UNIVERSITÀ DEGLI STUDI DI ROMA TOR VERGATA

MACROAREA DI SCIENZE

CORSO DI LAUREA IN INFORMATICA

Tesi di laurea triennale

Analisi e confronto di algoritmi di community detection per grafi di grandi dimensioni

2023-2024

RELATORE:
Paola Vocca

CANDIDATO:
Ludovica Petrella

CORRELATORE:
Giacomo De Luca

Indice

Ringraziamenti

Scrivere ringraziamenti

Introduzione

Nel contesto della crescente complessità dei sistemi sociali, tecnologici e informativi, le reti complesse emergono come modelli fondamentali per rappresentare e comprendere le relazioni tra nodi interconnessi. Questi sistemi, che spaziano dai servizi di social network alle reti di telefonia mobile e al web, possono includere milioni o addirittura miliardi di nodi, creando strutture complesse. La sfida principale nella gestione e nell'analisi di tali reti risiede nella loro dimensione e complessità, che richiedono metodi sofisticati per estrarre informazioni significative dalla loro struttura.

Una delle tecniche promettenti per affrontare questa sfida è la scomposizione delle reti in comunità o sotto-unità, che sono insiemi di nodi altamente interconnessi tra loro. Questa suddivisione è cruciale, poiché può rivelare moduli funzionali precedentemente sconosciuti. La meta-rete risultante, costituita dalle comunità come nodi, offre una rappresentazione semplificata e visivamente comprensibile della rete originale.

Il problema del rilevamento delle comunità consiste nel suddividere una rete in gruppi di nodi densamente connessi, con connessioni più deboli tra gruppi differenti. Questo problema di ottimizzazione, noto per la sua complessità computazionale, ha spinto allo sviluppo di vari algoritmi progettati per trovare partizioni di rete adeguate in tempi ragionevoli. Gli algoritmi per il rilevamento delle comunità si classificano principalmente in tre categorie: divisivi, agglomerativi e basati su ottimizzazione. La qualità delle partizioni ottenute è spesso valutata tramite la modularità, una mi-

sura che indica quanto i collegamenti all'interno delle comunità superano quelli tra comunità diverse.

Nonostante l'uso della modularità come funzione obiettivo, l'ottimizzazione esatta di tale misura rimane computazionalmente difficile, specialmente per reti di grandi dimensioni.

Inoltre, il rilevamento delle comunità si estende oltre la modularità, comprendendo metodi come il Modello Potts Costante (CPM), che affronta alcune delle limitazioni della modularità e introduce un parametro di risoluzione per adattare la densità delle comunità. Questi approcci sono essenziali per comprendere la struttura interna delle reti e per applicare tali conoscenze a vari contesti, dalle reti sociali alle reti biologiche.

La rilevazione delle comunità è quindi un campo di ricerca fondamentale per comprendere la struttura e la dinamica delle reti complesse. Gli sforzi per definire e ottimizzare le comunità nelle reti reali continuano a essere un'area di grande interesse e innovazione.

Capitolo 1

Algoritmi di Rilevamento delle Comunità

In questo capitolo verranno introdotti quattro algoritmi per il rilevamento delle comunità in reti complesse: Louvain, Leiden, Label Propagation e CONGA. Questi algoritmi differiscono per approccio, complessità computazionale e risultati ottenuti, ma condividono l'obiettivo di identificare strutture di comunità in grafi.

1.1 Algoritmi

1.1.1 Algoritmo Louvain

[1] L'algoritmo Louvain identifica gruppi di nodi in una rete massimizzando la "modularità", cioè creando comunità con molte connessioni interne e poche esterne. È veloce e applicabile a reti grandi, rivelando una struttura gerarchica. Ciò permette di analizzare le comunità, scompone la rete in gruppi (o comunità) con un approccio gerarchico. Diversamente dagli altri algoritmi di rilevazione delle comunità, l'algoritmo è limitato più dalla memoria disponibile che dal tempo di calcolo.

Si articola in due fasi, che vengono ripetute iterativamente.

Si assume di partire con una rete pesata di N nodi. Inizialmente, viene assegnata a ciascun nodo una comunità distinta, risultando in una partizione iniziale in cui il

numero di comunità è pari al numero di nodi. Successivamente, per ogni nodo i, si considerano i nodi vicini j e si valuta il guadagno di modularità che si otterrebbe spostando i dalla sua comunità attuale a quella di j. Il nodo i viene quindi collocato nella comunità che massimizza il guadagno di modularità (in caso di pareggio, si applica una regola di risoluzione), ma solo se tale guadagno è positivo. Se non è possibile ottenere un guadagno positivo, i rimane nella sua comunità originale. Questo processo viene applicato ripetutamente e sequenzialmente a tutti i nodi fino a quando non si ottiene ulteriore miglioramento, segnando la conclusione della prima fase. È importante notare che un nodo può essere considerato più volte durante questo processo. La prima fase termina quando si raggiunge un massimo locale della modularità, ovvero quando nessuna mossa individuale può ulteriormente migliorare la modularità.

È importante notare che il risultato dell'algoritmo dipende dall'ordine in cui vengono esaminati i nodi. I primi test indicano che l'ordine dei nodi non influisce molto sulla modularità ottenuta, ma può influenzare il tempo di calcolo. Quindi, studiare come scegliere l'ordine giusto potrebbe essere utile per ridurre i tempi di esecuzione.

Parte dell'efficienza dell'algoritmo deriva dal fatto che il guadagno in modularità ΔQ ottenuto spostando un nodo isolato i in una comunità C può essere facilmente calcolato tramite:

$$\Delta Q = \left[\frac{P_{in} + k_{i,in}}{2m} - \left(\frac{P_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{P_{in}}{2m} - \left(\frac{P_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right],$$

dove:

- P_{in} è la somma dei pesi dei collegamenti all'interno della comunità C,
- P_{tot} è la somma dei pesi dei collegamenti incidenti sui nodi nella comunità C,
- k_i è la somma dei pesi dei collegamenti incidenti al nodo i,

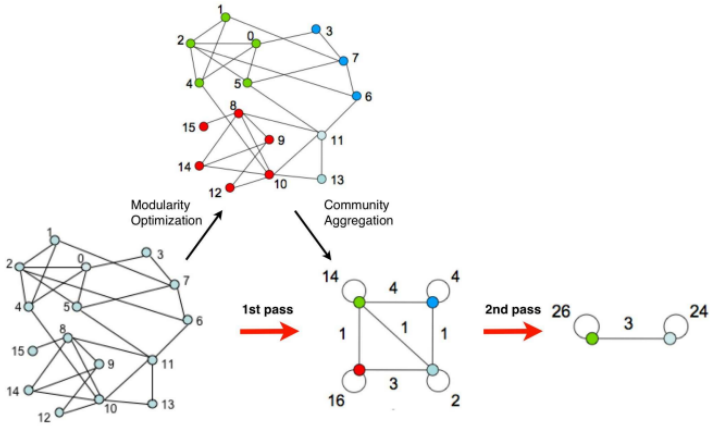
- $k_{i,in}$ è la somma dei pesi dei collegamenti da i ai nodi nella comunità C,
- m è la somma dei pesi di tutti i collegamenti nella rete.

Un'espressione simile viene utilizzata per valutare il cambiamento di modularità quando i viene rimosso dalla sua comunità. In pratica, si valuta il cambiamento di modularità rimuovendo i dalla sua comunità e successivamente spostandolo in una comunità vicina.

Nella seconda fase dell'algoritmo, si costruisce una nuova rete basata sulle comunità individuate nella prima fase. L'algoritmo costruisce una nuova rete ridotta, dove ogni comunità identificata nella prima fase viene trattata come un singolo nodo. Se due comunità (ora due super-nodi) erano connesse nella rete originale, allora ci sarà un collegamento tra i due super-nodi nella nuova rete. Il peso del collegamento tra due super-nodi sarà la somma dei pesi di tutti i collegamenti tra i nodi delle due comunità corrispondenti nella rete originale.

Nel processo descritto, i risultati dell'algoritmo mostrano che nel primo passaggio la rete viene suddivisa in gruppi (partizione naturale). Nel secondo passaggio, i cliques (gruppi di nodi molto connessi) vengono combinati a coppie, cercando di raggiungere il massimo valore di modularità, cioè la migliore suddivisione possibile in comunità.

Una volta completata la seconda fase, si può rieseguire la prima fase sulla nuova rete pesata e ripetere il processo. Ogni ciclo di queste due fasi è chiamato "passaggio". A ogni passaggio, il numero di meta-comunità (gruppi di comunità) si riduce, quindi la maggior parte del tempo di calcolo è speso nel primo passaggio. I passaggi si ripetono fino a quando la struttura della rete non cambia più e si raggiunge il massimo della modularità. L'algoritmo crea una gerarchia in cui si formano comunità di comunità. Il numero di passaggi determina l'altezza della gerarchia, che di solito è basso.

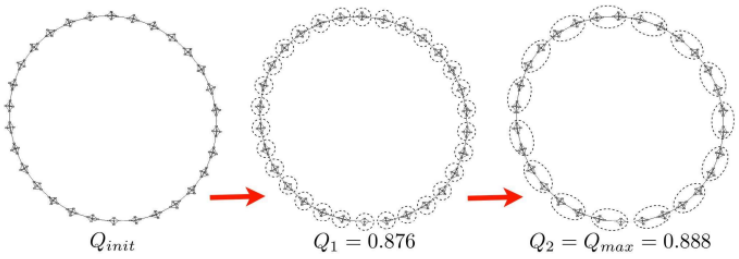


Questo algoritmo ha diversi vantaggi. È semplice da capire e implementare, e non richiede supervisione per funzionare. Inoltre, è molto veloce: le simulazioni su grandi reti mostrano che la sua complessità è lineare, soprattutto su dati tipici e sparsi. Questo perché i miglioramenti nella modularità sono facili da calcolare e il numero di comunità si riduce rapidamente dopo pochi passaggi, concentrando la maggior parte del tempo di esecuzione nelle prime iterazioni.

Louvain supera il problema del limite di risoluzione della modularità grazie alla sua struttura multilivello. È noto che l'ottimizzazione della modularità ha difficoltà a identificare comunità più piccole di una certa dimensione, creando un limite nella rilevazione delle comunità. Questo perché la modularità tende a favorire strutture più grandi e ben definite, ignorando le comunità più piccole. Tuttavia, in questo caso, la prima fase dell'algoritmo sposta i singoli nodi tra le comunità, il che riduce la probabilità che due comunità distinte vengano unite solo spostando nodi singoli. Queste comunità potrebbero essere fuse nei passaggi successivi, dopo che i nodi sono stati aggregati in gruppi più grandi.

L'algoritmo riesce a decomporre la rete in comunità a diversi livelli di dettaglio. Ad esempio, quando viene applicato a una rete di cliques (gruppi di nodi completamente

connessi) , i cliques restano distinti dopo il primo passaggio, anche se vengono poi uniti nella partizione finale. Questo indica che le soluzioni intermedie dell'algoritmo sono utili e che la struttura gerarchica scoperta permette di esaminare la rete a diversi livelli di dettaglio.



In altre parole, mentre l'ottimizzazione della modularità può avere difficoltà a rilevare comunità piccole a causa del limite di risoluzione, Louvain, grazie alla sua struttura multilivello, supera questo problema e fornisce una visione dettagliata della rete a vari livelli di dettaglio.

1.1.2 Leiden

[16] Per quanto Louvain sia un algoritmo molto efficace e ampiamente utilizzato per la rilevazione di comunità nelle reti, presenta alcune limitazioni. Ad esempio, può generare comunità con collegamenti deboli o addirittura produrre comunità che risultano scollegate al loro interno. Inoltre, la modularità utilizzata come funzione obiettivo nell'algoritmo può presentare delle ambiguità, portando a risoluzioni subottimali in cui le comunità non sono ben definite.

Per affrontare queste problematiche, è stato sviluppato un nuovo algoritmo che migliora sia la velocità che la qualità delle partizioni. Questo approccio introduce garanzie esplicite, assicurando che le comunità siano ben connesse al loro interno e garantiscono una convergenza verso una partizione stabile e ottimale. Grazie a questi

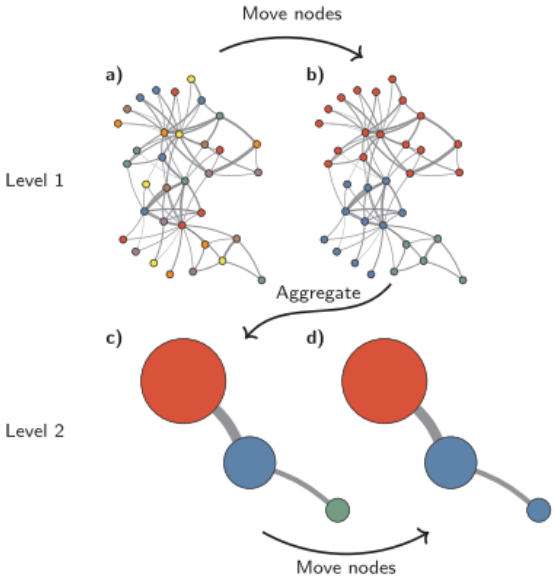
miglioramenti, si riescono a superare alcune delle sfide tipiche riscontrate con Louvain, ottenendo comunità meglio definite e più coese. L'algoritmo ha dimostrato eccellenti prestazioni su vari benchmark e reti reali.

Ottimizza una funzione di qualità come la modularità o il CPM(Constant Potts Model è una misura utilizzata per valutare la qualità delle partizioni delle comunità all'interno di una rete) attraverso due fasi principali:

1. il movimento locale dei nodi
2. l'aggregazione della rete.

Durante la fase di movimento locale, i nodi vengono spostati nella comunità che migliora di più la funzione di qualità. Nella fase di aggregazione, si crea una rete aggregata dove ogni comunità diventa un nodo. Le due fasi vengono ripetute finché la funzione di qualità non può essere migliorata ulteriormente.

Generalmente, l'algoritmo Louvain inizia con una partizione di singleton, in cui ogni nodo è nella propria comunità. Anziché iniziare con ogni nodo come una comunità separata, e si può iniziare con una partizione già identificata da un'iterazione precedente o da un altro algoritmo. Questo può aiutare a trovare miglioramenti in partizioni che erano già abbastanza buone ma non ottimali. L'idea di eseguire più iterazioni consecutive dell'algoritmo, utilizzando ogni volta la partizione dell'iterazione precedente, può essere vantaggiosa per trovare partizioni sempre migliori.

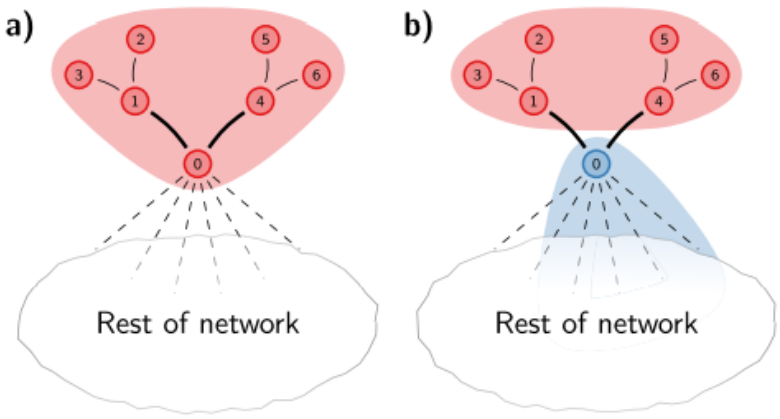


Tuttavia, uno dei limiti dell'algoritmo Louvain è che, pur migliorando il valore della funzione di qualità, come la modularità, può comunque produrre comunità che sono scollegate internamente, cioè parti della comunità possono essere raggiunte solo attraverso connessioni esterne. Questo problema, purtroppo, non è solo teorico ma si osserva frequentemente nella pratica e può persino peggiorare con iterazioni successive dell'algoritmo, nonostante l'apparente miglioramento del valore della funzione di qualità.

