Università degli Studi di Modena e Reggio Emilia

Dipartimento di Ingegneria “Enzo Ferrari”

Corso di Laurea Triennale in Ingegneria Informatica

Entity Resolution progressiva con graph embedding

Relatore:
Prof.ssa Sonia Bergamaschi

Candidato:
Michele Rinaldi

Correlatore: Luca Gagliardelli

ANNO ACCADEMICO 2020/2021

Sommario

INTRODUZIONE

L'obiettivo di questo elaborato è quello di migliorare le performance dei metodi che risolvono il problema della Entity Resolution progressiva sfruttando tecniche di *graph embedding*.

Nel primo capitolo verrà introdotto e definito il concetto di Entity Resolution, i concetti ad essa correlati ed infine verranno descritti in maniera generale gli step principali che compongono la risoluzione del problema.

Nel secondo capitolo verranno introdotti i concetti di Entity Resolution Progressiva e di Batch Entity Resolution, inoltre verrà fornita una breve spiegazione del metodo che allo stato dell'arte risolve nella maniera migliore il problema della Progressive Enity Resolution, ossia il PPS.

Nel terzo capitolo verrà introdotto in maniera generale il concetto di Graph embedding e le definizioni ad esso correlate.

Nel quarto capitolo, verranno spiegate in maniera specifica le tecniche che sono state implementate per provare a perseguire l'obiettivo di questo elaborato.

Nel quinto capitolo e nelle conclusioni verranno illustrati e commentati i risultati ottenuti.

1 ENTITY RESOLUTION

1.1 INTRODUZIONE ALL'ENTITY RESOLUTION

La nozione di ER emerse per la prima volta nel 1969 dagli studi di Fellegi e Sunter che lavoravano per la Boureau of Statistics in Canada.

I loro studi si focalizzarono sul trovare elementi duplicati tra due liste i cui elementi sono chiamati record, dove ognuna si assume non contenere duplicati. Chiamarono questo problema Record Linkage [1].

Fellegi e Sunter svilupparono un modello matematico per fornire un theoretical framework per una soluzione computer-oriented per riconoscere quei record all'interno di due file che rappresentano persone, oggetti o eventi identici. Nel corso di quegli anni si espanse l'interesse verso questo problema per diversi motivi che riassunsero in tre punti principali:

- Creazione di file di dimensioni sempre maggiori, i quali spesso richiedevano manutenzione per un lungo periodo di tempo e che spesso contenevano importanti informazioni statistiche il cui valore sarebbe potuto aumentare collegando e raggruppando record individuali provenienti da diversi file.
- Consapevolezza crescente della potenziale importanza del record-linkage in campi applicativi come la medicina e la ricerca genetica.
- Progressi nelle apparecchiature e nelle tecniche di elaborazione elettronica dei dati, che rendevano possibile, dal punto di vista tecnico ed economico, il gestire una grande quantità di operazioni dovute al comparare tra di loro i record provenienti da file di medie dimensioni.

Hernandez e Stolfo [2] studiarono nel contesto dei database relazionali il modo migliore e più efficiente per trovare e fondere i record che si riferiscono alle stesse entità all'interno della stessa sorgente di dati. Chiamarono questo problema purge/merge.

Il concetto di Entity Resolution iniziò ad apparire intorno al 2004 in articoli e presentazioni di ricercatori della Stanford University guidati dal professor Hector Molina nel 2006 [3].

Come si può notare gli obiettivi della ER di allora sono molto simili a quelli di adesso.

1.2 IMPORTANZA DELL'ENTITY RESOLUTION

L' Entity Resolution, in italiano risoluzione delle entità, è quel task che ha come obiettivo l'identificazione di elementi che rappresentano la stessa entità del mondo reale.

Il termine entità è inteso come oggetto del mondo reale. Per esempio, una persona, una cosa, un posto ecc…

Il termine Resolution è usato fondamentalemente perché l'Entity resolution è un processo di decisione (risoluzione), cerca di rispondere alla domanda: "Le referenze si riferiscono alla stessa entità o ad entità differenti? " [4].

ER è anche definito come "il processo di identificare e fondere i record che sono giudicati riferirsi entità del mondo reale".

L'ER è importante ed è applicabile a svariati campi applicativi, come la ricerca medica, l'analisi di dati di mercati azionari, analisi di dati per la sicurezza, analisi di dati per censimenti, insomma tutti i campi che coinvolgono una grande quantità di informazioni e necessitano di renderle pulite e di mantenerle organizzate.

1.3 CONCETTI PRELIMINARI

Per introdurre correttamente il concetto di entity resolution è importante introdurre il concetto di entity profile o in italiano profilo di entità o più semplicemente profilo, che corrisponde a una rappresentazione sotto forma di dati, per esempio parole, di una entità o oggetto del mondo reale. Ogni profilo rappresenta una e una sola entità del mondo reale, al contrario quest'ultima può avere diverse rappresentazioni sotto forma di profili differenti.

ID	Nome	Cognome	Squadra
p_i	Luigi	Datome	Olimpia Milano

Tabella 1

ID	Name	Team	Nationality
k_j	Luigi Datome	Milano	Italy

Tabella 2

Come si può notare nelle tabelle 1 e 2, i due profili p_i e k_j sono differenti, ma si riferiscono alla stessa entità del mondo reale, in questo caso un giocatore di pallacanestro.

Un profilo p_{id} è descritto da un insieme di attributi aventi un certo valore ed un identificatore univoco $id \in I$, dove I è l'insieme di tutti i valori di identificatori.

Un attributo a è una coppia $< n, v >$ dove $n \in N$, con N che indica il set infinito di tutti i possibili nomi di attributi e $v \in V$, con V che indica l'insieme di tutti i valori che sono associabili al nome di attributo n. L'insieme di attributi associati ad un identificatore id , ovvero l'insieme degli attributi che formano il profilo id, si denota con A_{id}. Un set di profili è chiamato collezione di entità o dataset P.

Si può ora definire formalmente il concetto di *Entity Resolution*:

Due profili p_i ed p_j corrispondono $p_i \equiv p_j$ se si riferiscono allo stesso oggetto reale, i profili che corrispondono si chiamano duplicati il compito dell'*Entity Resolution* è di trovare tutti i duplicati all'interno di uno o più dataset.

Si distinguono due tipologie di *ER*:

- *Deduplicazione*: dato un dataset P trovare tutte le coppie (p_i , p_j) di profili duplicati, tali che $(p_i , p_j) \in$ P e $p_i \equiv p_j$.
- *Record Linkage*: dati due dataset P_1 e P_2 liberi da duplicati al loro interno, si trovano tutte le coppie (p_i , p_j) di profili duplicati, tali che $p_i \equiv p_j$ e $p_i \in P_1$ e $p_j \in P_2$.

I dataset utilizzati per la deduplicazione, ovvero quelli che possono contenere duplicati si definiscono *dirty*. I dataset utilizzati per il *Record Linkage*, ovvero, quelli che al loro interno non contengono duplicati si definiscono clean-clean.

Perseguire l'obiettivo della *ER* è complesso a causa di diversi fattori.

In primo luogo, comparare ogni coppia di profili possibile è impensabile perché è un numero di comparazioni che cresce esponenzialmente all'aumentare dei profili e in grandi dataset diverrebbe un numero di comparazioni non scalabile. Quindi, è necessario utilizzare un metodo per ridurre le coppie da comparare. Si supponga di confrontare due dataset ciascuno avente 1 milione di elementi, quindi, ci saranno 1×10^{12} comparazioni da eseguire, si supponga inoltre che ogni comparazione viene eseguita mediamente in un tempo pari a 0.5 ms, ne deriva che i duplicati sono stati rilevati tutti, ma in un tempo pari a circa 50 milioni di secondi, che equivale a circa 578 giorni, che equivalgono a circa 1.6 anni.

In secondo luogo, è difficile confrontare insiemi di dati eterogenei che possono differire leggermente. Inoltre, è difficile trovare un modo universale per stabilire la similarità e comparare oggetti provenienti dai più svariati campi applicativi.

Per definire che una coppia di profili (e1, e2) è un duplicato bisogna accordare un metodo per calcolare la loro similarità. In modo più specifico, data una soglia di similarità θ e una funzione di similarità sim(.) si classificano due profili(p_1, p_2) come possibile coppia di duplicati nella modalità seguente:

$$\text{classify}(p_1, p_2) = \begin{cases} p_1 \not\equiv p_2 , & sim(p_1, p_2) \leq \theta \\ p_1 \equiv p_2 , & altrimenti \end{cases}$$

Dove il simbolo $\not\equiv$ indica che due profili non sono duplicati.
Intuitivamente più la similarità tra due profili è grande maggiore è la probabilità che questi profili siano duplicati. In modo analogo si può ragionare attraverso la distanza tra due profili:

Data una soglia di differenza θ e una funzione dist(.) che calcola la distanza tra una coppia di profili ovvero quanto differiscano tra loro, si classificano due profili (p_1, p_2) come possibile coppia di duplicati nella modalità seguente:

$$\text{classify}(p_1, p_2) = \begin{cases} p_1 \not\equiv p_2 , & dist(p_1, p_2) \geq \theta \\ p_1 \equiv p_2 , & altrimenti \end{cases}$$

Intuitivamente, scegliendo un valore di θ più vicino possibile allo 0, più la differenza tra due profili è prossima allo 0 maggiore è la probabilità che essi siano duplicati.

In entrambi casi θ deve essere scelto opportunamente. Per la similarità a un θ minore corrisponderanno un numero maggiore di coppie di profili ritenute duplicati; per la distanza più θ è grande e più saranno le coppie di profili ritenuti duplicati dalla funzione.

In caso di misurazione normalizzata della similarità, in cui le funzioni ritornano un valore compreso tra 0 e 1 si ha che:

$$dist(p_1, p_2) = 1 - sim(p_1, p_2)$$

Queste funzioni vengono definite funzioni di *matching*, perché calcolano se due profili corrispondono (*match*).

Per ridurre il numero di coppie di profili da comparare, tenendo solo quelle che hanno una maggiore probabilità di essere duplicati e cercando di scartare il meno possibile di duplicati reali, si implementano tecniche di *blocking* [5].

L'idea è di raggruppare in blocchi i profili, in questo modo, più saranno i blocchi in cui appariranno contemporaneamente due profili, più questi saranno simili e di conseguenza avranno una probabilità maggiore di essere duplicati. Si assume che due profili simili condividano almeno un blocco in comune, quindi, per iniziare si scartano tutte le coppie di profili che non condividono blocchi. Il criterio con cui i blocchi sono generati è detto "*schema di blocking"*.

Schema di blocking: Data una collezione di entità P, lo *schema di blocking* è una funzione $f_b: P \rightarrow B$ che mappa una collezione di profili in una collezione di blocchi B. questa funzione è composta da due funzioni, una funzione di trasformazione $f_T: P \rightarrow C(K)$ che mappa ogni profilo $p \in \mathrm{P}$ in una collezione di *chiavi di blocking* C(K); e una funzione di assegnamento $f_A: K \rightarrow B$ che associa ogni *chiave di blocking* ad un blocco $b \in \mathrm{B}$. [5]

Questa definizione applica la *deduplication* ma può essere facilmente estesa al *record linkage*.

Si denota con D(B) il set di tutte le coppie di duplicati rimaste dopo aver applicato il blocking e si denota con D(E) il set di tutti i duplicati, si ha che $D(B) \subseteq D(E)$.

Si definisce la cardinalità di B e si denota con ||B|| il numero di possibili comparazioni che sono presenti in B. $\big||\mathrm{B}|\big| = \sum_{b \in B} \big||b|\big|$, dove ||b|| indica il numero di possibili comparazioni contenute in un singolo blocco b.

