UNIVERSITÀ DEGLI STUDI ROMA TRE

Dipartimento di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica (LM-32)

Tesi Di Laurea

Interfacce visuali per grafi clusterizzati: primitive di interazione e strumenti di semplificazione

Relatore

Prof. Maurizio Patrignani

Candidato

Luca De Silvestris

Matricola 486652

Correlatore

Prof. Giuseppe Di Battista

Anno Accademico 2018/2019

Ai miei genitori

Indice

Elenco delle figure

Introduzione

Ringraziamenti

Una laurea.

Una laurea è passione, sacrificio e dedizione. Una laurea è un percorso di crescita che porta alla gloria di un singolo giorno. Eppure nonostante questa consapevolezza, quel singolo giorno di una vita, idealizzato mediante una corona di alloro, ha il sorriso e la gioia della consapevolezza, del traguardo e della vittoria. Del potersi guardare la mattina dopo con uno sguardo diverso. Una laurea è il desiderio di rivalsa con se stessi in quella eterna lotta che si vive cercando una utopistica vittoria con se stessi, che anche se per un solo giorno diventa reale. Detto questo, non è facile andare a ritroso negli anni e rievocare ricordi, emozioni e sensazioni per arrivare ad avere una lista di coloro che mi hanno reso ciò che sono e che spero contribuiranno ad essere ciò che sarò. Desidero ringraziare tutte quelle persone che, con suggerimenti, critiche e osservazioni, hanno fornito un importante aiuto nell'esperienza di questi cinque anni appena trascorsi, con la promessa di migliorarmi sempre. Ringrazio mia madre, mio padre e mia sorella a cui ho dedicato questi miei anni di studio, per aver creduto in me fin dall'inizio di questo percorso. Una dedica, in particolare, va a mia sorella Diandra: nella speranza che tu possa sempre gioire dei miei traguardi come io dei tuoi e con la consapevolezza che ti appoggerò sempre, anche se con una visione critica e cinica, in qualunque tua scelta futura.

Luca De Silvestris

Premessa

Il lavoro che verrà descritto in questa tesi rappresenta la relazione finale di un progetto svolto dal candidato nell'ambito della visualizzazione delle informazioni sul tema dei grafi clusterizzati. Il progetto ha riguardato lo studio delle primitive di interazione e degli strumenti di semplificazione di grafi clusterizzati. Lo studio teorico delle primitive è supportato dallo sviluppo del primo editor per clustered graphs.

Le macro-aree di interesse sono quindi la visualizzazione delle informazioni, la teoria dei grafi le primitive di integrazione e lo sviluppo software web. Tutti i concetti enunciati nella descrizione sintetica qui riportata verranno approfonditi, illustrati, documentati e motivati nel corso della lettura.
Il documento si concentra in questi undici capitoli di seguito anticipati:

1. **Definizioni preliminari:** breve panoramica sui concetti di base;
2. **Stato dell'arte:** Digressione su ciò che attualmente è utilizzabile;
3. **Grafi clusterizzati:** definizione dei concetti di studio;
4. **Analisi degli obiettivi**
5. **Primitive atomiche di interazione:** elenco delle possibili primitive di interazione per grafi clusterizzati
6. **Operazioni di semplificazione per grafi clusterizzati** .
7. **Editor: interfacce e strutture dati**
8. **Editor: interazioni e semplificazioni**
9. **Esempio reale di utilizzo**
10. **Conclusioni**

Capitolo 1

Definizioni Preliminari

L'intero lavoro svolto si basa su due importanti settori informatici quali la teoria dei grafi e la visualizzazione delle informazioni. Queste due aree di studio saranno viste di seguito nel dettaglio.

1.1 Visualizzazione delle informazioni

La visualizzazione delle informazioni può essere definita come l'utilizzo o comunque l'impiego di rappresentazioni visuali ed interattive di dati astratti, con il compito di amplificarne la conoscenza degli stessi o anche come la comunicazione di dati attraverso l'utilizzo di interfacce visuali. Questo particolare settore è di grande rilievo sia per l'analisi dei dati che per la presentazione e quindi per la comunicazione degli stessi. Importante però è non assegnare alla visualizzazione delle informazioni il mero scopo di creazione di grafici privi di informazione in quanto essa è la base su cui si andrà a lavorare.
Per quanto riguarda i dati astratti citati prima, essi possono essere di due tipi, strutturati e non strutturati. I dati **strutturati** sono rigidamente formattati, si stima che siano soltato il 20% dei dati che si producono e sono solitamente

organizzati in tabelle. Quelli **non strutturati** risultano essere invece difficili da analizzare tanto che uno dei primi compiti antecedente la visualizzazione è quello di riuscire a trasformare dati non strutturati in dati tabellari o dati flat. Per la rappresentazione dei dati inoltre sono di notevole importanza i principi di questo ambito informatico che prevedono le seguenti operazioni che precedono la visualizzazione dei dati:

- **Estrazione:** download,caricamento o conversione dei formati;
- **Semplificazione:** operazioni di filtraggio rimozione e aggregazione;
- **Aggiustamento:** può riguardare operazioni matematiche e variazioni sul tipo di dato.

Definita la visualizzazione delle informazioni è necessario, evidenziare i principi per una rappresentazione dei dati corretta e soprattutto il modello di progettazione. **Tamara Macushla Munzner** propone un modello per la visualizzazione orientato al task model composto da 4 fasi:

- **Dominio:** identificazione del problema e del dominio di interesse;
- **Astrazione dei dati:** si modellano i dati che dovranno essere rappresentati;
- **idioma di codifica:** si decide come visualizzare i dati e quali interazione implementare all'interfaccia;
- **Algoritmo**.

Tra tutti questi principi del modello di Munzner inoltre esiste un rapporto di inclusione in cui dalla fase più esterna ad una più interna esiste un rapporto di molteplicità. In altre parole a puro titolo di esempio diversi idiomi possono corrispondere alla stessa astrazione e diverse astrazioni possono corrispondere allo

stesso dominio.

Esistono due tipologie di approccio per una visualizzazione orientata al task model. Il primo è quello dell'approccio **Top-Down**, ovvero guidato dal problema e risolto dal dominio fino all'algoritmo applicando le varie scelte progettuali. Il secondo è invece quello dell'approccio **Bottom-up** ed è guidato dalla tecnica. Per concludere la digressione sui principi della visualizzazione è importante anche definire il concetto di **User-Task** come scopo dell'utente ovvero cosa deve poter fare sui dati. Nella visualizzazione delle informazioni ci sono due principali tipi di scopi:

- **Presentazione**, contesto in cui si conosce l'informazione e la visualizzazione è tesa a rappresentarla;
- **Analisi**, contesto in cui la visualizzazione è tesa all'estrazione delle informazioni dei dati rappresentati.

1.2 Interazione

Definendo la disciplina della visualizzazione delle informazioni come il mezzo per poter creare interfacce interattive per riuscire ad estrarre informazione dai dati è necessaria adesso una digressione sul concetto di interazione dell'utente e delle sue primitive. L'interazione assume definizioni diverse ma non discordanti tra loro a seconda dell'ambito di lavoro e del campo a cui questa fa parte. Per quanto concerne il campo della ricerca **HCI**, ovvero della Human Computer Interaction, l'interazione è stata definita già nel 1998 come la comunicazione tra uomo e macchina. Per definire a pieno lo human-computer interaction e l'usabilità bisogna far riferimento al modello di Donald Arthur **Norman**, psicologo e ingegnere

statunitense. Egli identifica l'interazione utente-calcolatore nelle 7 fasi riportate di seguito:

1. formulare l'obiettivo
2. formulare l'intenzione
3. identificare l'azione
4. eseguire l'azione
5. percepire lo stato del sistema
6. interpretare lo stato del sistema
7. valutare il risultato rispetto all'obiettivo

Secondo **Norman** inoltre un principio fondamentale è capire gli utenti e i compiti che intendono svolgere facendo anche in modo che le interfacce utente consentano tali compiti nel modo più immediato e intuitivo possibile. Norman infine fa notare, nel suo libro "La caffettiera del masochista", come questo principio sia spesso inutilizzato.
Nel campo della ricerca nella visualizzazione delle informazioni all'interazione furono date più definizioni, una successiva temporalmente alle altre. Di seguito ne sono riportate due fondamentali:

- **2002**: l'interazione consente la manipolazione diretta della rappresentazione grafica del dato;
- **2007**: l'interazione fornisce agli utenti la capacità di manipolare direttamente o indirettamente e modificare le rappresentazioni.

Già da queste definizioni è possibile notare la fondamentale importanza dell'interazione dell'utente poiché una rappresentazione senza la possibilità di modifica, restando comunque utile all'utente finale, potrà solo essere analizzata staticamente in quel suo ambito di verità. Grazie all'interazione l'utente non solo potrà eseguire task di analisi ma anche di manipolazione creando a sua volta nuove modalità di analisi ed acquisendo consapevolezza della fondamentale importanza del fattore umano sulla macchina.
Date le varie definizioni dell'interazione e definito il suo impiego nella visualizzazione delle informazioni è facile notare come tutte le interazioni dipendono da vari fattori che ne influenzano il comportamento e spesso anche il successo oppure il declino di una particolare modalità di visualizzazione. Ogni interazione può inoltre essere classificata a seconda di vari fattori che sono introdotti di seguito:

- **Tempo di risposta**:l'ordine di tempo a cui appartiene il tempo di risposta dell'interazione al comando dell'utente al sistema è la prima grande classificazione. Se il tempo di risposta risulta essere dell'ordine dei 10^-1 secondi l'interazione è velocissima e fa uso di animazioni e l'utente potrà eseguite tante operazioni in un tempo irrisorio, se risulta dell'ordine dei 10^0 secondi l'animazione farà uso di dialoghi o finestre di dialogo che molto spesso indirizzeranno l'utente per future interazioni e se infine sarà superiore ai 10^1 secondi interazioni saranno cognitive ovvero task che verranno eseguiti prima di ottenere risposte.

- **Azioni atomiche**: rispondono alla domanda "in che modo si eseguiranno le interazioni". Queste potrebbero essere eseguibili tramite tasti o azioni eseguiti con l'ausilio di un mouse o ancora potrebbero essere azioni di più alto livello come vocali, gestuali o touch.

- **Tecnologia**: una interazione non è nulla senza il dispositivo su cui la si

esegue. Ricordando che differenti tipologie di dispositivi forniscono diversi tipi di interazioni si può classificare l'interazione anche mediante la scelta del dispositivo utilizzato che sia esso un laptops, un tablets o uno smartphone fino ad arrivare ad avere interazioni specifiche per dispositivi indossabili che riconoscono ad esempio i movimenti gestuali.

- **Grado di interazione**: ogni interazione possiede un grado definito come il livello di connessione e di padronanza delle operazioni che l'utente può ottenere interagendo con il sistema. In particolare il grado di interazione può essere:

  1. **statico** ovvero grado 1 in cui non è prevista alcun genere di interazione riducendo la rappresentazione ad una singola vista per una analisi non modificabile;
  2. **manipolabile** ovvero di grado 2 in cui è possibile cambiare la vista della scena e quindi evidenziare ed analizzare meglio un dato dalla sua rappresentazione non potendo però modificare i dati da cui è nata la visualizzazione;
  3. **trasformabile** ovvero di grado 3 dove l'interazione permette di intervenire nella fase di processo e cambiare il dato di input.

- **paradigma**: intesa come tipologia di interazione vera e propria che potrà essere tradizionale dove c'è un punto di comunicazione tra l'uomo e la macchina, realtà aumentata o la realtà virtuale.

Date le definizioni inserenti l'ambito della visualizzazione delle informazioni, si passa ora alle definizioni inerenti la teoria dei grafi.

1.3 Grafi

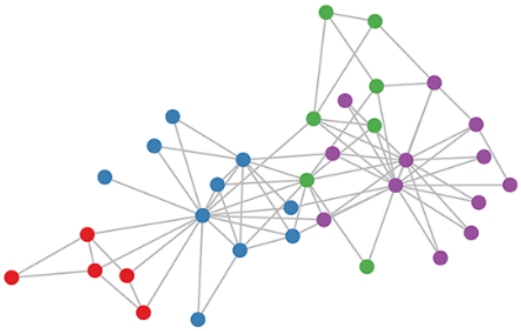

Figura 1.1: Esempio di grafo indiretto connesso

La teoria dei grafi è la disciplina che si occupa dello studio dei grafi, oggetti discreti che permettono di schematizzare una grande varietà di situazioni, processi e spesso di consentire delle analisi in termini quantitativi ed algoritmici. Un grafo G è definito da una coppia di insiemi $< V, E >$ in cui V è un insieme di nodi o vertici che possono essere connessi tra loro mediante l'insieme di archi E tale che i suoi elementi siano coppie di elementi dell'insieme V esprimibile come

$$E \subseteq V * V$$

Si definisce poi grafo **completo** un grafo in cui ogni vertice è collegato con tutti gli altri vertici rimanenti, di modo che l'insieme degli archi E è esplimibile come $E = V * V$. Dato un grafo è possibile avere un disegno $\tau(G)$ come mapping dei vertici su punti distinti del piano.
Si distinguono due tipi di grafi:

- **non orientati**, dove la relazione E è simmetrica, quindi

$$(a, b) \in E \Rightarrow (b, a) \in E$$

. In questo tipo di grafo, gli archi sono solitamente denominati spigoli e i nodi vertici.

- **orientati**, dove la relazione E non è simmetrica ed esiste una relazione d'ordine tra i nodi.

In un grafo infine si definisce ciclo (o circuito) di lunghezza m una catena $(v0, ..., vm-1, v0)$. Si dice inoltre ciclo semplice un ciclo che non passa due volte dallo stesso nodo, o formalmente un ciclo $(v0, ..., vm-1, v0)$ per il quale:

$$\forall i, k = 1, ..., m-1, (i \neq k \Rightarrow v_i \neq v_k)$$

I grafi risultano comunque essere una delle strutture fondamentali su cui si basano molte applicazioni reali ed anche il principio fondamentale alla base del lavoro svolto.

1.4 Alberi

Data la definizione di grafo, si può definire un albero **tree** come un grafo connesso non orientato e senza cicli come mostrato in **Figura 1.2**.
Per essere tale quindi il grafo in questione deve possedere un solo cammino per ogni coppia di vertici oppure essere aciclico massimale. Ogni nodo appartenente ad un albero possiede un livello dato alla somma $1 + l(padre)$ intendendo che la radice dell'albero ha livello 0. La radice è definita come l'unico nodo dell'albero che non possiede archi entranti ma solo archi uscenti e a cui solitamente viene data rappresentazione del livello più basso dell'albero in altre parole è l'unico nodo che non presenta nodi con livello inferiore(i nodi padri) ma solo nodi con livello superiore(i nodi figli). Per completezza si voglio dare anche le definizioni di profondità di un nodo e di altezza di un albero rispettivamente come la lunghezza

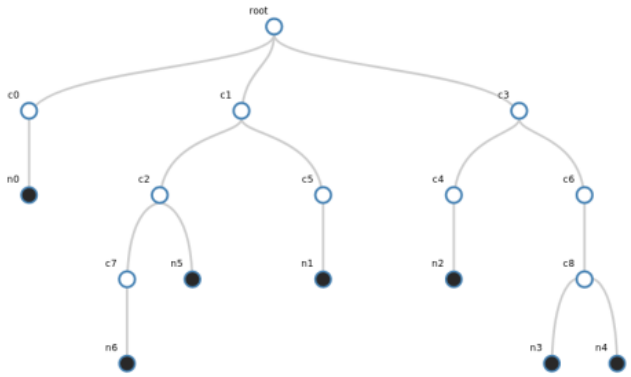

Figura 1.2: esempio di albero n-ario

del cammino dalla radice al nodo e la profondità massima dei suoi nodi. I nodi dell'albero diversi dalla radice possono inoltre essere partizionati in due categorie:

- **nodi interni**: definiti come nodi con almeno un figlio e suddivisibili ulteriormente in nodi bassi, in cui tutti i figli sono foglie ed in nodi alti, che possiedono almeno un nodo interno come figlio;
- **foglie** che non possiedono figli e per questo non hanno archi uscenti.

Un nodo è inoltre definibile **omogeneo** se tutti i suoi figli sono foglie oppure tutti nodi interni. Data la definizione di nodo omogeneo, viene detto che un albero è omogeneo se e solo se tutti i suoi nodi sono nodi omogenei. Si definisce poi $S(T)$ come la dimensione dell'albero T, ovvero il numero di nodi superiori di T diversi dalla radice. Date queste definizioni introduttive relative agli alberi è possibile definire un albero flat (come mostrato in **Figura 1.3**) con il lemma seguente:

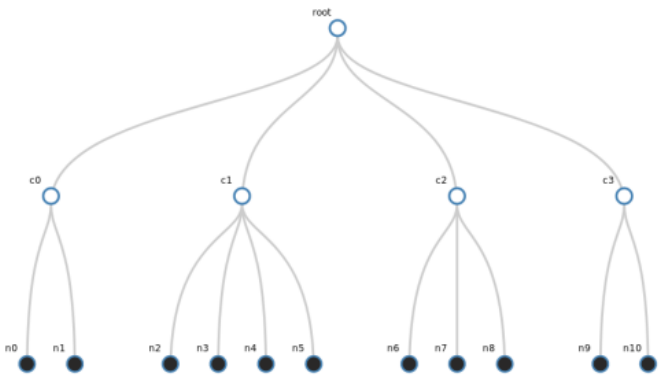

Figura 1.3: esempio di flat Tree

Un albero omogeneo T di altezza $h(T) >= 2$ e dimensione $S(T) > 0$ e che contiene almeno un nodo $\mu^* \neq r(T)$ in modo tale che $T(\mu^*)$ è flat

Definendo in questo modo un albero flat è facile notare come ogni albero flat è anche omogeneo, avendo tutti i nodi interni che hanno come figli esclusivamente foglie ed il nodo radice che possiede solo nodi interni.

Capitolo 2

Stato dell'arte

Di seguito saranno analizzate le possibili tecniche di visualizzazione di grafi ed alberi, come definito anche nel libro "Graph Drawing: Algorithms for the Visualization of Graphs" [BETT98] e saranno elencate le modalità di interazione per le rappresentazioni di grafi concludendo infine con una lista di software attualmente in commercio che svolgono tali compiti.

2.1 Visualizzazione alberi

Per quanto concerne la rappresentazione dei grafi ed in particolare degli alberi, ci sono due possibili modelli: node-link e space-filling. Nella strategia node-link i nodi solo rappresentati come dei punti mentre i link ovvero i collegamenti (gli archi) sono raffigurati come linee. A seconda poi della rappresentazione Node-link scelta si possono avere diverse tipologie di visualizzazione. Ne è un esempio la rappresentazione HV-drawing mostrata nella **Figura 2.1**, in cui gli archi possono essere solamente orizzontali e verticali e che risulta essere molto utile per la rappresentazione di circuiti elettronici.

Nella strategia Space-filling si tenta invece di superare i limiti del Node-link che

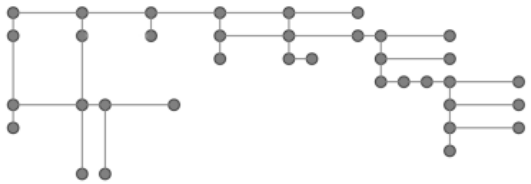

Figura 2.1: rappresentazione node-link HW drawing di un albero

riguardano la gestione non ottimale dello spazio orizzontale in quanto la parte alta della rappresentazione solitamente risulta essere poco densa di informazioni al contrario di quella bassa.
Nella strategia Space-filling forme geometriche, solitamente rettangolari, rappresentano i nodi e i figli sono rappresentati da altre forme geometriche inserite all'interno del padre. Questa strategia presenta però vari problemi tra cui la distinzione difficile di tagli orizzontali da quelli verticali e la gestione non ottimale di alberi di grande profondità. Nella **Figura 2.2** sono mostrate alcune rappresentazioni diverse che seguono la strategia Space-filling.

Figura 2.2: rappresentazioni Space-filling

Essendo quest'ultima di difficile lettura solitamente però si opta per la strategia Node-link scegliendo una rappresentazione "layered drawing",di cui ne è un esempio la Figura 2.3. In questa rappresentazione i nodi sono organizzati in livelli e per

ogni nodo la coordinata y dipende da esso mentre la coordinata x deve esser trovata e dipende: dal numero di figli per ogni livello e dello spazio orizzontale a disposizione per la rappresentazione.

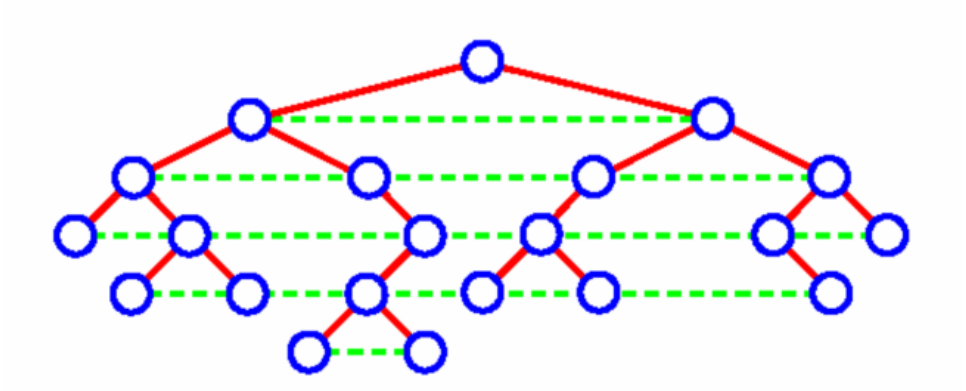

Figura 2.3: rappresentazione node-Link layered Drawing

2.2 Visualizzazione grafi

Il modello principale utilizzato per la visualizzazione di grafi è il **Force directed** che utilizza una rappresentazione Node-link ed emula il sistema fisico formato da cariche elettriche rappresentate dai nodi e da molle rappresentati dagli archi. L'idea alla base del modello è quella che il raggiungimento di una condizione di equilibro corrisponda ad una rappresentazione grafica gradevole. Ogni rappresentazione di un grafo è dunque formata da due componenti: il **modello** che descrive le forze in gioco e l'**algoritmo** ovvero la tecnica che permette di stabilire il punto di equilibrio accettabile. Esistono sono molte varianti del modello Force-directed ma che condividono lo scopo del raggiungimento della condizione di equilibrio. Il primo modello è Barycenter method (Tutte - 1963), seguito dallo Springs & electrical forces (Eades - 1984) e dal Magnetic fields (Sugiyama & Misue - 1995).

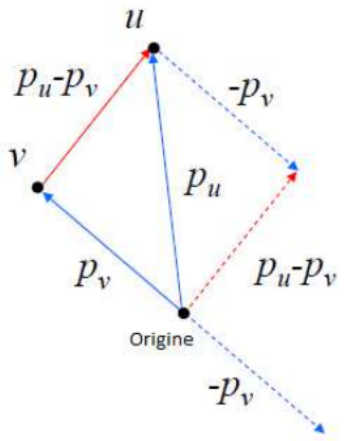

Figura 2.4: Baricenter method

Il barycenter method è il primo e il più semplice tra i modelli Force directed. Supponiamo di avere due nodi u e v aventi coordinate rispetto ad un punto di origine e consideriamo poi i due vettori P_u e P_v che partono dall'origine ed arrivano a u e v. Eseguendo la sottrazione dei vettori $Pu - Pv$, il risultato è il vettore che parte da v e termina in u. Questo vettore rappresenta la forza attrattiva di v verso u ed ogni nodo vicino a v attrae v stesso.

$$F(v) = \sum_{\forall u \in N(v)} (P_u - P_v)$$

La configurazione di equilibrio si ha quando la somma di tutte le forze è pari a 0, $F(v) = 0$.

