Università degli Studi di Camerino

Scuola di Scienze e Tecnologie

Corso di Laurea in Informatica (Classe L-31)

Sviluppo di una struttura dati per la trascrizione e gestione di sogni

Laureando
Marco Caputo

Matricola 119136

Relatore
Prof.ssa Emanuela Merelli

A.A. 2023/2024

Indice

Elenco dei codici

Elenco delle figure

Elenco delle tabelle

1. Introduzione

Numerosi e ragguardevoli sono stati i traguardi raggiunti dagli strumenti di elaborazione automatica di testi scritti sviluppati ed affinati negli ultimi decenni. In particolare, la branca dell'intelligenza artificiale dell'elaborazione automatica del linguaggio naturale (*Natural Language Processing* o *NLP*) ha visto una crescita esponenziale negli ultimi anni, grazie all'impiego di tecniche di deep learning e all'incremento della potenza di calcolo a disposizione. Tuttavia, nonostante i progressi compiuti circa le capacità, la comprensione del testo scritto rimane un compito complesso per i sistemi automatici, che richiedono enormi quantitativi di dati annotati per poter apprendere modelli di linguaggio sufficientemente accurati. Si pensi che i dataset per l'addestramento di modelli di linguaggio come *GPT-3* si sono rapidamente espansi, culminando in dimensioni dell'ordine del trilione di parole [4].

D'altra parte, l'applicazione di queste tecniche di analisi automatica di testi scritti o parlati sui sogni hanno già trovato applicazioni in ambito psicologico e psichiatrico, applicando tecniche di NLP su piccoli corpus di testi di sogni [1], mentre la rappresentazione di testi sui sogni attraverso grafi si è rivelato un'utile strumento a supporto della diagnosi e predizione di disturbi come la schizofrenia o il disturbo bipolare [9, 10] .

In questa tesi si discuterà di come una struttura di grafi a più livelli possa essere utilizzata per rappresentare e analizzare piccoli dataset di testi provenienti da trascrizioni di sogni e come essa possa essere sfruttata per realizzare dei modelli di linguaggio minimali rappresentativi di un particolare sognatore, dimostrando come informazioni legate alla semantica delle parole possano essere estrapolate a partire da aspetti sintattici.

1.1 Motivazione

Sebbene gli strumenti di NLP siano stati ampiamente utilizzati per l'analisi sintattica e semantica di testi scritti, le motivazioni che spingono alla realizzazione di una struttura dati applicabile alla trasposizione di sogni in semplici modelli di linguaggio sono legate al tentativo di individuare i rapporti sintattici e semantici tra parole e contesti di parole in relazione alla specifica persona, ovvero allo spazio semantico di un sognatore.

L'estrapolazione di informazioni legate al significato delle parole a partire da aspetti sintattici è un principio fondamentale della semantica computazionale, noto come *ipotesi distribuzionale*, che si basa sul principio per cui il significato di una parola è determinato dal suo contesto di utilizzo, e che le parole che appaiono nello stesso contesto

tendono ad avere significati simili.

Lo spazio semantico di un sognatore può essere rappresentato, quindi, attraverso una struttura basata su grafi in cui i nodi rappresentino le parole o i contesti di parole presenti nei sogni e gli archi siano ricavati dalle relazioni sintattiche tra di esse, come l'immediata vicinanza o la co-occorrenza.

realizzare una struttura dati generica per la rappresentazione di una gerarchia di grafi a più livelli, dove ogni livello rappresenti una diversa astrazione del grafo iniziale, e i grafi ai livelli inferiori possano essere ottenuti dall'espansione ricorsiva dei nodi ai livelli superiori.

Da questa necessità nasce l'idea di definire e realizzare nella maniera pi‘u generale possibile una struttura gerarchica di grafi a pi‘u livelli dove ogni livello rappresenti una diversa astrazione del grafo inziale, e i grafi ai livelli inferiori possono essere ottenuti dall'espansione ricorsiva dei nodi ai livelli superiori, permettendo espansioni locali o globali. Ci‘o pu‘o essere utile per descrivere caratteristiche strutturali di grafi grandi e sparsi a partire dall'individuazione di schemi topologici che siano significativi, che essi siano, ad esempio, cricche, componenti fortemente connesse, circuiti, stelle o cammini, e fornire operazioni che permettano di operare ad un determinato livello di astrazione nella gerarchia, come ottenere il grafo corrispondente ad un nodo, il calcolo della distanza o la modifica della struttura.

1.2 Obiettivi

L'obiettivo principale di questa tesi è, quindi, proporre una definizione formale della struttura dati astratta del *Grafo Muli-Livello* e delle operazioni che possono essere eseguite su di essa, nonché di valutarne complessità computazionale e spaziale.

Il secondo obiettivo è quello di analizzarne le possibili applicazioni, in particolare per la rappresentazione di spazi di parole e contesti in relazione ai sogni trascritti da un determinato sognatore.

1.3 Struttura della Tesi

La tesi è strutturata come segue:

- Nel Capitolo **??** verranno presentati i concetti di base relativi alla teoria dei grafi, con particolare attenzione agli aspetti legati al partizionamento e agli approcci multilivello esistenti per la risoluzione di problemi di partizionamento su grafi.

- Nel Capitolo **??** verranno illustrati gli algoritmi per l'individuazione di pattern strutturali in grafi utili per la costruzione di una gerarchia di grafi a più livelli.

- Nei Capitoli **??** verrà presentata e definita la struttura dati del Grafo Multi-Livello, le operazioni che possono essere eseguite su di essa e i relativi algoritmi

- Nei Capitoli **??** verranno discusse le possibili applicazioni del Grafo Multi-Livello, evidenziandone limiti e punti di forza. Particolare attenzione sarà rivolta all'ambito della trascrizione dei sogni, e verranno discussi i risultati preliminari ottenuti applicando la struttura dati a dataset di sogni di esempio.

2. Grafi e Approccio Multi-Livello

In questo capitolo sono presentati alcuni concetti introduttivi utili alla definzione e alla comprensione dei *Grafi Multi-livello*. Verrà esplorato il concetto fondamentale di grafo, una struttura matematica in grado di rappresentare relazioni tra elementi discreti, e verranno illustrati i fondamenti della teoria dei grafi [5, 7], la disciplina che si occupa dello studio di queste strutture, utile in svariati ambiti applicativi, come l'informatica, l'ingegneria, la biologia, la chimica ed altri. Maggiore attenzione sarà rivolta alle definizioni pertinenti al partizionamento e alla contrazione di grafi [12], vicine alle caratteritiche salienti dei *Grafi Multi-livello*, evidenziando gli aspetti già trattati nella letteratura esistente e quelli che verranno approfonditi in questa tesi.

2.1 Cenni di Teoria dei Grafi

Un grafo è una struttura matematica costruita su un insieme di elementi in cui coppie di elementi possono essere in relazione tra loro. I grafi possono essere orientati o non orentati, a seconda che esista una direzione o un ordine tra le coppie di elementi che si trovano in relazione. In questa sezione ci concentreremo eclusivamente sui grafi diretti, in quanto più generali, visto che grafi non orientati possono sempre essere rappresentati come particolari grafi orientati, ed in quanto la struttura dei *Grafi Multi-livello* si basa su di essi.

2.1.1 Grafo Orientato

Definizione 2.1.1 (Grafo orientato)
*Un **grafo orientato** G è una coppia (V, E), dove:*

- $V = \{v_1, v_2, \ldots, v_n\}$ *è un di un insieme finito non vuoto di elementi detti **nodi** (o **vertici**).*
- $E = \{(v_i, v_j) \mid v_i, v_j \in V\} \subseteq V \times V$ *è un insieme di coppie ordinate di nodi dette **archi** (o **spigoli**).*

Nelle rappresentazioni grafiche dei grafi orientati, i nodi sono solitamente rappresentati come cerchi o punti, mentre gli archi come frecce. Nella figura 2.1 è mostrato un esempio di grafo orientato con insieme di nodi $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$ e insieme di archi $E = \{(v_1, v_2), (v_2, v_1), (v_2, v_3), (v_3, v_1)(v_3, v_4),$
$(v_4, v_4), (v_5, v_6), (v_6, v_5)\}$.

Si noti che sono ammessi **cappi**, ovvero archi che collegano un nodo a se stesso, ma nella normale nozione di grafo orientato non sono ammessi archi multipli tra due nodi.

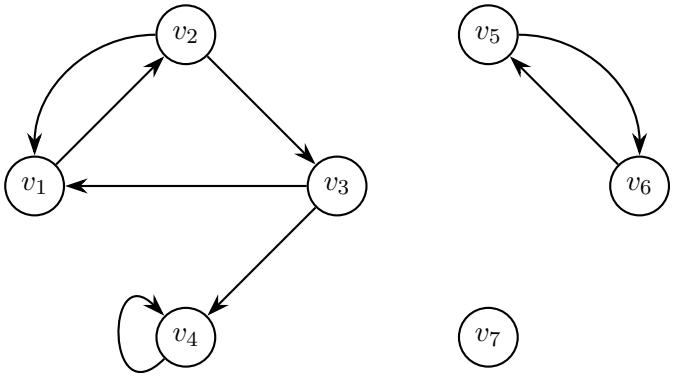

Figura 2.1: Esempio di grafo orientato

Le cardintalità degli insiemi di nodi e archi di un grafo orientato sono rispettivamente $|V|= n$ e $|E|= m$, e vengono dette rispettivamente **ordine** e **dimensione** del grafo.

Essendo definiti su un insieme di elementi e di archi, possono essere definite relazioni di inclusione tra grafi. Un grafo $G' = (V', E')$ è un sottografo di $G = (V, E)$, e lo si indica con $G' \subseteq G$ se $V' \subseteq V$ e $E' \subseteq E$.
Inoltre, dato un certo insieme $V' \subseteq V$, si definisce il sottografo di G **indotto** da V', e lo si indica con la notazione $G[V']$, il grafo avente come insieme di nodi V' e come insieme di archi l'insieme di tutti gli archi in G che rappresentino relazioni tra tali nodi, ovvero il grafo $G' = (V', E')$ dove $E' = \{(u, v) \in E : u, v \in V'\}$.

Essendo il contenuto informativo rilevante di un grafo orientato contenuto nei suoi archi, e quindi nelle relazioni tra nodi, il concetto di eguaglianza tra grafi orientati non è banale. Una relazione tra grafi orientati, utile per valutare la loro equivalenza in termini di informazione espressa, è l'isomorfismo (dal greco *iso* = uguale e *morphè* = forma). Così come per tutte le struttre matematiche, intuitivamente, due grafi si dicono **isomorfi** quando per ogni parte della struttura di uno esiste una corrispondente parte della struttura dell'altro, e viceversa. Formalmente, due grafi orientati $G = (V, E)$ e $H = (W, F)$ si dicono isomorfi, e lo si indica con $G \cong H$ se esiste una biiezione $f : V \to W$ tale per cui $(u, v) \in E$ se e solo se $(f(u), f(v)) \in F$ per ogni $u, v \in V$.

2.1.2 Archi e nodi

A seguire alcune definizioni relative ai nodi e agli archi di un grafo orientato:

Sia $(u, v) \in E$ un arco di un grafo orientato $G = (V, E)$, allora:

- l'arco (u, v) **esce** dal nodo u ed entra nel nodo v. Ad esempio, gli archi uscenti dal nodo v_2 nel grafo della figura 2.1 sono (v_2, v_1) e (v_2, v_3), mentre l'unico arco entrante nel nodo v_5 è (v_6, v_5).

- l'arco (u, v) si dice **incidente** in entrambi i vertici u e v.
- il nodo v è detto **adiacente** al nodo u, in quanto esiste un arco $(u, v) \in E$.

Sia $v \in V$ un nodo di un grafo orientato $G = (V, E)$, allora:

- il **grado uscente** di un nodo v è il numero di archi che escono da v.
- il **grado entrante** di un nodo v è il numero di archi che entrano in v.
- il **grado** di un nodo v è la somma del grado uscente e del grado entrante di v.

2.1.3 Cammini

I cammini sono concetti fondamentali della teoria dei grafi e sono alla base di molti algoritmi e problemi noti relativi ai grafi.

Sia $G = (V, E)$ un grafo orientato, siano $u, v \in V$ due nodi di G, allora un **cammino** da u a v in G è una sequenza ordinata di nodi $\langle v_0, v_1, \ldots, v_k \rangle$ tale che $(v_i, v_{i+1}) \in E$ per ogni $i = 0, 1, \ldots, k-1$ con $v_1 = u$ e $v_k = v$. La **lunghezza** k di un cammino è data dal numero di archi che lo compongono. Ad esempio, $\langle v_1, v_2, v_3, v_4 \rangle$ è un cammino di lunghezza 3 nel grafo della figura 2.1.

Se esiste un cammino p da u a v in G, allora si dice che il nodo v è **raggiungibile** da u attraverso p in G, e questo può essere indicato con la notazione $u \overset{p}{\rightsquigarrow} v$.

A seguire alcune definizioni relative ai cammini su un grafo orientato:

- Un cammino si dice **semplice** se non contiene nodi ripetuti, ad eventuale eccezione del primo e dell'ultimo nodo.
- Un cammino si dice **elementare** se non contiene archi ripetuti. Si noti che un cammino semplice è sempre elementare.
- Un cammino $\langle v_0, v_1, \ldots, v_k \rangle$ di lunghezza $k \geq 1$ si dice **ciclo** se $v_1 = v_k$, ovvero se il suo nodo iniziale coincide con il suo nodo finale. Un **ciclo semplice** è un cammino in cui tutti i nodi sono distinti, ad eccezione del primo e dell'ultimo nodo, mentre un **ciclo elementare** (o **circuito**) è un ciclo in cui tutti gli archi sono distinti. Ad esempio, nel grafo in figura 2.1, il cammino $\langle v_1, v_2, v_3, v_1 \rangle$ è un circuito semplice di lunghezza 3. Inoltre, un grafo diretto che non contiene cicli semplici è detto grafo diretto **aciciclico** (o **DAG**).
- Un cammino si dice **cammino hemiltoniano** in nel grafo G se attraversa ogni nodo di G esattamente una volta.
- Un cammino $\langle v_0, v_1, \ldots, v_k \rangle$ si dice **ciclo hemiltoniano** in nel grafo G se esso è un ciclo e ogni nodo di G appare una ed una sola volta tra i nodi $\langle v_0, v_1, \ldots, v_{k-1} \rangle$ e $v_k = v_0$.
- Un cammino si dice **cammino euleriano** nel grafo G se attraversa ogni arco di G esattamente una volta.
- Un cammino si dice **ciclo euleriano** nel grafo G se esso è un ciclo e ogni arco di G appare una ed una sola volta tra gli archi del ciclo.

2.1.4 Connessione tra nodi

Una caratteristica importante dei grafi orientati, basata sul concetto di raggiungibilità e adiacienza, è la connessione dei suoi nodi.

Un grafo orientato $G = (V, E)$ si dice **fortemente connesso** se per ogni coppia di nodi $u, v \in V$ esiste un cammino da u a v.
In un tale grafo, quindi, ogni nodo è mutualmente raggiungibile da ogni altro nodo. Le **componenti fortemente connesse** di un grafo sono le classi di equivalenza dei nodi secondo la relazione ëssere mutualmente raggiungibili". Ad esempio, nel grafo in figura 2.1, le componenti fortemente connesse sono $\{\{v_1, v_2, v_3\}, \{v_4\}, \{v_5, v_6\}, \{v_7\}\}$. Si noti che l'insieme di nodi V di un grafo fortemente connesso è per definizione una unica componente connessa.

Un maggiore grado di connessione tra nodi è dato dalla presenza di singoli archi tra ogni coppia di nodi anzichè di cammini.
Un grafo orientato $G = (V, E)$ si dice **completo** se esiste un arco $(u, v) \in E$ per ogni coppia di nodi distinti $u, v \in V$. In un tale grafo, quindi, ogni coppia di nodi distinti è adiaciente. Le **cricche** di un grafo sono le classi di equivalenza dei nodi secondo la relazione "essere mutualmente adiacienti". Ad esempio, nel grafo in figura 2.1, le cricche sono $\{\{v_1, v_2\}, \{v_3\}, \{v_4\}, \{v_5, v_6\}, \{v_7\}\}$. Si noti che l'insieme di nodi V di un grafo completo è per definizione un'unica cricca.

2.2 Contrazione di Grafi

Nella teoria dei grafi la contrazione di un grafo è un'operazione che permette di ridurre la dimensione di un grafo senza alterarne la struttura fondamentale.
La contrazione di archi o di sottoinsiemi di nodi è un operazione fondamentale nella teoria dei grafi minori, dove si studiano le proprietà di un grafo in relazione alla presenza di sottostrutture minori ottenibili attraverso rimozione di archi e nodi o contrazioni. Queste tecniche di contrazione trovano applicazione in tutti quei casi in cui si vuole semplificare un grafo identificando i vertici che possono essere considerati equivalenti in relazione ad una certa proprietà, e risultano essere utili in svariati problemi di ottimizzazione e partizionamento di grafi. In letteratura, le operazioni di contrazione di grafi sono state utilizzate anche a scopo di compressione di grafi, al fine di renderli più compatti e trattabili con algoritmi di analisi altrimenti troppo costosi, individuando schemi di contrazione d'interesse e cercando di evitare perdita di informazione [6].

