UNIVERSITÀ DEGLI STUDI ROMA TRE
Sezione di Informatica e Automazione

Dipartimento di Ingegneria
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi Di Laurea

Editor per grafi clusterizzati con supporto per operazioni di semplificazione e trasformazione

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

Elenco delle tabelle

Introduzione

Ringraziamenti

Lorem ipsum dolor sit amet, consectetur adipisci elit, sed do eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullamco laboriosam, nisi ut aliquid ex ea commodi consequatur. Duis aute irure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.Lorem ipsum dolor sit amet, consectetur adipisci elit, sed do eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullamco laboriosam, nisi ut aliquid ex ea commodi consequatur. Duis aute irure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.Lorem ipsum dolor sit amet, consectetur adipisci elit, sed do eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrum exercitationem ullamco laboriosam, nisi ut aliquid ex ea commodi consequatur. Duis aute irure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

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 sviluppo del primo editor per clustered graphs e con supporto per operazioni di semplificazione e trasformazione e "flattizzazione" del grafo.

Le macro-aree di interesse sono quindi la visualizzazione delle informazioni, la teoria dei grafi, la programmazione web e Javascript. Tutti i concetti enunciati nella descrizione sintetica qui riportata verranno approfonditi, illustrati, documentati e motivati nel corso della lettura. Alla base dello studio vi è la necessità della creazione di un editor che faciliti lo sviluppo di applicazioni grafiche per la ricerca scientifica sui grafi.
La tesi si concentra in questi sei capitoli di seguito anticipati:

1. Stato dell' arte: breve panoramica sui concetti chiave del lavoro;

2. Strumenti e metodologie: un focus sugli strumenti e sulle tecnologie usate per risolvere i problemi e sulle metodologie;

3. Analisi dei requisiti e problemi: definizione degli obiettivi e raccolta dei requisiti progettuali e problemi da risolvere individuati durante le fasi di analisi;

4. Progettazione e implementazione: la descrizione dell' architettura del progetto finale, con approfondimento delle scelte progettuali;

5. Esempi di utilizzo e sviluppi futuri: considerazioni sulle possibili implementazioni future e esempi di applicazione;

6. Conclusioni: considerazioni sui risultati ottenuti.

Capitolo 1

Stato dell'arte

1.1 Definizioni preliminari

Di seguito andremo a definire le varie nozioni preliminari che dovranno essere ben note e delucidate prima di poter proseguire con la definizione di grafo clusterizzato. Le seguenti definizioni non hanno l'obiettivo di chiarire e delucidare a pieno il lettore riguardo gli argomenti trattati ma sono definizioni e spunti per poter definire almeno in minima parte l'ambito di studio su cui si andrà a sviluppare il lavoro svolto e anticipazioni per quanto concerne la struttura dati creata su cui l'editor andrà a lavorare.

1.1.1 Grafi

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

$$E \subseteq V * V$$

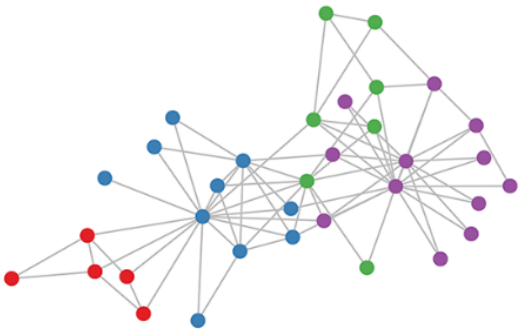

Figura 1.1: Esempio di grafo indiretto connesso

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. Data la definizione di grafo è necessario ,ai fini di una introduzione al mondo della teoria dei grafi ed in particolare a quello dei grafi clusterizzati, enunciare la definizione di grafo **planare**.

Un grafo **planare** nella teoria dei grafi è definito come un grafo che può essere raffigurato in un piano in modo che non si abbiano archi che si intersecano. È facile da intuire che non tutti i grafi sono planari, e se ne riportano due famosi esempi in **Figura 1.2**

In particolare questi due grafi sono chiamati anche grafi di Kuratowski mediante i quali si può dare la definizione di grafo Planare anche 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 $K_{3,3}$

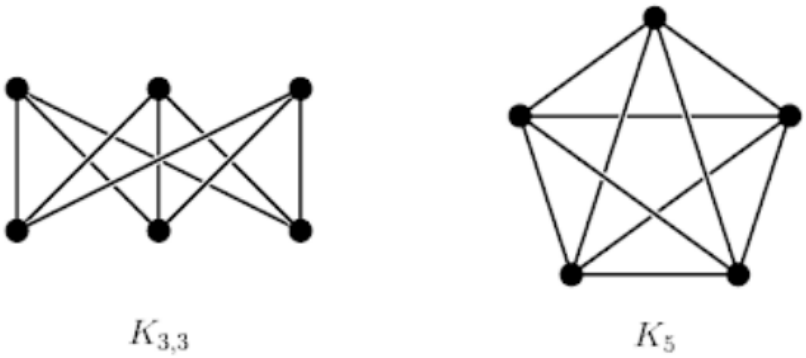

Figura 1.2: grafi di Kuratowski

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 ogni arco non interseca nulla eccetto i due vertici che connette. Si richiama inoltre 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 ed è possibile formulare il seguente teorema

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

Questo è un punto chiave nella teoria dei grafi quando poi si andrà a discutere dei grafi clusterizzati poco più avanti. I grafi risultano comunque essere una delle strutture fondamentali su cui si basano molte applicazioni reali.

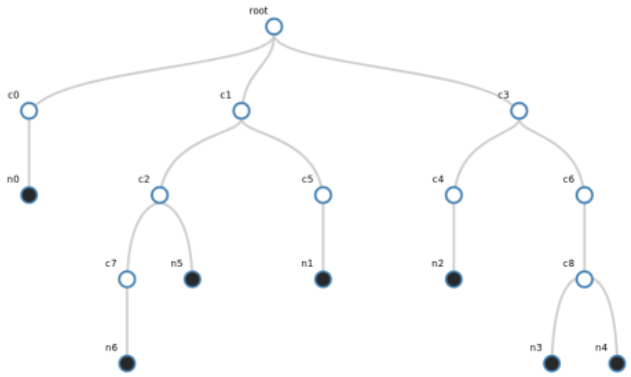

Figura 1.3: esempio di albero n-ario

1.1.2 Alberi

Data la definizione di grafo si può dunque definire un albero **tree** come un grafo connesso non orientato e senza cicli come mostrato in **Figura 1.3**. Per essere tale quindi il grafo in questione o 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). 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 e nodi alti che possiedono almeno

un figlio nodo interno;

- **foglie** che non possiedono figli e per questo non hanno archi uscenti.

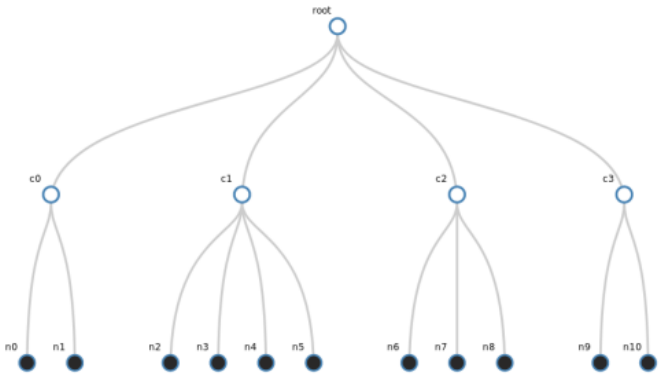

Figura 1.4: esempio di flat Tree

Un nodo è inoltre definibile **omogeneo** se tutti i figli di quel nodo sono foglie o sono 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 di inclusione 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 1.4**) come

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 come figli.
Per completezza si voglio dare anche le definizioni di profondità di un nodo e di

altezza di un albero rispettivamente come la lunghezza del cammino dalla radice al nodo e la profondità massima dei suoi nodi. Una volta terminate le definizioni iniziali si può proseguire con quella di grafo clusterizzato.