Uno schema di blocking dovrebbe trovare un equilibrio tra minimizzare il numero di comparazioni e minimizzare il numero di duplicati persi. Questi due obiettivi sono espressi attraverso le seguenti misure:

- *Pair completeness* PC, corrisponde alla *recall*, valuta la porzione dei duplicati che condividono almeno un blocco e che quindi possono essere rilevati. È dato dal rapporto tra il numero di duplicati rilevabili dopo il blocking e il numero di tutti i duplicati presenti nella/e collezione/i di partenza. $PC(B) = D(B)/D(P) \in [0,1]$ o $PC(B) = D(B)/D(P_1 \times P_2) \in [0,1]$.
- *Pair quality* PQ, corrisponde alla *precision*, valuta la porzione delle comparazioni in B che corrispondono a duplicati reali. $PQ(B) = D(B)/\big||B|\big| \in [0,1]$

- *Reduction Ratio* RR, misura la riduzione in termini di comparazioni di coppie di profili in B rispetto all'approccio forza-bruta di comparare tutte le possibili coppie di profili. $RR(B) = 1 - \big||B|\big|/\big||P|\big| \in [0,1]$.

PC è collegato all'efficacia, ovvero, all'obiettivo di massimizzare il numero di duplicati rilevati; mentre, PQ e RR sono collegati all'efficienza, ovvero, all'obiettivo di minimizzare il numero di comparazioni per rilevare i duplicati. Queste definizioni si riferiscono al deduplication, ma possono essere facilmente estese al record linkage.

Fatte queste precisazioni si può dare una definizione formale di blocking:

*D*ata una collezione di entità, *blocking* raggruppa profili simili in una collezione di blocchi B in modo tale che allo stesso tempo siano massimizzate PC(B), PQ(B) e RR(B).

Il blocking può essere applicato più volte allo stesso o agli stessi dataset avvalendosi di schemi di blocking differenti, in questo modo, si enfatizzano maggiormente e si rivelano con più facilità le similarità e le diversità tra profili, dal momento che se tramite più schemi di blocking due profili risultano simili, questi hanno una maggiore probabilità di essere duplicati effettivi. Parimenti, possono emergere delle similarità che una funzione di blocking non riesce a rilevare.

Per questi motivi la scelta delle tipologie di blocking e schemi di blocking svolge un ruolo cruciale nell'ER. Tale scelta varia a seconda della tipologia dello schema che seguono i dataset e se si conosce tale schema, se i dataset di ingresso sono dirty o clean-clean, ecc…

Meta-Blocking.

Come detto precedentemente la qualità di una collezione di blocchi B si misura in termini della sua cardinalità $||B||$, l'obiettivo del *meta-blocking* è quello di ristrutturare B in modo tale che ne venga migliorata la qualità.

Definizione di Meta-blocking [6]. Data una collezione di blocchi B il compito del Meta-blocking è quello di ristrutturarla in un'altra collezione B' che raggiunga una maggiore efficienza e mantenga simile l'efficacia, quindi:

$$PQ(B') \gg \mathrm{PQ(B)}, \qquad \mathrm{RR(B')} \gg \mathrm{RR(B)}, \qquad \mathrm{PC(B')} \approx \mathrm{PC(B)}$$

Per perseguire l'obiettivo del Meta-blocking è necessario introdurre il concetto di blocking graph.

Definizione di blocking-graph. Data una collezione di blocchi B, il corrispondente blocking graph $G_B = \{V_B, E_B, WS\}$ è un grafo pesato in cui ogni profilo $\mathrm{p} \in P\big(p \in P_i \cup P_j \ in\ caso\ di\ record\ Linkage\big)$ che appartiene ad almeno un blocco $b \in B$ è rappresentato da un nodo $v \in V_B$, E_B corrisponde al set degli archi indiretti dove ogni arco $e_{ij} \in E_B$ collega due nodi (v_i, v_j) associati a due profili (p_i, p_j), WS rappresenta uno schema di pesatura che associa ad ogni arco e_{ij} un peso $w_{ij} \in \mathrm{R}$.

Un primo approccio per pesare gli archi può essere ottenuto contando il numero n di blocchi in cui due profili (p_i, p_j) appaiono contemporaneamente, in questo caso si avrà che $w_{ij} = n$.

Schemi di pesatura più avanzati tengono conto anche di fattori come l'importanza dei blocchi, quindi per esempio, ad ogni blocco viene assegnato un certo coefficiente e il peso dei blocchi viene

moltiplicato per tale valore. Il peso assegnato agli archi può essere anche normalizzato, ovvero, ogni arco e_{ij} avrà associato un peso $w_{ij} \in [0,1]$.

Un incremento addizionale dell'efficienza può essere raggiunto attraverso il *pruning*, ovvero la potatura del blocking graph: gli archi che collegano due profili che non sono duplicati possono essere gradualmente rimossi dal blocking graph scartando comparazioni non necessarie senza influenzare la PC.

Gli archi da potare sono scelti tramite algoritmi di *pruning*, come primo facile esempio, gli archi e_{ij} aventi un peso w_{ij} inferiore ad una certa soglia $\Theta \in \mathrm{R}$ o $\Theta \in [0,1]$ per la soglia normalizzata sono scartati. Più avanti nel corso di questo elaborato verrà descritta brevemente una tecnica di *pruning* chiamata *Wheighted node pruning* WNP.

Al termine di questo processo le uniche comparazioni che si devono eseguire sono quelle tra profili (p_i, p_j) che sono ancora collegati tramite un arco e_{ij} o al limite, se non collegati tramite un arco, devono condividere un determinato numero di profili confinanti condivisi, o per ultimo si deve dimostrare che facciano parte di un *clique* cioè, di un agglomerato di nodi simili.

Le tecniche di meta-blocking sono molteplici ed eseguibili in serie, lo scopo di questo elaborato è di utilizzare la tecnica del graph embedding come tecnica di meta-blocking per migliorare la struttura del blocking graph e rilevare più facilmente le coppie di profili duplicati.

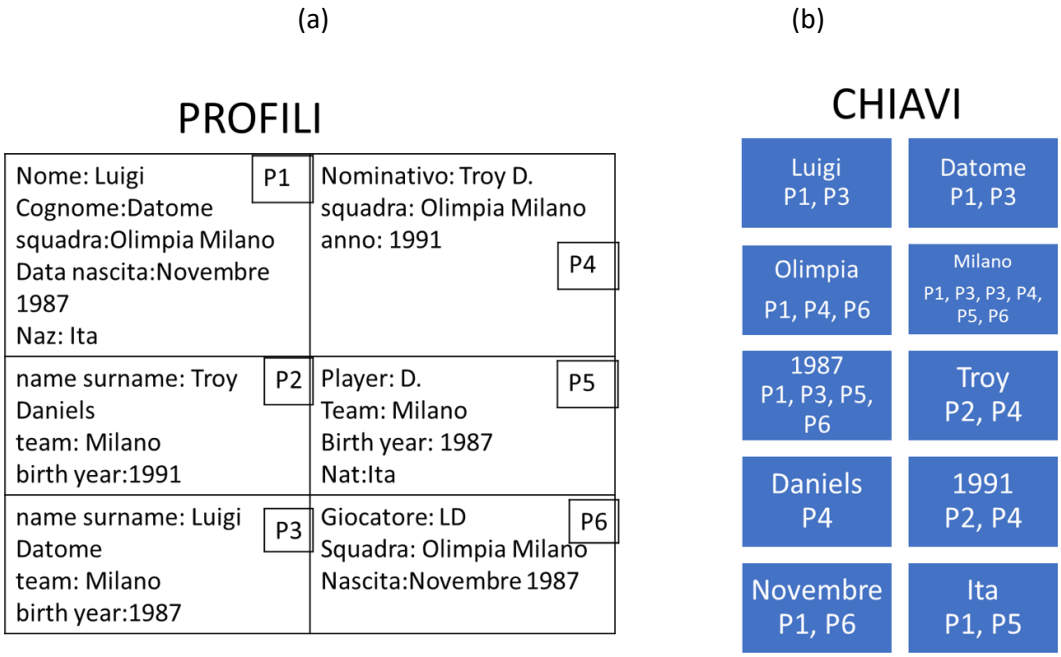


Figura 1 Blocking Generico

Figura 1 mostra a sinistra (a) l'insieme dei profili su cui è applicato il blocking, mentre a destra (b) mostra l'insieme di blocchi ricavato.

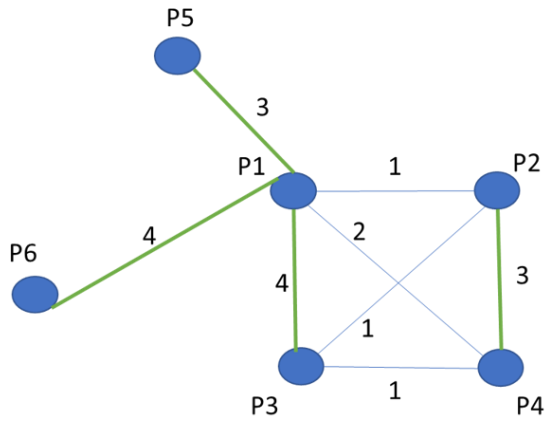


Figura 2 blocking graph esempio

Figura 2 mostra il blocking graph, ricavato dai blocchi di figura 1, i collegamenti più accentuati sono le coppie di nodi che rappresentano profili inseriti contemporaneamente in un numero relativamente alto di blocchi.

2 PROGRESSIVE ENTITY RESOLUTION

ER può essere distinto in due categorie principali:

- Off-line ER o BATCH ER che ha l'obiettivo di fornire una soluzione completa dopo che tutti i processi di blocking sono terminati.
- On-line o progressive ER che ha l'obiettivo di fornire la parziale soluzione migliore possibile quando i tempi e/o le risorse computazionali sono limitate.

La progressive ER usa un approccio *pay as you go*, nel senso che assegnato un budget BG, come per esempio il tempo o il numero di comparazioni, l'obiettivo della progressive ER è di perseguire la migliore soluzione parziale utilizzando al più BG unità di costo.

Progressive ER sta diventando molto importante dal momento che il numero dei dati e informazioni che deve essere gestito si sta moltiplicando.

È importante anche per contesti in cui si hanno a disposizione risorse computazionali o periodi di tempo limitati, come l'analisi dei mercati finanziari o la sicurezza.

Nel BATCH ER la cosa importante è che, come richiede la entity resolution, siano massimizzate la PC, RR e PQ, ma solo all'interno di un set di possibili duplicati e non importa l'ordine con cui si svolgono le comparazioni.

Sviluppare un metodo ottimo per la PER è inammissibile in pratica, dal momento che richiederebbe un oracolo, ovvero una funzione ideale che riesce a scegliere in avanzamento il set composto dalle migliori coppie da comparare, che una volta comparate si avrà il massimo incremento possibile parziale della qualità del risultato. L'obiettivo della ER si concentra nel trovare una "buona" strategia che utilizzi nel modo migliore possibile il budget fornito, salvando i costi su due livelli. Per prima cosa, questa strategia dovrebbe concentrarsi su quelle parti dei dataset che influenzano maggiormente la qualità del risultato. Questo significa trovare il maggior numero di duplicati possibili nel limite di budget fornito, poiché per definizione risolvendo i duplicati si avrà un incremento della qualità del metodo. Tale scopo si raggiunge comparando progressivamente le coppie di profili che hanno una maggiore probabilità di essere duplicati, questa probabilità può essere definita tramite molteplici metodi. In secondo luogo, se il costo di comparazione di una coppia è diverso dal costo di comparazione di un'altra coppia, a parità di probabilità di essere duplicati, si preferisce comparare prima i profili appartenenti alla coppia che ha costo di comparazione minore.

Data una collezione di blocchi B e detto T_0 il tempo impiegato per svolgere il BATCH ER su B, la Progressive ER è formalmente definita come segue [7]:

- Migliora prima la qualità, se sia il BATCH ER che Il Progressive ER sono applicati a B, prendendo in considerazione un tempo $t_0 \ll T_0$, allora il numero di duplicati rilevati dalla Progressive ER dall'inizio fino a t_0 deve essere un numero significativamente di quelli rilevati dalla BATCH ER nello stesso periodo di tempo.
- Stessa qualità finale, la recall prodotta al tempo T_0 deve essere uguale, o al più variata in modo non significativo, sia per il BATCH ER che per il Progressive ER, Questo per verificare la correttezza dei metodi, essendo che i duplicati rilevati dopo la fase di blocking sono gli stessi.

