UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

Corso di Laurea in Informatica

Nuovi approcci al Clustering Deletion

Relatore:
Ch.mo Prof.
Ugo VACCARO

Candidato:
Giuseppe
AMBROSIO
Mat. 0522500834

ANNO ACCADEMICO 2020/2021

Sommario

Nel corso della tesi si approfondisce il problema del Clustering Deletion. Il Clustering Deletion è un problema di ottimizzazione combinatoria su grafi il cui scopo è: dato un grafo $G = (V, E)$, determinare il minimo numero di edge da eliminare, in G, affinchè il grafo risultante a seguito dell'eliminazioni di edge sia un cluster graph, ovvero un insieme disgiunto di clique. Il problema del Clustering Deletion è un noto problema NP-hard.

I principali scopi della tesi sono stati: lo studio del Clustering Deletion per particolari tipi di grafi, chiamati grafi lineari - al fine di determinare se su questi tipi di grafi il problema fosse ancora NP-hard o meno; e lo studio del Clustering Deletion per grafi qualsiasi - per il quale è stato ideato un approccio basato su un'operazione nota in teoria dei grafi come edge contraction.

Nel capitolo 1 vengono date le principali definizioni che vengono utilizzate nella tesi. Nel capitolo 2 viene descritto un modello di PLI per il problema. Il capitolo 3 è un capitolo che ha lo scopo di spiegare da dove è nato l'interesse per il problema del Clustering Deletion e in particolare per il problema del Clustering Deletion su grafi lineari. Nel capitolo 4 vengono date le definizioni di grafi lineari e oggetti ordinati, discusse le principali caratteristiche di interesse e in più viene definito un modello di generazione uniforme per i grafi lineari. Nel capitolo 5 viene analizzato il problema del Clustering Deletion su grafi lineari, al fine cercare di capire se tale problema è un problema NP-hard anche su i grafi lineari. Vengono dapprima provati degli approcci greedy e poi sviluppato un algoritmo di programmazione dinamica polinonimale, dimostrando così che il problema non è NP-hard. Nel capitolo 6 viene studiato il problema del Clustering Deletion su grafi qualsiasi. Viene studiato un approccio basato su edge contraction dal quale sono stati progettati algoritmi probabilistici ed euristici al problema. Infine, nel capitolo 7 vengono tratte le conclusioni del lavoto di tesi e viene fatto un sunto dei possibili sviluppi futuri ai problemi trattati.

I pricipali risultati ottenuti sono stati: progettazione di un algoritmo di programmazione dinamica per il Clustering Deletion su grafi lineari di complessità temporale $\mathcal{O}(n^2)$, definizione di un modello di generazione uniforme per grafi lineari e progettazione di un algoritmo basato sull'operazione di edge contraction per il problema del Clustering Deletion su grafi.

INDICE

CAPITOLO 1

INTRODUZIONE

In questo capitolo viene data una descrizione ad alto livello di cosa verrà trattato nel corso della tesi. Nella sezione 1.1 viene spiegato cos'è il clustering e come si differenzia dal Graph Clustering. Nella sezione 1.2 viene descritto in maniera un pò più dettagliata il problema del Clustering Deletion e brevemente la struttura della tesi.

1.1 Graph Clustering

Il Clustering è uno dei principali problemi di ottimizzazione combinatoria e riveste un ruolo fondamentale in molteplici aree quali: machine learning, network analysis, data compression, bioinformatics, ecc. L'input del problema del clustering è tipicamente un insieme di elementi e un valore di similarità per ogni coppia di elementi. L'obiettivo è partizionare gli elementi in sottoinsiemi, i quali sono chiamati cluster, in modo tale che due meta-criteri sono soddisfatti: omogeneità - elementi nello stesso cluster sono molto simili tra di loro; e separazione - elementi tra cluster diversi sono poco simili tra loro. Una concreta realizzazione di tali meta-criteri genera differenti problemi di ottimizzazione combinatoria [1],[2]. In questa tesi verrà studiato il problema del Clustering

Deletion il quale è un problema di Graph Clustering, ovvero un problema di clustering su grafi. Diamo le principali definizioni necessarie a poter lavorare con i grafi.

Definizione 1.1.1. Grafo

Un grafo è un coppia ordinata $G = (V, E)$ di insiemi. L'insieme V viene chiamato comunemente insieme di vertici e l'insieme $E \subseteq V \times V$ insieme di archi.

Definizione 1.1.2. Vicinato di un nodo v, $N(v)$

Sia $G = (V, E)$ un grafo e sia $v \in V$ un nodo di G. Denotiamo con $N(v)$ l'insieme di tutti i nodi in G connessi con un edge al nodo v, ovvero:

$$N(v) = \{\, u \in V \mid (v, u) \in E \,\}$$

Definizione 1.1.3. Archi aventi un estremo uguale a v, $W(v)$

Sia $G = (V, E)$ un grafo e sia $v \in V$ un nodo di G. Denotiamo con $W(v)$ l'insieme di tutti gl'archi aventi un estremo uguale a v in G, ovvero:

$$W(v) = \{\, (v, u) \in E \mid u \in V \,\}$$

In un approccio basato sulla teoria dei grafi il clustering prevede la costruzione di un grafo a partire da: un insieme di elementi, un valore di similarità per ogni coppia di elementi e una threshold. L'insieme di elementi costituisce l'insieme V dei nodi, mentre l'insieme E degli archi è costituito da tutte le coppie di elementi che hanno come valore di similarità, un valore superiore alla threshold scelta. Nel problema del Clustering Deletion la realizzazione dei due meta-criteri sopra menzionati va in un certo senso a privilegiare il primo, ovvero l'omogeneità. Questo perchè viene richiesto che ogni cluster formi una clique, ovvero un insieme di elementi a due a due collegati da un arco.

Definizione 1.1.4. Clique

Sia $G = (V, E)$ un grafo e sia S un sottoinsieme di V, diciamo che S è una clique se il sottografo indotto di S su G è un sottografo completo, ovvero se:

$$\forall u, v \in S \Rightarrow (u, v) \in E$$

Definizione 1.1.5. Sottografo indotto da un sottoinsieme di vertici
Sia $G = (V, E)$ un grafo e sia S un sottoinsieme di V. Chiameremo il grafo $G_S = (S, E_S)$, con $E_S = \{ (u, v) \in E \mid u, v \in S \}$, sottografo di G indotto dal sottoinsieme S.

Mentre per quanto riguarda il secondo meta-criterio, ovvero la separazione si cerca di minimizzare. Vale a dire si cerca di minimizzare la presenza di nodi collegati da un edge in cluster diversi.

1.2 Clustering Deletion

Il Graph Modification Problem (o Edge Modification Problem [3]) è un problema su grafi definito nel modo che segue: dato un $G = (V, E)$ e una proprietà Π, determinare il minimo numero di edge da aggiungere o da eliminare affichè il grafo risultante soddisfi la proprietà Π. Se consideriamo quindi come proprietà l'essere un cluster graph, ovvero un insieme disgiunto di clique, abbiamo il Clustering Modification Problem.

Definizione 1.2.1. Cluster graph
Sia $G = (V, E)$ un grafo, G è cluster graph se è costituito da un insieme disgiunto di clique.

Più nello specifico possiamo distinguere tre problemi diversi a seconda delle operazioni di modifica che permettiamo: il Clustering Editing, il Clustering Completion e il Clustering Deletion. Nel problema del Clustering Editing sono permesse sia operazioni di aggiunta di edge che di rimozione, tale problema è un problema NP-hard [1]. Nel problema del Clustering Completion sono permesse solo operazioni di aggiunta di edge, tale problema non è un problema NP-hard e può quindi essere risolto in tempi rapidi all'ottimo [1]. Infine nel problema del Clustering Deletion sono permesse solo operazioni di rimozione di edge, tale problema è un problema NP-hard [3]. Nel corso nella tesi verrà approfondito lo studio del problema del Clustering Deletion, il quale ricapitolando: dato un grafo $G = (V, E)$, consiste nel determinare il

minimo numero di edge da eliminare, in G, affinchè il grafo risultante a seguito dell'eliminazioni di edge sia un cluster graph. Tale problema può essere definito in due modi alternativi che si differenziano per l'output richiesto. Può essere definito in termini di edge da eliminare e quindi in questo caso l'output è un insieme di edge, oppure equivalentemente in termini di partizione di nodi e quindi in questo caso l'output è una partizione dei nodi che induce il cluster graph ottimale, dove con cluster graph ottimale intendiamo il cluster graph ottenuto eliminando l'insieme di edge minimale.

Definizione 1.2.2. Sottografo indotto da una partizione di vertici
Sia $G = (V, E)$ un grafo e sia P una partizione di V. Chiameremo il grafo $G_P = (V, E_P)$, con $E_P = \{ (u, v) \in E \mid u, v \in T, \text{con } T \in P \}$, sottografo di G indotto dalla partizione P.

Chiaramente avendo il cluster graph è possibile determinare l'insieme di edge da eliminare e viceversa avendo l'insieme di edge da eliminare è possibile determinare il cluster graph. Di seguito sono riportate le due definizioni alternative del problema e nel corso della tesi verrà utilizzata una piuttosto che l'altra a seconda delle necessità.

Clustering Deletion in termini di edge da eliminare

Input:

- $G = (V, E)$ grafo

Output: $S \subseteq E$ tale che:

- $G' = (V, E')$ con $E' = E \setminus S$ è un cluster graph
- S è di cardinalità minima

Clustering Deletion in termini di partizione

Input:

- $G = (V, E)$ grafo

Output: Una partizione P dei nodi tale che:

- $G' = (V, E')$ il sottografo indotto dalla partizione P è un cluster graph
- $|E| - |E'|$ è minimizzato

Il problema del Clustering Deletion è un problema NP-hard [3]. Per i problemi NP-hard gli approcci risolutivi che solitamente vengono utilizzati sono: algoritmi approssimati, algoritmi euristici, algoritmi probabilistici, ecc. Nel capitolo 6 viene descritto un approccio basato su un'operazione nota in teoria dei grafi come edge contraction. A partire da tale approccio vengono proposti algoritmi probabilistici e euristici per il problema.

Capita spesso che quando si ha a che fare con problemi reali, essi vengano individuati come problemi NP-hard, ma che le reali istanze che si devono risolvere godano di determinate caratteristiche. Tali caratteristiche, se sfruttate, possono far sì che la risoluzione del problema su quel particolare tipo di input non sia più un problema NP-hard, e quindi invece di accontentarsi di una soluzione approssimata è possibile progettare un algoritmo che permette di trovare l'ottimo in tempi rapidi. Nel capitolo 3 viene descritto il problema della validazione delle dipendenze funzionali rilassate su un data stream, per cui è possibile adoperare un approccio risolutivo che prevede la risoluzione del problema del Clustering Deletion. Uno studio sulle caratteristiche dell'input ci permette, però di affermare che l'input ha una struttura ben definita diversa dalla struttura di un generico grafo. In particolare si mostra che l'input sono determinati tipi di grafi, i grafi lineari, che verranno introdotti brevemente nel capitolo 3 e descritti in maniera più dettagliata nel capitolo 4. Per i grafi lineari il problema del Clustering Deletion non è un problema NP-hard, infatti,

nel capitolo 5 viene proposto un algoritmo di programmazione dinamica per la sua risoluzione all'ottimo.

Dalla definizione del problema è facile notare che se un grafo G ha k componenti connesse $G_1, \ldots, G_k$, è possibile risolvere il Clustering Deletion su G risolvendo il Clustering Deletion su G_1, su G_2, ... e su G_k. L'unione della soluzione di quest'ultimi fornisce la soluzione per G.

Da questo momento in poi possiamo quindi concentrarci senza perdita di generalità solo su grafi connessi, in quanto se non connessi basta risolvere il problema per ogni componente connessa.

CAPITOLO 2

MODELLO DI PLI

In questo capitolo viene fornito un modello di Programmazione Lineare Intera (PLI) per il problema del Clustering Deletion. Ciò viene fatto principalmente perchè di solito quando si a che fare con un problema non conosciuto il modello di PLI permette di comprenderne meglio le caratteristiche principali. Nella sezione 2.1 vengono introdotte le principali definizioni utili alla compressione del modello di PLI il quale è descritto nella sezione 2.2.

2.1 Definizioni e relazioni con altri problemi

Definizione 2.1.1. Indipendent set

Sia $G = (V, E)$ un grafo e sia S un sottoinsieme di V, diciamo che S è un indipendent set se non ci sono edge tra nodi in S, ovvero se:

$$\forall (u, v) \in E \Rightarrow u \notin S \vee v \notin S$$

Definizione 2.1.2. Grafo complementare

Sia $G = (V, E)$ un grafo, il grafo complementare di G è il grafo $\overline{G} = (V, \overline{E})$, dove $\overline{E} = \{ (u, v) \mid u, v \notin E \text{ e } u \neq v \}$.

Osservazione 2.1.1. Relazione tra indipendent set e clique

Sia $G = (V, E)$ un grafo, $\overline{G} = (V, \overline{E})$ il grafo complementare di G e S un

sottoinsieme di V allora possiamo notare che:

$$S \text{ è un clique in } G \iff S \text{ è un indipendent set in } \overline{G}$$

Definizione 2.1.3. Colorazione

Sia $G = (V, E)$ un grafo, C un insieme di colori e $f : v \in V \to f(v) \in C$ una funzione che associa ad ogni vertice un colore.

La funzione f è un colorazione ammissibili se a nodo connessi con un edge assegna colori diversi, ovvero:

$$\forall (u, v) \in E \Rightarrow f(u) \neq f(v)$$

Osservazione 2.1.2. Sia $G = (V, E)$ un grafo e f una colorazione ammissibile. Una colorazione ammissibile ci permette di costruire una partizione di V tale che ogni insieme in tale partizione è un indipendent set.

Osservazione 2.1.3. Una colorazione ammissibile su $\overline{G}$ determina una partizione di V tale che ogni insieme in tale partizione individua una clique in G

Esempio 2.1.1. Consideriamo il grafo in figura 2.1 e il grafo complementare a esso associato in figura 2.2. Una possibile colorazione ammissibile sul grafo complementare è illustrata nella figura 2.3. Si può notare come la partizione di V individuata dalla colorazione sul grafo G individua un cluster graph.

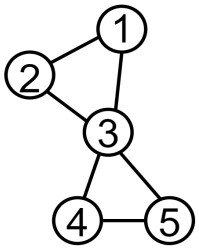

Figura 2.1: Grafo

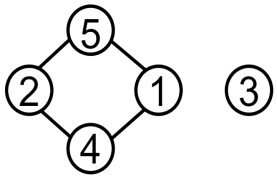

Figura 2.2: Grafo complementare

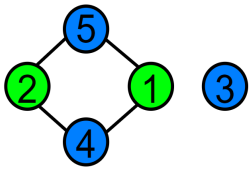

Figura 2.3: Colorazione ammissibile sul grafo complementare

2.2 Modello di programmazione lineare intera

Dalle osservazioni fatte segue che possiamo individuare la soluzione ottima per il Clustering Deletion determinando quella colorazione ammissibile sul grafo complementare tale che il numero di nodi colorati diversamente e connessi da un edge in G sia minimo.
Denotando con $n = |V|$ e $m = |E|$ scriviamo il modello di PLI per il Clustering Deletion. Come prima cosa stabiliamo quali sono le variabili decisionali. Poichè vogliamo modellare una colorazione che ci permette di individuare tutte le possibili partizioni possiamo utilizzare le variabili x_{ik}, per $i, k = 1, \ldots, n$:

$$x_{ik} = \begin{cases} 1 & \text{se al nodo } i \text{ viene assegnato il colore } k \\ 0 & \text{altrimenti} \end{cases} \tag{2.1}$$

Poichè vogliamo anche determinare gli edge da eliminare al fine di avere un cluster graph introduciamo anche le variabili y_{ij}, con $i, j = 1, \ldots, n$:

$$y_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \text{ e al nodo } i \text{ e al nodo } j \text{ vengono assegnati colori diversi} \\ 0 & \text{altrimenti} \end{cases} \tag{2.2}$$

Il modello è il seguente 2.3:

$$\underset{x_{ik},\, y_{ij}}{\text{minimize}} \quad \sum_{(i,j)\in E} y_{ij} \tag{F}$$

$$\begin{aligned}
\text{subject to} \quad & x_{ik} + x_{jk} \leq 1, && \forall (i,j) \in \overline{E} && \text{(C1)}\\
& \sum_{k=1}^{n} x_{ik} = 1, && \forall i \in V && \text{(C2)}\\
& x_{ik} - x_{jk} \leq y_{ij}, && \forall (i,j) \in E, \forall k && \text{(C3)}\\
& x_{jk} - x_{ik} \leq y_{ij}, && \forall (i,j) \in E, \forall k && \text{(C4)}\\
& y_{ij} \leq x_{ik} + x_{jk}, && \forall (i,j) \in E, \forall k && \text{(C5)}\\
& y_{ij} \leq 2 - x_{ik} - x_{jk}, && \forall (i,j) \in E, \forall k && \text{(C6)}\\
& x_{ik} \in \{\, 0, 1 \,\}, && \forall i, k && \text{(C7)}\\
& y_{ij} \in \{\, 0, 1 \,\}, && \forall i, j && \text{(C8)}
\end{aligned}$$

La funzione obiettivo (F) codifica opportunamente il fatto che tra i possibili cluster graph, ovvero tutte le possibili partizioni che individuano un insieme disgiunto di clique, vogliamo quella che ha il minor numero di edge tra clique diverse.

Passiamo ora alla descrizione dei vincoli. Occorre innanzitutto codificare opportunamente il fatto che le variabili x_{ik} debbano individuare solo cluster graph, o equivalentemente per le osservazioni fatte precedentemente una colorazione ammissibile sul grafo complementare. Ciò è fatto dal set di vincoli (C1) e (C2). I quali più nello specifico codificano rispettivamente il fatto che se esiste un edge nel grafo complementare tra due nodi i e j allora essi devono essere colorati diversamente e che a ogni nodo venga assegnato un colore. Dopodichè occorre codificare la consisteza tra le variabili x_{ik},x_{jk} e y_{ij}. Per fare ciò possiamo notare che fondamentalmente la variabile y_{ij} assume come valori lo XOR delle variabili x_{ik},x_{jk}. Ovvero vale 1 se e solo se le variabili x_{ik},x_{jk} assumono valori diversi e 0 altrimenti. I vincoli (C3),(C4),(C5) e (C6), codificano opportunamente quanto richiesto. La tabella 2.1 rende chiaro perchè ciò è vero. Essa per ogni possibile valore di x_{ik}, x_{jk} indica i valori che può assumere y_{ij} secondo ogni vincolo. Inoltre nell'ultima colonna sono riportati i

valori che può assumere y_{ij} se vengono utilizzati tutti e quattro i vincoli e si può notare come questi sono di fatto lo XOR tra le variabili x_{ik} e x_{jk}.

x_{ik}	x_{jk}	y_{ij}	(C3)	(C4)	(C5)	(C6)	(C3),(C4),(C5),(C6)
0	0	0	0,1	0,1	0	0,1	0
0	1	1	0,1	1	0,1	0,1	1
1	0	1	1	0,1	0,1	0,1	1
1	1	0	0,1	0,1	0,1	0	0

Tabella 2.1: Tabella che mostra come i vincoli (C3),(C4),(C5),(C6) computano correttamente lo XOR

Esempio 2.2.1. Se consideriamo il grafo dell'esempio 2.1.1 gli assegnamenti alle variabili decisionali che lo codificano sarebbero (considerando che al colore blu è associato il primo colore e al colore verde il secondo):

i \ k	1	2	3	4	5
1	0	1	0	0	0
2	0	1	0	0	0
3	1	0	0	0	0
4	1	0	0	0	0
5	1	0	0	0	0

Tabella 2.2: Tabella delle variabili x_{ik}

i \ j	1	2	3	4	5
1	0	0	1	0	0
2	0	0	1	0	0
3	1	1	0	0	0
4	0	0	0	0	0
5	0	0	0	0	0

Tabella 2.3: Tabella delle variabili y_{ij}

Possiamo notare come tali variabili rispettano i vincoli e che il valore della funzione obiettivo è 2.

CAPITOLO 3

RFD VALIDATION CLUSTERING BASED

In questo capitolo verranno date delle motivazioni pratiche all'interesse nella risoluzione del problema del clustering deletion su grafi lineari. Nella sezione 3.1 vengono date le definizioni utilizzate nel capitolo. Nelle sezioni 3.2, 3.3 e 3.4 viene studiato il problema del RFD validation clustering based. Nella sezione 3.5 viene descritto come i risultati ottenuti nella tesi, possono migliorare la bontà dei risultati ottenibili dall'algoritmo di validazione delle RFD proposto.

3.1 Nozioni preliminari

In questa sezione sono riportati brevemente alcuni concetti e notazioni utilizzate in seguito. In particolare, in 3.1.1 sono discusse brevemente le dipendenze funzionali, la loro generalizzazione e il problema del discovery e della validazione nella loro forma rilassata; in 3.1.2 una breve introduzione al clustering; in 3.1.4 è riportata la definizione di grafo delle distanze.

3.1.1 Dipendenze funzionali

In questa sezione, prima di dare la definizione di dipendenza funzionale nella forma canonica, è presentata qualche nozione propedeutica alla definizione di dipendenza funzionale. Successivamente, è presentata la definizione di RFD, un rilassamento delle FD *canoniche* e il problema del discovery e della validazione delle RFD.

Schema di relazione

Uno schema di relazione R è costituito da un insieme di attributi $X = A_1, A_2...A_n$ ed è indicato con $R(X)$. A ciascun attributo $A \in X$ è associato un dominio $dom(A)$. Un'istanza di relazione r su $R(X)$ è un insieme di tuple su X. Per ogni istanza $r \in R(X)$, per ogni tupla $t \in r$ e per ogni attributo $A \in X$, $t[A]$ rappresenta la proiezione di t su A [4].

Dipendenze funzionali canoniche

Una dipendenza funzionale, abbreviata FD, di uno schema relazionale R, definita su un insieme di attributi $attr(R)$ è una dichiarazione $X \rightarrow Y$ con $X, Y \subseteq attr(R)$, tale che, data un'istanza r di R, $X \rightarrow Y$ è soddisfatta in r se e solo se per ogni coppia di tuple (t_1, t_2) in r, ogni volta che $t_1[x] = t_2[x]$, allora $t_1[y] = t_2[y]$ [5].

Dipendenze funzionali rilassate

Una dipendenza funzionale rilassata, abbreviata RFD, generalizza il paradigma di confronto delle FD per includere confronti basati sulla somiglianza, e ammettendo la possibilità che la dipendenza valga solo per un sottoinsieme di dati.

Formalmente, dato uno schema di database relazionale D, e $R = (A_1, ..., A_m)$ uno dei suoi schemi di relazione, una RFD ϱ su D è denotata da

$D : X_{\Phi_1} \xrightarrow{\Psi \leq \varepsilon} Y_{\Phi_2}$

dove

- $X = X_1, ..., X_h$ e $Y = Y_1, ..., Y_k$, con $X, Y \subseteq attr(R)$;
- Φ_1 e Φ_2 sono insiemi di vincoli su X e Y;
- Ψ è la coverage measure definita su $dom(R)$ che quantifica il numero di tuple che violano o soddisfano ϱ. Le coverage measure più comunemente usate includono la confidenza, il g3-error e la probabilità;
- ε è una soglia che indica il limite superiore (o il limite inferiore nel caso in cui l'operatore di confronto sia $\geq$) per il risultato della coverage measure.

La semantica di una RFD afferma che un'istanza di relazione r di R soddisfa la RFD ϱ se e solo se $\forall t_1, t_2 \in r$, se Φ_1 indica vero allora *quasi sempre* Φ_2 indica vero, dove *quasi sempre* è espresso dal vincolo $\Psi \leq \varepsilon$ [5].

Discovery e validazione RFD

Il problema della validazione delle RFD consiste, dati un insieme di tuple D e una RFD ϱ, nel determinare se ϱ è valida nell'insieme D.

Algorithm 1: Validazione RFD

Input: D insieme di tuple, $X \rightarrow Y$ RFD da validare con le rispettive soglie

Output: SI se $X \rightarrow Y$ vale su D, NO se $X \rightarrow Y$ non vale su D

```
1: for all (t1, t2) con t1 ∈ D, t2 ∈ D do
2:     if t1 e t2 sono simili rispetto LHS and non simili rispetto RHS then
3:         return NO
4:     end if
5: end for
```

3.1.2 Clustering

Il clustering è il processo in cui si esaminano una collezione di *punti*, e si raggruppano questi punti in *cluster* in base a una funzione di distanza. L'obiettivo del clustering è quello di raggruppare nello stesso cluster punti che hanno una piccola distanza l'uno dall'altro, mentre posizionare in cluster diversi punti che hanno una grande distanza l'uno dall'altro.

3.1.3 Funzioni di distanza e similarità

Una funzione di distanza $d(x, y)$ è una funzione che prende in input due punti e da in output un numero reale. Una funzione di distanza soddisfa i seguenti assiomi:

1. $d(x, y) \geq 0$ (nessuna distanza è negativa)
2. $d(x, y) = 0 \iff x = y$ (le distanze sono positive, tranne la distanza da un punto a se stesso)
3. $d(x, y) = d(y, x)$ (le distanze sono simmetriche)
4. $d(x, y) \leq d(x, z) + d(z, y)$ (disuguaglianza triangolare)

Tra le funzioni di distanza più famose abbiamo la distanza Euclidea, la distanza di Jaccard, la distanza di Edit, la distanza di Hamming e la distanza Coseno.

Una funzione di similarità $s(x, y)$ è una funzione che prende in input due punti e da in output un valore che ne quantifica la similarità. Una funzione di similarità soddisfa le seguenti condizioni:

1. $s(x, x) = 1$
2. $0 \leq s(x, y) \leq 1$
3. $s(x, y) = s(x, y)$ *per ogni* x *e* y

Talvolta, congiuntamente a tali funzioni, vengono considerate delle soglie che indicano quanto due punti sono sufficientemente vicini o simili. Il valore

delle soglie assume significati differenti con le funzioni di similarità o di distanza. Quando si parla di soglie con funzioni di similarità si intende un lower-bound sopra il quale due punti sono considerati simili; invece, quando si parla di soglie con funzioni di distanza si intende un upper-bound sotto il quale due punti sono considerati vicini. Senza perdita di generalità in questa trattazione prenderemo in considerazione funzioni di distanza, usando i termini *vicino* e *simile* in modo interscambiabile.

Algoritmi di clustering

Si possono dividere gli algoritmi di clustering in due grandi categorie che seguono fondamentalmente due strategie.

1. Algoritmi *gerarchici* o *agglomerativi* che iniziano ognuno con un punto nel proprio cluster. I cluster sono poi combinati tra loro in base alla vicinanza, usando una funzione di distanza o di similarità. La combinazione dei cluster si interrompe quando un'ulteriore combinazione porta ad avere cluster non desiderati per qualche motivo.

2. Algoritmi ad *assegnamento di punti*; i punti sono considerati in un certo ordine e ciascuno è assegnato al cluster al quale si adatta meglio. Questo processo è normalmente preceduto da una breve fase in cui sono stimati i cluster iniziali. La famiglia più nota di algoritmi di clustering di questo tipo è chiamata k-means.

Online clustering

Generalmente gli algoritmi di clustering estraggono i dati da un database; i dati quindi sono sempre disponibili. Nel *Stream Data Model* si presuppone che i dati arrivino in uno o più flussi e se non vengono elaborati o memorizzati immediatamente sono persi per sempre. Inoltre, questo modello suppone che i dati arrivino così rapidamente che non è possibile archiviarli in una memoria attiva (un database tradizionale), e quindi interagire con essi al momento del bisogno. Una strategia per gestire gli stream consiste nel mantenere una

sommarizzazione degli stream, sufficiente a rispondere alla domande che ci si aspetta vengano fatte sui dati. Un secondo approccio consiste nel mantenere una finestra scorrevole dei dati arrivati più di recente.

Nel problema dell'online clustering un algoritmo di clustering deve assegnare un nuovo punto che arriva a uno dei k cluster. Per fare questo l'algoritmo ad ogni passo temporale deve mantenere un insieme di k candidati, in quanto non può memorizzare tutti i dati che vede poiché sono potenzialmente infiniti. Il clustering dei nuovi punti deve essere vicino al cluster ottimale ottenuto da tutti i dati visti fino a quel punto. [6].

3.1.4 Grafo delle distanze

Sia D un insieme di item, f una funzione di distanza su $dom(D)$ e ϵ la soglia ad essa associata. Allora un grafo pesato $G = (V, E)$ con:

- $V = D$
- $E = \{(t_i, t_j) \mid f(t_i, t_j) \leq \epsilon, \text{ con } t_i, t_j \in D\}$
- $c(e) = f(e)\ \forall e \in E$

è detto grafo delle distanze dell'insieme D.

3.2 Validazione RFD su clustering

In questa sezione è descritto un possibile approccio alla validazione di RFD basato su clustering. Nella sottosezione 3.2.1 una descrizione ad alto livello dell'approccio RFD basato su clustering. Nella sottosezione 3.2.2 sono discusse e analizzate le proprietà desiderabili per i cluster. Nelle sottosezioni 3.2.3 e 3.2.4 sono stati modellati rispettivamente i problemi $P_1 max P_2$ e $P_2 max P_1$. Nella sottosezione 3.2.5 è presentato l'algoritmo offline.

3.2.1 Approccio clustering RFD

L'obiettivo del clustering è raggruppare un insieme di oggetti in base ad una funzione di distanza, ovvero creare gruppi di oggetti con la caratteristica che oggetti nello stesso gruppo sono molto vicini tra loro e oggetti in gruppi diversi sono molto distanti tra loro. Nel nostro caso, il clustering è utilizzato al fine di permettere la sommarizzazione di oggetti su uno stream di dati.

Le RFD sul confronto utilizzano funzioni di distanza per determinare se due tuple, fissata una soglia, sono simili rispetto a un determinato attributo. Definiamo $F_{LHS}(t)$ e $F_{RHS}(t)$ due funzioni di clustering che rispettivamente, data una tupla t, restituiscono i cluster rispetto LHS e RHS di t, $\forall t \in D$, con D insieme di tuple.

