti gli archi che non sono interni ad una singola comunità ma passano da

una all'altra, ignorando ora l'assunzione base di Newman e Girvan si può dire che gli unici archi considerati sono quelli che rimangono interni ad almeno una comunità anche se è permesso loro essere di collegamento fra altre. Il funzionamento di questo componente è illustrato bene nella seconda versione della formula (eq: 3.6), grazie al sistema si nota che se la condizione è verificata si può ignorare tutta la prima parte.
Oltre a filtrare gli archi utilizzabili questo componente assegna un importanza maggiore a quegli archi che connettono nodi che condividono molte comunità.
Osserviamo ora qual'è la base che si va ad ignorare o moltiplicare. Si può vedere che si sta parlando di un valore positivo, se l'arco esiste (questo grazie ad $A[u, v]$).

$$\frac{d_{in}(u) \cdot d_{out}(v)}{M} \tag{3.7}$$

Mentre si sottrae al peso dell'arco il fattore definito nell'espressione 3.7, questo rappresenta la rilevanza di un arco, indipendentemente dalla sua esistenza. Se questo collega due nodi con gradi elevati risulta essere uno fra molti e quindi ha un'importanza limitata, se diversamente collega nodi con gradi bassi allora assume molta più importanza perché è un caso isolato e quindi prezioso.

Bibliografia

[1] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, pages 855–864. ACM, 2016.

[2] Anna Lázár, Dániel Abel, and Tamás Vicsek. Modularity measure of networks with overlapping communities. EPL (Europhysics Letters), 90(1):18001, 2010.

[3] David Meyer. How exactly does word2vec work? 2016.

[4] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 701–710. ACM, 2014.

[5] Cunchao Tu, Hao Wang, Xiangkai Zeng, Zhiyuan Liu, and Maosong Sun. Community-enhanced network representation learning for network analysis. arXiv preprint arXiv:1611.06645, 2016.

[6] Hongyi Zhang, Irwin King, and Michael R Lyu. Incorporating implicit link preference into overlapping community detection. In AAAI, pages 396–402, 2015.

[7] Zhiya zhiyzuo. python-modularity-maximization. `https://github.com/zhiyzuo/python-modularity-maximization`.