Le due fasi principali della Progressive entity resolution sono:

1. La fase iniziale, nella quale si preparano i blocchi attraverso tecniche di blocking, si crea il blocking graph e si processa con tecniche di metablocking prima che siano effettuati dei paragoni.
2. Fase di emissione, nella quale ritorna la coppia di profili che hanno maggiore probabilità di essere duplicati presi da una lista di coppie candidate ordinate in ordine decrescente di probabilità, quindi vengono restituite ad ogni iterazione le coppie rimanenti che hanno maggiore probabilità di matchare. Questa fase può essere ripetuta fino a quando è richiesta una nuova comparazione da processare.

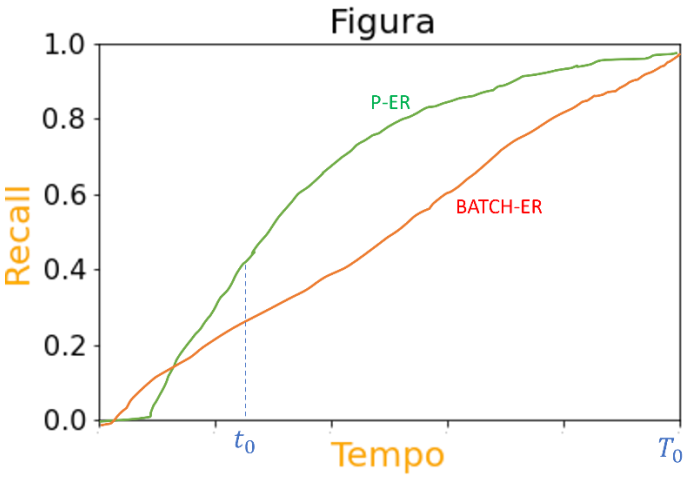
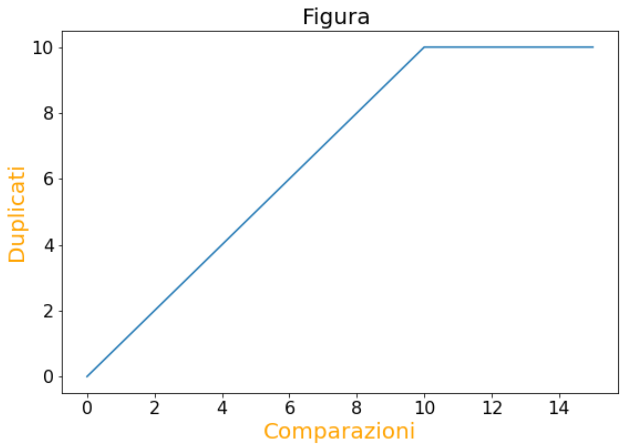


Figura 3 BATCH-ER vs Progressive ER

Figura 3 mostra la differenza tra l'andamento tra la BATCH ER e le progressive ER. Sull'asse delle ordinate è rappresentata la recall, numero di duplicati rilevati rapportato al numero di duplicati totali, mentre sull'asse delle ascisse è rappresentato il tempo, ma può valere anche per altre misure come, per esempio, il numero di comparazioni eseguite. Come si può notare la Progressive ER necessita di un periodo di set-up t_s per riordinare il blocking graph in modo da rendere più facile il rilevamento progressivo e veloce dei duplicati. Poiché nella BATCH ER non sono state eseguite operazioni di set-up, i duplicati sono disposti casualmente all'interno del set delle comparazioni da eseguire, quindi nel caso medio, il numero di coppie di profili duplicati rilevati cresce linearmente all'aumentare delle comparazioni eseguite. Fatta eccezione per il t_s iniziale, prendendo un tempo $t_0 \ll T_0$ il numero di duplicati rilevati dalla progressiva ER è maggiore del numero trovato dalla BATCH-ER, di norma $PER(t_0) > BATCHER(t_0)$.

Figura 4 funzione oracolo



In Figura 4 viene mostrato l'andamento della funzione oracolo, o funzione ideale. Sull'asse delle ascisse è rappresentato il numero di comparazioni eseguite, mentre sull'asse delle ordinate è rappresentato il numero di duplicati rilevati. A scopo esemplificativo, si assume che il dataset in questione abbia 10 duplicati, la funzione oracolo dopo 10 comparazioni ha trovato tutti e 10 duplicati, procedere con le comparazioni sarebbe inutile perché i duplicati sono già stati tutti rilevati quindi, questo numero non può salire.

PPS [7] (progressive record scheduling) è il metodo che allo stato dell'arte persegue nel modo migliore l'obiettivo di risolvere il problema della Entity Resolution. PPS è basato sul concetto di duplication likelihood, ovvero, la probabilità di un profilo p_i di avere duplicati. Dapprima calcola secondo alcuni metodi, per ogni profilo la sua duplication likelihood, inserendoli in seguito in un lista ordinata in ordine decrescente chiamata sorted profile list. Poi per ogni profilo della lista valuta la migliore comparazione possible.

3 GRAPH EMBEDDING

L'obiettivo della tesi è di utilizzare tenciche di graph embeddings durante il metablocking per raffinare e riordinare in modo più preciso il blocking graph, quindi di cercare un metodo che, applicato al problema della progressive entity resolution, raggiunga prestazioni migliori di quelle che raggiungono le tecniche presenti nello stato dell'arte.

I metodi di graph embedding hanno lo scopo di "imparare" delle rappresentazioni latenti a bassa dimensione dei nodi in un grafo. Queste rappresentazioni possono essere utilizzate per i più svariati obiettivi e metodi, come la classificazione, il raggruppamento di entità simili e la visualizzazione.

Dai social network o dal WWW, i grafi forniscono una maniera generale per rappresentare un set di informazioni eterogenee di entità del mondo reale.

Un embedding è una rappresentazione tramite vettore dei nodi di un grafo che evidenzia meglio le similarità tra di essi, riuscendo anche a rilevare le similarità latenti, inoltre la dimensione della matrice degli embeddings è minore della dimensione del grafo.

Considerando un grafo indiretto pesato G(V, E, W) dove V indica l'insieme dei nodi, E indica l'insieme degli archi che connettono due nodi e W indica la pesatura degli archi.

L'idea centrale è quella di trovare una funzione di mappatura che converte ogni nodo del blocking graph in una rappresentazione vettoriale a più bassa dimensione. Gli embedding devono avere le seguenti caratteristiche [8]:

- Adattabilità: I grafi e le reti reali sono costantemente in evoluzione, estensioni e modifiche del grafo non dovrebbero ripetere il processo di apprendimento del grafo laddove non è necessario.
- Scalabilità: I grafi sono spesso di elevate dimensioni, comprendono un elevato numero di archi tra i nodi, questi fattori rendono difficile il processo di capire correttamente e velocemente i collegamenti più importanti tra i nodi, gli embedding devono essere in grado di processare grafi di larga scala in un periodo di tempo relativamente breve, fornendo al contempo una rappresentazione più organizzata del grafo.
- Bassa dimensione: una modello di piccole dimensioni generalizza meglio la struttura del grafo e ne migliora la velocità di convergenza.
- Community aware: Consapevoli della comunità. È importante che le similarità latenti tra i nodi del grafo emergano facilmente, con latente si intendono tutte quelle similarità che sono presenti tra i nodi ma che non sono rilevabili dalla struttura del grafo.
- Spazio continuo: è importante che le relazioni latenti tra i nodi del grafo siano rappresentate in uno spazio continuo, cosicché sia più facile rilevare le differenze tra le diverse comunità di nodi, rendendo così la rappresentazione più robusta.

Il concetto di graph embedding è espresso formalmente come segue:

Dato un grafo pesato G(V, E, W) un embedding di G è una funzione di mappatura $f_m: V \rightarrow R^{|V| \times d}$, dove |V| è il numero dei nodi del grafo e $d \ll |V|$. Ogni nodo del grafo è rappresentato tramite un vettore d-dimensionale.

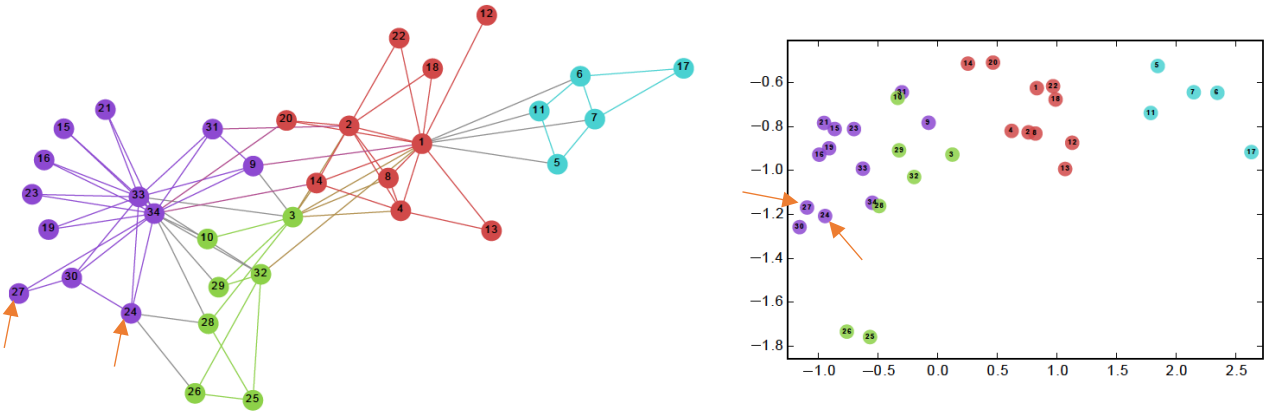


Figura 5 Zakary's karate club network embedding.

Figura 5 ripresa da [9], raffigura a sinistra la Zakary's Karate Network [10], a destra è rappresentato il suo embedding 2-dimensionale sul piano cartesiano. Come si può notare dalla figura i nodi "24" e "27", che sono indicati tramite due frecce arancioni, nella versione originale del grafo non sono collegati tramite un arco, quindi, a prima vista non sembrerebbero nodi simili, nella raffigurazione dell'embedding, invece, i due nodi sono molto vicini, quindi potenzialmente simili.

4 PROCEDIMENTO

Nello specifico per svolgere questo elaborato si è utilizzata la libreria SparkER scritta nel linguaggio Python.

SparkER [11] è una libreria di Python che deriva da Apache pySpark [S1]. Apache Spark è un motore analitico che serve per processare dati di larga scala. SparkER utilizza e modifica le funzionalità di pySpark per perseguire gli obiettivi della entity resolution. Più nello specifico per condurre gli esperimenti spiegati nel corso di questo capitolo, SparkER è stata utilizzata fino all'applicazione delle tecniche di pesatura del Blocking graph.

Questo elaborato tratta unicamente la Progressive ER per quanto riguarda dataset clean-clean, ovvero record-linkage, con opportuni accorgimenti il procedimento può essere applicato anche a dataset dirty per la deduplication.

4.1 ATTRIBUTE CLUSTERING

Prima ancora di procedere con il blocking, è importante procedere con una tecnica di pre-blocking chiamata *Attribute-match induction* [12], che viene definita come segue:

Dati due dataset (P_1, P_2) e detti (A_{P_1}, A_{P_2}) il set dei propri attributi, l'*attribute match induction* è il processo di identificare le coppie $\{< a_i, a_j > \mid a_i \in A_{P_1}, a_j \in A_{P_2}\}$ di attributi simili secondo una misura di similarità.

Tutti gli attributi disaccoppiati vengono inseriti in un cluster unico e vengono confrontati come se appartenessero allo stesso cluster in fase di blocking.

Questa tecnica utilizza il *locality sensitive hashing*, ma si possono impiegare anche altre misure, per calcolare la similarità, gli attributi che hanno una distanza minore di una certa soglia normalizzata vengono raggruppati nello stesso cluster. Questo passaggio è fondamentale per evitare ambiguità che potrebbero emergere nel procedere direttamente con la fase di Blocking.