Algorithm 2: Validazione RFD su clustering

Input:

- Insieme di tuple D
- Funzioni di distanza rispetto LHS
- Soglie rispetto LHS
- Funzione di distanza rispetto RHS
- Soglia rispetto RHS

Output: $True$ se la RFD vale, $False$ altrimenti

1: Clustering rispetto LHS
2: Clustering rispetto RHS
3: **if** $\exists(t_i, t_j)$ con $t_i, t_j \in D$ tc $F_{LHS}(t_i) = F_{LHS}(t_j)$, $F_{RHS}(t_i) \neq F_{RHS}(t_j)$ **then**
4: **return** False
5: **else**
6: **return** True
7: **end if**

Dati i due cluster rispetto LHS e RHS, per ogni coppia di tuple t_1, t_2 abbiamo uno dei seguenti 4 casi:

1. $F_{LHS}(t_1) = F_{LHS}(t_2)$ e $F_{RHS}(t_1) = F_{RHS}(t_2)$.
2. $F_{LHS}(t_1) = F_{LHS}(t_2)$ e $F_{RHS}(t_1) \neq F_{RHS}(t_2)$.
3. $F_{LHS}(t_1) \neq F_{LHS}(t_2)$ e $F_{RHS}(t_1) = F_{RHS}(t_2)$.
4. $F_{LHS}(t_1) \neq F_{LHS}(t_2)$ e $F_{RHS}(t_1) \neq F_{RHS}(t_2)$.

Un possibile approccio al problema della validazione delle RFD basato su clustering è descritto dall'algoritmo 2.

3.2.2 Proprietà desiderabili su LHS e RHS

Proprietà *P*

Un algoritmo di validazione ideale, data una RFD, determina con precisione 1 sia le RFD valide sia le RFD non valide, come mostrato nella tabella 3.1.

Reale \ Predetta	valida	non valida
valida	1	0
non valida	0	1

Tabella 3.1: Precisione validazione RFD ottimale

Per poter raggiungere la precisione ottimale nella validazione delle RFD, è necessario determinare con *precisione 1* le coppie di tuple simili e non simili, rispetto LHS e RHS. Il nostro algoritmo individua che due tuple sono:

- simili rispetto LHS (o RHS) se sono nello stesso cluster rispetto LHS (o RHS),
- non simili rispetto LHS (o RHS) se sono in cluster diversi rispetto LHS (o RHS).

Definizione 3.2.1. Proprietà P

Siano t_1, t_2 due tuple appartenenti ad un dataset. Definiamo la proprietà P su un insieme di cluster C come:

$$P(C) : \text{"}\ \forall t_1, t_2 \in D,\ t_1, t_2 \text{ simili } \iff t_1, t_2 \text{ stesso cluster rispetto } C\text{"} \quad (3.1)$$

Se potessimo sempre avere un cluster sia sul LHS che sul RHS che rispetta la proprietà P allora potremmo determinare con *precisione 1* le coppie simili e non simili sia sul LHS che sul RHS, come mostrato nella tabella 3.2. Di conseguenza potremmo determinare con *precisione 1* sia le RFD valide che le non valide. Tuttavia, non è sempre possibile determinare un cluster che soddisfa la proprietà P.

Reale \ Predetta	simili	non simili
simili	1	0
non simili	0	1

Tabella 3.2: Precisione similarità proprietà P

Proposizione 3.2.1. Esiste un insieme D di tuple per cui non vale la proprietà P.

	X
t_1	1
t_2	2
t_3	3
t_4	5
t_5	7
t_6	8
t_7	9

Tabella 3.3: Insieme tuple D

Dimostrazione. Sia $D = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$ il datatset riportato nella tabella 3.3 e f la funzione di distanza valore assoluto sull'attributo X con soglia 2. Notiamo che la proprietà P può essere riscritta in maniera del tutto equivalente nel seguente modo: t_1, t_2 *simili* $\Rightarrow$ t_1, t_2 stesso cluster $\wedge$ t_1, t_2 stesso cluster $\Rightarrow$ t_1, t_2 *simili*. L'unico insieme di cluster che raggruppa ogni coppia di tuple simili nello stesso cluster è $C = \{\{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}\}$. Tuttavia, C non soddisfa P, in quanto non è vero che ogni coppia di tuple nello stesso cluster è una coppia di tuple simili. Ad esempio, t_1 e t_4 sono nello stesso cluster anche se $f(t_1[X], t_4[X]) = 4$. □

Definizione 3.2.2. (Relazione di similarità definita da una funzione di distanza)
Sia f una funzione di distanza ed ϵ la soglia ad essa associata, e sia R_s una relazione tra tuple così definita:

$$t_1 \; R_s \; t_2 \iff t_1, t_2 \textit{ sono simili, ovvero } f(t_1, t_2) \leq \epsilon \tag{3.2}$$

Proposizione 3.2.2. La relazione R_s non è una relazione di equivalenza.

Dimostrazione. Per provare che la relazione R_s non è una relazione di equivalenza occorre provare che non soddisfa almeno una delle tre proprietà di una relazione di equivalenza. La relazione R_s non soddisfa la proprietà transitiva, ovvero esistono t_1, t_2 e t_3 tuple tali che $t_1 \; R_s \; t_2$ e $t_2 \; R_s \; t_3$ ma $t_1 \not R_s \; t_3$. Ad esempio, se consideriamo le 3 tuple della tabella 3.4 e la funzione di distanza valore assoluto con soglia 2 abbiamo che $t_1 \; R_s \; t_2$ e $t_2 \; R_s \; t_3$ ma $t_1 \not R_s \; t_3$.

	X
t_1	1
t_2	3
t_3	5

Tabella 3.4: Controesempio proposizione 3.2.2

□

Notiamo che la proprietà P non è sempre raggiungibile perché la relazione R_s non è una relazione di equivalenza.

Proprietà P_1 e P_2

Proprietà P_1 e P_2 su cluster

Data l'impossibilità di avere sempre un insieme di cluster che soddisfa la proprietà P, deriviamo due proprietà P_1 e P_2 da P su insiemi di cluster C.

Definizione 3.2.3. Siano $P_1(C)$ e $P_2(C)$ i seguenti predicati su insiemi di cluster:

$$P_1(C) : \text{" } \forall t_1, t_2 \in D,\ t_1, t_2 \textit{ simili } \Rightarrow t_1, t_2 \textit{ stesso cluster rispetto } C \text{"} \quad (3.3)$$

$$P_2(C) : \text{" } \forall t_1, t_2 \in D,\ t_1, t_2 \textit{ stesso cluster rispetto } C \Rightarrow\ t_1, t_2 \textit{ simili } \text{"} \quad (3.4)$$

Proposizione 3.2.3. Dato un insieme D di tuple è sempre possibile trovare un cluster che soddisfa P_1.

Dimostrazione. Dato un generico insieme D di tuple, l'insieme di cluster $C = \{D\}$ può essere sempre costruito e soddisfa P_1, perché per ogni coppia di tuple t_1, t_2, se esse sono simili allora sono nello stesso cluster. □

Proposizione 3.2.4. Dato un insieme D di tuple è sempre possibile trovare un cluster che soddisfa P_2.

Dimostrazione. Dato un generico insieme D di tuple, l'insieme di cluster $C = \{\{t\} \mid \forall t \in D\}$ in cui ogni cluster è un singleton di una tupla può essere sempre costruito e soddisfa P_2, perché non è possibile trovare due tuple in uno stesso cluster che non sono simili. □

Proprietà tra tuple

Definiamo le seguenti proprietà tra tuple.

Definizione 3.2.4. Sia f una funzione di distanza ed ϵ la soglia ad essa associata. S_{f_ϵ} è un predicato tra tuple tale che:

$$S_{f_\epsilon}(t_1, t_2) : \text{" } t_1, t_2 \textit{ simili, ovvero } f(t_1, t_2) \leq \epsilon \text{"} \tag{3.5}$$

Definizione 3.2.5. Sia F una funzione di clustering. K_F è un predicato tra tuple tale che:

$$K_F(t_1, t_2) : \text{" } F(t_1) = F(t_2) \text{"} \tag{3.6}$$

Proprietà P_1 e P_2 tra tuple

Notiamo che le proprietà P_1 e P_2 sono proprietà su cluster definite in termini delle proprietà S_{f_ϵ} e K_F tra tuple, ovvero dato un cluster su un insieme di tuple, considerando tutte le coppie di tuple possiamo determinare se le proprietà sono soddisfatte o meno.

Definizione 3.2.6. Siano $P_1(t_i, t_j)$ e $P_2(t_i, t_j)$ i seguenti predicati su coppie di tuple:

$$P_1(t_i, t_j) : \text{" } t_i, t_j \textit{ simili } \Rightarrow t_i, t_j \textit{ stesso cluster } \text{"} \tag{3.7}$$

$$P_2(t_i, t_j) : \text{" } t_i, t_j \textit{ stesso cluster } \Rightarrow \textit{ } t_i, t_j \textit{ simili } \text{"} \tag{3.8}$$

I predicati su cluster $P_1(C)$ e $P_2(C)$ possono essere espressi in funzione dei predicati $P_1(t_i, t_j)$ e $P_2(t_i, t_j)$ nel seguente modo:

$$P_1(C) : \text{" } \forall t_i, t_j \in D, P_1(t_i, t_j) \text{"} \tag{3.9}$$

$$P_2(C) : \text{" } \forall t_i, t_j \in D, P_2(t_i, t_j) \text{"} \tag{3.10}$$

Osservazione 3.2.1. Una proposizione $p \Rightarrow q$ è equivalente alla proposizione $\neg p \vee q$. Di conseguenza possiamo riformulare i predicati su tuple $P_1(t_i, t_j)$ e $P_2(t_i, t_j)$ nel seguente modo del tutto equivalente:

$$P_1(t_i, t_j) : \text{" } t_i, t_j \textit{ non simili } \vee\, t_i, t_j \textit{ stesso cluster"} \tag{3.11}$$

$$P_2(t_i, t_j) : \text{" } t_i, t_j \textit{ cluster diversi } \vee\, t_i, t_j \textit{ simili"} \tag{3.12}$$

Dato una generica coppia di tuple t_i, t_j e F una funzione di clustering su D esplicitiamo i possibili output dei predicati $P_1(t_i, t_j)$ e $P_2(t_i, t_j)$:

$$P_1(t_i, t_j) = \begin{cases} 1 & \text{if } S_{f_\epsilon}(t_i, t_j) \wedge K_F(t_i, t_j) \\ 0 & \text{if } S_{f_\epsilon}(t_i, t_j) \wedge \neg K_F(t_i, t_j) \\ 1 & \text{if } \neg S_{f_\epsilon}(t_i, t_j) \wedge K_F(t_i, t_j) \\ 1 & \text{if } \neg S_{f_\epsilon}(t_i, t_j) \wedge \neg K_F(t_i, t_j) \end{cases} \tag{3.13}$$

$$P_2(t_i, t_j) = \begin{cases} 1 & \text{if } S_{f_\epsilon}(t_i, t_j) \wedge K_F(t_i, t_j) \\ 1 & \text{if } S_{f_\epsilon}(t_i, t_j) \wedge \neg K_F(t_i, t_j) \\ 0 & \text{if } \neg S_{f_\epsilon}(t_i, t_j) \wedge K_F(t_i, t_j) \\ 1 & \text{if } \neg S_{f_\epsilon}(t_i, t_j) \wedge \neg K_F(t_i, t_j) \end{cases} \tag{3.14}$$

Con α_1 e α_2 definiamo rispettivamente i gradi di verità delle proprietà P_1 e P_2.

$$\alpha_1 = \frac{\sum_{t_i, t_j \in D} P_1(t_i, t_j)}{\binom{|D|}{2}} \tag{3.15}$$

$$\alpha_2 = \frac{\sum_{t_i, t_j \in D} P_2(t_i, t_j)}{\binom{|D|}{2}} \tag{3.16}$$

Nelle proposizioni 3.2.5 e 3.2.6, analizziamo a cosa si va in contro se un insieme di cluster soddisfa la proprietà P_1 o la proprietà P_2.

Proposizione 3.2.5. Se l'insieme di cluster soddisfa la proprietà P_1 allora l'algoritmo è capace di trovare con precisione 1 le tuple simili e con precisione α_2 le tuple non simili, come riportato nella tabella 3.5.

Reale \ Predetta	simili	non simili
simili	1	0
non simili	$1 - \alpha_2$	α_2

Tabella 3.5: Precisione similarità proprietà P_1

Dimostrazione.
Precisione 1 tuple simili. Siano C un insieme di cluster che soddisfa la proprietà P_1, $H_s = \{(t_i, t_j) \mid S_{f_\epsilon}(t_1, t_2) = 1, t_i, t_j \in D\}$ e $W_s = \{(t_i, t_j) \mid K_F(t_1, t_2) = 1, t_i, t_j \in D\}$. Notiamo che l'insieme W_s è l'insieme delle coppie di tuple considerate simili dall'algoritmo. Siccome C soddisfa la proprietà P_1 abbiamo che per ogni coppia di tuple $t_i, t_j \in D$ se $S_{f_\epsilon}(t_1, t_2) = 1$ allora $K_F(t_1, t_2) = 1$ quindi risulta $H_s \subseteq W_s$. Di conseguenza tutte le coppie di tuple simili nell'insieme D di tuple vengono individuate come simili dall'algoritmo.
Precisione α_2 tuple non simili. L'algoritmo individua con precisione α_2 le tuple non simili perché date due tuple non simili t_1, t_2 esse vengono considerate non simili quando sono in cluster diversi. Ciò accade per quelle tuple per cui $P_2(t_1, t_2) = 1$, poiché la frazione di tuple per cui P_2 è vera è α_2, l'asserto è vero. □

Proposizione 3.2.6. Se l'insieme di cluster soddisfa la proprietà P_2 allora l'algoritmo è capace di trovare con precisione 1 le tuple non simili e con precisione α_1 le tuple simili, come riportato nella tabella 3.6.

Reale \ Predetta	simili	non simili
simili	α_1	1-α_1
non simili	0	1

Tabella 3.6: Precisione similarità proprietà P_2

Dimostrazione.

Precisione 1 tuple non simili. Siano C un insieme di cluster che soddisfa la proprietà P_2, $H_{ns} = \{(t_i, t_j) \mid S_{f_\epsilon}(t_1, t_2) = 0, t_i, t_j \in D\}$ e $W_{ns} = \{(t_i, t_j) \mid K_F(t_1, t_2) = 0, t_i, t_j \in D\}$. Notiamo che l'insieme W_{ns} è l'insieme delle coppie di tuple considerate non simili dall'algoritmo. Siccome C soddisfa la proprietà P_2 abbiamo che per ogni coppia di tuple $t_i, t_j \in D$ se $K_F(t_1, t_2) = 1$ allora $S_{f_\epsilon}(t_1, t_2) = 1$, considerando il contronominale di tale proposizione ovvero $S_{f_\epsilon}(t_1, t_2) = 0$ allora $K_F(t_1, t_2) = 0$ abbiamo che $H_{ns} \subseteq W_{ns}$. Di conseguenza tutte le coppie di tuple non simili nell'insieme D di tuple vengono individuate come non simili dall'algoritmo.

Precisione α_1 tuple simili. L'algoritmo individua con precisione $\alpha 1$ le tuple simili perché date due tuple simili t_1, t_2 esse vengono considerate simili quando sono nello stesso cluster. Ciò accade per quelle tuple per cui $P_1(t_1, t_2) = 1$, poiché la frazione di tuple per cui P_1 è vera è α_1, l'asserto è vero. □

Trade-off su falsi positivi e falsi negativi

Dal momento che non è sempre possibile avere un insieme di cluster che soddisfa la proprietà P, è necessario un compromesso su falsi positivi e falsi negativi. Vorremmo che i falsi positivi individuati dall'algoritmo fossero 0 mentre i falsi negativi il minimo possibile.

Teorema 3.2.1. Precisione algoritmo di validazione

Siano:

- D insieme di tuple,
- $X \to Y$ una RFD su D,

- C_X l'insieme di cluster sul LHS individuato dall'algoritmo 2,
- C_Y l'insieme di cluster sul RHS individuato dall'algoritmo 2.

Se C_X e C_Y soddisfano rispettivamente P_1 e P_2 allora l'algoritmo 2 valida una RFD $X \to Y$ su D con le precisioni riportate nella tabella 3.7.

Reale \ Predetta	valida	non valida
valida	k	$1-k$
non valida	0	1

Tabella 3.7: Precisione validazione RFD con proprietà P_1 su X e P_2 su Y

Dimostrazione.
L'algoritmo 2 riesce ad avere 0 falsi positivi se e solo se ogni RFD $X \to Y$ che non vale su D è validata correttamente. Proviamo che l'algoritmo 2 ha 0 falsi positivi se utilizza C_X e C_Y.

Definiamo i seguenti insiemi:

- T l'insieme di RFD valide
- W l'insieme di RFD valide determinate dall'algoritmo
- $H_{s,X}$ l'insieme delle coppie di tuple simili sul LHS
- $W_{s,X}$ l'insieme delle coppie di tuple simili determinate dall'algoritmo
- $H_{ns,Y}$ l'insieme delle coppie di tuple non simili sul RHS
- $W_{ns,Y}$ l'insieme delle coppie di tuple non simili determinate dall'algoritmo

Se una RFD $X \to Y$ non vale allora esiste una coppia di tuple simili sul LHS e non simili sul RHS, ovvero esiste una coppia $t_i, t_j \in H_{s,X} \cap H_{ns,Y}$. Se una tale coppia di tuple esiste, per garantire che venga sempre individuata dall'algoritmo 2 è necessario che per ogni $X \to Y$ si abbia $H_{s,X} \cap H_{ns,Y} \subseteq W_{s,X} \cap W_{ns,Y}$.

Ricapitolando se una RFD non vale in D allora esiste una coppia di tuple t_i e t_j simili sul LHS e dissimili sul RHS, tale coppia di tuple per definizione apparterrà all'insieme $H_{s,X} \cap H_{ns,Y}$ di conseguenza apparterrà anche all'insieme $W_{s,X} \cap W_{ns,Y}$, quindi sicuramente verrà individuata correttamente dall'algoritmo se l'algoritmo utilizza C_X e C_Y per via delle proposizioni 3.2.5 e 3.2.6. □

3.2.3 Modellazione del problema $P_1 max P_2$

Definiamo il problema $P_1 max P_2$ come segue:

Problema $P_1 max P_2$

Input:

- Grafo delle distanze $G = (V, E)$ di un insieme D di tuple;

Output: $C = \{C_1, ..., C_k\}$ tale che:

- $C_q \subseteq V$, con $q = 1, ..., k$;
- C soddisfa la proprietà P_1: $\forall\ v_i, v_j \in V$, se v_i, v_j sono simili allora sono nello stesso cluster;
- Il numero di coppie di nodi non simili nello stesso cluster sia il minimo possibile.

Possiamo modellare il problema $P_1 max P_2$ attraverso un grafo delle distanze dell'insieme D delle tuple.

Esempio 3.2.1. Dato un dataset $D = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$, riportato nella tabella 3.8, e una soglia di similarità $\beta = 1$. Rappresentando il problema su grafi otterremo i due cluster $C_1 = \{t_1, t_2, t_3, t_4\}, C_2 = \{t_5, t_6, t_7\}$ riportati nella figura 3.1.

	X
t_1	2
t_2	3
t_3	4
t_4	5
t_5	8
t_6	9
t_7	10

Tabella 3.8: Dataset

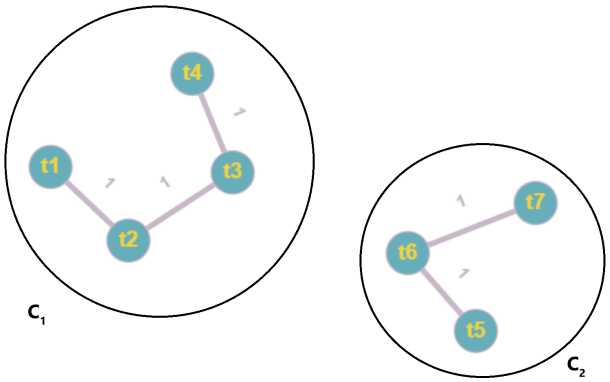

Figura 3.1: Grafo delle distanze $P_1 max P_2$ di D e con i derivanti cluster

Definizione 3.2.7 (Relazione di similarità indiretta)**.** Sia R_{si} la seguente relazione definita sull'insieme D di tuple:

$$tR_{si}q \iff \exists t_{i_1}, t_{i_2}, ..., t_{i_j} \text{ tc} \quad \begin{array}{l} \text{t simile a } t_{i_1}, \\ t_{i_1} \text{ simile a } t_{i_2}, \\ ..., \\ t_{i_j} \text{ simile a q,} \end{array}$$

che chiameremo *relazione di similarità indiretta.*

Proposizione 3.2.7. La relazione R_{si} è una relazione di equivalenza:

Dimostrazione. Per provare che la relazione R_{si} è una relazione di equivalenza dobbiamo provare che soddisfa le proprietà di una relazione di equivalenza.

1. Riflessiva: $t_1 R_{si} t_1$, perché chiaramente t_1 è simile a t_1;
2. Simmetrica: $t_1 R_{si} t_2$ allora $t_2 R_{si} t_1$, perché la similarità è simmetrica;
3. Transitività: $t_1 R_{si} t_2$ e $t_2 R_{si} t_3$ allora $t_1 R_{si} t_3$:
   - per $t_1 R_{si} t_2$ abbiamo che $\exists t_{i_1}, t_{i_2}, ..., t_{i_j}$ tali che t_1 simile a t_{i_1}, t_{i_1} simile a $t_{i_2}, ..., t_{i_j}$ simile a t_2;
   - per $t_2 R_{si} t_3$ abbiamo che $\exists t_{r_1}, t_{r_2}, ..., t_{r_c}$ tali che t_2 simile a t_{r_1}, t_{r_1} simile a $t_{r_2}, ..., t_{r_c}$ simile a t_3.

   Poiché t_2 è simile a se stesso allora abbiamo che $\exists t_{i_1}, t_{i_2}, ..., t_{i_j}, t_2, t_{r_1}, t_{r_2}, ..., t_{r_c}$ tali che:
   - t_1 simile a t_{i_1}, t_{i_1} simile a $t_{i_2}, ..., t_{i_j}$ simile a t_2,
   - t_2 simile a t_{r_1}, t_{r_1} simile a $t_{r_2}, ..., t_{r_c}$ simile a t_3,

   e quindi $t_1 R t_3$.

□

Notiamo che t_i R_{si} t_j in termini di grafo delle distanze è equivalente a dire che esiste un cammino tra i nodi corrispondenti alle tuple t_i e t_j. Di conseguenza la classe di equivalenza di una tupla t è uguale alla componente connessa del nodo corrispondente alla tupla t nel grafo delle distanze.

Algoritmo $P_1 max P_2$

Un possibile algoritmo risolutivo per il problema $P_1 max P_2$ è riportato in 3.

Algorithm 3: Algoritmo $P_1 max P_2$

Input: Grafo $G = (V, E)$

Output: Cluster C che soddisfa P_1 e massimizza P_2

1: $C \leftarrow \{C_i \mid C_i$ è una componente connessa di $G\}$

2: **return** C

L'algoritmo 3 può essere implementato con un qualsiasi algoritmo di visita in ampiezza o in profondità su grafi. La complessità è $\mathcal{O}(n^2)$, dove n è numero di nodi.

Dal momento che la partizione definita da R_{si} corrisponde all'insieme delle componenti connesse del grafo delle distanze, la proposizione 3.2.8 prova la correttezza dell'algoritmo 3.

Proposizione 3.2.8. C insieme di cluster ottimo $P_1 max P_2$ sse C partizione definita da R_{si}.

Dimostrazione. Dimostriamo le due implicazioni separatamente.
"⇒" Se C insieme di cluster ottimo $P_1 max P_2$ allora C è una partizione definita da R_{si}. Dimostriamo che ogni cluster di C è una classe di equivalenza rispetto la relazione mostrando che soddisfa le due proprietà di una classe di equivalenza di R_{si}:

- *Per ogni* t_1,$t_2 \in T$, *con* $T \in C$, t_1 R_{si} t_2*:*
  Se per assurdo così non fosse avremmo che esistono due tuple t_1 e t_2 tali che t_1 e t_2 non sono in relazione secondo R_{si}. Ciò vorrebbe dire che non sono simili né direttamente né indirettamente, quindi la coppia t_1 e t_2 sarebbe un coppia di tuple non simili nello stesso cluster, che non avendo legami indiretti, potrebbe essere divisa in modo da far rispettare sempre P_1 ma incrementare P_2. Questo è un assurdo perché per ipotesi C è il cluster ottimo $P_1 max P_2$.

- *Per ogni* $t_1 \in T$, $t_2 \in T'$ t_1 *non è in relazione* R_{si} *con* t_2*:*
  Se per assurdo così non fosse avremmo che esistono due tuple $t_1 \in T$ e $t_2 \in T'$ con t_1 R_{si} t_2, ovvero esistono $t_{i_1}, t_{i_2}, ..., t_{i_j}$ tali che t_1 è simile a t_{i_1},..., t_{i_j} è simile a t_2. Siccome t_1 e t_2 vengono messi in cluster diversi, la sequenza di tuple simili ad un certo punto verrà spezzata e avremmo che esiste una coppia di tuple simili in cluster diversi violando la proprietà P_1. Ciò porta ad una contraddizione perché C è il cluster ottimo $P_1 max P_2$.

"$\Leftarrow$" Se C è la partizione definita da R_{si} allora C è l'insieme di cluster ottimo $P_1 max P_2$. Dimostriamo che la partizione C soddisfa le proprietà di ottimalità del problema $P_1 max P_2$:

- *C soddisfa P_1:*

  Notiamo che se due tuple t_1 e t_2 sono simili allora t_1 R_{si} t_2, quindi siccome C è la partizione definita da R_{si} abbiamo che per ogni coppia di tuple se esse sono simili, e quindi sono anche in relazione secondo R_{si}, allora vengono messe nella stessa classe di equivalenza. Di conseguenza la proprietà P_1 è soddisfatta.

- *C massimizza P_2:*

  Notiamo che l'unico modo di aumentare P_2 è determinare una coppia di tuple non simili nello stesso cluster, dopodiché metterle in cluster diversi. Ciò può essere fatto senza violare P_1 solo se non vengono separate tuple simili. Se per assurdo esistesse una tale coppia di tuple vorrebbe dire che tali tuple in realtà non sono in relazione secondo R_{si} e che quindi la partizione C non sarebbe corretta, ma ciò è assurdo.

□

3.2.4 Modellazione del problema P_2maxP_1

Definiamo il problema P_2maxP_1 come segue.

Problema P_2maxP_1

Input:

- Grafo delle distanze $G = (V, E)$ di un insieme D di tuple;

Output: $C = \{C_1, ..., C_k\}$ tale che

- $C_q \subseteq V$, con $q = 1, ..., k$;
- C soddisfa la proprietà P_2: $\forall\ v_i,\ v_j \in C_q$ sono simili, con $q = 1, ..., k$;
- Il numero di coppie di nodi simili in cluster diversi sia il minimo possibile.

Possiamo modellare il problema P_2maxP_1 attraverso un grafo delle distanze dell'insieme D delle tuple.

	X
t_1	2
t_2	3
t_3	4
t_4	6
t_5	8
t_6	9
t_7	10

Tabella 3.9: Dataset

Esempio 3.2.2. Dato un dataset $D = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7\}$, riportato nella tabella 3.9, e una soglia di similarità $\beta = 2$. Rappresentando il problema

su grafi otterremo i tre cluster $C_1 = \{t_1, t_2, t_3\}$, $C_2 = \{t_4\}$, $C_3 = \{t_5, t_6, t_7\}$ riportati nella figura 3.2. Notiamo che le tuple t_3, t_4 e t_5 non formano un cluster, in quanto non costituiscono una clique e quindi non rispettano la proprietà P_2.

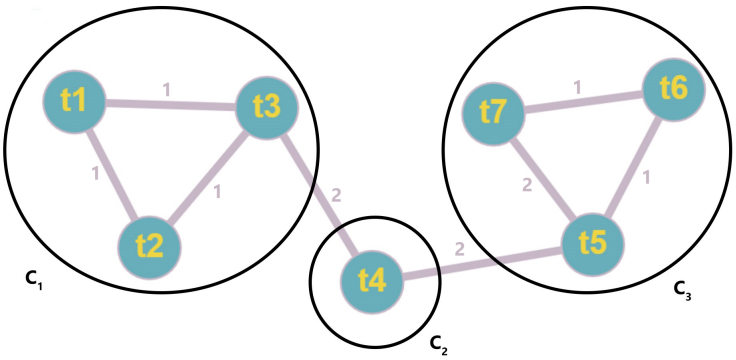

Figura 3.2: Grafo delle distanze P_2maxP_1 di D e con i derivanti cluster

Algoritmo P_2maxP_1

Il Graph Clustering si riferisce al clustering di dati sotto forma di grafi. Dato un grafo G, il problema del Graph Clustering può essere visto come il problema di determinare il minimo numero di modifiche da apportare a G affinché G sia un cluster graph. Un cluster graph, in letteratura [7], è un grafo formato da un insieme disgiunto di grafi completi, come quello mostrato nella figura 3.3.

Il Cluster Deletion (CD) è un caso particolare di Graph Clustering in cui sono ammesse solo cancellazioni di archi tra le possibili modifiche da apportare al grafo in input. Il problema P_2maxP_1 e il problema CD sono di fatto lo stesso problema. In particolare, ogni clique che compone il cluster graph individuato da CD è un cluster dell'insieme dei cluster individuato da P_2maxP_1. Il problema CD come dimostrato in [7] è un problema NP-hard; di conseguenza anche il problema P_2maxP_1 è NP-hard.

L'algoritmo 4 è un possibile algoritmo greedy per risolvere il problema P_2maxP_1. Notiamo che questo algoritmo alla riga 4 richiama una subroutine per individuare la massima clique in G, il quale è un noto problema NP-hard ampiamente analizzato e per cui esistono vari algoritmo approssimato, come

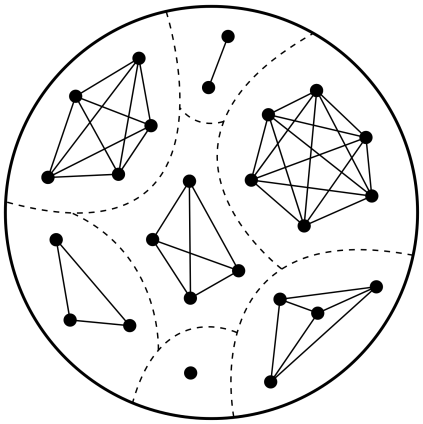

Figura 3.3: Un cluster graph con cluster (sottografi completi) di taglia 1, 2, 3, 4, 4, 5, 6

ad esempio [8] che richiede tempo $\mathcal{O}(\frac{|V|}{(log|V|)^2})$. L'algoritmo 4 può essere quindi implementato in tempo $\mathcal{O}(\frac{|V|^2}{(log|V|)^2})$.