2.2.1 Contrazione di archi

La contrazione di archi, spesso riferita come **contrazione di spigoli**, di un grafo orientato $G = (V, E)$ è un'operazione che consiste nella rimozione di un arco $e = (u, v) \in E$ e nella simultanea fusione dei nodi u e v in un unico nodo w. Quando ciò avviene, tutti gli archi che entrano in u e v diventano archi entranti in w, e, analogamente, tutti gli archi che escono da u e v diventano archi uscenti da w. Il risultato di una tale operazione è, quindi, un nuovo grafo ottenuto da G mediante la contrazione dell'arco e, che può essere indicato con G/e (da non confondersi con la sottrazione insiemistica $\setminus$). Si noti

che, secondo la definzione data, una tale operazione applicata ad un grafo orientato semplice può risultare in un grafo con archi multipli e cappi, a seconda della struttura del grafo iniziale, e per questo è spesso previsto nella definzione di contrazione di archi che vengano applicate le ulteriori operazioni necessarie ad ottenere come risultato un nuovo grafo semplice.

Definizione 2.2.1 (Contrazione di archi)

Sia $G = (V, E)$ un grafo orientato e sia $e = (u, v) \in E$ un arco di G con $u \neq v$, sia f una funzione su V che associa ogni nodo in $V \setminus \{u, v\}$ a se stesso, o ad un nuovo nodo w altrimenti.
La contrazione di e su G è un nuovo grafo $G' = (V', E')$ dove:

- $V' = (V \setminus \{u, v\}) \cup \{w\}$ *con* $w \notin V$
- $E' = \{(f(x), f(y)) \mid (x, y) \in E \setminus \{e\}\}$

In figura 2.2 è mostrato un esempio di contrazione di un arco (u, v) in un nuovo nodo w in un grafo orientato, che include la rimozione di archi multipli e di cappi. Più in generale, una tale operazione può essere eeguita su un insieme di archi, contraendo ciacuno di essi in un qualsiasi ordine.

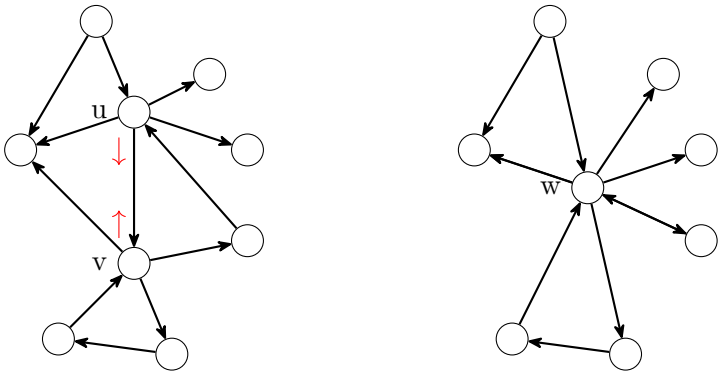

Figura 2.2: Esempio di contrazione di un arco in un grafo orientato

2.2.2 Contrazione di sottografi

Un'operazione simile alla contrazione di archi, ma più generale, è la **contrazione di vertici** (o **identificazione di vertici**) di un grafo. Essa può essere vista come una generalizzazione della contrazione di archi, in quanto rimuove la restrizione che la coppia di nodi da contrarre sia adiacente, rendendo la contrazione per archi un suo caso particolare. Si immagini, pertanto, di avere il grafo a sinistra della figura 2.3 privato, però, dell'arco (u, v). La contrazione per vertici permetterebbe di contrarre la coppia non adiacente di nodi u e v, risultando, comunque, nel grafo a destra della figura 2.3.

L' operazione di contrazione di vertici può essere generalizzata nella **contrazione di sottografi**, un'operazione che permette di contrarre un qualsiasi sottoinsieme di nodi di un grafo in un unico nodo. Dato un grafo $G = (E_G, V_G)$ ed un suo sottografo

$H = (V_H, E_H)$, quindi, il grafo risultante dalla contrazione di H mantiene tutti gli archi incidenti su coppie di nodi in $E_G \setminus E_H$, sostituendo quegli archi incidenti tra nodi in $V_G \setminus V_H$ e V_H con nuovi archi incidenti sul nuovo nodo contratto.

Definizione 2.2.2 (Contrazione di sottografi)

Sia $G = (V, E)$ un grafo orientato, sia $W \subseteq V$ un sottoinsieme di nodi di G, sia $H = G[W] = (W, F)$ il sottografo indotto da W in G. Sia f una funzione su V che associa ogni nodo in $V \setminus W$ a se stesso, o ad un nuovo nodo w altrimenti. La contrazione di H su G è un nuovo grafo $G' = (V', E')$ dove:

- $V' = (V \setminus W) \cup \{w\}$ *con* $w \notin V$
- $E' = \{(f(u), f(v)) \mid (u, v) \in E \setminus F\}$

In figura 2.3 è mostrato un esempio di contrazione di un sottografo $G[\{v_1, v_2, v_3, v_4\}]$ del grafo orientato G in un nuovo nodo w, che include la rimozione di archi multipli e di cappi.

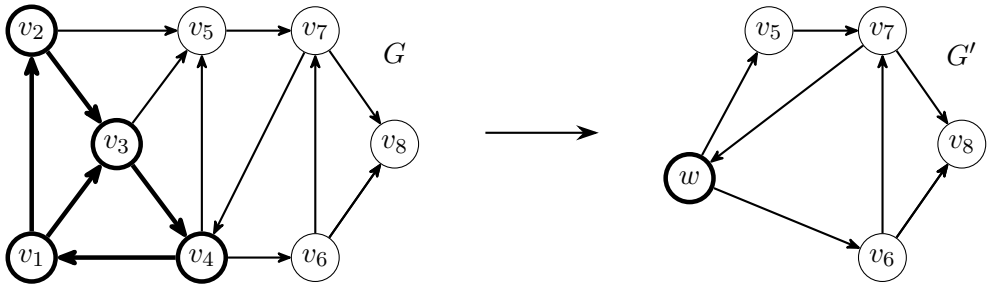

Figura 2.3: Esempio di contrazione di un sottografo in un grafo orientato

Alla luce delle definizioni delle operazioni presentate, valgono le seguenti considerazioni:

- Il risultato della contrazione di una coppia di nodi adiacienti su un certo grafo G può produrre un grafo isomorfo a quello della contrazione di una coppia di nodi non adiacenti in un altro grafo G' non isomorfo a G. E' il caso precedentemente considerato applicato al grafo a sinistra in figura 2.2.
- Il risultato della contrazione di un sottografo su un certo grafo G può produrre un grafo isomorfo a quello della contrazione di un sottografo in un altro grafo G' non isomorfo a G. Come esempio analogo, basta considerare un grafo J ottenuto a apartire dal grafo G a sinistra in figura 2.3 rimuovendo il nodo v_1 e i suoi archi incidenti. I grafi risultanti dalla contrazione di $G[\{v_1, v_2, v_3, v_4\}]$ in G e dalla contrazione di $J[\{v_2, v_3, v_4\}]$ in G' sono certamente isomorfi.

Questo significa che la contrazione di vertici e di sottografi non sono operazioni invertibili, in quanto rappresentano funzioni suriettive, e quindi non iniettive. Di fatti queste operazioni non mantengono alcuna informazione legata alla struttura originale del grafo su cui sono applicate. Come mostrato nei prossimi capitoli, tra gli obiettivi della definizione del grafo multi-livello, vi è proprio quello di mantenere le informazioni legate alla struttura dei grafi a cui sono applicate contrazioni, permettendo anche operazioni di decontrazione.

2.2.3 Grafi quoziente

Nella teoria dei grafi, un grafo quoziente è una visione astratta di un grafo partizionato in sottoinsiemi di nodi che rappresenta le relazioni tra tali sottoinsiemi. In un grafo quoziente G' ottenuto a partire da un grafo $G = (V, E)$, i nodi rappresentano blocchi di nodi di G che fanno parte dello stesso insieme per una qualche partizione di V. Per quanto riguarda gli archi di G', dati due blocchi di nodi B_1 e B_2 in G', un arco tra B_1 e B_2 sta ad indicare la presenza di almeno un arco tra un nodo di B_1 e un nodo di B_2 in G.

Se intuitivamente si potrebbe dire che il grafo quoziente permette di accorpare gruppi di nodi e archi tra loro per formare un nuovo grafo, una descrizione più formale utilizzerebbe il concetto di contrazione di sottografi, definendo il grafo quoziente come il risultato delle contrazioni dei sottografi indotti dalla data partizione di nodi.

Definizione 2.2.3 (Grafo Quoziente)
Sia $G = (V, E)$ un grafo orientato, sia $P \subseteq \mathcal{P}(V)$ una partizione di V, sia R la relazione d'equivalenza su V indotta dalla partizione P. Il grafo quoziente di G rispetto a P è il grafo $G' = (V', E')$ dove:

- *V' è l'insieme quoziente V/R, ovvero l'insieme delle classi di equivalenza di R su V.*
- *$E' = \{([u]_R, [v]_R) \mid (u, v) \in E\}$, dove $[u]_R$ e $[v]_R$ sono rispettivamente le classi di equivalenza dei nodi u e v rispetto a R.*

La figura 2.4 mostra un esempio di grafo quoziente sulla destra ottenuto a partire dal grafo orientato sulla sinistra e una partizione $P = \{A, B, C\}$.

Come evidente dalla definzione, il nome del grafo quoziente è dovuto al fatto che la sua struttura è strettamente legata all'insieme quoziente di una qualche relazione di equivalenza definita sui nodi del grafo. Sebbene, assieme al grafo di partenza, l'ingrediente fondamentale per la definizione di un grafo quoziente sia la una partizione dei suoi nodi, una relazione di equivalenza sugli stessi sarebbe un parametro equivalente, in quanto ogni relazione di equivalenza induce una partizione degli elementi del suo dominio in classi di equivalenza.

Le relazioni di equivalenza, cosí come le partizioni, possono essere comparate tra loro secondo il concetto di raffinamento: una relazione di equivalenza R_1 si dice **più fine** (in inglese **finer**) di un'altra relazione di equivalenza R_2 se ogni classe di equivalenza di R_1 è contenuta in una classe di equivalenza di R_2. In tal caso si dice che R_2 è **più grezza** (in inglese **coarser**) di R_1, in quanto ogni classe di equivalenza di R_2 può essere ottenuta come l'unione di classi di equivalenza di R_1.

Per questo è interessante notare che tale concetto di finezza può essere facilmente esteso ai grafi quoziente:

- Ogni grafo può banalmente considerarsi come il grafo quoziente di se stesso rispetto alla relazione di equivalenza di ugualianza, in quanto ogni nodo di un grafo è uguale unicamente a se stesso. La relazione di equivalenza di ugualianza, infatti, è la relazione di equivalenza più fine, e per questo genera il grafo quoziente più fine possibile a partire da qualunque grafo.

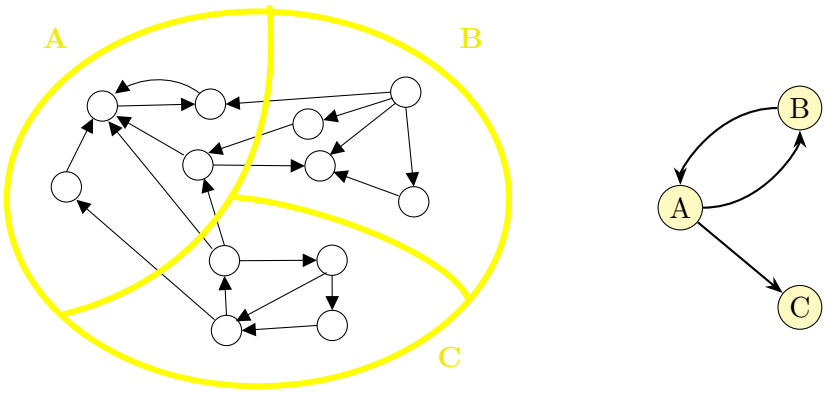

Figura 2.4: Esempio di grafo quoziente di un grafo orientato

- Analogamente, il grafo composto di un unico nodo e nessun arco risulta il grafo quoziente di ogni grafo rispetto alla relazione di equivalenza universale, che mette in relazione qualsiasi coppia di elementi ed identifica tutti i nodi di qualunque grafo in un unico blocco. La relazione di equivalenza universale, infatti, è la relazione di equivalenza più grezza e, come è intuibile pensare, genera il grafo quoziente più grezzo possibile a partire da qualunque grafo.

Una particolare relazione di equivalenza che ben si presta alla definizione di un grafo quoziente è la relazione di mutua raggiungibilità tra nodi di un grafo, che ne definisce le componenti fortemente connesse. Il grafo quoziente di un grafo rispetto a tale relazione di equivalenza prende il nome di **condensazione** (o grafo delle componenti fortemente connesse), e si dimostra essere un grafo diretto aciclico.

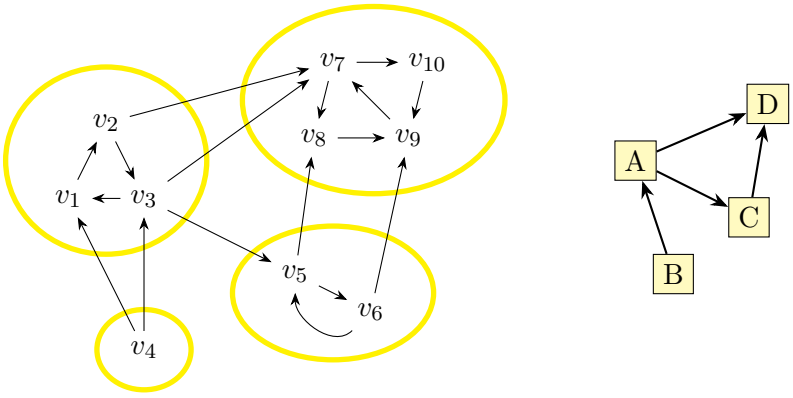

Figura 2.5: Esempio di condensazione di un grafo orientato

In figura 2.5 è mostrato un esempio di grafo orientato sulla sinistra, in cui le componenti fortemente connesse sono evidenziate in giallo, e al sua condensazione sulla destra.
Si noti che il grafo condensato, in quanto aciclico, non contiene cicli semplici.
Se si volesse aggiungere un arco affinchè il grafo condensato contenesse un ciclo, ad esempio aggiungendo un arco uscente da un nodo nella compoente C e entrante in un

nodo nella componente A, si otterrebbe una nuova componente fortemente connessa data dall'unione delle componenti A, B e C, ovvero le componenti i cui corrispondenti nodi nel grafo condensato sarebbero contentenuti in un ciclo semplice.

2.3 Approccio Multi-Livello

In questa sezione verrà introdotto il concetto di contrazione a più livelli di grafi.
L'idea di base di un approccio multi-livello è quella di partire da un grafo di partenza e di applicare ripetutamente operazioni di contrazione, ottenendo una sequenza di grafi contratti di dimensione via via minore. Sebbene tale approccio sia stato generalmente utilizzato per ridurre la dimensione di un grafo al fine di applicare efficientemente algoritmi di partizionamento [11], non mancano esempi di applicazioni in contesti diversi, come quello della visualizzazione di grafi di grandi dimensioni [2].

In ogni caso, lo scopo di un approccio multi-livello è quello di costruire una gerarchia di grafi contratti che rappresentino la struttura del grafo di partenza, comprimendo in nodi in meta-nodi in accordo a determinate caratteristiche di interesse. Così come per la terminologia usata per le partizioni, i grafi risultanti dalle contrazioni sono spesso detti grossolani (*coarse*), mentre quelli risultanti dalle decontrazioni sono detti raffinati (*fine*). I grafi grossolani, ottenuti ricorsivamente a partire dai grafi più raffinati, sono quindi da considerarsi come rappresentazioni più astratte di questi ultimi.

2.3.1 Partizionamento multilivello di grafi

Il **partizionamento multilivello di grafi** (in inglese *multilevel graph partitioning* o *MGP*) è un approccio euristico per la risoluzione di problemi su grafi in cui si vuole dividere un grafo in un dato numero di blocchi che abbiano approssimativamente la stessa dimensione, affinchè una certa funzione obiettivo sia minimizzata.
Un esempio di tale problema dalla grande utilit'a pratica è quello in cui l'obiettivo del partizionamento è quello di minimizzare il numero di archi che connettono i blocchi, che trova applicazione in importanti contesti legati all'informatica, ad esempio per la decomposizione di strutture dati per la computazione parallela, e all'ingegneria, ad esempio per il partizionamento di circuiti integrati.