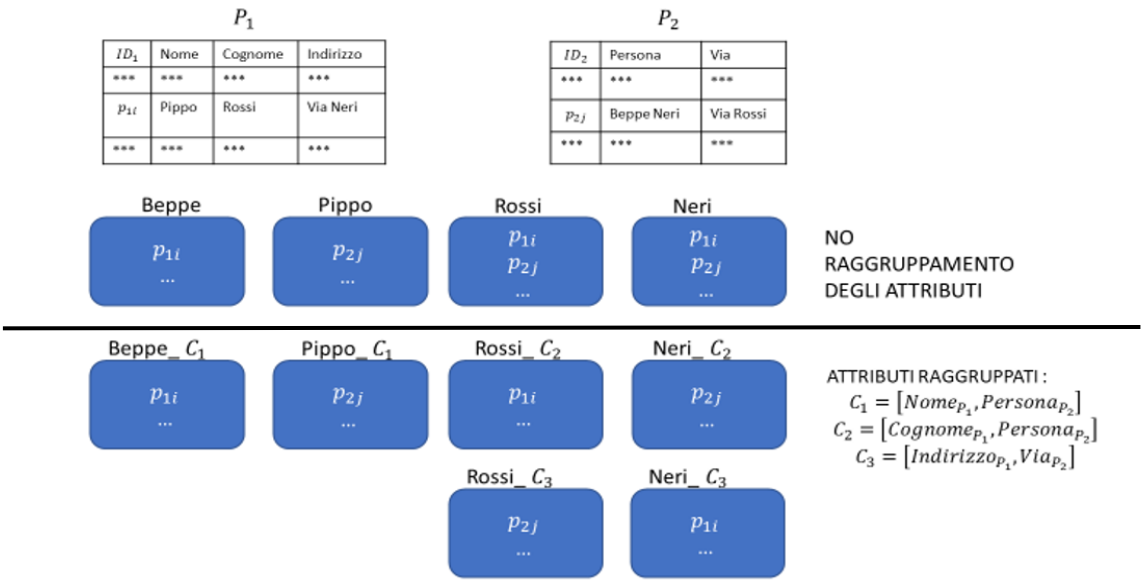


Figura 6 Attribute Clustering esempio parte 1

Figura 7 Attribute Clustering esempio parte

Come si può notare in Figura 7 i due profili esaminati in Figura 6 non sono simili, però in assenza dell'Attribute Clustering possono emergere

In assenza di questa fase potrebbero emergere coppie simili che in realtà non lo sono e di conseguenza produrre comparazioni inutili, e ancor peggio potrebbe risultare che queste coppie si riferiscono alla stessa entità e risultare possibili duplicati.

Quindi in fase di blocking per produrre un blocco, si deve tenere conto sia della chiave di blocking che del cluster di attributi in cui si trova. Ne deriva che due profili che condividono una chiave di blocking ma provenienti da attributi appartenenti a cluster differenti non condividono nessun token.

Ad ogni cluster di attributi è assegnato un coefficiente informativo calcolato in base all'entropia di Shannon, che rappresenta il contenuto informativo di ogni singolo raggruppamento. Questo coefficiente sarà utilizzato più avanti per il calcolo dei pesi del blocking graph.

4.2 TOKEN BLOCKING

La tipologia di blocking che è stata impiegata per gli esperimenti descritti in questo elaborato è il Token Blocking [13].

Attraverso questa tecnica ogni profilo proveniente dai dataset viene suddiviso in diverse chiavi di blocking K_{ij} dove i si riferisce al dataset P_i e j si riferisce al profilo $p_j \in P_i$.

Si dice K l'insieme totale delle chiavi di blocking, $k \in K$ è un riferimento ad una chiave di blocking generica

Le chiavi di blocking vengono estratte in base a diversi possibili fattori decisi in precedenza, come segni di punteggiatura, per esempio punti e virgole, trattini, underscore, spazi bianchi, numeri, caratteri speciali, o qualsiasi carattere o insieme di caratteri stabiliti essere separatori. Poiché questi separatori sono disposti in maniera generalmente non costante si ha come conseguenza che le chiavi di blocking sono stringhe di lunghezza variabile. Ogni token dipende anche dal cluster degli attributi in cui è inserito, per cluster diversi, ma chiavi di blocking uguali vengono prodotti token differenti.

Per ogni chiave di blocking e per ogni cluster di attributi in cui appare la chiave di blocking viene prodotto un blocco che contiene gli identificatori dei profili in cui appare.

La similarità tra due nodi può essere semplicemente misurata in fase iniziale come il numero di token che condividono, tenendo conto anche del coefficiente informativo del cluster calcolato precedentemente. Si assume che due profili duplicati abbiano almeno un blocco in comune; quindi, non verranno considerati come possibili duplicati le coppie di profili che non condividono blocchi.

In seguito a questa fase possono essere eseguite o meno operazioni di filtraggio, che eliminano, per esempio i blocchi con il più alto numero di profili, difficili da processare e con un contenuto informativo basso; oppure eliminare i blocchi in cui appare un solo profilo, che sono ovviamente inutili.

Al termine di questa fase viene prodotto un grafo pesato e indiretto, il blocking graph, avente come nodi i profili. In questo grafo un arco è presente o meno se i due profili condividono almeno un blocco. Il peso di ogni arco è calcolato attraverso metodi di pesatura che tengono conto del numero di blocchi che due profili condividono e del contenuto informativo dei blocchi in cui sono inseriti tali profili.

Il blocking graph non tiene conto delle similità tra profili appartenenti allo stesso dataset, quindi sarà un grafo bipartito.

Queste operazioni di filtraggio sono volte ad innalzare l'efficienza PQ ed RR, mantenendo un'efficacia PC invariata, o diminuita in maniera non significativa.

4.3 META-BLOCKING INIZIALE

Una volta ottenuto il Blocking Graph $G(V, E, W)$ la prima cosa da fare è rimuovere attraverso alcuni metodi gli archi meno importanti, ovvero gli accoppiamenti tra due nodi che si riferiscono a due profili che hanno poca probabilità di essere duplicati.

In particolare, per condurre gli esperimenti descritti in questo elaborato si è utilizzato il Wehigted Node Pruning WNP [12].

Tramite questo metodo, per ogni nodo p_i viene calcolata una soglia locales_i, $s_i = M_i/c$, dove M_i è il peso massimo tra quelli degli archi adiacenti a p_i e c è un valore costante per esempio c=2.

Ogni arco $e_{ij} \in E$ che collega due profili (p_i, p_j) ha come peso $w_{ij} \in W$, sia s_{ij} la soglia locale dell'arco w_{ij} calcolata come segue, $s_{ij} = s_i + s_j/d$ dove d è una costante, per esempio d=2, se $w_{ij} < s_{ij}$ allora l'arco e_{ij} viene potato e si considerano i profili (p_i, p_j) associati ai nodi (v_i, v_j) non duplicati.

Terminate queste operazioni si può procedere con la fase di graph embedding.

4.4 EMBEDDING DEL META-BLOCKING GRAPH

L'obiettivo come specificato precedentemente è quello di trovare una funzione di mappatura che rende più facile il rilevamento delle similarità tra nodi e renda più intuibile la complessità del grafo. Nel corso di questo capitolo verranno descritte le tecniche utilizzate per perseguire tale scopo.

4.4.1 DEEPWALK

DeepWalk [14] utilizza le informazioni ricavate dal grafo attraverso cammini casuali per imparare le rappresentazioni latenti di esso. DeepWalk è un metodo basato sul metodo word2vec [15]. Il secondo richiede in ingresso un corpus e restituisce un insieme di vettori che rappresentano la distribuzione semantica delle parole nel testo, il primo richiede in ingresso un grafo e restituisce un insieme di vettori che rappresentano la distribuzione dei vertici nel grafo. Quindi, DeepWalk parallelamente a word2vec tratta il grafo come se fosse un testo e i nodi come se fossero parole.

Definizione di cammino casuale: Dato un grafo G(V, E, W) si definisce cammino casuale Υ_{v_i} ruotato intorno al vertice $v_i \in V$ di lunghezza l un insieme ordinato di vertici $(v_i, v_{i+1}, \ldots, v_{i+n-1})$ tale che $v_{i+k} (\forall k < l, k \in \mathbb{N})$ sia un vertice scelto casualmente secondo una determinata probabilità tra i vertici collegati tramite un arco a v_i, ovvero tra i suoi vicini.

Sia V_{v_i} l'insieme dei nodi collegati al vertice v_i, sia $W_{v_i} = \sum_{w \in W_{v_i}} w$ ovvero, la somma dei pesi degli archi dei nodi ad esso adiacenti, la probabilità di passare dal nodo v_i a quello $v_j \in V_{v_i}$ è data dal rapporto tra il peso w_{ij} e la somma di tutti i pesi degli archi associati a v_i:

$$P_{ij} = \frac{w_{ij}}{W_{v_i}}$$

Un altro metodo preliminare per capire come lavora DeepWalk è l'algoritmo Skip-Gram [16], che in questo elaborato non verrà trattato. Skip-Gram ha il compito di aggiornare gli embedding per ogni cammino casuale secondo la distribuzione di probabilità dei vertici nei cammini casuali.

L'algoritmo deepwalk necessita di un numero di parametri:

- d, è la dimensione dei vettori d-dimensionali associati ad ogni nodo del grafo, che costituiscono l'embedding.
- n, è il numero di cammini casuali Υ_{v_i} per ogni nodo $v_i \in V$, quindi durante l'algoritmo verranno prodotti $|V| \times n$ cammini casuali.
- l, rappresenta la lunghezza di ogni cammino casuale.

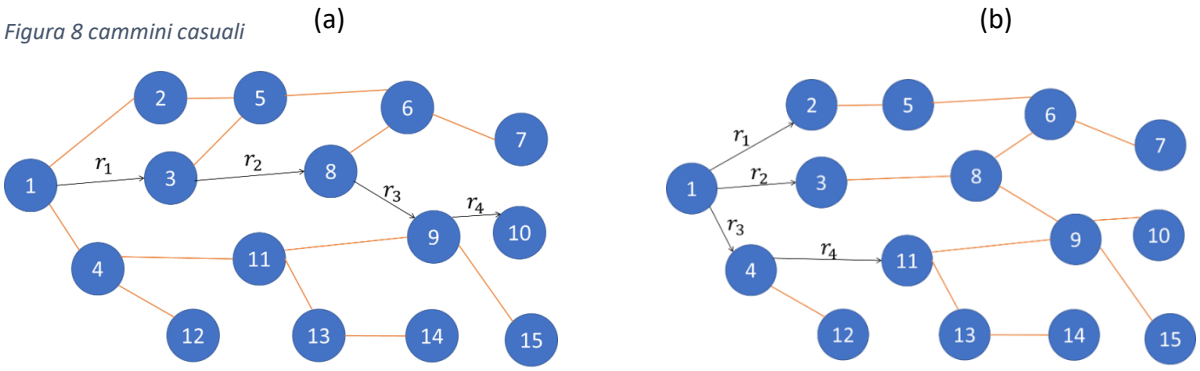
Come output DeepWalk fornisce una matrice $\Phi \in \mathbb{R}^{|V| \times d}$, dove ogni riga r_i rappresenta l'embedding del nodo $v_i \in V$.

4.4.2 NODE2VEC

Il node2vec [17] è simile al DeepWalk, l'unica differenza si trova nella costruzione dei cammini, che in questo metodo sono semi-casuali.

In un cammino semi-casuale Υ_{v_i} appaiono prima tutti i nodi adiacenti a $v_i \in V$, una volta terminati questi nodi si procede come in deepwalk. Quindi, se il nodo v_i ha $|V_{v_i}|$ vicini e la lunghezza del cammino casuale è l, si avrà che i primi $|V_{v_i}| + 1$ nodi del cammino sono sempre costituiti da v_i (in prima posizione) e dai nodi ad esso collegati, gli altri $l - (|V_{v_i}| - 1)$ nodi sono inseriti tramite lo stesso metodo utilizzato da deepwalk a partire dall'ultimo nodo $v_j \in V_{v_i}$ inserito nel cammino.