Algorithm 4: $P_2 max P_1$ Greedy

Input: Grafo $G = (V, E)$

1: $T = V$
2: $K = E$
3: **while** $T \neq \emptyset$ **do**
4: $(V', E') = CliqueMax((T, K))$
5: $T = T - V'$
6: $K = K - E'$
7: **end while**

È stato dimostrato in [9] che l'algoritmo 4 è un algoritmo 2-approssimato per classi di grafi per i quali è possibile trovare in modo efficiente le clique massime.

3.2.5 Algoritmo offline

La procedura di clusterizzazione delle tuple rispetto il LHS e il RHS su dataset statici al fine di validare una data RFD è sintetizzata nella figura 3.4.

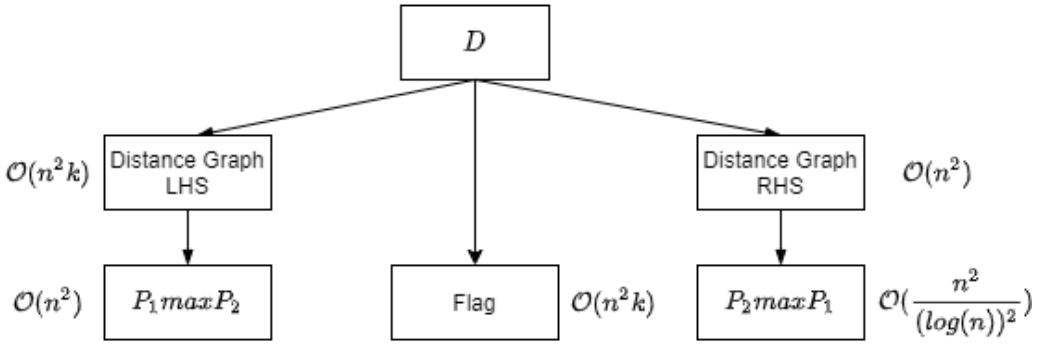

Figura 3.4: Diagramma di flusso algoritmo offline

Dato un dataset D l'algoritmo offline ha un triplice scopo. Innanzitutto, attraverso un flag, determina se la RFD di input è valida o meno in D. Inoltre, determina gli insiemi di cluster sul LHS e sul RHS per un'eventuale sommarizzazione su uno stream di dati. L'intera procedura richiede tempo $\mathcal{O}(n^2k)$, con n tuple in D e cardinalità k del LHS. In 5 è riportato lo pseudocodice dell'algoritmo offline.

Algorithm 5: Algoritmo offline

Input:

- Insieme di tuple D
- Funzioni di distanza rispetto LHS $f_1, \dots f_k$
- Soglie rispetto LHS $\beta_1, \dots \beta_k$
- Funzione di distanza rispetto RHS f_{k+1}
- Soglia rispetto RHS β_{k+1}

Output:

- Flag che indica se la RFD vale
- C_{RHS} Cluster sul RHS
- C_{LHS} Cluster sul LHS

1: $G_{LHS} \leftarrow$ *DISTANCE-GRAPH-LHS(D,* $f_1, \dots f_k$, $\beta_1, \dots \beta_k$*)*
2: $G_{RHS} \leftarrow$ *DISTANCE-GRAPH-RHS(D,* f_{k+1}, β_{k+1}*)*
3: $C_{LHS} \leftarrow P_1 max P_2(G_{LHS})$
4: $C_{RHS} \leftarrow P_2 max P_1(G_{RHS})$
5: *Flag* $\leftarrow$ *True se la RFD vale False altrimenti*
6: **return** *Flag,* C_{RHS}, C_{LHS}

3.3 Applicazione su datastream

In questa sezione è descritta una metodologia per applicare la validazione di RFD basata su cluster a uno stream di dati. Nella sottosezione 3.3.1 è descritta una struttura dati per memorizzare le sommarizzazioni. Nella sottosezione 3.3.2 si trasferiscono le proprietà P_1 e P_2 su uno stream di dati. Nella sottosezione 3.3.3 è descritta la struttura dell'algoritmo online.

3.3.1 Matrice di contingenza

Diamo la definizione di matrice di contingenza della tabella 3.10.

Definizione 3.3.1. (Matrice di contingenza)

Y \ X	X_1	...	X_m	
Y_1	$c_{1,1}$	...	$c_{1,m}$	C_{Y_1}
...	...	...	...	...
Y_n	$c_{n,1}$	...	$c_{n,m}$	C_{Y_n}
	C_{X_1}	...	C_{X_m}	

Tabella 3.10: Matrice di contingenza

Sia M una struttura dati che permette di memorizzare le informazioni inerenti a due differenti clusterizzazioni di un insieme di item.

Stato:

- m = numero di cluster *tipo1*
- n = numero di cluster *tipo2*
- $M(i, j)$ = numero di tuple nel cluster i di *tipo1* e j di *tipo2*
- *wrapper_cluster_tipo1* = lista di wrapper cluster di *tipo1*
- *wrapper_cluster_tipo2* = lista di wrapper cluster di *tipo2*

Funzioni:

- *bool merge*($clusterTipo1, clusterTipo2$): *clusterTipo1* e *clusterTipo2* sono due liste di cluster rispettivamente di *tipo1* e di *tipo2*.
  I cluster vengono sommarizzati ai rispettivi cluster presenti in M; è restituito $False$ se esiste una coppia di item in uno stesso cluster di *tipo1* ma in differenti cluster di *tipo2*

Wrapper Cluster

- id = identificativo cluster
- rappresentante = codifica che permette di sommarizzare il cluster id; dipende dalla tecnica di sommarizzazione
- k = numero items nel cluster id
- items = items nel cluster id

Una volta costruita la matrice M, sommando gli elementi per righe si avrà il numero di tuple in ogni cluster su *tipo2*, mentre sommando gli elementi per colonne si avrà il numero di tuple in ogni cluster su *tipo1*. Chiameremo una tale matrice M matrice di contingenza.

Possiamo utilizzare la matrice di contingenza per i nostri scopi, con cluster di *tipo1* e di *tipo2* i cluster rispettivamente sul LHS e sul RHS, al fine di:

- sommarizzare i cluster per l'esecuzione dell'algoritmo 5 su datastream,
- verificare se esiste un coppia di tuple t_1 e t_2 tale che $F_{LHS}(t_1) = F_{LHS}(t_2)$ e $F_{RHS}(t_1) \neq F_{RHS}(t_2)$.

Osservazione 3.3.1. Per ogni coppia di tuple si possono verificare i seguenti 4 casi:

1	stesso cluster sul RHS e stesso cluster sul LHS	stessa riga e stessa colonna su M
2	stesso cluster sul RHS e diverso cluster sul LHS	stessa riga e diversa colonna su M
3	diverso cluster sul RHS e stesso cluster sul LHS	diversa riga e stessa colonna su M
4	diverso cluster sul RHS e diverso cluster sul LHS	diversa riga e diversa colonna su M

Si può verificare l'esistenza di una coppia di tuple che sono nello stesso cluster sul LHS ma cluster diverso sul RHS andando a verificare se in una colonna esistono più di due numeri diversi da 0. Questo perché la matrice

codifica nella stessa colonna ma riga diversa due tuple che sono nello stesso cluster sul LHS e diverso cluster sul RHS.

Dato un insieme di tuple e due funzioni di clustering F_{LHS} e F_{RHS}, l'algoritmo 6 determina se esiste una coppia di tuple t_1 e t_2 tale che $F_{LHS}(t_1) = F_{LHS}(t_2)$ e $F_{RHS}(t_1) \neq F_{RHS}(t_2)$. Tale algoritmo ha complessità di tempo di $O(|D|)$.

Algorithm 6: Validazione RFD matrice di contingenza

Input:

- D insieme di tuple
- RFD $X \rightarrow Y$
- F_X clustering su X
- F_Y clustering su Y

Output: $True$ se RFD $X \rightarrow Y$ vale, $False$ altrimenti

1: Sia n = numero cluster su Y e m = numero cluster su X
2: Sia M una matrice $n \times m$, $\forall\ i, j\ M[i, j] = 0$
3: Sia $flag$ un vettore di booleani di dimensione m $\forall\ i\ flag[i] = false$
4: **for all** $t_i \in D$ **do**
5: **if** $flag[F_X(t_i)] = false$ **then**
6: $flag[F_X(t_i)] = true$
7: $M[F_Y(t_i), F_X(t_i)] + +$
8: **else**
9: **if** $M[F_Y(t_i), F_X(t_i)] \neq 0$ **then**
10: $M[F_Y(t_i), F_X(t_i)] + +$
11: **else**
12: **return** *False*
13: **end if**
14: **end if**
15: **end for**
16: **return** *True*

Nota: l'algoritmo 6 può essere ottimizzato in termini di spazio, in quanto la matrice di contingenza è di fatto una matrice sparsa.

Presentiamo ora una dimostrazione della correttezza della validazione delle RFD da parte della matrice di contingenza.

Osservazione 3.3.2. $M(i, j) \geq 1$ sse esiste una tupla nel cluster i rispetto RHS e nel cluster j rispetto LHS.

Proposizione 3.3.1. Sia M la matrice di contingenza. Non esiste un coppia di tuple t_1 e t_2 tale che $F_{LHS}(t_1) = F_{LHS}(t_2)$ e $F_{RHS}(t_1) \neq F_{RHS}(t_2) \iff$ per ogni colonna in M esiste al più un numero maggiore o uguale di 1.

Dimostrazione. Dimostriamo la prima implicazione da sinistra verso destra. Se non esiste un coppia di tuple t_1 e t_2 tale che $F_{LHS}(t_1) = F_{LHS}(t_2)$ e $F_{RHS}(t_1) \neq F_{RHS}(t_2)$, in termini di matrice abbiamo che non esiste un coppia di tuple con la stessa colonna ma con riga diversa, di conseguenza per ogni colonna in M esiste al più un numero maggiore o uguale di 1.

Dimostriamo la contronominale dell'inverso. Se esiste un coppia di tuple t_1 e t_2 tale che $F_{LHS}(t_1) = F_{LHS}(t_2)$ e $F_{RHS}(t_1) \neq F_{RHS}(t_2)$, in termini di matrice abbiamo che esiste un coppia di tuple con la stessa colonna ma con riga diversa, di conseguenza tale colonna avrà più di un numero maggiore o uguale di 1. □

Esempio 3.3.1 (Caso in cui la RFD vale sul dataset)**.** Consideriamo il dataset riportato nella tabella 3.11. Supponiamo di avere l'insieme di cluster riportato nella tabella 3.12. Otteniamo la matrice riportata nella tabella 3.13. Da notare che siccome la RFD vale sul dataset, in ogni colonna vi è al più un numero maggiore o uguale a 1.

	X	Y
t_1	xxx	10
t_2	xxa	9
t_3	yyy	16
t_4	yya	17
t_5	yyb	15
t_6	ttt	20
t_7	ttc	21

Tabella 3.11: Dataset D per validazione $X_1 \to Y_2$

	$F_y(.)$	$F_x(.)$
t_1	1	1
t_2	1	1
t_3	2	2
t_4	2	2
t_5	2	2
t_6	3	3
t_7	3	3

Tabella 3.12: Clustering delle tuple

$F_y(.)$ \ $F_x(.)$	1	2	3	
1	2	0	0	2
2	0	3	0	3
3	0	0	2	2
	2	3	2	7

Tabella 3.13: Matrice

Esempio 3.3.2 (Caso in cui la RFD non vale sul dataset)**.** Consideriamo il dataset riportato nella tabella 3.14. Supponiamo di avere il clustering riportato nella tabella 3.15. Otteniamo la matrice riportata nella tabella 3.16. La coppia di tuple t_1, t_5 è simile rispetto l'attributo X ma non simile rispetto Y, quindi la RFD non vale. Siccome la RFD non vale, esiste una colonna in cui vi sono almeno due celle con valori maggiori o uguali a 1.

	X	Y
t_1	yay	11
t_2	zzz	19
t_3	zzz	13
t_4	zzz	15
t_5	yyy	17
t_6	yay	12
t_7	zzy	21

Tabella 3.14: Dataset D per validazione $X_2 \rightarrow Y_2$

	$F_{RHS}(.)$	$F_{LHS}(.)$
t_1	1	1
t_2	3	1
t_3	1	1
t_4	2	1
t_5	2	1
t_6	1	1
t_7	3	1

Tabella 3.15: Clustering delle tuple

$F_{RHS}(.)$ \ $F_{LHS}(.)$	1	
1	3	3
2	2	2
3	2	2
	7	7

Tabella 3.16: Matrice

3.3.2 P1 e P2 online

L'algoritmo 5 è un algoritmo offline che prende in input un singolo dataset. Per poterlo eseguire su uno flusso continuo di dati è necessario che i cluster su LHS e RHS continuino a soddisfare le proprietà P_1 e P_2 anche online, al fine di poter continuare a garantire zero falsi positivi. Di seguito saranno utilizzate le seguenti notazioni:

- $D_1, ..., D_{i-1}, D_i$ i dataset in input rispettivamente agli istanti di tempo $1, ..., i-1, i$,
- $D = \bigcup_{j=1}^{i} D_j$.
- G_{LHS} il grafo delle distanze sul LHS associato a D,
- G_{RHS} il grafo delle distanze sul RHS associato a D,
- M_1^{i-1} la matrice di contingenza fino all'istante $i-1$.

P1 online

Nell'istante di tempo i quando arriva il dataset D_i, il grafo G_i delle distanze sul LHS associato a D_i deve essere sommarizzato con i dataset precedenti in maniera tale da soddisfare la proprietà P_1 e massimizzare P_2 su G_{LHS}. In particolare, si ha il seguente problema da risolvere: dato un cluster C, come deve essere sommarizzato C in M_1^{i-1} per non violare la proprietà P_1 massimizzando P_2?

Siano:

- $t \in C$ una tupla,
- $Q_t = \{S \textit{ cluster sul LHS in } M_1^{i-1} \mid \exists q \in S \textit{ tale che } tR_{si}q\}$.

Si possono verificare tre casi:

- $|Q_t| = 0$. Non esiste $q \in S$ con S in M_1^{i-1}, tale che $tR_{si}q$. Di conseguenza dobbiamo aggiungere tutte le tuple in C in un nuovo cluster in M_1^{i-1};

- $|Q_t| = 1$. È necessario unire il cluster C con il cluster in Q_t in M_1^{i-1};
- $|Q_t| \geq 2$. È necessario unire il cluster C con tutti i distinti cluster in Q_t in M_1^{i-1}.

P2 online

Nell'istante di tempo i quando arriva il dataset D_i, il grafo G_i delle distanze sul RHS associato a D_i deve essere sommarizzato con i dataset precedenti in maniera tale da soddisfare la proprietà P_2 su G_{RHS}. In particolare, si ha il seguente problema da risolvere: dato un cluster C, come deve essere sommarizzato C in M_1^{i-1} per non violare la proprietà P_2?

Siano $C_1, ..., C_n$ i cluster in M_1^{i-1} sul RHS. Consideriamo l'insieme di cluster $C' = \{C, C_j\}$, per $j = 1, ..., n$.

I casi che si possono verificare sono i seguenti:

- Non esiste C' che soddisfa la proprietà P_2. In tal caso è necessario aggiungere C in un nuovo cluster in M_1^{i-1} per soddisfare la proprietà P_2.
- Esiste un cluster $C' = \{C, C_j\}$ che soddisfa la proprietà P_2. In tal caso si unisce C con il cluster C_j che soddisfa P_2 e massimizza P_1.

3.3.3 Algoritmo online

La procedura di validazione di RFD su un flusso di dati è sintetizzata nella figura 3.5.

L'algoritmo 7 è eseguito su uno stream di dati. Ad ogni istante di tempo l'algoritmo prende in input un dataset D_i con n_i tuple. Questo dataset è passato in input all'algoritmo 5 che restituisce una tripla $(flag, C_{RHS}, C_{LHS})$. Se il $flag$ è $False$ la RFD non vale su D_i, e di conseguenza non varrà nemmeno su D. In caso contario, i cluster sul LHS e sul RHS sono sommarizzati ai rispettivi cluster nella matrice $M_1^{(i-1)}$ in maniera da rispettare le proprietà P_1 e P_2. Al termine della sommarizzazione, la funzione di merge della matrice

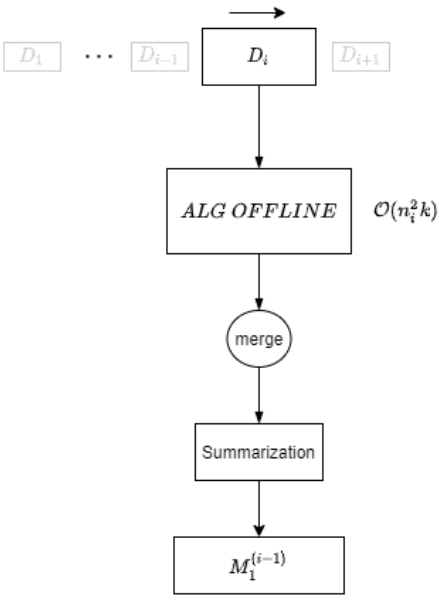

Figura 3.5: Diagramma di flusso algoritmo online

restituisce un flag booleano che indica se è possibile invalidare o meno la RFD. La complessità di tempo dell'algoritmo online su ogni dataset dipende dalla complessità dell'algoritmo offline e della funzione di merge che effettua la sommarizzazione.

3.4 Sommarizzazione per oggetti ordinabili

In questo capitolo è presentata un'applicazione su datastream di oggetti ordinabili. Nella sezione 3.4.1 è descritto il concetto di oggetti ordinabili. Nella sezione 3.4.2, dopo aver mostrato che l'insieme $\mathbb{R}$ è un insieme di oggetti ordinabili, è mostrato un esempio di esecuzione della procedura di validazione su uno stream di numeri reali.

3.4.1 Oggetti ordinabili

Nel capitolo 3.3 è stata descritta la struttura generale dell'algoritmo 7 senza scendere nei dettagli per quanto riguarda la sommarizzazione. In questo capitolo verrà descritto il concetto di oggetti ordinabili e una sommarizzazione ad essi connessa.

Algorithm 7: Algoritmo online

Input:

- Insieme di tuple D
- Funzioni di distanza rispetto LHS $f_1, \dots f_k$
- Soglie rispetto LHS $\beta_1, \dots \beta_k$
- Funzione di distanza rispetto RHS f_{k+1}
- Soglia rispetto RHS β_{k+1}

Output: $True$ se la RFD è valida, $False$ altrimenti

1: $M_1^{(i-1)}$ *matrice di contingenza fino all'istante i-1*
2: $flag, C_{RHS}, C_{LHS} \leftarrow$ *ALG-OFFLINE(D,* $f_1, \dots f_k$, $\beta_1, \dots \beta_k$, f_{k+1}, β_{k+1}*)*
3: **if** $flag$ è $True$ **then**
4: **return** $M_1^{(i-1)}.merge(C_{RHS}, C_{LHS})$
5: **else**
6: **return** False
7: **end if**

Definizione 3.4.1. Oggetti ordinabili

Un insieme U di oggetti è un insieme di oggetti ordinabili se e solo se:

1. È possibile definire su U una relazione d'ordine $R_{\leq}$ totale
2. È definita su U una funzione di distanza $d(x, y)$ $\forall x, y \in U$
3. $\forall x, y, z \in U$: se $xR_{\leq}y$ e $yR_{\leq}z$ allora $d(x, y) \leq d(x, z)$ e $d(y, z) \leq d(x, z)$

Una conseguenza del punto 3 della definizione 4.3.2 è che la distanza massima tra due oggetti in U è $d(m(U), M(U))$.

Di seguito saranno utilizzate le seguenti notazioni:

- $X_1, \ldots, X_k \rightarrow Y$ una RFD, con soglie $\beta_1, \ldots, \beta_k$ sul LHS e soglia β_{k+1} sul RHS, dove $\forall I \in \{X_1, \ldots, X_k, Y\}$, $dom(I)$ è un insieme di oggetti ordinabili.
- $D_1, ..., D_{i-1}, D_i$ i dataset in input rispettivamente agli istanti di tempo $1, ..., i-1, i$,
- $D = \bigcup_{j=1}^{i} D_j$.
- M_1^{i-1} la matrice di contingenza fino all'istante $i-1$.
- $C_{LHS} = \{C_{1,LHS}, \ldots, C_{w,LHS}\}$ l'insieme di cluster sul LHS in M_1^{i-1}
- $C_{RHS} = \{C_{1,RHS}, \ldots, C_{v,RHS}\}$ l'insieme di cluster sul RHS in M_1^{i-1}

Sia Q un generico cluster di tuple e sia $I \in \{X_1, \ldots, X_k, Y\}$, $Q[I]$ corrisponde alla project delle tuple in Q rispetto l'attributo I.

Definizione 3.4.2. Un generico cluster Q sul LHS di $X_1, \ldots, X_k \rightarrow Y$ può essere sommarizzato con $c_{j,min}, c_{j,max}$ per $j = 1, \ldots, k$ (ovvero k coppie di oggetti ordinabili):

- $c_{j,min} = m(Q[X_j])$
- $c_{j,max} = M(Q[X_j])$

Definizione 3.4.3. Un generico cluster T sul RHS di $X_1, \ldots, X_k \rightarrow Y$ può essere sommarizzato con una coppia di oggetti ordinabili c_{min} e c_{max}:

- $c_{min} = m(T[Y])$
- $c_{max} = M(T[Y])$

P1 online per oggetti ordinabili

Nell'istante di tempo i quando arriva il dataset D_i, il grafo G_i delle distanze sul LHS associato a D_i, clusterizzato dall'algoritmo offline in $C'_{LHS} = \{C'_{1,LHS}, \ldots, C'_{w',LHS}\}$, deve essere sommarizzato con i dataset precedenti in maniera tale da soddisfare la proprietà P_1 e massimizzare P_2 su G_{LHS}. In

particolare, si ha il seguente problema da risolvere: dato un cluster $C' \in C'_{LHS}$, come deve essere sommarizzato C' in M_1^{i-1} per non violare la proprietà P_1 massimizzando P_2?

Sia $C \in C_{LHS}$, C' e C possono essere uniti soddisfando P_1 e massimizzando P_2 se e solo se una delle seguenti affermazioni si verifica $\forall j = 1, \ldots, k$. Supponiamo che $c_{j,min} R_{\leq} c'_{j,min}$, senza perdita di generalità, in quanto in caso fosse $c'_{j,min} R_{\leq} c_{j,min}$ basterebbe invertire il ragionamento:

- $c'_{j,min}\ R_{\leq}\ c_{j,max}$
- $c_{j,max}\ R_{\leq}\ c'_{j,min}$ e $d(c_{j,max}, c'_{j,min}) \leq \beta_j$

P2 online per oggetti ordinabili

Nell'istante di tempo i quando arriva il dataset D_i, il grafo G_i delle distanze sul RHS associato a D_i, clusterizzato dall'algoritmo offline in $C'_{RHS} = \{C'_{1,RHS}, \ldots, C'_{v',RHS}\}$, deve essere sommarizzato con i dataset precedenti in maniera tale da soddisfare la proprietà P_2 su G_{RHS}. In particolare, si ha il seguente problema da risolvere: dato un cluster $C' \in C'_{RHS}$, come deve essere sommarizzato C' in M_1^{i-1} per non violare la proprietà P_2?

Sia $C \in C_{RHS}$, C' e C possono essere uniti rispettando P_2 se e solo se $m(\{c_{min} \cup c'_{min}\}) = m$ e $M(\{c_{max} \cup c'_{max}\}) = M$ sono tali che $d(m, M) \leq \beta_{k+1}$.

Funzione merge per oggetti ordinabili

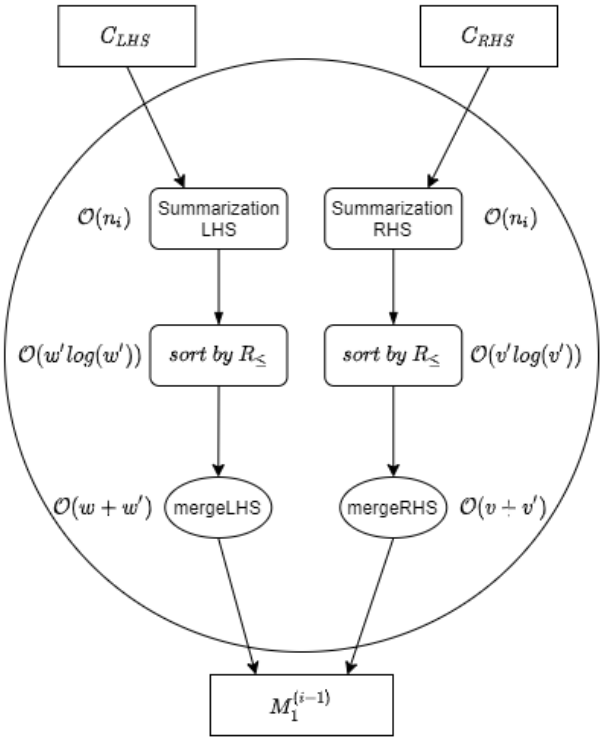

Figura 3.6: Funzione merge

La funzione merge per oggetti ordinabili, il cui pseudocodice è riportato in 8, è caratterizzata da 3 passi principali:

1. Sommarizzazione dei nuovi cluster, come mostrato in 3.4.2 e 3.4.3.

2. Ordinamento dei nuovi wrapper cluster rispetto la $R_{\leq}$ presa in considerazione. Notiamo che al fine di effettuare il merge i wrapper cluster sul LHS possono essere ordinati rispetto uno qualsiasi degli attributi in $\{X_1, \ldots, X_k\}$.

3. Merge dei nuovi wrapper cluster con quelli presenti nella matrice. Poiché le liste cluster arrivano già ordinate il merge consiste nell'unione di due liste ordinate, mentre il merge dei singoli cluster può essere effettuato come descritto in 3.4.1 e 3.4.1.

Algorithm 8: Funzione merge

Input:

- $C'_{LHS} = \{C'_{1,LHS}, \ldots, C'_{w',LHS}\}$ l'insieme di cluster sul LHS di D_i
- $C'_{RHS} = \{C'_{1,RHS}, \ldots, C'_{v',RHS}\}$ l'insieme di cluster sul RHS di D_i

Output: $True$ se la RFD è valida, $False$ altrimenti

Stato struttura dati:

- w = numero di wrapper cluster LHS
- v = numero di wrapper cluster RHS
- $WrapperClusterLHSsorted$ l'insieme di cluster sul LHS di D_i
- $WrapperClusterRHSsorted$ l'insieme di cluster sul RHS di D_i
- Matrice $M_1^{(i-1)}$

1: $WrapperClusterLHS \leftarrow SUMMARIZATION(C_{LHS})$
2: $WrapperClusterRHS \leftarrow SUMMARIZATION(C_{RHS})$
3: $WrapperClusterLHSsorted \leftarrow sortbyR_{\leq}(WrapperClusterLHS, X_1)$
4: $WrapperClusterRHSsorted \leftarrow sortbyR_{\leq}(WrapperClusterRHS, Y)$
5: *Merge WrapperClusterLHSsorted* e $M_1^{(i-1)}.WrapperClusterLHSsorted$ rispettando P_1 e massimizzando P_2
6: *Merge WrapperClusterRHSsorted* e $M_1^{(i-1)}.WrapperClusterRHSsorted$ rispettando P_2
7: $flag \leftarrow$ *Validazione RFD matrice di contingenza*
8: **return** $flag$

3.4.2 Stream di numeri reali

In questa sezione si dimostra che l'insieme $\mathbb{R}$ dei numeri reali è un insieme di oggetti ordinabili. Successivamente è riportato un esempio di esecuzione dell'algoritmo di validazione su uno stream di numeri reali.

$\mathbb{R}$ è un insieme di oggetti ordinabili

Proposizione 3.4.1. L'insieme dei numeri reali è un insieme di oggetti ordinabili.

Dimostrazione. Per dimostrare che l'insieme dei numeri reali è un insieme di oggetti ordinabili occorre provare che soddisfa le tre proprietà della definizione 4.3.2:

1. È possibile definire su $\mathbb{R}$ una relazione d'ordine $R_{\leq}$ totale:
   $\forall x, y \in \mathbb{R} : xR_{\leq}y \iff x \leq y$;

2. È definita su $\mathbb{R}$ una funzione di distanza $d(x, y)$ $\forall x, y \in \mathbb{R}$:
   Come funzione di distanza può essere utilizzata la funzione di distanza valore assoluto $d(x, y) = |x - y|$;

3. $\forall x, y, z \in \mathbb{R}$: se $xR_{\leq}y$ e $yR_{\leq}z$ allora $d(x, y) \leq d(x, z)$ e $d(y, z) \leq d(x, z)$:
   Per ogni $x, y, z \in \mathbb{R}$ se $xR_{\leq}y$ e $yR_{\leq}z$ allora $d(x, z) = d(x, y) + d(y, z)$; di conseguenza $d(x, y) \leq d(x, z)$ e $d(y, z) \leq d(x, z)$.

□

Validazione di uno stream di numeri reali

Mostriamo un'esecuzione dell'intera procedura di validazione sullo stream S di dati numerici riportato nella tabella 3.17. Supponiamo che ad ogni iterazione sia considerata una slice di dimensione 4. Inoltre, sul LHS, rappresentato dall'attributo X, consideriamo una soglia $\beta_1 = 3$, mentre sul RHS, rappresentato dall'attributo Y, una soglia $\beta_2 = 2$. La funzione di distanza usata per entrambi gli attributi è $d(x, y) = |x - y|$.