L'approccio multi-livello, per la prima volta introdotto da Hendrickson e Leland nel 1995 [**Hendrickson:1995:MPG:221253.221279**], si è rivelato essere quello di maggior successo per la risoluzione di problemi di partizionamento di grafi di grandi dimensioni, in quanto permette di ottenere partizioni di alta qualità in tempi ragionevoli, nonostante il problema di partizionamento sia NP-completo per la maggior parte delle funzioni obiettivo.
L'utilità di questa tecnica si basa sull'intuizione per cui una buona partizione ad un livello grossolano della gerarchia rimarrà tale anche ad un livello più raffinato, e che, in questo modo, la ricerca di una partizione ottimale può essere effettuata su grafi più piccoli e più semplici.

L'approccio multi-livello per la partizione di grafi si articola in tre fasi principali:

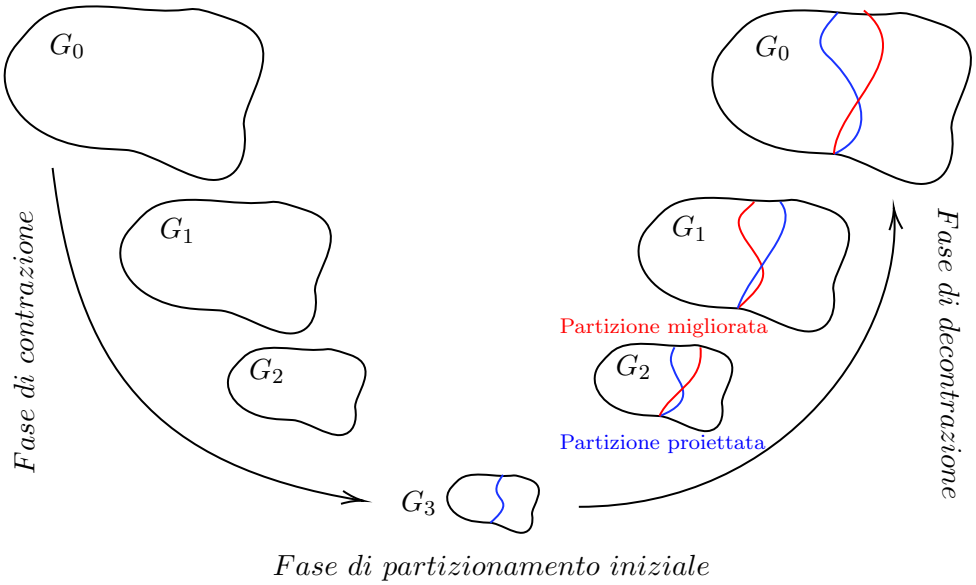

Figura 2.6: Schema grafico del partizionamento multilivello

1. **Fase di contrazione** (*contraction/coarsening phase*): si crea una gerarchia di grafi riducendo iterativamente la dimensione del grafo iniziale. Questo viene fatto comunemente individuando e contraendo coppie di nodi adiacenti, ovvero individuando un sottoinsieme degli archi del grafo da contrarre, $M \subseteq E$ detti *match*. Si noti che la scelta di contrarre coppie di nodi adiacienti porta a grafi grossolani i cui nodi rappresentano sottografi del grafo iniziale densamente connessi. In base allo specifico problema, una determinata funzione di *rating* classifica gli archi individuando quale sottoinsieme degli archi E debba essere assegnato ad M affinchè la somma dei *rating* degli in M sia globalmente massimizzata. Si consideri che un nodo già contratto non può essere più coinvolto in un ulteriore *matching* allo stesso livello della gerarchia.

2. **Fase di partizionamento iniziale** (*initial partitioning phase*): quando a seguito delle contazioni il grafo risulta essere di ordine abbastanza piccolo, in relazione ad un qualche threshold, esso può essere partizionato direttamente con algoritmi costosi, fornendo una partizione inizile sul grafo più grossolano della gerarchia.

3. **Fase di decontrazione** (*refinement/uncoarsening phase*), i matchings vengono iterativamente decontratti, e i relativi nodi vengono associati a blocchi della partizione del grafo più grossolano, proiettandola sul grafo più raffinato. Per fare in modo che la partizione al livello più grossolano tenga conto delle sotto-strutture ai livelli più raffinati, un algoritmo di miglioramento locale (*local improvement*) ricolloca i nodi tra i blocchi per migliorare la dimensione del taglio o l'equilibrio delle dimensioni tra i blocchi.

3. Grafi Multi-livello

Come si è potuto vedere dal contenuto del Capitolo 1, le operazioni di contrazione e la costruzione di gerarchie di grafi a più livelli siano concetti già ampiamente utilizzati e consolidati nella teoria dei grafi e nelle sue applicazioni. Tuttavia, ciò che non è stato ancora propriamente considerato nella letteratura esistente è la possibilità di definire e formalizzare una vera e propria struttura dati astratta che rappresenti un grafo multi-livello come un'entità a sè stante. In questo capitolo saranno proposte definizioni e proprietà di base di una struttura dati per la rappresentazione di gerarchie di grafi a più livelli, in cui sia possibile ottenere informazioni sull'intera struttura gerarchica in relazione ai singoli grafi che la compongono, come attraverso l'espansione e contrazione di nodi. In particolare, riprendendo dalla terminologia e dai concetti esistenti in letteratura, si proporranno definizioni orginali di *Grafi decontraibili* e di *Grafi multi-livello*.

3.1 Grafo decontraibile

A partire dal concetto noto di grafo quoziente e di contrazione, si vuole definire una particolare tipologia di grafi quoziente che, oltre a rappresentare le caratteristiche strutturali ad alto livello dei grafi da cui sono derivati, siano in grado di mantenere l'informazione originale di questi ultimi, ed in che modo essa sia legata alla sua rappresentazione astratta.
Nasce così il concetto di grafo decontraibile, che intuitivamente può essere considerato come un grafo in cui i nodi sono riconducibili ad un grafo e gli archi ad un insieme di archi tra i nodi dei grafi associati ai nodi coinvolti.

Definizione 3.1.1 (Grafo Decontraibile)
Un ***grafo decontraibile*** *è una quadrupla* $G = (V, E, dec_V, dec_E)$ *dove:*

- V *è un insieme di elementi detti* ***supernodi****;*
- $E \subseteq V \times V$ *è un insieme di coppie ordinate di supernodi, dette* ***superarchi****;*
- $dec_V : V \to \mathcal{G}_D$ *è una biiezione tale per cui* $dec_V(v) = (\mathcal{V}_v, \mathcal{E}_v, dec_{\mathcal{V}_v}, dec_{\mathcal{E}_v})$ *è un grafo decontraibile rappresentato dal supernodo* v*;*
- $dec_E : E \to (\mathcal{V} \times \mathcal{V})$ *con* $\mathcal{V} = \bigcup_{v \in V} \mathcal{V}_v$*, è una biiezione tale per cui* $\forall\ e = (u, v)$*,* $dec_E(e) = \mathcal{E}_e \subseteq \{(a, b) \mid a \in \mathcal{V}_u \wedge b \in \mathcal{V}_v\}$ *è un insieme di archi rappresentati dal superarco* e*.*

Nella definzione, così come d'ora in avanti, si utilizzerà la notazione $\mathcal{V}$ e $\mathcal{E}$ per indicare, rispettivamente, insiemi di supernodi e superarchi ottenibili attraverso decontrazioni

e, quindi, di livello inferiore rispetto al grafo decontraito di riferimento. Nello specifico, le notazioni $\mathcal{V}_v$ e $\mathcal{E}_v$ saranno utilizzate per indicare, rispettivamente, insiemi di nodi e archi di grafi ottenibili attraverso la decontrazione di un certo supernodo v. Nel contesto di un determinato grafo decontraibile, per portare maggiore distinzione si continuerà ad utilizzare il termine di nodo ed arco per riferirsi ai supenodi e superarchi di tale livello inferiore.

Si noti che è possibile usare una notazione basata su attributi alternativa a quella delle funzioni per descrivere le proprietà caratteristiche di nodi ed archi che rendono tale un grafo decontrabile. Tale notazione sarà utilizzata in seguito per semplificare la descrizione di algoritmi.
In particolare, si può definire un grafo decontraibile come un normale grafo diretto sotto forma di coppia (V, E) dove:

- $\forall\ v \in V$, è definito un attributo $v.dec = G_v$ dove $G_v = (\mathcal{V}_v, \mathcal{E}_v)$ è un grafo decontraibile rappresentato da v, quindi tale per cui $dec_V(v) = v.dec$.
- $\forall\ e = (u, v) \in E$, è definito una attributo $e.dec = E_e$ dove $\mathcal{E}_e \subseteq \{(a, b) \mid a \in \mathcal{V}_u \wedge b \in \mathcal{V}_v\}$ è un insieme di archi rappresentati da e, quindi tale per cui $dec_E(e) = e.dec$.

Si consideri la natura ricorsiva definizione di grafo decontraibile, per cui se un supernodo v può essere rappresentato da un grafo decontraibile G_v, allora i nodi di G_v saranno a loro volta dei supernodi. Stessa cosa vale per gli archi, che possono essere rappresentati da insiemi di archi tra i nodi dei grafi. La scelta di rendere il grafo decontraibile una struttura ricorsiva, così come la definizione in se stessa, è utile alle successive definizoni legate ai grafi multi-livello.

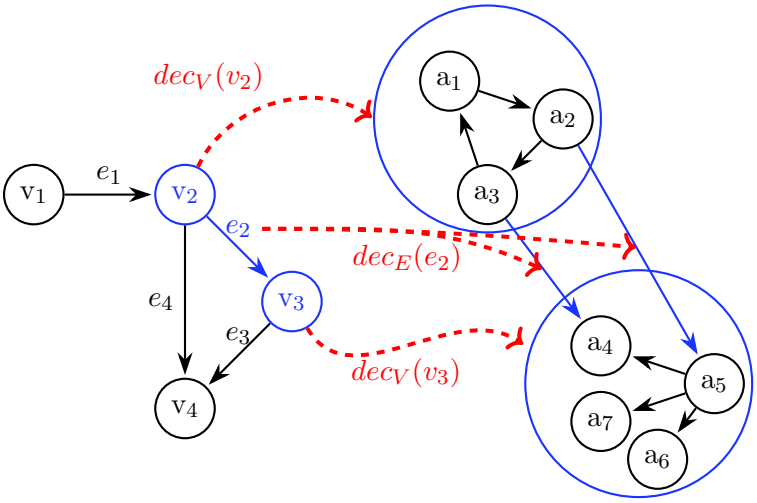

Figura 3.1: Un esempio di decontrazione locale di un grafo decontraibile

In Figura 3.1 è mostrato un esempio di grafo decontraibile sulla sinistra, mentre sulla destra sono rappresentati i grafi associati ai supernodi v_2 e v_3, assieme all'insieme di archi associato al super-arco e_2. Si osservi che il grafo $dec_V(v_2)$ composto dai nodi a_1,

a_2 e a_3, ad esempio, manca degli archi collegati esternamente che definiscono il contesto in cui tale grafo si colloca, e solo uno degli archi incidenti in v_1 è stato espanso. Questo fornisce una visione parziale dell'informazione contenuta nel grafo decontraibile a sinistra, e per questo si può dire che il grafo a destra è il risultato di un'espansione locale.

Inoltre, dal momento in cui i grafi decontraibili sono a tutti gli effetti dei grafi, tutte le definizioni date sui grafi standard continuano ad essere utilizzate in modo equivalente per i grafi decontraibili. Analogamente, un supernodo può essere considerato come un particolare tipo di nodo, e lo stesso vale per i superarchi. In particolare, il concetto di isomorfismo può essere esteso ai grafi decontraibili senza particolari modifiche, ignorando l'aspetto decontraibiledi nodi e archi, ovvero le funzioni dec_V e dec_E di entrambi i grafi di cui si vuole valutare l'isomofismo.

3.2 Contrazioni

A partire dalla decontrazione, che rappresenta un aspetto intrinseco alla definzione dei grafi decontrabili, la relazione di contrazione è quella che, intuitivamente, permette di legare grafi decontraibili nel senso opposto. Se da un lato la decontrazione permette di costruire grafi che rappresentino espansioni locali, con una definizone di contrazione si vogliono stabilire le condizioni che permettono di legare interi grafi decontraibili ad altri che ne forniscano una loro rappresentazione astratta, affinchè la tale rappresentazione sia coerente con il concetto di contrazione esistente nella teoria dei grafi.

Definizione 3.2.1 (Contrazione di un Grafo Decontraibile)
*Sia $G = (V, E, dec_V, dec_E)$ un grafo decontraibile, il grafo decontraibile $G' = (\mathfrak{V}, \mathfrak{E}, dec_{\mathfrak{V}}, dec_{\mathfrak{E}})$ è una sua **contrazione** se e solamente se:*

- *l'insieme $\{V_\alpha \mid \alpha \in \mathfrak{V}\}$ è una partizione di V*
- *l'insieme $(\{E_\alpha \mid \alpha \in \mathfrak{V}\} \setminus \{\emptyset\}) \cup \{dec_{\mathfrak{E}}(\epsilon) \mid \epsilon \in \mathfrak{E}\}$ è una partizione di E.*

Nella definzione, così come d'ora in avanti, si utilizzerà la notazione $\mathfrak{V}$ e $\mathfrak{E}$ per indicare, rispettivamente, gli insiemi di supernodi e superarchi di un grafo contratto, e quindi di livello superiore, rispetto al grafo di riferimento.

Nella figura 3.2, sulla sinistra è mostrato un esempio di grafo decontraibile contrazione del grafo decontraibile a destra, dove sono annotati per ogni supernodo e superarco i nodi e gli archi ottenibili dalle loro decontrazioni.

Dalla definizione si evince che le seguenti sono condizioni necessarie affinchè un grafo decontraibile G' possa essere una contrazione di un grafo decontraibile G:

(i) I nodi di G' devono avere tutti un grafo non vuoto come decontrazione, ovvero $V_\alpha \neq \emptyset$ per ogni $\alpha \in \mathfrak{V}$. Infatti se fosse che $V_\alpha = \emptyset$ per qualche $\alpha \in \mathfrak{V}$, allora l'insieme $\{V_\alpha \mid \alpha \in \mathfrak{V}\}$ non costituirebbe una partizione di V, in quanto, per definzione, l'insieme vuoto non può essere incluso in una partizione.

(ii) Gli insiemi di nodi delle decontrazioni dei supernodi in G' devono essere a due a due disgiunti, ovvero non possono esistere nodi a cui corrispondono contemporaneamente due supernodi distinti, in quanto anche in questo caso l'insieme $\{V_\alpha \mid \alpha \in \mathfrak{V}\}$ non costituirebbe una partizione di V.

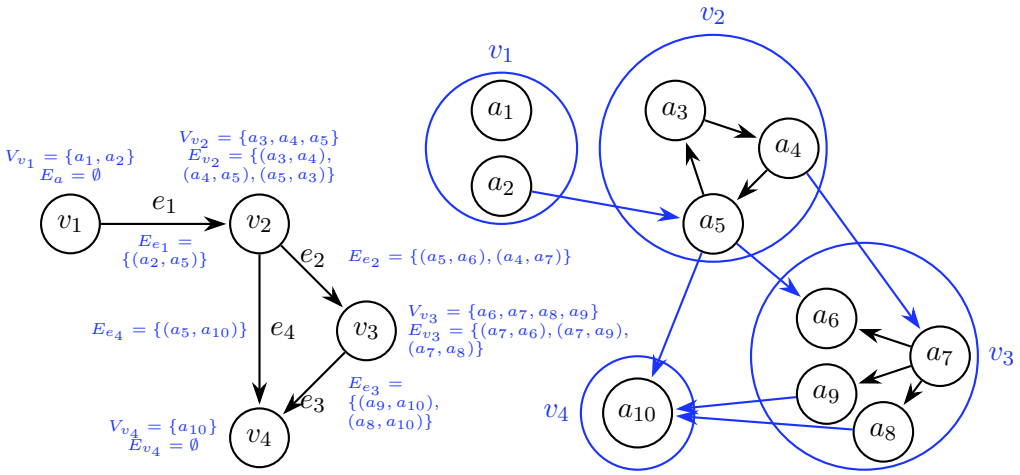

Figura 3.2: Esempio di contrazione di un grafo decontraibile

(iii) I superarchi di G' devono avere tutti una decontrazione non vuota. Infatti, se fosse che $dec_{\mathfrak{E}}(\epsilon) = \emptyset$ per qualche $\epsilon \in \mathfrak{E}$, allora l' insieme $(\{E_\alpha \mid \alpha \in \mathfrak{V}\} \setminus \{\emptyset\}) \cup \{dec_{\mathfrak{E}}(\epsilon) \mid \epsilon \in \mathfrak{E}\}$ non costituirebbe una partizione di E, analogamente a quanto detto nel primo punto.
Si noti che la condizione di disgiunzione tra gli insiemi di archi delle decontrazioni dei superarchi è automaticamente soddisfatta dalla definzione di grafo decontraibile e delle decontrazioni dei suoi archi.