Figura 8 cammini casuali



Nel grafo Figura 8 (a) è rappresentato un possibile cammino casuale secondo la tecnica utilizzata per deepwalk $\Upsilon_{v_1} = (v_1, v_3, v_8, v_9, v_{10})$, mentre nel grafo (b) secondo la tecnica node2vec $\Upsilon_{v_1} = (v_1, v_2, v_3, v_4, v_{11})$. Come mostrato, il primo cammino esplora direttamente il grafo in "profondità", il secondo cammino prima esplora tutti i nodi vicini a v_1 ed in seguito esplora il grafo in "profondità".

4.4.3 HARP

Come metodo di embedding si è utilizzato HARP [18].

Questo metodo allo stato dell'arte è il migliore nel catturare le rappresentazioni latenti del grafo, ovvero le similarità nascoste tra i nodi che non sono rilevabili nelle rappresentazioni del grafo di partenza.

HARP(Hierarchical Rapresentation Learning for networks) è una meta-strategia che fa uso di tecniche di graph embedding per perseguire gli stessi obiettivi di quest'ultime. Per svolgere gli esperimenti spiegati nel corso di questo elaborato, si è utilizzato HARP abbinato a deepwalk e node2vec, poiché le prestazioni di HARP sono migliori delle prestazioni dei due metodi applicati direttamente al grafo.

HARP riceve in input un grafo $G(V, E, W)$ e una funzione di embedding (come, per esempio, deepwalk o node2vec con i relativi parametri). Dapprima, tramite opportuni algoritmi viene generata una gerarchia di grafi di dimensione decrescente "collassando" ,ovvero, unendo ad ogni iterazione i nodi della rete più simili tramite tecniche di collapsing; una volta raggiunto un numero di nodi significativamente basso viene generato l'embedding del grafo di dimensione minore con la funzione passata in input; in seguito, questo embedding viene esteso e aggiornato ad ogni iterazione ripercorrendo la successione di grafi crescente per numero di nodi applicando ad ogni grafo il metodo di embedding, fino ad applicare la funzione di embedding al grafo di partenza.

Questo metodo è migliore rispetto ai singoli node2vec e deepwalk perché partendo da grafi minori l'embedding riesce a rappresentare meglio la configurazione strutturale di alto livello del grafo di partenza; quindi, poiché l'embedding è aggiornato ad ogni iterazione, imparando così sempre più nello specifico il legame tra i nodi del grafo, riesce a rappresentare una struttura a bassa dimensione e altamente ordinata del grafo di partenza.

5 RISULTATI E GRAFICI

Sono stati condotti esperimenti attraverso sei coppie di dataset clean-clean aventi le caratteristiche elencate in Tabella 1.

Ogni dataset si riferisce ad entità del mondo reale.

Tabella 1

Dataset	|D1|	|D2|	|GT|	Recall	Precision
Imdb_tmdb	5.1k	6.0k	1.9k	0.988	$1.78*10^{-2}$
Imdb_tvdb	5.1k	7.8k	1.1k	0.985	$8.90*10^{-3}$
scholarDblp	2.5k	61.3k	2.3k	0.998	$2.80*10^{-3}$
Tmdb_tvdb	6.0k	7.8k	1.1k	0.989	$5.50*10^{-3}$
DblpAcm	2.6k	2.3k	2.2k	0.999	$4.81*10^{-2}$
movies	27.6	23.1	22.8k	0.976	$8.59*10^{-4}$

Nella Tabella 1. |D1| indica il numero di elementi nel primo dataset della coppia, |D2| il secondo. |GT| indica la grandezza del groundtruth, ovvero il numero delle coppie di duplicati reali. Recall indica il rapporto tra il numero di coppie di duplicati richiamati dopo il meta-blocking e il numero di duplicati effettivi, ovvero|GT|. Precision indica il rapporto tra il numero di duplicati richiamati dopo il meta-blocking e il numero di comparazioni possibili tra coppie richiamate dopo il meta-blocking.

Le coppie di dataset imdb_tmdb, imdb_tvdb, tmdb_tvdb, comparano film e serie TV estratte da IMDB, TheMovieDB e TheTVDB. ScholarDblp compara articoli scientifici estratti da scholar.google.com e dblp.org. DblpAcm compara articoli scientifici estratti da dblp.org e dl.acm.org. Movies compara informazioni riguardante i film che sono estratte da imdb.com and dbpedia.org.

Definizione: si definisce metodo ideale, quello che compara tutte e sole le coppie di duplicati, quindi, quello che dopo |GT| comparazioni ha richiamato il 100% di duplicati.

Dato un grafico cartesiano, avente per ascisse il numero di comparazioni effettuate e avente sull'asse delle ordinate la recall, si definisce AUC, area under the curve, la porzione di grafico che è contenuta al di sotto della curva e al di sopra dell'asse delle ascisse.

In particolare, AUC del metodo ideale dopo n comparazioni si calcola nel seguente modo:

$$AUC\_I@n = \begin{cases} \sum_{k=1}^{n} \frac{k}{|GT|}, & n < |GT| \\ \frac{|GT|+1}{2} + (n - |GT|), & n \geq |GT| \end{cases}$$

Essendo che ad ogni comparazione il numero di coppie rilevate aumenta di 1 per il metodo ideale, la recall aumenta progressivamente. Dopo |GT| comparazioni, la recall totale ha raggiunto il massimo, ovvero 1. Quindi, dopo ogni comparazione che si effettua in seguito la recall rimarrà invariata ad 1.

AUC di un metodo M (HARP(DeepWalk), HARP(node2vec), PPS) dopo n comparazioni si calcola nel seguente modo:

$$AUC\_M@n = \sum_{k=1}^{n} \frac{M[k]}{|GT|}$$

Dove M[k] indica il numero di duplicati rilevati dal metodo M dopo k comparazioni.

Si indica con $AUC\_M@ngt$ AUC del metodo M dopo n*|GT| comparazioni.

In questo elaborato verranno considerate AUC normalizzate rispetto al metodo ideale:

$$AUC\_M@ngt' = \frac{AUC\_M@ngt}{AUC\_I@ngt}$$

AUC di un metodo è importante per verificarne la propria velocità nel trovare duplicati.

Per ogni dataset vengono mostrate cinque immagini:

- Un grafico cartesiano che presenta sull'asse delle ordinate la recall raggiunta, mentre sull'asse delle ascisse presenta il numero di comparazioni eseguite, la legenda è spiegata in seguito.
- Quattro grafici a barre, uno per ogni n=1,2,3,5 di $AUC\_M@ngt'$, ciascuno avente tre barre rappresentanti rispettivamente M=HARP(DW), HARP(n2v), PPS.

SPIEGAZIONE DEI GRAFICI

Ogni grafico cartesiano contiene un segmento rosso tratteggiato che raffigura il metodo ideale; una curva di colore blu che rappresenta HARP(deepwalk); una curva di colore arancione che rappresenta HARP(node2vec); una curva di colore verde che rappresenta PPS.

Il metodo ideale è un segmento, a differenza degli altri metodi, perché la sua recall, come spiegato in precedenza cresce progressivamente.

5.1 DblpAcm

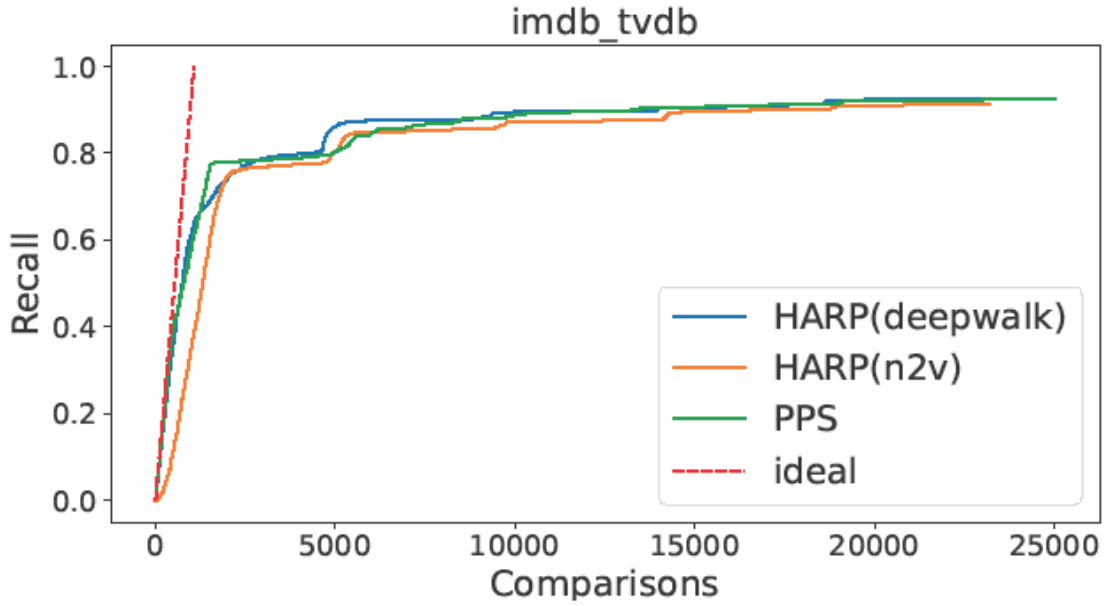


Figura 9 DblpAcm Recall

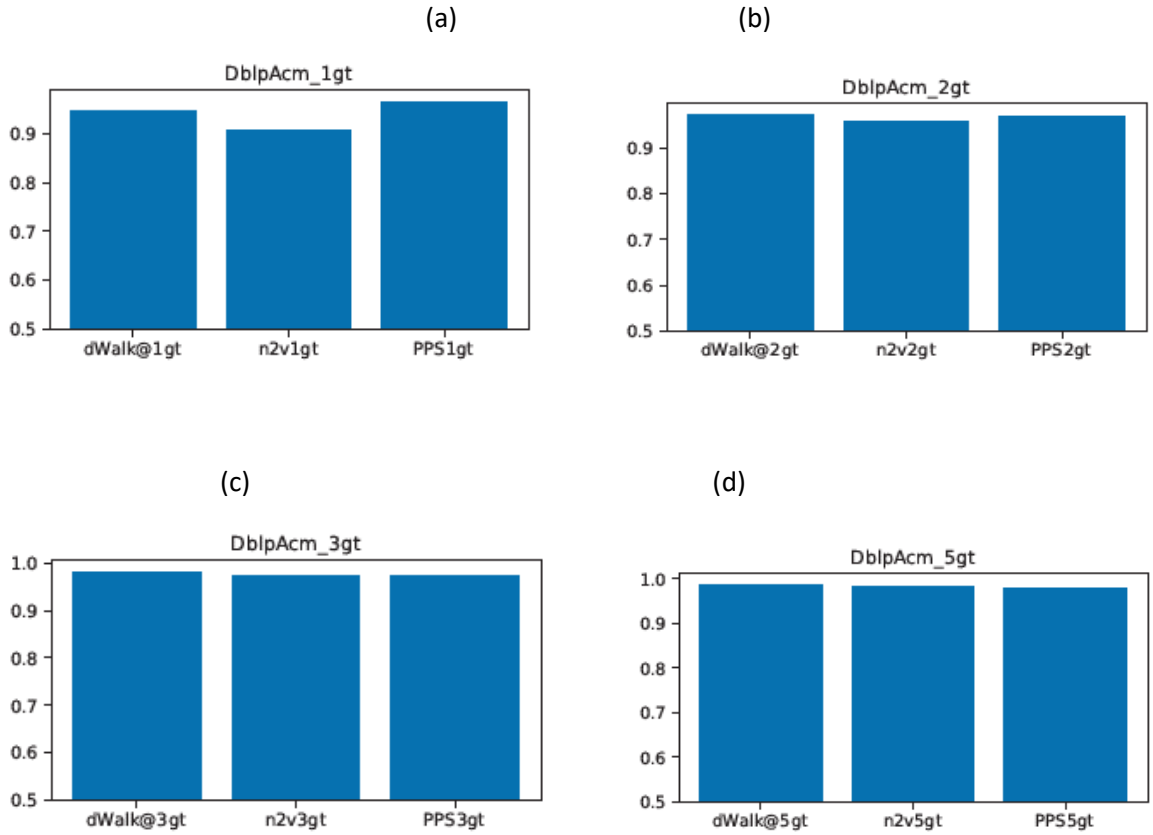


Figura 10 AUC_DblpACM

Dopo le prime |GT| comparazioni PPS ha trovato una percentuale di duplicati leggermente maggiore rispetto ad HARP(dw) (f.10-a), in seguito le percentuali di elementi duplicati rilevati sono circa le stesse per ogni metodo(f.10-b, 10-c, 10-d).