	X	Y
t_1	3	7
t_2	9	2
t_3	17	1
t_4	2	9
t_5	8	0
t_6	16	0
t_7	11	0
t_8	2	9
t_9	12	6
t_{10}	8	2
t_{11}	15	8
t_{12}	7	6

Tabella 3.17: Stream di dati S

Consideriamo la matrice di contingenza della figura 3.10. Ogni cella dell'ultima colonna della matrice contiene le seguenti informazioni sui wrapper cluster C_{Y_i}:

- numero di tuple,
- tuple,
- centro (in quanto nel dominio di $\mathbb{R}$ può essere calcolato il centro di un intervallo),
- minorante e maggiorante.

Ogni cella dell'ultima riga della matrice contiene le seguenti informazioni sui wrapper cluster C_{X_j}:

- numero di tuple,
- tuple del cluster,

- minorante e maggiorante.

Le restanti celle riportano le informazioni sugli elementi $c_{i,j}$:

- numero di tuple,
- tuple.

Da notare che le tuple dei cluster sono state riportate solo a scopo illustrativo per questo specifico esempio. Nella pratica l'algoritmo non mantiene questa informazione.

Alla prima iterazione, sono passate all'algoritmo online (algoritmo 7) le tuple $t_1, \ldots, t_4$ di S. Indichiamo tali tuple con D_1. A sua volta, l'algoritmo online passa in input D_1 all'algoritmo offline (algoritmo 5). Quest'ultimo, dopo aver verificato che la RFD è valida su D_1, calcola il grafo delle distanze sul LHS, riportato in figura 3.7 e il grafo delle distanze sul RHS, riportato in figura 3.8. Sono quindi calcolati, a partire da tali grafi, i cluster rispettivamente sul LHS e sul RHS attraverso le procedure $P_1 max P_2$ e $P_2 max P_1$. L'algoritmo offline termina restituendo la tripla $(True, cluster_{RHS}, cluster_{LHS})$ all'algoritmo online.

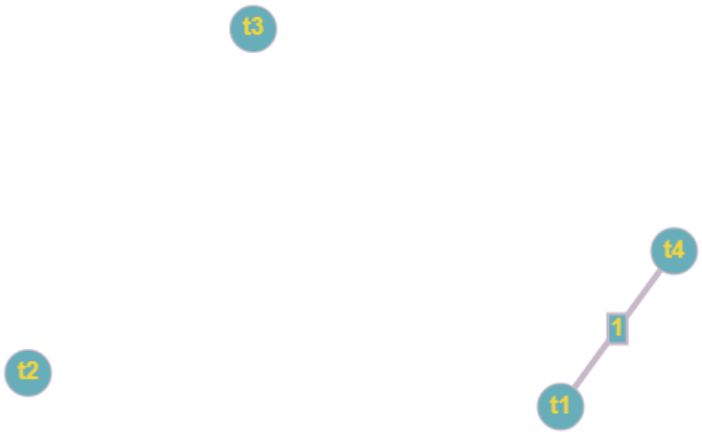

Figura 3.7: Grafo delle distanze sul LHS D_1

Poiché la RFD è valida su D_1, l'algoritmo online sommarizza i cluster ricevuti. Poiché questa è la prima iterazione la matrice M_1^1, riportata in figura 3.18, conterrà solo tali cluster.

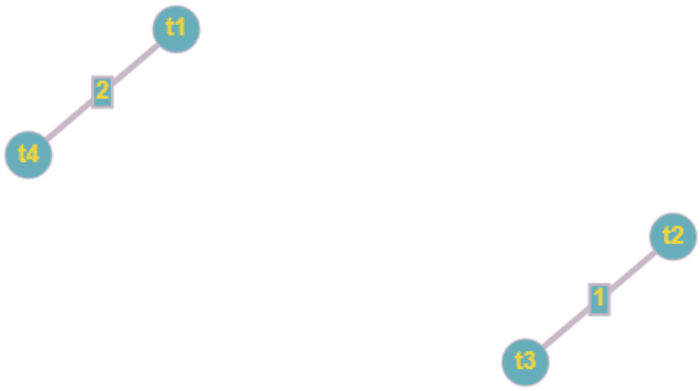

Figura 3.8: Grafo delle distanze sul RHS D_1

LHS / RHS	1	2	3	
1	0 {} 	1 {t2} 	1 {t3} 	2 {t2,t3} 1.5 [1,2]
2	2 {t1,t4} 	0 {} 	0 {} 	2 {t1,t4} 8 [7,9]
	2 {t1,t4} [2,3]	1 {t2} [9,9]	1 {t3} [17,17]	
Valida	True			

Tabella 3.18: Matrice di contingenza M_1^1

Nella seconda iterazione, sono considerate le successive 4 tuple di S, cioè $D_2 = \{t_5, \ldots, t_8\}$. Come prima l'algoritmo offline, dopo aver verificato che la RFD vale su D_2, calcola i grafi delle distanze riportati nelle figure 3.9 e 3.10 e determina i cluster riportati nella matrice M_2 (tabella 3.19).

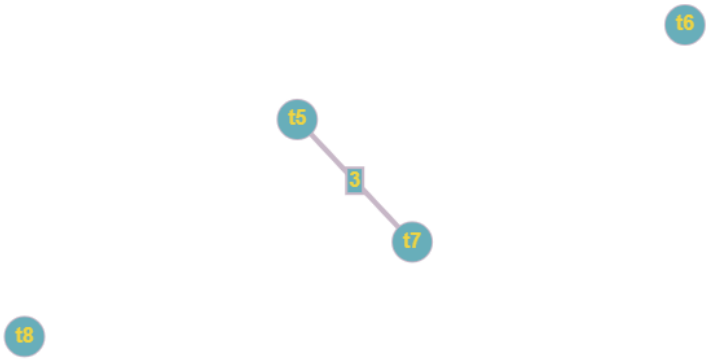

Figura 3.9: Grafo delle distanze sul LHS D_2

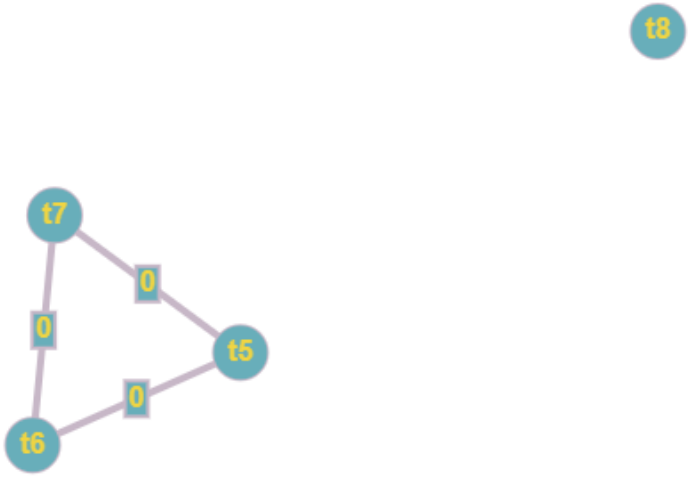

Figura 3.10: Grafo delle distanze sul RHS D_2

RHS \ *LHS*	1	2	3	
1	0 {} [0,0]	2 {t5,t7} [8,11]	1 {t6} [16,16]	3 {t5,t6,t7} 0 [0,0]
2	1 {t8} [2,2]	0 {} [0,0]	0 {} [0,0]	1 {t8} 9 [9,9]
	1 {t8} [2,2]	2 {t5,t7} [8,11]	1 {t6} [16,16]	
Valida	True			

Tabella 3.19: Matrice di contingenza M_2

L'algoritmo online, quindi, sommarizza i cluster riportati nella matrice M_2, con quelli della matrice M_1^1. La matrice di contingenza evolve nella matrice riportata nella figura M_1^2.

RHS \ *LHS*	1	2	3	
1	0 {} 	3 {t2,t5,t7} 	2 {t3,t6} 	5 {t2,t3,t5,t6,t7} 1 [0,2]
2	3 {t1,t4,t8} 	0 {} 	0 {} 	3 {t1,t4,t8} 8 [7,9]
	3 {t1,t4,t8} [2,3]	3 {t2,t5,t7} [8,11]	2 {t3,t6} [16,17]	
Valida	True			

Tabella 3.20: Matrice di contingenza M_1^2

Osservando le colonne della matrice M_1^2 possiamo verificare che la RFD è valida su tutti i dataset D_i finora considerati.

Nella terza iterazione, sono considerate le successive tuple $D_3 = \{t_9, \ldots, t_{12}\}$ di S. Questa volta l'algoritmo offline restituirà il flag $False$ poiché la RFD non è valida, come si evince dalla matrice M_3 riportata in figura 3.21. Infatti, nella prima colonna esiste più di un valore diverso da zero. In questo caso l'algoritmo online non calcola i grafi riportati nelle figure 3.11 e 3.12, poiché le tuple t_{10} e t_{12} sono simili su X ma dissimili su Y.

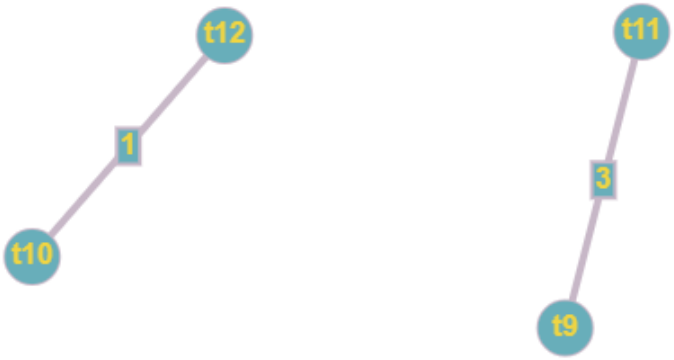

Figura 3.11: Grafo delle distanze sul LHS D_3

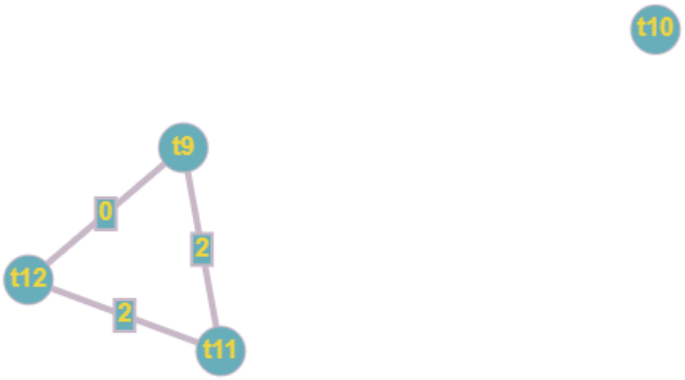

Figura 3.12: Grafo delle distanze sul RHS D_3

RHS \ *LHS*	1	2	
1	1 {t10}	0 {}	1 {t10} 2 [2,2]
2	1 {t12}	2 {t9,t11}	3 {t9,t11,t12} 7 [6,8]
	2 {t10,t12} [7,8]	2 {t9,t11} [12,15]	
Valida	False		

Tabella 3.21: Matrice di contingenza M_3

L'algoritmo online quindi non effettua il merge e termina restituendo il valore $False$.

3.5 Clustering Deletion per RFD

Nelle precedenti sezioni del capitolo 3 è stato descritto un possibile approccio al problema della validazione delle RFD su uno stream di dati. In questa sezioni verrà descritto in che modo i risultati ottenuti dalla tesi possono migliorare la bontà dei risultati ottenibili dall'algoritmo di validazione delle RFD proposto.

L'algoritmo di validazione delle RFD basato su tecniche di clustering necessita della risoluzione del problema $P_2 max P_1$, il quale abbiamo visto essere di fatto il problema del Clustering Deletion. Poichè è un problema NP-hard non è possibile risolverlo all'ottimo e non risolvere all'ottimo questo problema ha conseguenze negative sulla precisione della validazione delle RFD

dell'algoritmo, nonchè sui tempi necessari per avere una buona soluzione se utilizzati algoritmi approssimati o euristici. Se siamo interessati però all'utilizzo dell'algoritmo di validazione per oggetti ordinabili, segue che anche il problema del Clustering Deletion diventa il problema del Clustering Deletion su oggetti ordinabili. Sorge spontanea quindi la domanda se tale problema per oggetti ordinabili è ancora NP-hard. Se così non fosse potremmo risolverlo all'ottimo in tempi rapidi e avendo risultati migliori di validazione. Più nel dettaglio ciò che di fatto facciamo è a partire da un insieme di oggetti ordinabili costruiamo un grafo delle distanze (β similarity graph 4.4) e su questo grafo eseguiamo l'algoritmo. In 4.4 viene provato che un grafo delle distanze costruito a partire da un insieme di oggetti ordinabili è un grafo lineare. Quindi di fatto vogliamo risolvere il problema del Clustering Deletion su grafi lineari. Da qui nasce l'interesse per il problema studiato nella tesi.

CAPITOLO 4

GRAFI LINEARI E OGGETTI ORDINATI

In questo capitolo vengono studiati i grafi lineari e gli oggetti ordinati in maniera più dettagliata di quanto fatto nel capitolo 3. Nella sezione 4.1 vengono introdotti i grafi lineari e discusse le principali caratteristiche. Nella sezione 4.2 è descritto un modello di generazione uniforme di grafi lineari. Nella sezione 4.3 vengono introdotti gli oggetti ordinabili. Nella sezione 4.4 vengono introdotti i β similarity graph e discusse le relazioni esistenti tra i β similarity graph, i grafi lineari e gli oggetti ordinabili.

4.1 Grafi Lineari

Definizione 4.1.1. Grafo linearizzabile

Sia $G = (V, E)$ un grafo, G è linearizzabile se e solo se è esiste una funzione $\pi : V \to \{1, \ldots, n\}$ tale che:

$\forall j$:

- Se $(v_{\pi(j)}, v_{\pi(k)}) \in E$ con $j < k \Rightarrow (v_{\pi(j)}, v_{\pi(r)}) \in E \ \forall r : j < r < k$
- Se $(v_{\pi(k')}, v_{\pi(j)}) \in E$ con $k' < j \Rightarrow (v_{\pi(r)}, v_{\pi(j)}) \in E \ \forall r : k' < r < j$

Diremo invece che un grafo è lineare se la funzione sui nodi $\pi(.)$ è dato. Il nominativo di grafi lineari, da un punto di vista intuitivo, serve per rievocare il fatto che tali grafi, per via dell'ordinamento, possono essere visti come disposti su una linea orizzontale. Il problema di determinare se un grafo $G = (V, E)$, con $|V| = n$ e $|E| = m$ è lineare, ovvero il problema di determinare un ordinamento $\pi(.)$, è un problema risolubile in tempo $\mathcal{O}(n + m)$ [10].

I grafi lineari hanno molteplici caratterizzazioni e proprietà [11]. In letteratura sono noti attraverso vari nominativi *indifference graph*, *unit interval graph*, *proper interval graph*, ecc.

Osservazione 4.1.1. Ogni insieme disgiunto di clique è un grafo lineare.

Osservazione 4.1.2. Non è vero che ogni grafo lineare è un insieme disgiunto di clique. Guardare l'esempio 4.1.1, in cui è presente un grafo lineare che non è un insieme disgiunto di clique.

Esempio 4.1.1. Per comprendere meglio cos'è un grafo lineare consideriamo il grafo in figura 4.1. Tale grafo è un grafo lineare da come è possibile osservare nella figura 4.2, nella quale è presente un'ordinamento $\pi(.)$ che soddisfa la proprietà enunciata nella definizione 4.1.1.

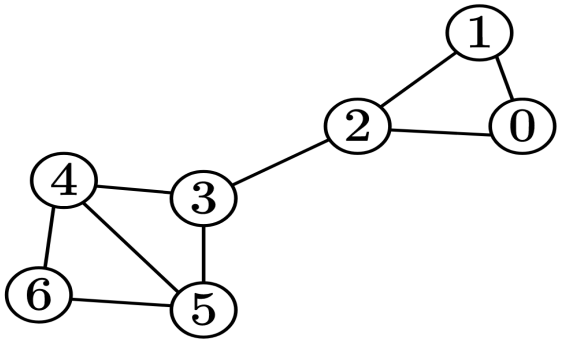

Figura 4.1: Grafo

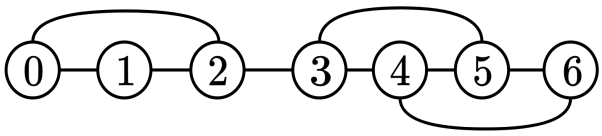

Figura 4.2: Grafo lineare, ordinamento $\pi(.)$

Osservazione 4.1.3. Notiamo che l'ordinamento $\pi(.)$ non è necessariamente unico, infatti nel grafo dell'esempio 4.1.1 è facile individuare un'altro ordinamento $\pi'(.)$ valido mostrato nella figura 4.3.

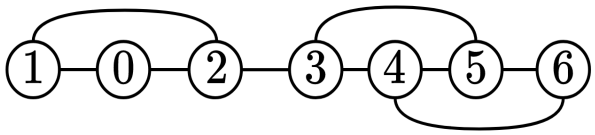

Figura 4.3: Grafo lineare, ordinamento $\pi'(.)$

Un grafo lineare può essere memorizzato in maniera più compatta rispetto ad un generico grafo, in quanto è possibile memorizzare per ogni nodo semplicemente i nodi estremi. Infatti, dalla definizione di grafo lineare segue che un grafo lineare, $G = (V, E)$, con $n = |V|$ e $m = |E|$ il cui ordinamento dei nodi è $v_0, v_1, \ldots, v_{n-1}$, può essere memorizzato utilizzando un numero di edge pari $\mathcal{O}(2n)$ senza perdita di informazione. Infatti se per un generico nodo i consideriamo $l(i)$ e $u(i)$:

$$l(i) := min\{\, j \mid j < i \text{ e } (j,i) \in E \,\}$$
$$u(i) := max\{\, j \mid i < j \text{ e } (i,j) \in E \,\}$$

per ogni nodo i possiamo memorizzare semplicemente $l(i)$ e $u(i)$ e per determinare se per una coppia di nodi $j \neq i$ esiste l'edge (i, j) basta semplicemente controllare se $l(i) \leq j \leq u(i)$, in caso ciò sia vero vuol dire che esiste l'edge, altrimenti non esiste.

Nel grafo dell'esempio 4.1.1 i valori di $u(.)$ e $l(.)$ sono riportati rispettivamente nelle tabelle 4.1 e 4.2.

	0	1	2	3	4	5	6
$u(i)$	2	2	3	5	6	6	-

Tabella 4.1: $u(.)$ per il grafo in figura 5.18

	0	1	2	3	4	5	6
$l(i)$	-	0	0	2	3	3	4

Tabella 4.2: $l(.)$ per il grafo in figura 5.18

Osservazione 4.1.4. Notiamo che per l'$(n-1)-$esimo nodo, ovvero l'ultimo nodo nell'ordinamento, $u(.)$ non è definito. Mentre per il $0-$esimo nodo, ovvero il primo nodo nell'ordinamento, $l(.)$ non è definito.

Osservazione 4.1.5. Tra gli $l(i)$ e $u(i)$ sussiste la seguente relazione:

$$l(0) \leq l(1) \leq \cdots \leq l(n-1)$$

$$u(0) \leq u(1) \leq \cdots \leq u(n-1)$$

Tra le $u(i)$ e le $l(i)$ esiste un'interessante relazione enunciata dalla proposizione che segue.

Proposizione 4.1.1. Fissate le $u(i)$ $(l(i))$ è possibile determinare univocamente le $l(i)$ $(u(i))$. Siano $u(0) \leq u(1) \leq \cdots \leq u(n-2)$, le $l(i)$ possono così essere determinate:

$$l(i) = min\{\, j \mid j \leq i \leq u(j) \,\} \tag{4.1}$$

Dimostrazione. Per prima cosa partiamo dalla definizione delle $l(i)$ e confrontiamola con $l(i) = min\{\, j \mid j \leq i \leq u(j) \,\}$, ovvero quella per cui dobbiamo provare la correttezza. La definizione delle $l(i)$ è la seguente:

$$l(i) := min\{\, j \mid j < i \text{ e } (j,i) \in E \,\}$$

Possiamo osservare che l'unica differenza sta nel fatto che, mentre nella definizione delle $l(i)$ avendo a disposizione tutti gli edge possiamo determinare direttamente se i nodi i e j sono connessi da un arco, nella 4.1, invece, non avendo più a disposizione tutti gli edge, ma bensì solo le $u(i)$, possiamo utilizzare solo questi per poter determinare le $l(i)$. Anche se le informazioni in nostro posseso nei due casi sono differenti il nostro obiettivo è il medesimo

ovvero determinare se i nodi i e j sono connessi da un arco, al fine di poter individuare le $l(i)$. Dobbiamo quindi provare che $j \leq i \leq u(j)$ è equivalente a $j < i$ e $(j,i) \in E$, il quale è equivalente a suppore che $j \leq i$ e provare che $i \leq u(j) \iff (j,i) \in E$. La cui prova segue immediatamento dalla definizione delle $u(i)$. □

A titolo di esempio applichiamo il risultato della proposizione 4.1.1 al grafo lineare in figura 4.2. Fissate le $u(i)$ vediamo come determinare ad esempio $l(5)$. Dalla tabella si vede facilemente che:

$$l(5) = 3 = min\,\{\,3,4\,\}$$

Notiamo che il risultato della proposizione 4.1.1 ci permette di ridurre ulteriormente lo spazio necessario per memorizzare un grafo lineare senza perdita di informazione. Infatti memorizzando solo le $u(i)$ possiamo determinare per j se esiste l'edge (j,i) calcolando a run-time il valore di $l(i)$ e testando se $l(i) \leq j \leq u(i)$. Chiaramente questo approccio ci fa sì risparmiare spazio, ma il test è chiaramente più costoso in termini di tempo. Da un punto di vista asintotico nel caso pessimo potremmo pagare $\mathcal{O}(n)$. Possiamo ridurre tale costo osservando che possiamo utilizzare la ricerca binaria per poter determinare $l(i)$ in questo modo il costo nel caso peggiore diventerebbe $\mathcal{O}(\log n)$. In definitiva il modo migliore per poter memorizzare un grafo lineare è da definirsi in base all'utilizzo che se ne deve fare.

Come conseguenza diretta della proposizione 4.1.1 abbiamo che ogni grafo lineare è caratterizzato univocamente dal vettore $u(.)$, in quanto comprimendo il grafo nel vettore $u(.)$ non abbiamo perdita di informazione.

Proposizione 4.1.2. Ogni grafo lineare è univocamente caratterizzato dal vettore $u(.)$.

Un grafo lineare può essere visto come una catena di clique massimali:

$$(0, u(0))$$

$$(1, u(1)) \iff u(0) < u(1)$$

$$(2, u(2)) \iff u(1) < u(2)$$

$$\dots$$

$$(n-2, u(n-2)) \iff u(n-3) < u(n-2)$$

Quindi se C è l'insieme delle clique massimali di G risulta:

$$C = \{\, (i, u(i)) \mid i = 0, \dots, n-2 \text{ e } u(i-1) < u(i) \,\}$$

Nel grafo dell'esempio abbiamo le seguenti clique massimali:

$$C = \{\, \{\, 0, 1, 2 \,\}, \{\, 2, 3 \,\}, \{\, 3, 4, 5 \,\}, \{\, 4, 5, 6 \,\} \,\}$$

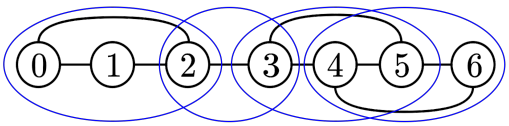

Figura 4.4: Grafo lineare clique massimali

Dato il vettore $u(i)$ caratteristico di un grafo lineare è possibile contare il numero di edge di tale grafo, attraverso la seguente formula:

$$\sum_{i=0}^{n-2} u(i) - i$$

4.1.1 Sul numero dei grafi lineari

Per quanto provato in [12] il numero di grafi lineari con un numero di vertici pari a n è uguale a C_n, dove con C_n facciamo riferimento all'$n-$esimo numero di Catalan. I numeri di Catalan definiscono una sequenza di numeri naturali che occorrono in vari problemi di calcolo combinatorio [13]. L'$n-$esimo numero di Catalan è:

$$C_n = \frac{1}{(n+1)} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1}$$

più precisamente in [12] vengono contati il numero di proper interval graph, ma poichè è noto che i grafi lineari e i proper interval graph, seppur caratterizzati

in modo differente, sono di fatto equivalenti abbiamo che il risultato ottenuto in [12] vale anche per i grafi lineari. É possibile però osservare che il modo a cui si arriva a C_n è dipendente dalla caratterizzazione che hanno i proper interval graph. Ciò che vogliamo fare noi è invece arrivarci sfruttando il risultato ottenuto dalla proposizione 4.1.1, la quale ci permette di associare ad ogni grafo lineare un vettore $u(.)$ di upper bound. Dalla rappresentazione dei grafi lineari come a vettori di upper bound è possibile determinare una corrispondenza biettiva con i numeri di Catalan.

Esistono numerose intepretazioni associate ai numeri di calatalano, tra queste vi è quella che conta il numero di cammini che rispettano un certa proprietà su un struttura a reticolo. Più in particolare fissato un numero intero n, consideriamo il reticolo sul piano cartesiano composto da tutti i punti del tipo (i, j) con $i, j = 0, 1, \ldots, n$. Ad esempio per $n = 7$ abbiamo il seguente reticolo:

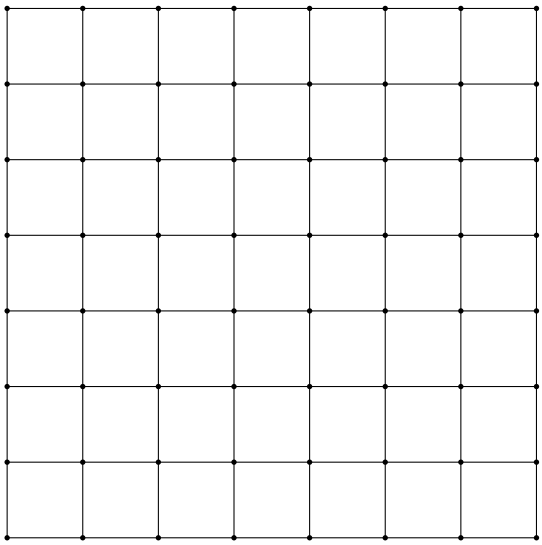

Figura 4.5: Reticolo 7×7

Ora immaginiamo di volere contare tutti i cammini che partono dal punto $(0, 0)$ e terminano nel punto (n, n). In tali cammini possono essere effettuati solo due tipi di passi: passi verso l'alto e passi verso destra. Entrambi i possibili passi che possono essere fatti sono passi unitari, quindi i passi verso l'alto sono passi del tipo $(0, +1)$ e i passi verso destra sono passi del tipo $(+1, 0)$. I

cammini in questione sono inoltre soggetti ad un particolare tipo di vincolo, ovvero sono considerati cammini validi solo quelli che stanno sempre al di sotto della diagonale (i, i). Possiamo formalizzare questo vincolo richiedendo che per ogni cella (i, j) del reticolo visitata dal cammino vale che $j \leq i$ dove con i facciamo riferimento all'asse orizzontale e con j all'asse verticale. Per n fissato il numero di tali cammini sono esattamente pari a C_n. È evidente che per simmetria il numero di tali cammini è uguale al numero di cammini dello stesso tipo ma caratterizzati dal fatto che non possono andare al di sotto della diagonale (i, i), o in altri termini che per ogni cella (i, j) del reticolo visitata dal cammino vale che $i \leq j$.

Osservazione 4.1.6. Quindi il numero di cammini che non possono superare la diagonale (i, i) è uguale al numero di cammini che non possono andare al di sotto della diagonale (i, i). E tale numero di cammini per n fissato è C_n.

Nella proposizione 4.1.1 abbiamo provato che ogni grafo lineare è caratterizzato in modo univoco dal vettore delle $u(.)$. Per queste ragioni ogni grafo lineare può essere visto come una funzione del tipo:

$$u : \{ 1, \ldots, n \} \longrightarrow \{ 1, \ldots, n \}$$

caratterizzata dal fatto che:

1. $u(1) \leq u(2) \leq \cdots \leq u(n-1)$
2. $i \leq u(i)$

Notiamo che per definire grafi lineari connessi la prima proprietà non cambia mentre la seconda diventa più stringente, ovvero diventa $i < u(i)$.

La prova del fatto che tali path, su un reticolo di dimensione n, sono esattamente C_n si basa sul fatto che è possibile codificare ogni path con una particolare stringa costruita su un alfabeto a due simboli {U,R}. Tale associazione viene fatta sfruttando il fatto che ogni cammino alla fine dei conti è una sequenza di passi susseguiti in uno specifico ordine. Tali passi ricordiamo possono essere solo di due tipi, passi verso l'alto U (Up) e passi verso destra

R (Right). Il vincolo di non superare mai la diagonale (i, i) in termini di stringa diventa che in qualsiasi punto la stringa, codificante un certo path, venga considerata si ha che il numero di R è sempre maggiore del numero di U.

Proposizione 4.1.3. Ogni path valido può essere messo in corrispondenza biunivoca con un grafo lineare.

Dimostrazione. Per provare l'asserto dobbiamo provare che:

1. dato un path valido riusciamo a determinare in modo univoco un grafo lineare

2. dato un grafo lineare possiamo costruire in maniera univoca un path valido

Possiamo giungere alla corrispondenza se sfruttiamo il risulto della proposizione 4.1.1 la quale ci permette di identificare in modo univoco un grafo lineare attraverso le $u(.)$. Dimostriamo prima il primo caso e dopo il secondo, tenendo sempre come riferimento un reticolo di dimensione n. In entrambi i casi faremo riferimento a path che non possono andare al di sotto della diagonale (i, i).

Dato un path possiamo costruirci un vettore di $u(.)$. Associamo l'asse orizzontale a un indice $i = 0, \ldots, n$ il quale è associato alle label dei nodi del grafo lineare. Associamo, invece, l'asse verticale a $u(i)$ ovvero l'upper bound corrispondente al nodo con label i, per ogni $i = 0, \ldots, n$. Dato il path, consideriamo la stringa sull'alfabeto {U,R} ad esso associata. A partire da tale stringa possiamo costruire il vettore $u(.)$ in maniera incrementale nel seguente modo. Partiamo col definire $u(1)$. $u(1)$ è uguale al numero di U presenti prima della prima R. Appena incontriamo la prima R ci fermiamo. Definiamo ora $u(2)$, il quale è uguale al numero di U che incontriamo finchè non troviamo la seconda R. Appena incontriamo la seconda R ci fermiamo e definiamo $u(3)$. Questo procedimento si itera fin quando non esauriamo tutta la stringa. All'esaurimento di tutta la stringa avremo anche definito $u(i)$ per ogni $i = 1, \ldots, n$.