Il modello Springs & electrical forces che permette di avere una rappresentazione simile a quella nella figura **Figura 2.5** Combina forze elettriche e forze meccaniche. Tra i nodi viene inserita una molla caratterizzata da una certa costante elastica, se avvicino i nodi li respingono, se li allontano la molla li avvicina. La molla tende ad una situazione di equilibrio. Ad ogni coppia di vertici è possibile associare una

molla con lunghezza diversa. In formula:

$$F(v) = \sum_{(u,v)\in E} f_{uv} + \sum_{(u,v)\in VxV} g_{uv}$$

con g forza repulsiva elettrostatica ed f forza della molla.

Il metodo Spring Embedding produce buoni disegni, mette in rilievo anche

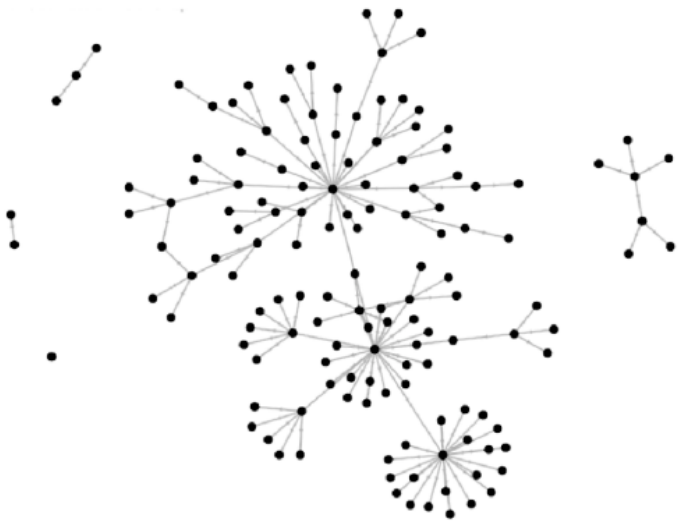

Figura 2.5: rappresentazione Spring Embedding

i cluster di vertici ed enfatizza le simmetrie. Queste proprietà non sono però scientificamente previsti. La sua caratteristica fondamentale è la flessibilità, non ci sono restrizioni sul grafo. A differenza del metodo del baricentro di Tutte qui può essere connesso, non connesso ed è inoltre possibile inserire forze a piacimento. Per quanto concerne invece la complessità computazionale, il metodo non garantisce il numero di iterazione minimo per poter raggiungere una configurazione di equilibrio ed ogni passo iterativo ha complessità quadratica per i nodi e lineare per tutte le iterazioni avendo quindi un costo complessivo finale

$O(n^3)$ senza lasciare però nessuna garanzia sul layout del grafo rappresentato. Nella rappresentazione Magnetic fields infine le molle vengono magnetizzate in modo che reagiscono a dei campi magnetici esterni. La molla può essere magnetizzata in modo unidirezionale o bidirezionale. Per ogni arco, quindi per ogni molla, si va a calcolare la forza

$$d(p_u, p_v)^{\alpha} x sin(\theta)^{\beta}$$

in cui α e β sono due costanti positive e le forze su u e v tendono a stendere la molla lungo il campo magnetico.
Le linee di forza del campo magnetico esterno possono essere parallele, radiali e concentriche, potendo quindi dar vita a diverse rappresentazioni grafiche dello stesso grafo. All'aumentare però del numero di forze in gioco il sistema rallenta e difficilmente il disegno riesce poi a soddisfarle tutte, la rappresentazione grafica perde in bellezza.

2.3 Primitive di interazione per visualizzazioni di grafi

Per quanto riguarda le interazioni per la rappresentazione dei grafi si elencano qui di sequito quelle elementari:

- creazione di un nodo: ogni nodo avrà conoscenza degli archi che sanno collegati ad esso ed avrà delle caratteristiche quali: coordinate nella visualizzazione, colore ed una etichetta che lo contraddistingue;
- creazione di un arco: ogni arco avrà conoscenza dei due nodi che connette;
- cancellazione arco: eliminando un arco lo si dovrà rimuovere anche dalla lista degli archi collegati ai nodi che esso connetteva;

- cancellazione nodo: ogni qualvolta si elimina un nodo se questo aveva degli archi allora si dovranno cancellare anche i suddetti in quanto un arco è un collegamento tra due nodi.

- modifica caratteristiche di un nodo

Definite le interazioni elementari da poter eseguire su un grafo si passa adesso a definire le operazioni più semplici valide per tutte le visualizzazioni non solo di grafi. Le operazioni semplici che vedremo nel dettaglio sono quelle di select, panning e lo zooming sia spaziale che logico.

Per quando concerne l'operazione di **select**, mediante una interazione è possibile selezionare un certo numero di punti da rappresentare. Immaginando di avere a disposizione dei dati tabellari riferiti ad un gran numero di nodi ed archi da rappresentare è di grande aiuto poter avere una rappresentazione di un numero di dati selezionato. Lo **zooming** ovvero la possibilità di effettuare un ingrandimento o un ridimensionamento per potersi focalizzare su una parte spaziale di uno oggetto risulta essere una tra le principali primitive di interazione dell'utente. Lo zooming viene utilizzato per per mostrare o nascondere dati, ha significato sia spaziale che logico. E' possibile eseguire una operazione di zooming e di panning anche contemporaneamente. Immaginiamo di avere un oggetto grafico disegnato su di un piano $< x, y >$ ma replicato su più livelli con grado di ingrandimento diverso lungo v, come mostrato nella Figura 2.6. La distanza dal piano $< x, y >$ determina il grado di zooming mentre il movimento sul piano determina il panning.

Tenendo la finestra della visualizzazione bassa riesco a includere dentro ad essa molti oggetti, man mano che si esegue lo zoom out lungo v in un punto anche diverso dal centro, gli oggetti inclusi dalla finestra diminuiscono. Inoltre un

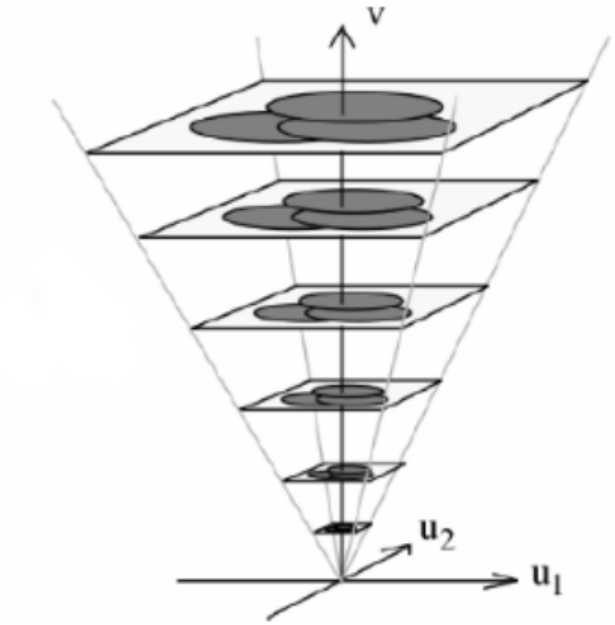

Figura 2.6: Operazione di zooming su un piano $< x, y >$

principio da seguire è che per muoversi lungo il piano ad un livello di zoom molto basso, ovvero quando si è lontani dall'origine e si possono vedere pochi oggetti, piuttosto che fare un panning complesso risulta più efficiente eseguire un zoom out avvicinandosi all'origine, un piccolo panning e poi uno zoom in sino al punto di arrivo sullo stesso piano iniziale come mostrato nella Figura 2.7

Per terminare con le primitive delle operazioni *semplici* sarebbe necessario nominare anche il labelling ovvero l'operazione che grazie ad una interazione spesso con un tempo di risposta dell'ordine del secondo porta ad una etichettatura di un oggetto del grafo. Anche il labelling però risulta comunque una forma di zooming poiché impiegando una interazione si mostra nella rappresentazione un numero di dati più elevato come fosse uno zooming focalizzato su tutta la visualizzazione.

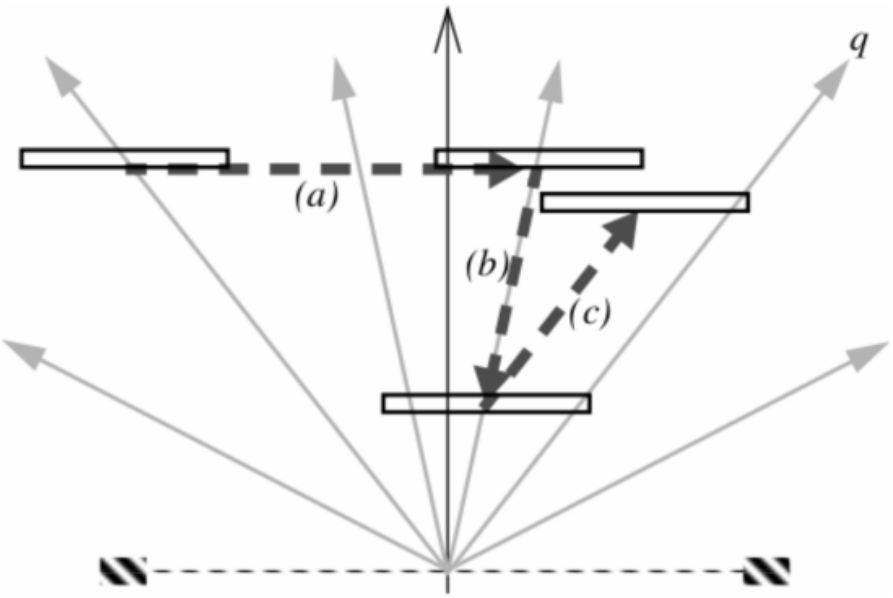

Figura 2.7: Operazioni di panning e zooming su un piano $< x, y >$

Dalle primitive di integrazione che portano alle operazioni più semplici e pressoché necessarie in ogni visualizzazione saranno ora viste nel dettaglio alcune operazioni, date comunque dall'interazione uomo-macchina con le modalità descritte prima, definibili di completezza per un sistema. Di grande importanza infatti rappresentano la possibilità da parte dell'utente e della sua interazione di riconfigurare e di eseguire un encode dei dati. Per quanto concerne la **riconfigurazione**, questa operazione è definita come la possibilità mediante una interazione di poter cambiare gli attributi o i dati visualizzati.E' in questo modo possibile modificare cosa visualizzare, cambiando gli intervalli, la scala o ordinare i dati. L'**encode** invece è definito come una modifica della visualizzazione e non dei dati. Solitamente mediante una operazione di encode si cambia il tipo di visualizzazione passando,

come mostrato a titolo di esempio nella figura Figura 2.8, da una rappresentazione a diagramma a barre ad una torta. Include anche operazione di modifica sui colori, sulle forme e l'orientamento della visualizzazione. La primitiva che risulta

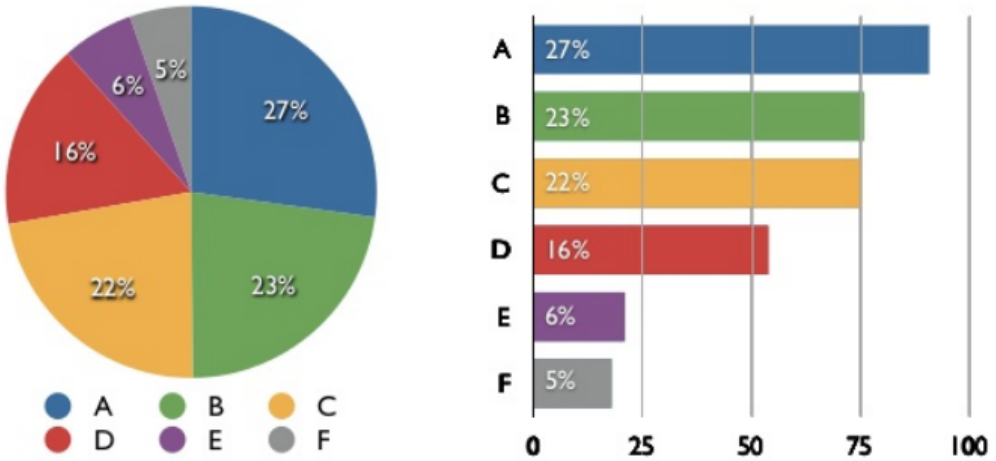

Figura 2.8: Esempio di encoding da un diagramma a torta ad uno a barre

comunque essere maggiormente complessa da realizzare è però proprio quella su cui si basano le proprietà di creazione e di modifica di un oggetto facente parte del grafo: la **connessione**. Riuscire infatti a dare la possibilità di eseguire con una interazione una operazione di connessione non solo a livello grafico mediante la visualizzazione ma anche dei dati risulta essere uno degli obiettivi più complessi da raggiungere.

Per concludere la digressione introduttiva delle primitive di iterazione e delle operazioni eseguibili mediante le stesse e per completezza si vuol dare la definizione della primitiva di filtraggio. Il **filtraggio** è l'operazione che permette di mostrare un sottoinsieme di dati secondo certe regole, ovvero ciò che può esser definito

come uno zoom semantico. Può far uso di query tradizionali ma anche di query dinamiche. Con le query dinamiche l'operazione è rapida, immediata,cambiando un parametro la visualizzazione cambia di conseguenza, ed è immediatamente reversibile, i tempi di attesa sono minori del decimo di secondo. Nelle query classiche invece l'attesa è più alta, l'utente forma le query che vengono eseguite sulla base di un evento, che nel nostro caso risulta essere la pressione di un bottone. Tipicamente però i filtraggi eseguiti con le query dinamiche non sono complesse e le operazioni riguardano tutti i dati. È naturale pero immaginare come aumentando delle dimensioni dei dati l'interazione diviene sempre più lenta.

2.4 Editors per grafi

Definite le primitive di interazione e le modalità di visualizzazione si passa adesso all'elenco e alla descrizione dei software per la loro creazione. Attualmente sono disponibili pochi editor **specifici** per la visualizzazione e anche per la creazione e la conseguente eventuale modifica di grafi. Un esempio famoso ne è Graphviz, software di visualizzazione di grafi open source. Utilizzando Graphviz non si ha però la conoscenza della struttura dati da cui si genera la visualizzazione in quanto rappresenta un mero software grafico. Altri software per la manipolazione di grafi, ognuno per un particolare interesse nell'ambito della teoria dei grafi, sono:

- **Gephi**: software di visualizzazione ed esplorazione per tutti i tipi di grafi e reti, open source e gratuito.
- **Porgy**: software che mira a progettare rappresentazioni grafiche pertinenti e interazioni adeguate sui grafi dinamici che emergono dai sistemi di riscrittura dei grafi.

- **MS-AGL** :Microsoft Automatic Graph Layout è uno strumento .NET per il layout e la visualizzazione dei grafi open source e basato sullo schema di Kozo Sugiyama([SM91], [SM95]) .

Esistono poi molti programmi di editor non specializzati e molto più generici come ad esempio Draw.io software di diagrammi online gratuito per la realizzazione di diagrammi di flusso, diagrammi di processo, organigrammi, UML, ER e diagrammi di rete. Anche se non sono presenti molti editor specifici per la creazione di dati relativi ai grafi e la loro conseguente visualizzazione, le primitive di interazione elencate prima sono spesso riscontrabili, anche se in buona parte o con funzioni non totalmente esclusive negli editor non specializzati quali appunto Draw.io. Un esempio di grafo realizzato mediante uno dei tanti modelli disponibili dal software Draw.io è mostrata nella figura Figura 2.9 Un altro esempio, sempre non specifico

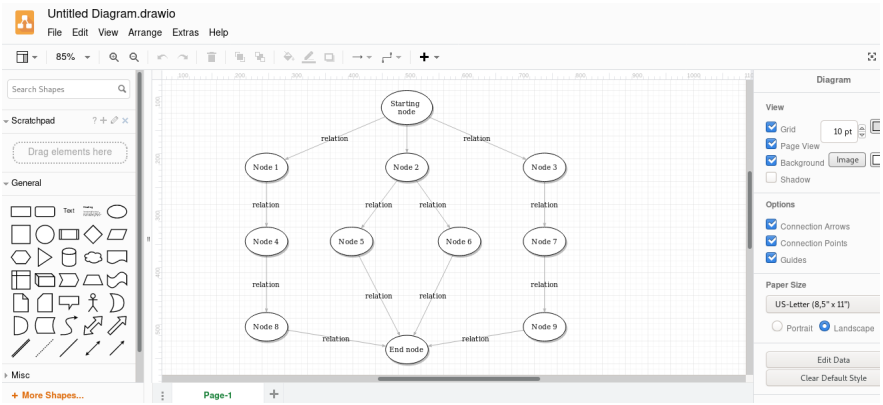

Figura 2.9: interfaccia del software draw.io

ed esclusivo per la creazione di grafi, è Microsoft Visio,software proprietario per la creazione di grafici e diagrammi, sviluppato da Microsoft per i sistemi operativi Microsoft Windows ed è rilasciato per diverse edizioni di Microsoft Office. Esiste poi la sua controparte **SmartDraw** utile per la visualizzazione dei dati in grafici

ma non supporta molto la creazione di strutture dati come i grafi. È facile notare come dunque nella teoria dei grafi si abbia la necessità di visualizzare le strutture dati e come questo problema non sia facilmente risolvibile dai software attualmente a disposizione.

Capitolo 3

Grafi Clusterizzati

3.1 Planarità

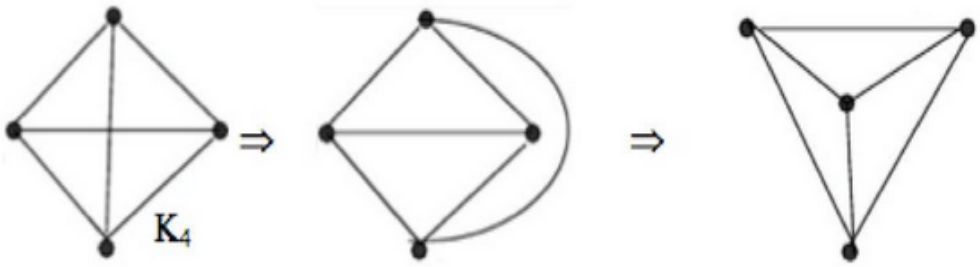

Figura 3.1: trasformazione del disegno di un grafo in un disegno planare

Data la definizione di grafo nel capitolo dedicato alle definizioni preliminari è necessario, ai fini di una introduzione al mondo della teoria dei grafi per ciò che concerne i grafi clusterizzati, enunciare la definizione di grafo planare.
Un grafo **planare** G, come mostrato nella Figura 3.1, è definito come un grafo tale che il suo disegno $\tau(G)$ può essere raffigurato in un piano in modo che non si

abbiano archi che si intersecano tra loro. È facile da intuire che non tutti i grafi sono planari e se ne riportano due famosi esempi in **Figura 3.2**. In particolare questi due grafi sono chiamati anche grafi di Kuratowski mediante i quali si può dare la definizione di grafo planare come enunciato del ***Teorema di Kuratowski***:

Un grafo è planare se e solo se non contiene alcun sottografo che sia una espansione di K_5 o una espansione di $K_{3,3}$

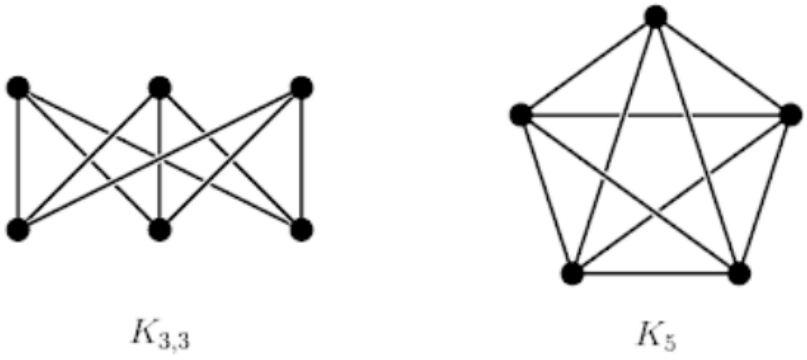

Figura 3.2: grafi di Kuratowski

Si ricorda inoltre che se un grafo è planare allora ammette un disegno planare $\tau(G)$. Dando un altra definizione si può evidenziare che un disegno $\tau(G)$ è un disegno planare se ogni arco non interseca nulla eccetto i due vertici che connette.
Si richiama ora all'algoritmo di Auslander e Parter del 1961.
Mediante questo algoritmo, del costo non lineare a livello di complessità computazionale ma uguale a $O(n^3)$, si è in grado di calcolare se un grafo in input è planare o meno mediante due fasi. Nella prima si decompone un grafo arbitrario nelle sue

componenti biconnesse e nella seconda si tenta di planarizzare ogni componente connessa. Definito grafo planare è anche possibile formulare il seguente teorema

un grafo è planare se il suo disegno è sempre colorabile con 4 colori diversi

La planarità è un punto chiave nella teoria dei grafi e definizione fondamentale per l'introduzione ai grafi clusterizzati.

3.2 C-graph

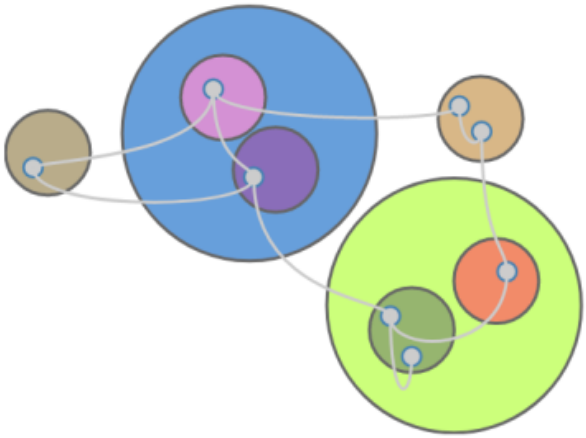

Figura 3.3: esempio di grafo clusterizzato

Un grafo clusterizzato (**c-graph**), di cui ne è un esempio la **Figura 3.3**, è definito come un grafo **planare** con una gerarchia ricorsiva definita sui suoi vertici. In altre parole un *c-graph* C è definito come

C=<G,T>

in cui:

- $G = (V, E)$ è un grafo planare, formato quindi da nodi e da archi le cui rappresentazioni non intersecano nulla se non il punto iniziale e finale, definito come *underlying graph* di C;
- T è un albero radicato definito come *inclusion tree* in cui l'insieme delle foglie di T coincide con l'insieme dei nodi V dell'underlying graph G ed ogni nodo interno è rappresentato da un **cluster**.

Un cluster in informatica può essere definito come un insieme di oggetti simili spesso connessi tra loro a cui al suo interno può contenere altri sotto-cluster. Nel caso specifico del termine si denota come un cluster potrà quindi contenere altri cluster al suo interno e/o nodi dell'insieme V dell'Underlying graph G. Un cluster essendo un nodo dell'albero di inclusione T dovrà quindi avere una "conoscenza" del nodo precedente(genitore) e dei nodi successivi(figli)

3.3 clustered planarity

Avendo definito la struttura di un grafo clusterizzato C si può andare ad analizzare il comportamento per quanto concerne il disegno $\tau(C)$. Avendo inoltre definito l'underlying graph Gcome un grafo planare si devono fare delle considerazioni per quanto riguarda il disegno dell'intero grafo clusterizzato C([FCE95], [Fen97], [CW06]). Questo disegno $\tau(C)$ sarà un disegno planare clusterizzato **c-planar** se:

(i) ogni cluster è rappresentato da una singola regione del piano che contiene solo i suoi vertici e gli archi a cui essi sono collegati;

(ii) definendo il perimetro del cluster come bordo del cluster, si ha che un disegno sarà planare se nessun bordo del disegno $\tau(C)$ si interseca con altri bordi di altri cluster;

(iii) ogni arco si interseca con un cluster al più una volta;

Per ciò che concerne la teoria dell'informatica e dei grafi, i problemi relativi alla planarità e al disegno planare di grafi clusterizzati risultano essere ancora di grande importanza e oggetto di molte ricerche in quanto ancora non si è in grado di definire la complessità del decidere quando un generico grafo clusterizzato C ammette un disegno planare clusterizzato $\tau(C)$. In altre parole risulta essere un problema aperto quello di capire a quale classe di complessità appartiene il problema sopra definito, se in P o NP. La classe di complessità P consiste di tutti quei problemi di decisione che possono essere risolti con una macchina di Turing deterministica in un tempo che è polinomiale rispetto alla dimensione dei dati di ingresso. La classe **NP** consiste invece di tutti quei problemi di decisione le cui soluzioni positive possono essere verificate in tempo polinomiale avendo le giuste informazioni, o, equivalentemente, la cui soluzione può essere trovata in tempo polinomiale con una macchina di Turing non deterministica. Tutto questo poi richiama anche il problema del millennio *P contro NP* che risulta ancora non risolto e che consiste nel capire se esistono problemi computazionali per cui è possibile "verificare" una soluzione in tempo polinomiale ma non è possibile "decidere", sempre in tempo polinomiale, se questa soluzione esiste. Si vuol precisare che le definizioni di macchina di Turing e di grammatiche di Chomsky non saranno date in questo lavoro poiché non necessarie ai fini dell'argomento trattato.