1.2 C-Graph

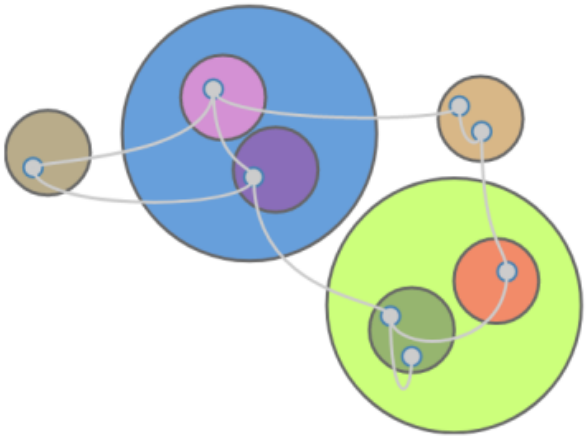

Figura 1.5: esempio di grafo clusterizzato

Un grafo clusterizzato (**c-graph**),di cui ne è un esempio la **Figura 1.5**, è definito come un grafo **planare** con una gerarchia ricorsiva definita sui suoi vertici. In altre parole un *c-graph* C è definito come

$$C =< \mathbf{G}, \mathbf{T} >$$

Ovvero come una coppia $< G, T >$ dove:

- $G = (V, E)$ 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 infatti si denota che 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 dovrà quindi avere una "conoscenza" del nodo precedente(genitore) e dei nodi successivi(figli) Per quanto riguarda la struttura dati quella sarà analizzata con più consapevolezza nei capitoli successivi in cui verranno elencate e motivate scelte di progetto partendo dalle definizioni date ora.

1.2.1 clustered planarity

Avendo definito la struttura c-graph C si può andare ad analizzare come si comporta per quanto concerne il disegno $\tau(C)$. Avendo definito l'underlying graph come un grafo planare si devono fare delle considerazioni per quanto riguarda il disegno dell'intero C-graph.
Questo disegno sarà un disegno planare **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 τ(c-graph) 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 c-graph ammette un disegno planare clusterizzato. 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 mentre la classe **NP** consiste 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 poichè non necessarie ai fini dell'argomento trattato.

1.2.2 flat clustered planarity

Prima di poter analizzare i progressi svolti nell'ambito del disegno di grafi clusterizzti è necessario dare una definizione di riduzione polinomiale che sarà poi impiegata in questa sezione. 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

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

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

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

può essere ridotto il *c-graph* C definito come C(G,T) in una istanza **equivalente** $C_f(G_f, T_F)$ in cui T_f è flat come espresso nel lemma che segue:

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 $C_f(G_f, T_F)$ in cui T è omogeneo, $lar(T)$ definita come la radice dell'albero ha almeno due figli e $h(T) <= n-1$

Mediante questa riduzione si può trasformare qualunque c-graph in un grafo in cui tutti i cluster sono nodi interni di un albero flat e e che essendo flat risulta essere anche omogeneo. Come visto risolvendo questo problema definito Flat Clustered planarity ed essendo Clustered Planarity riducibile polinomialmente a questo allora si risolverebbe anche esso Questa riduzione è stata poi implementata nel lavoro svolto e di possibile impiego per l'utente che andrà a lavorare sul sistema quindi sarà vista con uno sguardo più attento nel capitoli successivi.

Inoltre si vuol precisare che molte delle immagini esplicative utilizzate in questo scritto sono state prese direttamente dall'editor durante istanze di lavoro. Nel capitolo successivo verranno descritti gli strumenti utilizzati per la realizzazione per passare poi alle tecnologie e alle metodologie utilizzate con relativi esempi di impiego e motivazioni.

Capitolo 2

Tecnologie e metodologie

Di seguito sono riportate tutte le tecnologie utilizzate, ognuna accompagnata da una descrizione, che a volte comprenderà anche un esempio di utilizzo e la motivazione che ha spinto ad utilizzare tale tecnologia. Oltre questo sono anche riportate tutte le metodologie applicate per poter realizzare il progetto.

2.1 Visualizzazione delle informazioni

L'intero lavoro svolto si basa su due importanti settori informatici quali la teoria dei grafi che abbiamo detto essere la disciplina che si occupa dello studio dei grafi e spesso di consentirne delle analisi in termini quantitativi e algoritmici, con basi di teoria della complessità, e la visualizzazione delle informazioni. Per quanto concerne quest'ultimo, può essere definito 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 semplice scopo di creazione di grafici per scopi 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 e si stima che siano soltato il 20% dei dati che si producono e sono solitamente organizzati in tabelle. Quelli non strutturati sono invece difficile analizzare tanto che uno dei primi compiti è quello di riuscire a trasformare dati non strutturati in dati tabellari o dati flat. Per la rappresentazione dei dati che rappresentano la struttura del c-graph sono stati di notevole importanza i principi di questo ambito informatico che prevedono le seguenti operazioni precedenti alla 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

Di seguito vedremo le strategie adottate dal sistema per la rappresentazione del c-graph ricordando che esso è una coppia $< G, T >$ con inclusion tree T e underlying graph G.

2.1.1 rappresentazione inclusion-Tree

Per quanto concerne la rappresentazione dell'albero di inclusione , essendo un albero radicato essendoci un rapporto padre-figlio non binario ci sono due possibili strategie: 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 organizzazione ne è un esempio della rappresentazione HV drawing la stessa della **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.

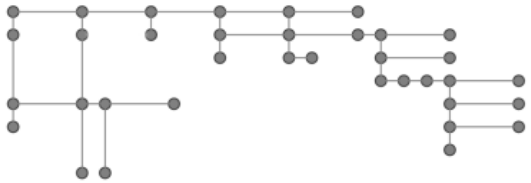

Figura 2.1: rappresentazione node-link HW drawing di un albero

Nella strategia Space-filling si tenta invece di superare i limiti del Node-link che 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 anche se anche questa rappresentazione presenta svariati problemi tra cui la distinzione difficile di tagli orizzontali da quelli verticali o la gestione non ottimale di alberi di grande profondità. Nella **Figura 2.2** sono mostrate alcune rappresentazioni diverse che seguono la strategia Space-filling.

Essendo quest'ultima di difficile lettura e soprattutto non ottimale per la rappresentazione di un albero collegato ad un grafo si è optato per la strategia Node-link scegliendo la rappresentazione "layered drawing",di cui ne è un esempio la Figuralayered,in cui i nodi sono organizzati in livelli e per ogni nodo la coordinata y dipende dal

Figura 2.2: rappresentazioni Space-filling

livello mentre la coordinata x deve esser trovata e dipende dal numero di figli del livello e dello spazio orizzontale a disposizione per la rappresentazione.

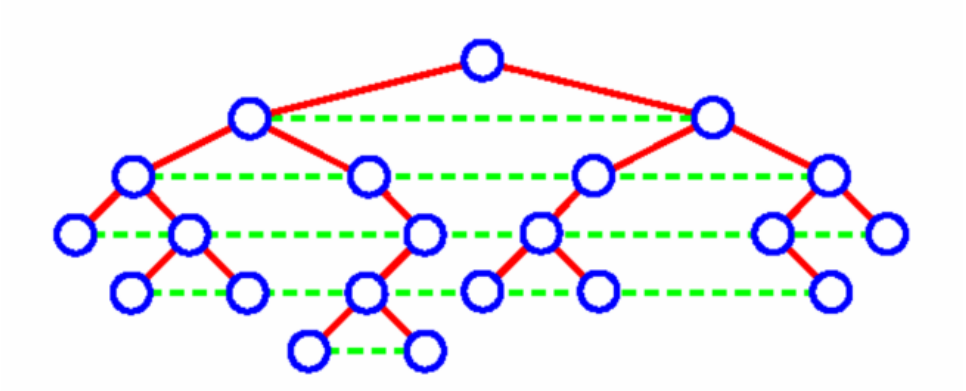

Figura 2.3: rappresentazione node-Link layered Drawing

2.1.2 rappresentazione underlying Graph

Avendo discusso della rappresentazione dell'inclusion Tree è necessaria la stessa attenzione per la rappresentazione dell'Underlying graph. Essendo un albero un grafo orientato senza cicli i modelli di rappresentazione sono gli stessi visti per gli alberi. Il modello utilizzato è il **Force directed**, modello principale utilizzato per la visualizzazione di grafi che utilizza una rappresentazione Node-link, che fa in modo di emulare 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. Ne esistono diverse varianti del modello Force directed e la scelta per quanto concerne il progetto in questone è ricaduta sul modello Spring Embedding, basato sulla combinazione di forze elettriche e forze meccaniche che andrà a formare una rappresentazione simile a quella mostrata nella **Figura 2.4** Per

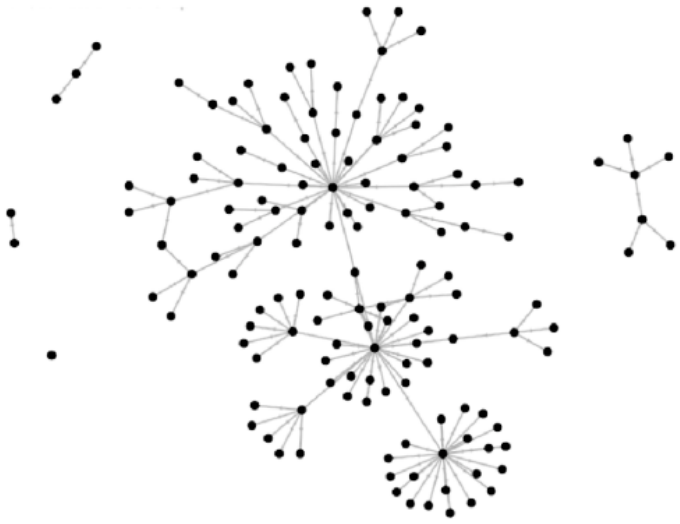

Figura 2.4: rappresentazione Spring Embedding

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.