Vicerversa dato un grafo lineare è possibile costruire un path valido ragionando in maniera inversa. Sia $u(i)$ per $i = 1, \ldots, n$ il vettore di upper bound associato al grafo lineare, possiamo costruire in maniera incrementale nel seguente modo. Partiamo da $u(1)$, facciamo $u(1)$ passi verso l'alto e 1 passo verso destra. Dopodichè facciamo $u(2)$ passi verso l'alto e 1 passo verso destra. Iteriamo questo ragionamento per ogni $u(i)$ per $i = 1, \ldots, n$ e così facendo otterremo un path valido che parte da $(0, 0)$ e termina in (n, n). □

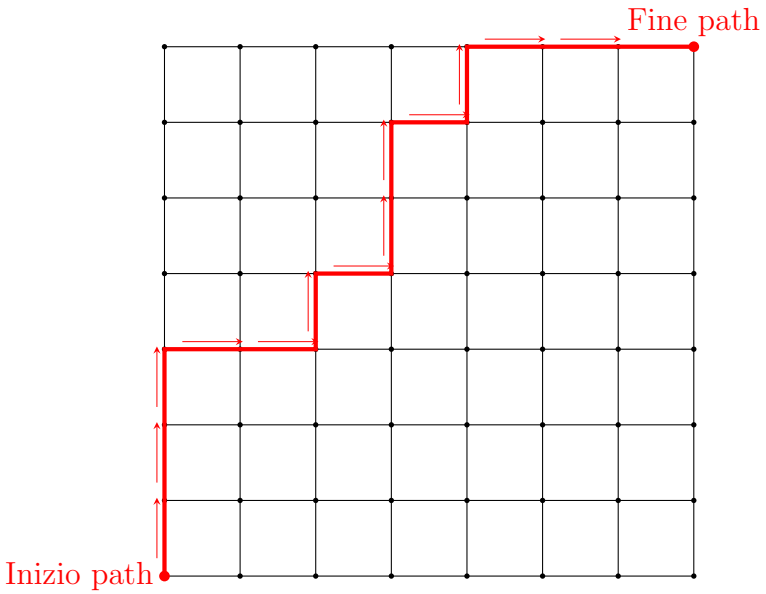

Figura 4.6: Reticolo 7×7 con path corrispondente al grafo lineare in fig. 5.18

A titolo di esempio, se consideriamo il grafo lineare in fig 5.18 il path corrispondente a tale grafo è mostrato in fig 4.6.

La stringa associata al path in figura 4.6 è UUURRURUURURRR.

4.2 Random Linear Graph

4.2.1 Generazione uniforme di oggetti combinatorici

Consideriamo una qualsiasi famiglia di oggetti combinatorici e poniamoci il problema di definire un modello di generazione uniforme di un oggetto da tale famiglia. Un approccio a tale problema potrebbe essere il seguente:

consideriamo l'albero associato alla famiglia di oggetti in cui ci sono tante foglie quanti sono gli oggetti nella famiglia. Ogni cammino dalla radice ad una foglia definisce la costruzione di un oggetto nella famiglia. A ogni nodo interno dell'albero sono associate delle decisioni. Un path che parte dalla radice e termina in una foglia corrisponde ad una decisione per ogni nodo interno attraversato. Poichè esiste una corrispondenza 1 a 1 tra l'insieme degl'oggetti della famiglia e l'insieme di tutti i possibili path nell'albero abbiamo che una generazione uniforme di path corrisponde ad una generazione uniforme degl'oggetti. Un modo per generare path in modo uniforme potrebbe essere il seguente: possiamo associare ad ogni arco (u, v) dell'albero una determinata probabilità. Sia $c(u)$ il numero di foglie nel sottoalbero radicato in u, allora la probabilità sopra menzionata è:

$$p_{u,v} = \frac{c(v)}{c(u)}$$

In questo modo ogni path e quindi ogni oggetto avrà probabilità di essere generato $\frac{1}{c(t)}$ dove t è la radice dell'albero. Sia $\mathbf{p} = (t, t_1, t_2, t_3 \ldots, t_{k-2}, t_{k-1}, t_k)$ la probabilità $\mathbb{P}(\mathbf{p})$ di questo path è:

$$\begin{aligned}\mathbb{P}(\mathbf{p}) =& p_{t,t_1} \times p_{t_1,t_2} \times p_{t_2,t_3} \times \cdots \times p_{t_{k-2},t_{k-1}} \times p_{t_{k-1},t_k} \\ =& \frac{c(t_1)}{c(t)} \times \frac{c(t_2)}{c(t_1)} \times \frac{c(t_3)}{c(t_2)} \times \cdots \times \frac{c(t_{k-1})}{c(t_{k-2})} \times \frac{1}{c(t_{k-1})} = \frac{1}{c(t)}\end{aligned}$$

Osservazione 4.2.1. Notiamo che se t_k è una foglia allora $c(t_k) = 1$

Una volta definite le probabilità per ogni edge la generazione di una path e quindi di un oggetto della famiglia consiste nella simulazione delle variabili casuali associate ai nodi dell'albero. In pratica si parte dalla radice, definiamo una variabile casuale che ha tanti possibili esiti quanti sono i figli della radice. Ogni esito ha una probabilità pari a quella associata all'edge corrispondente e dopodichè di effettua la simulazione della variabile casuale. In base all'esito della simulazione si raggiunge un nodo piuttosto che un altro. Su tale nodo raggiunto si definisce una nuova variabile casuale e si ripete il raggionamento fin quando non si raggiunge un nodo foglia.

Osserviamo che il metodo appena descritto è estremamente generico, le uniche cose necessarie per poterlo utilizzare sono le seguenti:

1. Definire l'albero associato alla famiglia di oggetti combinatorici.
2. Computare $c(u)$ con u un generico nodo dell'albero.

La generazione di un path $t, t_1, \ldots, t_{k-1}, t_k$ comporta il calcolo delle probabilità ad ogni nodo attraversato, quindi se il calcolo delle probabilità ad un generico un nodo x ha un costo di $\mathcal{O}(w(x))$ il costo della generazione del path è dell'ordine di $\mathcal{O}(w(x)k)$.

È possibile, inoltre computare la complessità media della generazione di un path. Infatti poichè in generale non tutti i path sono della stessa lunghezza abbiamo che la generazione di un path può essere più veloce della generazione di un altro path. Se denotiamo con U l'insieme di tutti i possibili path e con $\ell(.)$ la lunghezza di un generico path, risulta che la lunghezza media di un path è:

$$\sum_{i \in U} \ell(i) \frac{1}{c(t)} = \frac{1}{c(t)} \sum_{i \in U} \ell(i)$$

Sia n la lunghezza del path più lungo nell'albero, Y = lunghezza path generato. Y è una variabile aleatoria in $\{1, \ldots, n\}$. Se siamo in grado di contare il numero di path di una data lunghezza j, per ogni $j = 1, \ldots, n$ sia $a(j)$ = numero di path di lunghezza j, allora risulta che

$$Y = \begin{pmatrix} 1 & 2 & \cdots & n \\ \frac{a(1)}{c(t)} & \frac{a(2)}{c(t)} & \cdots & \frac{a(n)}{c(t)} \end{pmatrix}$$

In quanto tutti i path hanno la stessa probabilità di essere emessi. Quindi:

$$E[Y] = \sum_{i=1}^{n} i \frac{a(i)}{c(t)} = \frac{1}{c(t)} \sum_{i=1}^{n} i a(i)$$

Quindi se siamo in grado di computare $a(i)$ per ogni $i = 1, \ldots, n$ possiamo allora computare al complessità media della generazione.

4.2.2 Generazione uniforme di grafi lineari

Per poter utilizzare l'approccio di generazione basato sull'albero, occorre definire l'albero dei grafi lineari di dimensione n e essere in grado di contare il numero di foglie nel sottoalbero radicato ad un generico nodo interno dell'albero fornito.

L'albero associato ai grafi lineari con n nodi è un albero con n livelli uno per ogni $u(i)$. Ogni nodo ha tanti figli quanti sono i possibili valori che può assumere $u(i)$, i quali chiaramente sono strettamente dipendenti da $u(i-1)$.

Per questo albero contare il numero di foglie nel sottoalbero radicato in un generico nodo interno ad un livello $i+1 = 2, \ldots, n$, caratterizzato dal fatto di avere un edge col nodo padre etichettato con j, corrisponde al domandarsi quanti sono i grafi lineari che hanno $u(i) = j$. Per la proposizione 4.1.3 possiamo trasportare il ragionamento sui path. Dobbiamo quindi determinare quanti sono i possibili path che partono da $(i, u(i))$ e terminano in (n, n). Tale numero di path viene contanto dalla seguente formula:

$$\binom{2n-i-u(i)}{n-i} - \binom{2n-i-u(i)}{n-i+1} \tag{4.2}$$

La quale può essere riscritta nel seguente modo:

$$\begin{aligned}
&= \binom{2n-i-u(i)}{n-i} - \binom{2n-i-u(i)}{n-i+1} \\
&= \frac{(2n-i-u(i))!}{(n-i)!(n-u(i))!} - \frac{(2n-i-u(i))!}{(n-i+1)!(n-u(i)-1)!} \\
&= \frac{(2n-i-u(i))!}{(n-i)!(n-u(i))!} - \frac{(2n-i-u(i))!(n-u(i))}{(n-i+1)(n-i)!(n-u(i)-1)!(n-u(i))} \\
&= \binom{2n-i-u(i)}{n-i}\left(1 - \frac{(n-u(i))}{(n-i+1)}\right) \\
&= \binom{2n-i-u(i)}{n-i}\frac{u(i)-i+1}{n-i+1}
\end{aligned}$$

Possiamo osservare che in un certo senso la formula 4.2 generalizza la formula dei numeri di Catalan. Infatti se poniamo $i = 0$ e $u(i) = 0$ riotteniamo la formula dei numeri di Catalan. Possiamo interpretare questa

generalizzazione nel seguente modo: mentre la formula dei numeri di Catalan conta il numero di path che stanno al di sotto (o equivalentemente al di sopra) della diagonale (i, i) che partono da $(0, 0)$ e arrivano in (n, n), la formula 4.2 conta sempre lo stesso tipo di path che però possono partire in un punto $(i, u(i))$ tale che $i \leq u(i)$.

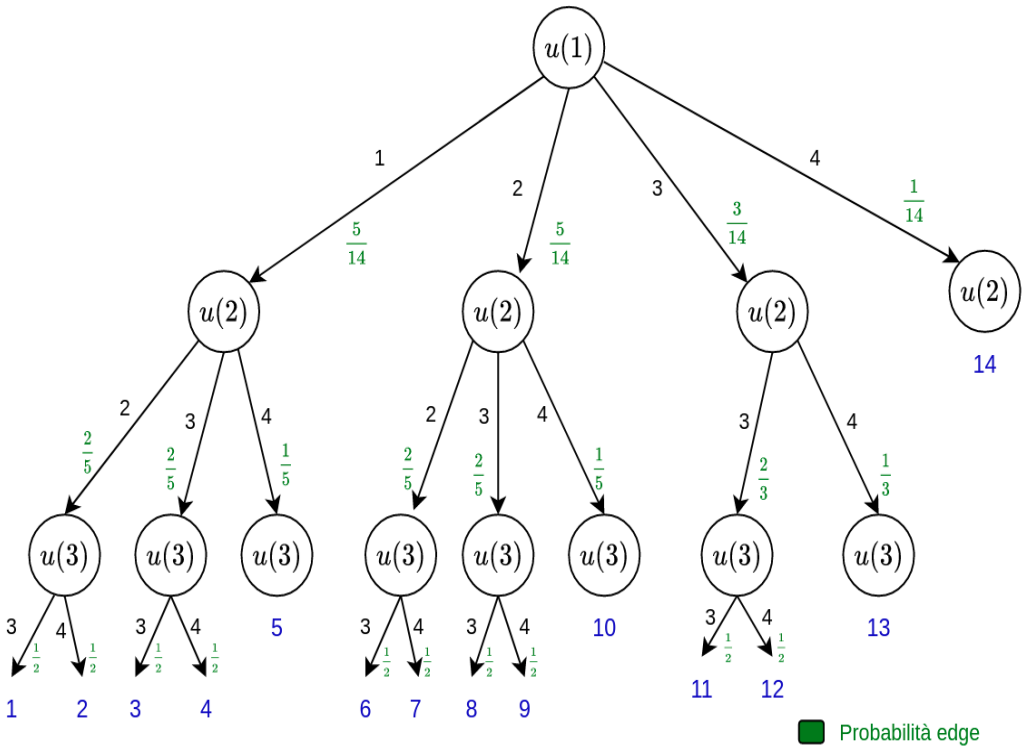

Figura 4.7: Albero grafi lineari per $n = 4$.

A titolo di esempio applicchiamo la formula 4.2 all'albero associato ai grafi lineari per $n = 4$ riportato in figura 4.7. Se volessimo contare il numero di foglie nel sottoalbero radicato nel nodo interno al livello 2 avente l'edge col nodo padre etichettato con 1. Dobbiamo utilizzare la formula per $i = 1$ e $u(1) = 1$:

$$\begin{aligned}\binom{2n-i-u(i)}{n-i} - \binom{2n-i-u(i)}{n-i+1} &= \binom{8-1-1}{4-1} - \binom{8-1-1}{4-1+1} \\ &= \binom{6}{3} - \binom{6}{4} \\ &= 20 - 15 = 5\end{aligned}$$

Se invece volessimo contare il numero di foglie nel sottoalberto radicato nel nodo interno al livello 2 avente l'edge col nodo padre etichettato con 3. Dobbiamo utilizzare la formula per $i = 1$ e $u(1) = 3$:

$$\begin{aligned}\binom{2n-i-u(i)}{n-i} - \binom{2n-i-u(i)}{n-i+1} &= \binom{8-1-3}{4-1} - \binom{8-1-3}{4-1+1} \\ &= \binom{4}{1} - \binom{4}{4} \\ &= 4 - 1 = 3\end{aligned}$$

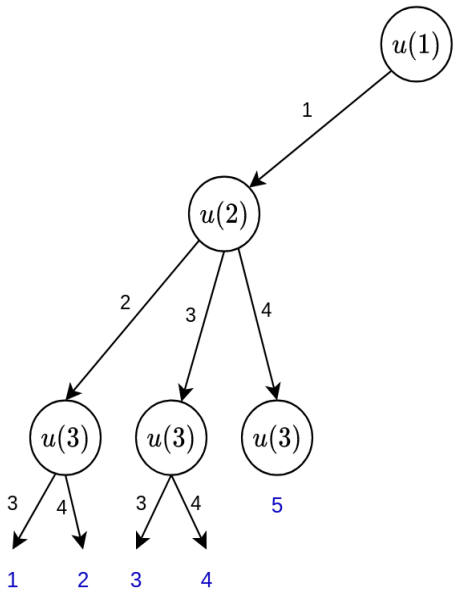

Figura 4.8: Sottoalbero nodo interno al livello 2 avente l'edge col nodo padre etichettato con 1.

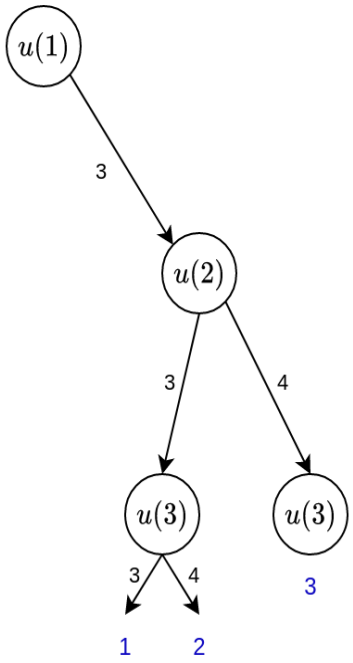

Figura 4.9: Sottoalbero nodo interno al livello 2 avente l'edge col nodo padre etichettato con 3.

Osservazione 4.2.2. L'albero associato ai grafi lineari definisce una codifica prefisso di quest'ultimi.

Osservazione 4.2.3. Possiamo notare che ogni simbolo che leggiamo nella decodifica della codifica di un dato grafo lineare ci da un certa quantità di informazione e che più questo simbolo è grande e più informazioni riceviamo perchè più il simbolo è grande e meno grafi con quella codifica esistono e quindi meno siamo incerti.

4.3 Oggetti ordinabili e oggetti ordinati

Definizione 4.3.1. Oggetti ordinabili

Un insieme di elementi U è un insieme di oggetti ordinabili se e solo se esistono su U:

1. una relazione $R_{\leq}$ d'ordine totale su U
2. una funzione di distanza d definita su U, $d(x, y)$ $\forall x, y \in U$

3. $\forall x, y, z \in U$: se $xR_{\leq}y$ e $yR_{\leq}z$ allora $d(x, y) \leq d(x, z)$ e $d(y, z) \leq d(x, z)$

Chiameremo invece un insieme di oggetti ordinabili, per cui sono noti $R_{\leq}$ e d, insieme di oggetti ordinati.

Definizione 4.3.2. Oggetti ordinati
Una tripla del tipo $(U, R_{\leq}, d)$, è un insieme di oggetti ordinati se si verificano le seguenti condizioni.
Dove U è un generico insieme di elementi e:

1. $R_{\leq}$ è una relazione d'ordine totale su U
2. d è una funzione di distanza definita su U, $d(x, y)$ $\forall x, y \in U$
3. $\forall x, y, z \in U$: se $xR_{\leq}y$ e $yR_{\leq}z$ allora $d(x, y) \leq d(x, z)$ e $d(y, z) \leq d(x, z)$

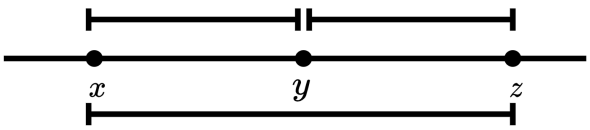

Figura 4.10: Relazione tra $R_{\leq}$ e d oggetti ordinati

Una conseguenza del punto 3 della definizione 4.3.2 è che la distanza massima tra due oggetti in U è $d(m(U), M(U))$, dove $m(U)$ e $M(U)$ sono rispettivamente il minorante e il maggiorante di U rispetto la relazione d'ordine $R_{\leq}$.

Notiamo che un esempio di oggetti ordinabili può essere l'insieme dei numeri reali, con funzione di distanza la distanza in valore assoluto.

4.4 G_{β} similarity graph

Definizione 4.4.1. G_{β} similarity graph
Sia U un insieme di elementi su cui è definita una funzione di distanza e sia β una soglia di similarità, prende il nome di β similarity graph, il grafo $G_{\beta} = (V, E)$:

- $V = U$
- $E = \{ (u, v) \mid d(u, v) \leq \beta, \forall u, v \in U \}$

Le seguenti due proposizioni mostrano le relazioni esistenti tra i grafi lineari, gli oggetti ordinati e i β similarity graph.

Proposizione 4.4.1. Un G_β similarity graph costruito a partire da un insieme di oggetti ordinati è un grafo lineare.

Dimostrazione. Sia $G_\beta = (V, E)$ un β similarity graph costruito a partire da un insieme di oggetti ordinati $(U, R_\leq, d)$. Dal momento che $V = U$ possiamo ordinare i nodi di G_β secondo la relazione d'ordine $R_\leq$ su U. Ciò che dobbiamo fare è dimostrare che tale ordinamento rispetta le proprietà di un grafo lineare. Per ogni j, k se $(v_{\pi(j)}, v_{\pi(k)}) \in E$ con $j < k$ abbiamo che $d(v_{\pi(j)}, v_{\pi(k)}) \leq \beta$ di conseguenza $\forall r : j \leq r \leq k$, ovvero $v_{\pi(j)} R_\leq v_{\pi(r)}$ e $v_{\pi(r)} R_\leq v_{\pi(k)}$ da cui segue per la proprietà (3) degl'oggetti ordinabili che $d(v_{\pi(j)}, v_{\pi(r)}) \leq d(v_{\pi(j)}, v_{\pi(k)})$. Da ciò segue che anche $d(v_{\pi(j)}, v_{\pi(r)}) \leq \beta$ ovvero $(v_{\pi(j)}, v_{\pi(r)}) \in E$. Si può procedere in maniera analoga per il lato sinistro. □

Proposizione 4.4.2. Sia $G = (V, E)$ un grafo lineare è possibile costruire a partire da G un insieme di oggetti ordinati $(U, R_\leq, d)$.

Dimostrazione. Dato un grafo lineare G con ordinamento $\pi(.)$, possiamo costruire un insieme di oggetti ordinati $(U, R_\leq, d)$ siffatto. Poniamo $U = V$ e $R_\leq = \pi(.)$. Dopodichè per costruire la funzione di distanza ragioniamo nel seguente modo: dati due generici nodi $i, j \in V$ possiamo avere due possibili casi, $(i, j) \in E$ oppure $(i, j) \notin E$. Nel primo caso posiamo $d(v_i, v_j) = 1$, mentre nel secondo caso poniamo $d(v_i, v_j) = \infty$. □

CAPITOLO 5

CLUSTERING DELETION SU GRAFI LINEARI

Questo capitolo tratta lo studio e l'analisi di vari algoritmi per il problema del Clustering Deletion per grafi lineari. Nella sezione 5.1 sono analizzati alcuni approcci greedy al problema. Nella sezione 5.2 viene descritto un algoritmo risolutivo basato su programmazione dinamica. Nella sottosezione 5.2.1 viene introdotta e dimostrata una proprietà importante a cui si deve l'ottimalità dell'algoritmo di programmazione dinamica, mentre nelle sottosezioni 5.2.2, 5.2.3 viene prima descritto l'algoritmo di programmazione dinamica e la relazione di ricorrenza ad esso associata e dopodichè viene analizzata la complessità dell'algoritmo proposto.

5.1 Approcci greedy

L'ordinamento sui vertici di un grafo lineare porta spesso e volentieri ad algoritmi greedy ottimali [10] su molti problemi algoritmici quali ad esempio l'individuazione della massima clique, il problema della colorazione, ecc.

Per queste ragioni è ragionevole chiedersi se anche per il problema del Clustering Deletion sia così, ovvero esista un algoritmo greedy ottimo. Di seguito verrano analizzati alcuni approcci greedy al problema.
Gli approcci greedy analizzati sono:

- Approccio greedy basato sulla clique massima del grafo,
- Approccio greedy basato sulla clique massima a partire da destra.

5.1.1 Approccio greedy basato sulla clique massima del grafo

L'approccio greedy basato sulla clique massima del grafo consiste nel: dato il grafo lineare individuare la clique massima, il quale ricordiamo è un problema facile da risolvere per questi grafi, e dopodichè eliminarla dal grafo ottenendo così un grafo lineare di dimensioni ridotte. Iterare questo processo fino a quando non consumiamo tutti i nodi del grafo. Le clique che vengono a mano a mano eliminate dal grafo iniziale vanno a comporre il cluster graph di output che quindi per costruzione è banalmente un insieme disgiunto di clique.

L'algoritmo 9 implementa l'approccio greedy basato sulla massima clique. Abbiamo argomententato che l'algoritmo 9 è corretto, dovremmo in teoria ora provare l'ottimalità. Purtroppo però tale algoritmo è in generale non ottimo, in quanto un possibile grafo lineare su cui non produce la soluzione ottima è il grafo in figura 5.1.

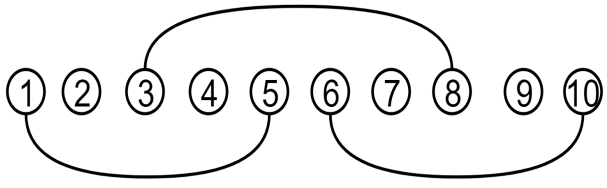

Figura 5.1: Grafo lineare controesempio approccio greedy massima clique

Algorithm 9: CDGL Approccio greedy massima clique

Input:

- Grafo $G = (V, E)$
- $v_0, \ldots, v_{n-1}$

Output:

- P partizione di V il cui sottografo indotto definisce il cluster graph richiesto

1: $T = V$
2: $K = E$
3: $P = \emptyset$
4: **while** $T \neq \emptyset$ **do**
5: $\quad (V', E') = CliqueMax((T, K))$
6: $\quad P = P \cup \{ V' \}$
7: $\quad T = T - V'$
8: $\quad K = K - E'$
9: **end while**

La clique massima di tale grafo è l'insieme $\{ 3, 4, 5, 6, 7, 8 \}$ perciò l'algoritmo individua il cluster graph, riportato in figura 5.2:

$$P = \{ \{ 1, 2 \}, \{ 3, 4, 5, 6, 7, 8 \}, \{ 9, 10 \} \}$$

$$f(P) = 2 * 3 + 3 * 2 = 12$$

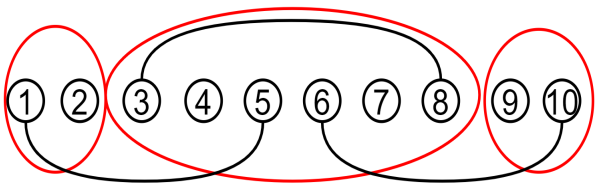

Figura 5.2: Soluzione algoritmo greedy max clique

Il cluster graph ottimale è invece, riportato in figura 5.3:

$$P^* = \{ \{ 1, 2, 3, 4, 5 \}, \{ 6, 7, 8, 9, 10 \} \}$$

$$f(P^*) = 3 * 3 = 9$$

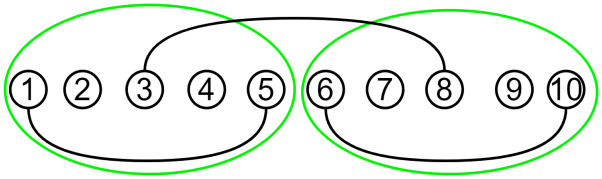

Figura 5.3: Soluzione ottima

In conclusione l'approccio greedy basato sulla massima clique non è ottimo. In [14] viene però provato che per tutti quei grafi per cui il problema della massima clique è un problema facile l'approccio greedy basato sulla massima clique è un algoritmo *2-approssimato*. Poichè il problema della massima clique per i grafi lineari è un problema facile segue che l'algoritmo greedy proposto, suppur non ottimo, è comunque un algoritmo *2-approssimato*.

5.1.2 Approccio greedy basato sulla clique massima a partire da destra

Consideriamo ora un altro approccio basato sulla clique massima. In tale approccio dato un grafo lineare il cui ordinamento dei nodi è $v_0, v_1, \ldots, v_{n-1}$, invece di prendere la clique più grande in assoluto nel grafo, prendiamo la clique più grande nel grafo che contiene il nodo v_{n-1}. Da qui il nome approccio greedy basato sulla clique massima a partire da destra, in quanto il nodo v_{n-1} è il nodo più a destra nell'ordinamento.

Osservazione 5.1.1. Notiamo che determinare la massima clique del nodo v_i più a destra nell'ordinamento è un problema facile da risolvere. In particolare se abbiamo a disposizione gli $l(i)$ la possiamo individuare in tempo $\mathcal{O}(1)$.

Individuata tale clique, procediamo in maniera analoga all'approccio greedy basato sulla massima clique, quindi la eliminiamo dal grafo e la inseriamo nel cluster graph di output. Iteriamo il procedimento fin quando non finiscono tutti i nodi. Tale approccio è chiaramente corretto, ovvero restituisce

un insieme disgiunto di clique, ma purtroppo non è ottimo. Tale approccio non è ottimo in quanto se consideriamo il grafo in figura 5.4 abbiamo che la soluzione fornita dall'algoritmo greedy è

$$P = \{\{1, 2, 3\}, \{4, 5\}\}$$

$$f(P) = 3 * 1 = 4$$

riportata in figura 5.5, mentre il cluster graph ottimale è

$$P = \{\{1, 2, 3, 4\}, \{5\}\}$$

$$f(P) = 1 = 1$$

riportato in figura 5.10.

Figura 5.4: Grafo lineare

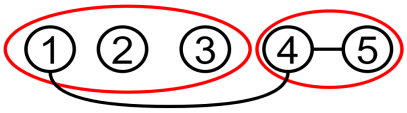

Figura 5.5: Soluzione non ottima approccio greedy

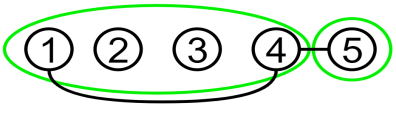

Figura 5.6: Soluzione ottima

Dal controesempio fornito si potrebbe pensare un'ulteriore approccio greedy, il quale determina un cluster graph in maniera greedy partendo da destra e inoltre ne determina un'altro sempre in maniera greedy a partire da sinistra e tra i due ritorna il migliore. Anche tale approccio non è ottimo infatti

se consideriamo il grafo in figura 5.7 abbiamo che il cluster graph determinato in maniera greedy partendo da destra è

$$P = \{\{1\}, \{2,3,4,5,6,7\}, \{8,9,10\}\}$$

$$f(P) = 2 + 2 * 6 = 14$$

riportato in figura 5.8, mentre il cluster graph determinato in maniera greedy partendo da sinistra è

$$P = \{\{1,2,3\}, \{4,5,6,7,8,9\}, \{10\}\}$$

$$f(P) = 6 * 2 + 2 = 14$$

riportato in figura 5.9.

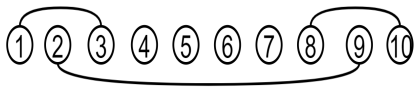

Figura 5.7: Grafo lineare controesempio approccio greedy massima clique da sinistra e da destra

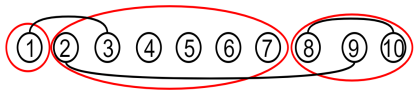

Figura 5.8: Soluzione approccio greedy da destra

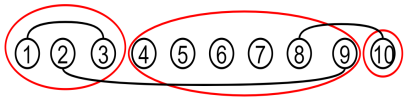

Figura 5.9: Soluzione approccio greedy da sinistra

Notiamo che hanno lo stesso valore di funzione obiettivo, e quindi è indifferente prendere uno o l'altro. In particolare nessuno dei due è ottimo, in quanto il cluster graph ottimale è

$$P = \{\{1\}, \{2,34,5,6,7,8,9\}, \{10\}\}$$

$$f(P) = 2 + 2 = 4$$

riportato in figura 5.10.