Capitolo 4

Analisi degli obiettivi

Di seguito saranno trattati i problemi riscontrati nello stato dell'arte. Come accennato nel capitolo 2, è palese la difficoltà nel reperire un software che tratti in maniera esaustiva ed esclusiva delle sole primitive di interazione dell'utente nell'ambito dei grafi. Anche avendo la possibilità di una visualizzazione spesso in tali sistemi non si ha però una struttura fissa alla base. Ne è un esempio GraphVis. Ad aggravare il tutto c'è anche il problema che

le primitive di interazione e le operazioni utente viste per quanto riguarda i grafi non possono essere utilizzare per la rappresentazione di grafi clusterizzati.

Tali primitive di interazione, che sono ben chiare e fisse per quanto concerne non solo la realizzazione ma anche la modifica e la possibilità di analisi di una visualizzazione per un grafo, tendono a portare a degli errori oppure ad una non completezza nel caso dei grafi clusterizzati. Avendo definito un grafo clusterizzato come una coppia $< G, T >$ con Underliyng Graph G e inclusion tree T, le primitive di interazione riportate nel capitolo due andrebbero a coprire solo ciò che concerne il grafo G. Risulta dunque necessaria una analisi iniziale sulla struttura

che compone il grafo clusterizzato di modo che sia possibile avere una lista di primitive standardizzate e di operazioni di trasformazione universali che potranno essere poi impiegate in qualunque sistema di creazione e visualizzazione di grafi clusterizzati. L'obiettivo principale posto è dunque quello di poter trovare una soluzione al problema dell'assenza di primitive di interazione relative a queste strutture. In altre parole il lavoro svolto verterà alla risoluzione dei problemi elencati di seguito:

- Le primitive viste per i grafi non sono utilizzabili per i grafi clusterizzati;
- Non vi sono operazioni di semplificazione per i grafi clusterizzati;
- I software per la realizzazione di visualizzazioni grafiche di grafi non sono esaustivi e non possono essere utilizzati per la rappresentazione di grafi clusterizzati.

Una volta analizzate e fissate le primitive di interazione è quindi necessario sopperire anche al problema dell'assenza di software relativi a queste strutture dati. Sarà dunque mostrato un utilizzo delle soluzioni, riportate risolvendo i primi due problemi, mediante la creazione di un editor per grafi clusterizzati che servirà da mero esempio pratico di come sia possibile, una volta analizzate sotto gli aspetti teorici, poter utilizzare queste primitive di interazione in un caso pratico. Il sistema in esempio, mediante le primitive analizzate in precedenza, avrà la capacità di eseguire vari task. Le varie modalità di utilizzo sono dunque riportate di seguito in ordine di tipica esecuzione di una sessione di lavoro:

I Creazione dei dati relativi ad un grafo clusterizzato;

II Visualizzazione mediante diverse interfacce;

III Modifica della struttura dati e della sua rappresentazione,

IV Esecuzione di operazioni di semplificazione.

Per il conseguimento dell'obiettivo (I) sono state utilizzate le strutture dati permettono un collegamento diretto tra i dati, la loro modifica e la loro rappresentazione sul piano di lavoro.
Per permettere una visualizzazione completa (II) si sceglierà di dare all'utente la possibilità di creare/eliminare e modificare ogni elemento dell'underlying graph potendo sempre passare ad una visualizzazione dell'Inclusion Tree che risulta essere d'aiuto durante le modifiche e soprattutto dopo la riduzione a grafo flat. La creazione dell'editor di esempio inoltre rientra in tutta quella branca dell'informatica definita ingegneria del software che divide la creazione del progetto in tre parti distinte ma sviluppate quasi in parallelo: analisi, progettazione ed implementazione. Sulla base della specifica dei requisiti prodotta dall'analisi, la progettazione ne definirà i modi in cui tali requisiti saranno soddisfatti, entrando nel merito della struttura che dovrà essere data al sistema software che deve essere implementato. In particolare durante il lavoro svolto è stata utilizzata una metodologia definita come **Agile**. Nell'ingegneria del software, l'espressione "metodologia agile", o sviluppo agile del software, si riferisce a un insieme di metodi di sviluppo del software emersi a partire dai primi anni 2000. Di grande spunto è stato anche il libro "Agile Software Development: Principles, Patterns, and Practices"[Mar02]. I metodi agili si contrappongono al modello a cascata e altri processi software tradizionali, proponendo un approccio meno strutturato e focalizzato sull'obiettivo

di consegnare al cliente, in tempi brevi e frequentemente, software funzionante e di qualità. Questi principi sono definiti nel "Manifesto per lo sviluppo agile del software" [KB01], pubblicato nel 2001 da Kent Beck, Robert C. Martin e Martin Fowler in cui si specificano le seguenti considerazioni:

Gli individui e le interazioni pù che i processi e gli strumenti

Il software funzionante più che la documentazione esaustiva

La collaborazione col cliente più che la negoziazione dei contratti

Rispondere al cambiamento più che seguire un piano

Fra le pratiche promosse dai metodi agili si trovano: la formazione di team di sviluppo piccoli, poli-funzionali e auto-organizzati, lo sviluppo iterativo e incrementale, la pianificazione adattiva, e il coinvolgimento diretto e continuo del cliente nel processo di sviluppo.

Capitolo 5

Primitive atomiche di integrazione

Nel capitolo che segue sarà trattata l'analisi e lo studio delle primitive di interazione applicabili ad un grafo clusterizzato. In particolare saranno divise a seconda della tipologia di operazione che si andrà ad eseguire sulla struttura dati o sulla loro visualizzazione mediante l'interazione. Le soluzioni trovate sono applicabili a qualunque grafo clusterizzato e tentano di dare almeno in parte uno standard per ciò che ne concerne la visualizzazione. Le seguenti primitive di interazione sono realizzabili ed eseguibili in qualunque sistema di grado 3 ovvero in grado di creare e trasformare i dati e la conseguente rappresentazione.

5.1 creazione di un oggetto

Per quanto riguarda le primitive di interazione mediante le quali viene eseguita una operazione di creazione, è possibile creare tre tipologie di elementi definiti nodi, cluster ed archi, con proprietà diverse a seconda della propria rappresentazione. Si ricorda inoltre che la visualizzazione è esclusiva per i dati che sono conosciuti o sono stati importati e non è possibile avere una visualizzazione senza la presenza di un dato. Come sarà visto meglio di seguito è possibile visualizzare i dati in più

viste diverse essendo un grafo clusterizzato una composizione di un grafo e di un albero di inclusione.

Per la rappresentazione di un **cluster** si deve tener conto di diversi fattori. Ogni cluster è un nodo interno dell'albero di inclusione ed avrà dunque conoscenza del nodo che lo precede nel cammino dalla radice al cluster stesso e dei nodi figli. Inoltre dovrà tener conto dei nodi del grafo sottostante che possiede, in quanto ogni nodo è sempre creato all'interno di un cluster. Optando per una visualizzazione a grafo, la rappresentazione di un cluster dunque dovrà tener conto anche del proprio livello e del numero di elementi posizionati al suo interno. Nella rappresentazione a grafo è possibile immaginare il grafo clusterizzato come visto dall'alto. In questo modo la radice dell'albero di inclusione è rappresentato dal piano di lavoro stesso su cui si basa la visualizzazione. La creazione di un oggetto cluster nella struttura dati conseguirà la sua aggiunta all'interno del disegno del grafo clusterizzato come una rappresentazione di un elemento inizialmente vuoto. Un cluster potrà poi contenere due tipi di elementi: altri cluster o nodi. Andando ad aumentare il numero di oggetti all'interno di un cluster sarà necessario anche aumentare il raggio di quell'oggetto e di tutti i cluster che fanno parte all'interno dell'albero di inclusione del cammino che va dalla radice a cluster selezionato. Altri elementi importanti nella visualizzazione a grafo per quanto concerne i cluster sono rappresentati dall'uso del colore e il raggio. Il primo deve necessariamente variare per ogni cluster definendo cosi una ulteriore variabile che ne definisce l'unicità e la distinzione. Il raggio dei cluster r_c è dipendente dal numero di oggetti al suo interno. È possibile definire questo raggio come:

$$r_c = k_c * (f + 1) + 2 * n$$

$$\forall n >= 0$$

$$\forall f >= 0$$

con k_c definito come un valore scelto come raggio di un cluster creato vuoto ,n come il numero di nodi all'interno del cluster e f come il numero di figli del cluster.

Ogni volta che viene creato un sotto-cluster all'interno di un determinato nodo dell'albero di inclusione, come detto per la visualizzazione risulta fondamentale il raggio, mentre per la struttura dati alla base il livello del cluster l_c. Ogni volta che si crea un sotto-cluster a in un cluster b il suo livello sarà definito come $l_a = l_b + 1$. Un esempio è mostrato nella Figura 5.1

Per quanto riguarda gli archi ed i nodi non vi è differenza nella visualizzazione con quella mostrata per i grafi. Discorso diverso invece riguarda la struttura dati alla base. Per poter dare un ordine all'interno del grafo clusterizzato la creazione di un nodo mediante una interazione deve prevedere anche a conoscenza da parte del nodo del cluster di appartenenza eseguendo quindi anche una operazione di connessione tra i dati di diverso tipo quali in questo caso nodi e cluster di appartenenza. Ogni volta che viene creato un oggetto utilizzando una visualizzazione

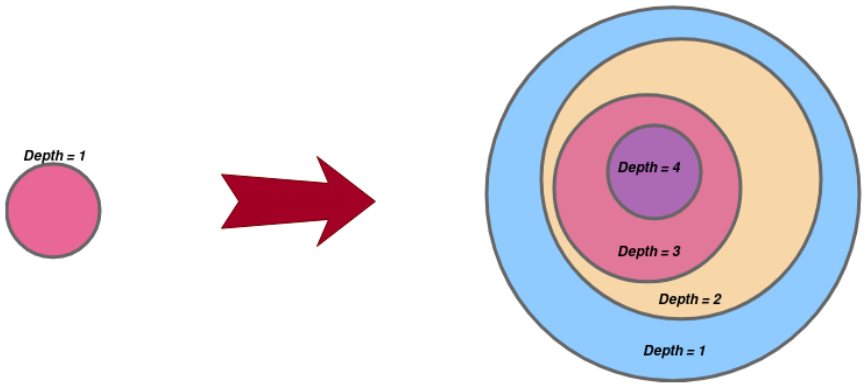

Figura 5.1: Esempio di variazione degli attributi di un cluster

a grafo con modello force directed è necessario un ricalcolo delle forze per ritrovare una posizione di equilibrio che porta ad una visualizzazione gradevole degli oggetti. Per concludere le primitive di interazione relative alla creazione della visualizzazione di un oggetto del grafo clusterizzato, è di notevole importanza l'ordine con cui queste interazioni vengono eseguite che porta a definire le attenzioni riportate di seguito:

- non è possibile creare un arco senza la presenza di due nodi;
- non è possibile creare un nodo senza la presenza di un cluster;
- non è possibile creare un sotto-cluster senza la presenza di un cluster direttamente collegato alla radice dell'albero di inclusione;

5.2 Rimozione di un oggetto

È necessario definire primitive di interazione che portano ad operazioni di oggetti. In questo caso eliminando un oggetto dalla visualizzazione esso verrà eliminato anche dalla struttura dati alla base. Essendo un grafo clusterizzato come visto caratterizzato da tre tipologie di oggetti ognuna di esse avrà delle specifiche caratteristiche. Avendo detto poi che ogni elemento del grafo clusterizzato deve avere un interconnessione con ciò che lo circonda, la cancellazione di un oggetto e della sua conseguente visualizzazione porterà a modifiche anche degli elementi ad esso collegati. La rimozione mediante una interazione di un arco porterà alla cancellazione dell'arco stesso dalla struttura dati e sarà eliminato anche dalla lista degli archi che i nodi, che in precedenza collegava, possiedono. La rimozione di un nodo causerà:

- la rimozione della sua visualizzazione;

- la rimozione di tutti gli archi connessi a quel nodo e la loro visualizzazione;
- l'eliminazione dalla lista dei nodi del cluster di appartenenza;
- nel caso in cui il cluster di appartenenza risultasse vuoto dopo l'eliminazione, dovrà essere eliminato anche suddetto poiché la rappresentazione di un cluster sena nessun oggetto non ha necessità di esistere.

Infine la rimozione di un cluster causerà:

- la rimozione della sua visualizzazione;
- la cancellazione di tutti i nodi appartenenti a quel cluster e di conseguenza di tutti gli archi che intersecavano tali nodi e la loro visualizzazione;
- nel caso in cui il cluster direttamente connesso nel cammino dalla radice al cluster eliminato risultasse vuoto dopo l'eliminazione, dovrà essere eliminato anche suddetto;
- dovrà essere eseguito un ricalcolo del raggio di tutti i cluster nel cammino dalla radice al cluster cancellato in quanto il raggio è dipendente dal numero di oggetti al suo interno.

Ogni qualvolta venga eliminato un oggetto inoltre, nella visualizzazione a grafo con il modello force-directed, si dovranno ricalcolare le forze in gioco per poter trovare nuovamente la posizione di equilibro.

5.3 modifica di un oggetto o dell'intero grafo

Si riportano le operazioni di modifica che possono essere apportate ad un grafo clusterizzato e che saranno analizzate di seguito:

- Encode
- Riconfigurazione
- Spostamento degli oggetti

Per quanto concerne lo zooming ed il panning della visualizzazione esse risultano essere uguali a quelle riportate nel capitolo due nella sezione relativa alle primitive di integrazione, in quanto eseguendo queste operazioni in quanto è ininfluente la tipologia di oggetto rappresentato.

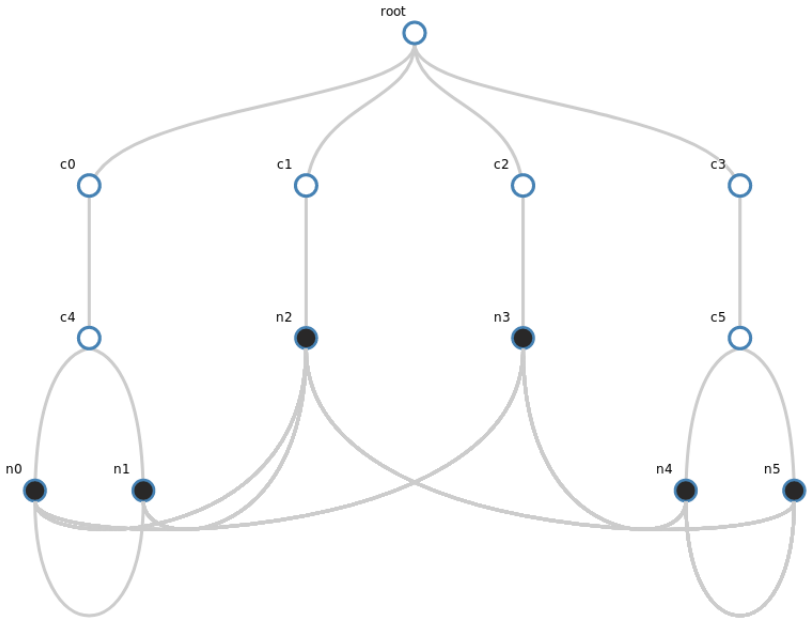

Figura 5.2: esempio di grafo clusterizzato nella visualizzazione ad albero

5.3.1 Encode

Per quanto concerne l'**encode** dei dati,essendo un grafo clusterizzato C definito come una coppia $< G, T >$ è possibile eseguire mediante una interazione l'operazione di trasformazione di una visualizzazione. Avendo dei dati è possibile passare da

una rappresentazione a grafo ad una ad albero e viceversa avendo cosi diverse possibilità di analisi e tipologie di interazione applicabili. Un esempio della visualizzazione ad albero di un grafo clusterizzato è mostrato nella figura Figura 5.2.Adottare questo tipo di visualizzazione è molto utile per una visione più chiara della distinzione tra l'albero di inclusione e del grafo sottostante ma la visualizzazione ad albero rende più chiaro il rapporto di inclusione e la distinzione degli archi e dei nodi facendo uso delle coordinate del piano $< x, y >$. Nella visualizzazione ad albero sono inoltre necessarie meno variabili di quelle descritte sopra e riguardanti la visualizzazione a grafo in quanto il raggio degli oggetti è fisso e non è necessario impiegare colori diversi per la distinzione dei vari cluster in quanto è possibile eseguire uno zoom semantico mediante l'utilizzo di etichette che caratterizzano gli elementi del grafo clusterizzato ed in quanto si andrebbe contro il principio per la visualizzazione secondo cui è controproducente utilizzare più variabili di quelle di cui si ha necessità. Inoltre una interfaccia a colori è buona quanto la stessa interfaccia rappresentata in bianco e nero è ben leggibile. Il colore è dunque necessario solo nel caso in cui si ha bisogno di una codifica aggiuntiva di una variabile, come era nel caso della visualizzazione a grafo per i cluster. Per quanto riguarda i nodi essi possono essere visualizzati con un colore diverso da quello dei cluster.

5.3.2 Riconfigurazione

Mediante la **riconfigurazione** è possibile riconfigurare una o più caratteristiche associate alla visualizzazione. Utilizzando una rappresentazione a grafo sarà dunque possibile il raggio degli oggetti visualizzati che siano essi nodi o cluster, la palette di colore o aggiungere una descrizione agli oggetti. Si noti che la riconfigurazione può agire sulla visualizzazione dei dati senza modificarne la

struttura o cambiare una caratteristica, quale ad esempio il raggio, di cui i dati ne hanno conoscenza. Ogni volta che viene eseguita una riconfigurazione si ha un ridisegno della visualizzazione che spesso porta ad un sostanziale cambiamento della pura rappresentazione avendo così la possibilità di avere molteplici interfacce differenziate per caratteristiche che comunque seguono lo stesso modello scelto e sono collegate agli stessi dati.

5.3.3 traslazione

Per quanto riguarda l'operazione di **spostamento** degli oggetti, al contrario che nei grafi in cui uno spostamento sul piano della visualizzazione di un nodo deve esclusivamente portare ad uno conseguente di tutti gli archi collegati, nei grafi clusterizzati per lo spostamento di un oggetto è necessario tener conto del mondo circostante avendo quindi due possibili metodi di esecuzione:

- impedire la traslazione oltre l'oggetto di appartenenza dando proprietà di solidità ai bordi degli oggetti. Nel caso dei nodi quindi non si potrà traslare oltre il cluster di appartenenza come nel caso del cluster con profondità maggiore di 1 che avrà anche il vincolo dei nodi al suo interno. Questa soluzione limita molto l'interazione in quanto diventa subordinata ai vincoli dati dalla dimensione degli oggetti.

- spostare tutti gli elementi collegati a quel nodo in maniera automatica o dovendo eseguire più interazioni. Nel primo caso traslando un oggetto e facendo uso delle connessioni tra i dati si ha la possibilità di muovere tutto ciò che a lui è collegato in una unica interazione. Nel secondo caso risulta essere un processo più lento che porta ad eseguire tante interazioni quanti sono gli oggetti connessi all'oggetto da traslare

Ad ogni interazione si dovrà poi ricalcolare il disegno del grafo clusterizzato.Tutte le primitive analizzate danno vita ad uno standard di visualizzazione dei grafi clusterizzati e sono state poi realmente utilizzate per la creazione dell'editor di grafi riportato di seguito per test di analisi delle stesse.

Capitolo 6

Operazioni di semplificazione per grafi clusterizzati

Avendo analizzato nel capitolo precedente le primitive di interazione per la visualizzazione di un grafo clusterizzato ora si tratterà delle operazioni di semplificazione, frutto di anni di ricerche([AFP10], [CP18], [eMP18], [AL16], [ADL16], [DBDM02]). Data la complessità nel definire se in tempo polinomiale è possibile dato un grafo clusterizzato costruire un disegno planare clusterizzato, si sono realizzate delle riduzioni polinomiali che semplificano il problema.

6.1 Riduzione polinomiale

Prima di poter analizzare i progressi svolti nell'ambito della semplificazione dei grafi clusterizzati è necessario fornire la definizione di riduzione polinomiale poiché sarà poi ampiamente utilizzata nel capitolo essendo le due semplificazioni riportate di seguito frutto appunto di riduzioni polinomiali. La **Riduzione polinomiale** tra problemi è definibile come segue:

*Un problema **P1** si riduce in tempo polinomiale a un problema **P2**, in formule* $P1 <=_p P2$*, se esiste una funzione* f *calcolabile in tempo polinomiale tale che* X *è una soluzione di* $P1$ ***se e solo se*** $f(X)$ *è una soluzione di* $P2$

In altre parole un problema **A** si riduce polinomialmente ad un problema **B** se, data un'istanza a di A, è possibile costruire in tempo polinomiale un'istanza b del problema B tale che a è affermativa se e solo se b è affermativa.Si noti inoltre che se un problema A si riduce ad un problema B allora risolvendo efficientemente B siamo in grado di risolvere anche A. Avendo dato le definizioni di classi di complessità P e NP , un problema A è definibile **NP-completo** se

$$A \in NPeB <=_p P \forall B \in NP$$

6.2 Flat clustered-planarity

Mediante il lavoro svolto dal professor Maurizio Patrignani nel 2018 si visto che tramite una riduzione polinomiale definibile come:

$$C\_planarity <=_p Flat\ C\_planarity$$

il *c-graph* C definito come $C(G,T)$ può essere ridotto in una istanza **equivalente** $C_f(G_f,T_F)$ in cui T_f è flat. Definendo che una istanza $C(G,T)$ di un c-graph con n vertici e c cluster può essere ridotta in tempo $O(n+c)$ in una istanza equivalente in cui Tè omogeneo, la $r(T)$ definita come la radice dell'albero ha almeno due figli e $h(T) <= n-1$, è possibile ottenere in un tempo $O(n+c)$ che T è omogeneo e $S(T) \in O(n)$. Come visto dalla definizione di riduzione polinomiale risolvendo questo problema definito **Flat Clustered planarity** allora si risolverebbe anche il problema Clustered Planarity essendo polinomialmente riducibile al primo.

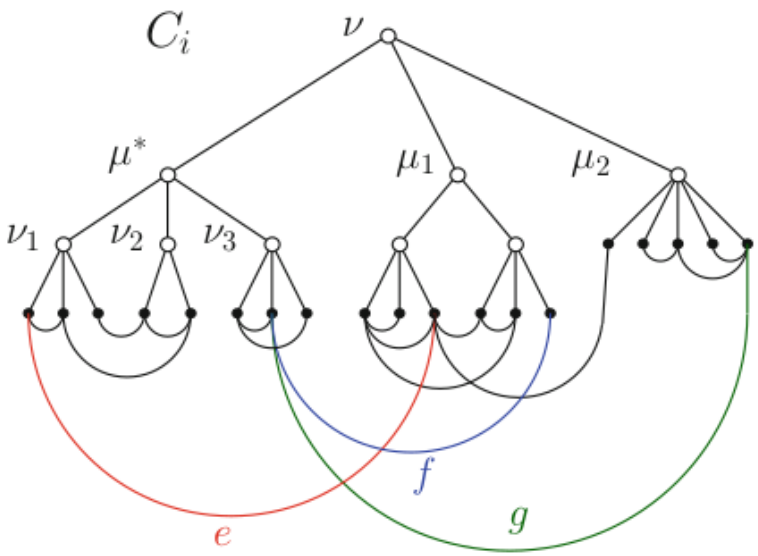

Figura 6.1: C_i con albero di inclusione T omogeneo

La riduzione consiste in una sequenza di trasformazioni di $C = C_0$ in C_1, C_2,. . . , $C_{S(T)} = C_f$, dove:

$$C_i =< G_i, T_i >$$

$$i = 0, 1, ..., S(T)$$

in cui: C_i ha un albero di inclusione T_i omogeneo; ogni trasformazione richiede tempo polinomiale $O(n)$; $S(T)$ è definito come la dimensione dell'albero di inclusione T, ovvero il numero di nodi superiori di T diversi dalla radice.