E' rilevante notare che, data una contrazione G' del grafo decontraibile G, essa contiene tutte le informazioni necessarie a calcolare la struttura di archi e nodi di G. Infatti, sia $G' = (\mathfrak{V}, \mathfrak{E})$ una contrazione di G, allora dalla definizione si ha:

$$G = (\bigcup_{\alpha \in \mathfrak{V}} V_\alpha, (\bigcup_{\alpha \in \mathfrak{V}} E_\alpha \cup \bigcup_{\epsilon \in \mathfrak{E}} dec_{\mathfrak{E}}(\epsilon)), dev_V, dec_E)$$

dove dec_V è ottenuto dall'unione delle funzioni $dec_{\mathfrak{V}_v}$ per ciascun $v \in \mathfrak{V}$, e dec_E è ottenuto dall'unione delle funzioni $dec_{\mathfrak{E}_e}$ per ciascun $e \in \mathfrak{E}$ combinato con tutte le mappature tra superarchi in $\bigcup \epsilon \in \mathfrak{E} dec_{\mathfrak{E}}(\epsilon)$ e le loro decontrazioni. Definiamo, quindi, l'operatore unario $.^D : \mathcal{G}_D \to \mathcal{G}_D$ come l'operatore di **decontrazione completa** che, dato un grafo decontraibile $G = (V, E)$ restituisce il grafo decontrabile G^D ottenuto dalla decontrazione di tutti i supernodi e superarchi del grafo in input.

$$G^D = (\bigcup_{v \in V} \mathcal{V}_v, (\bigcup_{v \in V} \mathcal{E}_v \cup \bigcup_{e \in E} dec_E(e)), dec_{\mathcal{V}}, dec_{\mathcal{E}})$$

Una contrazione di un grafo decontraibile G, può quindi essere alternativamente definta come un suo grafo quoziente decontrabile G' la cui decontrazione completa $(G')^D$ è proprio G.

Si noti che, in generale, un grafo decontraibile G può non essere una contrazione di G^D. Questo può verificarsi unicamente quando G non soddisfa tutte le condizioni (i), (ii) e (iii) presentate in precendenza.

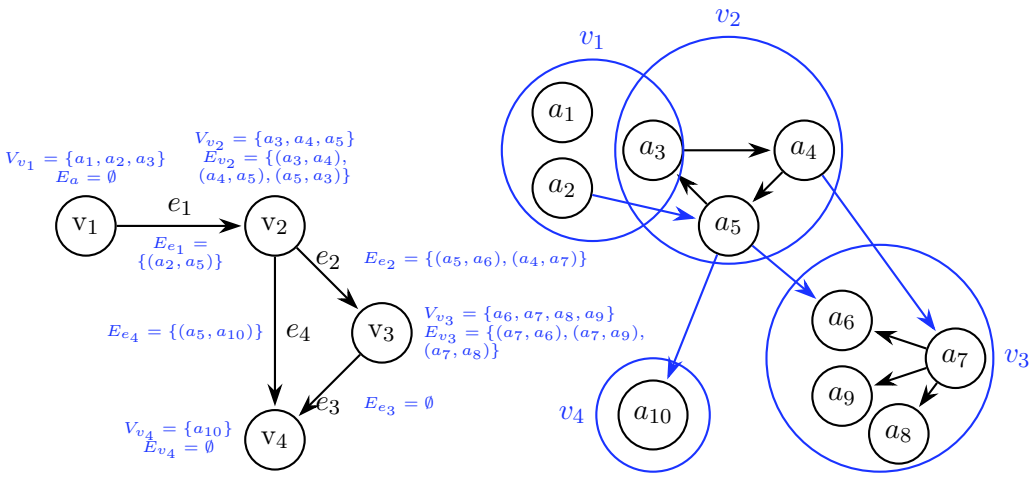

Figura 3.3: Esempio di grafo decontraibile che non è contrazione della sua decontrazione completa

Per rendere chiaro questo aspetto, in figura 3.3 è proposta una variazione dell'esempio precedente, in cui il grafo decontraibile a sinistra, ottenuto come decontrazione completa del grafo destra, non è una sua contrazione, in quanto il super-nodo a_3 appartiene contemporaneamente a V_a e V_b, e il super-arco e_3 viene decontratto in un insieme vuoto di archi, violando le condizioni (ii) e (iii).

Sarebbe, quindi, scorretto dire che un grafo G' è una contrazione di G se $(G')^D = G$. In particolare, considerato quanto già detto, si può facilmente dimostrare la seguente proposizione.

Proposizione 3.2.1. *Siano $G = (V, E)$ e $G' = (\mathfrak{V}, \mathfrak{E})$ due grafi decontraibili. Allora:*

$$\left.\begin{cases} V_\alpha \neq \emptyset & \forall \alpha \in \mathfrak{V} \\ V_\alpha \cap V_\beta = \emptyset & \forall \alpha, \beta \in \mathfrak{V},\ \alpha \neq \beta \\ E_\epsilon \neq \emptyset & \forall \epsilon \in \mathfrak{E} \end{cases}\right\} \wedge G = (G')^D \quad \Longleftrightarrow \quad G' \text{ è una contrazione di } G$$

In aggiunta alla precendente proposizione, grazie alla definizione di contrazione tra grafi decontraibili, si propone un'altra propoizione che descrive come il concetto di grafo indotto può essere usato per descrivere le decontrazioni.

Proposizione 3.2.2. *Sia $G = (V, E)$ un grafo decontraibile e sia $G' = (\mathfrak{V}, \mathfrak{E})$ una sua contrazione, sia α un super-nodo appartenente a $\mathfrak{V}$. Allora $dec_{\mathfrak{V}}(\alpha) = (V_\alpha, E_\alpha)$ è il sottografo di G indotto da V_α.*

$$dec_{\mathfrak{V}}(\alpha) = G[V_\alpha]$$

Dimostrazione Sia $H = (W, F)$ il sottografo di $G = (V, E)$ indotto da V_α con $\alpha \in \mathfrak{V}$, per definizione di grafo indotto, H deve essere definito sull'insieme di nodi V_α, ovvero deve essere $W = V_\alpha$. Si vuole ora dimostrare che $F = E_\alpha$.

L'inclusione $F \subseteq E_\alpha$ può essere dimostrata notando che per definizione H, che è un grafo indotto da V_α, si ha:

$$(x, y) \in F \implies (x, y) \in E \quad \text{con} \quad x, y \in V_\alpha$$

Essendo che $\{E_\beta \mid \beta \in \mathfrak{V}\} \cup \{dec_{\mathfrak{E}}(\epsilon) \mid \epsilon \in \mathfrak{E}\}$ è un ricoprimento di E, si nota che l'unico insieme del ricoprimento che può contenere archi definiti in $V_\alpha \times V_\alpha$ è proprio E_α. Si conclude allora $(x, y) \in F \implies (x, y) \in E \implies (x, y) \in E_\alpha$.

L'inclusione $E_\alpha \subseteq F$ può essere dimostrata notando che per la proprietà delle contrazioni, per cui $\{E_\beta \mid \beta \in \mathfrak{V}\} \cup \{dec_{\mathfrak{E}}(\epsilon) \mid \epsilon \in \mathfrak{E}\}$ è una copertura di E, si ha:

$$(u, v) \in E_\alpha \implies (u, v) \in E$$

Essendo che $(u, v) \in E_\alpha \implies (u, v) \in V_\alpha \times V_\alpha$ per definzione di E_α, si deve avere $u, v \in V_\alpha$. Segue quindi $(u, v) \in E_\alpha \wedge u, v \in V_\alpha \implies (u, v) \in F$.

3.3 Grafo multi-livello

A partire dalla definizione di grafo decontraibile e di contrazione, può essere definita una struttura gerarchica a più livelli di grafi decontraibili che siano l'uno la contrazione dell'altro, dove i grafi ai livelli inferiori o loro sottografi possano essere ottenuti attraverso, rispettivamente, decontrazioni complete o espansioni locali dei livelli superiori.

3.3.1 Funzioni

Definizione 3.3.1 (Funzione di contrazione)
Una ***funzione di contrazione*** $f_C : \mathcal{G}_D \to \mathcal{G}_D$ *è una funzione che dato un grafo decontraibile* G *essa produce un nuovo grafo decontraibile* $f_C(G) = G'$ *che sia una contrazione di* G*.*

Una funzione di contrazione rappresenta quindi un particolare schema di contrazione dove dominio e codominio sono coincidenti e sono rappresentati dall'insieme dei grafi decontraibili $\mathcal{G}_D$. Essa produce contrazioni dei grafi decontraibili in input secondo determinate logiche definite dalla funzione stessa. Nel corso di questo capitolo, i termini *funzione di contrazione* e *schema di contrazione* saranno utilizzati in modo intercambiabile.
É importante notare il fatto che la funzione sia chiusa rispetto all'insieme dei grafi decontraibili: questo vuol dire che è possibile comporre più funzioni di contrazione in sequenza a partire da un dato grafo decontraibile.

Definizione 3.3.2 (Funzione di trasformazione naturale)
Definiamo ***funzione di trasformazione naturale*** η *una funzione che dato un grafo standard* $H = (W, F)$*, produce il corrispondente grafo decontraibile* $G = (V, E, dec_V, dec_E)$ *con le seguenti proprietà:*

- $dec_V(v) = (\emptyset, \emptyset) \quad \forall v \in V$
- $dec_E(e) = \emptyset \quad \forall e \in E$
- H *e* G *sono isomorfi*

La funzione trasformazione naturale è quindi la funzione che permette di trasformare un dato grafo standard in un grafo decontraibile isomorfo per cui le due funzioni di decontrazione dei nodi e degli archi siano definite, seppur producano rispettivamente un grafo e un insieme di archi vuoto.
Si può osservare che queste proprietà garantiscono che il grafo decontraibile ottenuto non possa essere contrazione di alcun altro grafo decontraibile.

Il nome di funzione di trasformazione naturale deriva dalla teoria delle categorie, dove una trasformazione naturale è una mappa tra due funtori che preserva la struttura tra i due oggetti e la coerenza delle operazioni che possono essere applicate su di essi. Sebbene in matematica i grafi non siano considerati essi stessi funtori, nella programmazione funzionale è possibile che i grafi vengano interpretati come tali.

3.3.2 Definizione dei Grafi Multi-Livello

Il concetto di grafo multi-livello è di seguito definito attraverso una decrizione bottom-up di tale struttura, indicandone il grafo inziale e gli schemi di contrazione, che descrivono il modo in cui dei sottografi di un determinato livello sono collassati in singoli supernodi del livello superiore, formando gerarchie di grafi decontraibili.

Definizione 3.3.3 (Grafo multi-livello)
Un ***grafo multi-livello*** M *è una coppia* (G, Γ) *dove:*

- $G = (V, E)$ *è un grafo*
- Γ *è una sequenza* $\langle f_{C_1}, f_{C_2}, .., f_{C_k} \rangle$ *di funzioni di contrazione*

Utilizzando la notazione G_i per indicare il grafo decontraibile collocato al livello i-esimo della gerarchia, si può consierare il grafo multi-livello M come una sequenza di grafi decontraibili $\langle G_0, G_1, .., G_k \rangle$, dove il grafo $G_0 = \eta(G)$ può essere ottenuto dalla funzione di trasformazione naturale η applicata al grafo standard G. Per questo, le funzioni di contrazione di un grafo multi-livello dovono essere tali che $f_{C_i}(G_{i-1}) = G_i$ per ogni $i \in \{1, .., k\}$.

Dato un grafo multi-livello $M = (G, \Gamma)$ con $\Gamma = \langle f_{C_1}, f_{C_2}, .., f_{C_k} \rangle$, la funzione che calcola il suo grafo decontraibile al livello k-esimo G_k può quindi essere descritta attraverso la seguente definizione ricorsiva:

$$con(M, k) = \begin{Bmatrix} f_{C_k}(con(M, k-1)) & \text{se } k > 0 \\ \eta(G) & \text{se } k = 0 \end{Bmatrix} = G_k$$

Si può notare che la funzione da applicare a G per ottenere G_k dovrà essere la composizione ordinata delle funzioni di contrazione in Γ fino al livello k abbinata alla funzione di contrazione η per ottenere G_0, ovvero:

$$G_k = (f_{C_k} \circ f_{C_{k-1}} \circ \ldots \circ f_{C_1} \circ \eta)(G)$$

Sia $\mathcal{M}$ l'insieme dei grafi decontraibili, definiamo, inoltre, la funzione altezza h, sia per grafi multi-livello che per i suoi grafi decontraibili, nel modo seguente:

- $h : \mathcal{M} \to \mathbb{N}$, tale che $h(M) = k$, con $M = (G, \Gamma)$ e k il numero di funzioni di contrazione in Γ. Ad esempio $h(M) = 0$ se $M = (G, \langle\rangle)$.
- $h : \mathcal{G}_D \to \mathbb{N}$, tale che $h(G_i) = i$, con i il numero di contrazioni necessarie per ottenere G_i a partire da $\eta(G)$. Ad esempio $h(\eta(G)) = 0$.

In figura 3.4 è mostrato un esempio di grafo multi-livello di altezza 2, rappresentato mediante una sequenza di grafi decontraibili G_0, G_1, e G_2. I grafi dei livelli superiori sono ottenuti rispettivamente attraverso funzioni di contrazione per cricche (contrazione da G_0 a G_1) e per componenti connesse (contrazione da G_1 a G_2).

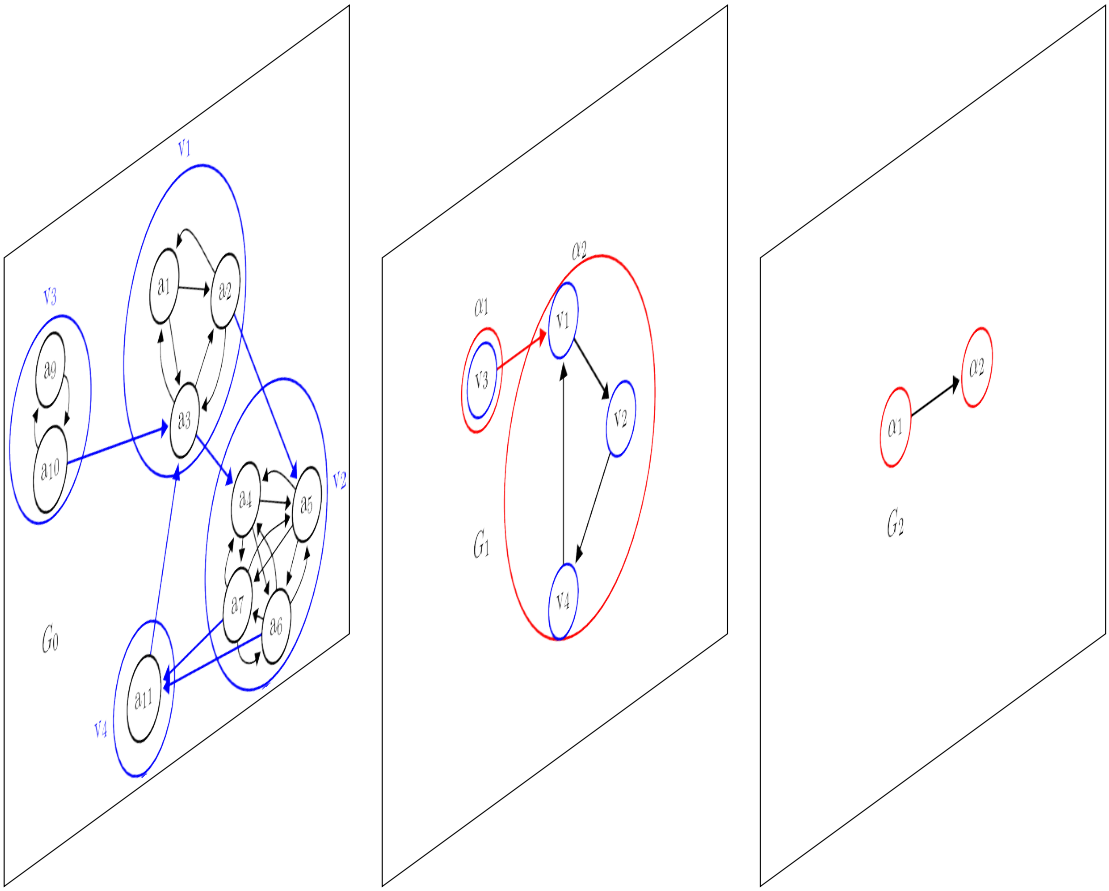

Figura 3.4: Esempio di grafo multilivello

3.3.3 Algoritmo di trasformazione naturale

La generica procedura algoritmica per realizzare la trasformazione naturale di un grafo standard $H = (W, F)$ preso in input in un grafo decontraibile $G = (V, E)$, può essere definita considerando la creazione di supernodi e superarchi, costruendo implicitamente la funzione biiettiva $f_V : W \to V$ che realizza l'isomorfismo tra i nodi di H e i supernodi di G.

Algorithm 1 NATURAL-TRANSFORMATION(H)

1: Sia $G = (V, E)$ un nuovo grafo decontraibile, con $V = \emptyset$ e $E = \emptyset$
2: **for** $x \in W$ **do**
3: Sia v un nuovo supernodo
4: $v.dec \coloneqq (\emptyset, \emptyset)$
5: $V \coloneqq V \cup \{v\}$
6: **end for**
7: **for** $(x, y) \in F$ **do**
8: Sia e un nuovo superarco
9: $e.dec \coloneqq \emptyset$
10: $E \coloneqq E \cup \{e\}$
11: **end for**
12: **return** G

Tramite l'ausilio di strutture dati con tempi di ricerca costanti, come insiemi di hash, la complessità delle operazioni all'interno dei due circli è $O(1)$, e, di conseguenza, la complessità dell'algoritmo è dettata dal numero di nodi e archi del grafo in input H, ovvero $\Theta(|W|+|F|)$.