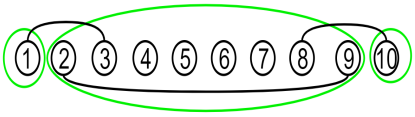

Figura 5.10: Soluzione ottima

Problema aperto 5.1.1. Sia $G = (V, E)$ un grafo lineare il cui ordinamento è $v_0, \ldots, v_{n-1}$ e sia C^* la sua clique massima. Per la natura dei grafi lineari le clique massime sono insiemi contigui. Se il minorante o il maggiorante di C^* fossero rispettivemente v_0 oppure v_{n-1}, ovvero se la clique massima si trovasse all'estrama sinistra oppure all'estrema destra dell'ordinamento $v_0, \ldots, v_{n-1}$ del grafo lineare è possibile affermare che allora tale clique sicuramente appartiene al cluster graph ottimale? Se così fosse, ogni qualvolta ci trovassimo in tale circostanza, sarebbe possibile semplificare il grafo: evitando così molti sottoproblemi e raggiungendo di conseguenza prima l'ottimo.

5.2 Approccio a programmazione dinamica

In questa sezione viene studiato e analizzato un approccio basato su programmazione dinamica alla risoluzione del problema del Clustering Deletion su grafi lineari. Nella sottosezione 5.2.1 viene introdotta e provata una proprietà sulla struttura dei cluster graph ottimali, che verrà utilizzata nella sottosezione 5.2.2 per dimostrare l'ottimalità dell'algoritmo di programmazione dinamica proposto descritto sempre nella suddetta sottosezione.

5.2.1 Ottimalità partizione contigua

Definizione 5.2.1. Maggiorante e minorante
Sia V un insieme su cui è definito un ordinamento $\pi(.)$ e $S \subseteq V, S \neq \emptyset$; si dice che un'elemento $s \in S$ [15]:

- è un maggiorante di S, rispetto π, se per ogni $v \in S$ si ha $\pi(v) \leq \pi(s)$,

- è un minorante di S, rispetto π, se per ogni $v \in S$ si ha che $\pi(s) \leq \pi(v)$.

Denoteremo il maggiorante e il minorante di S rispettivamente con $M(S)$ e $m(s)$.

Definizione 5.2.2. Sottoinsieme contiguo rispetto un ordinamento
Sia V un insieme su cui è definito un ordinamento $\pi(.)$. Un sottoinsieme S di V si dice contiguo rispetto l'ordinamento $\pi(.)$ se e solo se:

$$\forall i \text{ tale che } m(S) \leq v_{\pi(i)} \leq M(S) \text{ vale che } v_{\pi(i)} \in S$$

Esempio 5.2.1. Sia $V = \{0, 1, 2, 3, 4, 5, 6\}$ un insieme di interi ordinati secondo l'usuale ordinamento su numeri. Sia $S = \{1, 2, 3, 4\}$ e $S' = \{1, 3, 5\}$ due sottoinsiemi di V. Determiniamoci il minorante e il maggiorante di S e S':

- $m(S) = 1$, $m(S') = 1$
- $M(S) = 4$, $M(S') = 5$

Notiamo che ogni elemento compreso tra $m(S) = 1$ e $M(S) = 4$ appartiene a S quindi S è un sottoinsieme contiguo rispetto l'ordinamento, mentre dal momento che esiste un elemento compreso tra $m(S') = 1$ e $M(S') = 5$ che non appartiene a S', in particolare mancano sia 2 che 4, abbiamo che S' non è un sottoinsieme contiguo.

Definizione 5.2.3. Partizione contigua rispetto un ordinamento
Sia V un insieme su cui è definito un ordinamento $\pi(.)$, una partizione P di V si dice contigua rispetto l'ordinamento $\pi(.)$ se ogni sottoinsieme S di P è contiguo rispetto $\pi(.)$.

Esempio 5.2.2. Sia $V = \{0, 1, 2, 3, 4, 5, 6\}$ un insieme di interi ordinati secondo l'usuale ordinamento su numeri. Consideriamo le seguenti due partizioni di V:

$$P = \{\{0\}, \{1, 2, 3, 4, 5\}, \{6\}\}$$

$$P' = \{\{0, 2, 3\}, \{1, 4\}, \{5, 6\}\}$$

È facile verificare che ogni sottoinsieme in P di V è contiguo e quindi P è una partizione contigua, mentre non è vero che ogni sottoinsieme in P' di V è contiguo e di conseguenza P' non è contiguo.

Sia $G = (V, E)$ un grafo lineare e $\pi(.)$ il suo ordinamento dei nodi. Consideriamo i seguenti 3 insiemi a G associati:

- S l'insieme di tutte le possibili partizioni di V
- S' l'insieme di tutte le possibili partizioni V contigue rispetto $\pi(.)$
- S^* l'insieme di tutte le partizioni V ottimali per il problema del Clustering Deletion

Abbiamo che le relazioni che possono sussistere tra questi tre insiemi sono 3 e sono riportate nella figura 5.16.

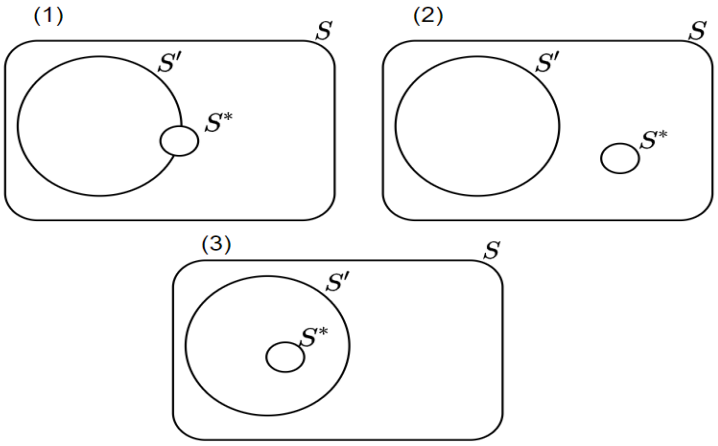

Figura 5.11: Possibili relazioni tra le soluzioni per il Clustering Deletion su grafi lineari

Questa osservazione è fondamentale in quanto se si potesse provare che una tra la (1) e la (3) è sempre verificata allora sarebbe possibile cercare la soluzione ottima per il problema del Clustering Deletion su un grafo lineare, cercando nello spazio delle soluzioni ridotto costituito dalle sole partizioni contigue. Ciò permetterebbe, altresì, di poter sfruttare la struttura di questo spazio delle soluzioni per progettare un algoritmo efficiente che lo esplori al fine di trovare l'ottimo in tempi ragionevoli.

Osservazione 5.2.1. Chiaramente la (3) è una condizione più forte ma ai fini della ricerca dell'ottimo la (1) e la (3) sono equivalenti.

Definizione 5.2.4. Funzione $g(.,.)$
Sia $G = (V, E)$ un grafo e siano S e S' due generici sottoinsiemi disgiunti di V. Denotiamo con:

$$g(S, S') = \text{numero di edge con un estremo in } S \text{ e uno in } S'$$

Osservazione 5.2.2. Si può osservare che la funzione obiettivo del Clustering Deletion su un generico grafo può essere scritta in funzione di $g(.,.)$. Infatti se $G = (V, E)$ è un generico grafo, P è partizione ammissibile di V, ovvero è un insieme disgiunto di clique e $f(P)$ la funzione obiettivo abbiamo che:

$$f(P) = \sum_{(A,B)\in P} g(A, B)$$

Proposizione 5.2.1. Proprietà funzione $g(.,.)$
Siano A e B due insiemi disgiunti di nodi e P_A e P_B le rispettive partizioni, allora vale che:

$$g(A, B) = \sum_{a\in P_A} \sum_{b\in P_B} g(a, b)$$

Proposizione 5.2.2. Swap di elementi tra due insiemi
Sia P una partizione di nodi. Siano $A, B \in P$ e siano $U \subseteq A$ e $W \subseteq B$ rispettivamente gli elementi di A spostati in B e gli elementi di B spostati in A. Quindi se indichiamo con A' e B' i nuovi insiemi ottenuti a seguito dello swap, risulta:

$$A' = (A - U) \cup W$$

$$B' = (B - W) \cup U$$

Sia $Z \in P$ con $A \neq Z \neq B$ allora vale:

$$g(A, Z) + g(B, Z) = g(A', Z) + g(B', Z)$$

Dimostrazione. Poichè $U \subseteq A$ e $W \subseteq B$ possiamo riscrivere A e B nel seguente modo:

$$A = (A - U) \cup U$$

$$B = (B - W) \cup W$$

Da cui segue che:

$$g(A, Z) + g(B, Z) = g(A - U, Z) + g(U, Z) + g(B - W, Z) + g(W, Z) \quad (5.1)$$

il che è chiaramente uguale a:

$$g(A', Z) + g(B', Z) = g(A - U, Z) + g(W, Z) + g(B - W, Z) + g(U, Z) \quad (5.2)$$

Notiamo che sia in 5.1 che in 5.2 abbiamo utilizzato la proposizione 5.2.1. □

Proposizione 5.2.3. Sia P una soluzione ammissibile per il problema del Clustering Deletion, ovvero una partizione il cui sottografo indotto è cluster graph. Siano $S, S' \in P$, la funzione obiettivo può essere riscritta nel seguente modo:

$$f(P) = \sum_{\substack{(A,B)\in P \\ A,B\neq S \\ A,B\neq S'}} g(A, B) + \sum_{\substack{A\in P \\ A\neq S'}} g(A, S) + \sum_{\substack{A\in P \\ A\neq S}} g(A, S') + g(S, S')$$

Se spostiamo nodi da S a S' o viceversa, risulta che $f(P) - g(S, S')$ non varia, ma cambia solamente $g(S, S')$. Questo perchè chiaramente la prima sommatoria coinvolgendo solo insiemi diversi sia da S che S' non viene minimamente intaccata. La seconda e la terza vengono modificate però per la proposizione precedente risulta che la loro somma non varia. Quindi in ultima istanza varia solo $g(S, S')$ e di conseguenza $f(P) - g(S, S')$ non varia.

Teorema 5.2.1. Sia G un grafo lineare con ordinamento sui nodi π e sia $P = \{ S_1, \ldots, S_m \}$ una partizione dei nodi di G:

Se P è ottimale allora P è una partizione contigua rispetto l'ordinamento π.

Dimostrazione. Sia G un grafo lineare, $P = \{ S_1, \ldots, S_m \}$ una partizione ottimale e $v_1, \ldots, v_n$ l'ordinamento sui nodi di G.

Se per assurdo P non fosse costituito da sottoinsiemi contigui vorrebbe dire che esiste un sottoinsieme S non contiguo. Se i_1 e j_1 sono rispettivamente la posizione del minorante e del maggiorante di S allora abbiamo che esiste un $k : i_1 < k < j_1$ con $v_k \notin S$. Sia S' il sottoinsieme tale che $v_k \in S'$, le cui posizioni del minorante e maggiorante sono rispettivamente i_2 e j_2. Abbiamo quindi la seguente situazione:

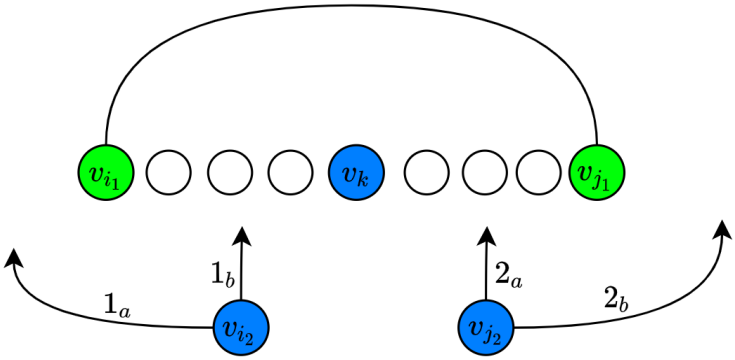

Figura 5.12: Situazione con i quattro possibili casi

Possiamo avere due casi per i_2:

- $1 \leq i_2 < i_1$ (1_a) oppure
- $i_1 < i_2 \leq k$ (1_b)

  E due casi per j_2:

- $k \leq j_2 < j_1$ (2_a) oppure
- $j_1 < j_2 \leq n$ (2_b)

Detto ciò, abbiamo in tutto quattro possibili casi e dobbiamo provare che in ognuno di essi si raggiunge un assurdo. Per farlo, in virtù della proposizione 5.2.3, se effettuiamo uno swap di elementi tra S e S' per provare che la nuova partizione risultante da tale operazione ha un valore migliore basta provare che $g(S, S')$ diminuisce.

Caso $(1_a, 2_b)$**:**

La situazione associata al caso $(1_a, 2_b)$ è riportata in figura 5.13. In questo

caso si raggiunge facilmente un assurdo in quanto S, S' formano una clique, quindi potremmo unire i due insiemi e costruire un unica clique, per cui $g(S \cup S', \emptyset) = 0$.

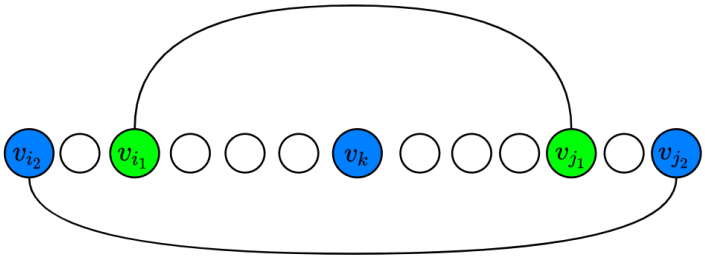

Figura 5.13: Caso $(1_a, 2_b)$

Caso $(1_b, 2_a)$**:**
La situazione associata al caso $(1_b, 2_a)$ è riportata in figura 5.14. In questo caso la contraddizione si raggiunge in maniera analoga al caso $(1_a, 2_b)$.

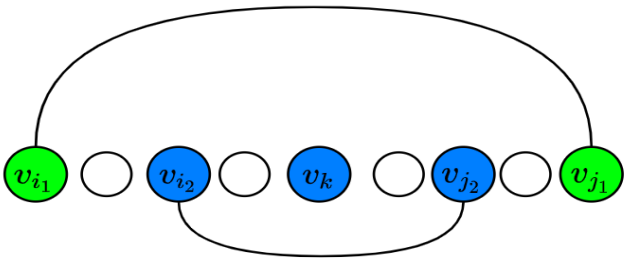

Figura 5.14: Caso $(1_b, 2_a)$

Caso $(1_b, 2_b)$**:**
La situazione associata al caso $(1_b, 2_b)$ è riportata in figura 5.15.

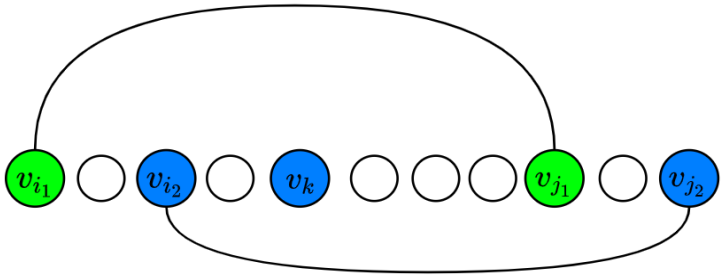

Figura 5.15: Caso $(1_b, 2_b)$

Per questo caso occorre fare delle analisi ulteriori. Come prima cosa introduciamo i seguenti insiemi:

- $A = \{ v_z \in S \mid i_1 \leq z < i_2 \}$, con $|A| = t$
- $B_1 = \{ v_z \in S \mid i_2 \leq z \leq j_1 \}$, con $|B_1| = r$
- $B_2 = \{ v_z \in S' \mid i_2 \leq z \leq j_1 \}$, con $|B_2| = r'$
- $B = B_1 \cup B_2$
- $C = \{ v_z \in S' \mid j_1 < z \leq j_2 \}$, con $|C| = t'$

Da cui segue che $S = A \cup B_1$ e $S' = B_2 \cup C$ e di conseguenza $|S| = t + r$, $|S'| = t' + r'$, in quanto gli insiemi appena introdotti sono chiaramente a due a due disgiunti.

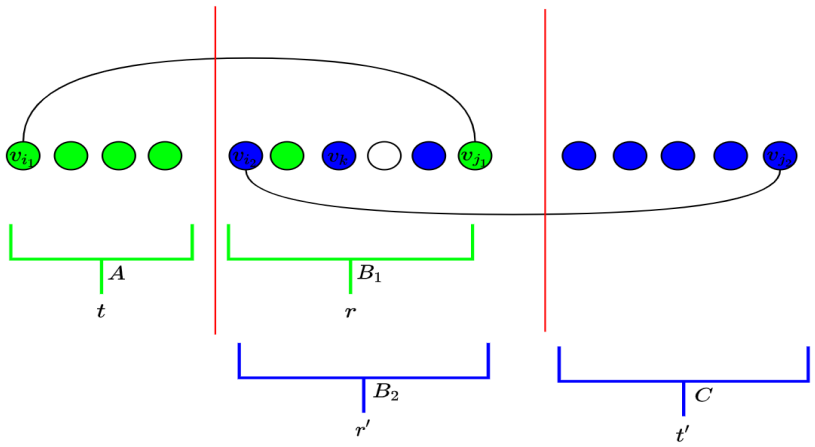

Figura 5.16: Caso $1_b, 2_b$ con insiemi A, B_1, B_2 e C

Possiamo innanzitutto osservare:

$$g(S, S') = g(A \cup B_1, B_2 \cup C) = g(A, B_2) + g(A, C) + g(B_1, B_2) + g(B_1, C)$$

Poichè $(v_{i_1}, v_{j_1}) \in E$ e $(v_{i_2}, v_{j_2}) \in E$, sappiamo che $A \cup B$ e $B \cup C$ formano due clique e di conseguenza segue che:

$$g(S, S') = tr' + g(A, C) + rr' + rt'$$

Dalle osservazioni fatte possiamo osservare che si potrebbe pensare di sostituire i due insiemi S e S' con una delle seguenti nuove coppie di insiemi $A \cup B$ e C oppure A e $B \cup C$. Vogliamo quindi dimostrare che:

$$g(S, S') > g(A \cup B, C) \text{ oppure } g(S, S') > g(A, B \cup C)$$

così facendo avremmo raggiunto l'assurdo cercato in quanto sarebbe possibile migliorare la soluzione ottima.
Ricapitolando abbiamo:

- $g(S, S') = tr' + rr' + rt' + g(A, C)$
- $g(A \cup B, C) = t'(r + r') + g(A, C)$
- $g(A, B \cup C) = t(r + r') + g(A, C)$

Possiamo dividere la prova in due casi:

- $t \geq t'$
- $t < t'$

Se $t \geq t'$ allora:

$$\begin{aligned} g(S, S') &= tr' + rr' + rt' + g(A, C) && \text{perchè } t \geq t' \\ &\geq t'r' + rr' + rt' + g(A, C) \\ &= t'(r + r') + rr' + g(A, C) && \text{perchè } rr' \geq 1 \\ &> t'(r + r') + g(A, C) = g(A \cup B, C) \end{aligned}$$

Se $t < t'$ allora:

$$\begin{aligned} g(S, S') &= tr' + rr' + rt' + g(A, C) && \text{perchè } t < t' \\ &> tr' + rr' + rt + g(A, C) \\ &= t(r + r') + rr' + g(A, C) && \text{perchè } rr' \geq 1 \\ &> t(r + r') + g(A, C) = g(A, B \cup C) \end{aligned}$$

Abbiamo quindi raggiunto la contraddizione.

Caso $(1_a, 2_a)$**:**

La situazione associata al caso $(1_a, 2_a)$ è riportata in figura 5.17. In questo caso la contraddizione può essere raggiunta ragionando in maniera simile al caso $(1_b, 2_b)$.

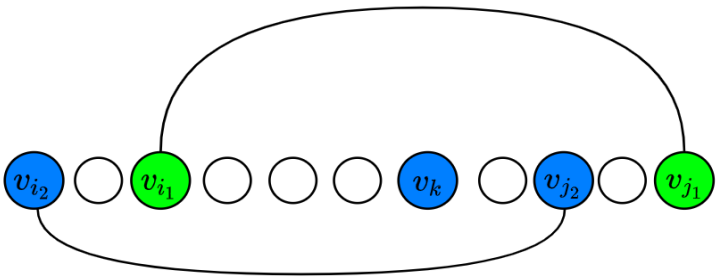

Figura 5.17: Caso $(1_a, 2_a)$

□

Osservazione 5.2.3. Per il teorema 5.2.1 possiamo affermare che tra le tre relazioni mostrate nella figura 5.16 sussiste la (3).

Ricapitolando il problema del Clustering Deletion su grafi lineari può essere riformulato per via del teorama 5.2.1 nel seguente modo.

Clustering Deletion su grafi lineari

Input:

- $G = (V, E)$ grafo lineare
- $v_0, \ldots, v_{n-1}$ ordinamento di $G = (V, E)$

Output: Una partizione P dei nodi contigua rispetto l'ordinamento $v_0, \ldots, v_{n-1}$:

- Il sottografo indotto da P è un cluster graph
- Il numero di edge che connettono due nodi in sottoinsiemi diversi è minimizzato

5.2.2 Algoritmo di programmazione dinamica

Come nel caso del Clustering Deletion su grafi generici anche qui possiamo considerare solo grafi lineari connessi. È possibile risolvere il problema del Clustering Deletion su grafi lineari in tempo polinomiale attraverso un algoritmo di programmazione dinamica.
Sia $G = (V, E)$ un grafo lineare il cui ordinamento è $v_0, v_1, \ldots, v_{n-1}$. I sottoproblemi sono individuati da $v_0, \ldots, v_i$, ovvero dal determinare il cluster graph ottimale per il sottografo indotto dall'insieme $\{ v_0, \ldots, v_i \}$. Introduciamo le seguenti notazioni:

- $p(i)$ soluzione al sottoproblema $v_0, \ldots, v_i$
- $S_{i,j} = \{ v_i, v_{i+1}, \ldots, v_{j-1}, v_j \}$ per $0 \leq i \leq j \leq n-1$, con la convenzione che $S_{i,j} = \emptyset$ altrimenti. Quindi ad esempio $S_{0,-1} = \emptyset$.

$$p(i) = \min_{\substack{t=i,\ldots,0 \\ S_{t,i} \text{ è una clique}}} \{p(t-1) + g(S_{0,t-1}, S_{t,i})\} \tag{5.3}$$

$$p(0) = p(-1) = 0 \tag{5.4}$$

Notiamo che utilizzando la notazione dei lower bound $l(.)$ possiamo riscrivere la relazione di ricorrenza nel seguente modo:

$$p(i) = \min_{t=i,i-1,\ldots,l(i)} \{p(t-1) + g(S_{0,t-1}, S_{t,i})\} \tag{5.5}$$

Questo poichè $S_{t,i}$ per ogni $t = i, \ldots, l(i)$ è una clique per definizione di $l(i)$. L'output per il grafo G è chiaramente $p(n-1)$.

Spesso è utile avere a disposizione sia la versione all'indietro di un relazione di ricorrenza sia una versione in avanti e utilizzare quella più naturale rispetto al problema che ci troviamo ad affrontare. Definiamo quindi la relazione di ricorrenza in avanti. Per fare ciò dobbiamo ragionare in maniera inversa a quanto precedentemente fatto. In questo caso invece di considerare $v_0, \ldots, v_i$ come sottoproblemi consideriamo $v_i, \ldots, v_{n-1}$. Quindi di conseguenza anche il significato di $p(i)$ varia, e diventa $p(i)$ soluzione al sottoproblema $v_i, \ldots, v_{n-1}$. La relazione di ricorrenza diventa quindi:

$$p(i) = \min_{\substack{t=i,\ldots,n-1 \\ S_{i,t} \text{ è una clique}}} \{g(S_{i,t}, S_{t+1,n-1}) + p(t+1)\} \tag{5.6}$$

$$p(n-1) = 0 \tag{5.7}$$

La quale in perfetta analogia con il caso precedente può essere riscritta utilizzando la notazione degl'upper bound $u(.)$:

$$p(i) = \min_{t=i,\ldots,u(i)} \{g(S_{i,t}, S_{t+1,n-1}) + p(t+1)\} \tag{5.8}$$

In questo caso l'output per il grafo G è $p(0)$.

Il ragionamento che sta dietro alla relazione di ricorrenza è il seguente. Considerando un generico sottoproblema di dimensione i, ciò che ci domandiamo è com'è fatto il sottoinsieme che contiene v_i nella partizione ottimale di $v_0, \ldots, v_i$? Per quanto provato tale sottoinsieme è un sottoinsieme

contiguo quindi sappiamo che il suo maggiorante è v_i ciò che non sappiamo è qual'è il suo minorante. Per determinarlo ciò che facciamo è considerare tutti i sottoinsiemi che sono clique (altrimenti non sarebbe una soluzione ammissibile) che hanno come maggiorante v_i. Tali sottoinsiemi sono chiaramente al più $i+1$:

$$S_{0,i}, S_{1,i}, S_{2,i}, \ldots, S_{i-1,i}, S_{i,i}$$

Da ciò segue che per individuare il sottoinsieme contiguo della soluzione ottimale possiamo determinare il valore associato ad ognuna di queste scelte e una volta fatto ciò prendere la migliore. Se ad esempio prendiamo il sottoinsieme $S_{t,i}$ vorra dire che il numero di edge che dovremo eliminare per ottenere un cluster graph è chiaramente $g(S_{0,t-1}, S_{t,i})$ più il valore della soluzione ottimale per il sottoproblema $v_0, \ldots, v_{t-1}$ così individuato, ovvero $p(t-1)$.

Esempio 5.2.3. Esecuzione algoritmo

Condieriamo il grafo lineare in figura 5.18. Nella tabella 5.1 sono riportati i valori di $g(.,.)$.

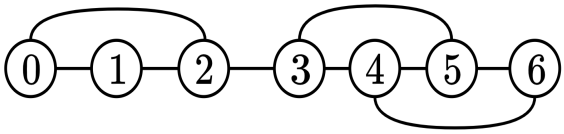

Figura 5.18: Grafo lineare

t \ i	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	-	1	2	0	0	0	0
2	-	-	2	2	0	0	0
3	-	-	-	1	1	0	0
4	-	-	-	-	1	1	2
5	-	-	-	-	-	1	3
6	-	-	-	-	-	-	2

Tabella 5.1: Funzione $g(S_{0,t-1}, S_{t,i})$ per il grafo in figura 5.18

$p(6) = min\{$

$p(5) + g(S_{0,5}, S_{6,6}) = 1 + 2 = 3,$

$p(4) + g(S_{0,4}, S_{5,6}) = 1 + 3 = 4,$

$p(3) + g(S_{0,3}, S_{4,6}) = 1 + 2 = 3$

$\} = 3$

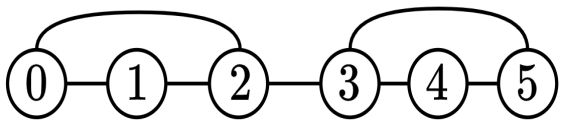

Figura 5.19: Sottografo indotto da $v_0, v_1, v_2, v_3, v_4, v_5$

$$p(5) = min\{$$
$$p(4) + g(S_{0,4}, S_{5,5}) = 1 + 1 = 2,$$
$$p(3) + g(S_{0,3}, S_{4,5}) = 0 + 1 = 1$$
$$\} = 1$$

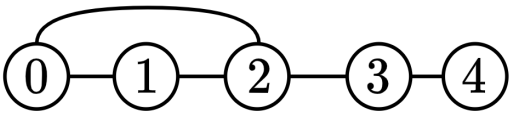

Figura 5.20: Sottografo indotto da v_0, v_1, v_2, v_3, v_4

$$p(4) = min\{$$
$$p(3) + g(S_{0,3}, S_{4,4}) = 1 + 1 = 2,$$
$$p(2) + g(S_{0,2}, S_{3,4}) = 0 + 1 = 1$$
$$\} = 1$$

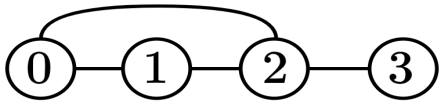

Figura 5.21: Sottografo indotto da v_0, v_1, v_2, v_3

$$p(3) = min\{$$
$$p(2) + g(S_{0,2}, S_{3,3}) = 0 + 1 = 1,$$
$$p(1) + g(S_{0,1}, S_{2,3}) = 0 + 2 = 2$$
$$\} = 1$$

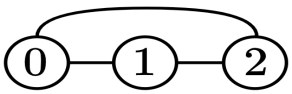

Figura 5.22: Sottografo indotto da v_0, v_1, v_2

$$p(2) = min\{$$
$$p(1) + g(S_{0,1}, S_{2,2}) = 0 + 2 = 2,$$
$$p(0) + g(S_{0,0}, S_{1,2}) = 0 + 2 = 2,$$
$$p(-1) + g(S_{0,-1}, S_{0,2}) = 0 + 0 = 0$$
$$\} = 0$$

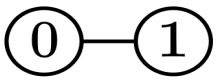

Figura 5.23: Sottografo indotto da v_0, v_1

$$p(1) = min\{$$
$$p(0) + g(S_{0,0}, S_{1,1}) = 0 + 1 = 1,$$
$$p(-1) + g(S_{0,-1}, S_{0,2}) = 0 + 0 = 0,$$
$$\} = 0$$

La soluzione ottima ha valore $p(6) = 3$. Per determinare la partizione ottima si può procedere come per un qualsiasi algoritmo di programmazione dinamica. In particolare per il grafo in figura 5.18, abbiamo due partizioni ottime le quali sono riportate nelle figure 5.24 e 5.25.

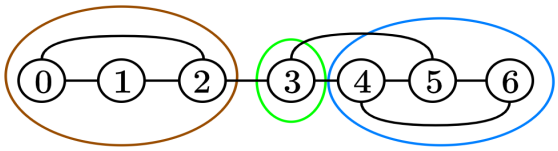

Figura 5.24: Partizione ottima 1

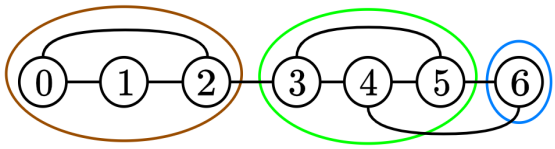

Figura 5.25: Partizione ottima 2

Segue in 10 lo pseudo-codice dell'algoritmo basato su programmazione dinamica per il problema del clustering su grafi lineari.