Prendendo, come mostrato in figura Figura 6.1, un grafo clusterizzato C_i nella sua rappresentazione ad albero "layered" è possibile vedere come esso rispetti le condizioni imposte in quanto l'albero di inclusione T_i risulta omogeneo avendo definito in precedenza che un albero è omogeneo se e solo se tutti i suoi nodi sono

nodi omogenei ovvero se tutti i figli di quel nodo sono foglie o sono nodi interni. È possibile dunque costruire $C_{i+1} =< G_{i+1}, T_{i+1} >$ come segue:

- G_{i+1} è ottenuto introducendo $\forall e =< u, v >$ di μ^*, dove μ^* è un cluster ed e è un arco inter-cluster, ovvero un arco si connessione tra i due nodi u e v appartenenti a cluster diversi, due nuovi vertici definiti e_χ ed e_φ e sostituendo e con un percorso

$$(u, e_\chi)(e_\chi, e\varphi)(e_\varphi, v)$$

- T_{i+1} è ottenuto rimuovendo μ^*, attaccando i suoi figli $v_1, v_2, ..., v_h$ direttamente al cluster radice v e aggiungendo ad esso due nuovi figli χ e φ, i quali conterranno tutti i vertici introdotti nella sostituzione di un arco inter-cluster di μ^* con un sentiero.

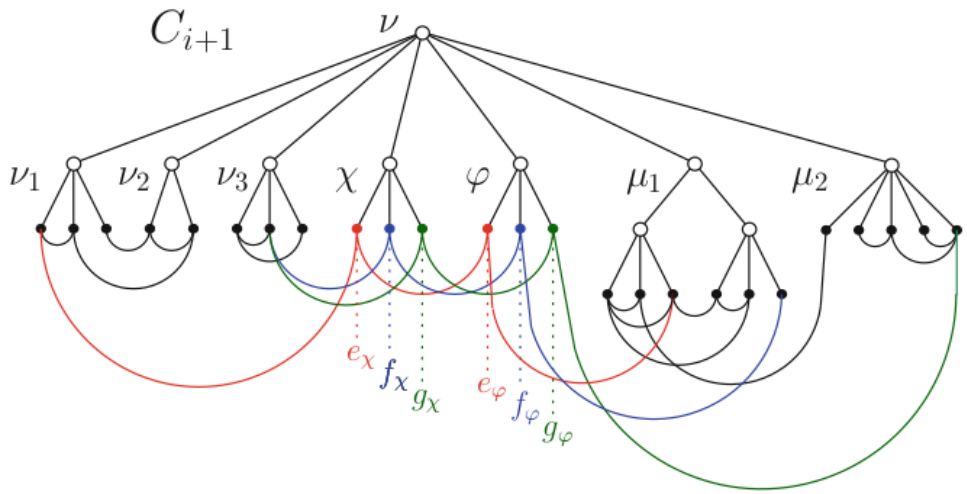

Figura 6.2: C_{i+1} ottenuto dopo parte del processo di riduzione

Ottenendo in questo modo la rappresentazione mostrata nella figura Figura 6.2.

Iterando la costruzione $\forall i$ si arriva ad avere un albero di inclusione T di profondità $d = 2$ in cui ogni cluster c è un nodo interno dell'albero ed ogni foglia è rappresentata da un nodo dell'underlying graph, ovvero T flat e nel grafo non si è persa conoscenza o connessione tra i nodi che risulteranno essere connessi comunque tra loro.

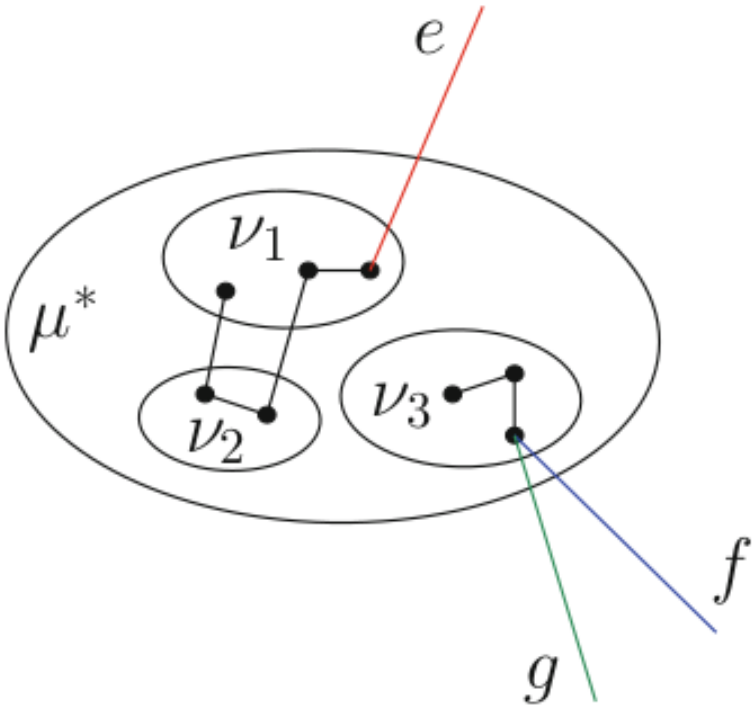

Figura 6.3: Dettaglio del cluster μ^* con rappresentazione Spring-enbedding

È inoltre possibile visualizzare la riduzione senza dover necessariamente utilizzare l'albero di inclusione. Partendo da una rappresentazione node-link, utilizzando il modello Force-Directed, del grafo cluterizzato ed in particolare soffermandosi solamente sul dettaglio del cluster μ^* di livello $l > 2$, visualizzato in Figura 6.3, è possibile comunque eseguire la riduzione in quanto indipendente dal tipo di

visualizzazione. Seguendo i passi visti prima per la costruzione del grafo clusterizzato $C_{i+1} =< G_{i+1}, T_{i+1} >$, con un modello Node-link si otterrà una visualizzazione simile a quella mostrata nella Figura 6.4. Come si nota i cluster aggiunti χ e φ, che vanno a sostituire l'originale μ^*, hanno una forma non più circolare ma ad anello. Questi cluster χ e φ per permettere una maggiore copertura per quanto riguarda i possibili nodi e_χ ed e_φ circonderanno i cluster che in precedenza appartenevano alla lista dei figli di μ^*. Questo fa in modo di avere una visualizzazione a grafo non più puramente node-link ma è comunque possibile applicate le forze in gioco per poter raggiungere un equilibrio nel caso di un modello force-directed. Gli anelli dei cluster χ e φ sono solo una delle possibili visualizzazioni in quanto essi possono anche esser rappresentati allo stesso modo dei cluster.

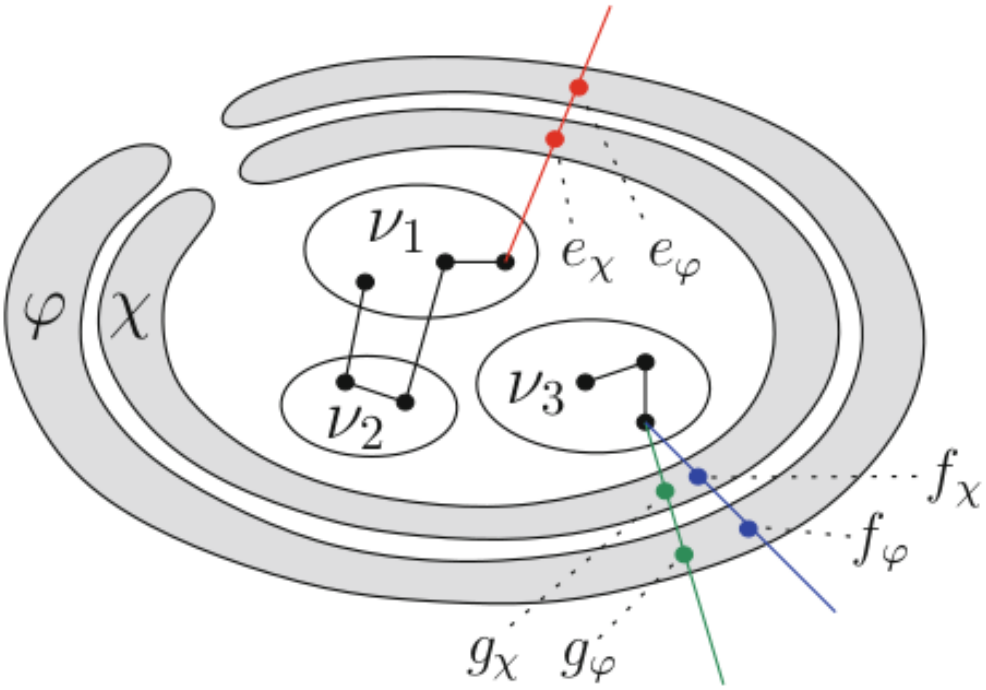

Figura 6.4: Cluster μ^* dopo la riduzione con rappresentazione semi Spring-enbedding

Per completezza si riportano le caratteristiche saltate in questa sezione fino ad ora che rappresentano ovvietà non specificate:

- Se l'albero di inclusione T_i è omogeneo, allora T_{i+1} è omogeneo, in quanto ogni nodo interno non avrà comunque nodi cluster e non al suo interno dopo la riduzione.
- $S(T_{i+1}) = S(T_i) - 1$, ricordando che $S(T)$ è la dimenzione dell'albero T
- il grafo clusterizzato $C_f = C_{S(T)}$ è flat.

Per concludere si può dunque denotare che

$C_i(G_i, T_i)$ è un disegno planare **c-planar** di un grafo clusterizzato se e solo se anche $C_{i+1}(G_{i+1}, T_{i+1})$ è un disegno planare

6.3 Pipe clustered planarity

È possibile la realizzazione anche della riduzione in disegno planare Pipe che sarà introdotta di seguito e la cui composizione, partendo da un disegno planare Flat fino alla costruzione del disegno c-planare Pipe, è mostrata nella Figura 6.5.

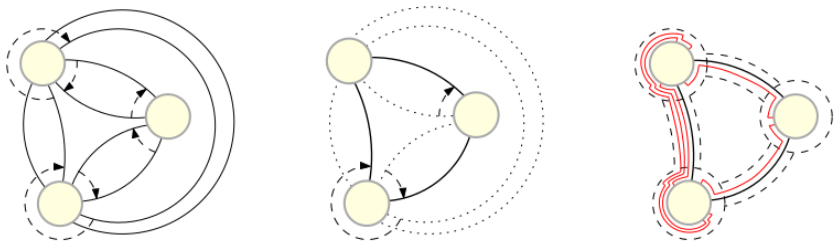

Figura 6.5: creazione di un disegno c-planare Pipe partendo da un disegno c-planare Flat

Avendo già definito i concetti di grafo clusterizzato, disegno planare, riduzione polinomiale e tutto ciò che ne concerne definiamo adesso il concetto di pipe su cui si basa la riduzione. Sia $\Gamma(C)$ un disegno c-planare di un c-grafico C. Per ogni cluster μ_i di C, il disegno Γ induce un ordine $O_{orario}(\mu_i)$ in senso orario dei bordi tra cluster di μ_i, ovvero l'ordine circolare in cui si incontrano i bordi tra i cluster di μ_i mentre si attraversa in senso orario il limite di $R(\mu_i)$ in Γ. Allo stesso modo, definiamo l'ordinamento antiorario $O_{antiorario}(\mu_i)$ dei bordi tra cluster di μ_i. Un disegno c-planare Pipe $\Gamma_p(C)$ di un grafo flat $C(G,T)$ è un disegno c-planare di C tale che, per qualsiasi coppia di cluster vicini μ_i e μ_j, esiste un semplice regione chiusa $R(\mu_i,\mu_j)$, detta Pipe, che tocca $R(\mu_i)$ e $R(\mu_j)$ e che contiene esclusivamente la porzione dei bordi inter-cluster in $E(\mu_i,\mu_j)$ all'esterno di $R(\mu_i)$ e $R(\mu_j)$.
Viene di seguito data la condizione necessaria affinché una istanza flat possa essere ridotta ad una pipe:

Il grafo clusterizzato C ammette un disegno c-planare $\tau(C)$ tale che per $\forall E(\mu_i,\mu_j)$ arco di C, i bordi di $E(\mu_i,\mu_j)$ compaiono consecutivamente nell'ordine orario $O_{orario}(\mu_i)$ nello stesso ordine in cui appaiono consecutivamente nell'ordine $O_{antiorario}(\mu_j)$.

Definita questa condizione è possibile eseguire una riduzione di una istanza pipe clustered planarity in una flat clustered planarity e viceversa.

Definita C_p una istanza di pipe clustered planarity e C_f una di flat clustered planarity è dunque possibile definire la riduzione Pipe C_planarity $<=_p$ Flat C_planarity e viceversa come segue:

Se $C_p = (G_p, T_p)$ *è un'istanza positiva di pipe clustered planarity allora* $C_f(G_f, T_f)$ *è un'istanza positiva di Flat clustered planarity*

La dimostrazione si basa sulla costruzione di un disegno c-planare $\tau(C_f)$ a partire da un disegno c-planare pipe $\tau(C_p)$ come mostrato nella Figura 6.6.

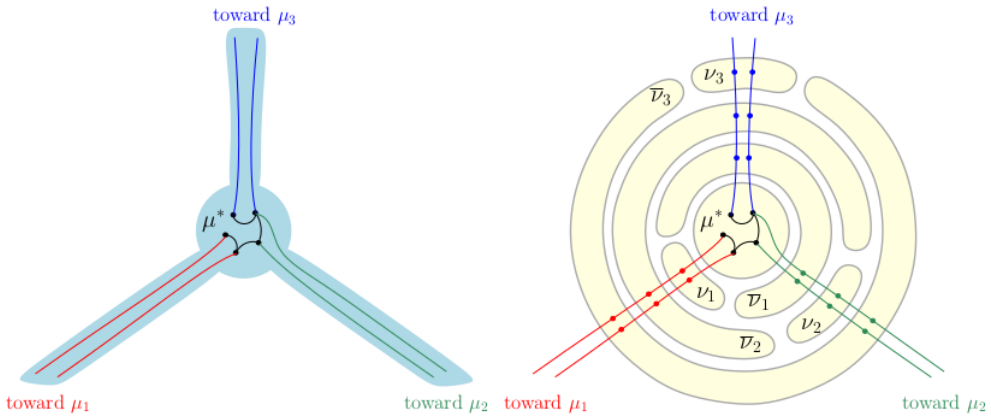

Figura 6.6: Dettaglio di un cluster in c-planare Pipe ed in c-planare Flat

Capitolo 7

Editor: interfacce e strutture dati

Partendo dalle primitive di interazione analizzate durante lo studio del capitolo 5 si passa ora a mostrare la realizzazione di un esempio di come queste possano essere impiegate in un caso reale. Tornando ai principi della visualizzazione ed in particolare ai suoi scopi, che come detto possono essere di analisi o di presentazione, per il software di esempio realizzato lo scopo è sia di presentazione che di analisi proprio per poter lasciare all'utente la possibilità di visualizzare dati senza modificarne i valori ma anche di effettuare modifiche ed analizzarne il contenuto, supportando la ricerca mediante l'automatismo delle operazioni

7.1 Strumenti e struttura dati

Avendo definito l'obiettivo prefissato per il software che si andrà a realizzare resta da discutere su quale delle possibili tecnologie utilizzare allo scopo della rappresentazione dinamica e user-friendly. Si è optato quindi per l'impiego di Javascript e della libreria D3.js che saranno viste nel dettaglio per motivazioni e punti di forza.

7.1.1 Javascript

JavaScript è un linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client. Essendo il sistema ideato pensato per essere un software web con molti input da parte dell'utente a cui far fronte, Javascript consente un'ottima gestione di questi eventi. Le caratteristiche principali sono la debole tipizzazione e l'essere un linguaggio interpretato e non compilato con una sintassi relativamente simile a quella Java.
Un altro punto a favore è la notevole facilità con cui si può lavorare mediante file con estensione .Json, che saranno impiegati proprio per l'import o l'export dei dati da rappresentare. Fondamentale è il supporto alla programmazione asincrona, cioè alla possibilità di eseguire attività in background che non interferiscono con il flusso di elaborazione principale, che per Javascript risulta essere quasi una necessità essendo un linguaggio Single-threaded. I due principali elementi che consentono di sfruttare il modello di programmazione asincrono in JavaScript sono gli eventi e le callback. Per completezza si vuol dare anche la definizione di programmazione asincrona come una forma di programmazione parallela che permette ad un'unità di lavoro di funzionare separatamente dal thread principale ed "avvertire" quando avrà finito l'esecuzione di quel task.
Si ricorda inoltre che un buon sistema deve essere documentato. Per questo è stata utilizzata la libreria Javascript **JsDoc**. JSDoc è un generatore di documentazione per JavaScript, simile a Javadoc o phpDocumentor. Mediante l'aggiunta di commenti di documentazione all'interno del codice sorgente prima di ogni funzione o altro, JSDoc eseguirà la scansione del codice sorgente e generando un sito web HTML di documentazione. Essendo un sistema predisposto per l'utilizzo di potenziali utenti esperti è necessario dunque facilitare il riconoscimento del codice

sorgente per modifiche e/o motivi di studio.

7.1.2 D3.js

A supporto della scelta dell'utilizzo del linguaggio di scripting javascript vi è l'impiego della libreria utilizzata per la visualizzazione delle informazioni vista di seguito. D3.js (o solo D3 per Data-Driven Documents) è una libreria JavaScript per creare visualizzazioni dinamiche ed interattive partendo da dati organizzati, visibili attraverso un comune browser. Per fare ciò si serve largamente degli standard web: SVG, HTML5, e CSS.

La libreria JavaScript D3, incorporata in una pagina web HTML, utilizza funzioni JavaScript per selezionare elementi del DOM, creare elementi SVG, aggiungergli uno stile grafico, oppure transizioni, effetti di movimento e/o tooltip. Questi oggetti possono essere largamente personalizzati utilizzando lo standard web CSS. In questo modo grandi collezioni di dati possono essere facilmente convertiti in oggetti SVG usando semplici funzioni di D3 e così generare ricche rappresentazioni grafiche di numeri, testi, mappe e diagrammi. Per completezza si ricorda che Scalable Vector Graphics abbreviato in SVG, indica una tecnologia in grado di visualizzare oggetti di grafica vettoriale e, pertanto, di gestire immagini scalabili dimensionalmente. Le figure espresse mediante SVG possono essere dinamiche e interattive. Il Document Object Model (DOM) per SVG, che include il completo XML DOM, consente una animazione in grafica vettoriale diretta ed efficiente attraverso il Javascript. D3 consente dunque di associare dati arbitrari a un Document Object Model (DOM) e quindi applicare trasformazioni basate sui dati al documento per avere una manipolazione efficiente dei documenti basata sui dati.

Per completezza si specifica inoltre che il Document Object Model (DOM) è

un'interfaccia multi-piattaforma indipendente dal linguaggio utilizzato in un documento XML o HTML. Possiede una struttura ad albero come mostrato nella **Figura 7.1** in cui ciascun nodo è un oggetto che rappresenta una parte del documento formando nell'insieme un albero logico. Ogni ramo dell'albero termina in un nodo e ogni nodo contiene oggetti. I metodi DOM consentono l'accesso programmatico all'albero; mediante questi si può cambiare la struttura, lo stile o il contenuto di un documento. Ai nodi possono essere associati gestori di eventi. Una volta attivato un evento, i gestori degli eventi vengono eseguiti. Un altro punto a favore dell'impiego di D3 e quindi di javascript riguarda le prestazioni.
D3 è estremamente veloce, supporta set di dati di grandi dimensioni e comportamenti dinamici per l'interazione e l'animazione.

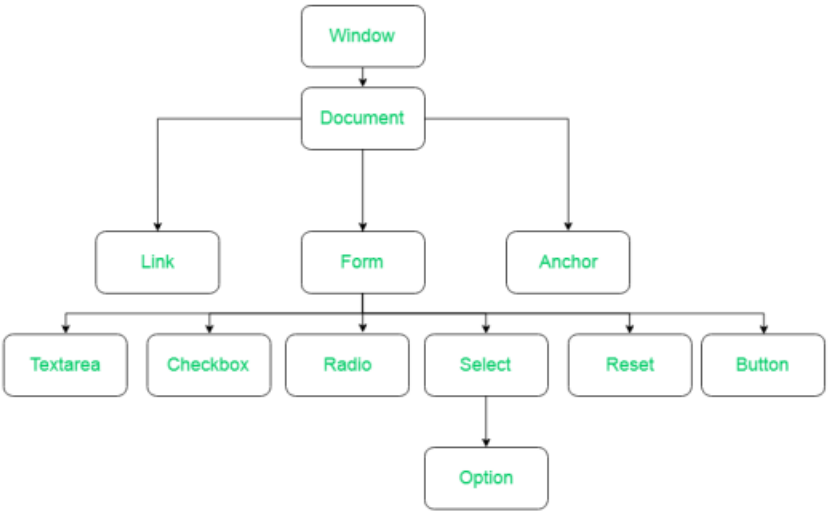

Figura 7.1: Rappresentazione Document Object Model

7.1.3 Oggetti di dominio

Come detto nel capitolo 4 la realizzazione è stata supportata dalla metolodologia agile, che ha portato ad avere più iterazioni riguardanti la fasi di analisi e l' ideazione degli oggetti di dominio che andranno a comporre il sistema. Di seguito in **Figura 7.2** è descritto in maniera molto semplificata il diagramma degli oggetti di dominio, risultato dall'analisi del progetto nella prima iterazione.

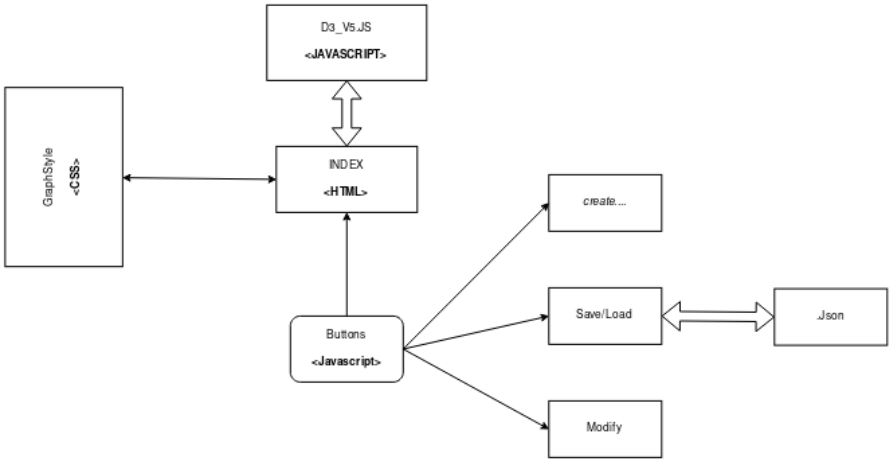

Figura 7.2: Prima iterazione degli oggetti di dominio

Come si nota la libreria D3.js è stata fin da subito pensata come punto centrale per le visualizzazioni. Per rendere la rappresentazione accessibile e utilizzabile in pieno si è scelto di realizzare tutte le primitive di interazione viste nel capitolo 5. Una volta definite le interazioni i problemi che sono stati affrontati con maggior interesse per quanto concerne l'editor in esempio riguardano due importanti fattori che si ritrovano i qualunque software grafico: la facilità di impiego e

l'ottenimento di una visualizzazione gradevole all'utente. Durante una successiva analisi si è poi optato per alcune modifiche progettuali che sono mostrate nel diagramma ad oggetti di seguito in Figura 7.3 seguendo i principi espressi dal libro "UML distilled. Guida rapida al linguaggio di modellazione standard"[Fow02].