4. Algoritmi di enumerazione

In informatica, un *algoritmo di enumerazione* è un algoritmo che elenca tutte le possibili risposte ad un problema computazionale in modo sistematico e completo. Tali algoritmi sono quindi progettati per ricevere un determinato input, generare una lista esaustiva di tutte le possibili soluzioni senza duplicati e, solo allora, terminare. Quando si parla di algoritmi di enumerazione applicati al dominio dei grafi, allora spesso si intende il processo di identificazione di sottoinsiemi di nodi o archi che soddisfano determinate caratteristiche. In questo capitolo si tratteranno e analizzeranno alcuni utili algoritmi di enumerazione presenti in letteratura per il riconoscimento di pattern strutturali all'interno di grafi. In particolare, si discuteranno algoritmi per l'enumerazione di componenti fortemente connesse, di cricche e di circuiti semplici, problemi di notevole importanza nella teoria dei grafi e ritenuti di interesse per la definizione di algoritmi di contrazione. Questi algoritmi costituiranno il m̈otored̈egli algoritmi presentati nei capitoli successivi, e saranno di importanza fondamentale alla definizione degli algoritmi di contrazione usati per la costruzione di grafi multi-livello.

4.1 Enumerazione di componenti fortemente connesse

Come citato nel Capitolo 2, la condensazione di un grafo diretto è il suo grafo quoziente dove le componenti fortemente connesse definiscono i blocchi della partizione. Enumerare le componenti fortemente connesse di un grafo diretto $G = (V, E)$ significa trovare tutti i sottoinsiemi che permettano di costruire una condensazione del grafo di partenza. In questa sezione si discuterà un classico algoritmo di enumerazione delle componenti fortemente connesse, chiamato da alcuni testi come l'algoritmo di Kosaraju [13] e al seguito si discuterà di un possibile adattamento dell'algoritmo per la costruzione contestuale di un grafo contratto in forma di grafo decontraibile.

4.1.1 Algoritmo di Kosaraju

L'algoritmo di Kosaraju (anche noto come algoritmo di Kosaraju-Sharir) è un algoritmo per l'enumerazione delle componenti fortemente connesse di un grafo diretto dalla complessità lineare scoperto nel 1978 da S. Rao Kosaraju, ma pubblicato solamente nel 1981 da Micha Sharir, che lo scoprì indipendentemente. Esso sfrutta il principio per cui le componenti fortemente connesse di un grafo diretto sono le stesse del suo grafo trasposto, ovvero il grafo ottenuto invertendo l'orientamento di tutti gli archi.

A seguire alcune utili nozioni preliminari per la comprensione dell'algoritmo:

Grafo trasposto Dato un grafo diretto $G = (V, E)$, il suo grafo trasposto $G^T = (V, E^T)$ è il grafo ottenuto invertendo l'orientamento di tutti gli archi di G, ovvero $E^T = \{(u, v) \mid (v, u) \in E\}$. É interessante notare che G^T ha le stesse componenti fortemente connesse di G: se un cammino da u a v esiste in G, allora esiste anche un cammino da v a u in G^T. Essendo le componenti fortemente connesse basate sulla mutua raggiungibilità dei nodi, esse non cambiano quando si invertono gli archi.

Una procedura algoritmica per il calcolo di G^T non farebbe altro che scorrere linsieme di archi di G e invertirli, e sarebbe quindi di complessità lineare.

Visita in profondità La visita in profondità di un grafo (in inglese *depth-first search* o *DFS*) è un particolare algoritmo di visita che, in quanto tale, permette di visitare tutti i nodi di un grafo partendo da un nodo iniziale, e di scoprire tutti i nodi raggiungibili da esso. Ciò viene fatto in modo ricorsivo: nel corso della visita di un nodo si considerano uno ad uno i nodi ad esso adiacenti, e ai nodi non ancora marcati come visitati viene applicata immediatamente la stessa procedura di visita.

Algorithm 2 DFS(G)

```
1: for v ∈ G.V do
2:     v.color := WHITE
3:     u.π := NIL
4: end for
5: time := 0
6: for v ∈ G.V do
7:     if v.color == WHITE then
8:         DFS − VISIT(G, v)
9:     end if
10: end for
```

Algorithm 3 DFS-VISIT(G, u)

```
1: u.color := GRAY
2: time := time + 1
3: u.d := time
4: for v ∈ G.Adj[u] do
5:     if v.color == WHITE then
6:         v.π := u
7:         DFS − VISIT(G, v)
8:     end if
9: end for
10: u.color := BLACK
11: time := time + 1
12: u.f := time
```

Nel corso dell'algoritmo i nodi vengono colorati in tre colori: bianco, grigio e nero, ad indicare rispettivamente che il nodo non è stato visitato, che è in fase di visita e che è stato visitato. Questo permette di non incorrere in cicli di visita infiniti nel caso non si stia visitando un grafo aciclico. Nel corso dell'algoritmo vengono anche assegnati ai nodi due valori interi: il tempo di scoperta d e il tempo di fine visita f, che permettono

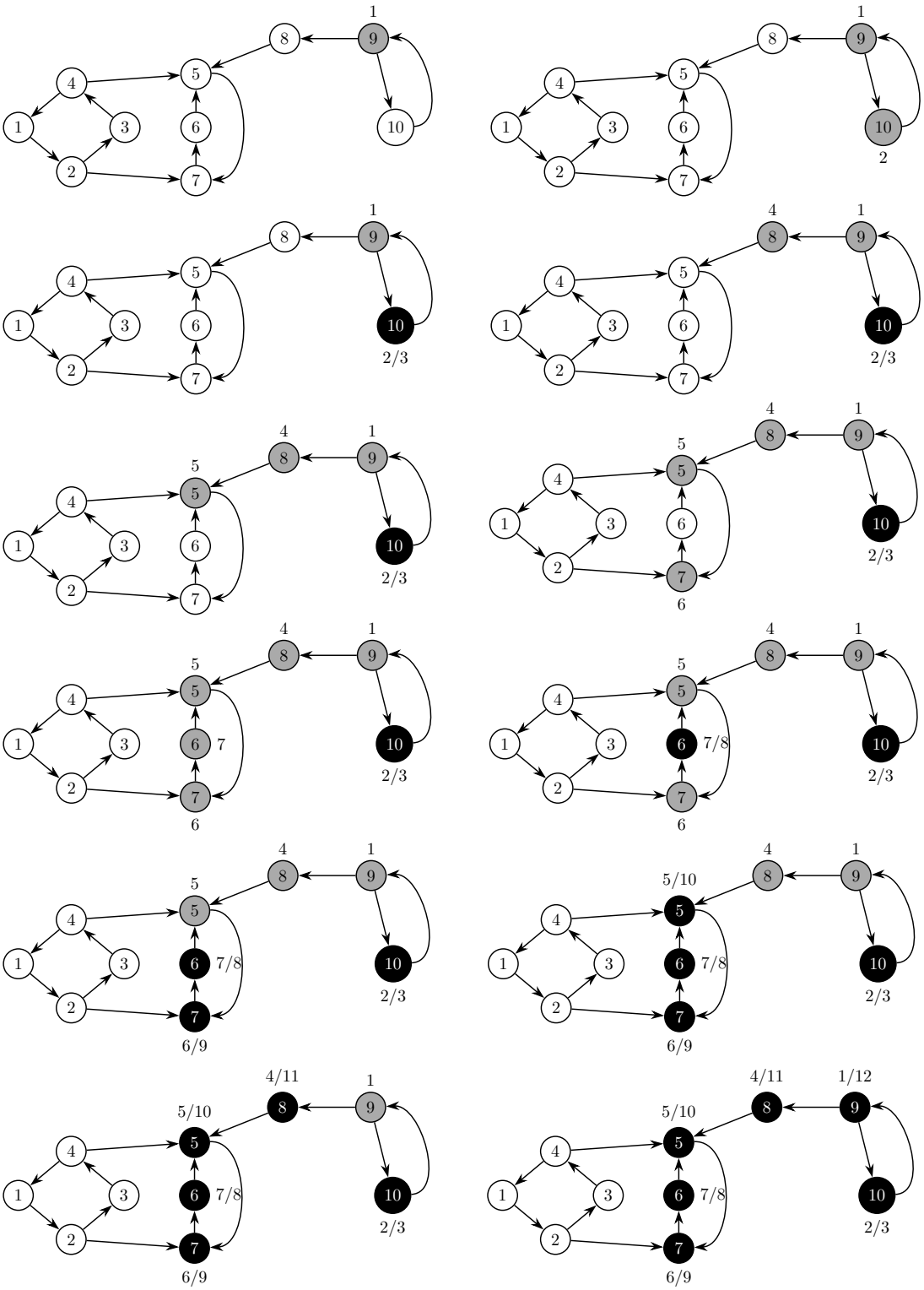

Figura 4.1: Esempio di esecuzione di una visita in profondità su un grafo diretto

di determinare, rispettivamente, il momento in cui i nodi vengono scoperti e colorati di grigio e il tempo in cui la visita di un nodo termina, colorandosi di nero. Un attributo aggiuntivo, il predecessore π, permette di memorizzare il nodo da cui si è scoperto il nodo corrente e, con esso, di ricostruire il cammino di visita sotto forma di albero, detto albero di visita in profondità.

In figura 4.1 è rappresentato un esempio di esecuzione della procedura DFS-VISIT su un grafo diretto, eseguita a partire dal nodo con etichetta 9, in cui lo stato del grafo è rappresentato ad ogni incremento della variabile *time*.

La procedura si mantiene in tempo lineare rispetto al numero di nodi e di archi del grafo, in quanto ogni nodo viene visitato una sola volta e ogni arco viene esaminato al massimo una volta.

Ordinamento topologico Dato un grafo diretto aciclico $G = (V, E)$, un ordinamento topologico di G è una particolare sequenza dei suoi nodi $\langle v_1, v_2, \ldots, v_n \rangle$ tale che per ogni arco $(v_i, v_j) \in E$, $i < j$. Si noti, quindi, che un ordinamento topologico di un grafo diretto può esistere solo se il grafo non contiene cicli. Un ordinamento topologico fornisce una disposizione tale che i nodi raggiungibili da un certo nodo v_i vengano disposti dopo di esso, e che i nodi che v_j può raggiungere vengano disposti dopo di esso.

Una procedura algoritmica per il calcolo di un ordinamento topologico di G può essere effettuata tramite una visita in profondità del grafo, raccogliendo in una lista concatenata i nodi in ordine decrescente di tempo di fine visita, man mano che vengono visitati. Per via del normale costo di una visita in profondità, la complessità di tale procedura è lineare al numero di nodi e di archi del grafo

Algorithm 4 KOSARAJU-ALGORITHM(G)

1: Esegui $DFS(G)$ per calcolare il tempo di fine visita $u.f$ per ogni nodo u in G
2: Calcola G^T
3: Esegui $DFS(G^T)$, considerando $G.V$ in ordine decrescente di tempo di fine visita
4: Fornisci in output ogni albero di visita in profondità come componente fortemente connessa di G

Come descritto dallo pseudocodice, l'algoritmo di Kosaraju esegue due visite in profondità, una sul grafo originale G e una sul grafo trasposto G^T, in cui la seconda visita viene effettuata secondo l'ordinamento ottenuto dalla prima. Le motivazioni per cui la procedura di visita a riga 3 permette di visitare una componente fortemente connessa alla volta per ogni iterazione del ciclo for principale possono essere riassunte dai seguenti punti:

- La condensazione di G è un grafo aciclico, e quindi ammette un ordinamento topologico. Questo vuol dire che se esiste un cammino $u \rightsquigarrow v$ in G, con u e v nodi di componenti fortemente connesse distinte, allora di certo non può esistere un cammino $v \rightsquigarrow u$ in G, altrimenti i nodi in entrambe le componenti connesse sarebbero tutti mutualmente raggiungibili tra loro, e costituirebbero una unica componente fortemente connessa.
- Se C e C' sono componenti fortemente connesse di G distinte ed esiste in G un cammino $u \rightsquigarrow v$ con $u \in C$ e $v \in C'$, allora il tempo di fine visita massimo

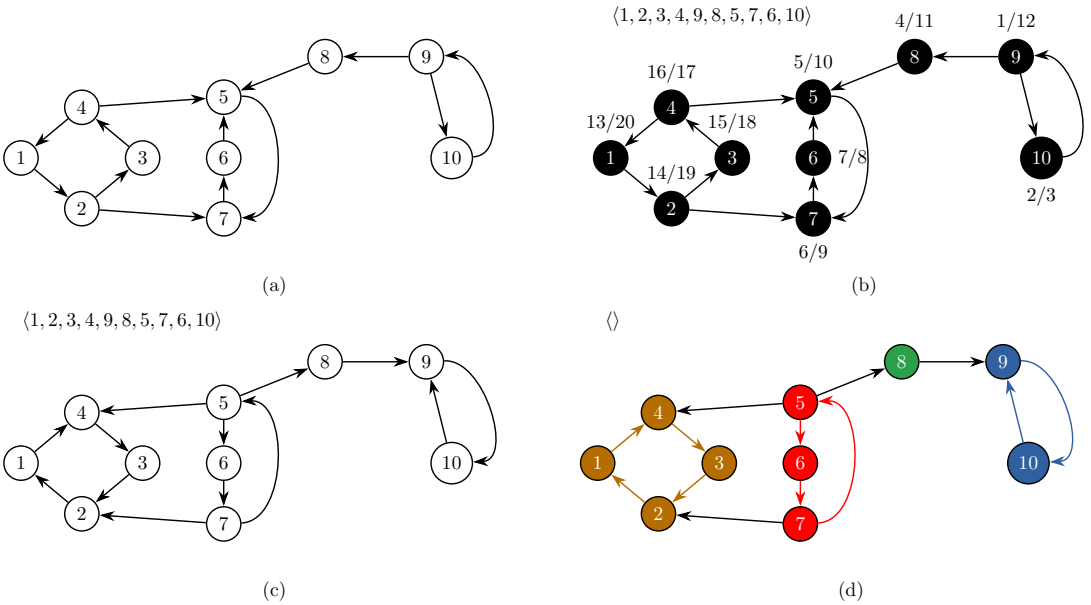

Figura 4.2: Esempio di esecuzione dell'algoritmo di Kosaraju su un grafo diretto

tra i nodi di C sarà maggiore del tempo di fine visita massimo tra i nodi di C'. Infatti, supponendo che venga visitato prima u, tutti i nodi raggiungibili da u, incluso v e tutti i nodi in C', saranno visitati prima che la visita di u termini. Al contempo, supponendo che venga visitato prima v, allora nessun nodo di C verrà visitato prima che la visita di v termini, in quanto, come detto nel primo punto, l'esistenza di un cammino $u \rightsquigarrow v$ esculde l'esistenza di un cammino $v \rightsquigarrow u$.

- Considerare i nodi in ordine decrescente di tempo di fine visita permette di visitare prima le componenti fortemente connesse meno profonde; ovvero quelle che non contengono nodi raggiungibili da altre componenti fortemente connesse. In altre parole, le componenti meno profonde sono le prime ad apparire nell'ordinamento topologico della condensazione.

- Invertendo il senso degli archi e visitando prima le componenti fortemente connesse meno profonde in G, si garantisce che non si possano raggiungere nodi di altre componenti, ma solo nodi della stessa componente. Invertendo il senso degli archi, infatti, si inverte l'ordinamento topologico della condensazione, visitando prima le compoenenti che si trovano in fondo al nuovo ordinamento.

In figura 4.2 sono rappresentate le fasi rilevanti dell'algoritmo di Kozaraju applicato al grafo in (a). In (b) è rappresentato lo stato del grafo al termine della procedura DFS, assieme alla lista dei nodi ordinata per tempi di fine visita. In (c) il grafo trasposto ottenuto invertendo gli archi del grafo in (a). In (d) lo stato del grafo trasposto in (c) al termine della seconda procedura di visita in profondità, dove colori dello stesso colore rappresentano nodi appartenenti alla stessa componente fortemente connessa fornita in output.

Complessità dell'algoritmo L'algoritmo di Kosaraju ha una complessità temporale di $\Theta(V + E)$, in quanto:

- A riga 1 viene eseguita una ricerca in profondità tradizionale, quindi con costo $\Theta(|V|+|E|)$.

- A riga 2 viene costruito il grafo trasposto G^T e come già detto, la sua costruzione può avvenire in un tempo $\Theta(|V|+|E|)$, iterando prima sull'insieme dei nodi di G e poi sui suoi archi, invertendone l'orientamento.

- L'ordinamento dei nodi di G^T può essere realizzato nel corso della visita a riga 1 attraverso la costruizione di una lista concatenata, per cui i singoli nodi possono essere aggiunti in testa alla lista nel momento in cui vengono colorati di nero, in tempo costante, per un totale di $\Theta(|V|)$, e non ci sono costi aggiuntivi per la costruzione dell'ordinamento.