5.2 Imdb_tmdb

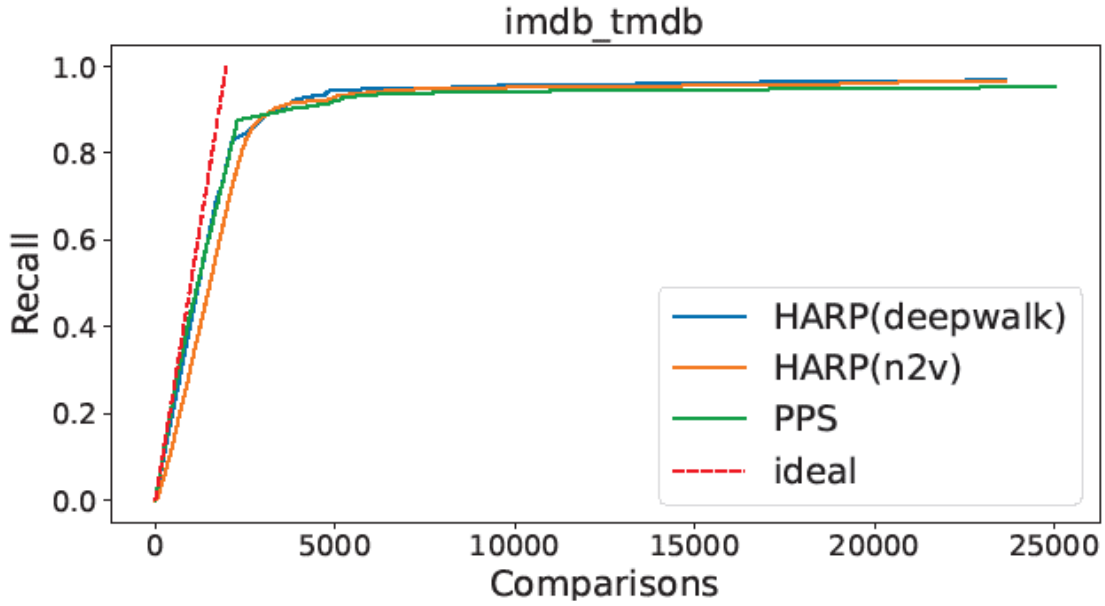


Figura 11 Imdb_tmdb recall

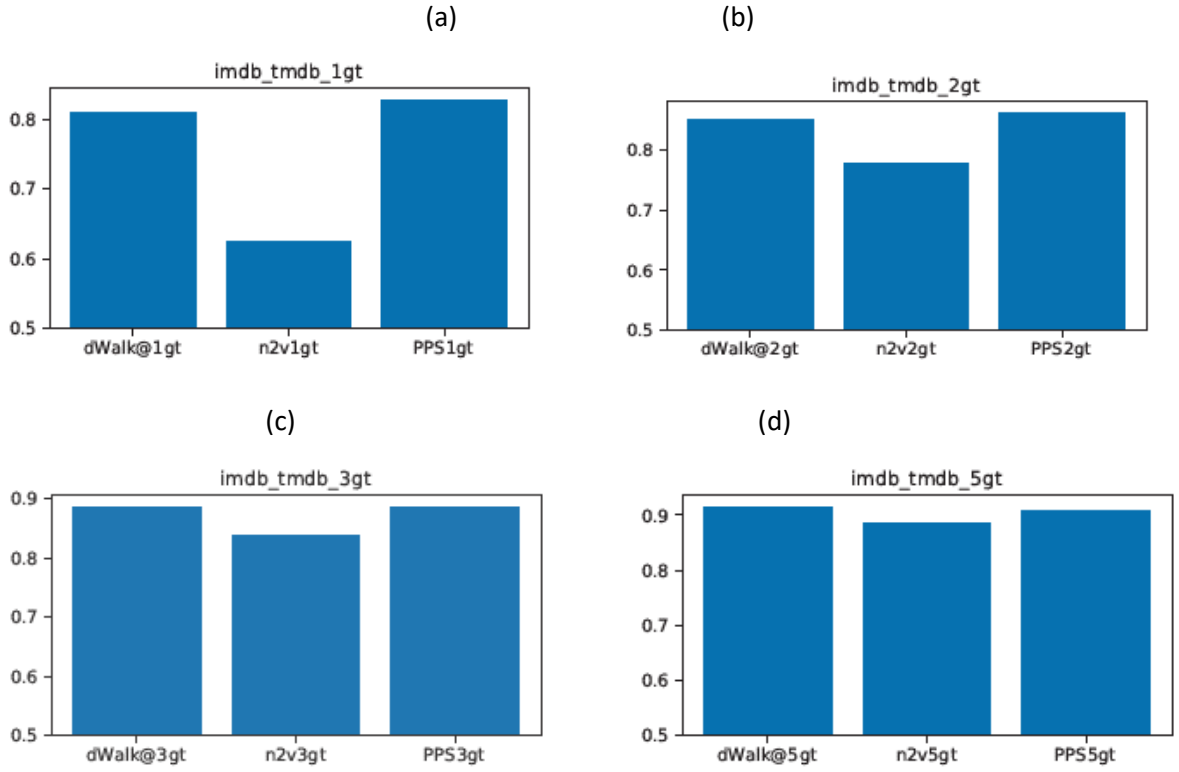


Figura 12 AUC_Imdb_tmdb

Anche in questo caso, dopo le prime |GT| comparazioni PPS ha trovato una percentuale di duplicati maggiore rispetto ad HARP(dw) e HARP(n2v)(f.12-a). Dopo $5 \times |GT|$ comparazioni il numero di duplicati rilevati da HARP(dw) e HARP(n2v) è maggiore di quelli del PPS (f.11), sebbene l'AUC di PPS sia ancora maggiore dell'AUC di HARP(n2v) (f.12-d).

5.3 ImdbTvdb

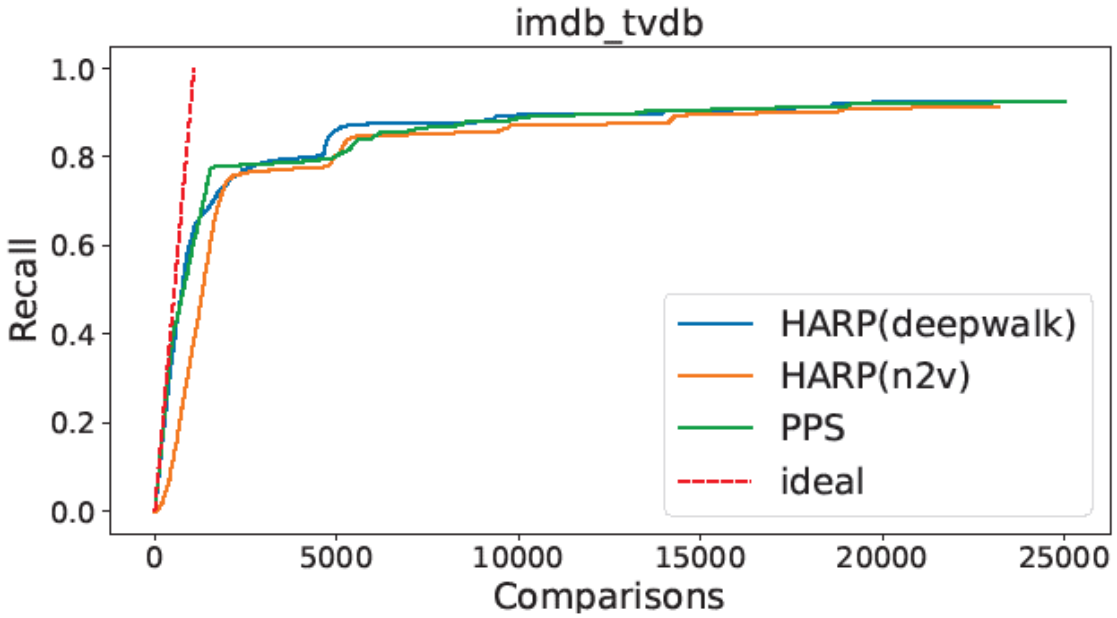
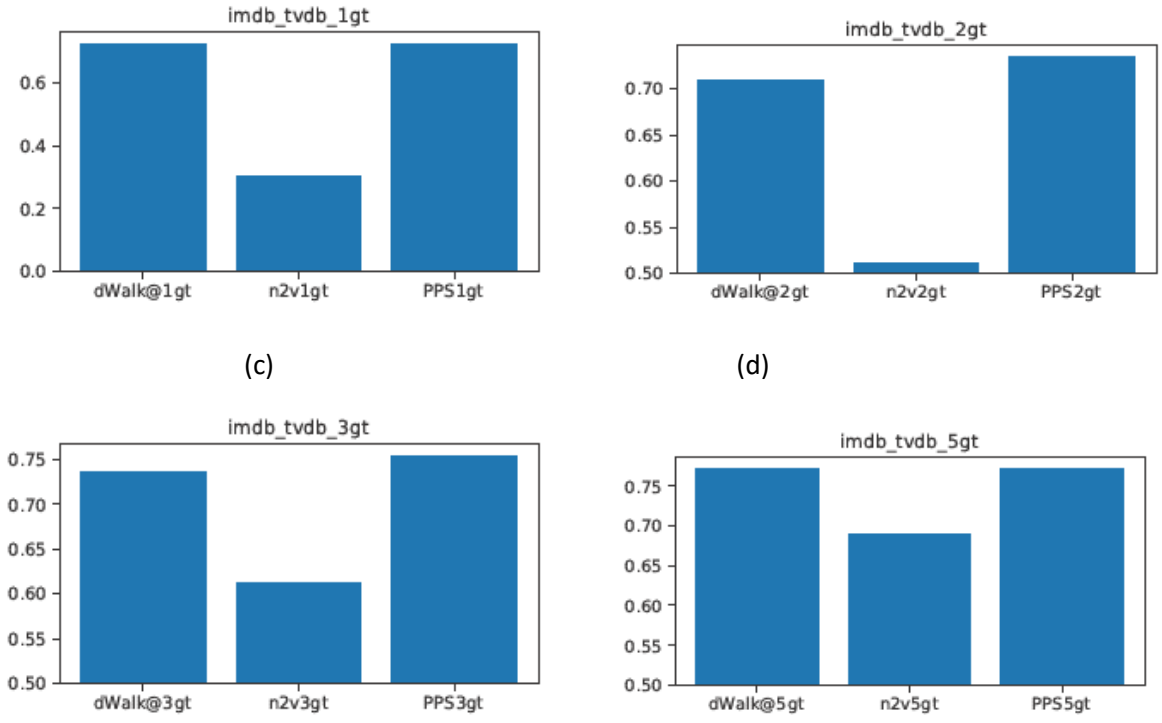


Figura 13 Imdb_tvdb recall

(a) (b)



(c) (d)

Figura 14 AUC_Imdb_tvdb

In questo caso, HARP(dw) e PPS hanno rilevato i duplicati con una velocità simile dopo un numero pari a |GT| comparazioni (f.13-a). Le prestazioni di HARP(dw) in seguito peggiorano leggermente(f.14-b, 14-c, 14-d). Dopo un numero relativamente elevato di comparazioni la recall è simile per tutti e tre i metodi(f.13), sebbene le velocità di rilevamento dei duplicati di HARP(dw) e PPS siano nettamente differenti da quelle di HARP(n2v) (f.14-d).