2.2 Linguaggi e librerie

Avendo parlato della visualizzazione delle informazioni, è ancora da discutere su quale delle possibili tecnologie è stata utilizzata allo scopo della rappresentazione dinamica e user-friendly per poter creare l'editor. 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.

2.2.1 Javascript

JavaScript è un linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client. Essendo un sistema web con molti input da parte dell'utente a cui far fronte, Javascript consente un'ottima gestione di questi eventi. Le caratteristiche principali del linguaggio sono la debole tipizzazione, 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 con file con estensione .Json utilizzata proprio per l'import o l'export dei dati da rappresentare o della struttura dati della rappresentazione. Un altro punto a favore 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 questo linguaggio 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 **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.

2.2.2 D3.js

A supporto della scelta dell'utilizzo del linguaggio di scripting javascript vi è l'utilizzo della libreria utilizzata per la visualizzazione delle informazioni D3. 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 multipiattaforma e indipendente dalla lingua che tratta un documento XML o HTML come una struttura ad albero come mostrato nella **Figura 2.5** in cui ciascun nodo è un oggetto che rappresenta una parte del documento. Il DOM rappresenta un documento con 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; con loro 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.

2.3 Oggetti per la struttura dati

Inizialmente la struttura dati del progetto si basava su una unica struttura dati, in particolare un Array, contenente tutte le informazioni relative al c-Graph. La scelta è stata subito abbandonata a causa dei problemi riscontrabili con l'avere una struttura definibile quasi monolitica. Si è dunque passati ad una struttura

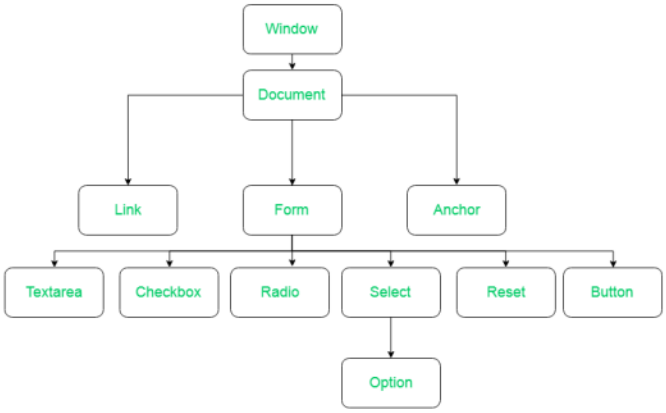

Figura 2.5: rappresentazioni Space-filling

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, sicurezza e 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, la map conserva l'ordine di inserimento e non ha alcun legame con la prototype chain.Per completezza si ricorda inoltre che un oggetto Map è un oggetto che 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* facili da iterare e i criteri di uguaglianza sulle chiavi della map sono più stretti di un normale === e hanno funzionamenti e prestazioni migliori in caso di frequenti aggiunte o rimozioni di valori. In particolare l'impiego degli oggetti sopra definiti e motivati nel caso in questione sarà discussa nel capitolo 4 in maniera molto approfondita.

Capitolo 3

Obiettivi e scelte progettuali

In questo capitolo, dopo aver preso nota delle tecnologie e metodologie utilizzate e descritte nel *Capitolo 2* ,si vedrà più in dettaglio lo studio dei requisiti e dei problemi risolti nel lavoro svolto,prima di passare alla descrizione della progettazione dell'applicativo.

3.1 Obiettivi

Gli obiettivi inizialmente pensati per essere realizzati durante lo svolgimento del lavoro,erano quelli di poter realizzare il primo editor in grado di:

I facilitare all'utente la creazione di dati inerenti ai grafi clustered;

II visualizzare questi grafi mediante diverse viste;

III caricare e salvare i propri lavori;

IV ridurre il c-graph in un grafo flat.

Per il conseguimento dell'obiettivo (I) sono state utilizzate le strutture dati viste nel capitolo 2 che hanno permesso 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. Essendo un editor creato per gestire una sola sessione di lavoro alla volta inoltre un altro obiettivo sarà quello di permettere all'utente in ogni momento di vedere le operazioni eseguite mediante una console richiamabile in qualunque momento. La possibilità di poter caricare e salvare i propri lavori (III) risulta essere necessario per consentire di riprendere una sessione di lavoro importante.
L'attività di definizione dei requisiti a breve termine è stata, piú che un punto di partenza,una fase ricorrente del lavoro alternata allo sviluppo vero e proprio dell'editor. Il paragrafo che segue raccoglie tutti i requisiti che sono stati individuati e le problematiche da risolvere prese in considerazione durante il lavoro.

3.2 Analisi dei requisiti e scelte progettuali

Gran parte dello studio rivolto all'analisi dei requisiti è stato svolto, come si addice ad un software sviluppato con **metodologia agile**, nelle prime iterazioni e riguarda tutte quelle esigenze descritte nel *Capitolo 1* e che poi hanno trovato un buon riscontro nelle tecnologie analizzate nel *Capitolo 2*. Di seguito in **Figura 3.1** è descritto in maniera molto semplificata il diagramma degli oggetti di dominio, risultato dall' analisi del progetto nella prima iterazione. Come si nota la libreria D3.js è stata fin da subito pensato come punto centrare per le visualizzazioni. Per rendere la suddetta accessibile e utilizzabile in pieno si è scelto di dare molte funzionalità all'utente mostrando dunque le operazioni di interesse ed oltre

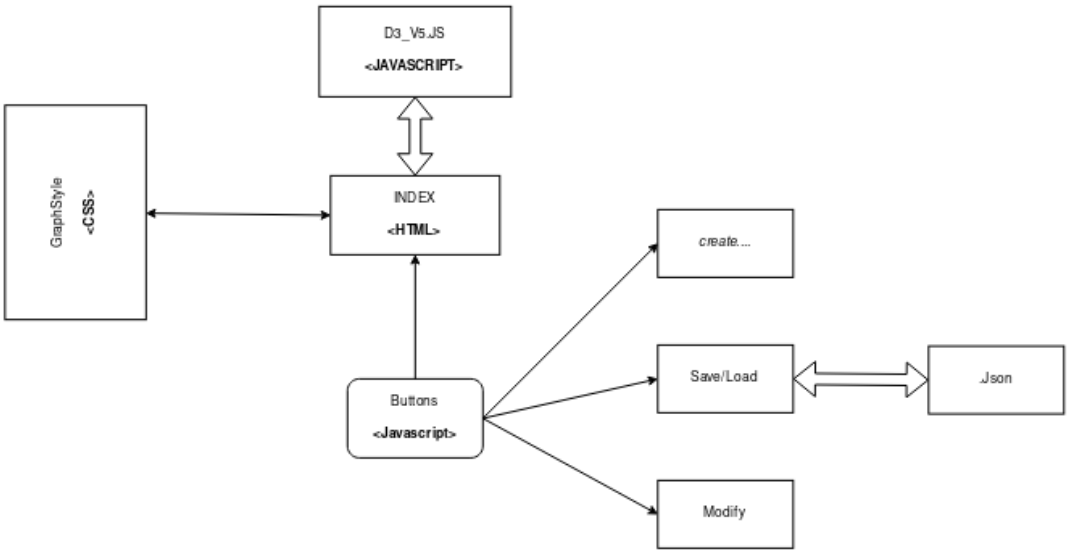

Figura 3.1: Prima iterazione degli oggetti di dominio

lasciando a lui la possibilità di inserire alcune condizioni di utilizzo che saranno mostrate nel capitolo successivo. I problemi che sono stati affrontati con maggior interesse riguardano due importanti fattori: la facilità di impiego e una visualizzazione che risultasse 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 3.2 seguendo i principi espressi dal libro "UML distilled. Guida rapida al linguaggio di modellazione standard"[Fow02] ed utilizzando **draw.io**, software di diagrammi online gratuito per la creazione di diagrammi di flusso, diagrammi di processo, organigrammi, ER e diagrammi di rete.