- A riga 3 viene eseguita la procedura DFS che corrisponde ad una visita in profondità di G^T, quindi sempre di complessità $\Theta(|V|+|E|)$.

4.2 Enumerazione di cricche

Come già discusso nel Capitolo 1, una cricca di un grafo è un sottoinsieme di nodi che sono tutti mutuamente adiacenti, ovvero un sottoinsieme di nodi che formano un sottografo completo. Nella teoria dei grafi, il concetto di cricca è normalmente riferito al contesto di un grafo non orientato, in cui è sufficiente un singolo arco non orientato $\{u, v\}$ per rendere i due nodi u e v tra loro mutualmente adiacienti. Tuttavia il concetto di cricca può essere esteso anche al contesto di grafi orientati: in particolare é sempre possibile considerare la versione non orientata alla base di un grafo orientato secondo due possibili modalità:

- Rimuovendo l'orientamento degli archi: in questo caso, è sufficiente un singolo arco orientato (u, v) per avere un arco non orientato u, v nella versione non orientata, rendendo quindi i nodi u e v tra loro mutuamente adiacienti anche se ciò non fosse vero nella versione orientata originale.

- Considerando come archi non diretti le relazioni simmetriche di adiacienza tra i nodi del grafo orientato originale: in questo caso, la versione non orientata conterrà un arco non diretto u, v se e solo se nel grafo orientato originale esistono entrambi gli archi (u, v) e (v, u).

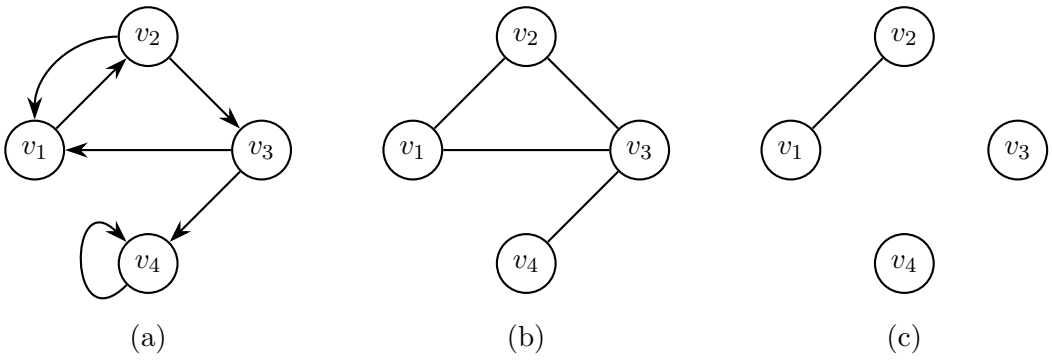

Figura 4.3: Esempio di grafo orientato e delle sue due possibili versioni non orientate

La definzione di cricca in un grafo diretto può quindi essere declinata in due modi a seconda di quanto si vuole rendere lasco il concetto di adiacienza tra i nodi. In figura 4.3 sono rappresentati in (a) un grafo diretto, in (b) la sua versione non diretta ottenuta rimuovendo l'orientamento degli archi e in (c) quella ottenuta considerando come archi non diretti le relazioni simmetriche di adiacienza tra i nodi. È utile notare che la versione non orientata (b) è mantenuta come grafo semplice, ovvero un grafo non orientato che non contiene archi multipli o cappi. Appare evidente di come la versione non orientata (c) sia più restrittiva rispetto alla versione (b): mentre in (b) le cricche di dimensione superiore a uno sono $\{v_1, v_2, v_3\}$ e $\{v_3, v_4\}$, in (c) l'unica cricca di dimensione superiore a uno è $\{v_1, v_2\}$.

D'ora in avanti, quando si farà riferimento al concetto di cricca nel contesto di grafo diretto, si farà uso dell'espressione *cricca non reciproca* per indicare le cricche presenti nella versione non diretta ottenuta rimuovendo semplicemente l'orientamento degli archi, come in (b), e *cricca reciproca* per indicare le cricche presenti nella versione non diretta ottenuta considerando le relazioni simmetriche di adiacienza tra i nodi, come in (c).

4.2.1 Algoritmo di Bron-Kerbosch

L'algoritmo di Bron-Kerbosch, descritto per la prima volta in nel 1973 [3] da Coen Bron e Joep Kerbosch, è un noto algoritmo per l'enumerazione delle cricche massimali in grafi non diretti. Con cricche massimali si intendono cricche che non possono essere estese aggiungendo ulteriori nodi senza violare la proprietà di cricca, ovvero cricche che non sono sottoinsieme proprio di altre cricche presenti nel grafo. Nella sua versione con pivoting, si tratta dell'algoritmo esatto considerato essere più efficiente nelle applicazioni pratiche rispetto ad altri algoritmi per l'enumerazione di cricche, oltre che essere stato dimostrato ottimale rispetto al numero massimo di cricche presenti in un grafo [15].

L'algoritmo si basa su un approccio ricorsivo di backtracking, in cui a partire da un insieme vuoto di nodi, si cerca di costruire una cricca massimale aggiungendo nodi uno alla volta, valutando solo i nodi che possono essere aggiunti alla cricca senza violare la proprietà di cricca. A partire dalla cricca corrente, ovvero la cricca attualmente in costruzione, ad ogni chiamata ricorsiva si valuta la scelta di aggiungere un nodo valido tra quelli disponibili, scendendo in profondità nell'albero della ricorsione. Al termine della chiamata riscorsiva, il nodo viene inserito in un insieme dei nodi già considerati, corrispondente alla scelta di escludere tale nodo dalla costruzione della cricca corrente in tutte le chiamate successive.

Algorithm 5 BRON-KERBOSH(R, P, X)

```
1: if P ∪ X = ∅ then
2:     Fornisci in output R come una cricca massimale
3: else
4:     for v ∈ P do
5:         BRON-KERBOSH(R ∪ {v}, P ∩ N(v), X ∩ N(v))
6:         P := P \ {v}
7:         X := X ∪ {v}
8:     end for
9: end if
```

Nello pseudocodice, la notazione $N(v)$ indica l'insieme dei vicini del nodo v, ovvero l'insieme dei nodi adiacenti a v tramite un arco non orientato. Nel corso della sua esecuzione, quindi, l'algoritmo manipola tre insiemi di nodi disgiunti che vengono passati alle procedure ricorsive, che sono:

- R: cricca correntemente in costruzione
- P: insieme di nodi adiacienti ad ogni nodo in R, ovvero i nodi candidati a far parte della cricca corrente non ancora considerati nella costruzione di una cricca massimale che includa R.
- X: insieme di nodi adiacienti ad ogni nodo in R, già considerati nella costruzione di una cricca massimale che includa R.

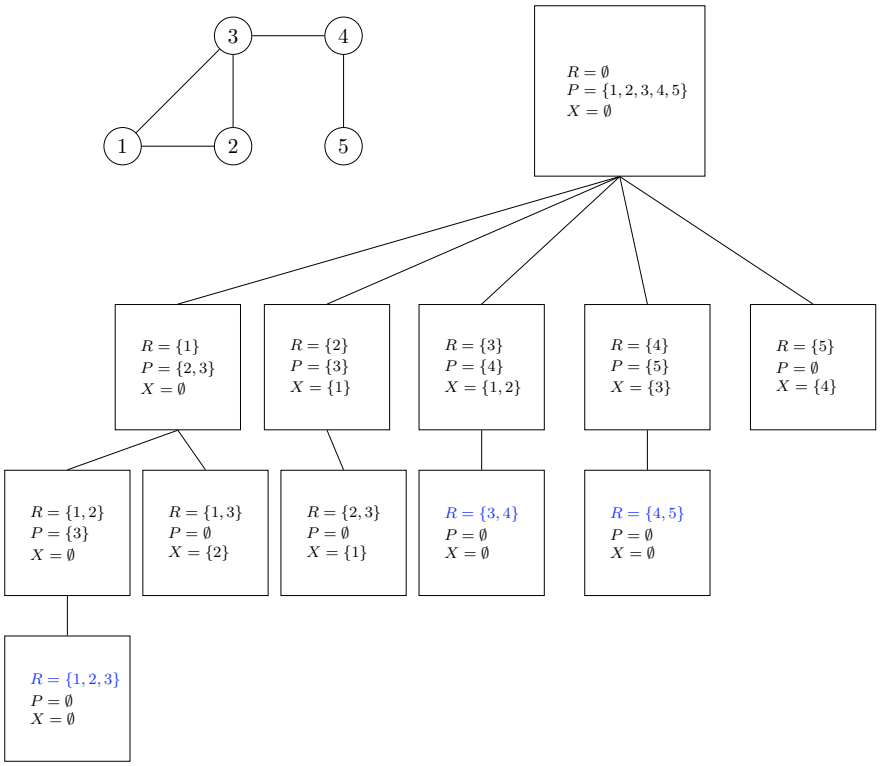

Figura 4.4: Esempio di albero di ricorsione dell'algoritmo di Bron-Kerbosch

Inizialmente gli insiemi R e X sono vuoti, mentre P contiene tutti i nodi del grafo. Le chiamate ricorsive valutano l'aggiunta di nodi da P in R, e gli insiemi P e X vengono

aggiornati mantenendo solo nodi che sono adiacenti a tutti i nodi in R, e che quindi, presi singolarmente, potrebbero essere aggiunti alla cricca corrente senza violarne la proprietà di cricca. Quando P é vuoto, le chiamate ricorsive nel ramo corrente della ricorsione terminano e, a quel punto, R rappresenterá una cricca. Tuttavia l'algoritmo riconosce se tale cricca sia massimale o meno valutando se al contempo l'insieme X sia, rispettivamente, vuoto o non vuoto. Si noti infatti che un insime X non vuoto indicherebbe che la cricca corrente potrebbe essere estesa aggiungendo nodi in X, concludendo non solo che la cricca in R non é massimale, ma anche che la cricca massimale che la contiene assieme ai nodi in X é già stata trovata.

Pivoting Bron e Kerbosch proposero nel loro articolo orginale una versione migliorata del loro algoritmo, la quale prevede l'utilizzo di una strategia di pivoting, che permette di ridurre le chiamate ricorsive della procedura trovando comunque tutte le cricche massimali.

Algorithm 6 BRON-KERBOSH-PIVOT(R, P, X, T)

```
1: if P ∪ X = ∅ then
2:     Fornisci in output R come una cricca massimale
3: else
4:     Scegli un nodo pivot u ∈ P ∪ X t. c. |P ∩ N(u)| = max_{v∈P∪X} |P ∩ N(v)|
5:     for v ∈ P \ N(u) do
6:         BRON-KERBOSH-PIVOT(R ∪ {v}, P ∩ N(v), X ∩ N(v))
7:         P := P \ {v}
8:         X := X ∪ {v}
9:     end for
10: end if
```

In particolare, ad ogni procedura ricorsiva BRON-KERBOSCH-PIVOT si sceglie un nodo pivot $u \in P$. Le chiamate ricorsive effettuate sull'insieme P saranno quindi ristrette ai nodi in P che non sono vicini di u, quindi u stesso e tutti i suoi non-vicini in P. Sebbene qualunque nodo in P possa essere scelto come pivot, come dimostrato da Tomita [15], la scelta migliore del pivot ricade sul nodo che abbia il maggior numero di vicini in P, scelta che di fatto permette di escludere il maggior numero di elementi da P e, quindi, di evitare il maggior numero di chiamate ricorsive. Per questo motivo, questa versione dell'algoritmo di Bron-Kerbosch viene spesso chiamata *Tomita*.

La tecnica del pivoting nasce dalla considerazione per cui la procedura ricorsiva applicata aggiungendo un arbitrario nodo u a R permette comunque a tutti i suoi vicini di poter essere aggiunti a R tramite le sotto-chiamate ricorsive immediatamene successive. In altre parole, i vicini di u in P faranno parte di ogni cricca massimale che contenga R che ammette anche u, in quanto ogni nodo in P preso singolarmente é adiacente ad ogni nodo in R.

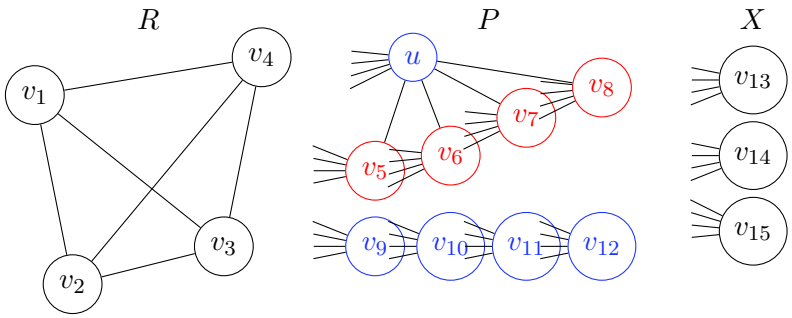

Figura 4.5: Rappresentazione dei tre insiemi R, P e X nel corso dell'algoritmo di Bron-Kerbosch con pivoting

La figura 4.5 tenta di dare un'idea migliore degli insiemi R, P ed X nel corso dell'esecuzione dell'algoritmo con strategia di pivoting. Il nodo u é scelto come pivot, e le chiamate ricorsive al livello corrente della ricorsione sono limitate ai nodi in blu. I segmenti uscenti da ogni nodo in P ed X rappresentano gli archi che li rendono adiacienti ad ogni nodo in R.

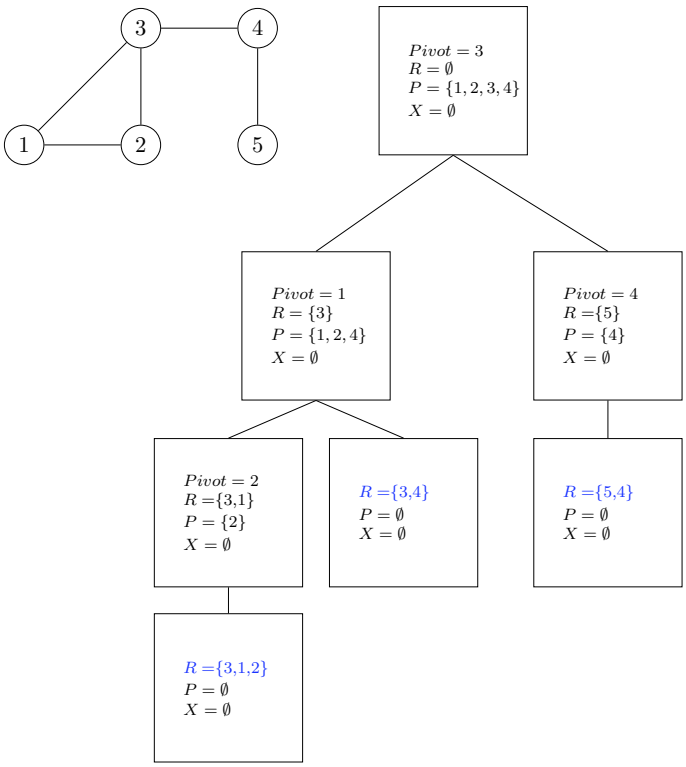

Figura 4.6: Esempio di albero di ricorsione dell'algoritmo di Bron-Kerbosch con pivoting

La figura 4.6 mostra l'albero di ricorsione derivante dall'applicazione dell'algoritmo di Bron-Kerbosch con pivoting sullo stesso grafo orientato presente in figura 4.4, rendendo evidente come la strategia di pivoting permetta di ridurre il numero di chiamate ricorsive rispetto alla versione senza pivoting.

Complessitá Come dimostrato da Tomita [15], adottando una strategia di pivoting affinché si scelga il nodo pivot w con il maggior numero di vicini in P, l'algoritmo di Bron-Kerbosch ha una complessitá temporale pari a $O(3^{n/3})$, che é una complessitá ottimale, in quanto il numero massimo di cricche massimali che si possono trovare in un grafo con n nodi é dimostrato essere $3^{n/3}$.

4.3 Enumerazione di circuiti semplici

Come già spiegato nel capitolo introduttivo, un circuito semplice di un grafo diretto é un circuito che non contiene nodi ripetuti, ad eccezione del nodo di partenza e di arrivo. Dal momento in cui, per definizione, un circuito non può contenere archi ripetuti, parlare anche solo di cicli semplici garantirebbe che esso non contenga neanche archi ripetuti. Si potrebbe notare che, mentre componenti fortemente connesse e cricche sono concetti che possono essere ricondotti ad un insieme non ordinato, i circuiti semplici comprendono in sé una sequenza di nodi in un determinato ordine. Per questo motivo diventa rilevante distinguere tra cicli e permutazioni cicliche, in quanto ogni ciclo può essere rappresentato da un insieme di permutazioni cicliche. Ad esempio, il ciclo $c_1 = \langle v_1, v_2, v_3, v_1 \rangle$ può essere rappresentato anche attraverso le permutazioni cicliche $\langle v_2, v_3, v_1, v_2 \rangle$ e $\langle v_3, v_1, v_2, v_3 \rangle$, in quanto entrambe rappresentano lo stesso anello di nodi percorsi nello stesso senso.
È interessante notare che il ciclo $c_2 = \langle v_1, v_3, v_2, v_1 \rangle$, nonostante sia un ciclo semplice composto degli stessi nodi di c_1, non può essere rappresentato da nessuna permutazione ciclica di c_1. Per questo motivo due cicli si dicono distinti se non sono l'uno una permutazione ciclica dell'altro.