Algorithm 10: CDGL Clustering Deletion su grafi lineari

Input:

- Grafo $G = (V, E)$
- $v_0, \ldots, v_{n-1}$

Output:

- val numero di edge che connettono due nodi in sottoinsiemi diversi di P

1: **if** $i == 0$ oppure $i == -1$ **then**
2: **return** 0
3: **else if** $p[i]$ è stato già computato **then**
4: **return** $p[i]$
5: **else**
6: $val = +\infty$
7: $t = i$
8: **while** $S_{t,i}$ è una clique **do**
9: $val = min(val, CDGL(t-1) + g(S_{0,t-1}, S_{t,i}))$
10: $t = t - 1$
11: **end while**
12: $p[i] = val$
13: **return** $p[i]$
14: **end if**

Teorema 5.2.2. Correttezza e ottimalità algoritmo
L'algoritmo è corretto e ottimale.

Dimostrazione. Il Clustering Deletion su grafi lineari può essere risolto con l'algoritmo di programmazione dinamica sopra descritto. Come prima cosa notiamo che $p(i) :=$ soluzione ottima per $v_0, \ldots, v_i$ individua correttamente i sottoproblemi su $v_0, \ldots, v_{n-1}$.
Proviamo dapprima la correttezza ovvero l'algoritmo proposto individua solo cluster graph. Per come funziona l'algoritmo su un generico sottoproblema di dimensione i esso va a determinare tutti i possibili sottoinsiemi $S_{t,i}$ tali che $S_{t,i} = \{ v_t, v_{t+1}, \ldots, v_{i-1}, v_i \}$ sia una clique. Da ciò segue che l'algoritmo prende in considerazione solo sottoinsiemi che individuano clique e di conseguenza la partizione determinata sarà necessariamente un cluster graph.
Proviamo ora che l'algoritmo restituisce la soluzione ottimale. L'algoritmo restituisce la partizione contigua rispetto $v_0, \ldots, v_i$ ottimale in quanto, considera tutti i possibili sottoinsiemi contigui che hanno come maggiorante v_i e per ognuno di essi determina il costo di tale scelta e il sottoproblema da risolvere associato. Tra tutte queste possibili scelte ritorna quella che gli assicura il valore minimo. Per il teorema 5.2.1 possiamo quindi affermare che ritorna la soluzione ottimale per il problema del Clustering Deletion su grafi lineari. □

Approccio greedy basato sulla funzione $g(.,.)$

É possibile definire un'approccio greedy basato sulla funzione $g(.,.)$. In particolare considerando un generico sottoproblema $v_0, v_1, \ldots, v_i$ invece di risolvere tutti i sottoproblemi associati potremmo effettuare una scelta greedy basata sulla funzione $g(S_{0,t-1}, S_{t,i})$. Con questo approccio ci stiamo chiedendo quale tra le seguenti clique

$$S_{l(i),i}, S_{l(i)+1,i}, \ldots, S_{i-1,i}, S_{i,i}$$

appartiene alla soluzione ottimale? La scelta greedy basata sulla funzione $g(.,.)$ consiste nel prendere quella per cui $g(S_{0,t-1}, S_{t,i})$ per $t = l(i), l(i)+1 \ldots, i$

è minimizzato. Una volta invididuata tale clique sia t_{min} l'indice associato risolviamo il sottoproblema $v_0, v_1, \ldots, v_{t_{min}-1}$ allo stesso modo. Ripetiamo il procedimento fin quando non esauriamo tutti i nodi.

Problema aperto 5.2.1. L'approccio greedy basato sulla funzione $g(.,.)$ è un algoritmo ottimale? E più in generale cosa possiamo dire sulla soluzione fornita.

5.2.3 Analisi complessità algoritmo di programmazione dinamica

L'analisi della complessità temporale, nel caso pessimo, per un algoritmo di programmazione come quello proposto è incetrata nell'analisi della relazione di ricorrenza su cui si basa l'algortimo. Ricordando la relazione di ricorrenza:

$$p(i) = \min_{t=i,i-1,\ldots,l(i)} \{p(t-1) + g(S_{0,t-1}, S_{t,i})\} \tag{5.9}$$

la sua analisi consiste nel determinare il costo impiegato su ogni sottoproblema, che in questo caso consiste nel determinare il costo per computare $g(S_{0,t-1}, S_{t,i})$, e determinare il numero di sottoproblemi che devono essere risolto per risolvere il problema principale.

Computazione funzione $g(.,.)$

Nella relazione di ricorrenza, in ogni sottoproblema, è necessario computare la funzione $g(.,.)$. Questa funzione può essere sia precalcolata e memorizzata in una matrice, e a tempo debito leggere l'opportuna cella della matrice, oppure può essere calcolata a run-time quando se ne ha bisogno. Ognuna delle due alternative ha vantaggi e svantaggi. La prima alternativa ci permette di precalcolare le entry della matrice e quindi quando necessario in tempo costante possiamo determinare il valore di $g(.,.)$, ma come svantaggi ha il fatto che è necessario salvare in memoria tutta la matrice e che in linea di principio, potremmo dover perdere tempo a calcolare entry della matrice che per la risoluzione della relazione di ricorrenza non verranno mai utilizzate. La

seconda alternativa ha come vantaggi il fatto che non dobbiamo memorizzare la matrice e che computiamo solo le entry necessarie, mentre ha come svantaggio il fatto che durante l'esecuzione dell'algoritmo, dovremo computare la funzione $g(.,.)$. Detto ciò conviene utilizzare il primo o il secondo approccio a seconda delle esigenze. A prescindere dall'approccio utilizzato possiamo computare la funzione per gl'input richiesti dall'algortimo, ovvero $g(S_{0,t-1}, S_{t,i})$ per ogni $t = i, i-1, \ldots, l(i)$, come segue. Come primo modo possiamo utilizzare un approccio naive che si basa semplimente sulla definizione 5.2.4 ovvero per ogni nodo in $S_{t,i}$ determinare quanti sono i nodi in $S_{0,t-1}$ con cui ha un edge. La somma di tali quantità restituisce $g(S_{0,t-1}, S_{t,i})$. Tale approccio ha una complessità temporale nel caso pessimo pari $\mathcal{O}(|S_{0,t-1}| \times |S_{t,i}|)$. Possiamo però utilizzare un approccio più efficiente se ci basiamo su gli $l(.)$ e calcolare $g(S_{0,t-1}, S_{t,i})$ come segue:

$$g(S_{0,t-1}, S_{t,i}) = \sum_{k=t}^{i} t - l(k) \tag{5.10}$$

Esempio 5.2.4. Riprendendo l'esempio 5.2.3 computiamo $g(S_{0,4}, S_{5,6})$ e $g(S_{0,3}, S_{4,6})$ utilizzando la formula 5.10:

$$g(S_{0,4}, S_{5,6}) = (5-3) + (5-4) = 2 + 1 = 3$$

$$g(S_{0,3}, S_{4,6}) = (4-3) + (4-3) + (4-4) = 1 + 1 = 2$$

Tale approccio ha una complessità temporale nel case pessimo $\mathcal{O}(i-t+1)$, che è nettamente inferiore rispetto all'approccio naive.

Complessità relazione di ricorrenza

Per semplicità di analisi assumeremo che il costo della funzione $g(.,.)$ sia costante, in quanto per le ragioni precedentemente esposte alla fine dei conti la funzione $g(.,.)$ può essere precomputata e memorizzata in una matrice e utilizzata quando necessario. L'algoritmo di programmazione dinamica proposto impiega nel caso pessimo un tempo quadratico rispetto al numero di nodi del grafo in input. Questo perchè l'algoritmo parte da destra (o indifferentemente da sinistra) e fondamentalmente quello che fa è visitare ogni

nodo del grafo in accordo all'ordinamento del grafo. Su ogni nodo che visita genera tanti sottoproblemi quanti sono i nodi che lo precedono (o lo seguono) nell'ordinamento a cui è collegato con un edge. Considerando la versione in avanti della relazione di ricorrenza del Clustering Deletion su grafi lineari risulta che per ogni nodo i occorre risolvere $u(i) - i + 1$ sottoproblemi quindi per un dato grafo lineare G dovremo risolvere un numero di sottoproblemi pari a:

$$\sum_{i=0}^{n-2} u(i) - i + 1$$

Il caso peggiore lo si ha per quel grafo per il quale è necessario generare il maggior numero di sottoproblemi. Quel grafo è quello per cui ogni $u(i)$ assume il valore più grande possibile, ovvero $u(i) = n - 1$. Tale grafo è il grafo completo per il quale dobbiamo pagare un costo di:

$$\begin{aligned}
\sum_{i=0}^{n-2} u(i) - i + 1 &= \sum_{i=0}^{n-2} n - 1 - i + 1 \\
&= \sum_{i=0}^{n-2} n - i \\
&= n + (n-1) + \cdots + 2 \\
&= \frac{n(n+1)}{2} - 1 = \mathcal{O}(n^2)
\end{aligned}$$

Questa è un'analisi nel caso pessimo e quindi è possibile che mediamente l'algoritmo impieghi meno tempo. Potremmo fare un'analisi nel caso medio, per cui in linea di principio ci potrebbe tornare utile il modello di generazione uniforme dei grafi lineari, proposto in 4.2.

CAPITOLO 6

CLUSTERING DELETION SU GRAFI

In questo capitolo viene studiato il problema del Clustering Deletion su grafi generici. Nella sezione 6.1 vengono introdotte le principali definizioni utilizzate nel capitolo e viene data un'idea ad alto livello della logica che sta dietro all'algoritmo di edge contraction proposto. Nella sezione 6.2 viene descritto in maniera più dettagliata il funzionamento dell'algoritmo di edge contraction.

6.1 Clustering deletion basato su edge contraction

Il questo capitolo verrà studiato il problema del Clustering Deletion su grafi generici. Tale problema su grafi qualsiasi è un problema NP-hard [3].

Clustering Deletion in termini di partizione

Input:

- $G = (V, E)$ grafo

Output: Una partizione P dei nodi tale che:

- $G' = (V, E')$ il sottografo indotto dalla partizione P è un cluster graph
- $|E| - |E'|$ è minimizzato

La logica fondante l'algoritmo che verrà proposto è basata sull'operazione di edge contraction. L'edge contraction è un'operazione fondamentale in teoria dei grafi minori [16]. Consiste nell'eliminare un edge $e = (u, v)$ da un grafo e contemporaneamente unire i due nodi u,v.

Definizione 6.1.1. Edge contraction
Dato un grafo $G = (V, E)$ e un edge $e = (u, v) \in E$, l'operazione di edge contraction consiste nell'eliminare l'edge e dal grafo G, unire i nodi u,v in un unico super nodo e modificare in modo opportuno gli edge incidenti sui nodi u, v.

Definizione 6.1.2. Super nodo
Un super nodo è un insieme di nodi in un grafo risultante da operazioni di edge contraction.

Esempio 6.1.1. A titolo di esempio consideriamo il grafo in figura 6.1. Tale grafo verrà ripreso più volte come riferimento nel corso del capitolo.

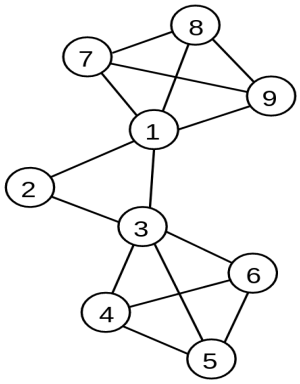

Figura 6.1: Grafo

Il grafo risultante a seguito di una o più operazione di edge contraction può essere memorizzato in differenti modi, a seconda dello scopo per cui facciamo l'edge contraction. Possiamo memorizzare il fatto che tra i super nodi che si vengono a creare con operazioni di edge contraction ci siano più di un edge. In questo caso il grafo risultante diventa un multigrafo, ovvero un grafo in cui un dato edge può occorrere più di una volta, o più formalmente un grafo per cui l'insieme E di edge è un multi-insieme.

Definizione 6.1.3. Multi-insieme
Un multi-insieme è una generalizzazione del concetto di insieme [17]. Mentre in un insieme le ripetizioni di un elemento non vengono considerate, in un multi-insieme ogni elemento è considerato con un determinato numero di ripetizioni. In questi termini si potrebbe definire un multi-insieme come una collezione di elementi, che possono comparire più di una volta e in cui l'ordine non è importante.

Ad esempio se sul grafo in figura 6.1, operiamo questa scelta, effettuando l'operazione di edge contraction sull'edge $(4, 5)$ abbiamo il multigrafo riportato in figura 6.2.

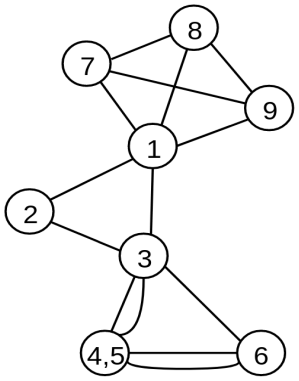

Figura 6.2: Multigrafo edge contraction $(4, 5)$

Questa informazione potrebbe però non interessarci e quindi potremmo non voler memorizzare gli edge ripetuti. In questo caso il grafo risultante è un grafo semplice. Ad esempio se sul grafo in figura 6.1, operiamo questa scelta, effettuando l'operazione di edge contraction sull'edge $(4, 5)$ abbiamo il grafo riportato in figura 6.3.

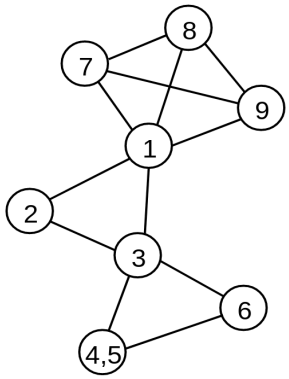

Figura 6.3: Grafo semplice edge contraction $(4, 5)$

Potremmo essere invece interessati al numero di edge tra i nodi contenuti in due super nodi. In questo caso il grafo risultante è un grafo pesato sugl'edge, in cui il peso di un edge indica il numero di edge tra i nodi contenuti nei i super nodi connessi dall'edge in questione. Ad esempio se sul grafo in figura 6.1, operiamo questa scelta, effettuando l'operazione di edge contraction sull'edge $(4, 5)$ abbiamo il grafo riportato in figura 6.4.

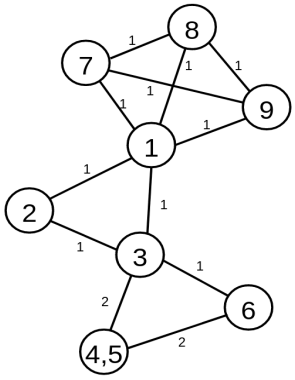

Figura 6.4: Grafo pesato edge contraction $(4, 5)$

Logica ad alto livello algoritmo di edge contractrion

Ad alto livello la logica che sta dietro all'algoritmo di edge contraction per il problema del Clustering Deletion prende spunto da due intuizioni. I super nodi ottenuti a seguito di operazioni di edge contraction individuano un partizionamento dei nodi del grafo di partenza, che è proprio ciò che noi stiamo cercando. O meglio noi stiamo cercando un particolare tipo di partizionamento, ovvero un partizionamento il cui sottografo indotto è un cluster graph. Detto in altri termini vogliamo quindi che ogni super nodo individui una clique. È possibile ottenere quanto detto, se prima di effettuare l'operazione di edge contraction facciamo un test che ci permette di stabilire se il super nodo che andremo a costruire costituisce o meno una clique. Per essere più precisi introduciamo una nuova definizione che caratterizza questi particolari edge.

Definizione 6.1.4. Edge estraibile

Chiameremo un edge $e = (U, V)$, con U e V super nodi, estraibile se e solo se $U \cup V$ è una clique nel grafo G.

Ricapitolando, le due intuizioni su cui si basa l'algoritmo sono:

- I super nodi individuano un partizionamento del grafo di input,
- Contrarre solo edge estraibili, permette di individuare partizioni i cui sottografi indotti sono cluster graph.

In 11 è riportato un pseudo-codice ad alto livello dell'algoritmo. L'algoritmo prendendo un grafo in input un grafo $G = (V, E)$ effettua iterativamente l'operazione di edge contraction, facendo però attenzione a contrarre solo edge estraibili. Nel momento in cui non ci sono più edge estraibili ritorna l'insieme dei super nodi finale.

Algorithm 11: Clustering Deletion edge contraction

Input: Grafo $G = (V, E)$

Output: Cluster graph di G

1: **while** $\exists$ *un edge estraibile in* E **do**
2: Prendi un edge e estraibile in E
3: Determina G' contraendo e su G
4: $G = G'$
5: **end while**
6: **return** V

6.2 Algoritmo edge contraction based

Sia $G = (V, E)$ il grafo in input all'algoritmo di edge contraction. Consideriamo la sequenza di grafi che vengono definiti dall'esecuzione dell'algoritmo: $G^{(0)}, G^{(1)}, \ldots, G^{(p)}$, dove il grafo $G^{(i)}$ corrisponde al grafo che determina l'algoritmo dopo esattamente i passi di computazione e il grafo $G^{(0)} = G$ corrisponde al grafo in input.

Osservazione 6.2.1. Se $|V| = n$, il numero di iterazioni p compiute dall'algoritmo è tale che $1 \leq p \leq n - 1$. In particolare possiamo osservare che $p = n - 1$ solo se il grafo in input è un grafo completo.

Al passo 1 l'algoritmo parte con l'individuare l'insieme degl'edge estraibili, ma poichè siamo alla prima iterazione, tale insieme coincide con E. Una volta individuato, determina un edge in tale insieme da contrarre e lo contrae ottenendo così il grafo $G^{(1)}$ nella sequenza sopra definita. Dato l'insieme degl'edge estraibili con quale criterio viene scelto l'edge da contrarre? Nella sottosezione 6.2.2 viene affrontata questa questione. Al

passo 2 l'algoritmo individua nell'insieme $E^{(1)}$ l'insieme degl'edge estraibili. Per poter compiere tale operazione in linea di principio, basandosi sulla sola definizione 6.1.4 occorrerebbe pagare un costo eccessivo per ogni edge. In particolare: dato un edge (U, V), dovremmo determinare se per ogni $u \in U$ esiste un edge con ogni $v \in V$, pagando un costo in termini di tempo pari a $\mathcal{O}(|U||V|)$ per ogni edge (U, V). Nella sottosezione 6.2.1 viene trattato questo problema mostrando un modo efficiente per gestire la proprietà di estraibilità. Una volta determinato l'insieme degl'edge estraibili l'algoritmo ne sceglie uno e lo contrae ottenendo così il grafo $G^{(2)}$. L'algoritmo procede in questo modo, iterazione dopo iterazione, fino a quando non arriva al passo p in cui il grafo $G^{(p)}$ non possiede edge estraibili. Quindi si ferma e restituisce l'insieme dei super nodi $V^{(p)}$ di $G^{(p)}$.

6.2.1 Gestione efficiente della proprietà di estraibilità

Abbiamo osservato come la gestione dell'insieme degl'edge estraibili se ci si basa solo sulla definizione 6.1.4 è troppo costosa. Possiamo renderela più efficiente osservando che non è necessario ad ogni iterazione prendere l'insieme $E^{(i)}$ e per ogni edge stabilire se questo è estraibile o meno, ma possiamo gestire in maniera efficiente l'insieme degl'edge estraibile in maniera da poter evitare tale controllo. Partiamo dall'osservare che l'operazione di edge contraction è un operazione locale nel grafo, in quanto modifica il grafo solo localmente all'edge contratto, il resto del grafo non viene per niente modificato. Questo ci fa capire che poichè $E^{(i)}$ viene costruito a partire da $E^{(i-1)}$, modificando opportuni edge, l'approccio inefficiente oltre a impiegare un test costoso effettua più volte gli stessi test.

Tra i tre approcci per la memorizzazione del grafo risultante a seguito di operazioni di edge contraction, illustrati nella sezione 6.1, utilizzeremo il terzo e di seguito ne daremo un descrizione più dettagliata.

Sia $G = (A, W)$ un grafo e $e = (U, V) \in W$ un edge. Denotiamo con $G/e = (A/e, W/e)$ il grafo ottenuto eseguendo su G la contrazione dell'edge e.

Il grafo G/e avrà gli stessi super nodi di G ad eccezione dei super nodi connessi dall'arco contratto che vengono fusi in un solo super nodo. Per quanto riguarda gli edge, G/e avrà gli stessi egde ad eccezione dell'edge (U, V) e degl'edge incidenti sui super nodi U e V. Se consideriamo tutti gli edge dei super nodi U e V possono capitare 3 situazione.

1. U,V hanno entrambi un edge con un super nodo R,
2. U ha edge con un super nodo R' mentre V no,
3. V ha un edge con un super nodo R'' mentre U no.

Tutti questi edge verranno cancellati e sostituiti con edge del tipo:

1. (Z, R) con costo $c(Z, R) = c(U, R) + c(V, R)$ nel primo caso,
2. (Z, R') con costo $c(Z, R') = c(U, R')$ nel secondo caso,
3. (Z, R'') con costo $c(Z, R'') = (V, R'')$ nel terzo caso.

Un pò più formalmente il grafo ottenuto a seguito di un operazione di edge contraction è descritto nella definizione che segue.

Definizione 6.2.1. Grafo G/e
Sia $G = (A, W)$ un grafo e $e = (U, V) \in W$ un arco, $G/e = (A/e, W/e)$ il grafo ottenuto eseguendo su G la contrazione dell'edge e. Allora G/e è siffatto: denotando con $Z = U \cup V$ il super nodo ottenuto

1. $A/e = A \setminus \{ U, V \} \cup \{ Z \}$,
2. $N(Z)^{11} = N(U) \cap N(V)$,
3. $N(Z)^{01} = N(U) \cap \overline{N(V)}$,
4. $N(Z)^{10} = \overline{N(U)} \cap N(V)$,
5. $N(Z) = N(Z)^{10} \cup N(Z)^{01} \cup N(Z)^{11}$,
6. $W/e = W \setminus \{ e \} \setminus W(U) \setminus W(V) \cup \{ (Z, Z') \mid Z' \in N(Z) \}$,

7. $c(Z,P) = c(P,U),\ P \in N(Z)^{01}$,

8. $c(Z,P) = c(P,V),\ P \in N(Z)^{10}$,

9. $c(Z,P) = c(P,U) + c(P,V),\ P \in N(Z)^{11}$.

L'approccio basato su i grafi pesati ci permette di determinare in tempo costante se un edge è estraibile o meno. Questo perchè un edge $e = (U,V)$ è estraibile se $U \cup V$ è una clique nel grafo di partenza, ma ciò in altre parole vuol dire che ogni nodo in U è connesso con ogni altro nodo in V e questo significa che $c(U,V) = |U||V|$. Per queste ragioni vale la proposizione che segue.

Proposizione 6.2.1. Un edge $e = (U,V)$ è estraibile se e solo se $c(U,V) = |U||V|$.

In questo modo possiamo determinare se un edge è estraibile in tempo $\mathcal{O}(1)$.

Introduciamo la seguente definizione al fine di essere più concisi in seguito.

Definizione 6.2.2. $E_t^{(i)}$
Sia $G^{(i)}$ il grafo all'iterazione i. Denotiamo con $E_t^{(i)}$ l'insieme di tutti gli edge estraibili in $G^{(i)}$, mentre con $E_f^{(i)}$ il suo complemento, ovvero l'insieme di tutti gli edge non estraibili in $G^{(i)}$. Notiamo che risulta che $E^{(i)} = E_t^{(i)} \cup E_f^{(i)}$.

Proposizione 6.2.2. Gli edge (Z,P) con $P \in N(Z)^{11}$ sono edge estraibili se e solo se (U,P) e (V,P) erano estraibili nell'iterazione precedente. Più formalmente:

$$(Z,P) \in E_t^{(i)} \iff (U,P) \in E_t^{(i-1)} \text{ e } (V,P) \in E_t^{(i-1)}, \quad \forall i \tag{6.1}$$

Dimostrazione. Osserviamo dapprima che agl'edge (Z,P) dalla definizione 6.2.1 viene assegnato un costo pari $c(Z,P) = c(P,U) + c(P,V)$.
$\Rightarrow$
Se $(Z,P) \in E_t^{(i)}$ allora significa che ogni nodo in Z è connesso con ogni nodo in P, ma poichè $Z = U \cup V$ significa che ogni nodo in U è connesso con ogni nodo in P e che ogni nodo in V è connesso con ogni nodo in P, e quindi (U,P)

e (V, P) erano entrambi edge estraibili all'iterazione precedenete.

$\Leftarrow$

Se $(U, P) \in E_t^{(i-1)}$ e $(V, P) \in E_t^{(i-1)}$ allora $c(U, P) = |U||P|$ e $c(V, P) = |V||P|$ ma ciò implica a sua volta che:

$$c(Z, P) = c(P, U) + c(P, V) = |U||P| + |V||P| = (|U| + |V|)|P| = |Z||P|$$

da cui segue che $c(Z, P) = |Z||P|$ e quindi (Z, P) è estraibile. □

Proposizione 6.2.3. Gli edge (Z, P) con $P \in N(Z)^{01}$ oppure con $P \in N(Z)^{10}$ sono edge non estraibili.

Dimostrazione. Osserviamo dapprima che agl'edge (Z, P) con $P \in N(Z)^{01}$ oppure con $P \in N(Z)^{10}$ dalla definizione 6.2.1 viene assegnato un costo pari a:

1. $c(U, P)$ se $P \in N(Z)^{01}$,
2. $c(V, P)$ se $P \in N(Z)^{10}$.

Poichè $Z = U \cup V$ e $U \cap V = \emptyset$ abbiamo che $|Z| = |U| + |V|$. Nel primo caso

$$c(Z, P) = c(U, P) \leq |U||P| < |Z||P|$$

e quindi sicuramente (Z, P) non è estraibile. Nel secondo caso similmente

$$c(Z, P) = c(V, P) \leq |V||P| < |Z||P|$$

e perciò sicuramente (Z, P) non è estraibile. □

Possiamo evitare di effettuare il test, seppur pagando un costo costante, osservando che:

$$E_t^{(i)} = E_t^{(i-1)} \setminus \{ e \} \setminus W(U) \setminus W(V) \cup \{ (Z, P) \mid P \in N(Z)^{11} \}$$

ovvero l'insieme degl'edge estraibili all'iterazione i è uguale all'insieme degl'edge estraibili all'iterazione $i - 1$ meno l'arco e che è stato contratto e meno tutti gli edge che hanno come estremo un nodo tra U e V. I restanti

edge essendo l'operazione di edge contraction un operazione locale non vengono modificati. A questo insieme vanno poi aggiunti gli edge del tipo (Z, P) con $P \in N(Z)^{11}$ per cui $(U, P) \in E_t^{(i-1)}$ e $(V, P) \in E_t^{(i-1)}$ in quanto dalla proposizione 6.2.2 possiamo concludere a priori che sono estraibili, mentre gli edge del tipo (Z, P) con $P \in N(Z)^{01}$ oppure $P \in N(Z)^{10}$ sappiamo che non sono estraibili per la proposizione 6.2.3.

Osservazione 6.2.2. Possiamo osservare che $E_t^{(i)}$ per ogni $i = 0, \ldots, p$ è tale che:

$$|E_t^{(i)}| < |E_t^{(i-1)}|$$

mentre $E_f^{(i)}$ per ogni $i = 0, \ldots, p$ è tale che:

$$|E_f^{(i)}| \geq |E_f^{(i-1)}|$$

ovvero gli edge estraibili con il procedere dell'esecuzione diventano sempre di meno, mentre gli edge non estraibili diventano sempre di più.

Osservazione 6.2.3. Osserviamo che per determinare il numero di edge che devono essere eliminati, affichè il partizionamento dei nodi fornito dall'algoritmo induca correttamente un cluster graph, basta sommare il costo associato agl'archi che non sono estraibili all'ultima iterazione, ovvero se denotiamo con S l'insieme degli archi che devono essere eliminati allora:

$$|S| = \sum_{e \in E_f^{(p)}} c(e)$$

Osservazione 6.2.4. Upper e lower bound clique U
Sia U un super nodo di un grafo $G^{(i)} = (V^{(i)}, E^{(i)})$ ad una generica iterazione i dell'algoritmo 11. La clique C in cui comparirà U nell'output restituito avrà dimensioni:

$$|U| \leq |C| \leq |U| + \sum_{V \in V^{(i)} : (U,V) \in E_t^{(i)}} |V|$$

Ovvero avrà almeno $|U|$ nodi e al più $|U|$ nodi più la somma della cardinalità dei super nodi connessi ad U con un edge estraibile.

Proviamo ora che l'algoritmo 11 contraendo solo edge estraibili ritorna partizioni i cui sottografi indotti sono sempre cluster graph.

Teorema 6.2.1. Correttezza Algoritmo
Ogni partizione restituita in output dall'algoritmo 11 ha come sottografo indotto un cluster graph.

Dimostrazione. La prova del teorema 6.2.1 può essere fatta per induzione sul numero di iterazione dell'algoritmo.
Sia $G^{(i)} = (V^{(i)}, E^{(i)})$ un generico grafo all'i-esima iterazione dell'algoritmo, occorre provare che tutti i super nodi di $G^{(i)}$ formano clique sul grafo in input, per ogni i.
Passo base:
Il passo base corrisponde ad andare a considerare $i = 0$, ovvero il grafo di input $G^{(0)} = G$, per il quale l'asserto è chiaramente verificato.
Passo induttivo:
Nel passo induttivo occorre provare che se l'asserto è verificato per $G^{(i)}$ allora è verificato anche per $G^{(i+1)}$. Per ipotesi induttiva sappiamo che ogni super nodo di $G^{(i)}$ è una clique nel grafo in input. Per dimostrare che ciò è vero anche per $G^{(i+1)}$ è necessario considerare come $G^{(i+1)}$ viene costruito a partire da $G^{(i)}$. L'algoritmo 11 ad ogni iterazione contrae solo edge estraibili, ovvero edge $e = (U, V)$ tali che $c(U, V) = |U||V|$, ma dal momento che ogni unità di costo rappresenta un edge tra $u \in U$ e $v \in V$, ciò significa che ogni nodo in U ha un edge con ogni nodo in V. Il nuovo super nodo costruito contraendo e sarà perciò una clique.
Questo conclude la prova. □

Abbiamo quindi provato che se contraiamo solo edge estraibili, a prescindere dal criterio utilizzato per la scelta dell'edge da contrarre, otteniamo in output una partizione dei nodi il cui sottografo indotto è un cluster graph.