Nel funzionamento dell'algoritmo, un nodo può essere spostato in una comunità diversa mentre era un ponte tra diversi gruppi nella sua vecchia comunità. Spostare questo nodo può scollegare la sua vecchia comunità. Si potrebbe pensare che altri nodi nella vecchia comunità vengano spostati per correggere questa disconnessione, ma non è sempre così. Alcuni nodi potrebbero rimanere nella vecchia comunità, anche se essa è diventata scollegata internamente.

Per chiarire il problema, si consideri l'esempio mostrato nella Figura 2. I bordi spessi rappresentano connessioni forti, mentre i bordi sottili indicano connessioni

deboli.



In Figura 2(a), l'algoritmo Louvain ha creato una comunità con i nodi 0–6. I nodi 1–6 sono ben collegati tra loro, mentre il nodo 0 ha molte connessioni esterne. Quando l'algoritmo sposta i nodi nella rete, potrebbe decidere di spostare il nodo 0 in una nuova comunità esterna, come mostrato in Figura 2(b). In questa nuova configurazione, i nodi 2, 3, 5 e 6 sono ben collegati all'interno della loro comunità. Tuttavia, i nodi 1 e 4 hanno connessioni sia interne che esterne e potrebbero essere ancora considerati ben posizionati nella loro comunità attuale, nonostante la comunità sia diventata scollegata internamente.

Sarebbe meglio suddividere la comunità in due gruppi: i nodi 1–3 in una comunità e i nodi 4–6 in un'altra. Tuttavia, l'algoritmo Louvain non considera questa possibilità perché si concentra solo sui movimenti individuali dei nodi. Una volta che i nodi non possono essere spostati ulteriormente, l'algoritmo aggrega la rete. Se una comunità scollegata diventa un nodo nella rete aggregata, non è più possibile suddividerla, quindi rimane scollegata a meno che non venga unita a un'altra comunità.

Questo esempio estremo mostra come Louvain possa trovare comunità scollegate,, ma esistono anche casi più sottili in cui le comunità risultano connesse da legami

molto deboli. In generale, Louvain può individuare comunità con strutture interne poco coese, generando partizioni che presentano collegamenti di qualità insufficiente.

Il problema descritto non è lo stesso del limite di risoluzione della modularità. Il limite di risoluzione può far sì che la modularità unisca comunità più piccole in comunità più grandi, "nascondendo" così le comunità più piccole e creando strutture più grandi di quanto sia realmente significativo. Tuttavia, il CPM non ha questo problema.

Nonostante il CPM non soffra del problema del limite di risoluzione, l'algoritmo Louvain può comunque trovare comunità con collegamenti molto scarsi. Questo significa che l'algoritmo Louvain può avere problemi nel trovare comunità ben collegate, indipendentemente dal fatto che si utilizzi la modularità o il CPM. Quindi, il problema di Louvain con le comunità scollegate è diverso dal limite di risoluzione della modularità.

Anche se l'algoritmo Louvain sembra efficace nel trovare buone partizioni della rete, nella sua forma standard ha solo una garanzia: le comunità rimangono ben separate e non vengono unite.

Tuttavia, iterare l'algoritmo, cioè usarlo ripetutamente migliorando ogni volta la partizione, può sembrare utile. In effetti, quando si itera Louvain, la qualità delle partizioni migliora fino a quando non è più possibile fare altri miglioramenti. A questo punto, ogni nodo è assegnato in modo ottimale, e ci sono due garanzie:

1. nessuna comunità può essere unita

2. nessun nodo può essere spostato ulteriormente

Ma iterare Louvain può anche peggiorare il problema delle comunità mal collegate. Dopo la prima iterazione, l'algoritmo potrebbe creare comunità scollegate, simili al

nodo 0 nell'esempio, senza un modo per correggerle. Quindi, mentre la partizione migliora in alcuni aspetti, può degradarsi in altri.

Questo problema delle comunità scollegate non è nuovo, ma finora non è stato studiato specificamente per Louvain. Le comunità scollegate sono solo un esempio estremo di un problema più ampio: le comunità mal collegate. Risolvere solo i componenti connessi delle comunità non basta, quindi è necessario un approccio più approfondito.

Per questo motivo, è stato sviluppato l'algoritmo Leiden. Questo algoritmo affronta il problema delle comunità mal collegate garantendo che le comunità siano ben connesse. Leiden migliora Louvain utilizzando tecniche di spostamento locale intelligente e movimenti casuali tra vicini, offrendo una soluzione più avanzata e promettente.

L'algoritmo Leiden funziona in tre fasi principali:

1. **Spostamento Locale dei Nodi:** I nodi vengono spostati tra le comunità per migliorare la qualità della partizione.

2. **Raffinamento della Partizione:** La partizione ottenuta viene migliorata ulteriormente. Le comunità nella partizione iniziale possono essere suddivise in sottocomunità più piccole.

3. **Aggregazione della Rete:** Viene creata una nuova rete basata sulla partizione raffinata. Anche se la partizione iniziale viene usata per impostare la rete aggregata, il raffinamento consente di ottenere una partizione finale di qualità superiore.

L'algoritmo Leiden è più complesso del Louvain perché include il passaggio aggiuntivo di raffinamento. Questo permette di ottenere partizioni più precise e di alta qualità. Con il raffinamento, Leiden può migliorare significativamente le partizioni rispetto a Louvain e offre garanzie più solide sui risultati finali.

Nell'algoritmo Leiden, la fase di raffinamento della partizione si compone:

1. Inizializzazione: All'inizio, ogni nodo è considerato come una comunità a sé stante, quindi la partizione raffinata parte da una situazione in cui ogni nodo è isolato.

2. Fusione Locale dei Nodi: I nodi vengono poi fusi in gruppi. Solo i nodi che erano singoli nella partizione raffinata possono essere fusi con altre comunità, ma questa fusione avviene solo all'interno della stessa comunità della partizione iniziale. Inoltre, un nodo può essere fuso solo con una comunità nella partizione raffinata se entrambe le parti sono ben collegate nella partizione iniziale.

3. Selezione della Comunità per la Fusione: I nodi non vengono sempre fusi con la comunità che migliora di più la qualità. Invece, la fusione avviene con una comunità scelta casualmente, ma con una probabilità più alta per le comunità che offrono un incremento maggiore nella qualità. Questo processo di selezione casuale, regolato da un parametro θ, permette di esplorare meglio diverse opzioni e trovare una partizione migliore.

4. Esclusione delle Fusioni Dannose: Le fusioni che riducono la qualità non vengono considerate, il che rende la fase di raffinamento più veloce ed efficace.

La fase di raffinamento di Leiden è progettata per migliorare la partizione iniziale attraverso fusioni selettive e casuali, escludendo quelle che abbassano la qualità, rendendo il processo più efficiente e capace di trovare buone partizioni anche in reti grandi.

L'algoritmo Leiden migliora rispetto a Louvain nella fase di spostamento locale grazie a una procedura più veloce:

1. **Inizio**: Si crea una coda con tutti i nodi della rete, aggiungendoli in ordine casuale.

2. **Spostamento dei Nodi**: Si rimuove un nodo dalla coda e si verifica se spostarlo in una comunità diversa migliora la qualità. Se sì, si sposta il nodo.

3. **Aggiornamento della Coda**: Dopo aver spostato un nodo, si aggiungono alla coda solo i vicini del nodo spostato che potrebbero aver bisogno di essere rivisitati, se non appartengono alla nuova comunità del nodo spostato e non sono già nella coda.

4. **Ripetizione**: Si continua a rimuovere nodi dalla coda e a spostarli finché la coda è vuota.

A differenza dell'algoritmo Louvain, che visita tutti i nodi ogni volta, Leiden visita solo i nodi il cui vicinato è cambiato. Questo rende il processo molto più veloce ed efficiente, poiché evita di ripetere lavoro inutile su nodi che non richiedono ulteriori spostamenti.

L'algoritmo Leiden offre diverse garanzie sui risultati ottenuti durante il processo iterativo di ottimizzazione:

1. **Tutte le comunità sono γ-separate**: Questo significa che non esistono comunità che possono essere fuse tra loro senza compromettere la qualità della funzione ottimizzata (modularità o CPM). La separazione γ è garantita anche dall'algoritmo Louvain.

2. **Tutte le comunità sono γ-connesse**: Questo è un vincolo leggermente più forte rispetto alla semplice connettività. Mentre l'algoritmo Louvain non ga-

rantisce la connettività, l'algoritmo Leiden garantisce che ogni comunità sia connessa.

Un'iterazione stabile è quella in cui la partizione non cambia. Dopo un'iterazione stabile dell'algoritmo Leiden è garantito che:

1. **Tutti i nodi sono localmente ottimamente assegnati**: Non ci sono nodi che possano essere spostati in una comunità diversa migliorando la qualità della funzione. Questo è garantito anche dall'algoritmo Louvain dopo un'iterazione stabile.

2. **Tutte le comunità sono subpartizioni γ-dense**: Una comunità è considerata subpartizione γ-densa se può essere divisa in due parti tali che:

   (a) Le due parti sono ben connesse tra loro;

   (b) Nessuna parte può essere separata dalla comunità di appartenenza;

   (c) Ciascuna parte è anch'essa subpartizione γ-densa.

3. **La densità γ della subpartizione** non implica necessariamente che i nodi individuali siano localmente ottimamente assegnati, ma garantisce che i nodi siano ben connessi alla loro comunità.

Nel caso dell'algoritmo Louvain, dopo un'iterazione stabile, tutte le iterazioni successive saranno stabili, e non ci saranno ulteriori miglioramenti.

Tuttavia, per l'algoritmo Leiden tutte le comunità sono uniformemente γ-dense. Ogni comunità è ben collegata internamente. Non ci sono parti della comunità che possono essere separate senza compromettere la qualità della comunità stessa. In altre parole, anche se si divide una comunità in due, entrambe le parti resteranno ben collegate.

Tutte le comunità sono ottimali per sottoinsiemi: Ogni sottoinsieme di una comunità è ben organizzato. Nessun sottoinsieme può essere spostato in un'altra comunità per migliorare la qualità. Questa è la garanzia più forte dell'algoritmo Leiden e implica anche che le comunità siano uniformemente ben collegate.

Queste garanzie dimostrano che l'algoritmo Leiden non solo trova comunità di alta qualità, ma garantisce anche che queste comunità siano ben collegate e ottimamente organizzate, offrendo miglioramenti rispetto all'algoritmo Louvain.

1.1.3 Label Propagation Method/Algorithm(LPM, LPA)

[10] Questo algoritmo si basa, per trovare comunità nelle reti, sulla propagazione delle etichette. Ecco come funziona:

1. **Inizio**: Ogni nodo nella rete riceve un'etichetta unica.
2. **Iterazione**: Ad ogni passo dell'algoritmo, ogni nodo adotta l'etichetta più comune tra i suoi vicini. Se c'è un pareggio, l'etichetta è scelta casualmente.
3. **Completamento**: Alla fine dell'algoritmo, i nodi con la stessa etichetta sono raggruppati insieme per formare le comunità.

L'algoritmo è vantaggioso per la sua semplicità e velocità. Non cerca di ottimizzare nessuna misura specifica della qualità delle comunità e non richiede di sapere in anticipo quante comunità ci sono o quanto grandi saranno.

Non esiste una definizione unica di comunità in una rete, ma ci sono diversi modi per descrivere cosa può essere considerato una comunità:

- **Clique**: Una comunità è definita come una clique se ogni nodo è collegato a tutti gli altri nodi della comunità. In altre parole, ogni coppia di nodi nella

comunità ha un collegamento diretto tra di loro. Le clique sono un modo molto rigido di definire una comunità: anche la mancanza di un solo collegamento fa sì che il gruppo non sia più una clique.

- **Catena di Clique**: Una comunità è formata da una serie di clique adiacenti, dove due clique sono adiacenti se condividono un numero sufficiente di nodi.
- **k-clan**: Una k-clan è un gruppo di nodi in cui il percorso più breve tra ogni coppia di nodi è di lunghezza al massimo k. Questo significa che si può andare da un nodo all'altro passando solo attraverso nodi all'interno del gruppo.
- **k-club**: Simile alla k-clan, una k-club è un gruppo di nodi che formano una rete con un diametro massimo di k. Tuttavia, una k-club è definita come il più grande gruppo di nodi che soddisfa questa condizione all'interno della rete.
- **Comunità Basata sui Gradi**:
  1. **Comunità Forte**: Un gruppo è una comunità forte se ogni nodo ha più connessioni all'interno del gruppo rispetto a quelle all'esterno.
  2. **Comunità Debole**: Un gruppo è una comunità debole se la somma totale delle connessioni all'interno del gruppo è minore rispetto alla somma delle connessioni esterne.

Queste definizioni forniscono modi diversi di identificare e caratterizzare le comunità all'interno delle reti.

Quando si analizzano le comunità in una rete, ci possono essere diverse partizioni dei nodi che soddisfano una certa definizione di comunità. Per determinare quali partizioni sono le migliori, è utile avere un parametro che misuri la forza delle comunità identificate:

1. **Densità di Archi Osservata vs Attesa**:

   Una comunità U è considerata più forte o robusta se ha una densità di collegamenti interni significativamente più alta rispetto a quella che ci aspetteremmo se gli archi fossero distribuiti casualmente tra i nodi. Supponiamo che all'interno di una comunità U ci siano e_U archi osservati. Se gli archi fossero distribuiti casualmente tra i nodi, possiamo calcolare la probabilità P, cioè la probabilità che il numero di archi in una distribuzione casuale sia maggiore di e_U. Se questa probabilità P è molto piccola, significa che la configurazione osservata (il numero di archi e_U) è altamente improbabile in una rete casuale, e quindi la comunità ha una densità di archi molto superiore a quella attesa per caso. Questo suggerisce che la comunità è robusta e i suoi membri sono fortemente connessi rispetto a una situazione casuale.

2. **Modularità (Q)**: Newman ha introdotto una misura chiamata modularità per valutare la qualità delle comunità. La modularità confronta la densità di archi osservata all'interno delle comunità con la densità attesa in una rete casuale che conserva i gradi dei nodi.

La formula per la modularità è $Q = \sum_i (e_{ii} - a_i^2)$, dove:

- e_{ii} è la frazione di archi osservata all'interno della comunità i.
- a_i^2 è la frazione di archi che ci si aspetterebbe all'interno della comunità i in una rete casuale con gli stessi gradi di nodi.

Un valore di Q vicino a 0 indica che la densità di archi all'interno delle comunità è simile a quella attesa per caso, mentre un valore vicino a 1 indica una struttura comunitaria forte e ben definita.

Quando un algoritmo rileva le comunità in un grafo con n nodi e m archi, generalmente soddisfa due condizioni principali:

1. **Nessuna Sovrapposizione**: Ogni coppia di comunità è disgiunta, cioè, nessun nodo è presente in più di una comunità. Formalmente, $C_i \cap C_j = \emptyset$ per $i \neq j$.

2. **Copertura Completa**: L'unione di tutte le comunità trovate copre l'intero insieme di nodi del grafo. In altre parole, ogni nodo è incluso in almeno una delle comunità. Formalmente, $\bigcup_i C_i$ include tutti i nodi del grafo N.

Queste condizioni assicurano che la partizione risultante dall'algoritmo copra l'intera rete senza sovrapposizioni tra comunità.

Algoritmo Label è caratterizzato da:

1. **Inizio del Processo**:

   - Ogni nodo nella rete inizia con un'etichetta unica.
   - Man mano che l'algoritmo avanza, i nodi aggiornano le loro etichette basandosi su quelle dei loro vicini.

2. **Determinazione della Comunità**:

   - Un nodo decide a quale comunità appartenere in base all'etichetta più comune tra i suoi vicini.
   - In caso di pareggio, l'assegnazione viene fatta casualmente.