Nell'ingegneria del software, **UML**, acronimo di Unified Modeling Language

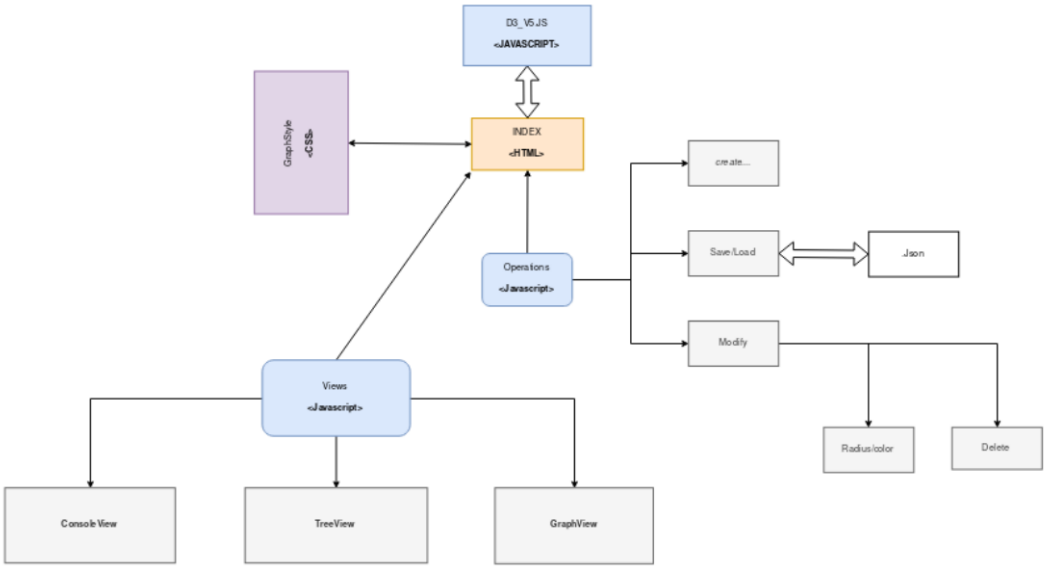

Figura 7.3: Seconda iterazione degli oggetti di dominio

ovvero di "linguaggio di modellizzazione unificato" è un linguaggio di specifica basato sul paradigma orientato agli oggetti. Il nucleo del linguaggio fu definito nel 1996 da Grady Booch, Jim Rumbaugh e Ivar Jacobson. Lo stantard è tuttora gestito dall'Object Management Group. UML svolge un'importantissima funzione di "lingua franca" nella comunità della progettazione e programmazione ad oggetti utilizzato anche dalla gran parte della letteratura del settore informatico per descrivere soluzioni analitiche e progettuali in modo sintetico e comprensibile ad un vasto pubblico.

7.1.4 Struttura dati

Per quanto concerne la struttura dati su cui si basa la visualizzazione si è optato per una struttura ad oggetti ed in particolare si è scelto di utilizzare le strutture dello standard ECMAScript 6 quali Map e Set risultando essere una scelta migliore rispetto all'utilizzo di array per ragioni prestazionali, di sicurezza e di miglior accesso ai dati da rappresentare, spesso senza bisogno di iterare completamente su tutti i valori indicizzati al loro interno. Inoltre risultano essere oggetti specializzati: il **Set** garantisce l'unicità dei valori mentre l'oggetto Map conserva l'ordine di inserimento. Per completezza si ricorda inoltre che un oggetto **Map** contiene coppie $< KEY, VALUE >$ in cui qualunque valore, sia esso oggetto o tipo primitivo, può esser usato come chiave e come valore. Ad avvalorare la scelta si nota che entrambi sono oggetti *iterable* quindi facili da iterare e hanno funzionamenti e prestazioni migliori in caso di frequenti aggiunte o rimozioni di valori. Prima ancora di mostrare l'interfaccia e le modalità di impiego delle primitive analizzate nel capitolo 5 è necessario definire con chiarezza la struttura dati con cui l'utente andrà a lavorare ricordando infatti che una visualizzazione per buona che sia non ha valore senza la possibilità di analisi del dato di cui ne è la rappresentazione. Come visto nel capitolo 3 un grafo clusterizzato C è definito come una coppia $< G, T >$ in cui G è l'underlying graph definito dalla coppia $< V, E >$ rispettivamente di nodi ed archi e T è l'inclusion tree. Avendo chiaro questa definizione vi è la rappresentazione del grafo clusterizzato così come mostrato nella figura Figura 7.4.

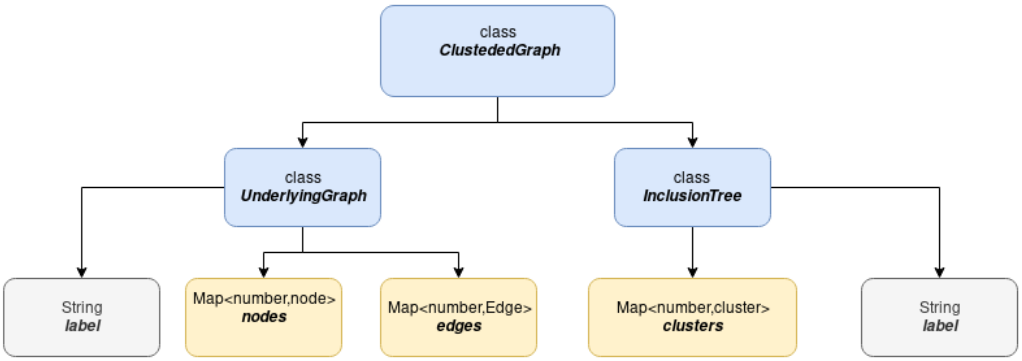

Figura 7.4: Oggetto clusteredGraph

Un *clusteredGraph* è definito mediante un oggetto relativo all'underlying graph ed uno relativo all'inclusion tree. Un *UnderlyingGraph* è definito mediante una primitiva String che ne rappresenta l'etichetta e mediante due oggetti Map: Map<number,Node>, in cui sono elencati tutti i nodi di cui è composto il grafo e Map<number,Edge>, per gli archi appartenenti al grafo clusterizzato. Andando poi a ritroso nelle definizioni, nella figura Figura 7.5 è mostrato l'oggetto Nodo. Esso può essere definito mediante un costruttore di tre valori:

- **label** di tipo primitivo String che ne definisce l'etichetta;
- **id** un numero, identificativo insieme alla sua etichetta ma ancor più personale, del nodo in questione;
- **rotationScheme** di tipo Set<number> in cui vengono salvati tutti gli ID degli archi che hanno quel nodo come nodo di partenza o di destinazione.

Ogni nodo è un elemento fondamentale dell'underlying graph da visualizzare e connettere. Restando sempre nella creazione dell'oggetto *UnderlyingGraph*

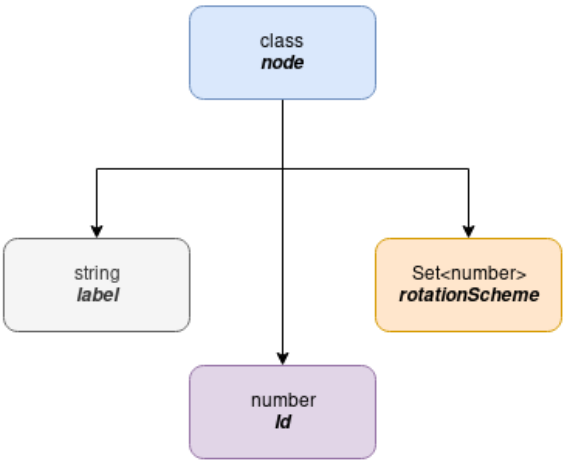

Figura 7.5: Oggetto nodo del underlying graph

del grafo clusterizzato risulta, esattamente come per l'oggetto *node*, necessario fornire la definizione dell'oggetto relativo agli archi. La classe dell'oggetto *Edge* rappresentata in Figura 7.6 è definita mediante una stringa e tre valori numerici. La prima si riferisce all'etichetta dell'arco e non necessariamente dovrà essere unica, i valori numerici invece fanno riferimento all'id dell'arco al nodo sorgente e al nodo di fine arco. L'id risulta necessario non solo per unificare l'arco creato ma anche per essere inserito nel Set<number> *rotationScheme* del nodo che lo vedrà collegato. Ogni arco collegherà poi un nodo di inizio fino ad un nodo di arrivo anche se non in maniera orientata, in quanto il nodo di inizio e il nodo di fine arco saranno solamente quelli relativi a dove l'utente vorrà che la visualizzazione dell'arco inizi e finisca. Per questo ogni elemento della classe arco avrà due valori numerici *source* che conterrà l'id del nodo di partenza e *target* con l'id del nodo di destinaione.

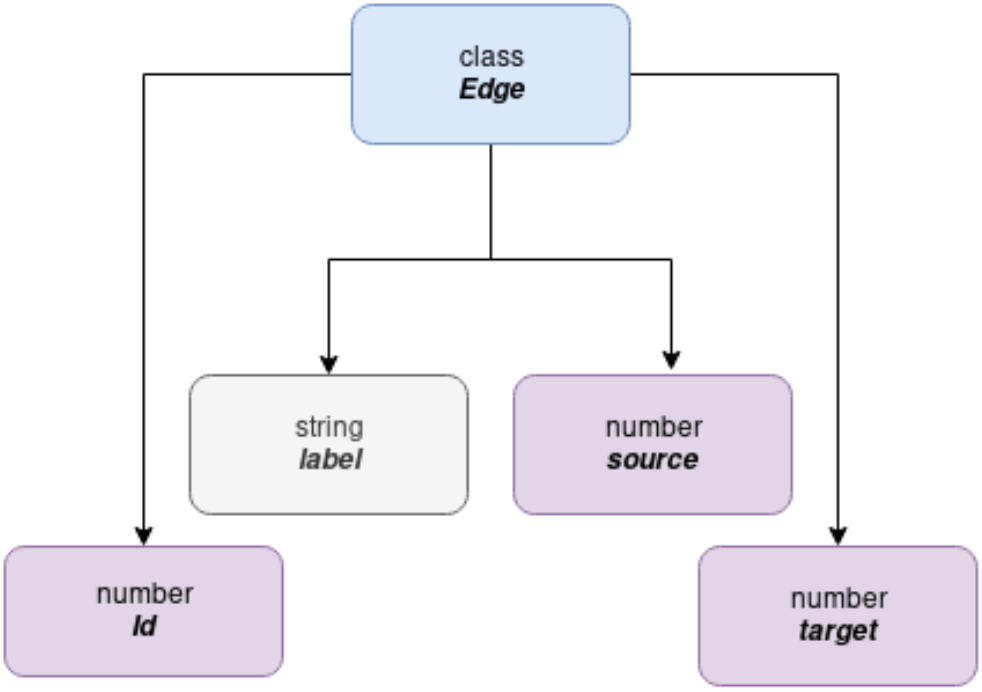

Figura 7.6: Oggetto arco dell'undelying graph

Termina la definizione della classe *UnderlyingGraph* si può passare a quella della classe *InclusionTree*. Ogni oggetto appartenente a questa classe possiede una etichetta definita come tipo String, esattamente come per la sua controparte grafo, e una lista contenente i cluster che fanno parte dell'albero di inclusione, di tipo Map<number,Cluster>. Andando a ritroso anche in questa struttura si vuol definire di seguito la classe *Cluster*. Un oggetto *cluster* come mostrato nella figura Figura 7.7 è definito mediante un costruttore con cinque attributi di seguito riportati:

- **label** di tipo String che ne rappresenta l'etichetta;

- **level** di tipo number, definisce il livello ovvero la profondità a cui il cluster si troverà all'interno dell'albero di inclusione;
- **cildren** di tipo Set<number> e che contiene gli id dei cluster di profondità superiore collegati con il cluster di interesse, ovvero i suoi figli;
- **parents** di tipo Set<number> contenente gli id dei cluster di profondità inferiore e che sono collegati al cluster di interesse, ovvero i suoi genitori;
- **nodes** di tipo Set<number> contenente gli id dei nodi che sono contenuti all'interno del cluster

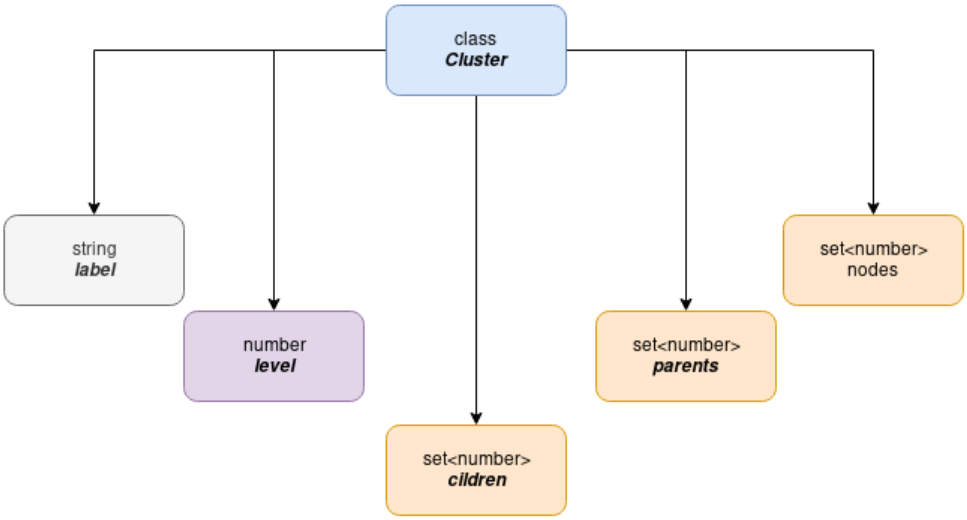

Figura 7.7: Oggetto cluster dell'inclusion Tree

7.2 Log-in ed interfacce

7.2.1 Inizializzazione

Al momento del log-in iniziale l'utente si troverà la schermata mostrata nella Figura 7.8. Come è possibile notare da subito sono stati utilizzati molti dei principi di visualizzazione visti in precedenza cominciando dall'impiego e della posizione dei bottoni necessari per le interazioni dell'utente. È stato lasciando inoltre grande spazio per quanto concerne il piano di lavoro principale che, come si vedrà successivamente, rimarrà inalterato anche cambiando l'interfaccia utilizzata. Anche l'impiego di colori caldi come il blu rispetto al piano di lavoro di colore chiaro lascia intendere la maggiore importanza ed evidenza che deve avere il secondo rispetto al resto dell'interfaccia. Ogni qualvolta l'utente vorrà interagire con il sistema sarà sufficiente cliccare una delle operazioni, che saranno viste nel capitolo successivo, per poter avere una risposta. Come definito nel capitolo

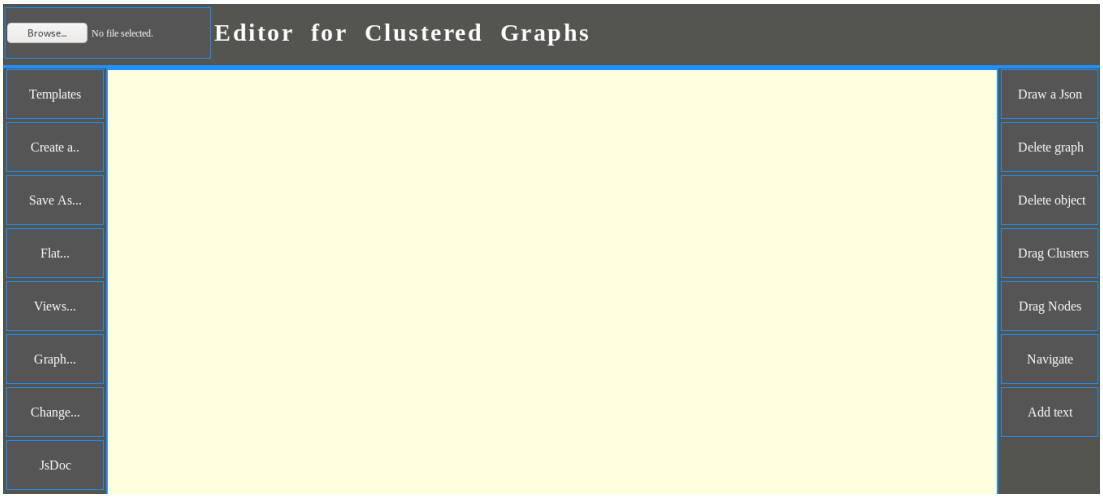

Figura 7.8: Log-in nel sistema

relativo alle primitive, è necessario poter dare all'utente la possibilità di eseguire

un encoding dei dati. Per questo ogni qualvolta verrà richiesto al sistema di eseguire il log-in iniziale oppure un encode della visualizzazione il sistema risponderà con una inizializzazione. Durante la prima inzializzazione in cui sarà creata la classe *clusteredGraph* e tutto ciò che ne comprende, sarà generato l'svg #cgraph su cui si basa il piano di lavoro e con cui l'utente andrà ad interagire ed analizzare. Tutte le operazioni e le visualizzazioni saranno eseguite proprio su questo svg creato con l'ausilio della libreria grafica **D3.js**.
Generato l'svg principale, sempre durante l'inizializzazione del log-in, verrà collegato ad altri tre oggetti svg di seguito elencati e che lo compongono:

- #**c_cluster** che andrà a rappresentare la visualizzazione di tutti i cluster;
- #**c_node** ovvero l'svg che rappresenta l'insieme dei nodi da visualizzare;
- #**c_edge** che rappresenterà gli archi che collegano i nodi dell'underlying graph.

La divisione degli svg che andranno a comporre quello relativo al piano di lavoro è necessaria per molteplici fattori, primo fra tutti il fatto che poter scindere una entità monolitica in molteplici oggetti è uno dei traguardi dello sviluppo software. Inoltre è possibile in questo modo poter trattare, come sarà analizzato meglio in seguito, le tre entità che rappresentano la classe *ClusteredGraph* come oggetti a se stanti.
Una volta che l'utente ha avuto accesso al sistema ed è stato introdotto il concetto di svg e di piano di lavoro, è possibile cominciare a definire quello di interfacce e dei diversi piani di lavoro. Essendo basato su una coppia $< G, T >$ che compongono il grafo clusterizzato C si è deciso di implementare la funzione di encode dando all'utente la possibilità di scelta tra due viste separate riportate nelle sezioni

seguenti. Prima di passare all'analisi di queste rappresentazioni è bene anticipare la struttura dell'algoritmo, mostrato nella Figura 7.9, che il sistema andrà ad eseguire ogni volta l'utente chiederà mediante una interazione l'operazione di encode dei dati. Ad ogni richiesta sopra descritta il sistema confronterà la vista su cui attualmente l'utente lavora con quella richiesta ritornando nel caso in cui le due viste siano uguali ed eliminando la vista visualizzata e creano quella richiesta nel caso contrario.

La strategia scelta dell'eliminare e ricreare un oggetto si sposa bene con i principi per la visualizzazione visti in precedenza. In questo modo l'utente andrà ad interfacciarsi con una unica visualizzazione alla volta ed il piano di lavoro resterà libero evitando ogni volta di dimezzare la grandezza dello stesso per far posto a visualizzazioni che non sono funzionali alle trasformazioni ma che hanno esclusivamente scopi di analisi.

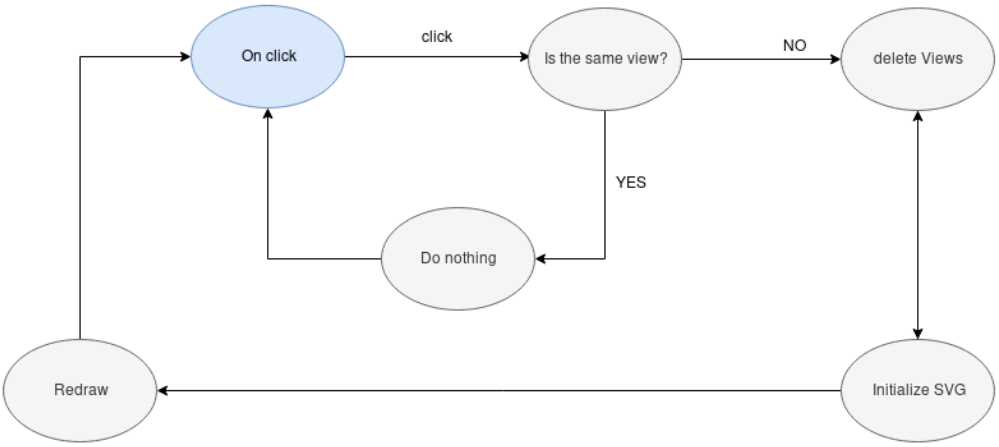

Figura 7.9: Struttura dell'algoritmo dell'operazione di encode

7.2.2 Graph-view

La visualizzazione a grafo, anche definita graph-view è la visualizzazione di default ed è anche quella con cui l'utente si interfaccerà maggiormente. La graph-view è infatti l'unica vista con cui l'utente può avere una interazione di grado tre, ovvero il grado massimo, in cui può interagire sulle modifiche e le trasformazioni non solo riguardo la visualizzazione ma anche sui dati.

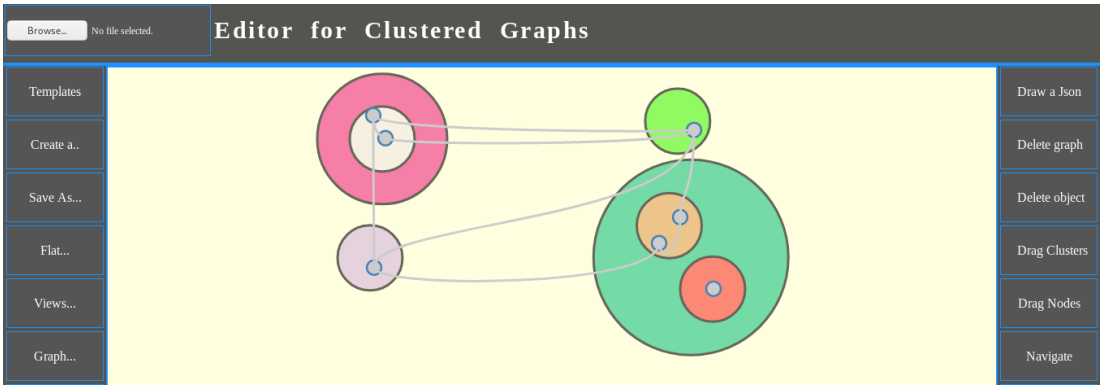

Figura 7.10: Graph View

Come si può notare dalla Figura 7.10 il modello di visualizzazione riprende quello Node-link in cui i "Node" sono differenziati in due elementi, rappresentando non solo i nodi del grafo ma anche i cluster dell'albero di inclusione. La differenza tra queste due entità a livello di visualizzazione può essere vista mediante due diversi fattori: il colore ed il raggio. Per quanto riguarda il colore dei nodi risulta essere fisso in quanto ogni nodo rappresenterà sempre lo stesso elemento che cambierà per quanto riguarda la posizione o il cluster di appartenenza ma non nella visualizzazione. Anche il raggio del nodo, che verrà definito r_n, ha un valore fisso non dipendente da nessuna variabile. Per quanto riguarda i cluster invece il colore, che di default presenta una casualità nella propria visualizzazione, varia

per ogni cluster definendo una ulteriore variabile che ne definisce l'unicità. Il raggio dei cluster definito r_c, al contrario di quello dei nodi che come abbiamo visto è fisso esso è dipendente da variabili come visto nel capitolo 5.

7.2.3 Tree-view

Non avendo un limite al numero di nodi e di cluster visualizzabili la graph view può portare ad una eccessivo disordine dato dalla quantità. L'occhio umano vede una rappresentazione organizzata in maniera notevolmente migliore rispetto ad una piatta e priva di profondità. Per questo l'utente in qualunque momento durante una sessione di lavoro potrà interagire con il sistema e passare da una visualizzazione a grafo ad una dell'inclusion tree in cui però saranno rappresentati anche i nodi e gli archi dell'underlying graph ovvero ad una visualizzazione ad albero.