In questa sezione tratteremo, quindi, un algoritmo per l'enumerazione di circuiti semplici in un grafo diretto, che risolve il problema di individuare tutti i cicli semplici distinti presenti in un grafo diretto.

4.3.1 Algoritmo di ricerca dei circuiti semplici di Johnson

L'algoritmo per la ricerca dei circuiti semplici di Donald B. Johnson, proposto nel 1975 [8], provvede ad enumerare tutti i cicli in un grafo orientato eseguendo delle visite in profondità.

A partire da un ordinamento arbitrario dei nodi, l'algoritmo esegue la procedura di visita ricorsiva CIRCUIT partendo da ciascuno di essi, individuando di volta in volta tutti e soli i circuiti semplici contenenti il nodo di partenza s. Nel corso di ogni visita viene mantenuto uno stack S, su cui si effettua una operazione di $PUSH(S, v)$ non appena v viene visitato invocando la procedura CIRCUIT su v, e una operazione di $POP(S)$ non appena la visita sul nodo termina. Lo stack, quindi, rappresenta in ogni momento il prefisso di un potenziale circuito semplice che inizia e finisce in s. Quando la visita a partire da un certo nodo s si conclude, si considera il nuovo grafo ottenuto come grafo indotto dal grafo precedente sui nodi rimanenti, ovvero il grafo della precedente iterazione privato del nodo s e di tutti gli archi incidenti su di esso. Questo permette di evitare l'individuazione di tutti i circuiti equivalenti che sono permutazioni cicliche degli stessi nodi.

Tuttavia, quelle elencate non sono le sole caratteristiche che hanno determinato il successo dell'algoritmo. Sebbene condivida con loro aspetti comuni, l'algoritmo risulta essere più efficente di altri algoritmi proposti per lo stesso problema, come quello di Tiernan [14] e Weinblatt [16], in quanto utilizza un criterio di blocco dei nodi che permette di evitare visite successive in aree del grafo che non conducono al nodo origine s. Ogni nodo è bloccato nel corso di una visita, e viene sbloccato all'uscita solamente se esso conduce ad almeno un cammino verso il nodo origine s (che non si interseca con lo stack dei nodi correntemente visitati, mantenendo la semplicità del ciclo). Ogni nodo v custodisce una lista $v.B$ di nodi che, informalmente, può considerarsi come l'insieme di nodi che sono stati bloccati a causa del blocco del nodo v nel corso di una visita. Per cui, quando un nodo viene sbloccato, la procedura UNBLOCK(v) provvede a sbloccare non solo il nodo stesso v, ma ricorsivamente anche tutti i nodi eventualmente presenti in $v.B$.

Algorithm 7 JHONSON-ALGORITHM(G)

1: Sia S una pila di nodi vuota
2: $s \coloneqq v_1$
3: **while** $i < n$ **do**
4: $K \coloneqq$ componente fortemente connessa contenente il vertice v_i nel sottografo indotto $G[\{v_i, v_{i+1}, \ldots, v_n\}]$
5: **if** $K.E \neq \emptyset$ **then**
6: $s \coloneqq v_i$
7: **for** $u \in K.V$ **do**
8: $u.blocked \coloneqq false$
9: $u.B \coloneqq \emptyset$
10: **end for**
11: CIRCUIT(s, s, K, S)
12: $i \coloneqq i + 1$
13: **else**
14: $i \coloneqq n$
15: **end if**
16: **end while**

I parametri passati alla procedura ricorsiva CIRCUIT includono

- il nodo da visitare v
- il nodo origine del circuito s
- il grafo indotto attuale K su cui avviene la ricerca
- lo stack dei nodi attualmente in corso di visita S

Algorithm 8 CIRCUIT(v, s, K, S)

```
1:  f := false
2:  PUSH(S, v)
3:  v.blocked := true
4:  for w ∈ K.adj[s] do
5:      if w == s then
6:          Fornisci in output S seguito da s come un circuito semplice
7:          f := true
8:      else if ¬w.blocked then
9:          if CIRCUIT(w) then
10:             f := true
11:         end if
12:     end if
13: end for
14: if f then UNBLOCK(v)
15: else
16:     for w ∈ K.adj[v] do
17:         w.B := w.B ∪ {v}
18:     end for
19: end if
20: POP(S)                                   ▷ removes v from stack
21: return f
```

Algorithm 9 UNBLOCK(v)

```
1: v.blocked := false
2: for w ∈ v.B do
3:     v.B := v.B \ {w}
4:     if w.blocked then UNBLOCK(w)
5:     end if
6: end for
```

In Figura è fornita una rappresentazione delle fasi rilevanti dell'applicazione della procedura CIRCUIT al nodo 1. I nodi in grigio rappresentano nodi bloccati. La figura (a) mostra l'inzio della procedura, in cui il nodo 1 viene aggiunto allo tack S. La figura (b) rappresenta il momento dell'individuazione del circuito $\langle 1, 3, 4, 1\rangle$, derivante dalla valutazione dell'arco $(4, 1)$. La figura (c) rappresenta gli effetti della procedura CIRCUIT(5), derivante dalla valutazione dell'arco $(4, 5)$. La figura (d) rappresenta gli effetti della procedura CIRCUIT(7), derivante dalla valutazione dell'arco $(4, 7)$. In entrambe le figure (c) e (d) i nodi visitati vengono aggiunti e rimossi dallo stack, ma

rimangono bloccati, in quanto non conducono ad un circuito elementare, e per questo vengono aggiornate le liste B dei relativi nodi adiacenti. Le figure (e) e (f) raffigurano le procedure ricorsive di UNBLOCK derivanti dallo sblocco del nodo 4.

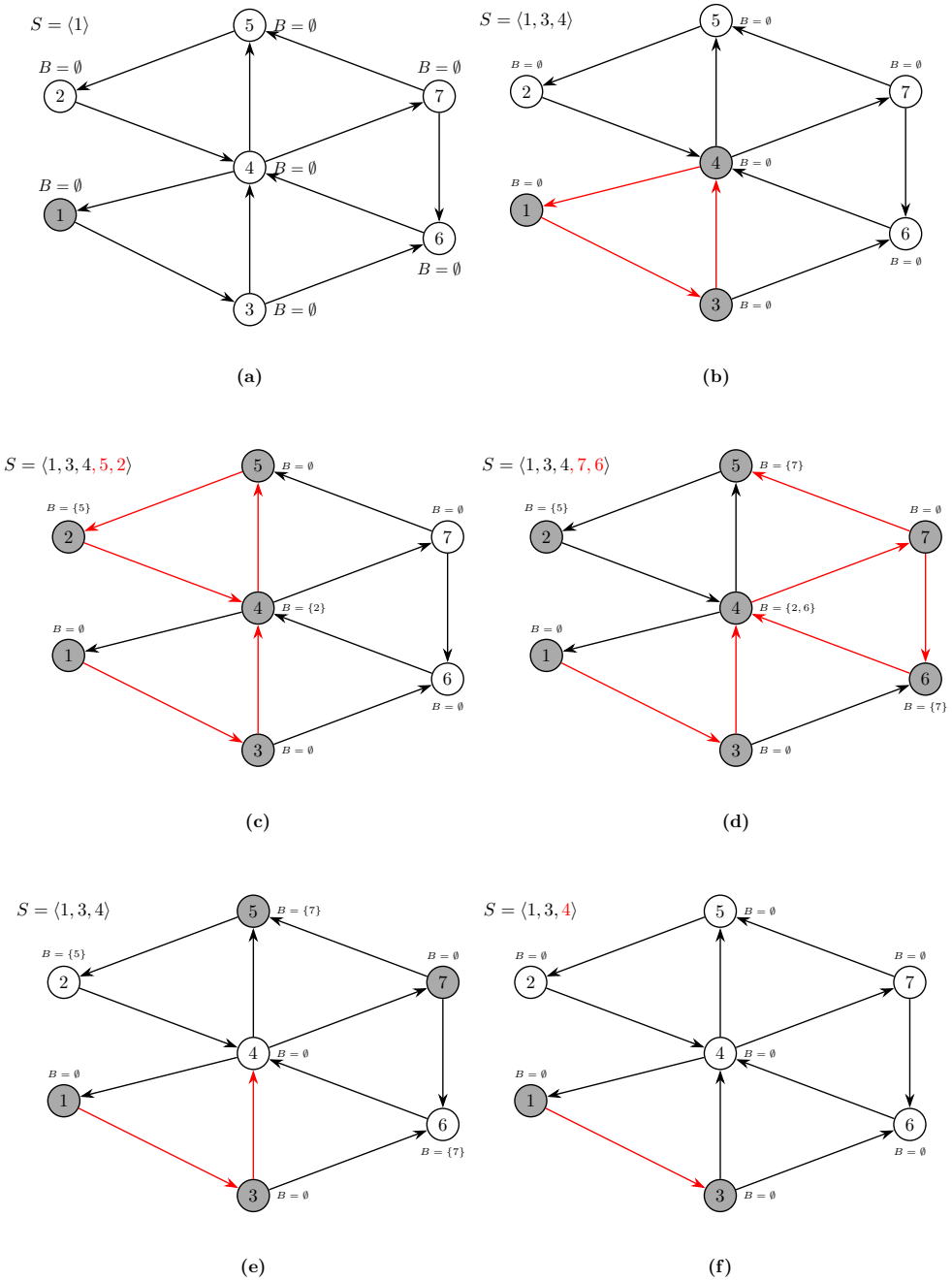

Figura 4.7: Fasi rilevanti dell'applicazione della procedura CIRCUIT

Complessità Come discusso nella trattazione originale [8], la complessità dell'algoritmo di Johnson è $O((n+m)(c+1))$, dove n è il numero di nodi, m il numero di archi e c il numero di circuiti semplici presenti nel grafo. Questo risultato è ottenuto considerando che tutti i nodi e gli archi del grafo possono essere visitati al più una volta per ogni circuito elementare presente, ovvero che l'algoritmo impiega al più $O(n+m)$ tempo tra l'output di un circuito e l'individuazione del successivo. In particolare, ciascun nodo può essere bloccato al più una volta per ogni circuito, e non può passare più di di $O(n+m)$ tempo tra il blocco di un nodo e il suo sblocco.

Si noti tuttavia che, pur essendo l'algoritmo di Johnson più efficiente rispetto ad altri algoritmi proposti, la complessità computazionale nel caso peggiore (ovvero quello di un grafo completo) risulta essere maggiore dell' esponenziale, dovuto al numero di circuiti semplici presenti, che in tal caso è pari a

$$c = \sum_{i=1}^{n-1} \binom{n}{n-i+1} (n-1)!$$

4.3.2 Algoritmo di ricerca dei circuiti semplici massimali

L'algoritmo di Johnson può essere modificato per individuare i circuiti semplici massimali, ovvero i circuiti semplici i cui nodi non sono inclusi nei nodi di altri circuiti semplici più grandi.
Una tale modifica può essere ottenuta inserendo la procedura di controllo ADD-MAXIMAL-SUBSET all'interno della procedura CIRCUIT a riga 6, al posto della normale aggiunta del sottoinsieme S con ADD-SUBSET, in modo da verificare che il circuito individuato sia massimale o meno rispetto ai nodi visitati.

Tale procedura sarà decritta nel dettaglio nel paragrafo successivo.

5. Procedure di Contrazione

In questo capitolo si entrerà nel merito delle dinamiche interne alla costruzione della struttura di un grafo multi-livello e alle sue funzioni di contrazione. Sebbene nel Capitolo 4 si siano già stati illustrati gli algoritmi specifici per il riconoscimento e l'elencazione di pattern strutturali all'interno di un grafo, nulla è stato ancora detto di come queste procedure di enumerazione debbano essere sfruttate nel contesto più complesso del grafo multi-livello. In particolare, si vuole dare una visione più chiara di come sia algoritmicamente possibile realizzare una rete di collegamenti tra supernodi su più livelli attenendosi alle definizioni fornite nel Capitolo 3, affinché si possa ottenere un tale risultato a partire da una qualsiasi sequenza di insiemi di nodi generati dalle procedure di enumerazione, evidenziandone problematiche e possibili soluzioni.

5.1 Riduzione disgiunta

Si vuole ora considerare il generico problema per cui, dato un grafo decontraibile $G = (V, E)$ e un insieme di sottoinsiemi di nodi $Q \subseteq \mathcal{P}(V)$ che costituisce una copertura di V, si vuole costruire un grafo decontraibile G' che rappresenti l'informazione fornita dal raggruppamento dei nodi descritto in Q. Aspetto di fondamentale importanza è che l'insieme Q non costituisce necessariamente una partizione. Q potrebbe quindi essere calcolato a partire da un grafo G attraverso un algoritmo di enumerazione, in modo tale che i suoi elementi siano insiemi di nodi appartenenti ad un certo pattern strutturale. Tuttavia, come si può immaginare, tali insiemi non sono necessariamente disgiunti: basti pensare alle cricche e i circuiti semplici, che possono avere nodi in intersezioni di più insiemi. Le componenti fortemente connesse invece, essendo costruite a partire da una relazione di equivalenza, sono un valido esempio di pattern strutturale che costituisce sempre una partizione di V.

D'ora in avanti, ci riferiremo agli elementi di una tale copertura di nodi Q (non necessariamente disgiunta) come **insiemi componenti**

Per mantenere le proprietà delle contrazioni su grafi è però necessario che ad ogni nodo corrisponda ad uno ed un solo supernodo. La presenza dello stesso nodo in più supernodi contratti non sarebbe coerente con la teoria dei grafi già affermata, oltre che portare ad un risultato che rischia di avere limitate applicabilità. Si vuole quindi definire una procedura per rappresentare l'informazione fornita da Q in un nuovo insieme che sia una partizione di V rappresentativa di Q e, a partire da questa, costruire un grafo decontraibile G'. Abbinare questa definzione generica, valida nella teoria degli insiemi, ad uno specifico algoritmo per calcolare Q a partire dalla struttura di un grafo, è il primo passo che permetterà di definire particolari funzioni di contrazione.

5.1.1 Definzione

Definizione 5.1.1 (Riduzione disgiunta)
Sia V un insieme di elementi. Sia $Q \subseteq \mathcal{P}(V)$ una copertura di V. Definiamo la ***riduzione disgiunta*** *di Q, e la indichiamo con $\mathcal{D}(Q)$, l'insieme di insiemi di elementi in V tale per cui:*

(i) $\emptyset \notin \mathcal{D}(Q)$

(ii) $\forall A \in \mathcal{D}(Q), \quad u, v \in A \Leftrightarrow \{C \in Q \mid u \in C\} = \{C \in Q \mid v \in C\}$

Una riduzione disgiunta di una copertura Q è quindi un insieme di insiemi di nodi in V tale per cui due nodi u e v appartengono allo stesso insieme in $\mathcal{D}(Q)$ se e solo se sono inclusi nella stessa combinazione di insiemi in Q.

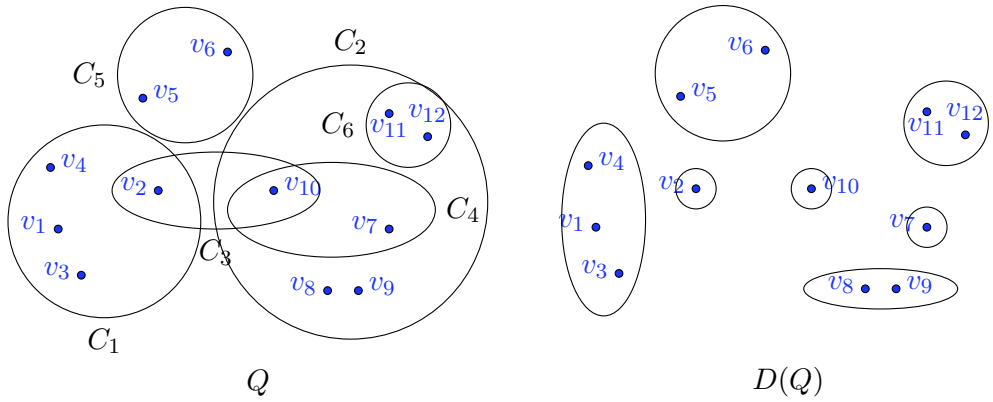

Figura 5.1: Esempio di riduzione disgiunta

L'esempio in Figura 5.1 fornisce una rappresentazione di una copertura Q di elementi (a sinistra) accompagnata dalla sua riduzione disgiunta $D(Q)$ (a destra).

Il nome di questa definizione deriva dal fatto che:

- il risultato di questo operatore su una copertura Q, da solo non contiene le informazioni sufficienti a descrivere la copertura che lo ha originato, che potrebbero essere molteplici (ma non infinite, se si ha a che fare con insiemi finiti). Da qui il termine "riduzione".
- il risultato di questo operatore su una copertura Q di V costituisce una partizione di V, come dimostrato in seguito, e fornisce un criterio per una suddivisione degli elementi in V in insiemi disgiunti tra loro. Da qui il termine "disgiunta".