3. **Propagazione**:

   - Le etichette si propagano attraverso la rete. Nodi con etichette simili si raggruppano insieme.

- Questo processo continua fino a quando le etichette stabilizzano e formano gruppi ben definiti.

Successivamente vengono aggiornate le Etichette:

1. **Aggiornamento Sincrono**:

   - In ogni iterazione, ogni nodo aggiorna la propria etichetta basandosi sulle etichette dei suoi vicini dell'iterazione precedente.
   - Formula:

$$C_x(t) = f(C_{x1}(t-1), \ldots, C_{xk}(t-1))$$

   - Problema: Nei grafi con strutture particolari, come i grafi a stella, questo metodo può portare a oscillazioni delle etichette.

2. **Aggiornamento Asincrono**:

   - Ogni nodo aggiorna la propria etichetta in base alle etichette aggiornate dei suoi vicini già modificati nella stessa iterazione, e quelle non ancora aggiornate.
   - Formula:

$$C_x(t) = f(C_{xi1}(t), \ldots, C_{xim}(t), C_{xi(m+1)}(t-1), \ldots, C_{xk}(t-1))$$

   - L'ordine in cui i nodi vengono aggiornati è casuale.

All'inizio, ogni nodo ha una propria etichetta unica. Con il tempo, il numero di etichette diminuisce mentre i nodi convergono a un'etichetta comune per ciascuna comunità. Alla fine del processo, i nodi con la stessa etichetta formano una comunità. L'algoritmo Label propagation è un metodo iterativo che riduce le etichette iniziali

a un numero inferiore, raggruppando i nodi in comunità basate sulle etichette più comuni tra i vicini.

L'algoritmo cerca di suddividere una rete in comunità, cioè gruppi di nodi che sono più densamente connessi tra loro rispetto ad altri nodi della rete.

1. **Inizializzazione**:
    - Ogni nodo inizia con un'etichetta unica (il suo identificativo).
2. **Iterazioni**:
    - **Passo 1**: Ordina casualmente i nodi.
    - **Passo 2**: Aggiorna le etichette di ogni nodo basandosi su quelle dei suoi vicini.
        - Ogni nodo sceglie l'etichetta più frequente tra i suoi vicini.
        - Se c'è un pareggio, l'etichetta viene scelta casualmente.
3. **Criterio di Arresto**:
    - Il processo continua fino a quando ogni nodo ha un'etichetta per cui il numero di vicini con la stessa etichetta è il massimo possibile.
    - Questo significa che, per ogni nodo, il numero di vicini con la stessa etichetta deve essere almeno grande quanto il numero di vicini con qualsiasi altra etichetta.

Alla fine, i nodi con la stessa etichetta formano una comunità. I gruppi formati tendono a essere più densi internamente rispetto a come sono collegati ad altri gruppi.

1. **Inizio**:

- Ogni nodo ha un'etichetta unica, quindi inizialmente tutti i nodi sono separati.

2. **Formazione dei Gruppi**:

   - Nelle prime iterazioni, nodi vicini con etichette diverse cominciano a formare piccoli gruppi con etichette condivise (consenso).
   - Questi gruppi si espandono e cercano di includere più nodi.

3. **Competizione**:

   - Quando un gruppo di consenso si avvicina a un altro gruppo, i nodi iniziano a "competere" per attrarre più membri.
   - La competizione dipende dal numero di connessioni tra e dentro i gruppi.

Il criterio finale usato dall'algoritmo è simile alla definizione di comunità forti, ma meno rigido. Invece di richiedere che ogni nodo abbia più vicini all'interno della propria comunità rispetto all'esterno, label richiede solo che il numero di vicini nella stessa comunità sia almeno uguale a quello con altre comunità.

L'algoritmo converge quando tutti i nodi hanno un'etichetta che rappresenta la comunità con il maggior numero di vicini appartenenti alla stessa etichetta. Tuttavia, la convergenza globale può variare a seconda della struttura della rete.

- **Reti Eterogenee**:
  - In reti con strutture di comunità complesse, come molte reti sociali, il processo di propagazione crea gruppi distinti e diversi.
  - Ogni gruppo o comunità tende a stabilire un consenso tra i suoi nodi, portando alla formazione di diverse comunità all'interno della rete.

- **Reti Omogenee**:
  - In reti senza una struttura di comunità evidente, tutti i nodi sono collegati in modo più o meno casuale.
  - In questo caso, l'algoritmo identifica l'intero componente connesso come una singola comunità.

Il criterio di arresto usato nell'algoritmo non cerca di ottimizzare una misura specifica. Si verifica se ogni nodo ha un'etichetta che riflette la maggioranza dei suoi vicini. Questo significa che possono esserci più soluzioni valide, ognuna delle quali soddisfa il criterio di arresto.

Se si conoscono a priori alcuni nodi che probabilmente agiscono come centri di comunità, è possibile inizializzare solo questi nodi con etichette uniche. Gli altri nodi senza etichetta iniziale tenderanno a unirsi ai gruppi guidati dai nodi centrali, facilitando la formazione di comunità più definite.

Se non si conoscono nodi centrali specifici, si inizializza ogni nodo con un'etichetta unica. Questo approccio garantisce che l'algoritmo inizi con una diversità di etichette e poi converga verso una struttura di comunità basata sulle connessioni tra nodi.

1.1.4 Conga

[3] L'algoritmo CONGA gestisce la sovrapposizione dei cluster, dando la possibilità di suddividere i vertici e di permettere che un vertice appartenga a più di un cluster.

L'algoritmo può essere descritto come segue:

1. **Inizializzazione:** Assegna un label unico a ciascun nodo della rete. Ogni nodo è inizialmente considerato come appartenente a un cluster unico.

2. **Calcolo della Betweenness:** Calcola la betweenness degli archi per tutti gli archi nella rete. La betweenness è una misura utilizzata per quantificare l'importanza di un nodo o di un arco all'interno di una rete, basandosi su quanti percorsi più brevi passano attraverso di esso .

3. **Rimozione degli Archi:** Trova l'arco con la betweenness più alta e lo rimuove dalla rete.

4. **Suddivisione dei Vertici:** Quando un arco viene rimosso, considera la possibilità di suddividere i vertici che erano connessi da quell'arco. Ogni vertice può essere suddiviso in più copie, ognuna delle quali può appartenere a un cluster diverso.

5. **Ricalcolo della Betweenness:** Ricalcola la betweenness degli archi rimanenti solo per le componenti contenenti l'arco rimosso.

6. **Ripetizione:** Si ripetono i passi 2-5 fino a quando non rimangono più archi nella rete o fino a quando non si raggiunge un criterio di arresto specifico.

Il passo cruciale dell'algoritmo è la suddivisione dei vertici. Quando un arco viene rimosso, i vertici associati a quell'arco possono essere suddivisi in modo tale che ogni suddivisione possa essere inclusa in diversi cluster. Questo processo consente la creazione di cluster sovrapposti, in contrasto con gli algoritmi di clustering tradizionali che generano cluster disgiunti.

L'algoritmo CONGA permette quindi una rappresentazione più flessibile e realistica delle strutture di comunità nelle reti complesse, dove le sovrapposizioni tra cluster sono comuni e significative.

Nel contesto degli algoritmi di clustering gerarchico divisivo, i cluster vengono suddivisi ripetutamente in cluster più piccoli che, insieme, contengono gli stessi elementi. Per permettere cluster sovrapposti, è necessario trovare un modo per duplicare un elemento in modo che possa essere incluso in più di un cluster quando il cluster si divide.

Nel clustering basato sulla rete, si assume che ogni vertice debba trovarsi nello stesso cluster di almeno uno dei suoi vicini, a meno che non sia in un cluster singleton o in nessun cluster. Pertanto, un vertice v dovrebbe essere suddiviso in al massimo d(v) copie, dove d(v) è il grado di v. Quando l'algoritmo suddivide un vertice, sta essenzialmente cercando di rappresentare il fatto che quel nodo appartiene a più comunità, permettendo al nodo di "duplicarsi" in più copie, con ciascuna copia connessa a una parte dei suoi vicini.

Il concetto chiave dell'algoritmo CONGA è la *betweenness di split* (*split betweenness*), che fornisce un metodo per decidere:

1. Quando suddividere un vertice, invece di rimuovere un bordo.

2. Quale vertice suddividere

3. Come suddividerlo.

Chiaramente, un vertice v dovrebbe essere suddiviso in v_1 e v_2 solo se questi due vertici appartengono a cluster differenti.

Per determinare questo, si può contare il numero di percorsi più brevi che passerebbero tra v_1 e v_2 se fossero connessi da un bordo. Se ci sono più percorsi più brevi tra v_1 e v_2 rispetto a qualsiasi bordo reale, il vertice dovrebbe essere suddiviso; altrimenti, un bordo dovrebbe essere rimosso come di consueto. Questo è il fondamento del metodo di suddivisione di un vertice utilizzato nell'algoritmo CONGA.

Si immagini di avere un nodo v in una rete e vogliamo studiare come il suo "dividere" in due nodi separati influisca sulla struttura della rete:

1. **Suddivisione del Nodo**:
   - Si supponga di dividere il nodo v in due nuovi nodi: v_1 e v_2.
   - Si aggiunga un "bordo immaginario" tra v_1 e v_2. Questo bordo immaginario non esiste nella rete originale, ma si usa per analizzare l'effetto della suddivisione.
2. **Percorsi Più Brevi**:
   - Nella rete originale, i percorsi più brevi che passano attraverso v (dai vicini di v) sono ora cambiati.
   - Invece di passare attraverso i bordi $\{u, v\}$ e $\{v, w\}$, i percorsi più brevi passano ora attraverso $\{u, v_1\}$, $\{v_1, v_2\}$, e $\{v_2, w\}$.
   - Il bordo immaginario ha costo zero, quindi non cambia la lunghezza totale dei percorsi che lo attraversano e non introduce nuovi percorsi più brevi.
3. **Calcolo della Betweenness**:
   - La betweenness di un bordo misura quanto spesso quel bordo fa parte dei percorsi più brevi tra coppie di nodi.
   - Per il bordo immaginario $\{v_1, v_2\}$, si calcola quanto spesso viene usato nei percorsi più brevi tra altre coppie di nodi nella rete.
4. **Trova la Migliore Suddivisione**:
   - Esistono $2d(v) - 2$ modi diversi per suddividere v, dove $d(v)$ è il numero di vicini di v.

- Ogni suddivisione produce un diverso valore di betweenness per il bordo immaginario $\{v_1, v_2\}$.
- La suddivisione che massimizza questo valore è considerata la migliore.

5. **Betweenness di Split**:

   - Il valore massimo della betweenness per il bordo immaginario $\{v_1, v_2\}$ attraverso tutte le suddivisioni è chiamato "betweenness di split" del nodo v. Se questo valore è maggiore della betweenness di qualsiasi altro bordo della rete, la suddivisione del nodo v è considerata vantaggiosa.

La betweenness di vertice $c_B(v)$ può essere calcolata a partire dalla betweenness degli spigoli $c_B(e)$ usando la formula seguente:

$$c_B(v) = \frac{1}{2} \sum_{e \in \Gamma(v)} \left(c_B(e) - \frac{1}{n-1} \right)$$

dove $\Gamma(v)$ è l'insieme degli spigoli incidenti su v e n è il numero totale di vertici nella componente contenente v. Questa formula calcola la betweenness di vertice come una media ponderata delle betweenness degli spigoli che passano attraverso v, con una correzione per la normale quantità di percorsi più brevi che non attraversano v.

Quando si deve determinare se suddividere un vertice v, si può utilizzare la betweenness di vertice come limite superiore alla betweenness di split. Se la betweenness di vertice di v non è superiore alla betweenness di nessun altro bordo, allora non è necessario calcolare la betweenness di split per v, poiché non sarebbe vantaggioso suddividerlo.

Per calcolare la betweenness di split e determinare la migliore suddivisione di v, si inizia calcolando la pair betweenness per ogni coppia di vicini di v. La pair betweenness di v per una coppia $\{u, w\}$, dove u e w sono vicini di v e $u \neq w$, rappresenta il

numero di percorsi più brevi che attraversano entrambi i bordi $\{u, v\}$ e $\{v, w\}$. La betweenness di vertice di v è quindi la somma di tutte queste pair betweennesses.

Per trovare la migliore suddivisione di un vertice v utilizzando le pair betweennesses, si segue un processo basato su un algoritmo iterativo. Inizialmente, rappresentiamo le pair betweennesses di v in un k-clique, dove ogni vertice è etichettato con uno dei vicini di v e ogni bordo $\{u, w\}$ ha un punteggio che indica la pair betweenness di v per la coppia $\{u, w\}$. La procedura per trovare la migliore suddivisione di v è la seguente:

1. **Scelta dell'Arco**: Identificare l'arco $\{u, w\}$ con il punteggio minimo tra tutti gli archi nel k-clique. Questo arco è considerato il candidato per essere unito in un singolo vertice.

2. **Unione dei Vertici**: Unire i vertici u e w in un unico vertice denominato uw.

3. **Aggiornamento dei Bordo**: Per ogni vertice x nel k-clique, sostituire i bordi $\{u, x\}$ e $\{w, x\}$ con un nuovo bordo $\{uw, x\}$ e aggiornare il punteggio di questo bordo a $b1 + b2$, dove $b1$ è il punteggio del bordo $\{u, x\}$ e $b2$ è il punteggio del bordo $\{w, x\}$.

4. **Ripetizione**: Ripetere i passi 1-3 per $k - 2$ volte, fino a ridurre il k-clique a due vertici.

Alla fine di questo processo, i due vertici rimanenti rappresentano la migliore suddivisione del vertice v e il punteggio del bordo tra questi due vertici è la betweenness di split di v.

Tuttavia, è importante notare che questo metodo "greedy" non garantisce sempre di trovare la migliore suddivisione possibile. Per ottenere la soluzione ottimale, sareb-

be necessario testare tutte le possibili suddivisioni, un compito che richiederebbe un tempo esponenziale. Nella pratica, il metodo "greedy" si dimostra molto più efficiente e spesso trova la migliore suddivisione o una buona approssimazione.

Dato che il calcolo delle pair betweennesses può comportare un notevole sovraccarico in termini di tempo e spazio, spesso queste informazioni non vengono utilizzate se la betweenness di vertice di v è inferiore alla betweenness di qualsiasi bordo, indicando che v non deve essere suddiviso. Pertanto, il calcolo della betweenness viene spesso diviso in due fasi per ottimizzare il processo.

1. Calcolare la betweenness di tutti gli spigoli nella rete.

2. Calcolare la betweenness di vertice dei vertici, a partire dalle betweennesses degli spigoli, utilizzando la formula

$$c_B(v) = \frac{1}{2} \sum_{e \in \Gamma(v)} \left(c_B(e) - \frac{1}{n-1} \right)$$

3. Trovare il set di candidati di vertici: quelli la cui betweenness di vertice è superiore alla betweenness di edge massima.

4. Se il set di candidati non è vuoto, calcolare le pair betweennesses dei vertici candidati e poi calcolare la betweenness di split dei vertici candidati, utilizzando la formula

$$c_B(v) = \frac{1}{2} \sum_{e \in \Gamma(v)} \left(c_B(e) - \frac{1}{n-1} \right)$$

5. Rimuovere l'arco con la betweenness di edge massima o suddividere il vertice con la betweenness di split massima (se è maggiore).

6. Ricalcolare la betweenness degli spigoli per tutti gli spigoli rimanenti nella stessa componente o componenti dell'arco rimosso o del vertice suddiviso.

7. Ripetere dal passo 2 fino a quando non rimangono più spigoli.

Nell'algoritmo CONGA, ogni nodo può dividersi in una media di fino a $\frac{2m}{n}$ nodi, portando il numero totale di nodi dopo la divisione a $O(m)$. Il numero di iterazioni resta $O(m)$, mentre il numero di archi rimane invariato, risultando in una complessità temporale nel caso peggiore di $O(m^3)$.