Ogni volta che si chiederà questa operazione, il sistema risponderà eliminando la visualizzazione e inizializzandone una nuova con un encode sugli stessi dati rappresentati in precedenza. Questa visualizzazione sarà un svg detto #_ctree che possiederà grandezza uguale a quella dell' #cgraph ma senza una divisione degli elementi che andranno a comporre la rappresentazione, come era invece necessario per la visualizzazione a grafo.

Questa rappresentazione risulterà comunque essere un modello node-link ma, al contrario che nella graph view, in cui non si avrà necessità di inserire forze per una visualizzazione ottimale essendo una semplice rappresentazione "layered" con l'aggiunta di archi che collegano le foglie tra loro. Nella Figura 7.11 è mostrato lo stesso grafo visualizzato in precedenza nella graph-view e visto nella figura Figura 7.10. Nella visualizzazione ad albero il raggio dei cluster e dei nodi risulta essere la stesso anche se ciò che caratterizza una foglia, che sia essa un cluster

vuoto oppure un nodo all'interno di un cluster, è il diverso colore diverso rispetto ai nodi interni. Essendo un sistema in cui non vi è una soglia massima del numero di nodi è di cluster definibili o importabili si possono presentare situazioni limite. Per questo sono state implementate due diverse tipologie di tree-view, entrambe discendenti: ad albero verticale ed orizzontale. All'utente è lasciata la scelta di quale delle due utilizzare. È consigliabile ai fini di una buona visualizzazione l'utilizzo di una rappresentazione verticale nel caso in cui si hanno tanti cluster sullo stesso livello ma la profondità dell'albero sia relativamente bassa mentre è preferibile utilizzare una visualizzazione ad albero orizzontale, con la radice centrale a sinistra dell'svg, nel caso in cui si hanno pochi cluster per ogni livello ma si abbia una profondità maggiore da gestire.

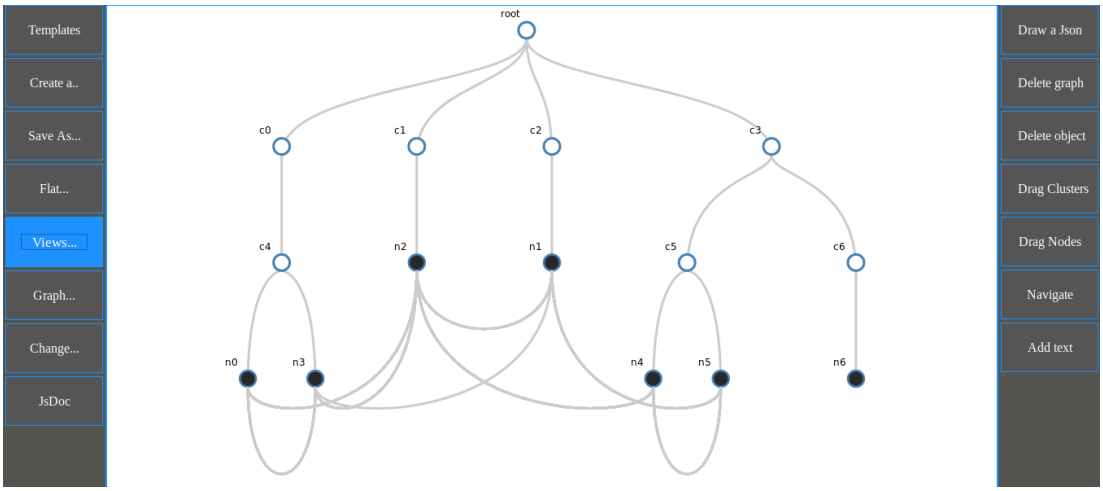

Figura 7.11: tree View

7.2.4 Console-view

Oltre all'operazione di encode vista si ha la necessità di dover mostrare, sotto richiesta dell'utente, dei log riguardanti i cambiamenti eseguiti durante una stessa

sessione di lavoro. Per questo è stata introdotta una terza vista definita *console-view* che risolve questa problematica. Mediante una interazione similare a quelle viste in precedenza, l'utente può accedere a questa visualizzazione. Il sistema risponderà eliminando il precedente svg che sia esso #cgraph oppure #ctree e creerà un nuovo svg di dimensioni uguali ed in cui saranno inseriti i messaggi lasciati all'utente delle ultime n operazioni eseguite. Ogni operazione infatti, nel momento in cui viene richiesta dall'utente, mentre viene eseguita salva un valore di tipo String all'interno di un oggetto Map<number,String> *logs* inzializzato al log-in all'inizio della sessione di lavoro dell'utente. La vista sarà similare a quella mostrata nella Figura 7.12, nella quale oltre al messaggio che ricorda l'operazione richiesta ed eseguita, sarà visualizzato l'orario in cui l'utente ha interagito con il sistema. Terminate le viste e delucidato il lettore sulle operazioni di encode eseguibili si passa adesso a definire con più chiarezza le operazioni di filtraggio

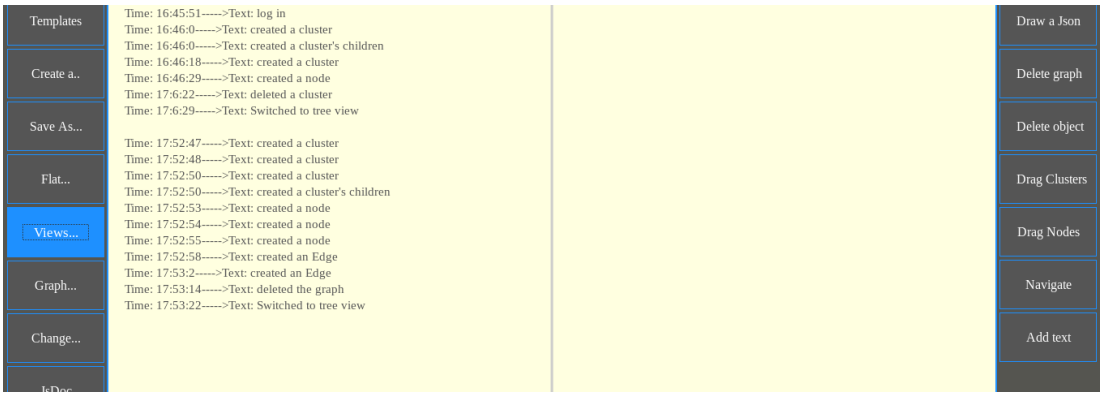

Figura 7.12: Graph View

7.2.5 Filtraggio degli oggetti

Una operazione di filtraggio permette di mostrare un sottoinsieme di dati secondo certe regole, ovvero ciò che può esser definito come uno zoom semantico. Si immagini un grafo clusterizzato di dimensioni notevoli o comunque con una delle classi di oggetti che lo compongono che, in un momento particolare, abbia un rilievo maggiore rispetto agli altri. Volendo lavorare solamente su un particolare oggetto è possibile stando nella *graph-view* eseguire una operazione di filtraggio su una delle tre componenti come mostrato nella Figura 7.13 che fa riferimento alla Figura 7.10 in cui però l'utente ha necessità di vedere solamente la disposizione dei cluster nel dettaglio. Essendo un grafo clusterizzato un grafo planare, per

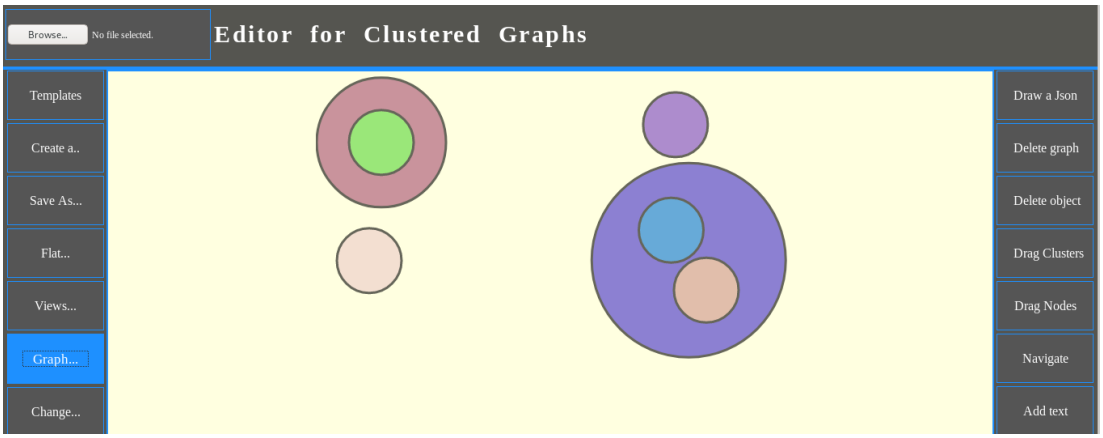

Figura 7.13: visualizzazione dei soli cluster

quanto riguarda gli archi si ha la necessità che essi non intersechino nulla ad di fuori del loro nodo *source* e del nodo *target*. In un grafo di grandi dimensioni o comunque con un numero relativamente alto di archi questa esigenza sarebbe di difficile analisi. Per questo facendo riferimento alla Figura 7.10 l'utente può avere accesso ad una rappresentazione filtrata sugli archi come mostrato in Figura 7.14. Partendo dalla visualizzazione *graph-view*, filtrando una particolare classe tra le

sue componenti, si avrà un maggiore dettaglio per quell'oggetto, ma si perderanno due gradi di integrazione, passando dalla possibilità di interazione che permette la trasformazione della visualizzazione e dei dati, a una integrazione di grado 0 che consentirà solo l'analisi, anche se è sempre possibile tornare alla rappresentazione di default per poter continuare la sessione di lavoro.

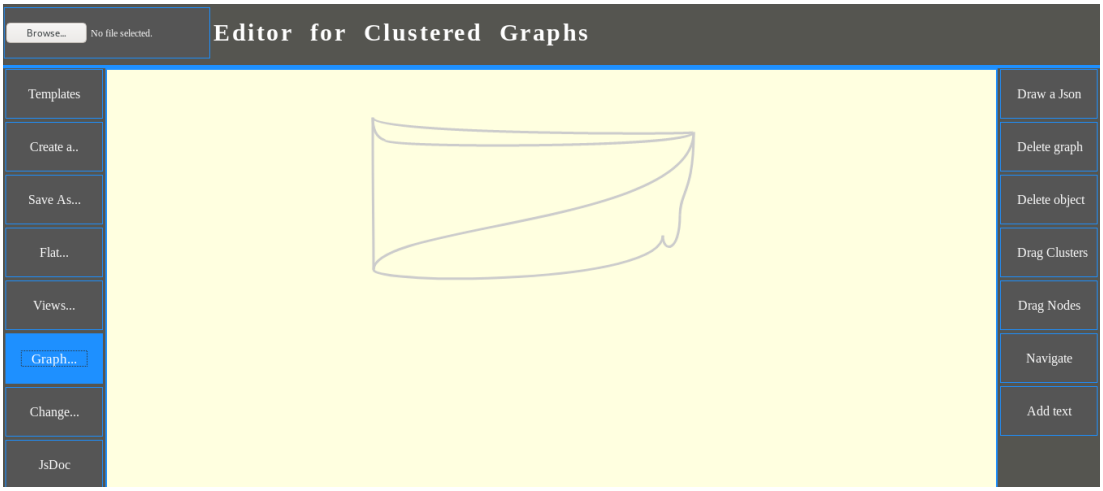

Figura 7.14: Visualizzazione dei soli archi

Capitolo 8

Editor: interazioni e semplificazioni

Di seguito saranno viste le varie operazioni eseguibili dall'utente sull'editor. Per comodità per il lettore esse saranno divise nelle tre sezioni di seguito riportate:

- **creazione** di un oggetto, di un grafo clusterizzato mediante file esterni o templates;
- **modifica** di un oggetto, aggiunta di informazioni o sostituzione di un attributo;
- **navigazione** all'interno del grafo creato;
- **semplificazione** del grafo clusterizzato;

Prima di passare alla descrizione dettagliata delle varie operazioni è necessario un focus sull'operazione di disegno del grafo nella graph-view in quanto questa è l'operazione che il sistema esegue ogni qual volta venga creato o modificato un oggetto sia della struttura dati che della sua rappresentazione.

8.1 disegno dei dati

Quasi tutte le operazioni di creazione o di modifica portano ad un cambiamento importante dei dati e quindi si ha la necessità di un costante cambiamento della loro visualizzazione. Ogni volta che l'utente esegue un cambiamento o l'aggiunta di un oggetto come mostrato schematicamente nella Figura 8.1 il sistema passerà ad effettuare una operazione di disegno dei dati per poi dare di nuovo il controllo all'utente.

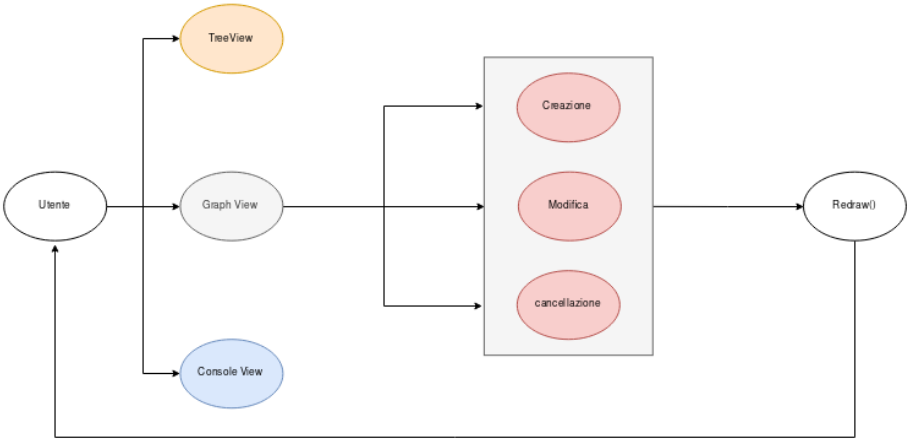

Figura 8.1: Schema dell'impiego della funzione di disegno dei dati

Quando questa operazione viene eseguita il sistema elimina completamente la sola visualizzazione degli svg lasciando però inalterato l'svg principale #cgraph e ridisegnando con i cambiamenti effettuati nuovamente i dati creati o importati. Ricordando poi che si sta operando con un modello Node-link con una rappresentazione spring-embedding una volta ridisegnati i dati si passerà all'aggiunta delle forze fisiche, che controlleranno il movimento e la rappresentazione degli oggetti, eseguita mediante le funzionalità di D3. Ogni cluster di livello l avrà dunque:

- una forza attrattiva direzionata verso il centro dello stesso esclusivamente per i nodi prenenti nel suo attributo *nodes*;
- una forza repulsiva verso gli altri cluster dello stesso livello l;
- una forza attrattiva direzionata verso il centro dello stesso esclusivamente per i cluster di livello inferiore presenti nel suo attributo *cildren*;

Ogni nodo invece avrà:

- una forza attrattiva direzionata verso il centro del cluster di appartenenza;
- una forza repulsiva verso gli altri nodi;

Queste forze sono mostrate schematicamente nella Figura 8.2 in cui ogni vettore rappresenta la forza attrattiva nel caso in cui il verso sia entrante o repulsiva nel caso in cui il verso sia uscente.
Come si nota le forze in gioco sono molteplici e il ricalcolo della condizione di equilibrio risulta comunque pesante a livello di complessità computazionale temporale. Una volta definita l'operazione di disegno degli oggetti da visualizzare è ora possibile procedere a definire come le primitive di interazione e le operazioni di semplificazione sono state impiegate nel sistema in esempio.

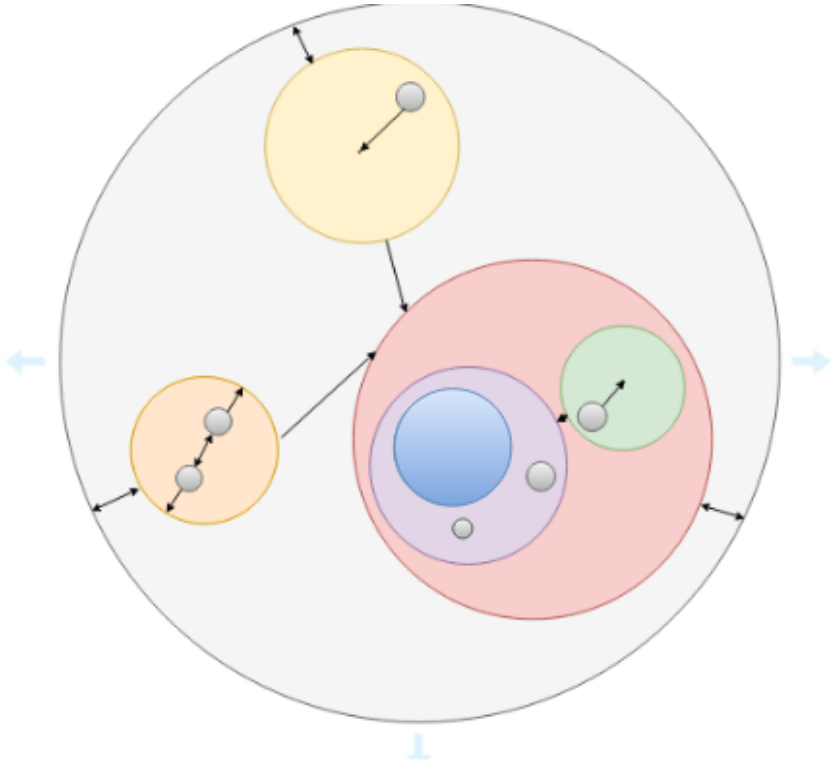

Figura 8.2: Esempio delle forze attrattive e repulsive utilizzate in fase di disegno

8.2 creazione

Iniziando una sessione di lavoro l'utente ha la possibilità come già accennato di cominciare a creare e rappresentare un grafo clusterizzato oppure di importare dei dati per poterne avere solo la loro visualizzazione. L'import può essere eseguito mediante dati definibili tabellari provenienti da file con estensione *.Json*.
Si ricorda, prima di procedere, che il JSON (JavaScript Object Notation) è un formato di scambio dati di facile lettura e scrittura, di generazione e analisi per il calcolatore. Si basa su un sottoinsieme dello standard di programmazione JavaScript ECMA-262 3a edizione del dicembre 1999. JSON è un formato di testo che è completamente indipendente dal linguaggio ma utilizza convenzioni

familiari ai programmatori della famiglia di linguaggi C, tra cui C, C ++, C #, Java, JavaScript, Perl, Python e molti altri. Queste proprietà rendono JSON un linguaggio di scambio dati ideale.
JSON è costruito su due strutture:

- Una raccolta di coppie nome / valore. In vari linguaggi, questo viene realizzato come oggetto, record, struct, dizionario, tabella hash, elenco con chiave o array associativo;
- Un elenco ordinato di valori che è realizzato come una matrice, un vettore o una sequenza.

Mediante un bottone di import è dunque possibile caricare un file json personale creato secondo la struttura fissa mostrata nella Figura 8.3 e passare questi dati al sistema per crearne la rappresentazione associata.


```
{
        "nodes":[
            {"id":0,"label":"n0","rotationScheme":[4,5],"cluster":"c4","key":0,"r":9,"index":0},
            {"id":1,"label":"n1","rotationScheme":[0,1,2],"cluster":"c2","key":1,"r":9,"index":0},
            {"id":2,"label":"n2","rotationScheme":[2,4,6,7],"cluster":"c1","key":2,"r":9,"index":0},
            {"id":3,"label":"n3","rotationScheme":[0,5,6],"cluster":"c4","key":3,"r":9,"index":1},
            {"id":4,"label":"n4","rotationScheme":[3,7],"cluster":"c5","key":4,"r":9,"index":0},
            {"id":5,"label":"n5","rotationScheme":[1,3],"cluster":"c5","key":5,"r":9,"index":1}
                ],
        "edges":[
            {"id":0,"label":"e0","source":3,"target":1},
            {"id":1,"label":"e1","source":1,"target":5},
            {"id":2,"label":"e2","source":2,"target":1},
            {"id":3,"label":"e3","source":5,"target":4},
            {"id":4,"label":"e4","source":0,"target":2},
            {"id":5,"label":"e5","source":3,"target":0},
            {"id":6,"label":"e6","source":3,"target":2},
            {"id":7,"label":"e7","source":4,"target":2}
                ],
        "clusters":[
            {"label":"c0","level":1,"cildren":[4],"parents":[],"nodes":[],"r":80,"fill":"#8AC267","key":0,"index":0},
            {"label":"c1","level":1,"cildren":[],"parents":[],"nodes":[2],"r":40,"fill":"#7DE42A","key":1,"index":1},
            {"label":"c2","level":1,"cildren":[],"parents":[],"nodes":[1],"r":40,"fill":"#770B54","key":2,"index":2},
            {"label":"c3","level":1,"cildren":[5],"parents":[],"nodes":[],"r":80,"fill":"#7AC563","key":3,"index":3},
            {"label":"c4","level":2,"cildren":[],"parents":[0],"nodes":[0,3],"r":40,"fill":"#F8F736","key":4,"index":0},
            {"label":"c5","level":2,"cildren":[],"parents":[3],"nodes":[4,5],"r":40,"fill":"#DF2169","key":5,"index":0}
        ]}
```


Figura 8.3: Esempio di file json per l'import di un grafo clusterizzato

In qualunque momento durante una sessione di lavoro è inoltre possibile l'export

del grafo su cui si sta lavorando. In particolare l'utente ha due possibilità di salvataggio: della struttura o della visualizzazione. La prima e più importante è riferita all'export dei dati mediante file json che potranno essere poi ricaricati e riutilizzati per una sessione di lavoro futura. La seconda invece è riferita alla possibilità di salvaggio della rappresentazione dei dati mediante file con estenzione *.PNG*. Si ricorda inoltre che il PNG (Portable Network Graphics) è un formato di file per memorizzare immagini. Ogni qualvolta che si sceglie di utilizzare l'import di un file esterno il sistema andrà prima ad eliminare tutto ciò che fa parte della sessione di lavoro su cui l'utente sta lavorando. In questo modo il sistema potrà importare i dati richiesti e procedere con la rappresentazione degli stessi.

Non avendo a disposizione file json per l'import dei dati o volendo semplicemente cominciare una sessione di lavoro non da una semplice pagina bianca l'utente ha a disposizione dei modelli predefiniti su cui potrà cominciare il lavoro ed andare a modificare a piacimento. I modelli non rappresentano solamente la visualizzazione in se ma si basano su dati tabellari. Esattamente come per l'import di file json anche l'utilizzo di modelli con un algoritmo simile a quello mostrato nella Figura 7.9. In altre parole ogni qual volta l'utente chiede al sistema di creare un modello questo risponde eliminando tutta la sessione di lavoro ed inizializzando nuovamente con i dati definiti durante la richiesta di creazione mediante il modello scelto. Di default all'inizio di una sessione utente il sistema presenterà, come visto nella Figura 7.8, una pagina di lavoro vuota proprio per lasciare all'utente la scelta non solo di tipologia di dati da poter creare ma anche della loro posizione sul piano di lavoro. Questo andrà però a pregiudicare il fatto che il conseguimento di una buona visualizzazione sarà compito dell'utente.

A prescindere dalla scelta di importare dati, utilizzare modelli o cominciare da un grafo vuoto, l'utente avrà la possibilità di inserire cluster, nodi e archi. È

consigliabile seguire un criterio, quando possibile, nella creazione degli oggetti. Un buon metodo di creazione è quello che applica una strategia di discesa dell'albero di inclusione top-down partendo dunque dalla creazione dei cluster di livello uno, fino ad arrivare a quelli più in profondità per poi passare alla creazione delle foglie dell'albero, ovvero dei nodi terminando con gli archi. Ovviamente questa non è una regola e non pregiudica la creazione di un grafo ma è un buon principio per la visualizzazione che un utente dovrebbe seguire. Per questo ed altri motivi sono stati realizzati dei messaggi di errore che aiutano l'utente e lo indirizzano verso l'approccio sopra definito. Supponendo ad esempio che l'utente abbia creato due cluster di livello uno ed un nodo all'interno di uno dei due e dia al sistema l'input per poter cominciare la creazione di archi esso risponderà con il messaggio di errore mostrato in Figura 8.4.