5.4 tmdb_tvdb

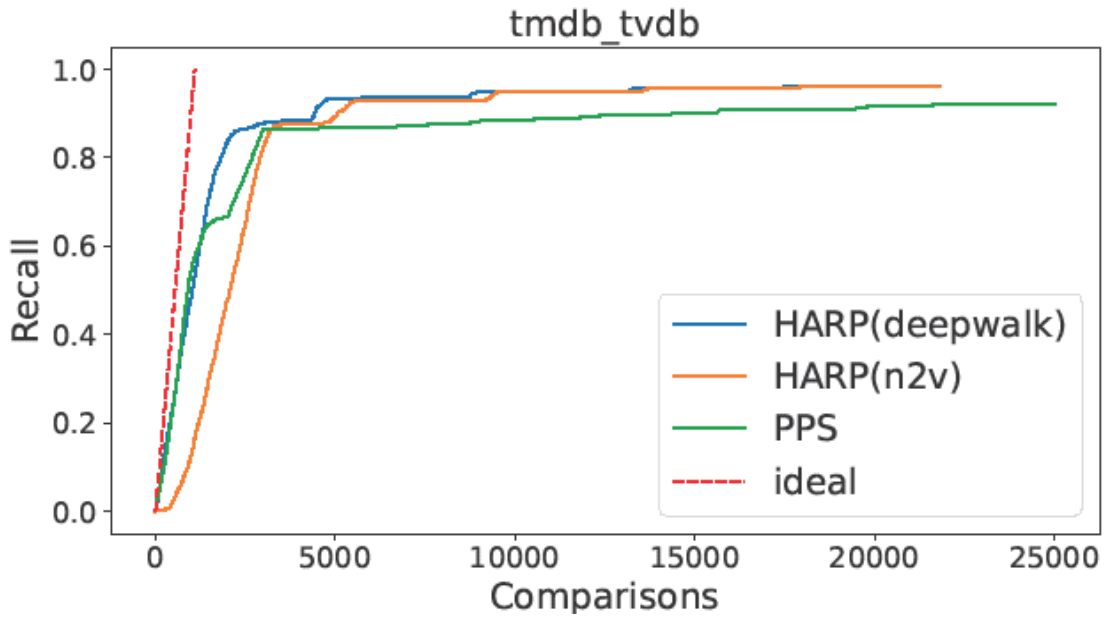


Figura 15 Tmdb_tvdb recall

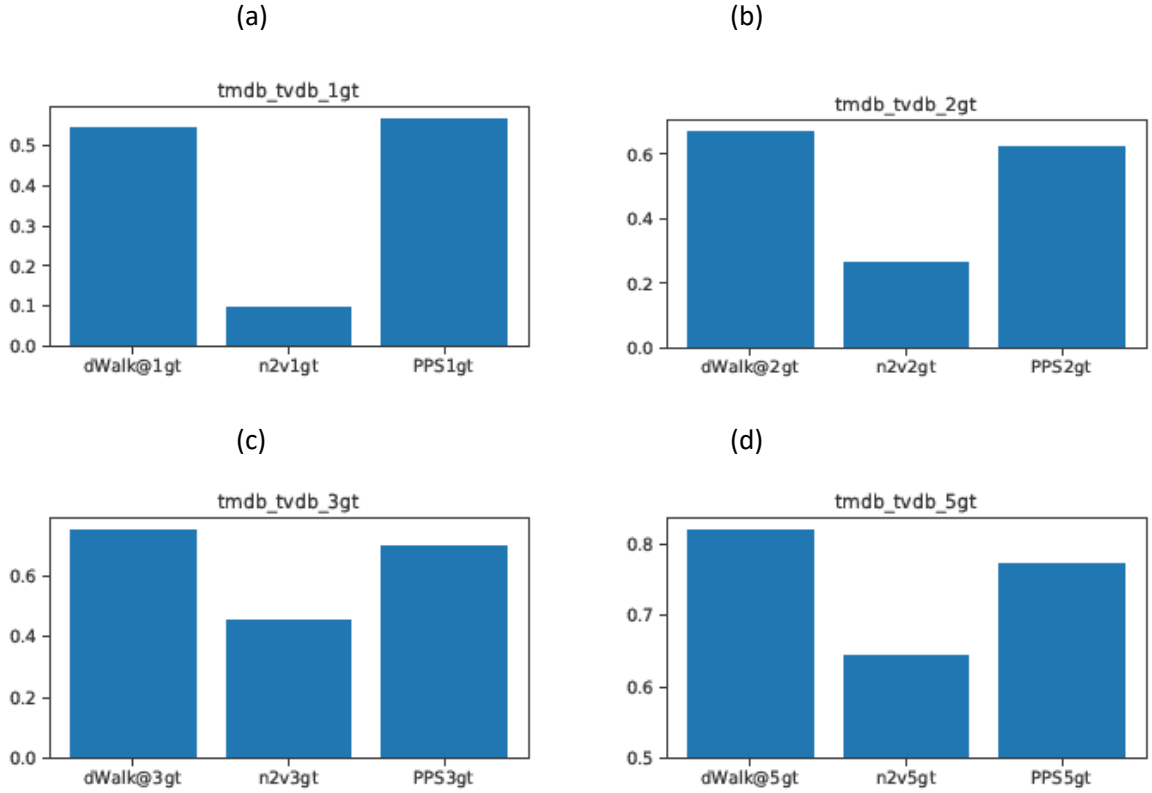


Figura 16 AUC_Tmdb_tvdb

In questo dataset, PPS ha rilevato i duplicati con una velocità leggermente maggiore rispetto ad HARP(dw) e le prestazioni di HARP(n2v) nella fase iniziale sono anche questa volta le peggiori tra le tre (f.16-a). Continuando con le comparazioni le prestazioni di HARP(dw) diventano le migliori (16-d). La recall finale per HARP(dw) e HARP(n2v) è circa la stessa (f.15).

5.5 ScholarDblp

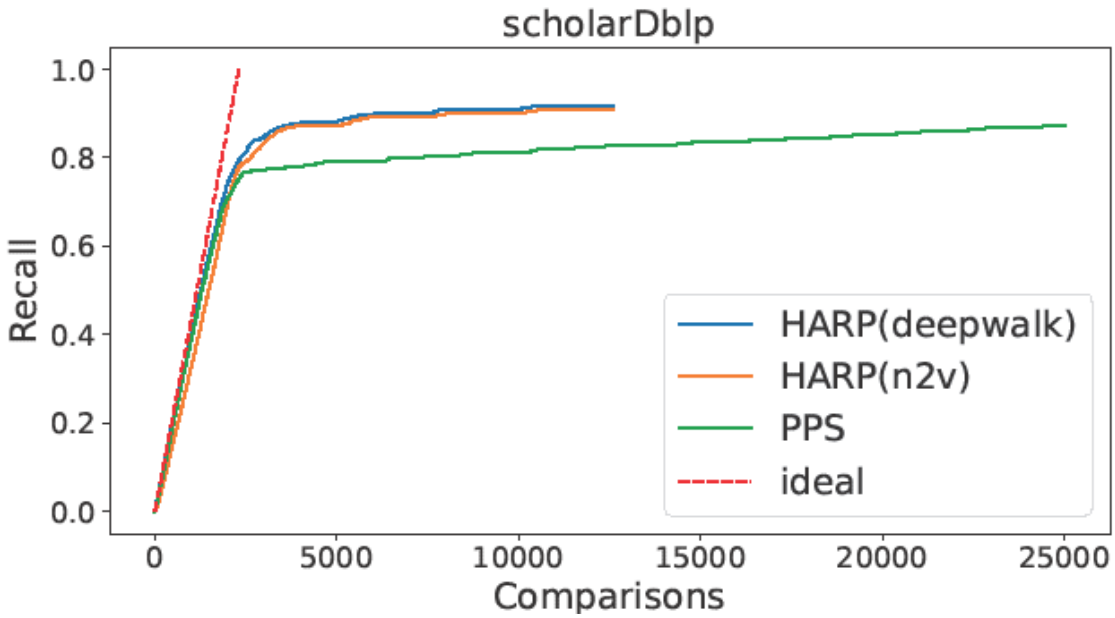
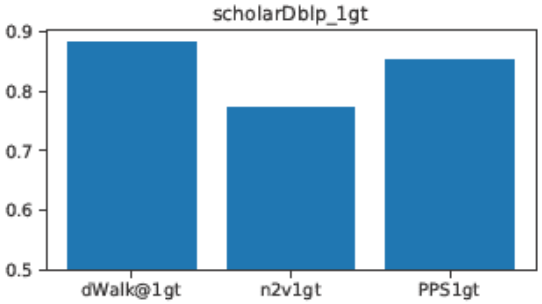
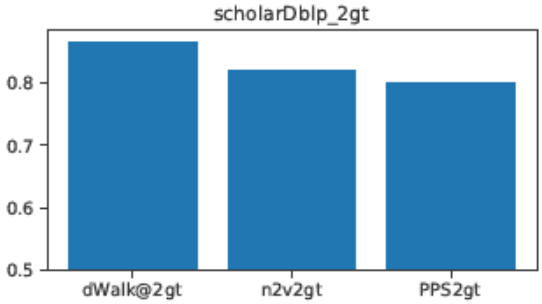


Figura 17 ScholarDblp recall

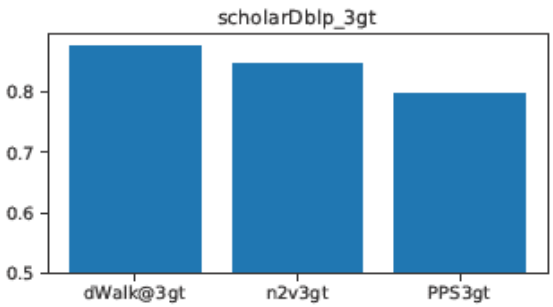
(a)



(b)



(c)



(d)

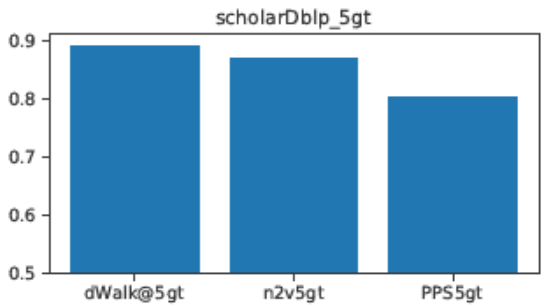


Figura 18 AUC_ScholarDblp

In questo caso, le prestazioni di HARP(dw) sono nettamente migliori dall'inizio alla fine rispetto a quelle di PPS (f.18-a, 18-b, 18-c, 18-d). Come in TmdbTvdb la recall finale di HARP(dw) e HARP(n2v) è la stessa (f.17), ma in questo caso le prestazioni finali differiscono leggermente (f.18-d). La netta differenza tra HARP(dw) e HARP(n2v) si trova nelle prestazioni iniziali, quelle più importanti, come si vede in (f18-a).