In pratica, la velocità dell'algoritmo dipende dal numero di nodi che vengono divisi: se ne vengono divisi molti, il numero di iterazioni aumenta, la rete cresce e il passo 4 deve essere eseguito più frequentemente. Tuttavia, la divisione dei nodi può anche causare la frammentazione della rete in componenti separate, riducendo così il tempo di esecuzione complessivo.

Capitolo 2

Misure per la Valutazione della Qualità degli Algoritmi nel Rilevamento delle Comunità

In questo capitolo verranno introdotte le principali misure utilizzate per valutare l'efficacia degli algoritmi di rilevamento delle comunità. Queste metriche servono per determinare quanto accuratamente un algoritmo riesce a identificare le strutture di comunità presenti in una rete.

2.1 Misure Utilizzate

F1 Score

[7] L'F1 score è una misura usata per capire quanto bene un modello di classificazione fa il suo lavoro. Si basa su:

precisione (quante delle risposte "positive" sono corrette);

richiamo (quante delle risposte "positive" sono state trovate dal modello). È come una media tra questi due valori, ma è calcolata in modo che entrambi abbiano lo stesso peso.

Il punteggio F1 va da 0 a 1:

- **1** significa che il modello ha funzionato perfettamente.

- **0** significa che ha funzionato male.

$$F_1 = 2 \cdot \frac{\text{precisione} \cdot \text{richiamo}}{\text{precisione} + \text{richiamo}}$$

Dove:

- **Precisione** è il rapporto tra i veri positivi (TP) e il totale dei risultati predetti positivi (TP + FP).
- **Richiamo** (o sensibilità) è il rapporto tra i veri positivi (TP) e il totale dei veri esempi positivi (TP + FN).

La formula si può anche riscrivere in termini di TP (veri positivi), FP (falsi positivi) e FN (falsi negativi):

$$F_1 = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

Quando non ci sono veri positivi, falsi positivi e falsi negativi per una certa classe (ovvero la classe è completamente assente), l'F1 score è indefinito e, per impostazione predefinita, viene restituito come 0.0, con un avviso di `UndefinedMetricWarning`. Questo comportamento può essere modificato tramite il parametro `zero_division`.

Confusion Matrix (Matrice di Confusione)

[8] La confusion matrix (matrice di confusione) è una tabella che mostra come le predizioni di un modello si confrontano con i risultati reali. Divide i risultati in quattro categorie principali:

- **True Positives (TP)**: il modello ha predetto correttamente la classe positiva.
- **True Negatives (TN)**: il modello ha predetto correttamente la classe negativa.

- **False Positives (FP)**: il modello ha predetto la classe positiva quando in realtà era negativa (detto anche errore di tipo I).
- **False Negatives (FN)**: il modello ha predetto la classe negativa quando in realtà era positiva (detto anche errore di tipo II).

Se si ha un modello di classificazione binaria, la matrice di confusione potrebbe essere:

	Predetto Negativo	Predetto Positivo
Vero Negativo (TN)	50	10
Vero Positivo (TP)	5	35

In questo esempio:

- 50 sono i veri negativi (**TN**).
- 35 sono i veri positivi (**TP**).
- 10 sono i falsi positivi (**FP**).
- 5 sono i falsi negativi (**FN**).

Ogni riga rappresenta i valori reali, e ogni colonna rappresenta i valori predetti. Le varie caselle mostrano il numero di campioni che appartengono a ciascuna combinazione di predizione reale/predetta.

2.1.1 Adjusted Rand Index (ARI)

[12] L'Adjusted Rand Index (ARI) è una misura di somiglianza tra due clustering, tenendo conto di tutte le coppie di campioni e misurando quante di queste sono assegnate allo stesso cluster o a cluster differenti sia nella predizione che nei dati reali. La formula dell'ARI è la seguente:

$$ARI = \frac{RI - \mathbb{E}[RI]}{\max(RI) - \mathbb{E}[RI]} \tag{2.1.1}$$

Dove:

- RI è il Rand Index, che misura la concordanza grezza tra due clustering.
- $\mathbb{E}[RI]$ è il valore atteso dell'indice di Rand per un'assegnazione casuale dei cluster.
- $\max(RI)$ è il massimo valore possibile dell'indice di Rand.

Il punteggio 1.0 indica una corrispondenza perfetta tra i cluster (a meno di permutazioni). Un punteggio vicino a 0.0 indica che i cluster sono stati assegnati in modo casuale. Il punteggio può arrivare fino a -0.5 quando i cluster sono particolarmente discordanti. L'ARI è simmetrico, quindi l'ordine degli argomenti non influenza il risultato:

$$\text{adjusted\_rand\_score}(a, b) = \text{adjusted\_rand\_score}(b, a)$$

dove:

- **labels_true**: le etichette reali (ground truth), un array di dimensione ($n\_samples$).
- **labels_pred**: le etichette predette, anch'esse un array di dimensione ($n\_samples$).

2.1.2 Adjusted Mutual Information (AMI)

[11] L'Adjusted Mutual Information (AMI) è una misura che ci aiuta a capire quanto due raggruppamenti (clustering) di dati siano simili, ma in modo intelligente: corregge il punteggio per evitare che risulti "gonfiato" per pura fortuna.

Mutual Information (MI): misura quanta informazione due raggruppamenti condividono. Più alta è la MI, più i raggruppamenti sembrano simili. Tuttavia, c'è un

problema: più si aumenta il numero di gruppi, più il punteggio MI aumenta anche se i raggruppamenti non sono davvero simili.

Adjusted Mutual Information (AMI): corregge il punteggio della MI per eliminare l'effetto dovuto al numero di cluster. In questo modo, l'AMI riflette meglio la vera somiglianza tra i raggruppamenti, evitando falsi risultati positivi dovuti al caso.

La formula dell'AMI è la seguente:

$$AMI(U,V) = \frac{MI(U,V) - \mathbb{E}[MI(U,V)]}{\text{avg}(H(U), H(V)) - \mathbb{E}[MI(U,V)]} \tag{2.1.2}$$

Dove:

- $MI(U,V)$ è la Mutual Information tra i due clustering U (etichette reali) e V (etichette predette).
- $\mathbb{E}[MI(U,V)]$ è il valore atteso della Mutual Information per cluster assegnati casualmente.
- $H(U)$ e $H(V)$ sono le entropie dei cluster U e V.
- $\text{avg}(H(U), H(V))$ è la media delle entropie, calcolata con il metodo specificato (ad esempio, media aritmetica).

L'AMI restituisce un punteggio di 1 quando i due clustering sono identici (perfettamente corrispondenti). Punteggi vicini a 0 indicano cluster casuali o indipendenti. L'AMI può essere negativo se i clustering sono particolarmente discordanti. È indipendente dai valori assoluti delle etichette, quindi la permutazione dei valori delle etichette non modifica il punteggio. L'AMI è simmetrico, quindi lo scambio delle etichette reali e predette non influisce sul risultato.

$$\text{adjusted\_mutual\_infotmation}(a,b) = \text{adjusted\_mutual\_information}(b,a)$$

- **labels_true**: array contenente le etichette reali ($n\_samples$).
- **labels_pred**: array contenente le etichette predette ($n\_samples$).
- **average_method**: metodo utilizzato per calcolare il normalizzatore nel denominatore. Può essere:
  - 'min': la minore delle due entropie.
  - 'geometric': la media geometrica.
  - 'arithmetic': la media aritmetica (predefinito).
  - 'max': la maggiore delle due entropie.

2.1.3 Homogeneity score

[14] L'Homogeneity Score è una metrica utilizzata per valutare quanto un risultato di clustering sia omogeneo rispetto alle etichette di verità a terra (ground truth). Un clustering è considerato omogeneo se tutti i punti all'interno di un singolo cluster appartengono alla stessa classe.

Un punteggio di 1.0 indica che ogni cluster contiene esclusivamente punti dati di una singola classe, quindi il clustering è perfettamente omogeneo. Un punteggio di 0.0 indica che i cluster contengono punti dati provenienti da più classi, quindi non c'è omogeneità. Il punteggio è indipendente dai valori assoluti delle etichette: una permutazione dei valori delle etichette non modifica il punteggio. Questa metrica non è simmetrica: se si invertono i ruoli delle etichette reali e di quelle predette, invece di valutare l'omogeneità, si valuta la completezza. (Homogeneity = "Tutti i punti dentro ogni cluster appartengono alla stessa classe?" Completeness = "Tutti i punti di una stessa classe sono messi nello stesso cluster?")

- **labels_true**: array di etichette reali di dimensione (*n_samples*), che rappresenta la verità a terra.
- **labels_pred**: array di etichette predette di dimensione (*n_samples*), che rappresenta i cluster ottenuti.

2.1.4 Completeness score

[13] Il Completeness Score è una metrica utilizzata per valutare quanto un clustering sia completo rispetto alle etichette di verità a terra (ground truth). Un clustering è considerato completo se tutti i punti appartenenti a una stessa classe (secondo la verità a terra) vengono assegnati allo stesso cluster.

Un punteggio di 1.0 indica che tutti i punti appartenenti a una classe sono assegnati allo stesso cluster, quindi il clustering è perfettamente completo. Un punteggio di 0.0 indica che i membri di una classe sono suddivisi in diversi cluster, quindi il clustering non è completo. Il punteggio è indipendente dai valori assoluti delle etichette: una permutazione delle etichette non cambia il risultato. La metrica non è simmetrica: invertire le etichette reali e predette restituirà invece l'Homogeneity *Score*, che valuta l'omogeneità del clustering.

- **labels_true**: array di etichette reali (ground truth) di dimensione (*n_samples*).
- **labels_pred**: array di etichette predette di dimensione (*n_samples*).

2.1.5 Normalized mutual information score (NMI)

[15] Il **Normalized Mutual Information Score (NMI)** è una misura che dice quanto due raggruppamenti (clustering) di dati sono simili, basandosi su quanta informazione condividono. È una versione normalizzata del punteggio di informazione

mutua (MI), scalata per essere compresa tra 0 e 1. 0 indica che non c'è alcuna informazione condivisa tra le due clusterizzazioni. 1 indica una correlazione perfetta tra le due clusterizzazioni.

- **Indipendente dai valori delle etichette**: una permutazione dei valori delle etichette non cambia il risultato del punteggio.
- **Simmetrico**: invertendo le etichette reali con quelle predette, il punteggio rimane lo stesso. Questo lo rende utile per confrontare due strategie di assegnazione delle etichette indipendenti, anche quando la verità a terra non è nota.
- **Non aggiustato per il caso casuale**: il NMI non tiene conto delle possibili somiglianze casuali tra le due assegnazioni di cluster, quindi può essere preferibile usare il Adjusted Mutual Information Score (AMI) se si desidera una misura aggiustata per il caso.
- **labels_true**: array di etichette reali di dimensione ($n\_samples$).
- **labels_pred**: array di etichette predette di dimensione ($n\_samples$).
- **average_method**: metodo per calcolare il normalizzatore nel denominatore. Può essere:
  - 'min': normalizzazione usando il valore minimo.
  - 'geometric': normalizzazione usando la media geometrica.
  - 'arithmetic': (predefinito) normalizzazione usando la media aritmetica.
  - 'max': normalizzazione usando il valore massimo.

Modularità

[9] La modularità Q è una misura della qualità di una partizione in comunità all'interno di una rete, definita come:

$$Q = \sum_i \left(e_{ii} - \frac{a_i^2}{2} \right)$$

dove:

- e_{ii}: rappresenta la frazione di collegamenti che uniscono nodi all'interno della comunità i rispetto al totale dei collegamenti nella rete. Maggiore è questo valore, meglio i nodi sono raggruppati all'interno delle comunità.
- a_i: è la frazione di tutti i collegamenti che coinvolgono almeno un nodo nella comunità i. Se a_i è elevato, significa che ci sono molti collegamenti che collegano la comunità al resto della rete.

Densità di Modularità

[9] La densità di modularità misura l'intensità dei collegamenti all'interno delle comunità rispetto a quelli che collegano le comunità al resto della rete, ed è definita come:

$$D = \sum_{i=1}^{m} \frac{L(N_i, N_i) - L(N_i, \overline{N_i})}{|N_i|}$$

dove:

- $L(N_i, N_i)$: è la somma dei collegamenti interni alla comunità N_i.
- $L(N_i, \overline{N_i})$: è la somma dei collegamenti tra la comunità N_i e il suo complemento $\overline{N_i}$ (tutti i nodi non appartenenti a N_i).
- $|N_i|$: è il numero di nodi nella comunità N_i.

Una versione generalizzata della densità di modularità è:

$$D_\lambda = \sum_{i=1}^{m} \frac{2\lambda L(N_i, N_i) - 2(1-\lambda)L(N_i, \overline{N})}{|N_i|}$$

Modificando il parametro λ, è possibile ottenere raggruppamenti più dettagliati e gerarchici.

Community Score

[9] La **Community Score** misura quanto bene un gruppo di nodi (chiamato sottografo R) forma una comunità all'interno di una rete più grande. In altre parole, dice quanto è "forte" o "coesa" quella comunità rispetto alla rete totale. La frazione di collegamenti di un nodo i rispetto al sottografo R è definita come:

$$\mu_i = \frac{1}{|R|} k_{in}(R)$$

dove $k_{in}(R)$ è il numero di collegamenti da i verso altri nodi in R.

Il potere medio $M(R)$ di ordine k è calcolato come:

$$M(R) = \sum_{i \in R} (\mu_i)^k$$

La volume V_r di una comunità è definita come il numero totale di collegamenti all'interno della comunità:

$$V_r = \sum_{i,j \in R} a_{ij}$$

Infine, la score della comunità R è definita come:

$$score(R) = M(R) \times V_r$$

La Community Score complessiva per una partizione $\{R_1, R_2, \ldots, R_m\}$ è quindi:

$$CS = \sum_{i=1}^{m} score(R_i)$$

Community Fitness

[9] La **fitness di comunità** $P(M)$ è una misura che valuta quanto bene una comunità M è strutturata all'interno di una rete, basandosi sui suoi collegamenti interni (tra i nodi della comunità) e esterni (tra i nodi della comunità e quelli fuori da essa).

$$P(M) = \sum_{i \in S} k_{in}(M) \left(k_{in}(M) + k_{out}(M)\right)^{\alpha}$$

dove:

- $k_{in}(M)$: è il grado interno del nodo i all'interno della comunità M.
- $k_{out}(M)$: è il grado esterno del nodo i rispetto alla comunità M.
- α: è un parametro di risoluzione che controlla la dimensione delle comunità; un valore più alto tende a produrre comunità più piccole e più coese.

Capitolo 3

Implementazione degli Algoritmi di Rilevamento delle Comunità

In questo capitolo, verranno mostrate l'implementazione di vari algoritmi di rilevamento delle comunità, tra cui Louvain, Leiden, Label Propagation e CONGA.

3.1 Descrizione del Datase

[9] Per confrontare l'efficacia dei quattro algoritmi di rilevamento delle comunità, sono stati utilizzati tre dataset di reti sociali reali. Questi dataset sono ampiamente noti nel campo del community detection e presentano strutture comunitarie ben definite, fornendo un riferimento affidabile per la validazione dei risultati ottenuti.

3.1.1 Rete del Karate Club di Zachary

[6] Questa rete è stata creata da Wayne Zachary, il quale ha studiato le interazioni tra 34 membri di un club di karate nel corso di due anni. Durante lo studio, emerse un disaccordo tra il capo del club e l'istruttore, che portò alla divisione del gruppo in due comunità separate.