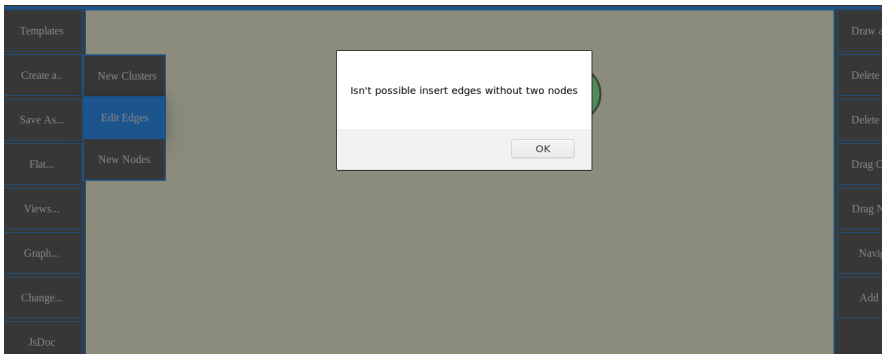

Figura 8.4: Messaggio di errore per il tentativo di creazione di un arco senza due nodi

Ogni volta che l'utente creerà un oggetto all'interno del piano di lavoro il sistema risponderà prima aggiornando la struttura dati a cui la visualizzazione fa riferimento ed andrà poi a creare nuovamente la loro rappresentazione. Una volta creati i cluster di un livello l'utente potrà creare i figli degli stessi. Recepita questa richiesta il sistema andrà ad aggiornare le strutture dati inizializzando il nuovo

oggetto e collegandolo, con gli attributi visti nella Figura 7.7, al cluster genitore. Infine per quando riguarda la creazione degli archi all'utente basterà selezionare un nodo che sarà definito sorgente e un nodo destinazione per collegarli. Si è scelto di non utilizzare linee dritte nella rappresentazione ma curvate di modo da evitare qualora possibile intersezioni tra essi come si concerne ad un grafo planare anche se è a discrezione dell'utente il poter definire un grafo in cui gli archi possono incrociarsi o meno. Nella figura Figura 8.5 è mostrato l'impiego di un gran numero di archi che tra loro non vanno ad intersecarsi al contrario di come sarebbe successo utilizzando linee rette.

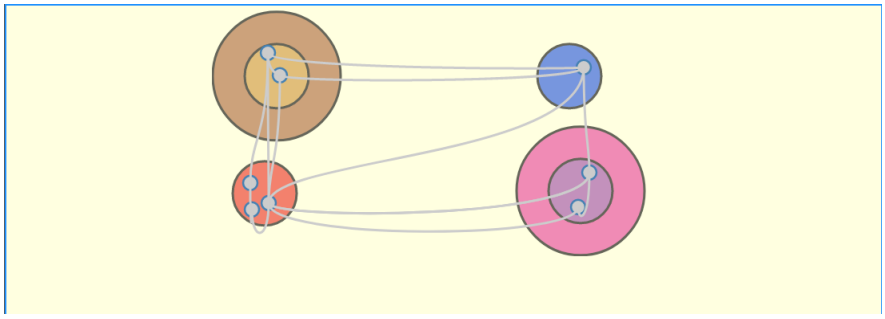

Figura 8.5: Esempio di planarità nella rappresentazione di un grafo clusterizzato

8.3 modifica

Il sistema lascia una buona libertà all'utente per quanto concerne la modifica degli oggetti le cui operazioni saranno viste nel dettaglio di seguito.

Per quanto riguarda gli oggetti visualizzati, che sono stati creati o importati durante la sessione di lavoro, l'utente ha la possibilità di spostamento sia di un oggetto cluster che di un nodo esattamente come richiesto dai principi espressi nel capitolo relativo alle primitive di interazione. Essendo l'utente finale non esente da possibili errori di creazione degli oggetti è stata realizzata una funzione per

la cancellazione di oggetti.Si riporta, in maniera semplificata, nella Figura 8.6 l'algoritmo di eliminazione di un oggetto.

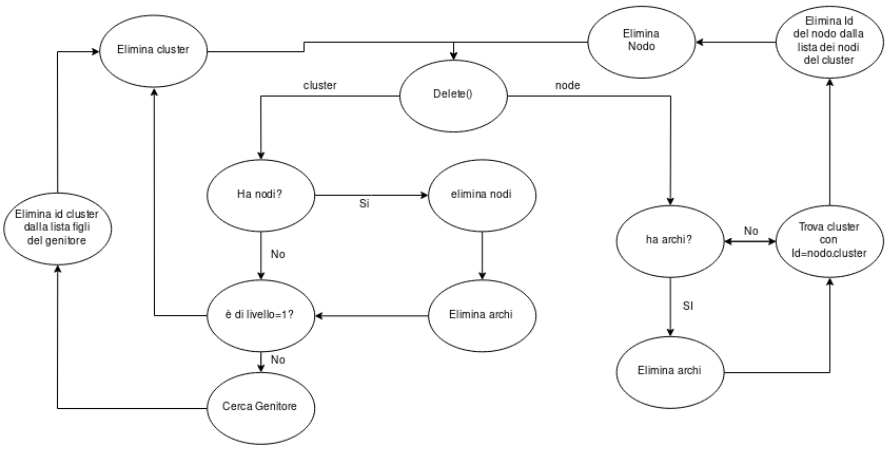

Figura 8.6: algoritmo di cancellazione di un oggetto

Oltre alle operazioni di cancellazione e spostamento l'utente ha la libertà di assegnare valori diversi da quelli di default per colori e raggio agli oggetti del grafo. In particolare, ricordando che il raggio di un cluster è definito come un valore di default k_c moltiplicato per il numero e all'entità degli oggetti mentre quello di un nodo è un semplice valore k_n, l'utente potrà andare a modificare la dimensione di questi due valori indicando al sistema la lunghezza desiderata in pixel del raggio. Il sistema applicherà la modifica a tutti gli oggetti della categoria dando un avvertimento all'utente nel caso in cui verrà inserito un valore eccessivamente grande per k_c o k_n. Avendo la dimensione dello spazio di lavoro fissa e dipendente dal proprio calcolatore, quella di poter cambiare i valori standard di un oggetto andrà si a modificarne la visualizzazione ma anche a poter avere un maggiore spazio a disposizione nel caso in cui il grafo clusterizzato creato durante la sessione di lavoro raggiungesse dimensioni considerevoli. Inoltre nel caso inverso in cui un grafo presentasse ad esempio un numero limitato di

cluster ed un significativo numero di nodi per ognuno di essi è così possibile avere dettaglio maggiore sui nodi. Ad esempio lo stesso grafo clusterizzato riportato nella Figura 7.10, andandone a dimezzare la dimensione solamente dei suoi cluster e eseguendo qualche piccolo accorgimento per quanto concerne lo spostamento degli oggetti, si nota,come è riportato nella figura Figura 8.7, come sia possibile avere a disposizione molto più spazio anche lavorando su un piano di lavoro delle stesse dimensioni di quello utilizzato in precedenza.

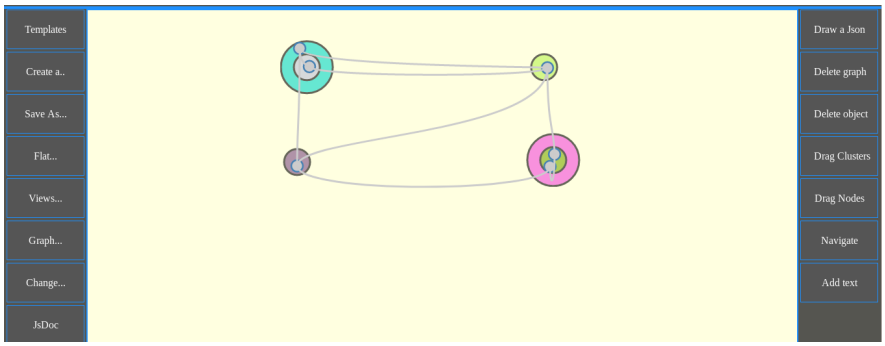

Figura 8.7: operazione di cambiamento del raggio dei cluster da parte dell'utente

Come già accennato ai cluster è assegnato un colore. Questo è di notevole importanza essendo un principio fondamentale per la rappresentazione, espresso anche nel capitolo relativo alle primitive della visualizzazione. Ad ogni cluster μ viene attribuito un valore random una volta che esso è stato aggiunto all'oggetto *ClusteredGraph* e deve esser rappresentato. L'utente può cambiare questa tipologia di colorazione in due modi di seguito illustrati.

Il primo è quello di cambiare palette di lavoro, avendo così a disposizione tre colorazioni diverse che si riferiscono a tutte le sfumature facenti parte di quella particolare tonalità. Quando l'utente sceglierà di lavorare su una precisa tonalità per il prossimo numero indefinito di cluster il sistema imposterà alcuni valori esadecimali fissi per poter avere una scala di colori che richiami solo la tonalità

richiesta dall'utente. Riprendendo nuovamente il grafo mostrato nella figura Figura 7.10 si può notare come essendo completamente casuale possono capitare anche colori non appariscenti o comunque non molto gradevoli all'occhio umano per questo l'utente mediante questa funzione potrà, a puro titolo di esempio, trasformarlo nel grafo mostrato nella Figura 8.8.

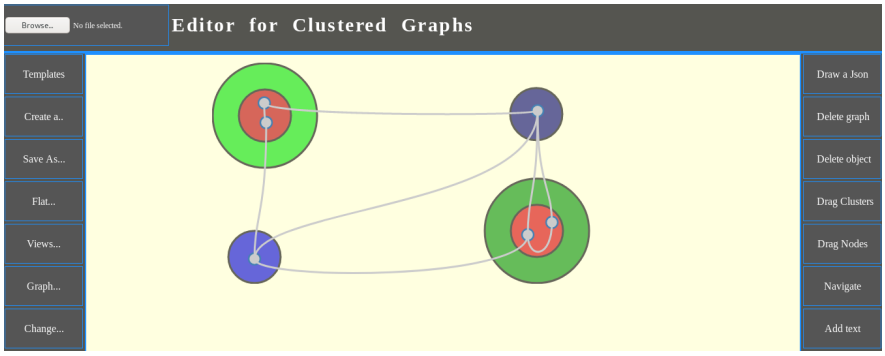

Figura 8.8: impiego di Palette diverse da quella di default da parte dell'utente

In questo modo è possibile anche catalogare e categorizzare in gruppi di cluster in base alla loro colorazione. Con il secondo metodo realizzato, l'utente ha la possibilità di sostituire il colore assegnato ad un singolo cluster. In questo caso il sistema chiederà all'utente di inserire il valore esadecimale scelto per il colore. Una volta inserito verrà chiesto all'utente di decidere l'oggetto a cui applicare tale modifica.

L'editor è predisposto per lavorare su un numero moderatamente grande di cluster e di oggetti in generali. Per questo può esser utile aggiungere sul piano di lavoro etichette o comunque descrizioni degli oggetti che sono stati rappresentati. A titolo di esempio si può porre il caso in cui ogni cluster debba rappresentare un particolare oggetto o comunque si abbia bisogno un aiuto descrittivo su cosa rappresenti un particolare elemento oppure risulta necessario lasciare dei brevi

commenti per essere visualizzati tra una sessione di lavoro e l'altra sullo stesso cluster. Questi problemi sono stati risposti dando la possibilità all'utente di poter richiedere al sistema un spazio in cui scrivere queste descrizioni per un particolare elemento del grafo clusterizzato. Una volta scritto il commento il sistema chiederà all'utente di selezionare l'oggetto a cui questa stringa sarà legata. Una volta visualizzata, la posizione all'interno della visualizzazione della stringa dipenderà dalle coordinate dell'oggetto associato di modo che in caso di modifica, spostamento o cancellazione, la posizione del testo verrà modificata e/o eliminata. Un esempio di utilizzo, facendo ancora una volta riferimento alla Figura 7.10 del capitolo precedente, potrebbe essere quello riportato nella Figura 8.9.

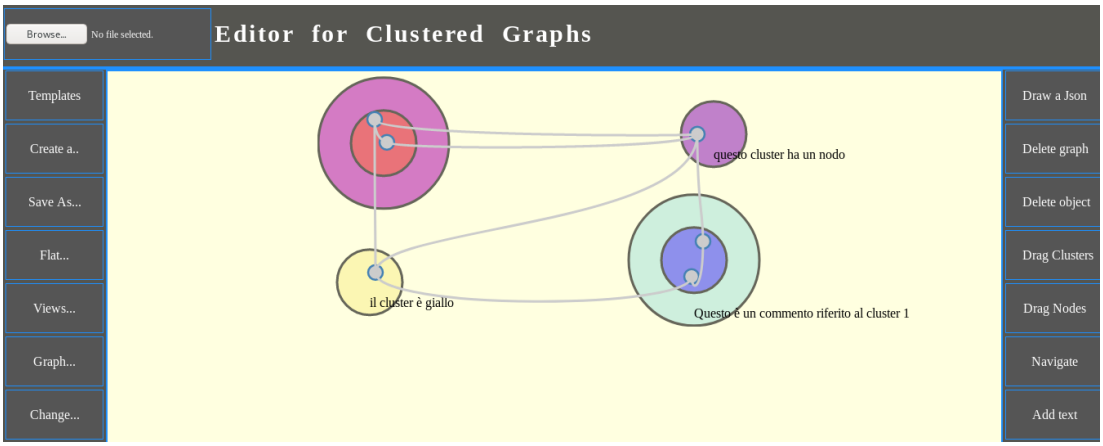

Figura 8.9: aggiunta di commenti da parte dell'utente

Nella figura i commenti visualizzati sono a puro titolo di esempio, ma basti pensare alla possibile utilità nel caso in cui si debba lavorare con decine di oggetti magari divisi in sottosezioni differenziate per colore ed in cui ogni raggruppamento ha uno specifico obiettivo che deve essere trasmesso da un utente ad un altro.

8.4 navigazione

La navigazione di una rappresentazione risulta essere una delle principali primitive per la visualizzazione come visto anche nel capitolo riferito proprio alle stesse.
Una buona visualizzazione deve poter dare all'utente la possibilità di navigare all'interno del piano di lavoro.
Non è però possibile modificare e navigare contemporaneamente in quanto risulterebbe difficoltoso e poco efficiente poter modificare un oggetto quando si sta eseguendo ad una operazione di focus su una particolare sezione di un grafo. Per questo, nel momento in cui l'utente ha necessità di eseguire queste operazioni relative ad uno zoom che sia semantico o non, dovrà prima richiedere al sistema di entrare in una modalità che può essere definita "*navigate-view*". Entrando per la prima volta durante una stessa sessione all'interno di questa modalità sarà visualizzato un messaggio di aiuto per l'utente, come mostrato nella figura Figura 8.10, che indicherà le operazioni eseguibili in questa modalità.

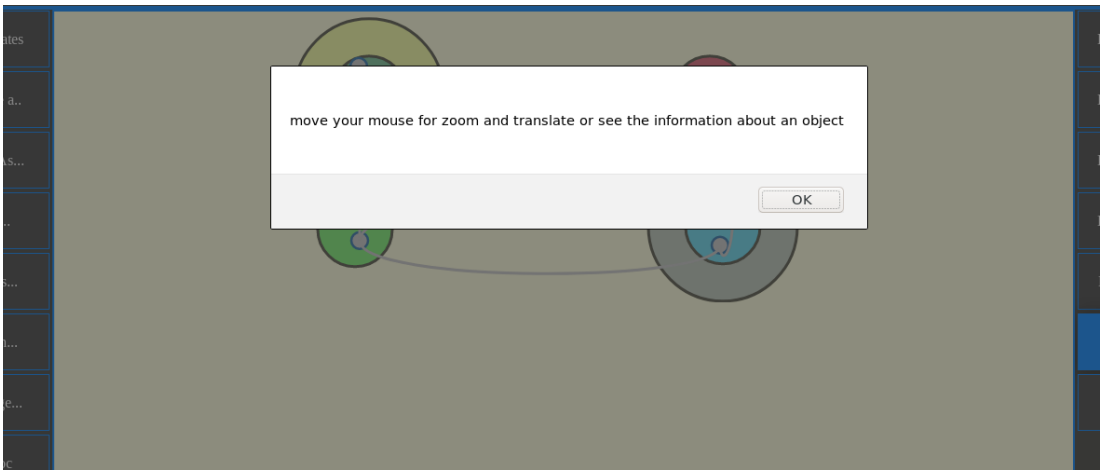

Figura 8.10: Messaggio di aiuto al primo accesso dell'utente alla navigate-view

Spostandosi sul piano di lavoro sarà possibile eseguire le primitive di traslazione e di zoom-in/out dell'intera rappresentazione del grafo clusterizzato come mostrato nella Figura 8.11 in cui si sono eseguite le operazioni di traslazione e zoom-In rispetto alla rappresentazione della Figura 7.10.

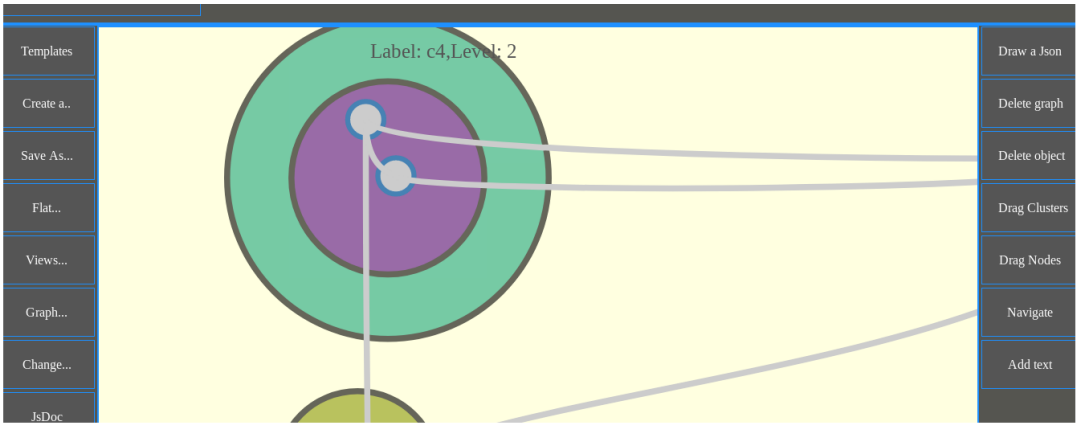

Figura 8.11: zooming di un grafo nella navigate-view

In questa modalità di visualizzazione inoltre passando sopra un oggetto sarà mostrato uno zoom semantico a schermo per dare più attenzione all'elemento. In particolare questo focus dipenderà dal tipo di oggetto evidenziato come mostrato di seguito:

- **Nodo**: il raggio r_n verrà moltiplicato per una costante e verrà visualizzata, in alto a destra, la sua etichetta e l'id del suo cluster di appartenenza;
- **Cluster**: il raggio r_c verrà moltiplicato per una costante e verrà visualizzata la sua etichetta insieme al livello e alla lista di nodi che possiede al suo interno;
- **Arco**: saranno visualizzati gli id dei nodi che l'arco collega ovvero i suoi attributi *source* e *target*.

Gli attributi evidenziati saranno poi eliminati una volta che l'utente sposterà il cursore verso un oggetto diverso o verso il piano di lavoro, così da poter lasciare la possibilità al sistema di poter evidenziare un altro oggetto o di proseguire con la sessione di lavoro una volta uscito dalla *navigate-view*.

8.5 Semplificazione

Definite e chiarite le riduzioni polinomiali nel capitolo 6 si passa ora alla realizzazione. Per semplicità si è scelto di utilizzare una sola delle due tipologie di semplificazioni. In particolare questa scelta è ricaduta sulla riduzione da grafo clusterizzato C in grafo Flat C_f. L'utente, mediante una interazione su un bottone On/Off, potrà in qualunque momento richiedere al sistema di eseguire la semplificazione analizzata. Si è scelto di dare la possibilità all'utente di poter tornare alla visualizzazione precedente la trasformazione dei dati di modo da poter continuare la sessione e vedere potenziali differenze tra grafi clusterizzati e le loro riduzioni.
Lavorando nella visualizzazione a grafo, ovvero la *graph-view*, ed avendo ultimato una prima analisi l'utente chiederà al sistema di eseguire la riduzione. Il sistema risponderà eseguendo l'algoritmo mostrato schematicamente nella Figura 8.12 e riportato di seguito in maniera semplificata. Alla base, l'algoritmo di riduzione in grafo clusterizzato flat è composto dai seguenti step:

- **Step 0** Inizializzazione;
- **Step 1** Creazione cluster sostitutivi;
- **Step 2** Creazione dei nodi aggiuntivi;
- **Step 3** Sostituzione archi con percorsi;

- **Step 4** visualizzazione della classe *clusteredGraph* ora resa Flat.

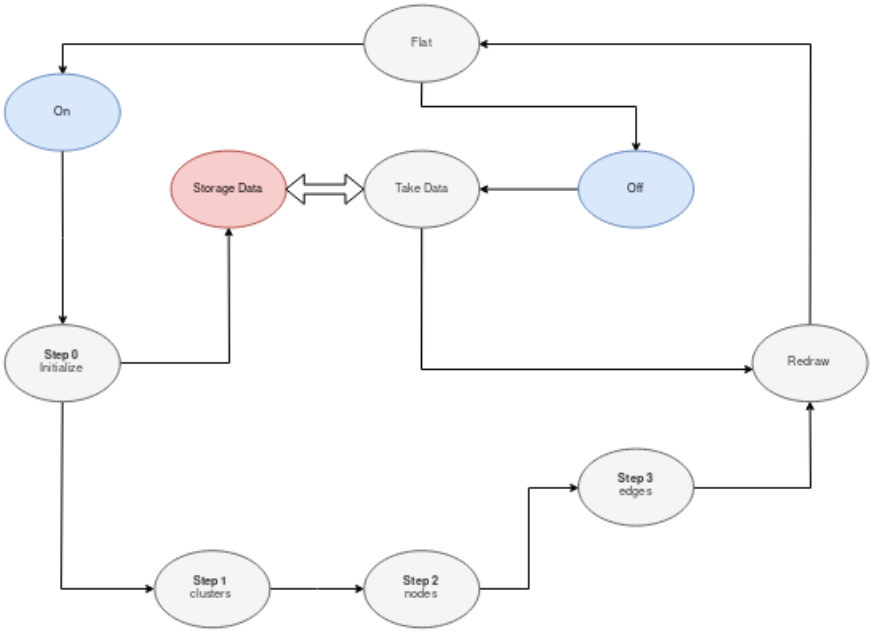

Figura 8.12: Struttura semplificata dell'algoritmo di flattizzazione

Nella fase di inizializzazione il sistema prima di eseguire qualunque cambiamento sugli oggetti creati e/o importati dall'utente provvederà ad eseguire una copia di questi dati per poter ritornare, se richiesto da una interazione utente, ai valori pre-riduzione e continuarne la modifica. Successivamente vengono create delle liste contenenti solo ed esclusivamente nodi e cluster del grafo su cui eseguire la riduzione. Gli elementi contenuti nella lista dei nodi saranno quelli con archi uscenti dal cluster e gli elementi della lista dei cluster saranno quelli con livello $l > 2$.

Superata la fase di inizializzazione si passa alla trasformazione dei dati. Nello step 1 $\forall$cluster $\mu_i, \forall i = 0, ..., clusters.length$, con $clusters$ uguale alla lista dei

cluster da cambiare inizializzata nello step 0, μ_i verrà eliminato ed al suo posto saranno inseriti due oggetti cluster X o Y che possiederanno come attributo label le rispettive etichette μ_i_X e μ_i_Y.

