1 Sommario

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

1.1 Introduzione

L'intero tirocinio è basato sull'articolo intitolato "Community-enhanced Network Representation Learning for Network Analysis" che può essere abbreviato con la sigle 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 magari non considerati dal metodo originale.
L'algoritmo proposto nell'articolo sfrutta diverse tecniche, alcune delle quali dettagliatamente spiegate negli articoli:

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

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 e archi. 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 per l'appunto 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 catalizzatori che legano molti altri elementi, magari dei punti isolati da tutto il resto, o più semplicemente hanno valori degli 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 cade 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 abbiamo adottato tre metodi differenti: mod_withMax , mod_overlap e modified modularity . I primi due nomi non sono ufficiali in quanto li abbiamo scelti al solo scopo di riconoscerle, mentre l'ultimo modified modularity è quello utilizzato nell'articolo su CNRL .
I dettagli del 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 poi 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 di 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 $x = n * w$:

- 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 c'erano 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 nella 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 simile visita in ampiezza fino ad una simile visita 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 fatte 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 questo 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 in un pozzo, sarebbe sempre possibile tornare indietro, infatti Node2vec permette di ritornare su nodi già visitati.

Durante la creazione di queste visite ci sono alcuni casi limite, perfettamente gestiti dall'algoritmo, ma comunque interessanti da essere riportati.

- nodo isolato su grafo orientato: si avrà un cammino di lunghezza uno 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 e quindi un array di nodi vien considerato come un array di parole interpretabili come una frase. I dettagli del funzionamento si possono vedere 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 li abbiamo investigati approfonditamente 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 (capitolo apposito) 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 abbiamo 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, 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 in questa colonna
- 1 il nodo ha l'attributo identificato in 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 in cui è possibile apportare modifiche si apre dopo che i dati sono stati caricati in input, e si chiude una volta che le comunità vengono estratte, ciò significa che la sola sezione in cui è possibile intervenire 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 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 dei 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 od 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 **??** e **??** 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
2. è mostrato il collegamento diretto, da ogni punto raggiungo qualsiasi altro elemento attraversando esattamente un arco, 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 **??** e attraverso il nodo 4 in figura **??**, 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

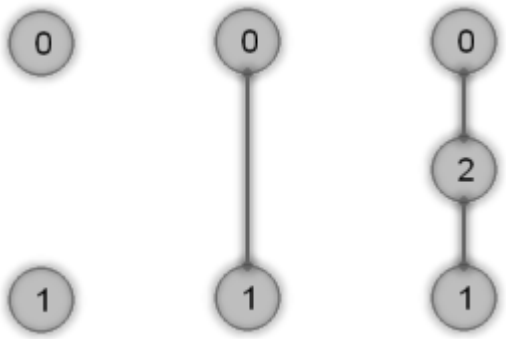

Figura 2.1: Due nodi condividono un attributo, ecco come possiamo legarli, 1-nessun arco, 2-ipergrafo (completo), 3-grafo bipartito (a stella)

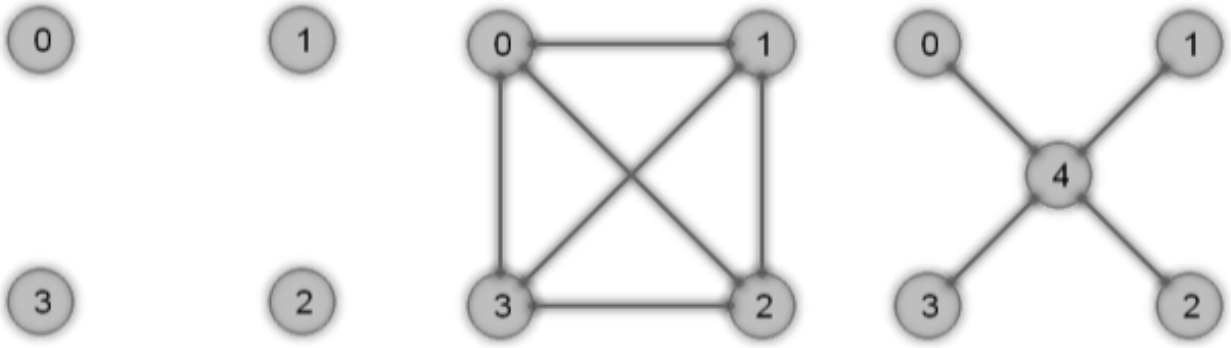

Figura 2.2: Quattro nodi condividono un attributo, ecco come possiamo legarli, 1-nessun arco, 2-ipergrafo (completo), 3-grafo bipartito (a stella)

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 **??** vengono aggiunti $\binom{4}{2} = 6$ archi però il numero di nodi rimane invariato.
Diversamente nella terza scena viene costruito un grafo a stella, 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 **??** 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 creare, 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 cammina sul grafo bipartito esistono 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 dare 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 **??** e **??** 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 ossia che 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