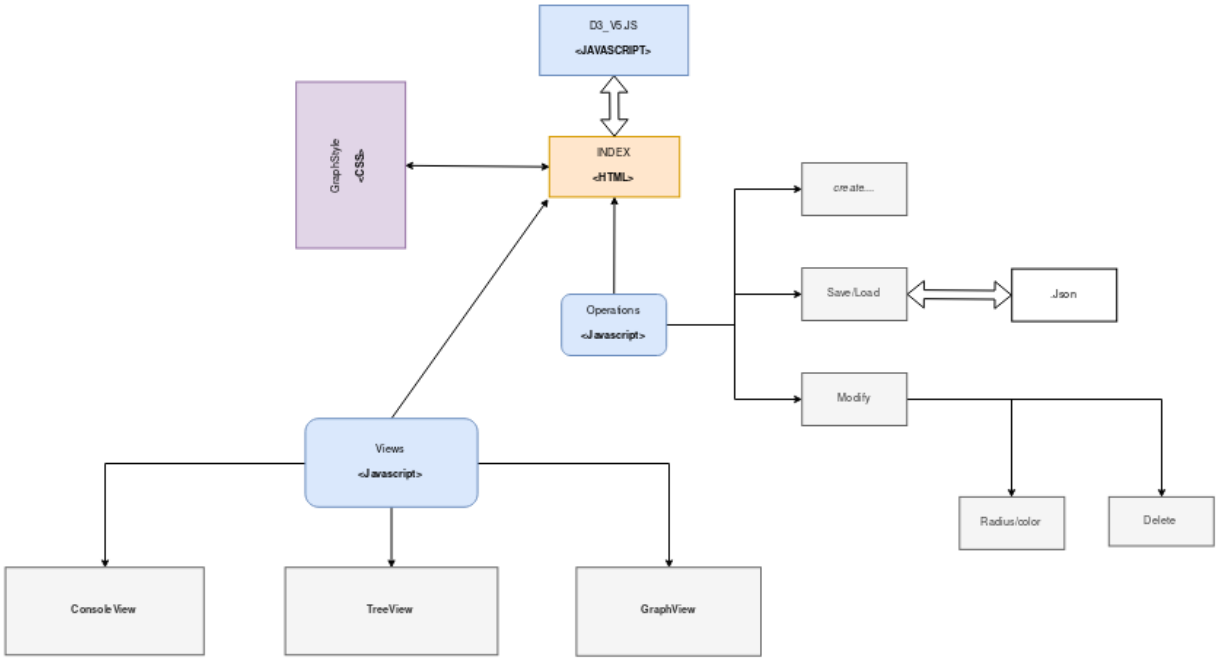

Figura 3.2: Seconda iterazione degli oggetti di dominio

Nell'ingegneria del software, UML, acronimo di unified modeling language ovvero di "linguaggio di modellizzazione unificato" è un linguaggio di modellazione e 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.

Tutto ciò che é stato descritto fino ad ora in questo capitolo 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. La progettazione è infatti una fase del ciclo di vita del software. 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.

3.3 Principi per la visualizzazione

Avendo definito nel capitolo precedente la visualizzazione delle informazioni è necessario, per comprendere gli obiettivi, evidenziare i principi per una visualizzazione 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 tutte questi principi del modello di Munzner inoltre esiste un rapporto di inclusione in cui la parte più esterna contraddistinta dal dominio a una più interna esiste un rapporto di molteplicità. In altre parole diversi idiomi possono corrispondere alla stessa astrazione e diverse astrazioni possono corrispondere allo stesso Dominio. Per il progetto sono stati di grande importanza i principi di Munzner e sono stati applicati con un approccio **Top-Down**, ovvero guidato dal problema e risolto dal dominio fino all'algoritmo applicando le scelte progettuali suddette. Per quanto riguarda invece le scelte di progetto, quali ad esempio se utilizzare colori nell'interfaccia o meno, se lasciare molti spazi bianchi o utilizzare molta grafica o se utilizzare la terza dimensione o meno, anche esse sono state decise partendo dai principi chiave e di Munzner spesso in contrapposizione con quelle di Edward Rolf Tufte.

Per concludere la digressione sui principi della visualizzazione è importante anche

definire il concetto di **User Task** come scopo dell'utente ovvero cosa l'utente deve poter fare sui dati. Nella visualizzazione delle informazioni ci sono due principali tipi di scopi: (i)Presentazione, contesto in cui si conosce l'informazione e la visualizzazione è tesa a rappresentarla e (ii) Analisi, contesto in cui la visualizzazione è tesa all'estrazione delle informazioni dei dati rappresentati. Per quanto concerne il progetto in questione la scelta progettuale su cui si basa 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 cambiare i dati e analizzarne contenuto supportando la ricerca mediante l'automatismo delle operazioni.

Capitolo 4

Primitive di interazione

Definendo la disciplina della visualizzazione delle informazioni come il mezzo per poter creare interfacce interattive per riuscire ad estrarre informazione dai dati ed avendo definito nel capitolo precedente i principi di questa particolare disciplina e alcune delle molteplici scelte applicabili è necessaria adesso una digressione sul concetto di interazione dell'utente e delle sue primitive.

4.1 definizioni dell'interazione

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 devono consentire 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 di manipolazione creando a sua volta nuove modalità di analisi

ed acquisendo consapevolezza della fondamentale importanza del fattore umano sulla macchina.

4.2 classificazione

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 un particolare sistema di visualizzazione. Mediante i fattori visti di seguito si potrà dunque dare una classificazione dell'interazione.

- **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.

Definite e classificate le interazioni mediante i vari fattori si può ora passare alle modalità di interazione e alle motivazioni che hanno portato a scelte utilizzate nel sistema.

4.3 modalità di interazione scelte e motivazioni

Per quanto riguarda la realizzazione del progetto essendo un editor completamente utilizzabile dall'utente sia per visualizzare che per definire strutture si è assegnato un grado di interazione pari a 3 rendendo il sistema trasformabile per consentire all'utente in qualunque momento durante la sessione di lavoro di poter intervenire e cambiare tutto ciò che è necessario ai fini di una buona creazione e visualizzazione. Ritornando ai tempi di risposta invece nella maggior parte delle operazioni l'utente potrà gestire interazioni con tempi dell'ordine dei 10^-1 secondi fatta eccezione per alcune operazioni che necessariamente dovranno far uso di dialoghi introduttivi o di finestre di dialogo per poter proseguire con la visualizzazione. Lavorando con le tecnologie Web e dovendo avere un piano di lavoro di una grandezza superiore a quella di un tablet o di uno smartphone il sistema è stato pensato per poter essere utilizzato su qualunque desktop o laptop avendo quindi la possibilità di eseguire aziondi mediante l'utilizzo di bottoni cliccabili con l'ausilio di un mouse potendo quindi contare sulla velocità delle interazioni, dell'ausilio di un piano di lavoro di dimensioni considerevoli e del grado più alto di interazione si definiranno ora le **primitive** che l'utente potrà utilizzare partendo da quelle definibili basilari.
Le operazioni più semplici che vedremo nel dettaglio sono quelle di select con la quale l'utente seleziona un certo numero di punti da rappresentare, panning nel quale utilizza lo scrool per navigare i dati e lo zooming sia spaziale che logico.

Per quando concerne l'operazione di **select**, ricordando che un grafo clusterizzato è composto da nodi, cluster ed archi, si è scelto di dare all'utente la possibilità di selezionare e visualizzare anche solamente una dei tre insiemi di cui è composto il grafo ma senza avere la possibilità di rappresentazione di un sottoinsieme di uno di loro. È possibile quindi selezionare tutti i nodi di un grafo senza visualizzare

gli archi o i cluster di cui è formato ma non è possibile selezionare esclusivamente parte di quei nodi.