Proposizione 5.1.1. *Sia V un insieme di elementi. Sia $Q \subseteq \mathcal{P}(V)$ una copertura di V. La riduzione disgiunta di Q costituisce una partizione di V.*

Dimostrazione Si definisce la relazione R_Q sull'insieme V come

$$uR_Qv \Leftrightarrow \{C \in Q \mid u \in C\} = \{C \in Q \mid v \in C\}$$

Essa è una relazione di equivalenza, in quanto rispetta le proprietà di riflessività, simmetria e transitività, che sono date dalle rispettive proprietà dell'uguaglianza. Segue che l'insieme quoziente V/R_Q deve rappresentare una partizione degli elementi in V.

Per il punto (ii) della definzione di $\mathcal{D}(Q)$, i suoi insiemi non vuoti rappresentano le classi di equivalenza di R_Q. Inoltre per il punto (i), l'insieme vuoto non può appartenere a $D(Q)$. Si conclude allora $D(Q) = V/R_Q$, e pertanto $D(Q)$ è una partizione di V.

Definizione tramite ipergrafi
Un ipergrafo è una struttura simile ad un grafo non orientato dove ogni arco, detto *iperarco*, anziché collegare una semplice coppia di nodi può connettere un arbitrario sottoinsieme di vertici. Più formalmente, un ipergrafo H è una coppia (X, E) dove X è un insieme di elementi chiamati nodi ed E è un insieme di sottoinsiemi di X chiamati iperarchi. Si consideri ora un ipergrafo non orientato $H = (X, E)$. Notando che un iperarco non orientato $e \in E$ è a tutti gli effetti un insieme di nodi $e \subseteq V$, si può estendere il concetto di rappresentazione disgiunta anche nel dominio degli ipergrafi.

Definizione 5.1.2 (Riduzione disgiunta di un ipergrafo)
*Dato un ipergrafo non orientato $H = (X, E)$ tale per cui E è un ricoprimento di X, definiamo **riduzione disgiunta** di H un nuovo ipergrafo non orientato $J = (W, F)$ tale per cui $W = V$ e $F = \mathcal{D}(E)$.*

Si può notare, quindi, che la funzione di riduzione disgiunta può essere considerata come una endofunzione idempotente sull'insieme degli ipergrafi non orientati.

Contrazione costruita da una partizione
Motivo per cui la riduzione disgiunta di una copertura è certamente utile nel momento in cui si voglia realizzare una funzione di contrazione, è che essa permette di ottenere una partizione di nodi. La seguente ulteriore definizione permette di chiarire il passaggio che porta alla produzione in output di un grafo decontraibile.

Definizione 5.1.3 (Contrazione costruita da una partizione)
*Sia $G = (V, E)$ un grafo decontraibile, sia P una partizione di V. Si definisce **contrazione di G costruita sulla partizione** P il grafo decontraibile $G' = (\mathfrak{V}, \mathfrak{E})$ contrazione di G tale per cui*

$$\{V_\alpha \mid \alpha \in \mathfrak{V}, dec_{\mathfrak{V}}(\alpha) = (V_\alpha, E_\alpha)\} = P$$

Dato un grafo decontraibile G e una sua partizione dei nodi, quindi, è sempre possibile calcolare la sua contrazione G' contraendo nodi di G in supernodi di G' secondo gli insiemi descritti dalla partizione. Si noti, infatti, che esiste una biiezione tra l'insieme delle possibili contrazioni di G e l'insieme delle partizioni di V e che, pertanto, data una partizione di V esiste una ed una sola contrazione di G costruita su di essa.

5.1.2 Algoritmo per la contrazione costruita da una riduzione disgiunta

Entrambi i concetti esposti nel paragrafo precedente, ovvero la riduzione disgiunta di una copertura e la contrazione costruita da una partizione, sono considerabili come elementi sufficientemente generici per la definizione di una qualsiasi procedura di contrazione. In particolare, il passaggio da una copertura di nodi ad un grafo decontraibile

può essere considerato come l'operazione successiva all'enumerazione degli insiemi di nodi che costituiscono la copertura stessa.

Risulta quindi utile definire un algoritmo che a partire da un grafo decontraibile G e una sua copertura Q, fornisca in output la contrazione di G costruita sulla riduzione disgiunta di Q.

Una delle possibili rappresentazioni della copertura Q, da fornire in input a tale algoritmo, è quella un *dizionario* (talvolta anche detto "mappa" o "tabella"), una struttura dati astratta che rappresenta una collezione di coppie di elementi chiave-valore e le cui operazioni fondamentali consistono nell'inserimento e cancellazione di coppie e ricerca di un valore mediante la sua chiave. È importante che in un dizionario ad ogni chiave corrisponda al più un solo valore. Negli pseudocodici, dato un dizionario T, l'operazione di ricerca di un valore associato ad una chiave k verrà indicata con $T[k]$, mentre l'operazione di inserimento di una coppia chiave-valore (k, v) con $T[k] = v$. Con $T.keys$ e $T.values$ si indicano rispettivamente l'insieme delle chiavi e l'insieme dei valori contenuti in T.

La copertura Q è quindi data da un dizionario T di dimensione $|V|$, in cui le coppie chiave-valore consistono di:

- Chiave: nodo $v \in V$
- Valore: l'insieme di insiemi componenti $C \in Q$ tali per cui $v \in C$

Nel corso dell'algoritmo un altro dizionario T' viene usato allo scopo di mappare la biiezione tra gli elementi di $\mathcal{D}(Q)$ (quindi le classi di equivalenza di V/R_Q) e i supernodi di G'. Aggiungendo coppie a T', naturalmente al termine dell'algoritmo il dizionario T' dovrà raggiungere la dimensione $|\mathcal{D}(Q)|$.

Nel ciclo a riga 3, si assegna ogni nodo del grafo decontaibile in input G ad un supernodo, che esso sia creato contestualmente o che sia il supernodo già creato corrispondente all'insieme di insiemi componenti in v. Essendo che le precondizioni impongono che T rappresenti una copertura di V, l'insieme $G.V$ a riga 3 potrebbe essere sostituito con $T.keys$. Nel ciclo a riga 15, si assegnano gli archi di G ai superarchi di G'. Se l'arco (v, w) è interno ad un supernodo, esso viene assegnato al suo insieme di archi. Se invece l'arco è tra collega due supernodi distinti, esso viene assegnato al superarco corrispondente, che potrebbe essere contestualmente creato.

Algorithm 10 MAKE-DECONTRACTIBLE-GRAPH(T, G)

Sia $G' = (\mathfrak{V}, \mathfrak{E})$ un nuovo grafo decontraibile, con $\mathfrak{V} = \emptyset$ e $\mathfrak{E} = \emptyset$
Sia T' una nuova tabella, con insiemi di nodi come chiavi e supernodi come valori
for $v \in G.V$ **do**
    if $T[v] \notin T'.keys$ **then**
        Sia β un nuovo supernodo con $G_\beta = (\emptyset, \emptyset)$
        $\mathfrak{V} \coloneqq \mathfrak{V} \cup \{\beta\}$
        $T'[T[v]] \coloneqq \beta$
        $\alpha \coloneqq \beta$
    else
        $\alpha \coloneqq T'[T[v]]$
    end if
    $V_\alpha \coloneqq V_\alpha \cup \{v\}$
    $v.supernode \coloneqq \alpha$
end for
for (v, w) in $G.E$ **do**
    $\alpha \coloneqq v.supernode$
    $\beta \coloneqq w.supernode$
    if $(\alpha == \beta)$ **then**
        $E_\alpha \coloneqq E_\alpha \cup \{(v, w)\}$
    else
        if $(\alpha, \beta) \notin \mathfrak{E}$ **then**
            $\mathfrak{E} \coloneqq \mathfrak{E} \cup (\alpha, \beta)$
        end if
        $(\alpha, \beta).dec \coloneqq (\alpha, \beta).dec \cup (v, w)$
    end if
end for
return G'

In Figura 5.2 sono rappresentati il grafo G, suddiviso in due insiemi componenente e il corrispondente dizionario T che ne rappresenta la copertura. Il dizionario T' è rappresentato nello stato seguente all'applicazione dell'algoritmo MAKE-DECONTRACTIBLE-GRAPH con input T e G, contenendo come insieme di valori i supernodi ottenuti dalla contrazione di G. Il risultato di MAKE-DECONTRACTIBLE-GRAPH(T, G) è in questo caso il grafo decontraibile $G' = (\{\alpha_1, \alpha_2, \alpha_3\}, \{(\alpha_1, \alpha_2), (\alpha_2, \alpha_3)\})$

T

v_1	$\{C_1\}$
v_2	$\{C_1\}$
v_3	$\{C_1, C_2\}$
v_4	$\{C_2\}$
v_5	$\{C_2\}$

T'

$\{C_1\}$	α_1
$\{C_1, C_2\}$	α_2
$\{C_2\}$	α_3

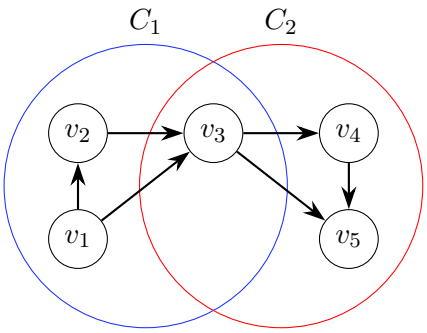

Figura 5.2: Esempio di

Complessità Facendo uso di tabelle di hash per rappresentare i dizionari T e T', l'algoritmo mantiene una complessità temporale di $O(|V|+|E|)$. Si noti infatti che:

- Il primo ciclo for scorre tutti i nodi in V. Esso inizializza il dizionario T', definisce i supernodi di G' e assegna loro i nodi in V. La complessità del ciclo è $O(|V|)$, infatti il controllo a riga 5 ha complessità costante, così come la ricerca in T' a riga 11.
- Il secondo ciclo for scorre tutti gli archi in E. Esso assegna gli archi di G ai grafi dei supernodi di G' o ai superarchi di G' a seconda dei supernodi di appartenenza dei nodi che li compongono, precedentemente calcolati nel primo ciclo for. La complessità del ciclo è $O(|E|)$.

6. Applicazioni dei Grafi Multilivello

6.0.1 Possibili applicazioni della contrazione

Nel contesto generale dell'analisi di grafi, la contrazione di nodi e archi può essere utile per:

- Ridurre la complessità dell'analisi strutturale di un grafo, sia che esso debba essere processato attraverso algoritmi costosi, sia che esso debba essere graficamente visualizzato, rendendolo più facilmente interpretabile ed evidenziando le caratteristiche strutturali di interesse.
- Studiare l'interrelazione di caratteristiche strutturali di un grafo, che rappresenti la navigabilità di uno spazio basato su componenti fortemente connesse, cicli, cricche ecc.
- Stabilire il grado di connettività di un grafo, individuando la rilevanza (intesa come il numero di insiemi componente che rappresentano i supernodi), il numero e la dimensione delle sue contrazioni.
- Misurare il grado di complessità dello spazio rappresentativo del grafo, in base al numero di nodi e archi presenti nelle contrazioni (grafi derivanti da specifici domini tendono ad avere un certo grado di complessità legato ad un concetto spaziale).
- Valutare l'influenza di singoli nodi e archi sulla struttura contratta del grafo, eseguendo analisi della sensitività e della robustezza del grafo.

Nel contesto dell'analisi di testi in linguaggio naturale, in particolare di sogni, e della loro rappresentazione tramite grafi, con l'ausilio di strumenti di analisi NLP per un eventuale preprocessing, la contrazione di nodi essere utile per:

- Individuare contesti sintattici (e possibilmente semantici) di parole e frasi, evidenziando l'interrelazione e la distanza tra gruppi di parole e frasi.
- Valutare la somiglianza di singoli brevi testi, come il racconto dei sogni, individuando le macro-caratteristiche strutturali comuni e le differenze tra esse.
- Individuare pattern ricorrenti di parole e insiemi di parole su un corpus più ampio di testi, con eventuale ausilio di strumenti statistici.
- Stabilire il grado di connettività del grafo delle parole in base al numero di insiemi componente (gruppi di parole) e di nodi presenti nelle contrazioni.

- Permettere un confronto automatico dei pattern strutturali con grafi multilivello che rappresentino un controllo, con l'eventuale ausilio di algoritmi che valutano il grado di somiglianza di grafi.

Bibliografia

[1] Edgar Altszyler et al. ≪The interpretation of dream meaning: Resolving ambiguity using Latent Semantic Analysis in a small corpus of text≫. In: *Consciousness and Cognition* 56 (2017), pp. 178–187. ISSN: 1053-8100. DOI: https://doi.org/10.1016/j.concog.2017.09.004. URL: https://www.sciencedirect.com/science/article/pii/S1053810017301034.

[2] Daniel Archambault, Tamara Munzner e David Auber. ≪TopoLayout: Multilevel Graph Layout by Topological Features≫. In: *IEEE Transactions on Visualization and Computer Graphics* 13.2 (2007), pp. 305–317. DOI: 10.1109/TVCG.2007.46.

[3] Coen Bron e Joep Kerbosch. ≪Algorithm 457: finding all cliques of an undirected graph≫. In: *Commun. ACM* 16.9 (1973), 575–577. ISSN: 0001-0782. DOI: 10.1145/362342.362367. URL: https://doi.org/10.1145/362342.362367.

[4] Tom B. Brown et al. ≪Language Models are Few-Shot Learners≫. In: *CoRR* abs/2005.14165 (2020). arXiv: 2005.14165. URL: https://arxiv.org/abs/2005.14165.

[5] Thomas H. Cormen et al. *Introduction to Algorithms*. 3ª ed. Cambridge, MA: MIT Press, 2009. ISBN: 978-0-262-03384-8.

[6] Wenfei Fan et al. ≪Making Graphs Compact by Lossless Contraction≫. In: *Proceedings of the 2021 International Conference on Management of Data*. SIGMOD '21. Virtual Event, China: Association for Computing Machinery, 2021, 472–484. ISBN: 9781450383431. DOI: 10.1145/3448016.3452797. URL: https://doi.org/10.1145/3448016.3452797.

[7] Yellen J. Anderson M. Gross J.L. *Graph Theory and Its Applications*. 3ª ed. Cambridge, MA: Chapman e Hall/CRC, 2018. DOI: https://doi.org/10.1201/9780429425134.

[8] Donald B. Johnson. ≪Finding All the Elementary Circuits of a Directed Graph≫. In: *SIAM Journal on Computing* 4.1 (1975), pp. 77–84. DOI: 10.1137/0204007. eprint: https://doi.org/10.1137/0204007. URL: https://doi.org/10.1137/0204007.

[9] N Mota, R Furtado, P Maia et al. ≪Graph analysis of dream reports is especially informative about psychosis≫. In: *Scientific Reports* 4 (2014), p. 3691. DOI: 10.1038/srep03691. URL: https://doi.org/10.1038/srep03691.

[10] N.B. Mota, M. Copelli e S. Ribeiro. ≪Thought disorder measured as random speech structure classifies negative symptoms and schizophrenia diagnosis 6 months in advance≫. In: *npj Schizophrenia* 3 (2017), p. 18. DOI: 10.1038/s41537-017-0019-3. URL: https://doi.org/10.1038/s41537-017-0019-3.

[11] Peter Sanders e Christian Schulz. «Engineering Multilevel Graph Partitioning Algorithms». In: *CoRR* abs/1012.0006 (2010). arXiv: 1012.0006. URL: http://arxiv.org/abs/1012.0006.

[12] Peter Sanders e Christian Schulz. «High quality graph partitioning». In: *Graph Partitioning and Graph Clustering*. 2012. URL: https://api.semanticscholar.org/CorpusID:8553747.

[13] M. Sharir. «A strong-connectivity algorithm and its applications in data flow analysis». In: *Computers & Mathematics with Applications* 7.1 (1981), pp. 67–72. ISSN: 0898-1221. DOI: https://doi.org/10.1016/0898-1221(81)90008-0. URL: https://www.sciencedirect.com/science/article/pii/0898122181900080.

[14] James C. Tiernan. «An efficient search algorithm to find the elementary circuits of a graph». In: *Commun. ACM* 13.12 (1970), 722–726. ISSN: 0001-0782. DOI: 10.1145/362814.362819. URL: https://doi.org/10.1145/362814.362819.

[15] Etsuji Tomita, Akira Tanaka e Haruhisa Takahashi. «The worst-case time complexity for generating all maximal cliques and computational experiments». In: *Theoretical Computer Science* 363.1 (2006). Computing and Combinatorics, pp. 28–42. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2006.06.015. URL: https://www.sciencedirect.com/science/article/pii/S0304397506003586.

[16] Herbert Weinblatt. «A New Search Algorithm for Finding the Simple Cycles of a Finite Directed Graph». In: *J. ACM* 19.1 (1972), 43–56. ISSN: 0004-5411. DOI: 10.1145/321679.321684. URL: https://doi.org/10.1145/321679.321684.