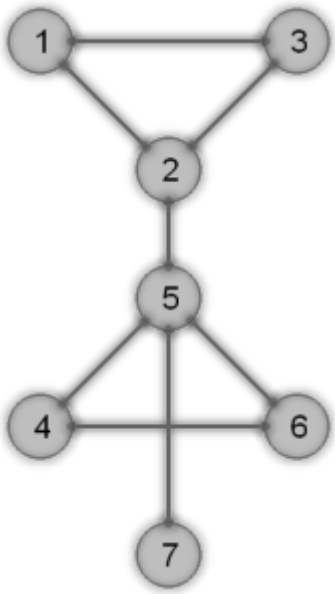

Figura 2.3: Rappresentazione della struttura originaria del grafo

Nella figura **??** si vede la struttura del grafo di partenza.
Nella tabella **??** si la lista di attributi che ogni nodo possiede.
Mentre la tabella **??** è l'inversa della precedente in quanto per ogni attributo sono mostrati tutti i nodi a cui appartiene.
Sono riportate entrambe per comodità, di comprensione, tipicamente solo una è sufficiente.
È ora possibile mostrare come i due tipi di grafi vengono generati.
Osserviamo per primo l'ipergrafo mostrato in figura **??**.

- 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 l 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

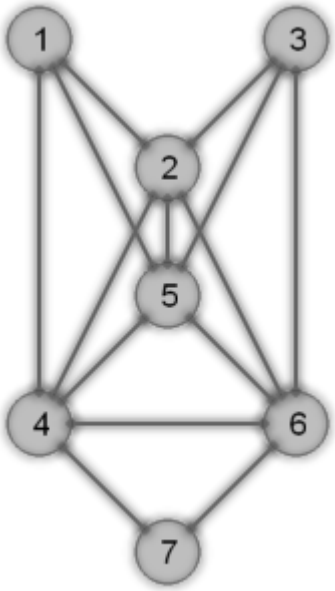

Figura 2.4: Rappresentazione dell'ipergrafo generato

- 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 (0°,1°, 2°) elemento

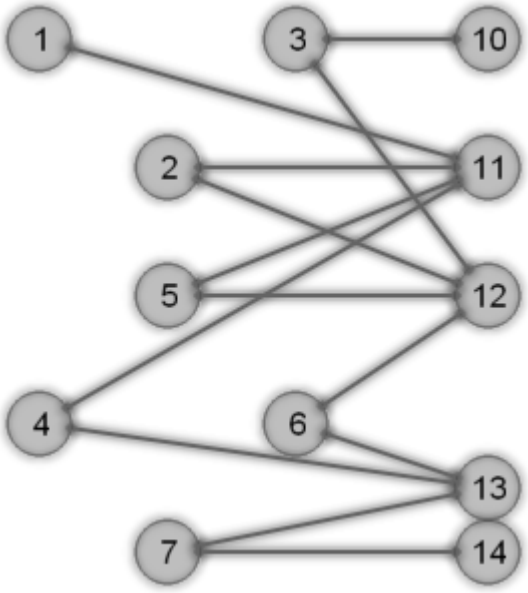

Figura 2.5: Rappresentazione del grafo bipartito generato

Osserviamo ora la più complessa situazione del grafo bipartito mostrato in figura **??**. 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, 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 ?? 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

3 Evaluation methods

In this section, I am going to explain how we evaluate the partitions created by the community detection algorithms, as I introduce in the summary, this method, that we decided to use, is called modularity. Its symbol is Q.
The modularity is a value that is typically contained in the range of $[-1, 1]$, but this is not all times true because it depends on the implementations. The first persons who have worked on the modularity were Newman and Girvan, that like us, they had needed a method for choosing between two partitions of communities on the nodes.
They start from an assumption a node can not belong to more than one community, this assumption is not always true for our situation but it simplifies a lot the complex problem of calculating modularity. So for the first steps, we use the same assumption.
During the time of the internship we use three different methods, not ideate from our, now I am going to explain them from the simplest to the most complex. The formulas reported are for undirected graphs at the end of each subsection there is the differences with directed graphs.

3.1 Modularity with Maximum

We named it in this method because it permits at maximum one community for each node, perfectly in line with the assumption of Newman and Girvan.
The idea is that this method calculates a modularity value for each community in the graph and the total modularity is simply the summation of all the parts. We must specify that, in this method, the overlapping is not considered then the summation does not consider the same elements more than one times.
This method is based on the observation that we can identify a community on the base of the frequency of its edges, in fact, if a set of points have a huge amount of edges inside it probably this is a community if it is not, this is a bad situation that must penalize all the partition.
We chose it because is easy to calculate and because it penalizes a lot the wrong partitions.
Here the formula explained:

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

Where the elements of the formula are:

- Q is the symbol for modularity
- C_r is the set of the communities
- c is the iterator on the communities
- L is the total number of edges in the graph
- l_c is the total number of edges inside, both nodes of each edge are inside the community c

- d_c is the degree of the community c, defined as the summation of the degree of the nodes belong to it

We can see that $\frac{l_c}{L}$ is the weight of the community compared to the others in the graph, and $\frac{d_c}{2L}$ is the density of edges, in other words, the percentage of edges that belong to this community compared to the total number.