Lo **zooming** ovvero la possibilità di effettuare un ingrandimento o un ridimensionamento

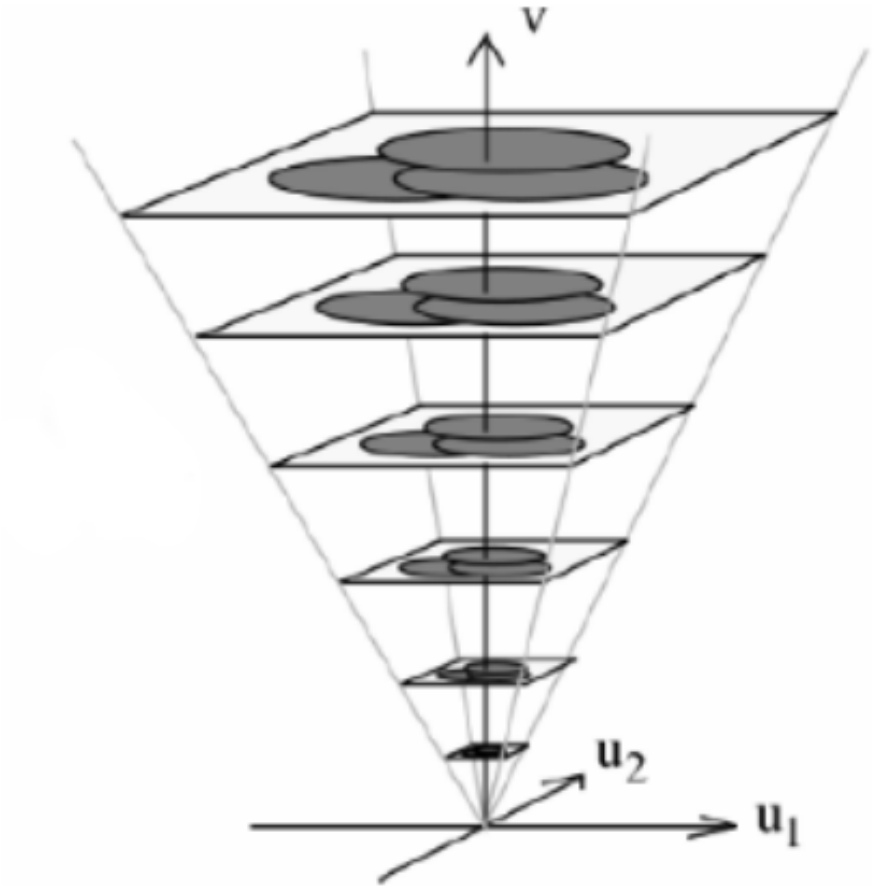

Figura 4.1: Operazione di zooming su un piano $< x, y >$

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 and panning. 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 4.1. La distanza dal piano x-y determina il grado di zooming mentre il movimento sul piano x-y determina il panning.
Quando tengo 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, per esempio dove è posizionato il puntatore del mouse, gli oggetti inclusi dalla finestra diminuiscono. Inoltre un 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 vdere 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 4.2

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.
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

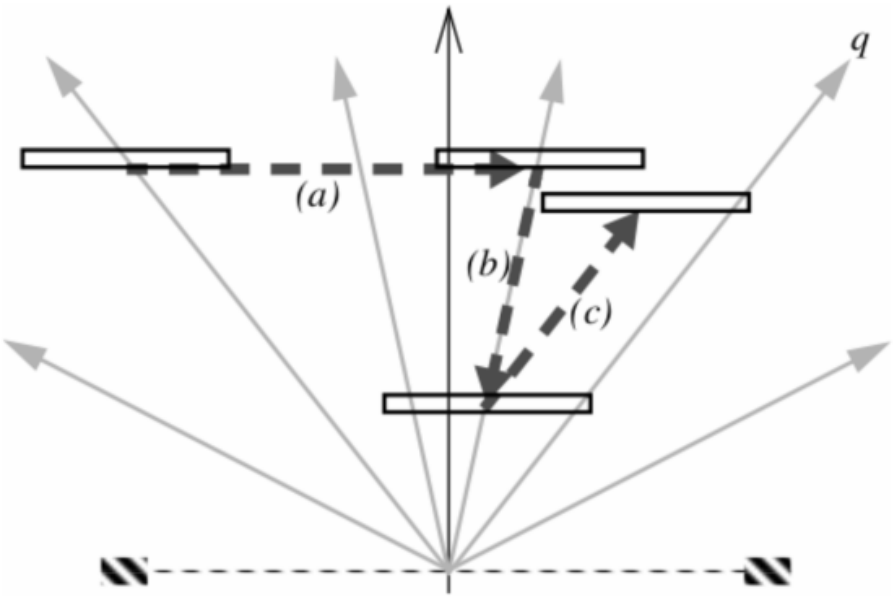

Figura 4.2: Operazioni di panning e zooming su un piano $< x, y >$

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 4.3, da una rappresentazione a diagramma a barre ad una torta. Include anche operazione di modifica sui colori, sulle forme e l'orientamento della visualizzazione. Per quanto riguarda la realizzazione del sistema in dettaglio, la primitiva che risulta 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

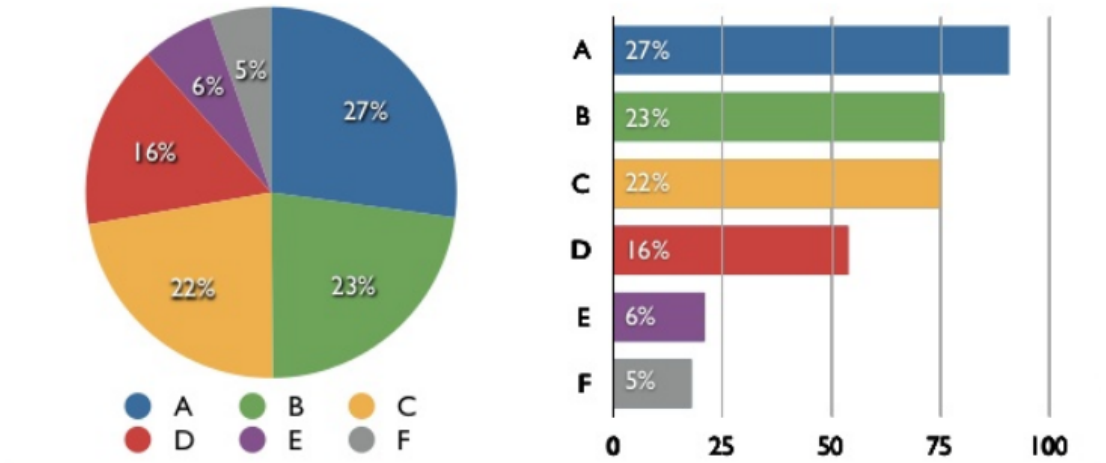

Figura 4.3: Esempio di encoding da un diagramma a torta ad uno a barre

connessione. Riuscire infatti a dare all'utente la possibilità 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. Il filtraggio può esser quello che, nella realizzazione del caso di studio, riguarda la possibilità di definire e differenziare i dati e le visualizzazioni dei nodi e dei cluster in quanto come visto nel capitolo relativo allo stato dell'arte presentano sostanziali differenze tra loro. Tutte le primitive di visualizzazione troveranno poi la loro realizzazione mediante una operazione dedicata come si vedrà nei capitoli successivi.

Capitolo 5

Interfacce e struttura dati

5.1 Struttura dati

Prima ancora di mostrare l'interfaccia e le scelte progettuali applicate 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. Ritornando al capitolo relativo allo stato dell'arte, si è definito un grafo clusterizzato C 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. Alla base di questo e seguendo quanto detto nel capitolo due relativo agli oggetti scelti per la struttura dati, vi è la rappresentazione del grafo clusterizzato così come mostrato nella figura Figura 5.1. Un oggetto *clusteredGraph* è definito mediante un oggetto relativo all'underlying graph ed uno relativo all'inclusion tree.

Come si nota un oggetto *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. Andando poi a ritroso nelle definizioni, nella figura Figura 5.2 è

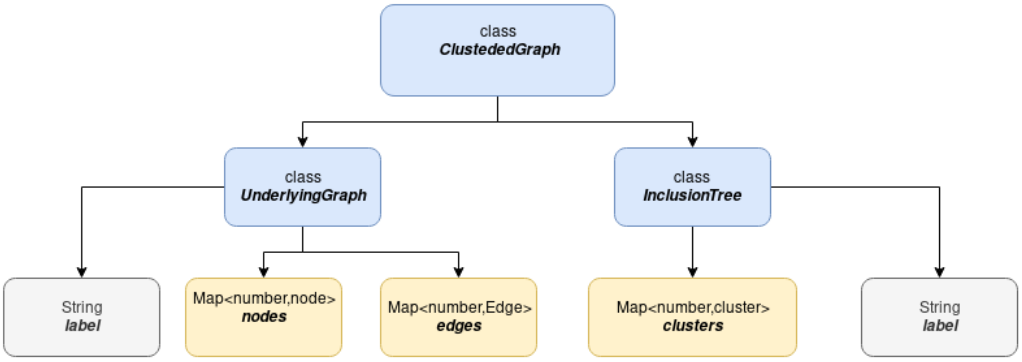

Figura 5.1: Oggetto clusteredGraph

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* 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 5.3 è definita mediante una stringa e tre valori numerici. La prima si riferisce all'etichetta dell'arco e non necessariamente dovrà essere

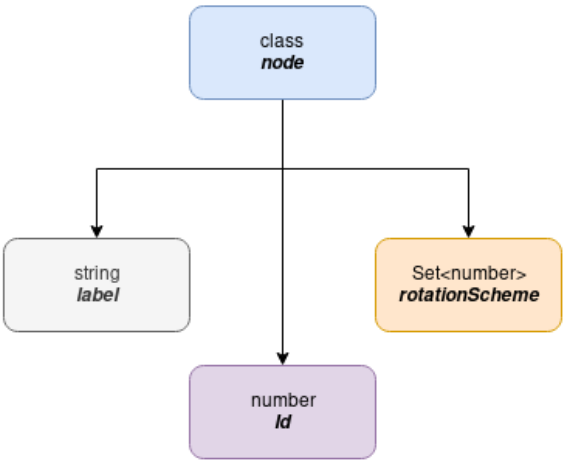

Figura 5.2: Oggetto nodo del underlying graph

unica, i valori numerici invece fanno riferimento all'id dell'arco, 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.