5.6 Movies

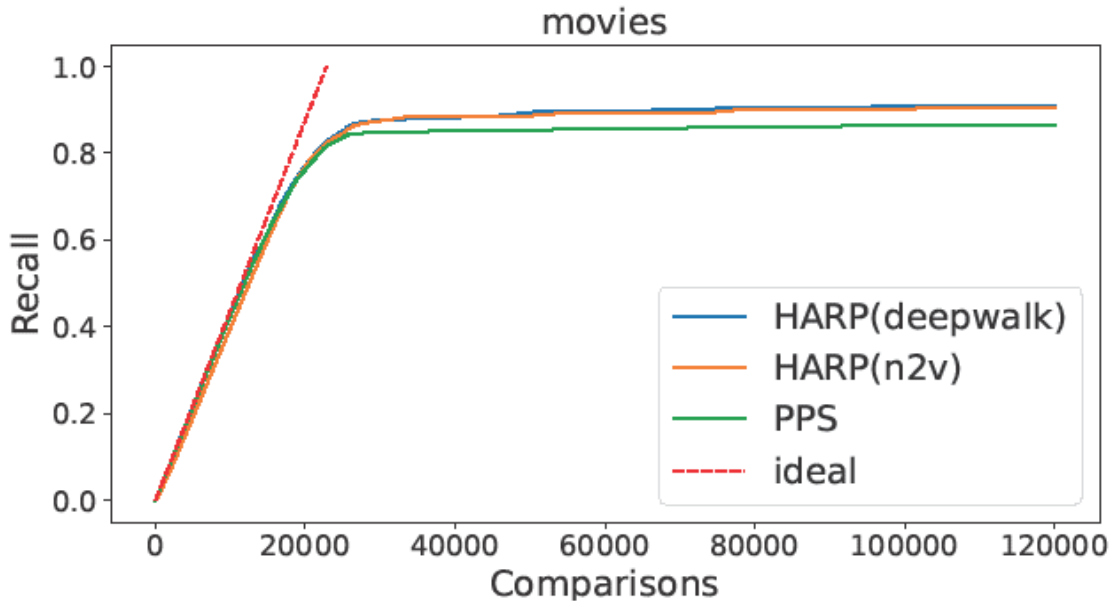


Figura 19 Movies recall

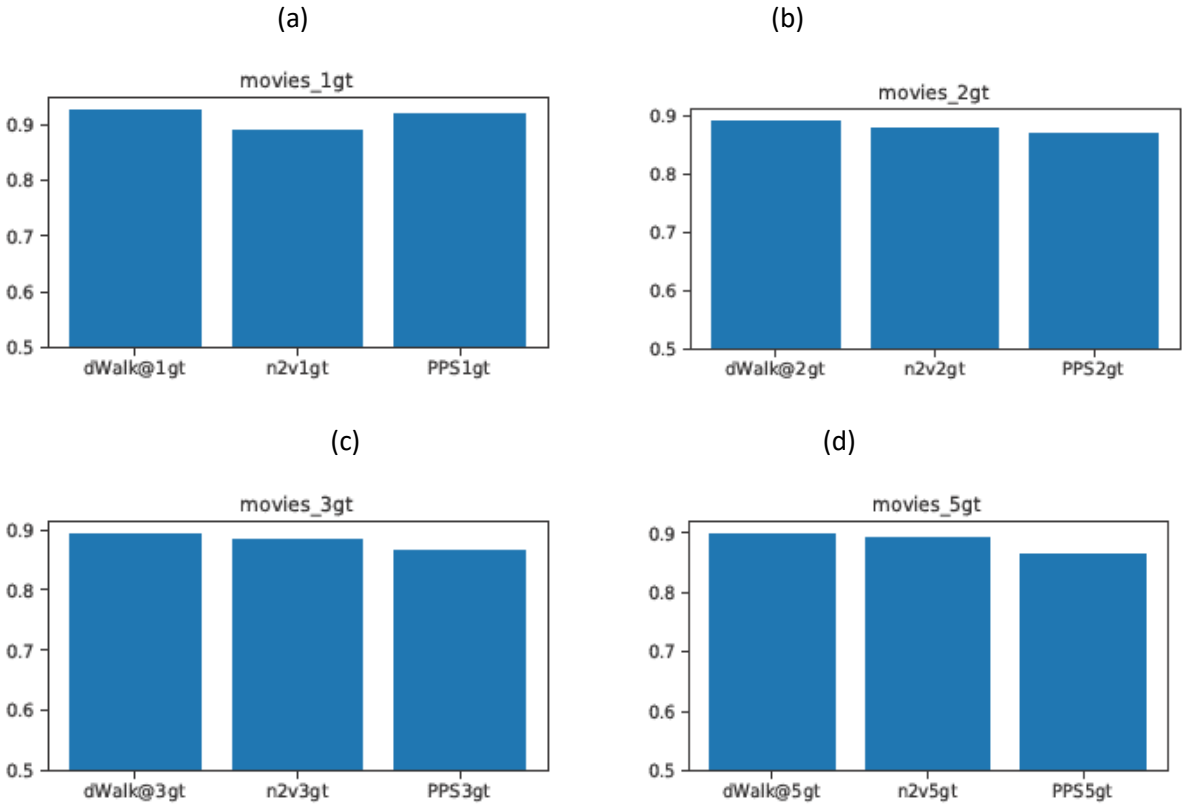


Figura 20 AUC_Movies

Anche in questo caso le prestazioni iniziali di HARP(dw) e PPS sono relativamente alte, con un leggero vantaggio del primo metodo(f.20-c). La recall e le prestazioni di PPS peggiorano progressivamente rispetto a quelle degli altri due metodi (f.20-b, 20-c, 20-d).

5.7 Media

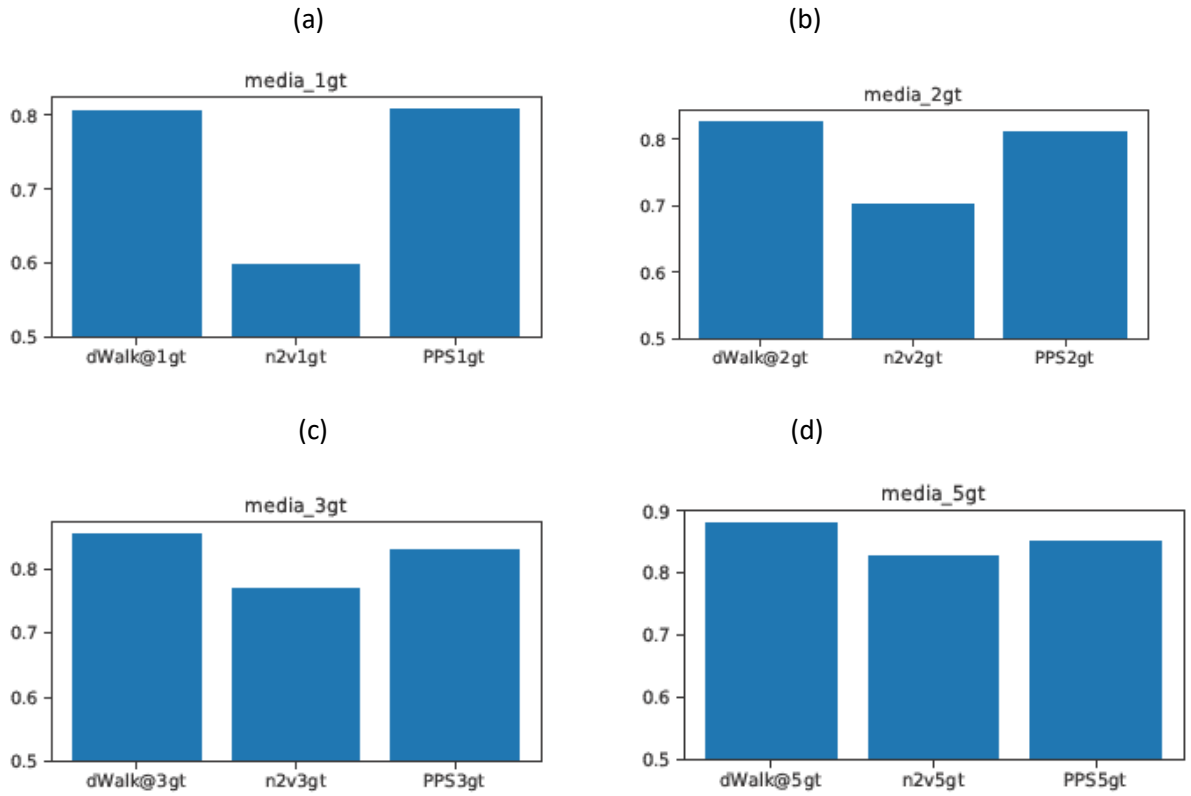


Figura 21 AUC_Media

In questi quattro grafici sono paragonate le AUC dei tre metodi differenti. In (f.21-a) si nota che dopo un numero di comparazioni pari a |GT| la velocità di rilevamento dei duplicati è molto simile per PPS e per HARP(dw), mentre è nettamente inferiore quella di HARP(n2v). Le (f.21-b, 21-c, 21-d) mostrano come continuando con le comparazioni le prestazioni di HARP(dw) diventano migliori rispetto a quelle del PPS. Le prestazioni di HARP(n2v) aumentano ma non abbastanza per raggiungere quelle degli altri due metodi.

CONCLUSIONI

In questa tesi è stato affrontato il problema dell' Entity Resolution (ER) progressiva, cioè il task di prioritizzare coppie di record potenzialmente duplicate quando si hanno vincoli temporali o scarse risorse.

Per fare ciò, è stato utilizzando un consolidato framework di meta-blocking che rappresenta coppie di possibili record duplicati in un grafo (*blocking graph*). In particolare, è stato studiato come sfruttare tecniche di graph embedding per sfruttare caratteristiche latenti del blocking graph, così da migliorare le performance rispetto alle tecniche esistenti in letteratura.

È stato sviluppato un prototipo in Python e condotto uno studio sperimentale su sei dataset reali. I risultati suggeriscono che il metodo proposto in questa tesi può essere impiegato quando le risorse e/o il tempo sono limitati per produrre un ER approssimata. Le sue performance, in termini di precision e recall sono sempre uguali o migliori del metodo PPS, lo stato dell'arte per quanto riguarda i metodi progressivi e basati su blocking graph.

Come lavoro futuro sarebbe interessante approfondire come integrare la fase di blocking e quella di matching, in un framework unificato.

BIBLIOGRAFIA

[1] Fellegi, I. P., & Sunter, A. B. (1969). A theory for record linkage. *Journal of the American Statistical Association*, *64*(328), 1183-1210.

[2] Hernández, M. A., & Stolfo, S. J. (1995). The merge/purge problem for large databases. *ACM Sigmod Record*, *24*(2), 127-138.

[3] Benjelloun, O., Garcia-Molina, H., Kawai, H., Larson, T. E., Menestrina, D., Su, Q., ... & Widom, J. (2006). *Generic entity resolution in the serf project*. Stanford InfoLab.

[4] Talburt, J. R. (2011). *Entity resolution and information quality*. Elsevier.

[5] Papadakis, G., Skoutas, D., Thanos, E., & Palpanas, T. (2020). Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys (CSUR)*, *53*(2), 1-42.

[6] Papadakis, G., Koutrika, G., Palpanas, T., & Nejdl, W. (2013). Meta-blocking: Taking entity resolutionto the next level. *IEEE Transactions on Knowledge and Data Engineering*, *26*(8), 1946-1960.

[7] Simonini, G., Papadakis, G., Palpanas, T., & Bergamaschi, S. (2018). Schema-agnostic progressive entity resolution. *IEEE Transactions on Knowledge and Data Engineering*, *31*(6), 1208-1221.

[8] Chen, H., Perozzi, B., Al-Rfou, R., & Skiena, S. (2018). A tutorial on network embeddings. *arXiv preprint arXiv:1808.02590*.

[9] Hamilton, W. L., Ying, R., & Leskovec, J. (2017). Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*.

[10] Zachary, W. W. (1977). An information flow model for conflict and fission in small groups. *Journal of anthropological research*, *33*(4), 452-473.

[11] Gagliardelli, L., Simonini, G., Beneventano, D., & Bergamaschi, S. (2019). Sparker: Scaling entity resolution in spark. In *EDBT 2019: 22nd International Conference on Extending Database Technology*. PRT.

[12] Simonini, G., Bergamaschi, S., & Jagadish, H. V. (2016). BLAST: a loosely schema-aware meta-blocking approach for entity resolution. *Proceedings of the VLDB Endowment*, *9*(12), 1173-1184.

[13] Papadakis, G., Ioannou, E., Palpanas, T., Niederée, C., & Nejdl, W. (2012). A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Transactions on Knowledge and Data Engineering*, *25*(12), 2665-2682.

[14] Perozzi, B., Al-Rfou, R., & Skiena, S. (2014, August). Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 701-710).

[15] Church, K. W. (2017). Word2Vec. *Natural Language Engineering*, *23*(1), 155-162.

[16] Guthrie, D., Allison, B., Liu, W., Guthrie, L., & Wilks, Y. (2006, May). A closer look at skip-gram modelling. In *LREC* (Vol. 6, pp. 1222-1225).

[17] Grover, A., & Leskovec, J. (2016, August). node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 855-864).

[18] Chen, H., Perozzi, B., Hu, Y., & Skiena, S. (2018, April). Harp: Hierarchical representation learning for networks. In *Proceedings of the AAAI conference on artificial intelligence* (Vol. 32, No. 1).

SITOGRAFIA

[S1] https://spark.apache.org/