Caratteristiche:

- Nodi: 34
- Archi: 78

Comunità	Membri
Mr. Hi	0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 16, 17, 19, 21
Officer	9, 14, 15, 18, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33

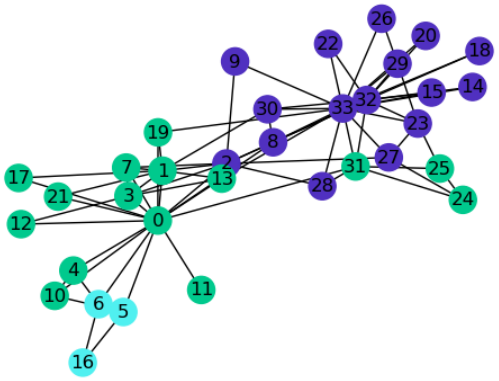


Figura 3.1: Karate club

3.1.2 Rete dei Delfini di Bottlenose

[17] Questa rete è composta da 62 delfini di bottlenose che vivono in Nuova Zelanda, studiata da Lusseau per sette anni. I legami tra i delfini sono stabiliti sulla base di associazioni statisticamente frequenti; ciò significa che se due delfini interagiscono frequentemente, vengono considerati "collegati". **Caratteristiche:**

- Nodi: 62
- Edge: Varie connessioni che riflettono l'interazione tra i delfini.

Comunità	Membri
Comunità 1	Beak, Zap, Shmuddel, TR88, PL, Bumper, SN96, Wave, TR77, Quasi, Zig, Number1, Feather, Gallatin, TR99, Hook, Haecksel, TR120, MN105, Topless, SN90, Cross, TSN103, TR88, Stripes, Upbang, Scabs, Fork, TR82, Thumper, SN63, Zipfel, Fish, DN21, SN96, Vau, Oscar, Number1, Feather, Ripplefluke, Knit, Shmuddel
Comunità 2	Ripplefluke, TSN83, Fish, Oscar, MN23, Jet, Knit, Web, SN89, Double, Five, SN4, Patchback, SN100, MN60, Beescratch, Web, Gallatin, SN89, Zap

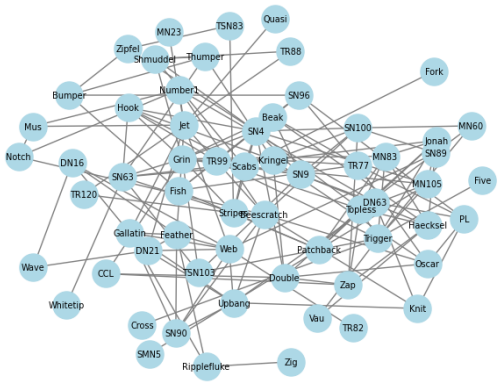


Figura 3.2: Rete dei Delfini di Bottlenose

3.1.3 Rete del Football Universitario Americano

[5] Questa rete, compilata da Girvan e Newman, rappresenta le partite di football americano tra le squadre universitarie. In questa rete, i vertici rappresentano le squadre di football e un edge esiste tra due squadre se hanno giocato una partita durante la stagione:

- Nodi: 115
- Edge: 616

Comunità	**Membri**
Comunità 0	1, 25, 33, 37, 45, 89, 103, 105, 109
Comunità 1	19, 29, 30, 35, 55, 79, 94, 101
Comunità 2	2, 6, 13, 15, 32, 39, 47, 60, 64, 100, 106
Comunità 3	3, 5, 10, 40, 52, 72, 74, 81, 84, 98, 102, 107
Comunità 4	44, 48, 57, 66, 75, 86, 91, 92, 110, 112
Comunità 5	36, 42, 80, 82, 90
Comunità 6	12, 14, 18, 26, 31, 34, 38, 43, 54, 61, 71, 85, 99
Comunità 7	0, 4, 9, 16, 23, 41, 93, 104
Comunità 8	7, 8, 21, 22, 51, 68, 77, 78, 108, 111
Comunità 9	17, 20, 27, 56, 62, 65, 70, 76, 87, 95, 96, 113
Comunità 10	11, 24, 50, 59, 63, 69, 97
Comunità 11	28, 46, 49, 53, 58, 67, 73, 83, 88, 114

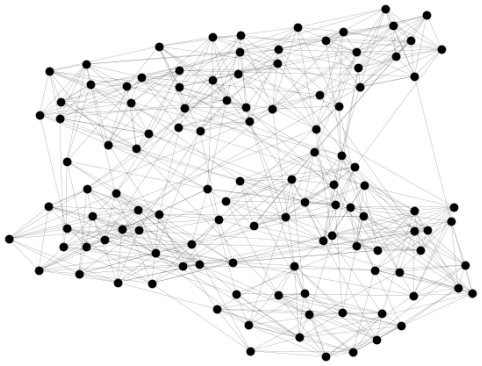


Figura 3.3: Rete del Football Universitario Americano

3.1.4 AS Internet topology

[4]

Il dataset "AS Internet topology" rappresenta la topologia Internet del giugno 2009, estratta dai dati raccolti dall'infrastruttura di misurazione attiva Archipelago, sviluppata dalla Cooperative Association for Internet Data Analysis (CAIDA). Questo dataset viene utilizzato per analizzare la struttura e la connettività della rete Internet, modellata come un grafo.

I nodi rappresentano sistemi autonomi (AS) e gli archi le connessioni tra di essi.

- Nodi: 23.752
- Edge: 58.416

Le comunità identificate nel dataset sono 176, ma a causa del numero elevato di componenti, non vengono riportate.

Dataset	**Numero di Comunità**
Zachary's Karate Club	2
Bottlenose Dolphin Network	2
American College Football Club	12
AS Internet topology	2

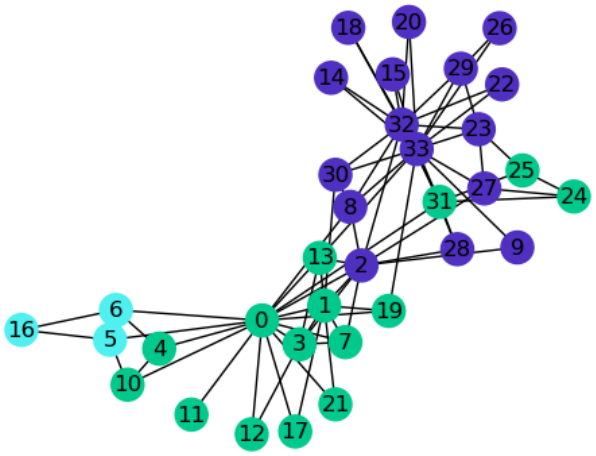
3.2 Implementazione

3.2.1 Dataset Zachary's Karate Club

3.2.1.1 Label propagation

L'algoritmo ha identificato 3 comunità:

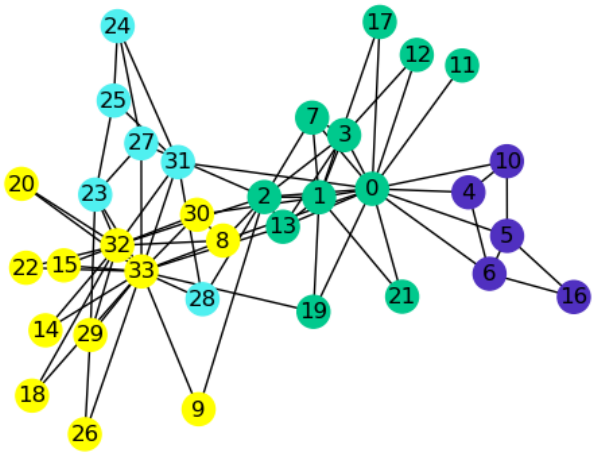
Comunità	Numero di nodi	Nodi
1	15	0, 1, 3, 4, 7, 10, 11, 12, 13, 17, 19, 21, 24, 25, 31
2	17	32, 33, 2, 8, 9, 14, 15, 18, 20, 22, 23, 26, 27, 28, 29, 30
3	3	16, 5, 6



3.2.1.2 Louvain

Questo algoritmo ha individuato 4 comunità distinte:

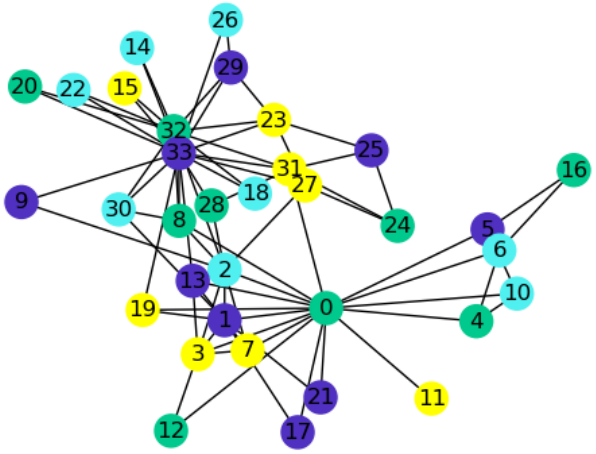
Comunità	Numero di nodi	Nodi
1	14	8, 9, 14, 15, 18, 20, 22, 23, 26, 27, 29, 30, 32, 33
2	4	24, 25, 28, 31
3	6	4, 5, 6, 10, 16
4	11	0, 1, 2, 3, 7, 11, 12, 13, 17, 19, 21



3.2.1.3 Leiden

Anche con questo algoritmo sono state identificate 4 comunità:

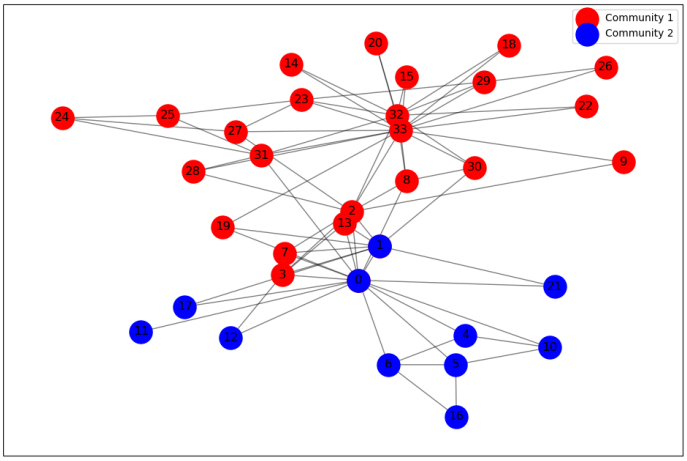
Comunità	Numero di nodi	Nodi
1	12	8, 9, 14, 15, 18, 20, 22, 26, 29, 30, 32, 33
2	12	0, 1, 2, 3, 7, 11, 12, 13, 17, 19, 21
3	5	4, 5, 6, 10, 16
4	6	23, 24, 25, 27, 28, 31



3.2.1.4 Conga

Questo algoritmo ha identificato 2 comunità principali:

Comunità	Numero di nodi	Nodi
1	25	2, 3, 7, 8, 9, 13, 14, 15, 18, 19, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 0, 1
2	12	0, 1, 4, 5, 6, 10, 11, 12, 16, 17, 21

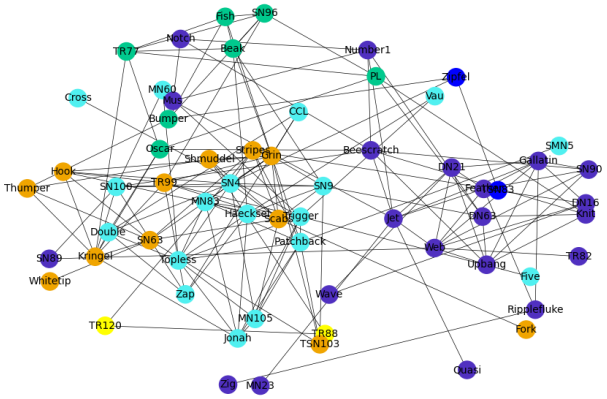


3.2.2 Dataset Bottlenose Dolphin Network

3.2.2.1 Label Propagation

L'algoritmo di Label Propagation ha individuato 6 comunità all'interno della rete:

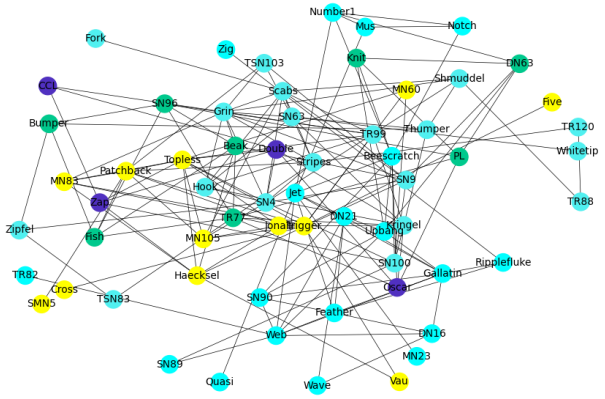
Comunità	Numero Nodi	Nodi
1	20	`Jet, MN23, Upbang, Mus, Gallatin, SN89, Zig, DN63, Feather, Notch, TR82, DN21, Web, Ripplefluke, Wave, Beescratch, Number1, Quasi, DN16, SN90, Knit`
2	17	`Zap, SN100, Patchback, SN9, MN60, SMN5, Five, CCL, SN4, Vau, Jonah, Cross, Haecksel, Trigger, Topless, MN105, Double, MN83`
3	12	`SN63, Scabs, Kringel, Whitetip, TSN103, Stripes, Thumper, Grin, Shmuddel, Hook, Fork, TR99`
4	7	`Bumper, Beak, Oscar, PL, TR77, SN96, Fish`
5	2	`TR120, TR88`
6	2	`TSN83, Zipfel`



3.2.2.2 Louvain

L'algoritmo Louvain ha individuato 5 comunità nella rete:

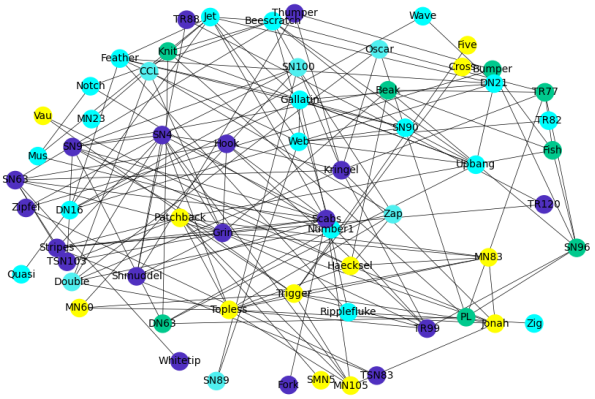
Comunità	Numero Nodi	Nodi
1	18	`Beescratch, DN16, DN21, Feather, Gallatin, Jet, MN23, Mus, Notch, Number1, Quasi, Ripplefluke, SN90, TR82, Upbang, Wave, Web, Zig`
2	17	`Fork, Grin, Hook, Kringel, Scabs, Shmuddel, SN4, SN63, SN9, Stripes, Thumper, TR120, TR88, TR99, TSN103, TSN83, Whitetip, Zipfel`
3	11	`Cross, Five, Haecksel, Jonah, MN105, MN60, MN83, Patchback, SMN5, Topless, Trigger, Vau`
4	8	`Beak, Bumper, DN63, Fish, Knit, PL, SN96, TR77`
5	6	`CCL, Double, Oscar, SN100, SN89, Zap`



3.2.2.3 Leiden

L'algoritmo Leiden ha identificato 5 comunità nella rete:

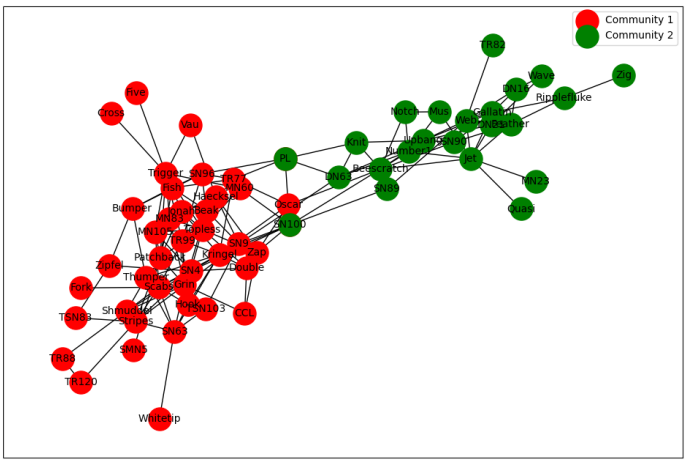
Comunità	Numero Nodi	Nodi
1	19	Beescratch, DN16, DN21, DN63, Feather, Gallatin, Jet, Knit, MN23, Mus, Notch, Number1, Quasi, Ripplefluke, SN90, TR82, Upbang, Wave, Web, Zig
2	16	Fork, Grin, Hook, Scabs, Shmuddel, SN4, SN63, SN9, Stripes, Thumper, TR120, TR88, TR99, TSN103, TSN83, Whitetip, Zipfel
3	12	Cross, Five, Haecksel, Jonah, MN105, MN60, MN83, Patchback, SMN5, Topless, Trigger, Vau
4	7	CCL, Double, Kringel, Oscar, SN100, SN89, Zap
5	6	Beak, Bumper, Fish, PL, SN96, TR77



3.2.2.4 Conga

L'algoritmo CONGA ha individuato 2 comunità principali:

Comunità	Numero Nodi	Nodi
1	38	Beak, Bumper, CCL, Cross, Double, Fish, Five, Fork, Grin, Haecksel, Hook, Jonah, Kringel, MN105, MN60, MN83, Oscar, Patchback, PL, Scabs, Shmuddel, SMN5, SN100, SN4, SN63, SN9, SN96, Stripes, Thumper, Topless, TR120, TR77, TR88, TR99, Trigger, TSN103, TSN83, Vau, Whitetip, Zap, Zipfel
2	21	Beescratch, DN16, DN21, DN63, Feather, Gallatin, Jet, Knit, MN23, Mus, Notch, Number1, Quasi, Ripplefluke, SN89, SN90, TR82, Upbang, Wave, Web, Zig



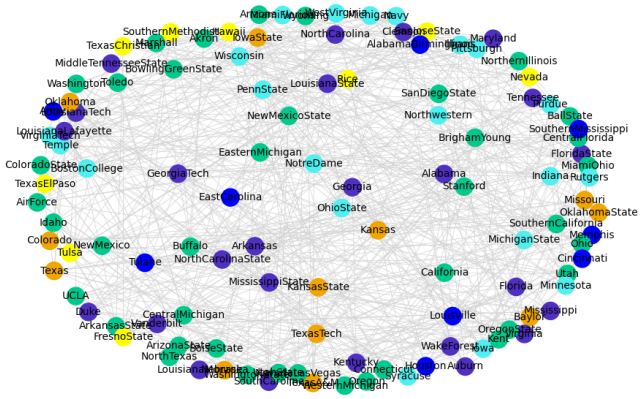
3.2.3 Dataset American College Football Club

3.2.3.1 Label Propagation

L'algoritmo di Label Propagation ha identificato 11 comunità all'interno della rete:

Comunità	Numero di Nodi	Nodi
1	17	LouisianaMonroe, MiddleTennesseeState, SouthCarolina, Mississippi, Auburn, LouisianaTech, Georgia, CentralFlorida, Vanderbilt, MississippiState, Kentucky, Alabama, Florida, LouisianaLafayette, Arkansas, Tennessee, LouisianaState
2	16	SouthernCalifornia, NorthTexas, NewMexicoState, OregonState, Idaho, UtahState, Washington, UCLA, ArkansasState, BoiseState, Stanford, Oregon, ArizonaState, California, WashingtonState, Arizona
3	12	Colorado, Texas_A&M, Texas, Baylor, OklahomaState, Oklahoma, IowaState, TexasTech, Kansas, Missouri, KansasState, Nebraska
4	11	OhioState, Northwestern, MichiganState, Wisconsin, PennState, Michigan, Purdue, Indiana, Iowa, Illinois, Minnesota
5	10	MiamiFlorida, Navy, NotreDame, WestVirginia, Syracuse, Pittsburgh, VirginiaTech, Temple, BostonCollege, Rutgers
6	9	Virginia, GeorgiaTech, NorthCarolina, Clemson, Maryland, NorthCarolinaState, WakeForest, Duke, FloridaState

Comunità	Numero di Nodi	Nodi
7	9	Louisville, SouthernMississippi, Cincinnati, Houston, AlabamaBirmingham, EastCarolina, Memphis, Army, Tulane
8	9	TexasChristian, Rice, SanJoseState, FresnoState, TexasElPaso, Tulsa, Hawaii, SouthernMethodist, Nevada
9	8	SanDiegoState, AirForce, ColoradoState, Wyoming, NevadaLasVegas, Utah, NewMexico, BrighamYoung
10	8	MiamiOhio, BowlingGreenState, Connecticut, Akron, Ohio, Marshall, Buffalo, Kent
11	6	CentralMichigan, BallState, NorthernIllinois, Toledo, EasternMichigan, WesternMichigan

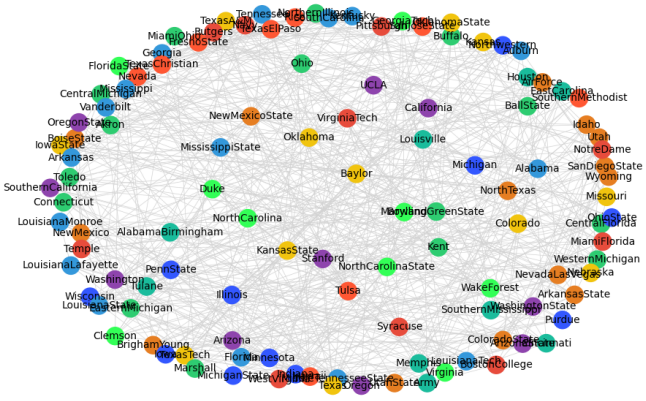


3.2.3.2 Louvain

L'algoritmo Louvain ha identificato 10 comunità all'interno della rete universitaria:

Comunità	Numero di Nodi	Nodi
1	17	Auburn, Alabama, Florida, CentralFlorida, Kentucky, LouisianaTech, LouisianaMonroe, Vanderbilt, MiddleTennesseeState, MississippiState, SouthCarolina, Tennessee, Mississippi, Georgia, LouisianaState, LouisianaLafayette, Arkansas
2	14	BrighamYoung, NewMexico, SanDiegoState, NorthTexas, Wyoming, Utah, ArkansasState, BoiseState, ColoradoState, Idaho, NewMexicoState, UtahState, AirForce, NevadaLasVegas
3	14	NorthernIllinois, WesternMichigan, Akron, BallState, BowlingGreenState, Buffalo, CentralMichigan, Connecticut, EasternMichigan, Kent, MiamiOhio, Ohio, Toledo, Marshall
4	12	KansasState, TexasTech, Baylor, Colorado, Kansas, IowaState, Nebraska, Texas_A&M, Oklahoma, Texas, Missouri, OklahomaState
5	11	Iowa, PennState, Northwestern, Wisconsin, Michigan, Purdue, OhioState, Minnesota, Illinois, MichiganState, Indiana

Comunità	Numero di Nodi	Nodi
6	10	SouthernCalifornia, ArizonaState, UCLA, Arizona, Washington, Oregon, Stanford, WashingtonState, OregonState, California
7	10	VirginiaTech, BostonCollege, WestVirginia, Syracuse, Pittsburgh, Temple, Navy, NotreDame, Rutgers, MiamiFlorida
8	9	FloridaState, NorthCarolinaState, Virginia, GeorgiaTech, Duke, NorthCarolina, Clemson, WakeForest, Maryland
9	9	EastCarolina, Houston, Louisville, Memphis, SouthernMississippi, Tulane, Army, Cincinnati, AlabamaBirmingham
10	9	FresnoState, Rice, SouthernMethodist, Nevada, SanJoseState, TexasElPaso, Tulsa, TexasChristian, Hawaii

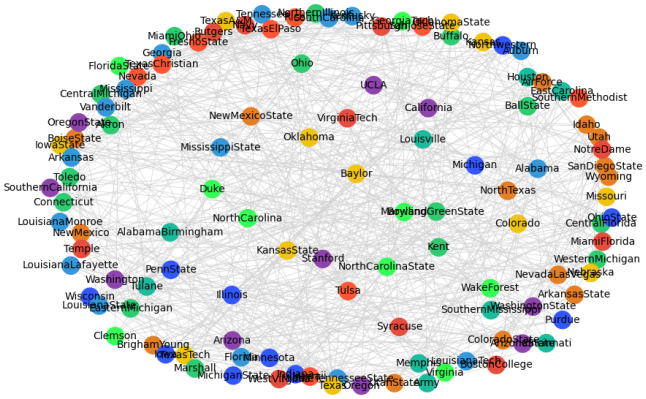


3.2.3.3 Leiden

L'algoritmo Leiden ha identificato 10 comunità all'interno della rete universitaria:

Comunità	Numero di Nodi	Nodi
1	16	Auburn, Alabama, Florida, Kentucky, LouisianaTech, LouisianaMonroe, Vanderbilt, MiddleTennesseeState, MississippiState, SouthCarolina, Tennessee, Mississippi, Georgia, LouisianaState, LouisianaLafayette, Arkansas
2	15	NorthernIllinois, WesternMichigan, Akron, BallState, BowlingGreenState, Buffalo, CentralFlorida, CentralMichigan, Connecticut, EasternMichigan, Kent, MiamiOhio, Ohio, Toledo, Marshall
3	14	BrighamYoung, NewMexico, SanDiegoState, NorthTexas, Wyoming, Utah, ArkansasState, BoiseState, ColoradoState, Idaho, NewMexicoState, UtahState, AirForce, NevadaLasVegas
4	12	KansasState, TexasTech, Baylor, Colorado, Kansas, IowaState, Nebraska, Texas_A&M, Oklahoma, Texas, Missouri, OklahomaState
5	11	Iowa, PennState, Northwestern, Wisconsin, Michigan, Purdue, OhioState, Minnesota, Illinois, MichiganState, Indiana

Comunità	Numero di Nodi	Nodi
6	10	VirginiaTech, BostonCollege, WestVirginia, Syracuse, Pittsburgh, Temple, Navy, NotreDame, Rutgers, MiamiFlorida
7	10	SouthernCalifornia, ArizonaState, UCLA, Arizona, Washington, Oregon, Stanford, WashingtonState, OregonState, California
8	9	FloridaState, NorthCarolinaState, Virginia, GeorgiaTech, Duke, NorthCarolina, Clemson, WakeForest, Maryland
9	9	FresnoState, Rice, SouthernMethodist, Nevada, SanJoseState, TexasElPaso, Tulsa, TexasChristian, Hawaii
10	9	EastCarolina, Houston, Louisville, Memphis, SouthernMississippi, Tulane, Army, Cincinnati, AlabamaBirmingham

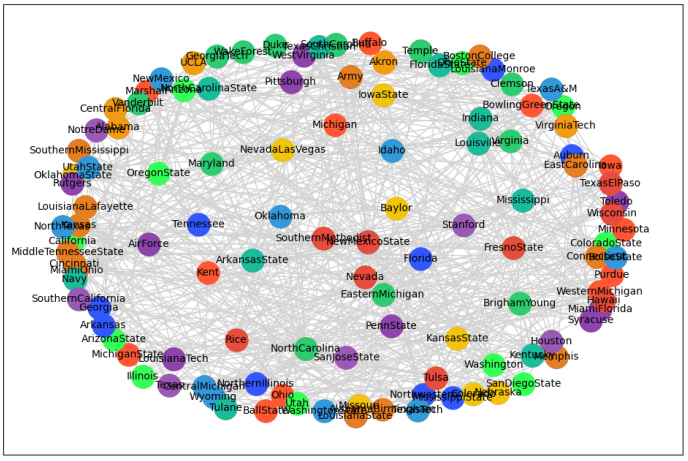


3.2.3.4 Conga

L'algoritmo di Conga ha identificato 12 comunità all'interno della rete universitaria:

Comunità	Numero di Nodi	Nodi
1	34	Iowa, NorthernIllinois, WesternMichigan, BallState, BowlingGreenState, Michigan, Buffalo, Syracuse, Purdue, Kent, Pittsburgh, Minnesota, MiamiOhio, Ohio, Toledo, NorthCarolina, Marshall, MichiganState, Missouri, NotreDame, CentralMichigan, PennState, Illinois, Wisconsin, OhioState, KansasState, IowaState, Nebraska, Cincinnati, Indiana, Northwestern, Connecticut, EasternMichigan, Akron
2	25	BrighamYoung, ArizonaState, SanDiegoState, Arizona, Utah, Colorado, ColoradoState, OhioState, Idaho, Washington, Illinois, Oregon, Stanford, OregonState, California, AirForce, NevadaLasVegas, UCLA, SouthernCalifornia, Nevada, FresnoState, Wyoming, NewMexico, UtahState, WashingtonState
3	22	Northwestern, Auburn, Florida, CentralFlorida, GeorgiaTech, LouisianaTech, LouisianaMonroe, MississippiState, NewMexicoState, SouthernMississippi, Tennessee, Georgia, Arkansas, Vanderbilt, NorthernIllinois, Alabama, SouthCarolina, Memphis, Mississippi, Kentucky, LouisianaState, MiddleTennesseeState, LouisianaLafayette
4	19	KansasState, TexasTech, Baylor, IowaState, Nebraska, Tulsa, NevadaLasVegas, OklahomaState, Hawaii, LouisianaTech, LouisianaLafayette, Colorado, Texas_A&M, Kansas, Baylor, Missouri, Oklahoma, NorthTexas, TexasTech, Texas

Comunità	Numero di Nodi	Nodi
5	15	PennState, VirginiaTech, WestVirginia, NotreDame, Army, AirForce, Rutgers, MiamiFlorida, Navy, EastCarolina, BostonCollege, Toledo, Temple, Syracuse, Pittsburgh, LouisianaTech
6	15	BostonCollege, Connecticut, EastCarolina, Kansas, Louisville, MiddleTennesseeState, Memphis, Cincinnati, LouisianaState, LouisianaLafayette, AlabamaBirmingham, SouthernMethodist, Houston, Army, SouthernMississippi, Tulane
7	14	NorthCarolinaState, Virginia, EasternMichigan, Duke, Vanderbilt, SouthCarolina, Temple, Clemson, WakeForest, Maryland, BrighamYoung, NorthCarolina, FloridaState, GeorgiaTech, Navy
8	14	NewMexico, NorthTexas, Wyoming, BoiseState, CentralMichigan, WashingtonState, Texas_A&M, TexasElPaso, Oklahoma, UtahState, ArkansasState, TexasTech, NewMexicoState, Idaho
9	11	Wisconsin, FresnoState, Rice, SouthernMethodist, Nevada, SanJoseState, TexasChristian, NewMexicoState, Tulsa, Hawaii, TexasElPaso
10	10	FloridaState, ArkansasState, Kentucky, Navy, Tulane, Mississippi, Indiana, TexasChristian, NorthCarolinaState, Louisville
11	7	SouthernCalifornia, Houston, Texas, Stanford, SanJoseState, NotreDame
12	5	Akron, Alabama, UCLA, CentralFlorida, VirginiaTech



3.2.4 AS Internet topology

Si riporta solo il numero delle comunità e non vengono mostrate le figure a causa dell'eccessivo numero di nodi.

Algoritmo	Comunità
Label Propagation	175
Louvain	27
Leiden	29

3.3 Risultati

L'algoritmo CONGA richiede di specificare il numero di comunità, a differenza di Label Propagation, Louvain e Leiden, che lo determinano automaticamente. CONGA è stato escluso per il dataset Polblogs a causa dei tempi troppo lunghi.