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 5.4 è definito mediante un costruttore con cinque attributi di seguito

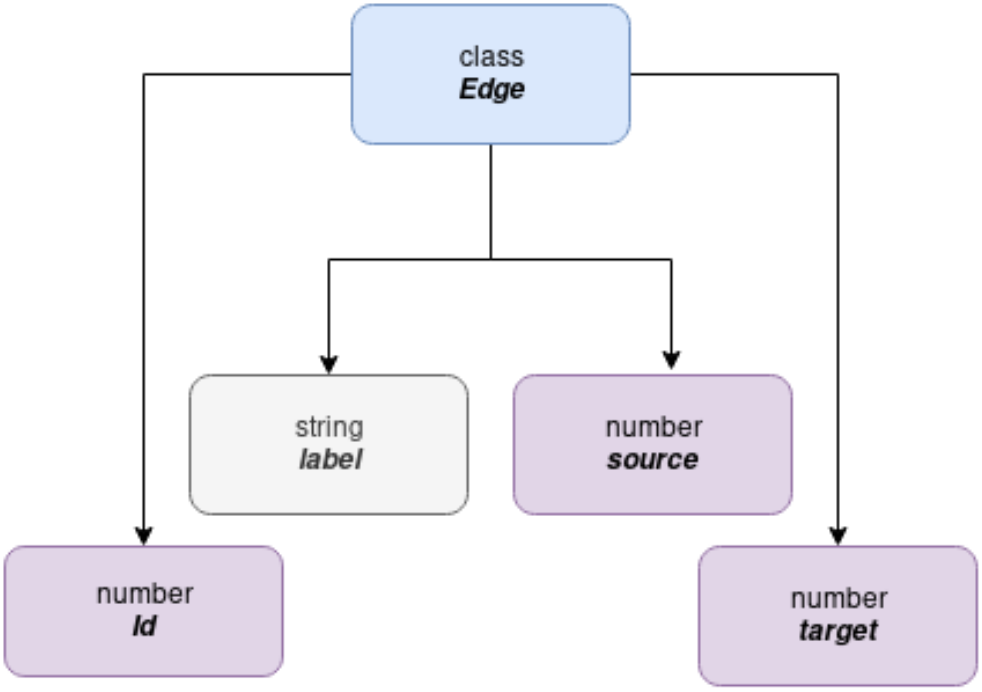

Figura 5.3: Oggetto arco dell'undelying graph

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

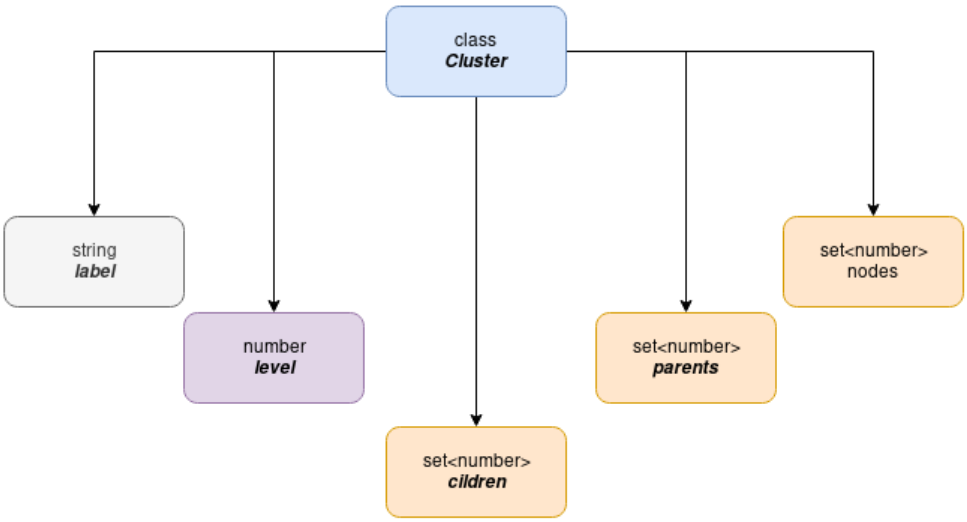

Figura 5.4: Oggetto cluster dell'inclusion Tree

5.2 Log-in ed inizializzazione

Al momento del log-in iniziale l'utente si troverà la schermata mostrata nella Figura 5.5. 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 poi grande spazio per quanto concerne il piano di lavoro principale che come si vedrà successivamente rimarrà inalterato anche cambiando l'interfaccia utilizzata. Anchel'impiego di colori caldi quale 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 già accennato sarà

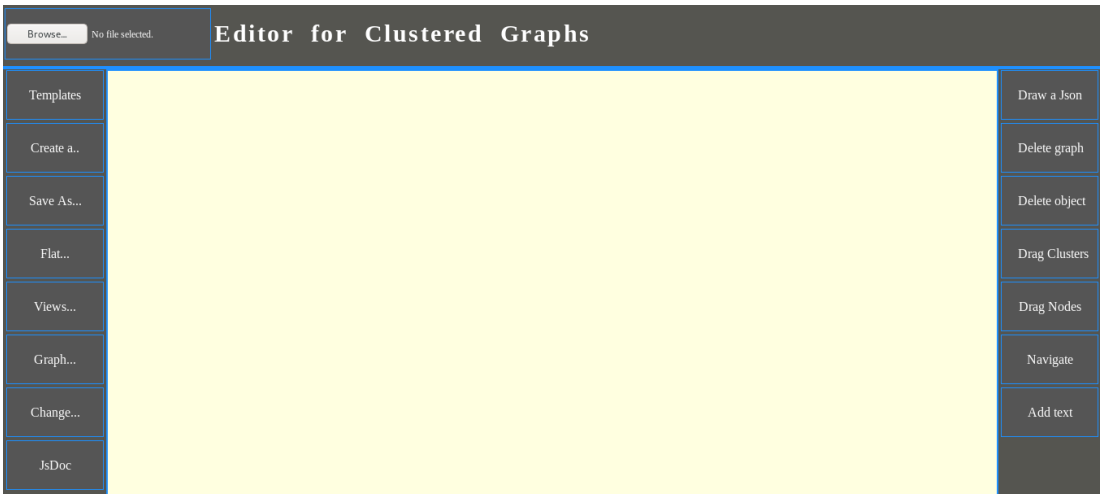

Figura 5.5: Log-in nel sistema

possibile per l'utente interagire con il sistema per poter effettuare operazioni di encoding dei dati. 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 tuo ciò che ne comprende e 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-iin esso 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 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 di piani di lavoro diversi. In particolare il sistema essendo basato su una coppia $< G, T >$ che compongono il grafo clusterizzato G 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 5.6,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.

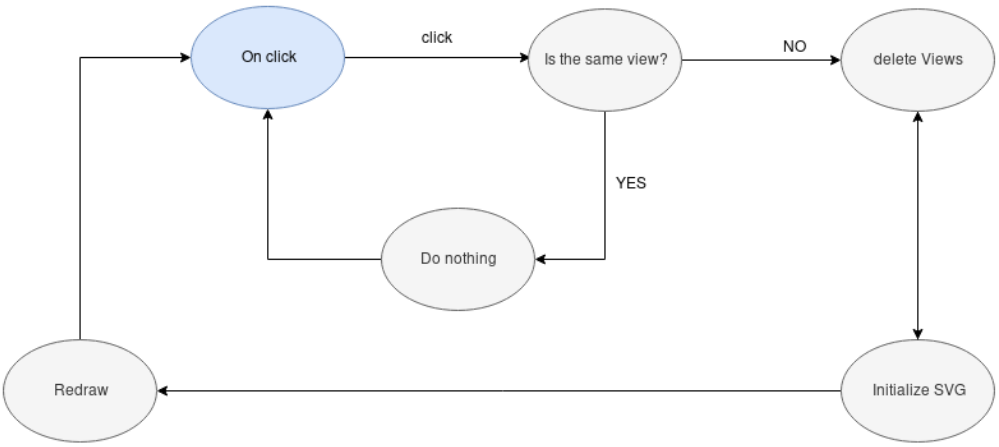

Figura 5.6: Struttura dell'algoritmo dell'operazione di encode

5.3 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 iterazione 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.