To note that when the graph is directed change only a multiplicative factor indeed the "density of edges" change from $\frac{d_c}{2L}$ to $\frac{d_c}{L}$, and this is all.

3.2 Modularity with Overlap

This algorithm is taken from the paper named "Modularity measure of networks with overlapping communities".
As the name says, now the overlapping of the community it is allowed. Then now we do not have a summation but an average. The results are inside a range of $[-1, 1]$.
Here the formula explained:

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

Where the elements of the formula are:

- K is the number of communities $|C_r|$
- c_r is the actual community
- n_{cr} number of nodes in the community
- $n^e_{c_r}$ number of edges in the community
- $\binom{n_{cr}}{2}$ maximum number of edges in the community
- d_i degree of the node i
- s_i number of community to which the node i belong
- a_{ij} 1 if the edge from the node i to the node j exist, 0 otherwise

Now, we try to figure out the particular things that appear in this formula.
As before we calculate the density of the community (with the equation number **??**), but not compared to the total number of edges of the graph, but compared to the hypothetical maximum number of edges inside the nodes considered:

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

In addition, we consider the relationship between the number of inward and outward edges, compared to the total number of edges of the community (in the equation **??**). Then if the outward edges are more than the inward edges, the modularity is negative otherwise is positive.

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

Lastly, when we look at the inward and outward edges we consider the two parameters d_i and s_i, this is important because if a node has only one edge and it belongs only to one community it has a high

weight, differently if it has a hundred of nodes and belongs to dozens of communities, the weight of one of its edges is very low.

We chose this method because it allows the overlapping of communities and this is necessary for our situation. In addition is it possible to figure out which community have a high value, then increase the final results and which one decrease it, cause the low value.
From each modularity value, we can understand two things of the graph:

- If the value is negative mean that the group of nodes is not densely connected inside it, but rather is connected to some external point. It is possible that there are more inwards edges compared to the outwards, but if the value is negative this means that the outwards edges are more relevant
- If the modularity in absolute value is very near at 0 this means that the graph is extremely sparse in fact the multiplication factor (shown in equation **??**) collapse all the elements of the formula near zero

To note that when the graph are directed only the equation **??** change in fact the maximum number of edges is no more $\binom{n_{c_r}}{2}$ but became $2\binom{n_{c_r}}{2}$.

3.3 Modified modularity

This algorithm is taken from the paper named "Incorporating Implicit Link Preference Into Overlapping Community Detection".
Finally, I arrive to explain this method, this is the algorithm choose in the paper of CNRL, for this reason, we looked for it. It is very important because allow our to replicate the results shown in the paper of CNRL, then we can proceed to find a fixed point. Unfortunately, this is the slowest algorithm of the three that I show you in this section because it is the most complex.
Let's look at how it works:

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

Where the elements of the formula are:

- M stands for m if the graph is undirected and for $2m$ if the graph is directed, where m is the number of edges of the graph
- u, v are the iterators on the graph's nodes named V
- $A\left[u,v\right]$ is the weight of the edge uv if the edge does not exist this value is 0, normally a node without weight is considered as 1. It is taken from the adjacency matrix
- $d_{in}\left(u\right)$ and $d_{out}\left(u\right)$ are the incoming/outgoing degree namely the number of incoming/outgoing edges of the node u. If the graph is undirected those values are equal
- C_u is the set of communities to which the node u belong
- $\left|C_u \cap C_v\right|$ is the number of communities that node u and v share

I try to explain this equation in words.[1]
Starting from the end we see the element $\left|C_u \cap C_v\right|$, this is important because allow ignoring the edges that link two nodes of different communities, so only the edges inside a community is considered. In addition, this part is a multiplicative factor, if an edge is inside lots of communities it has a high weight if it belongs only to one community is weight is lower.
But what is the base of this multiplier factor? We can see that this is a positive value if the edge exists (thanks to $A\left[u,v\right]$), in fact, more edges a community has more it is defined. From the weight of

[1] Non sono completamente sicuro delle meccaniche alla base della modularità modificata

the edge we subtract a factor (defined from $\frac{d_{in}(u) \cdot d_{out}(v)}{M}$), this represents the relevance/likelihood of the edge, if it connects two nodes with a high degree, this edge is not too important therefore we decrease its weight slightly, at the opposite is very important if connect two nodes that have low values of degree.