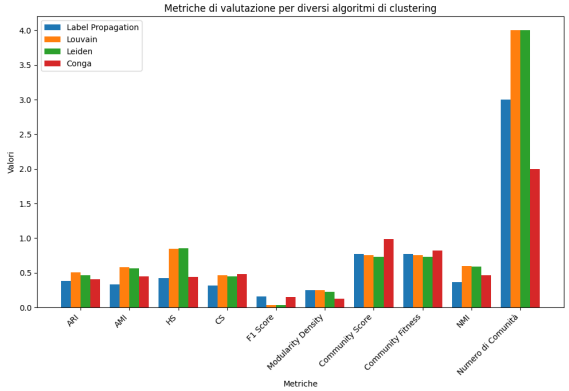


Figura 3.4: Zachary Karate Club

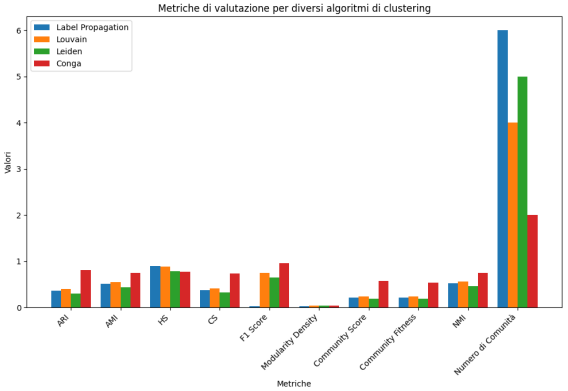


Figura 3.5: Bottlenose Dolphin Network

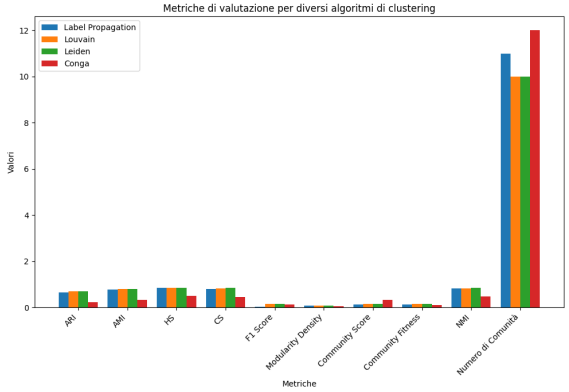


Figura 3.6: American College Football

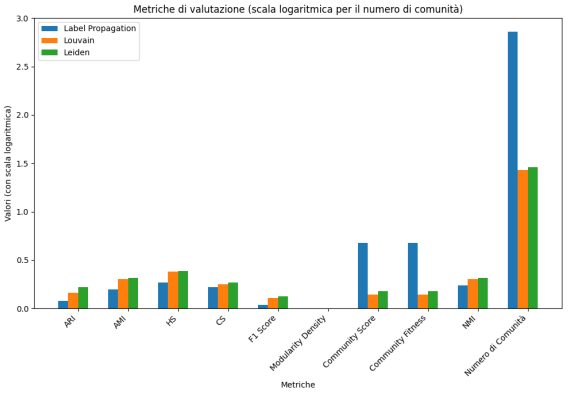


Figura 3.7: As Internet topology

3.3.1 Zachary Karate Club

Algoritmo	ARI	AMI	HS	CS	F1 Score	Modularity Density	Community Score	Community Fitness	NMI	Numero di Comunità
Label Propagation	0.383312	0.335286	0.425707	0.317306	0.154356	0.250794	0.769231	0.769231	0.363599	3.000000
Louvain	0.508864	0.578238	0.847140	0.464505	0.032258	0.246320	0.756410	0.756410	0.600011	4.000000
Leiden	0.464591	0.566666	0.853947	0.448190	0.034483	0.225379	0.730769	0.730769	0.587850	4.000000
Conga	0.402844	0.448029	0.439845	0.484315	0.150000	0.128106	0.987179	0.820513	0.461010	2.000000

ARI (Adjusted Rand Index): L'algoritmo Louvain ha ottenuto il punteggio più alto (0.508864), indicando una migliore corrispondenza tra le comunità identificate e quelle reali. Gli altri algoritmi, in particolare Label Propagation e Conga, mostrano punteggi più bassi, suggerendo una minore accuratezza nel raggruppamento.

AMI (Adjusted Mutual Information): Ancora una volta, Louvain risulta il migliore con 0.578238, seguito da Leiden (0.566666). Entrambi mostrano una forte capacità di identificare relazioni tra le comunità.

HS (Homogeneity Score): Qui, Leiden e Louvain hanno punteggi superiori a 0.84, il che indica che le comunità sono altamente omogenee. Gli altri algoritmi, in particolare Conga (0.439845), faticano a mantenere l'omogeneità.

CS (Community Score) e Community Fitness: Conga si distingue in queste metriche, mostrando valori più alti. Ciò implica che, sebbene possa non identificare comunità altrettanto bene quanto Louvain o Leiden, riesce a generare comunità più coese.

F1 Score : Questo punteggio è notevolmente basso per tutti gli algoritmi, con Louvain e Leiden che ottengono i punteggi peggiori. Ciò suggerisce che c'è un equilibrio insufficiente tra precisione e richiamo.

Modularity Density : Anche se Label Propagation e Louvain hanno punteggi simili, Conga ha un valore significativamente più basso, suggerendo che le comunità identificate non siano altrettanto dense.

Numero di Comunità : Infine, Louvain e Leiden identificano il numero più alto di comunità.

Louvain e Leiden emergono come algoritmi preferibili per il clustering rispetto agli altri, con punteggi più elevati in quasi tutte le metriche. Conga e Label Propagation si sono dimostrati migliori, in particolare nella coerenza e nell'accuratezza delle comunità identificate.

3.3.2 Bottlenose Dolphin Network

Algoritmo	ARI	AMI	HS	CS	F1 Score	Modularity Density	Community Score	Community Fitness	NMI	Numero di Comunità
Label Propagation	0.361361	0.505064	0.896877	0.373130	0.406452	0.025219	0.207547	0.207547	0.527008	6.000000
Louvain	0.403206	0.546690	0.877715	0.409577	0.000000	0.036728	0.232704	0.232704	0.558524	4.000000
Leiden	0.332950	0.496072	0.884606	0.360684	0.390377	0.035441	0.207547	0.207547	0.512432	5.000000
Conga	0.811720	0.749955	0.771559	0.735677	0.062531	0.038298	0.578616	0.540881	0.753191	2.000000

Adjusted Rand Index (ARI) : CONGA mostra la migliore capacità di identificare correttamente le comunità, mentre Leiden ha la performance più bassa, suggerendo una maggiore discrepanza rispetto alla verità di riferimento.

Adjusted Mutual Information (AMI) : CONGA eccelle nel riconoscere la struttura delle comunità, mentre Leiden mostra una performance meno efficace.

Homogeneity Score (HS) : Label Propagation è molto efficace nel mantenere l'omogeneità delle comunità, indicando che i nodi simili tendono ad essere assegnati alla stessa comunità.

Community Score (CS) F1 Score Modularity Density Community Fitness Normalized Mutual Information (NMI) : CONGA si dimostra capace nel riconoscere le relazioni tra le comunità, ottenendo i punteggi più alti tra gli altri algoritmi.

Numero di Comunità : al di fuori di Conga, gli altri algoritmi tendono a produrre più comunità, ma con punteggi di qualità più bassi.

Conga ottiene punteggi elevati in molteplici metriche, inclusi l'Adjusted Rand Index (ARI), l'Adjusted Mutual Information (AMI), il F1 Score, la Modularity Density e la Community Fitness.

Al contrario, l'algoritmo Leiden ha mostrato prestazioni più basse in diverse metriche, suggerendo che la sua identificazione delle comunità presenta una maggiore discrepanza rispetto alla verità di riferimento.

Label Propagation si è dimostrato efficace nel mantenere l'omogeneità delle comunità, evidenziando una buona capacità di raggruppare nodi simili.

3.3.3 American College Football Club

Algoritmo	ARI	AMI	HS	CS	F1 Score	Modularity Density	Community Score	Community Fitness	NMI	Numero di Comunità
Label Propagation	0.651490	0.774799	0.837962	0.802696	0.034504	0.072366	0.137031	0.137031	0.819950	11.000000
Louvain	0.700910	0.797416	0.839644	0.830509	0.139427	0.072002	0.141925	0.141925	0.835052	10.000000
Leiden	0.703916	0.804307	0.845491	0.835881	0.142029	0.071795	0.141925	0.141925	0.840659	10.000000
Conga	0.233261	0.324639	0.490390	0.450482	0.130385	0.049840	0.329527	0.097879	0.469590	12.000000

Adjusted Rand Index (ARI) : Louvain e Leiden mostrano un buon accordo con i dati reali, mentre CONGA presenta significative discrepanze.

Adjusted Mutual Information (AMI) : Anche qui, Leiden si distingue, suggerendo una buona capacità di identificare le comunità.

Homogeneity Score (HS) : Tutti gli algoritmi eccellono in questa misura, tranne CONGA, che mostra un'importante diminuzione dell'omogeneità.

Community Score (CS) : Louvain e Leiden riescono a formare comunità ben definite, mentre CONGA ha una qualità inferiore.

F1 Score : I punteggi sono relativamente bassi, suggerendo che gli algoritmi hanno difficoltà nel bilanciare le previsioni positive e negative, con Louvain e Leiden che si comportano leggermente meglio.

Modularity Density : Tutti gli algoritmi mostrano una densità relativamente alta, tranne CONGA.

Community Fitness : Louvain e Leiden dimostrano una forte stabilità rispetto agli altri algoritmi.

Normalized Mutual Information (NMI) : Louvain e Leiden dimostrano una forte capacità di riconoscere le relazioni tra le comunità.

Numero di Comunità : Louvain e Leiden mostrano un numero più bilanciato e qualità superiore. mentre Label Propagation ha un numero di comunità che si avvicina di più alla realtà.

Louvain e Leiden mostrano un buon accordo con i dati reali attraverso le metriche dell'Adjusted Rand Index (ARI) e dell'Adjusted Mutual Information (AMI). Questi risultati suggeriscono che entrambi gli algoritmi hanno una forte capacità di identificare le comunità e le loro relazioni. Tuttavia, Conga presenta significative discrepanze, evidenziando la sua incapacità di mantenere un buon livello di omogeneità all'interno delle comunità, come indicato dall'Homogeneity Score (HS). Per quanto riguarda la Modularity Density, Louvain e Leiden dimostrano una densità relativamente alta, mentre Conga risulta inferiore, indicando una qualità strutturale delle comunità meno soddisfacente. Label Propagation riesce a generare un numero di comunità più vicino alla realtà dei dati, ma con una qualità inferiore.

3.3.4 As Internet topology

Algoritmo	ARI	AMI	HS	CS	F1 Score	Modularity Density	Community Score	Community Fitness	NMI	Numero di Comunità
Label Propagation	0.78400	0.198283	0.264636	0.219469	0.035789	0.000579	0.677697	0.677697	0.239945	725.000000
Louvain	0.158764	0.301670	0.382108	0.251254	0.105855	0.000156	0.140121	0.140121	0.303163	27.000000
Leiden	0.221056	0.313268	0.383343	0.267221	0.125543	0.000392	0.179632	0.179632	0.314918	29.000000

ARI (Adjusted Rand Index) : L'algoritmo Leiden mostra le migliori prestazioni con un valore di 0.2211, seguito da Louvain con 0.1588 e infine Label Propagation con 0.0784. Questo indica che Leiden riesce a identificare comunità più coerenti con il riferimento atteso rispetto agli altri due algoritmi.

AMI (Adjusted Mutual Information): Anche qui Leiden ottiene i risultati migliori con un valore di 0.3133, seguito da Louvain con 0.3017 e Label Propagation con 0.1983. L'AMI suggerisce che Leiden è il più efficace nel rilevare le comunità rispetto ai dati di riferimento, ma la differenza con Louvain non è così ampia.

HS (Homogeneity Score): Leiden e Louvain ottengono valori simili il che indica che questi due algoritmi mantengono buoni livelli di omogeneità. Label Propagation è leggermente inferiore.

CS (Completeness Score): Anche in questo caso Leiden ottiene il punteggio più alto, seguito da Louvain e Label Propagation. Questo indica che Leiden è più completo nel raggruppare correttamente i membri della stessa classe.

F1 Score: Leiden raggiunge il miglior risultato con un valore di 0.1255, mentre Louvain è a 0.1059 e Label Propagation è significativamente più basso a 0.0358. Questo mostra che Leiden offre il miglior equilibrio tra precisione e richiamo rispetto agli altri due algoritmi.

Modularity Density: Label Propagation ottiene il valore più alto con 0.000579, seguita da Leiden con 0.000392 e Louvain con il valore più basso di 0.000156. Anche se i valori sono piccoli, questo suggerisce che Label Propagation è leggermente più efficace nel trovare comunità con una maggiore densità interna.

Community Score e Community Fitness: Label Propagation ha i valori più alti (0.6777), mentre Louvain e Leiden hanno valori molto inferiori (intorno a 0.1401 e 0.1796 rispettivamente). Ciò suggerisce che Label Propagation rileva comunità che sono più coese o ben definite.

NMI (Normalized Mutual Information): Leiden ottiene il valore più alto (0.3149), seguito da Louvain (0.3032) e Label Propagation (0.2399). Ancora una volta, Leiden si dimostra superiore nel riflettere accuratamente la struttura della rete.

Numero di Comunità: Il numero di comunità rilevate varia notevolmente tra gli algoritmi. Label Propagation rileva un numero estremamente alto di comunità (725), mentre Louvain e Leiden ne rilevano molte meno, rispettivamente 27 e 29. Questo può indicare che Label Propagation tende a frammentare la rete in troppe comunità, il che potrebbe spiegare i punteggi più bassi nelle altre metriche di somiglianza e omogeneità.

Leiden risulta complessivamente il miglior algoritmo in base alle metriche di accuratezza (ARI, AMI, HS, CS, F1 Score e NMI). Questo algoritmo rileva comunità ben formate e compatte, con buone prestazioni in termini di omogeneità e completezza. Louvain ha prestazioni simili a Leiden, ma leggermente inferiori su quasi tutte le metriche, sebbene la differenza sia relativamente piccola. Label Propagation mostra comportamenti molto diversi: rileva un numero eccessivamente elevato di comunità,

il che compromette le sue prestazioni nelle metriche di accuratezza. Tuttavia, ottiene risultati migliori nelle metriche di densità di modularità e punteggi di comunità, suggerendo che trova comunità dense ma troppo frammentate.

3.3.5 Considerazioni

Dall'analisi dei dataset, emerge che Label Propagation ha dimostrato di essere uno degli algoritmi più robusti tra quelli testati, con punteggi particolarmente elevati in termini di ARI e AMI nel dataset finale. Questo suggerisce una forte corrispondenza con le comunità reali, supportata anche da buoni valori di omogeneità e modularità. La sua capacità di identificare con precisione le comunità è evidente, soprattutto nelle reti complesse, dove riesce a mantenere una forte coesione interna.

L'algoritmo Louvain ha mostrato buone prestazioni in termini di rilevamento delle comunità, ma i suoi punteggi di ARI e AMI indicano una minore efficacia rispetto a Label Propagation. In generale, Louvain ha una buona densità di modularità e punteggi di omogeneità accettabili, il che implica che riesce a mantenere una certa coesione all'interno delle comunità, sebbene non raggiunga i livelli di coesione di Label Propagation. La sua capacità di rilevare strutture più grandi, come nel caso di dataset con 2 comunità, evidenzia una buona stabilità.

Simile a Louvain, Leiden ha fornito risultati competitivi con punteggi di ARI e AMI elevati. Con un numero di comunità trovato pari a 9 nel dataset di 12 comunità reali, Leiden ha dimostrato una tendenza a sottovalutare la complessità della struttura di rete. Questo suggerisce che, pur mantenendo una coesione buona all'interno delle comunità, può essere meno sensibile rispetto ad altri algoritmi nel catturare la vera variabilità delle comunità reali.