Come si può notare dalla Figura 5.7 il modello di visualizzazione come già accennato riprende quello Node-link in cui "Node" rappresenta 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 esso risulta 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 sostanza. 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 sarà visto nel dettaglio ne capitolo successivo.

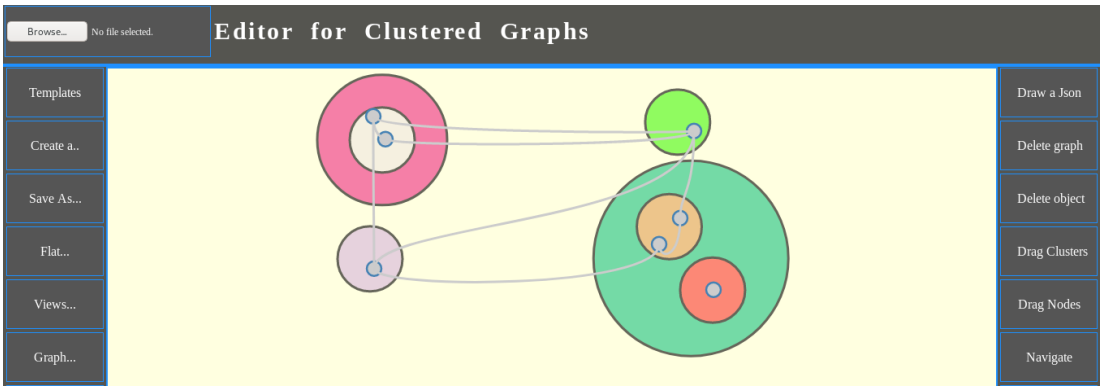

Figura 5.7: Graph View

5.4 Tree-view

Non avendo un limite al numero di nodi e di cluster visualizzabili la graph view può ad un primo impatto portare a visualizzare un gran numero di informazioni. È facile intuire come l'occhio umano veda una rappresentazione organizzata in maniera notevolmente migliore rispetto ad una piatta e prima di profondità anche se fittizia. Per questo l'utente in qualunque momento durante una sessione di lavoro potrà interagire con il sistema e passare da una visualizzazione a grafo clusterizzato ad una dell'inclusion tree in cui però saranno rappresentati anche i nodi e gli archi dell'underlying graph. Ogni volta che si chiederà questa operazione il sistema risponderà eliminando la visualizzazione e inizializzandone una nuova con un encode diverso degli stessi dati rappresentati in precedenza. Questa

visualizzazione inizializzata 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 poichè non necessari.

Al contrario che nella graph view, questa risulta comunque essere un modello node-link ma in cui non si avrà necessità di inserire forze per una rappresentazione ottimale in quanto sarà una semplice rappresentazione "layered" con l'aggiunta di archi che collegano le foglie tra loro come mostrato nella Figura 5.8 che contiene lo stesso grafo visualizzato in precedenza nella graph-view e visualizzato nella figura Figura 5.7. Come si nota inoltre il raggio dei cluster e dei nodi risulta essere la stessa anche se ciò che caratterizza una foglia, che sia essa un cluster vuoto oppure un nodo all'interno di un cluster, è che questa sarà di un 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 anche se 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 è relativamente bassa mentre è sicuramente preferibile utilizzare una visualizzazione orizzonante con la radice centrale a sinistra dell'svg nel caso in cui si hanno pochi cluster per ogni livello ma una profondità maggiore da gestire.

5.5 Console-view

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

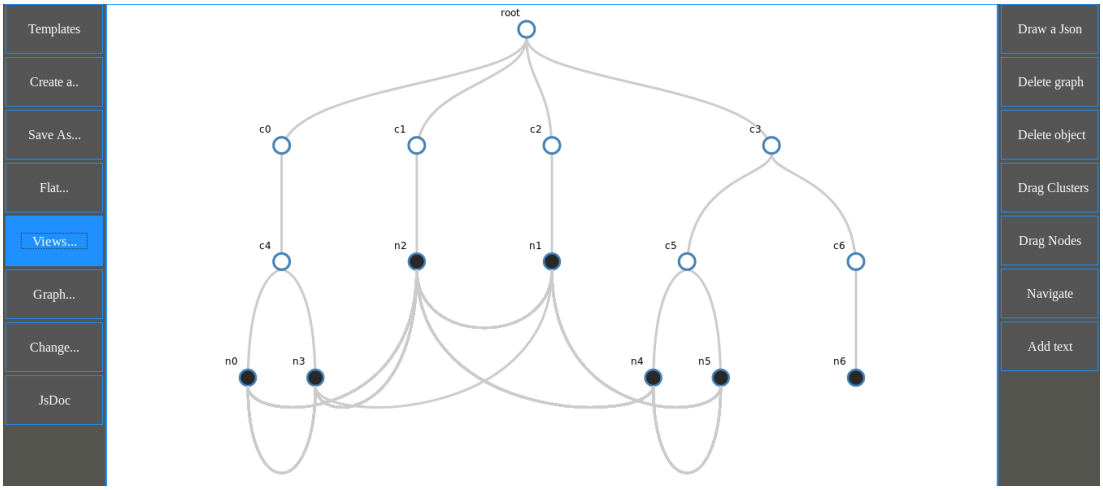

Figura 5.8: tree View

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 similari alle altre due ed in cui saranno inseriti i messaggi delle ultime n operazioni eseguite. Ogni operazione infatti nel momento in cui viene richiesta dall'utente nel mentre viene eseguita salva un valore di tipo String all'interno di un oggetto Map<number,String> *logs* inzializzato al log-in iniziale dell'utente. La vista sarà similare a quella mostrata nella Figura 5.9, nella quale oltre al messaggio che ricorda l'operazione eseguita e richiesta, 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

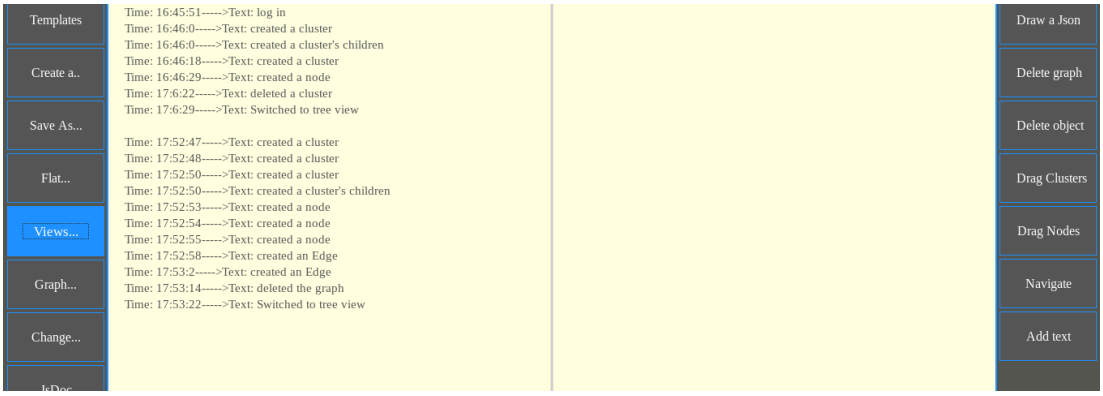

Figura 5.9: Graph View

5.6 Filtraggio degli oggetti

Come definito nel capitolo precedente 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 una maggiore importanza 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 Figura 5.10 che fa riferimento alla Figura 5.7 in cui però l'utente ha necessità di vedere solamente la disposizione dei cluster nel dettaglio. Essendo un grafo clusterizzato un grafo planare, per 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 sarebbe di difficile analisi questa esigenza. Per questo facendo riferimento alla Figura 5.7 l'utente può avere accesso ad una visualizzazione filtrata sugli archi come mostrato in figura Figura 5.11. Partendo

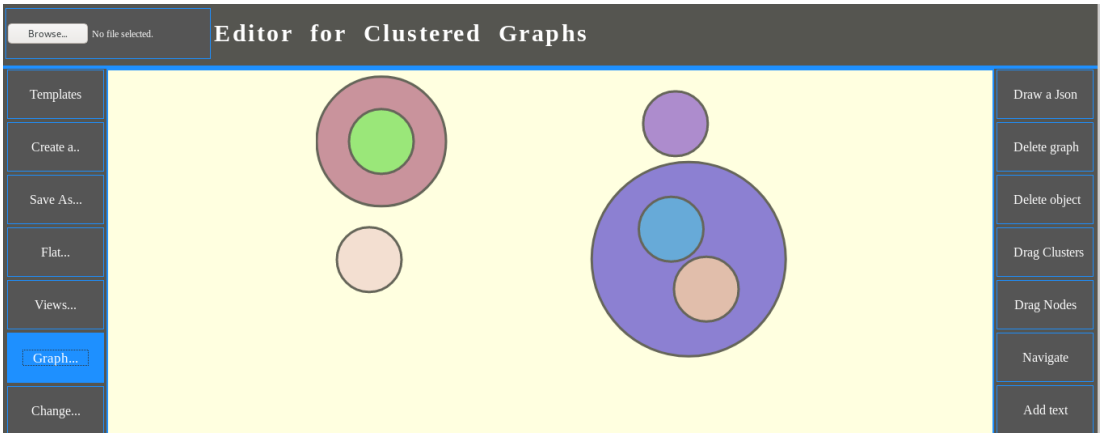

Figura 5.10: visualizzazione dei soli cluster

dalla visualizzazione *graph-view* che come detto dona all'utente un grado di integrazione massimo, 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 esattamente come visto per l'operazione di encode che porta alla *tree-view*, anche se è sempre possibile tornare alla rappresentazione di default per poter continuare la sessione di lavoro.