Creati i cluster si passa poi allo step 2, ovvero alla creazione dei nodi aggiuntivi. In particolare $\forall$nodo $n_i, \forall i = 0, ..., nodes.length$ con nodes uguale alla lista dei nodi da cambiare inizializzata nello step 0, si creeranno due nodi $n_{i,j}\_X$ ed $n_{i,j}\_Y$ rispettivamente interni ai cluster creati prima, $\forall$ arco $e_j, \forall j = 0, ..., n_i.rotationScheme.length$ uscente dal cluster di appartenenza di n_i.

Nello step 3, creati cluster e nodi, si può passare alla rimozione degli archi inter-cluster e alla creazione del percorso $(source, e_\chi)(e_\chi, e\varphi)(e_\varphi, target)$ Terminate le trasformazioni il sistema concluderà con una operazione di visualizzazione della struttura dati mediante la rappresentazione riportata nella Tree-view come mostrato nella Figura 8.13.

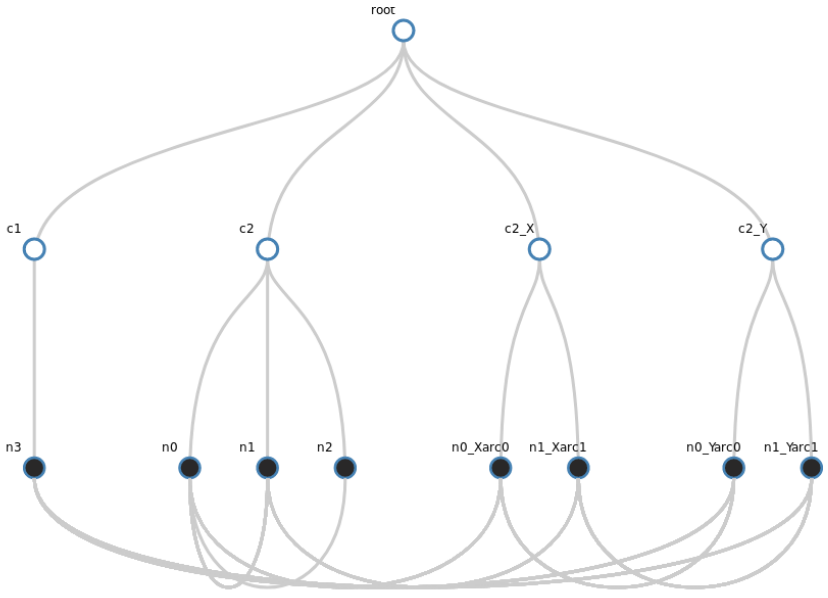

Figura 8.13: Esempio della riduzione di un grafo clusterizzato nella Tree-View

In qualunque momento l'utente potrà tornare alla visualizzazione del grafo non ancora ridotto e continuare il lavoro svolto. Esempi reali di utilizzo di questa riduzione saranno visti nel dettaglio nel capitolo successivo in cui verranno utilizzate anche la maggior parte delle interazioni ed operazioni dell'utente. Sarà eseguita una simulazione reale di utilizzo e di test non solo inerente le connessioni tra gli oggetti che compongono il grafo clusterizzato ma anche sulla loro visualizzazione. Il sistema infine non pone alcun messaggio di errore nel caso in cui l'utente decida di effettuare una operazione di riduzione del grafo clusterizzato contenente errori nella sua definizione o nella sua visualizzazione in quanto non previsto che vengano eseguiti controlli su grafi erroneamente disegnati. Questo porta sicuramente a errori iniziali da parte dell'utente nell'approccio con il sistema ma dona anche la possibilità di eseguire qualunque operazione e grafo si decida di realizzare lasciando pieno controllo umano.

Capitolo 9

Esempio reale di utilizzo

Durante lo svolgimento di caso di studio, verrà ricreata, visualizzata e semplificata la struttura del grafo clusterizzato della Figura8.2, utilizzato in precedenza a titolo di esempio per delucidare il lettore sulle forze in gioco tra cluster e nodi nel metodo spring-embedding. Inoltre sarà supposto che il cluster contenente tutto il resto del grafo sia definito come radice dell'albero di inclusione T.

9.1 Creazione e modifica

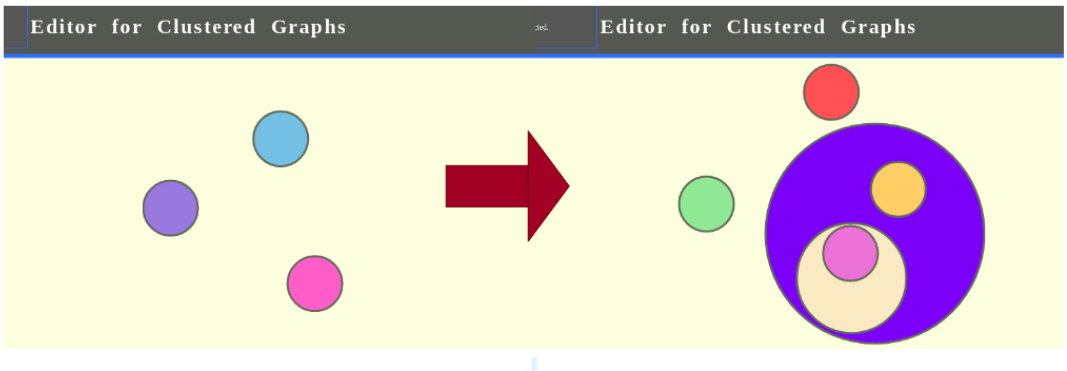

Figura 9.1: I cluster pre e post inserimento dei sotto-cluster

Dopo il log-in iniziale viene automaticamente creato il cluster radice, essendo rappresentato dal piano di lavoro #cgraph stesso. Essendo un caso di studio del quale non si possiede un file json prestabilito ed essendo differente da ogni modello preimpostato del software, risulta necessaria la creazione partendo da una pagina bianca. Detto ciò anche l'oggetto *clusteredGraph* sarà completamente vuoto e l'utente inizierà ad utilizzare il sistema partendo dalla creazione dei cluster di livello uno. Definiti questi oggetti si passerà alle interazioni per la creazioni dei cluster figli. In questo caso sarà necessaria dunque una interazione per ogni cluster figlio che si vorrà definire poiché ogni volta il sistema dovrà aggiornare la visualizzazione dei dati creati in quanto cambieranno il raggio $r_c \forall \mu$ cluster trasformato e nuove forze dovranno essere inserite. La creazione dei figli di un cluster può essere vista nella Figura9.1 in cui si mette a confronto la visualizzazione dei dati antecedente e conseguente l'inserimento dei sotto-cluster. Come si nota, prima e dopo le operazioni di inserimento, sono state cambiate automaticamente dalla funzione di redraw le posizioni degli oggetti e manualmente, mediante una interazione da parte dell'utente, i colori degli oggetti mediante le interazioni già analizzate.Si passa ora all'introduzione di nodi e delle loro connessioni. Avendo terminato le interazioni relative alla creazione degli elementi richiesti dal caso di studio si può passare ad eseguire operazioni di encode dei dati.

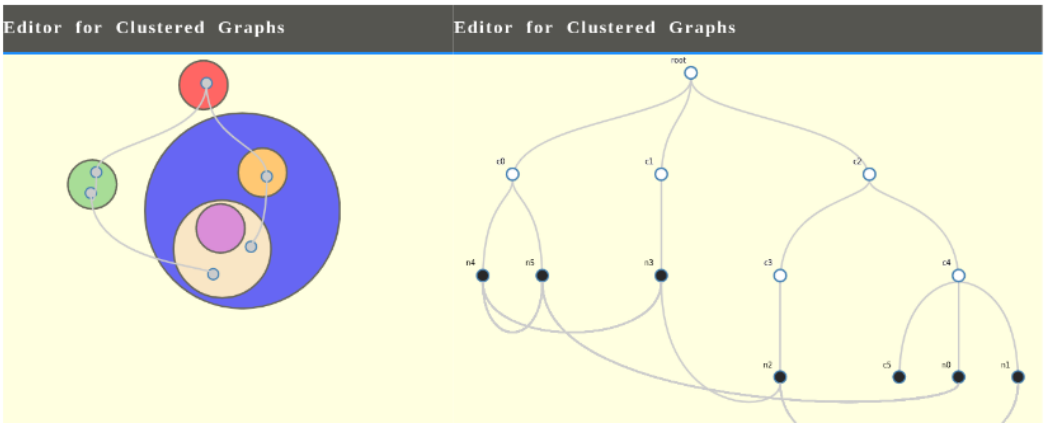

Figura 9.2: visualizzazione del grafo rappresentato nelle due visualizzazioni

Nella Figura9.2 è mostrato il compimento del grafo richiesto e della sua visualizzazione in entrambi gli encode disponibili. Si nota inoltre che vi è un cluster senza nodi e non connesso a nulla. L'utente può decidere di eliminarlo oppure lasciarlo nel caso di future modifiche magari inserendo un testo descrittivo per quel particolare elemento. Entrambe le operazioni sopra citate possono essere eseguite come mostrato nella figura Figura9.3 in cui sono state eseguite l'operazione di rimozione dell'oggetto superfluo a sinistra oppure l'operazione di descrizione a destra. Nel caso in cui l'elemento in questione venga rimosso, il sistema eseguirà un ricalcolo del raggio r_c e delle forze del modello Force-directed scelto per ritrovare una posizione di equilibrio, esattamente come descritto nel capitolo relativo alle primitive di interazione.Creato il grafo clusterizzato richiesto l'utente può salvare il suo lavoro in un file json che ne descrive la struttura. Salvando la sessione di lavoro nelle volte future in cui si dovranno eseguire modifiche o ulteriori creazioni di oggetti non sarà necessario ricreare completamente la struttura ma basterà caricare i progressi salvati nella sessione precedente risparmiando tempo ed imperfezioni.

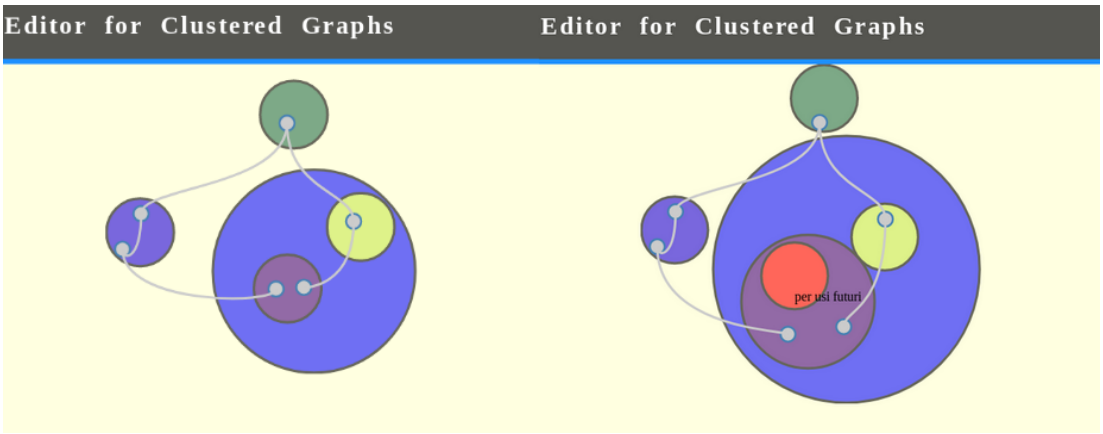

Figura 9.3: Rimozione o aggiunta di un testo su un cluster

Il file creato al salvataggio sarà lo stesso che in Figura9.4 È inoltre possibile come già visto un salvataggio del grafo clusterizzato in un file con estensione .png. La resa della visualizzazione resta comunque il compito principale dell'utente ed è variabile al tempo impiegato per la creazione degli oggetti e per il loro posizionamento esatto. Per questo se si hanno esigenze specifiche riguardanti punti esatti e colorazioni predefinite si consiglia sempre di cominciare una sessione di lavoro su un piano vuoto.

```
{
"nodes":
        [
        {"id":0,"label":"n0","rotationScheme":[2],"cluster":"c4","x":428,"y":323.79998779296875,"key":0,"r":9,"index":0,"vy":0,"vx":0},
        {"id":1,"label":"n1","rotationScheme":[1,2],"cluster":"c0","x":271,"y":218.8000030517578,"key":1,"r":9,"index":0,"vy":0,"vx":0},
        {"id":2,"label":"n2","rotationScheme":[0,1],"cluster":"c0","x":293,"y":176.8000030517578,"key":2,"r":9,"index":1,"vy":0,"vx":0},
        {"id":3,"label":"n3","rotationScheme":[4],"cluster":"c4","x":495,"y":314.79998779296875,"key":3,"r":9,"index":1,"vy":0,"vx":0},
        {"id":4,"label":"n4","rotationScheme":[0,3],"cluster":"c1","x":466,"y":69.80000305175781,"key":4,"r":9,"index":0,"vy":0,"vx":0},
        {"id":5,"label":"n5","rotationScheme":[3,4],"cluster":"c3","x":545,"y":185.8000030517578,"key":5,"r":9,"index":0,"vy":0,"vx":0}
        ],

"edges":
        [{"id":0,"label":"e0","source":4,"target":2,"x1":466,"y1":69,"x2":293,"y2":176},
        {"id":1,"label":"e1","source":2,"target":1,"x1":293,"y1":176,"x2":271,"y2":218},
        {"id":2,"label":"e2","source":0,"target":1,"x1":453,"y1":265,"x2":271,"y2":218},
        {"id":3,"label":"e3","source":5,"target":4,"x1":545,"y1":185,"x2":466,"y2":69},
        {"id":4,"label":"e4","source":5,"target":3,"x1":545,"y1":185,"x2":486,"y2":263}
        ],

"clusters":
        [
        {"label":"c0","level":1,"cildren":[],"parents":[],"nodes":[1,2],"x":292,"y":198.8000030517578,"r":40,"fill":"#2003DA","key":0,"index":0,"vy":0,"vx":0},
        {"label":"c1","level":1,"cildren":[],"parents":[],"nodes":[4],"x":471.5571795528045,"y":40.969928771291585,"r":40,"fill":"#27704B","key":1,"index":1},
        {"label":"c2","level":1,"cildren":[3,4,5],"parents":[],"nodes":[],"x":498.15267627794964,"y":245.97688269428687,"r":160,"fill":"#1010FF","key":2,"index":2},
        {"label":"c3","level":2,"cildren":[],"parents":[2],"nodes":[5],"x":544.2607071197976,"y":207.25619261917362,"r":40,"fill":"#C8E94F","key":3,"index":0},
        {"label":"c4","level":2,"cildren":[5],"parents":[2],"nodes":[0,3],"x":452.04464543610146,"y":284.69757276940015,"r":80,"fill":"#4A077F","key":4,"index":1},
        {"label":"c5","level":3,"cildren":[],"parents":[4,2],"nodes":[],"x":436,"y":253.79998779296875,"r":40,"fill":"#FF0101","key":5}
        ]
}
```


Figura 9.4: Json riferito al salvataggio della struttura del caso di studio

9.2 Riduzione e ripresa del grafo

Terminate le operazioni inerenti la creazione e la modifica di un elemento o dell'intero grafo clusterizzato, l'utente decide di rendere il grafo della Figura9.2 flat. Supponendo poi che sia stato eliminato il cluster senza nodi in quanto non necessario alla visualizzazione, mediante una ulteriore interazione è possibile eseguire l'operazione vista nel capitolo precedente. Il risultato è quello mostrato nella figura Figura9.5 mediante la quale si vede il risultato della semplificazione dell'istanza di grafo clusterizzato $C =< G, T >$ in una istanza $C_f =< G_f, T_f >$ con T_f flat. A questo punto si ha la possibilità di procedere in due modi:

- Analizzare l'istanza e salvare la struttura dati ricevuta sempre mediante file json come risultato ottenuto dalla sessione di lavoro;
- Ritornare allo stato della sessione precedente quella ottenuta dall'operazione di semplificazione.

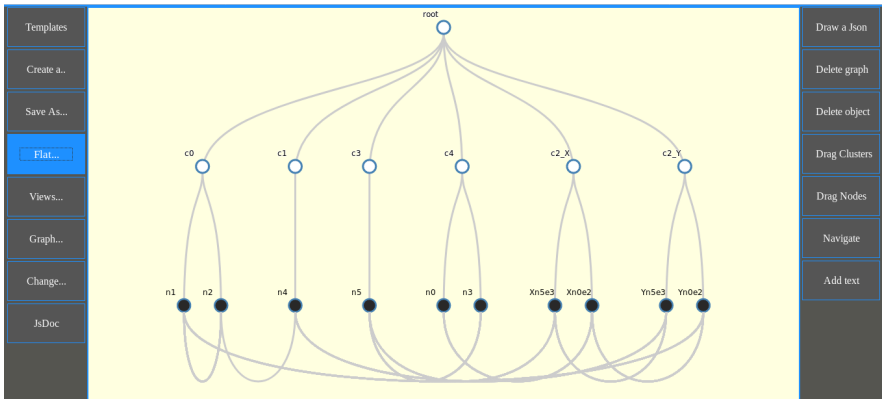

Figura 9.5: Visione nella treeView dell'istanza flat del grafo clusterizzato

È infatti possibile per l'utente continuare la modifica del grafo e ritornare ad avere i dati e la conseguente visualizzazione precedenti l'operazione di riduzione nell'istanza equivalente. Questo è possibile poiché, come visto nel capitolo precedente, durante il primo passo dell'algoritmo di semplificazione, si esegue il salvataggio dei dati relativi all'ultima visualizzazione antecedente la riduzione in grafo clusterizzato flat rendendo l'interazione relativa all'operazione di semplificazione binaria On/Off. Analizzando la graphView dopo la flattizzazione il risultato sarà non dissimile da quello riportato nella Figura9.6.

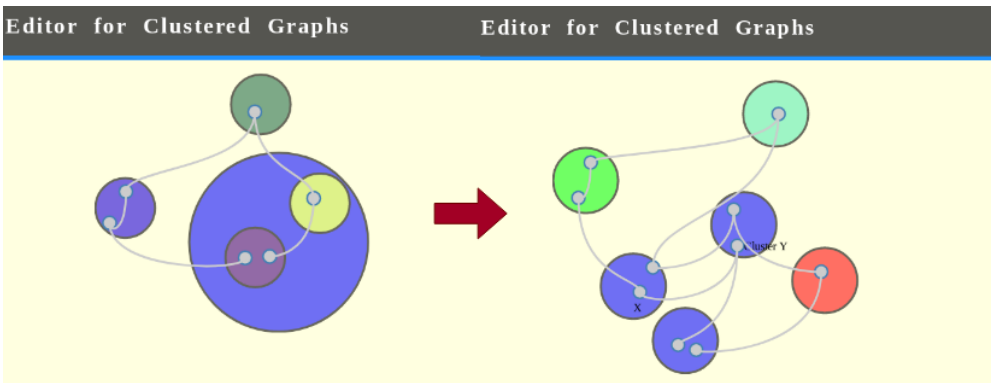

Figura 9.6: Visione nella graphView dell'istanza flat del grafo clusterizzato

Conclusioni e sviluppi futuri

Inizialmente sono stati analizzati e consolidati i problemi appartenenti a ciò che concerne la teoria dei grafi, relativamente ai grafi clusterizzati e alla loro visualizzazione. I problemi principali identificati sono:

1. le primitive di integrazione per i grafi non sono utilizzabili per i grafi clusterizzati;
2. l'assenza delle operazioni di semplificazione all'interno dei visualizzatori;
3. la mancanza di software specializzati in grado di trattarne la struttura dati e la conseguente visualizzazione.

I problemi 1 e 2 sono stati poi risolti mediante uno studio relativo alla creazione di uno "standard" sulle possibili primitive di interazione e sulle operazioni di semplificazione. Le primitive identificate sono ora utilizzabili per qualunque grafo clusterizzato e visualizzatore. In aggiunta i metodi per la semplificazione, frutto di anni di ricerche, rappresentano un traguardo nell'ambito della ricerca accademica relativa al disegno planare di grafi clusterizzati. La riduzione polinomiale FLAT C_PLANARITY riportata presenta inoltre un traguardo della ricerca che permette di eseguire indagini sulla complessità della planarità di un grafo clusterizzato legittimamente limitate a grafici flat di grafi planari(indipendenti), trascurando gerarchie più complesse dell'albero di inclusione.

È adesso possibile implementare in qualunque software queste semplificazioni migliorando il processo di ricerca. Lo studio è stato poi accompagnato dalla realizzazione del primo software specialistico in grado di lavorare sui grafi clusterizzati andando dunque a risolvere il problema 3 identificato all'inizio del lavoro svolto. Nonostante i risultati riportati è sempre possibile, essendo ancora un settore poco esplorato, ragionare ed immaginare sviluppi futuri. Di seguito è elencata una lista dei possibili sviluppi futuri riguardanti il puro studio teorico:

- studio di nuove operazioni di semplificazione;
- definire la complessità del decidere quando un generico grafo clusterizzato C ammette un disegno planare clusterizzato $\tau(C)$;
- definizione di nuove primitive di interazione per i grafi clusterizzati

Per quanto riguarda invece i visualizzatori ed il software di esempio realizzato possibili sviluppi futuri sono:

- creazione di nuovi visualizzatori mediante le primitive di interazione analizzate in questo lavoro;
- realizzazione di nuove operazioni di modifica o di creazione a disposizione dell'utente sul software in esempio;
- ideare nuove tipologie di visualizzazione da impiegare nel visualizzatori grafici

Per concludere la visualizzazione delle informazioni risulta essere un ambito che nel prossimo futuro rivestirà un ruolo di principale importanza in quanto si sta assistendo alla grande crescita relativa alla quantità di dati creati ed elaborati. Essendo poi, nella teoria dei grafi, i problemi riguardanti i grafi clusterizzati ed

il disegno planare un ambito di studio aperto da venti anni e ben lontano dalla fine delle ricerche, si riscontra il contributo portato dal lavoro svolto.

Bibliografia

[ADL16] P. Angelini and G. Da Lozzo. SEFE = C-Planarity? *The Computer Journal*, 59(12):1831–1838, 12 2016.

[AFP10] Patrizio Angelini, Fabrizio Frati, and Maurizio Patrignani. Splitting clusters to get c-planarity. In David Eppstein and Emden R. Gansner, editors, *Graph Drawing*, pages 57–68, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[AL16] Patrizio Angelini and Giordano Da Lozzo. Clustered Planarity with Pipes. In Seok-Hee Hong, editor, *27th International Symposium on Algorithms and Computation (ISAAC 2016)*, volume 64 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:13, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[BETT98] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1998.

[CP18] Pier Francesco Cortese and Maurizio Patrignani. Clustered planarity = flat clustered planarity. *CoRR*, abs/1808.07437, 2018.

[CW06] Sabine Cornelsen and Dorothea Wagner. Completely connected clustered graphs. *J. Discrete Algorithms*, 4(2):313–323, 2006.

[DBDM02] Giuseppe Di Battista, Walter Didimo, and A. Marcandalli. Planarization of clustered graphs. In Petra Mutzel, Michael Jünger, and Sebastian Leipert, editors, *Graph Drawing*, pages 60–74, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[eMP18] Pier Francesco Cortese e Maurizio Patrignani. Clustered planarity = pipe clustered planarity. 2018.

[FCE95] Qing-Wen Feng, Robert F Cohen, and Peter Eades. Planarity for clustered graphs. In *European Symposium on Algorithms*, pages 213–226. Springer, 1995.

[Fen97] Qingwen Feng. *Algorithms for drawing clustered graphs.* Citeseer, 1997.

[Fow02] Martin Fowler. *UML distilled. Guida rapida al linguaggio di modellazione standard.* Pearson; 4 edizione, 2002.

[KB01] Robert C. Martin e Martin Fowler Kent Beck. Manifesto for agile software development. 2001.

[Mar02] Rober C. Martin. *Agile Software Development: Principles, Patterns, and Practices.* Pearson; 1st edition, 2002.

[SM91] K. Sugiyama and K. Misue. Visualization of structural information: automatic drawing of compound digraphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(4):876–892, 1991.

[SM95] Kozo Sugiyama and Kazuo Misue. Graph drawing by the magnetic spring model. 6(3):217 – 231, 1995.