6.2.2 Scelta edge da contrarre

Abbiamo finora trattato la scelta dell'edge estraibile da contrarre come una black-box, in questa sottosezione esamineremo alcuni dei possibili approcci che possiamo utilizzare. Descriveremo dapprima un approccio random che definisce un algoritmo randomizzato e dopodichè un serie di possibili approcci greedy.

Prima però facciamo delle considerazione in merito all'ottimalità dell'algoritmo. Sia $G = (V, E)$ un grafo in input all'algoritmo di edge contraction e sia S^* l'insieme di edge la cui eliminazione dal grafo individua il cluster graph ottimale. Fissato un determinato criterio per la scelta degl'edge da contrarre ad ogni iterazione vale che l'algoritmo riesce a trovare l'ottimo se e solo se in nessuna iterazione viene contratto un edge in S^*. Più formalmente definiamo l'evento T_i:

$$T_i = \begin{cases} 1 & \text{se nessun arco di } S^* \text{ viene contratto all'iterazione } i \\ 0 & \text{altrimenti} \end{cases} \tag{6.2}$$

Proposizione 6.2.4.

$$\textit{L'algoritmo trova l'ottimo} \iff T_1 \wedge T_2 \wedge \cdots \wedge T_p \tag{6.3}$$

Random edge contraction

Il primo approccio random che può venire in mente per la scelta dell'edge estraibile da contrarre è prendere un edge in modo uniforme nell'insieme degl'edge estraibili $E_t^{(i)}$, per ogni iterazione i. Un altro possibile approccio random potrebbe, invece, sfruttare le informazioni codificate nei pesi degl'archi di ogni edge per definire una distribuzione di probabilità. Dato l'insieme degl'edge estraibili $E_t^{(i)}$ ad una generica iterazione estraiamo ogni edge $e \in E_t^{(i)}$ con una probabilità $\mathbb{P}(e)$ strettamente dipendente dal peso associato a quest'ultimo:

$$\mathbb{P}(e) = \frac{c(e)}{\sum_{a \in E_t^{(i)}} c(a)}$$

la quale correttamente individua una distribuzione di probabilità.

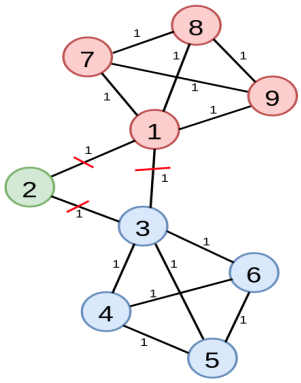

Figura 6.5: Soluzione ottimale Clustering Deletion

Se l'algoritmo effettua p iterazioni con $p = 0, \ldots, n-1$ restituirà la soluzione ottima S^* con probabilità:

$$\mathbb{P}(T_1, T_2, \ldots, T_p) = \mathbb{P}(T_1)\mathbb{P}(T_2|T_1)\ldots\mathbb{P}(T_p|T_{p-1}, \ldots, T_1) \tag{6.4}$$

Osservazione 6.2.5. Osserviamo che le probabilità $\mathbb{P}(T_i|T_{i-1}, \ldots, T_1)$ man mano che si procede con l'esecuzione crescono.

Esempio 6.2.1. A titolo di esempio al fine di comprendere meglio il funzionamento dell'algoritmo mostriamo una sua possibile esecuzione con input il grafo riportato in figura 6.1. Possibile esecuzione perchè contraendo edge estraibili a caso ogni esecuzione dell'algoritmo può essere diversa. Nelle figure che seguiranno al fine di distinguere gli edge estraibili da quelli non estraibili, gli edge estraibili verranno colorati di blu mentre quelli non estraibili in rosso. In figura 6.6 è riportato il grafo in input pesato, mentre nelle figure 6.7, 6.8, 6.9, 6.10, 6.11 e 6.12 sono riportati, in ordine, i grafi ottenuti ad ogni passo dell'esecuzione dopo aver contratto l'edge di riferimento. Il cluster graph ottimale del grafo 6.1 è riportato in figura 6.5.

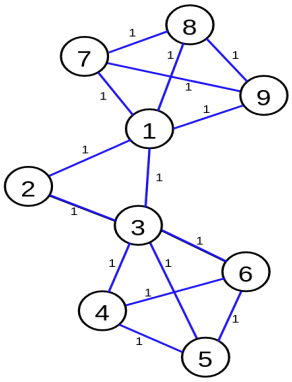

Figura 6.6: Grafo pesato iniziale

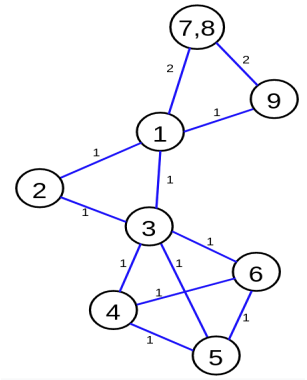

Figura 6.7: Grafo edge contraction $(7, 8)$, $\mathbb{P}\{(7, 8)\} = \frac{1}{15}$

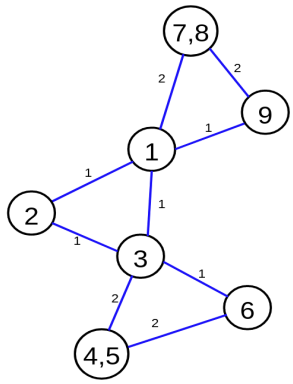

Figura 6.8: Grafo edge contraction $(4, 5)$, $\mathbb{P}\{(4, 5)\} = \frac{1}{14}$

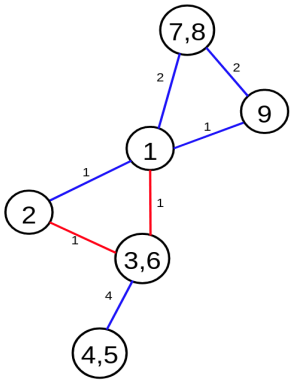

Figura 6.9: Grafo edge contraction $(3, 6)$, $\mathbb{P}\{(3, 6)\} = \frac{1}{13}$

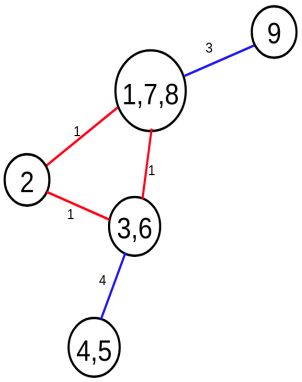

Figura 6.10: Grafo edge contraction $(78, 1)$, $\mathbb{P}\{(78, 1)\} = \frac{2}{10}$

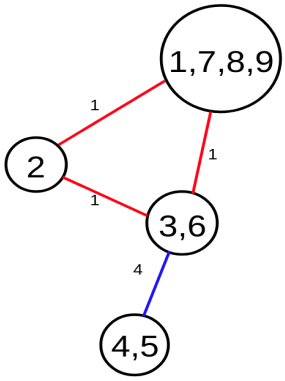

Figura 6.11: Grafo edge contraction $(178, 9)$, $\mathbb{P}\{(178, 9)\} = \frac{3}{7}$

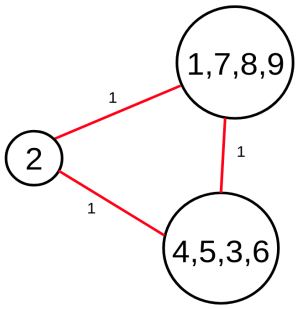

Figura 6.12: Grafo edge contraction $(45, 36)$, $\mathbb{P}\{(45, 36)\} = 1$

La probabilità di tale esecuzione è quindi:

$$\frac{1}{15} \times \frac{1}{14} \times \frac{1}{13} \times \frac{2}{10} \times \frac{3}{7} \times 1 = \frac{1}{31850}$$

Di seguito riportiamo un'altra possibile esecuzione al fine di osservare come esecuzioni diverse hanno probabilità di successo diverse. Nelle figure 6.13, 6.14, 6.15, 6.16, 6.17 e 6.18 sono riportati, in ordine, i grafi ottenuti ad ogni passo della seconda esecuzione dopo aver contratto l'edge di riferimento.

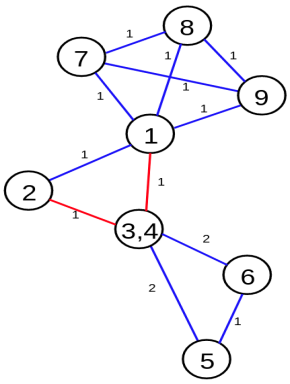

Figura 6.13: Grafo esecuzione 2 edge contraction $(3, 4)$, $\mathbb{P}\{(3, 4)\} = \frac{1}{15}$

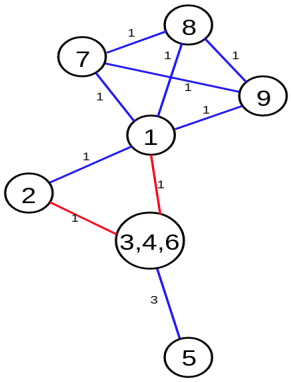

Figura 6.14: Grafo esecuzione 2 edge contraction $(34, 6)$, $\mathbb{P}\{(34, 6)\} = \frac{2}{12}$

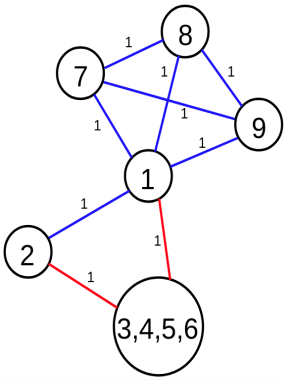

Figura 6.15: Grafo esecuzione 2 edge contraction $(346, 5)$, $\mathbb{P}\{(346, 5)\} = \frac{3}{10}$

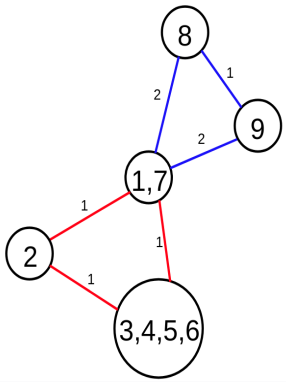

Figura 6.16: Grafo esecuzione 2 edge contraction $(1, 7)$, $\mathbb{P}\{(1, 7)\} = \frac{1}{7}$

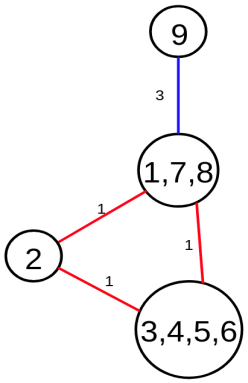

Figura 6.17: Grafo esecuzione 2 edge contraction $(17, 8)$, $\mathbb{P}\{(17, 8)\} = \frac{2}{5}$

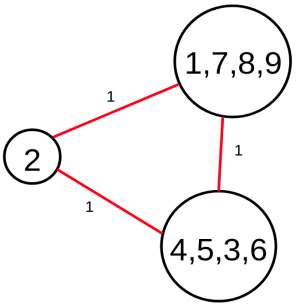

Figura 6.18: Grafo esecuzione 2 edge contraction $(178, 9)$, $\mathbb{P}\{(178, 9)\} = 1$

La probabilità di tale esecuzione è quindi:

$$\frac{1}{15} \times \frac{2}{12} \times \frac{3}{10} \times \frac{1}{7} \times \frac{2}{5} \times 1 = \frac{1}{5250}$$

Nella prima esecuzione la probabilità di successo era di $\frac{1}{31850}$ mentre nella seconda di $\frac{1}{5250}$. Ciò ci fa capire che contrarre un edge ad una iterazione piuttosto che un'altra incide notevolemente sulle probabilità di successo dell'algoritmo. Fare scelte corrette all'inizio in questo senso ci permette di aumentare le probabilità che l'esecuzioni trovi la soluzione ottimale, mentre fare scelte sbagliate all'inizio riduce le probabilità.

L'algoritmo di edge contraction che si basa su una scelta random definisce un algoritmo probabilistico 12 per il problema del Clustering Deletion. Tale algoritmo in una qualsiasi esecuzione, poichè contrae solo edge estraibili, ritorna correttamente un cluster graph, ciò che non è garantito è che tale cluster

graph sia quello ottimo. Tale algoritmo rientra nella categoria di algoritmi probabilistici Monte Carlo. Se $\mathbb{P}(T_1, T_2, \ldots, T_p) = p$ è la probabilità che l'algoritmo di random edge contraction ritorni il cluster graph ottimale avremo che l'esecuzione dell'algoritmo definirebbe una variabile aleatoria geometrica [18].

Definizione 6.2.3. Variabila aletoria geometrica
Consideriamo il seguente esperimento: ripetiamo in condizioni di indipendenza una certa prova avente probabilità di successo pari a p e probabilità di insucesso pari a $1-p$. Sia X il numero di prove necessarie per ottenere il primo successo, allora:

$$\mathbb{P}(X = n) = (1-p)^{n-1}p \qquad n = 1, 2, \ldots$$

sotto queste ipotesi X è una variabile aleatoria geometrica. Il valore atteso di una variabile alaeatoria geometrica è $E[X] = \frac{1}{p}$ ed assume il significato di numero atteso di prove necessarie prima di osservare il primo successo.

Quindi in questo caso si dovrebbe ripetere $\frac{1}{p}$ volte l'esecuzione dell'algoritmo e tra tutte queste esecuzioni prendere quella con il valore migliore. Chiaramente più la probabilità p è piccola e più $\frac{1}{p}$ è grande, ovvero più la probabilità di successo in ogni prova è piccola più prove, in media, saranno necessarie per osservare il primo successo. Notiamo che non è necessario essere in grado determinare l'esatto valore della probabilità che l'algoritmo trovi la soluzione ottima, ma è possibile anche ragionare determinando dei bound alla probabilità di successo dell'algoritmo. In particolare se siamo in grado di dire che $\mathbb{P}(T_1, T_2, \ldots, T_p) \geq p$ allora vuol dire che con un numero medio di prove pari ad al più $\frac{1}{p}$, siamo in grado di trovare l'ottimo.

Algorithm 12: Clustering Deletion edge contraction scelta random

Input: Grafo $G = (V, E)$

Output: Cluster graph di G

1: **while** $E_t \neq \emptyset$ **do**
2: $e = random(E_t)$
3: $G = G/e$
4: **end while**
5: **return** V

Greedy edge contraction

Possiamo definire delle funzioni da associare agl'edge contraibili, che codifichino determinate caratteristiche del problema, al fine di definire vari algoritmi euristici per la risoluzione del problema del Clustering Deletion. Tali algoritmi avranno tutti la medesima struttura, riportata nell'algoritmo 13, e si differenzieranno solo dalla funzione f su cui si effettua la scelta greedy.

Algorithm 13: Clustering Deletion edge contraction scelta greedy

Input: Grafo $G = (V, E)$

Output: Cluster graph di G

1: **while** $E_t \neq \emptyset$ **do**
2: $e = greedy(E_t, f)$
3: $G = G/e$
4: **end while**
5: **return** V

Costo associato all'edge e

Potremmo basare una scelta greedy basandoci sui costi degl'edge, ovvero ad ogni iterazione se $E_t^{(i)}$ è l'insieme degl'edge estraibili potremmo scegliere quell'edge $e \in E_t^{(i)}$ per cui $c(e)$ è massimo.

Numero di edge non estraibili

Basandoci sull'osservazione 6.2.3 possiamo notare che più $|S| = \sum_{e \in E_f^{(p)}} c(e)$

assume valori piccoli e meno edge dovremo eliminare per ottenere il cluster graph ottimale $\sum_{e \in E_f^{(p)}} c(e)$. Sia i l'iterazione corrente per cui dobbiamo scegliere un edge da contrarre, sia $E_t^{(i)}$ l'insieme degl'edge estraibili e $E_f^{(i)}$ l'insieme degl'edge non estraibili. Per le osservazioni fatte sembra ragionevole basare un approccio greedy sulla funzione $f(e)$ definita come segue. Denotando con $E_f^{(i+1)}$ l'insieme degl'edge non estraibili all'iterazione $i+1$ se scegliamo di contrarre l'edge e risulta:

$$f(e) = |E_f^{(i+1)}|$$

quindi dovremmo prendere quell'edge e per cui $f(e)$ è minimo.

Entropia

Definizione 6.2.4. Entropia

Sia X una variabile aleatoria su cui è definita una determinata distribuzione di probabilità. L'entropia $H(.)$ di X è così definita:

$$H(X) = -\sum_{x \in X} \mathbb{P}(x) \log \mathbb{P}(x)$$

Per ogni distribuzione di probabilità l'entropia è tale che $0 \leq H(X) \leq \log |X|$. Più in particolare l'entropia è uguale a 0 se la variabile aleatoria X è degenere, ovvero se la massa di probabilità è tutta concentrata in un unico punto: $\exists x \in X : \mathbb{P}(X = x) = 1$. Mentre è uguale a $\log |X|$ se la variabile aleatoria è uniforme, ovvero se la massa di probabilità è ugualmente distribuita su ogni punto: $\forall x \in X : \mathbb{P}(X = x) = \frac{1}{|X|}$.

L'entropia di una distribuzione di probabilità ha innumerevoli interpretazioni, una su tutte è che l'entropia misura il grado incertezza associato ai possibili esiti di una variabile aleatoria. In tal senso più l'entropia assume valori grandi (prossimi a $\log |X|$) più siamo incerti, mentre più assume valori piccoli (prossimi a 0) meno siamo incerti.

Se consideriamo la distribuzione di probabilità definita in una generica iterazione dell'algoritmo sugl'edge estraibili, possiamo notare che più questa distribuzione è prossima ad una distribuzione uniforme più siamo, in un certo senso, incerti sull'edge da contrarre, mentre più è sbilanciata più siamo certi

sull'edge da contrarre. Per quanto detto l'entropia ci permette di quantificare quanto una distribuzione di probabilità è prossima all'uniforme piuttosto che sbilanciata. In questi termini sembra ragionevole utilizzare un approccio greedy basato sulla funzione entropia. Sia $E_t^{(i)}$ l'insieme degl'edge estraibili all'iterazione i e sia $X^{(i)}$ la variabile aleatoria associata all'estrazione dell'edge estraibile con la rispettiva distribuzione di probabilità associata. La scelta greedy basata sulla funzione entropia consiste nel contrarre quell'edge tale che se con $E_t^{(i+1)}$ è l'insieme degl'edge estraibili all'iterazione successiva e $X^{(i+1)}$ la rispettiva variabile aleatoria associata risulta che l'edge contratto e è tale che minimizza:

$$H(X^i) - H(X^{i+1})$$

ovvero quell'edge che ci permette di ridurre maggiormente il nostro livello di incertezza.

6.3 Bound per k^*

La soluzione ottimale per ogni grafo rispetta sicuramente:

$$0 \leq k^* \leq m - 1 \leq \binom{n}{2} - 1$$

6.3.1 Bound basato sul teorema di Turán

Un possibile approccio per determinare un upper bound più stretto per k^* potrebbe essere il seguente. Dato un generico grafo $G = (V, E)$ con n nodi e m edge possiamo utilizzare ricorsivamente il teorema di Turán per provare l'esistenza di un clique.

Teorema 6.3.1. Teorema di Turán
Sia $G = (V, E)$ un grafo con n nodi e m edge. Per ogni $r \geq 2$ vale che se:

$$m \geq \frac{r-2}{2(r-1)} n^2 \tag{6.5}$$

allora esiste in G una r-clique.

Dall'equazione 6.5 isolando r otteniamo:

$$r \geq 1 + \frac{n^2}{n^2 - 2m}$$

da cui segue che:

$$r_{max} = max_r \left\{ r \geq 1 + \frac{n^2}{n^2 - 2m} \right\} = 1 + \left\lfloor \frac{n^2}{n^2 - 2m} \right\rfloor,$$

ossia la massima clique la cui esistenza è certificata dal teorema di Turán è $r_{max} = 1 + \left\lfloor \frac{n^2}{n^2-2m} \right\rfloor$.

Potremmo sfruttare il teorema di Turán in modo iterativo, ovvero partendo da un grafo $G = (V, E)$, con $|V| = n$ e $|E| = m$, usiamo il teorema di Turán per affermare l'esistenza di una r_0-clique. Eliminiamo tale r_0-clique da G ottenendo un grafo più piccolo $G_1 = (V_1, E_1)$, con $|V_1| = n_1$ e $|E_1| = m_1$. Su tale grafo riutilizziamo il teorema di Turán per affermare l'esistenza di una r_1-clique. Eliminiamo tale r_1-clique da G_1 ottenendo un grafo ancora più piccolo $G_2 = (V_2, E_2)$, con $|V_2| = n_2$ e $|E_2| = m_2$. Ripetiamo tale approccio in quando non esauriamo tutti i nodi. Le clique individuate vanno a formare correttamente un cluster graph.

Teorema 6.3.2. Sia $G = (V, E)$ un grafo con n nodi e m edge.

$$k^* \leq m - \sum_i \binom{r_i}{2}$$

Dimostrazione. Sia $G = (V, E)$ un grafo, con n nodi e m edge, e sia $r_0 = 1 + \left\lfloor \frac{n^2}{n^2-2m} \right\rfloor$. Per il teorema di Turán esiste una r_0-clique.
Consideriamo il grafo $G_1 = (V_1, E_1)$ ottenuto a partire da G eliminando la r_0-clique. Sia k_1 il numero di edge che hanno un estremo nella r_0-clique e uno in G_1. Dal momento che in G_1 ci sono $n_1 = n - r_0$ nodi risulta che $k_1 \leq (n - r_0)r_0$. Per tali ragioni G_1 ha m_1 edge:

$$m_1 = m - k_1 - \binom{r_0}{2} \geq m - (n - r_0)r_0 - \binom{r_0}{2}$$

Su G_1 possiamo riutilizzare il teorema di Turán e determinare

$$r_1 = max_r \left\{ r \geq 1 + \frac{n_1^2}{n_1^2 - 2m_1} \geq 1 + \frac{n_1^2}{n_1^2 - 2(m - (n - r_0)r_0 - \binom{r_0}{2})} \right\}.$$

Tale procedimento può essere iterato fino a quando non esauriamo tutti i nodi. Sia C l'insieme contente tutte le r_i-clique risulta che $|C| = \sum_i \binom{r_i}{2}$. C per costruzione è chiaramente un cluster graph di conseguenza essendo k^* ottimale risulta che $k^* \leq m - \sum_i \binom{r_i}{2}$ oppure equivalentemente $k^* \leq \sum_i (n_i - r_i) r_i$, con $n_i = n_{i-1} - r_{i-1}$

Osservazione 6.3.1. Notare che $k_{i+1} = (n - r_i)r_i$ significa che ogni nodo della r_i-clique è connesso con ogni nodo all'esterno, ma se fosse così alla si potrebbe chiaramente dire che un qualunque nodo fuori la r_i-clique forma una clique di dimensione $r_i + 1$ insieme ai nodi della r_0-clique. Quindi possiamo dire che $k_{i+1} = (n - r_i)(r_i - 1)$ in quanto vogliamo che ogni nodo al di fuori della r_i-clique sia connesso al più con $r_i - 1$ nodi altrimenti si formerebbe la $(r_i + 1)$-clique.

□

CAPITOLO 7

CONCLUSIONI E SVILUPPI FUTURI

In questo capitolo vengono tratte le conclusioni della tesi e descritti i principali sviluppi futuri. In particolare la sezione 7.1 è dedicata alle conclusioni e ai principali risultati ottenuti e la sezione 7.2 ai sviluppi futuri e ai principali problemi aperti.

7.1 Conclusioni

I principali risultati ottenuti sono:

- Definizione di un modello di generazione uniforme per grafi lineari. Tale modello permette di generare un grafo lineare in modo incrementale sfruttando la sua caratteristica dell'ordinamento sui nodi.
- Progettazione e analisi di un algoritmo di programmazione dinamica di complessità nel caso pessimo di $\mathcal{O}(n^2)$ per il problema del Clustering Deletion su grafi lineari.

- Progettazione di un algoritmo basato sull'operazione di edge contraction per il problema del Clustering Deletion su grafi. Dallo schema di tale algoritmo sono stati ricavati algoritmi probabilistici ed euristici.

7.2 Sviluppi futuri

I pricipali sviluppi futuri su cui si può lavorare sono:

- Utilizzando il modello di generazione uniforme per grafi lineari, analizzare la complessità media dell'algoritmo di programmazione dinamica, al fine di esaminare quanto questa si discosti dalla complessità nel caso pessimo.

- Studio basato sull'osservazione che l'albero associato ai grafi lineari definisce una codifica prefisso.

- Problema aperto 5.1.1. In caso il problema possa essere risolto in modo positivo, questo risultato andrebbe studiato insieme all'analisi della complessità media dell'algoritmo di programmazione dinamica.

- Analisi dell'algoritmo greedy basato sulla funzione $g(,.,)$ per il problema del Clustering Deletion su grafi lineari.

- Analisi probabilistica dell'algoritmo probabilistico proposto per il problema del Clustering Deletion su grafi.

- Analizzare l'algoritmo probabilistico di edge contraction proposto per grafi qualsiasi adattato ai grafi lineari. Dove l'adattamento deve sfruttare la proprietà di contiguità della soluzione ottima per il problema del Clustering Deletion su grafi lineari.

- Analisi della bontà delle soluzioni degl'algoritmi euristici proposti.

- Progettazione di un algoritmo per il problema del Clustering Deletion su grafi lineari su un data stream. Questo algoritmo potrebbe essere

interessante al fine di un utilizzo nell'algoritmo di validazione delle RFD, discusso nel capitolo 3.

RINGRAZIAMENTI

Voglio ringraziare la mia famiglia, in particolare mia madre senza la quale sicuramente non sarei quello che sono, mia sorella perchè mi sopporta quando discutiamo e mio fratello che nonostante la lontananza è sempre nei miei pensieri.

Voglio ringraziare mio nonno Peppe, mia nonna Natalina e i miei zii materni in particolare zio Sandro e zia Francesca.

Voglio ringraziare il prof. Ugo Vaccaro che mi ha seguito per tutto lo svolgimento della tesi. Lo ringrazio per tutte le ore passate in chiamata su Teams e per tutti i consigli che mi ha dato. Questo percorso di tesi è stato una crescita a livello tecnico, ma ancora di più a livello personale e questo lo devo a lui. Poichè è stato proprio il percorso che avrei voluto, averlo scelto come relatore è stata la scelta migliore che potessi fare.

Voglio ringranziare il prof. Antonio Di Crescenzo che mi ha seguito nel percorso di tesi della laurea triennale.

Voglio ringranziare la prof.ssa Clelia De Felice la cui conoscenza al mio primo anno di università ha sicuramente migliorato il mio percorso di studi.

RINGRAZIAMENTI

Voglio ringraziere i miei amici di Napoli conosciuti in questi 5 anni. In particolare Gennaro, Leo, Emanuele e Domenico.

Voglio ringraziare il mio storico compagno di progetto Lorenzo per l'amicizia condivisa in questi 5 anni di università e per tutti i progetti fatti assieme.

Voglio ringraziare i miei storici amici di Modena Abdel e Petri.

Voglio ringraziare il mio socio di progettone Giuseppe. Conquistiamo il mondo bb.

Infine voglio ringraziare zio Peppe, per essere sempre stato a MENTALITÀ.

BIBLIOGRAFIA

[1] Ron Shamir, Roded Sharan e Dekel Tsur. «Cluster graph modification problems». In: *Discrete Applied Mathematics* 144.1-2 (2004), pp. 173–182.

[2] Pierre Hansen e Brigitte Jaumard. «Cluster analysis and mathematical programming». In: *Mathematical programming* 79.1 (1997), pp. 191–215.

[3] Assaf Natanzon. *Complexity and approximation of some graph modification problems*. University of Tel-Aviv, 1999.

[4] Paolo Atzeni et al. *Basi di dati: modelli e linguaggi di interrogazione (seconda edizione)*. McGraw-Hill, 2006.

[5] Loredana Caruccio, Vincenzo Deufemia e Giuseppe Polese. «Mining relaxed functional dependencies from data». In: *Data Mining and Knowledge Discovery* 34.2 (2020), pp. 443–477.

[6] Jonathan A Silva et al. «Data stream clustering: A survey». In: *ACM Computing Surveys (CSUR)* 46.1 (2013), pp. 1–31.

[7] Ron Shamir, Roded Sharan e Dekel Tsur. «Cluster graph modification problems». In: *International Workshop on Graph-Theoretic Concepts in Computer Science*. Springer. 2002, pp. 379–390.

[8] Ravi Boppana e Magnús M Halldórsson. «Approximating maximum independent sets by excluding subgraphs». In: *BIT Numerical Mathematics* 32.2 (1992), pp. 180–196.

[9] Anders Dessmark et al. «On the approximability of maximum and minimum edge clique partition problems». eng. In: *Conferences in Research and Practice in Information Technology Series*. Vol. 168. Australian Computer Society, 2006, pp. 101–105. ISBN: 1-920682-33-3. URL: http://crpit.com/abstracts/CRPITV51Dessmark.html.

[10] Peter J Looges e Stephan Olariu. «Optimal greedy algorithms for indifference graphs». In: *Computers & Mathematics with Applications* 25.7 (1993), pp. 15–25.

[11] *Graphclass: indifference*. URL: https://www.graphclasses.org/classes/gc_555.html.

[12] Toshiki Saitoh et al. «Random generation and enumeration of proper interval graphs». In: *IEICE TRANSACTIONS on Information and Systems* 93.7 (2010), pp. 1816–1823.

[13] *Numeri di Catalan*. URL: https://en.wikipedia.org/wiki/Catalan_number.

[14] Anders Dessmark et al. «On the approximability of maximum and minimum edge clique partition problems». In: *International Journal of Foundations of Computer Science* 18.02 (2007), pp. 217–226.

[15] *Maggiorante e minorante*. URL: https://it.wikipedia.org/wiki/Maggiorante_e_minorante.

[16] *Edge contraction*. URL: https://en.wikipedia.org/wiki/Edge_contraction.

[17] *Multi-insieme*. URL: https://it.wikipedia.org/wiki/Multiinsieme.

[18] *Distribuzione geometrica*. URL: https://it.wikipedia.org/wiki/Distribuzione_geometrica.