L'algoritmo Conga ha mostrato prestazioni più variabili rispetto agli altri metodi.

Pur avendo il punteggio più alto in Community Score nel dataset di 12 comunità. I punteggi di ARI e AMI indicano che il rilevamento delle comunità da parte di Conga è meno accurato rispetto agli altri algoritmi. Inoltre, la modularità e la densità delle comunità sono inferiori, suggerendo una bassa coesione interna.

Capitolo 4

Benchmark LFR e GN

In questo capitolo, verranno presentati i benchmark GN (Girvan-Newman) e LFR (Lancichinetti-Fortunato-Radicchi), utilizzati per testare l'efficacia dei quattro algoritmi di rilevamento delle comunità: Louvain, Leiden, Label Propagation e CONGA. Saranno discusse le caratteristiche dei benchmark e i risultati ottenuti dall'applicazione degli algoritmi su di essi.

4.0.1 Benchmark GN

[2] Il Benchmark di Girvan-Newman (GN) è un modello di rete utilizzato per testare algoritmi di rilevamento delle comunità. Questo benchmark è stato sviluppato da Michelle Girvan e Mark Newman nel contesto dello studio delle strutture delle reti e si basa su un modello semplice che consente di "piantare" comunità all'interno di una rete(il termine "comunità piantate" si riferisce a comunità che sono state artificialmente inserite o predefinite nel grafo come parte di una simulazione o esperimento).

Struttura della rete:

- Il benchmark GN è composto da 50 nodi.
- Poiché il modello non specifica una distribuzione di gradi, ogni nodo avrà una distribuzione di grado piuttosto uniforme. Tuttavia, essendo un modello semplificato, potrebbe non riflettere le caratteristiche delle reti reali.

- I nodi sono divisi in 2 comunità da 25 nodi.

Probabilità di connessione:

- **Connessioni interne**: Ogni nodo ha una probabilità p_{in} (in questo caso, **0.1**) di connettersi ad altri nodi nella stessa comunità.

- Le comunità sono definite come insiemi di nodi che hanno una densità di connessioni interna più alta rispetto alle connessioni tra le comunità. Quando $p_{in} > p_{out}$, i gruppi di nodi formano comunità; viceversa, se $p_{in} \leq p_{out}$, , la rete si comporta essenzialmente come un grafo casuale senza una struttura di comunità evidente.

Il benchmark GN è frequentemente utilizzato per valutare le prestazioni di vari algoritmi di rilevamento delle comunità. Gli algoritmi possono essere confrontati sulla base di quanto bene riescono a identificare le comunità "piantate" nel grafo. Nonostante la sua semplicità, il benchmark GN ha alcune limitazioni significative:

- Tutti i nodi hanno lo stesso grado atteso, il che non riflette la distribuzione di grado eterogenea comunemente osservata nelle reti reali.

- Tutte le comunità sono di dimensioni uguali, un altro aspetto che non è realistico nei contesti di rete complessi.

Le caratteristiche di omogeneità del benchmark GN lo rendono meno propenso nel testare algoritmi in scenari più complessi. Per affrontare queste limitazioni, sono stati sviluppati nuovi benchmark, come il Benchmark LFR (Lancichinetti-Fortunato-Radicchi), che introduce distribuzioni di grado e dimensioni delle comunità basate su leggi di potenza, rendendo il test degli algoritmi più impegnativo e realistico.

4.0.2 Benchmark LFR

[2] Nel benchmark LFR, i gradi dei nodi seguono una distribuzione di potenza, caratterizzata da un esponente τ_1. Questo significa che ci saranno nodi con pochi collegamenti e altri con un numero molto elevato di collegamenti, seguendo una legge di potenza. Anche le dimensioni delle comunità sono distribuite secondo una legge di potenza, con un esponente τ_2. Pertanto, ci si aspetta che alcune comunità siano molto grandi, mentre altre siano più piccole.

Numero di Nodi N rappresenta il numero totale di nodi nella rete (50).

Gradi Interni ed Esterni ogni nodo ha un grado interno k_{in} (collegamenti verso nodi all'interno della sua comunità) e un grado esterno k_{out} (collegamenti verso nodi in altre comunità). Questi gradi rimangono fissi durante la costruzione del grafo.

Parametri di Collegamento p_{in} **e** p_{out} questi parametri definiscono le probabilità di collegamento tra nodi all'interno della stessa comunità (interni) e tra nodi di comunità diverse (esterni). Nel benchmark LFR, i valori di p_{in} e p_{out} dipendono dal parametro di miscelazione μ.

Parametro di Miscelazione μ esprime il rapporto tra il grado esterno di un nodo rispetto alla sua comunità e il grado totale del nodo. Viene usato per definire la densità di collegamenti all'interno e all'esterno delle comunità.

Per garantire l'esistenza delle comunità, deve valere la condizione che

$$p_{in} > p_{out}$$

$$p_{\text{out}} = \frac{k^i_{\text{out}}}{k^c_{\text{out}}} \sim \frac{k^i_{\text{out}}}{(N - n_c)\langle k \rangle},$$

$$p_{\text{in}} = \frac{k_{\text{in}}^i}{k_{\text{in}}^c} \sim \frac{k_{\text{in}}^i}{n_c \langle k \rangle}.$$

dove:

- k_{in}^i: Il numero di collegamenti che il nodo i ha dentro la sua comunità.
- k_{out}^i: Il numero di collegamenti che il nodo i ha fuori dalla sua comunità.
- N: Il numero totale di nodi nel grafo.
- n_c: Il numero di nodi nella comunità c a cui appartiene il nodo i.
- k: Il grado medio, ovvero il numero medio di collegamenti per nodo

La condizione per l'esistenza delle comunità $p_{\text{in}} > p_{\text{out}}$ diventa:

$$\frac{k_{\text{in}}^i}{n_c \langle k \rangle} > \frac{k_{\text{out}}^i}{(N - n_c)\langle k \rangle},$$

da cui otteniamo:

$$k_{\text{in}}^i > \frac{n_c k_{\text{out}}^i}{N - n_c} \tag{4.0.1}$$

D'altra parte, per definizione abbiamo che:

$$\mu = \frac{k_{\text{out}}^i}{k_{\text{in}}^i + k_{\text{out}}^i} \tag{4.0.2}$$

pertanto confrontando le due equazioni 4.0.1 e 4.0.2 risulta:

$$\mu < \frac{N - n_c}{N}$$

L'equazione stabilisce una condizione necessaria affinché esistano comunità nel grafo. Essa indica che il grado interno di un nodo deve essere maggiore di una certa quantità legata al grado esterno e alla dimensione della comunità.

La condizione finale sulla miscelazione mostra che, per mantenere ben definite le comunità, μ deve essere inferiore a una certa soglia che dipende dalla dimensione della comunità più grande. Questo implica che, per comunità di dimensioni diverse, ci sono limiti diversi per la definizione delle comunità stesse.

Anche se teoricamente le comunità sono definite fino a un certo valore di μ, ci possono essere difficoltà nel rilevarle nella pratica. Questo può succedere per via delle fluttuazioni nella distribuzione dei collegamenti, rendendo i grafi simili a grafi casuali.

4.1 Risultati

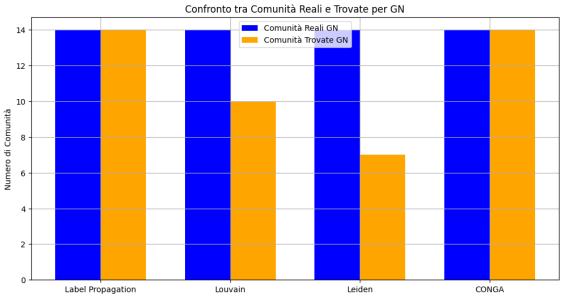


Figura 4.1: Risultati GN

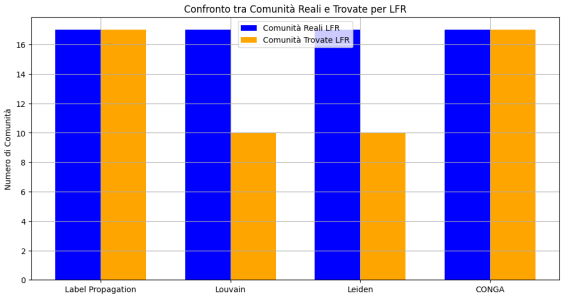


Figura 4.2: Risultati LFR

Comunità nel grafo GN: 14

Comunità nel grafo LFR: 17

Metodo	Modularità	Comunità
Label Propagation	0.5565	14
Louvain	0.6150	10
Leiden	0.1026	7
CONGA	0.5723	14

Tabella 4.1: Risultati dei vari metodi di rilevazione delle comunità nel grafo GN.

Metodo	Modularità	Comunità
Label Propagation	0.8012	17
Louvain	0.8730	10
Leiden	0.2192	10
CONGA	0.8029	17

Tabella 4.2: Risultati dei vari metodi di rilevazione delle comunità nel grafo LFR.

4.2 Analisi dei Risultati GN

4.2.1 Modularità

Louvain ha mostrato modularità elevata, indicando comunità ben definite e densamente collegate internamente. Leiden ha mostrato una modularità molto bassa, suggerendo difficoltà nel catturare la struttura della rete.

La variabilità nelle dimensioni delle comunità suggerisce che i metodi di rilevazione delle comunità interpretano diversamente la struttura sottostante del grafo GN. La

presenza di comunità più piccole e più grandi potrebbe influenzare l'accuratezza della rilevazione e la comprensione della rete.

Louvain risulta essere il metodo più efficace, seguito da Label Propagation, mentre Leiden mostra difficoltà nella rilevazione di comunità significative.

4.3 Analisi dei Risultati LFR

4.3.1 Modularità

Louvain e Label Propagation mostrano modularità elevata, indicando che le comunità trovate sono ben definite e densamente collegate internamente. La bassa modularità del metodo Leiden suggerisce che non riesce a catturare la struttura della rete in modo efficace.

La variabilità nelle dimensioni delle comunità identificate suggerisce che i metodi di rilevazione delle comunità possono interpretare diversamente la struttura del grafo LFR. La presenza di comunità più piccole e più grandi potrebbe influenzare l'accuratezza della rilevazione e la comprensione della rete.

Il metodo Louvain si dimostra il più efficace nella rilevazione delle comunità nel grafo LFR, dato il valore di modularità più alto. Label Propagation e CONGA forniscono risultati simili, ma con minori differenze nella dimensione delle comunità. Leiden sembra avere difficoltà a definire comunità significative.

4.4 Considerazioni

Il metodo Louvain risulta essere il più robusto in entrambi i grafi, suggerendo che è più adatto per l'analisi delle reti con comunità ben definite. Al contrario, Leiden mostra significative limitazioni, particolarmente nei grafi più complessi.

Conclusioni

L'analisi dei risultati ottenuti dai vari algoritmi di rilevamento delle comunità ha fornito una panoramica sulle loro performance in relazione a diversi dataset. In particolare, l'uso delle metriche di valutazione come l'Adjusted Rand Index (ARI), l'Adjusted Mutual Information (AMI), il Modularity, e altre misure, ha permesso di mettere in evidenza le differenze tra i metodi di rilevamento delle comunità.

I risultati dimostrano che Louvain si è distinto come il metodo più robusto, specialmente nel grafo LFR, dove ha raggiunto un punteggio di modularità elevati. Questo suggerisce che è in grado di identificare comunità dense e ben definite, evidenziando la sua capacità di lavorare efficacemente con reti complesse. Tuttavia, anche Conga ha mostrato performance competitive, riuscendo a rilevare comunità in scenari diversi, sebbene con risultati variabili. In particolare, il suo Community Score ha indicato che le comunità formate sono ben delineate, ma non sempre riescono a catturare la struttura sottostante dei dati.

D'altra parte, Label Propagation ha dimostrato una certa flessibilità nel rilevamento delle comunità, ma ha anche mostrato una tendenza a sovrastimare il numero di comunità nei grafi più semplici, come GN e LFR, dove sono state identificate ben 7 comunità, rispetto alle sole 2 comunità reali portando a una dispersione delle informazioni.

La performance di Leiden ha faticato a trovare comunità significative in tutti

i dataset. Le sue metriche di modularità erano inferiori rispetto agli altri algoritmi, suggerendo che potrebbe non essere sempre la scelta migliore per questo tipo di analisi.

Si nota che gli algoritmi tendono a performare meglio quando il numero di comunità reali è relativamente basso, come nei dataset con 2 comunità. In contesti più complessi Louvain e Leiden si sono dimostrati più efficaci.

Elenco delle figure

Elenco dei codici

Elenco degli algoritmi

Bibliografia

[1] Vincent D Blondel et al. *Fast unfolding of communities in large networks*. 2008. URL: https://arxiv.org/pdf/0803.0476.

[2] Ludovica Fortunato. *AALISI comparativa*. Accessed: 2024-10-07. 2023. URL: https://uniroma2.sharepoint.com/sites/Tesisti956/Documenti%20condivisi/General/Ludovica/FortunatoAalisicomparativa.pdf?CT=1728304125164&OR=ItemsView&wdOrigin=TEAMSFILE.FILEBROWSER.DOCUMENTLIBRARY.

[3] Steve Gregory. «An Algorithm to Find Overlapping Community Structure in Networks». In: *Lecture Notes in Computer Science*. Vol. 4702. Springer, 2007, pp. 91–102. DOI: 10.1007/978-3-540-74976-9_10. URL: https://typeset.io/pdf/an-algorithm-to-find-overlapping-community-structure-in-54rqmqd0gt.pdf.

[4] Jure Leskovec e Andrej Krevl. *SNAP Datasets: Stanford Large Network Dataset Collection*. https://snap.stanford.edu/data/email-Eu-core.html. Giu. 2014.

[5] NetworkX. *Football Graph Example*. Accessed: 2023-09-22. 2023. URL: https://networkx.org/documentation/stable/auto_examples/graph/plot_football.html.

[6] NetworkX. *Karate Club Graph Example*. Accessed: 2023-09-22. 2023. URL: https://networkx.org/documentation/stable/auto_examples/graph/plot_karate_club.html.

[7] F. Pedregosa et al. *Scikit-learn: Machine Learning in Python*. Online. Accessed: 2024-09-25. 2011. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.

[8] F. Pedregosa et al. *Scikit-learn: Machine Learning in Python*. Online. Accessed: 2024-09-25. 2011. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html.

[9] L. Procházka, J. Novák e V. Zeman. *Analysis of Cluster Structures in Complex Networks*. Neural Network World, vol. 26, no. 1, pp. 36-48. 2016. URL: http://www.nnw.cz/doi/2016/NNW.2016.26.036.pdf.

[10] Usha Nandini Raghavan, Réka Albert e Soundar Kumara. *Near linear time algorithm to detect community structures in large-scale networks*. 2007. DOI: 10.1103/PhysRevE.76.036106. URL: https://arxiv.org/pdf/0709.2938.

[11] Scikit-learn. *Adjusted Mutual Information Score*. 2023. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_mutual_info_score.html.

[12] Scikit-learn. *Adjusted Rand Index*. 2023. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html.

[13] Scikit-learn. *Completeness Score*. 2023. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.completeness_score.html.

[14] Scikit-learn. *Homogeneity Score*. 2023. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.homogeneity_score.html.

[15] Scikit-learn. *Normalized Mutual Information Score*. 2023. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.normalized_mutual_info_score.html.

[16] V. A. Traag, L. Waltman e N. J. van Eck. *From Louvain to Leiden: guaranteeing well-connected communities*. Mar. 2019. DOI: https://doi.org/10.1038/s41598-019-41695-z. URL: https://arxiv.org/pdf/1810.08473.

[17] Vladimir Livashkin. *Community Graphs*. Accessed: 2023-09-22. 2023. URL: https://github.com/vlivashkin/community-graphs.