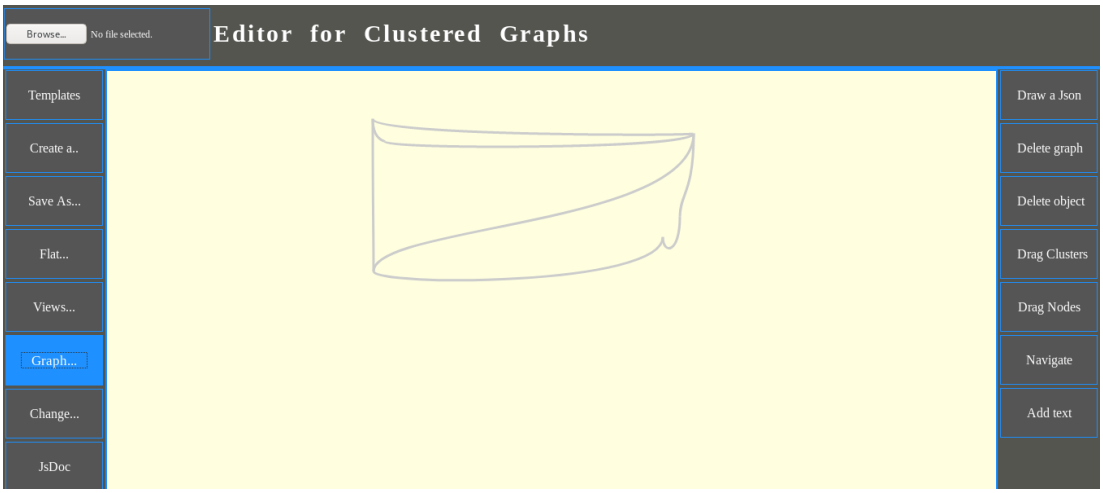

Figura 5.11: Visualizzazione dei soli archi

Capitolo 6

Funzioni utente

Di seguito saranno viste le varie operazioni eseguibili dall'utente sull'editor ad eccezione per la riduzione da grafo clusterizzato a grafo flat a cui è stato riservato il capitolo successivo. 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;

Prima di passare alla descrizione dettagliata delle varie operazioni è necessario un focus dettagliato 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.

6.1 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 6.1 il sistema passerà ad effettuare una operazione di disegno dei dati per poi dare di nuovo il controllo all'utente. Quando questa operazione viene eseguita il sistema

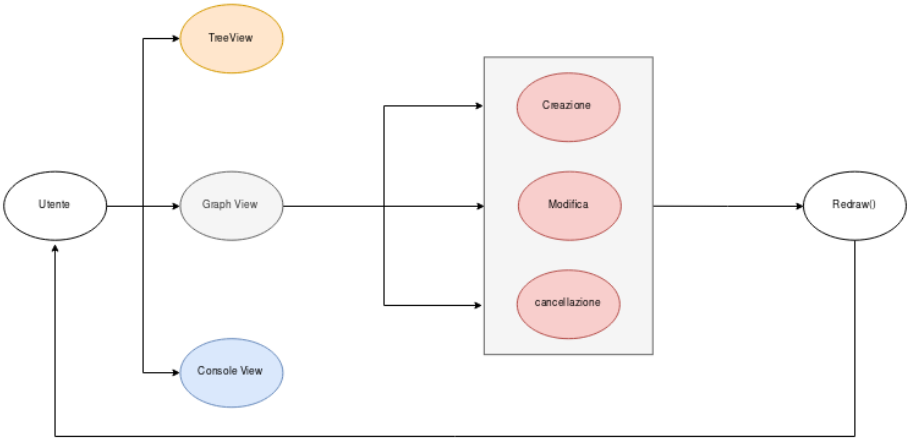

Figura 6.1: Schema dell'impiego della funzione di disegno dei dati

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 forse fisiche che controlleranno il movimento e la rappresentazione degli oggetti. In particolare questo viene eseguito 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 forse sono mostrate schematicamente nella figuraFigura 6.2 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.
Una volta definite le forze ed utilizzate l'algoritmo di ridisegno termina lasciando all'utente la possibilità di continuare la sessione di lavoro. Una volta definita l'operazione di disegno degli oggetti da visualizzare è ora possibile procedere al chiarimento delle funzioni utente negli ambiti della creazione e della modifica degli elementi e della navigazione all'interno della visualizzazione.

6.2 creazione

6.2.1 import/export

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. In particolare come visto l'import può essere eseguito mediante dati definibili tabellari provenienti da file

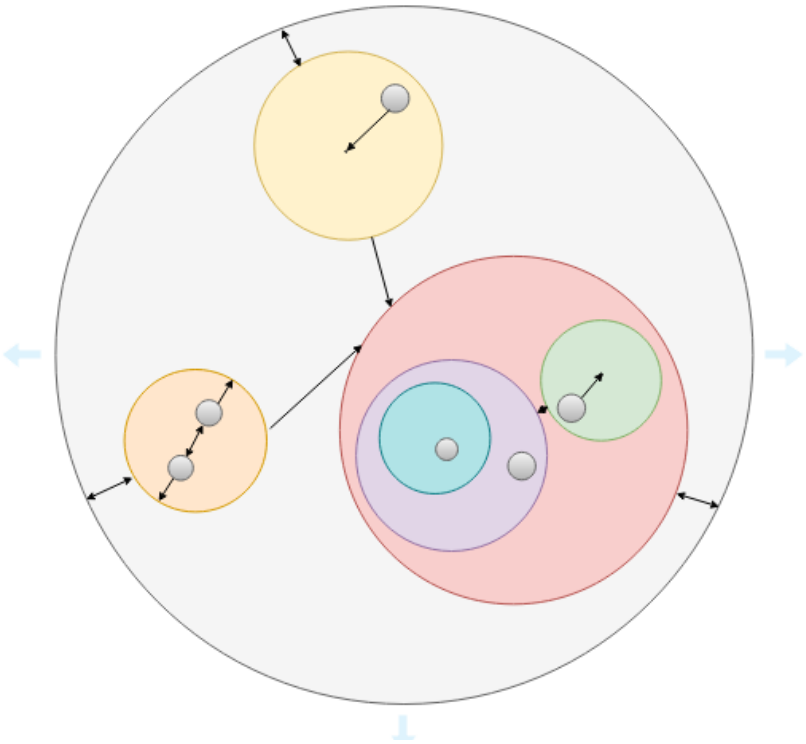

Figura 6.2: Esempio delle forze attrattive e repulsive utilizzate in fase di disegno

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. Inoltre risulta essere di facile analisi per il calcolatore e di facile generazione. Si basa su un sottoinsieme dello standard di programmazione JavaScript ECMA-262 3a edizione - 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 varie lingue, questo viene realizzato

come oggetto, record, struct, dizionario, tabella hash, elenco con chiave o array associativo;

- Un elenco ordinato di valori. Nella maggior parte delle lingue, questo è realizzato come una matrice, un vettore, un elenco o una sequenza.

Mediante un bottone di import è dunque possibile caricare un file json personale creato secondo la struttura fissa mostrata nella Figura 6.3e passare questi dati al sistema per crearne la rappresentazione associata.

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


```
{
        "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 6.3: Esempio di file json per l'import di un grafo clusterizzato

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.

6.2.2 template

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 5.6. 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 alla creazione della pagina di lavoro il sistema presenterà, come visto nella Figura 5.5, 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.

6.2.3 elemento del grafo

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 è 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 di essi e dia al sistema l'input per poter cominciare la creazione di archi esso risponderà con il messaggio di errore mostrato in Figura 6.4. Ogni volta che l'utente creerà un oggetto all'interno

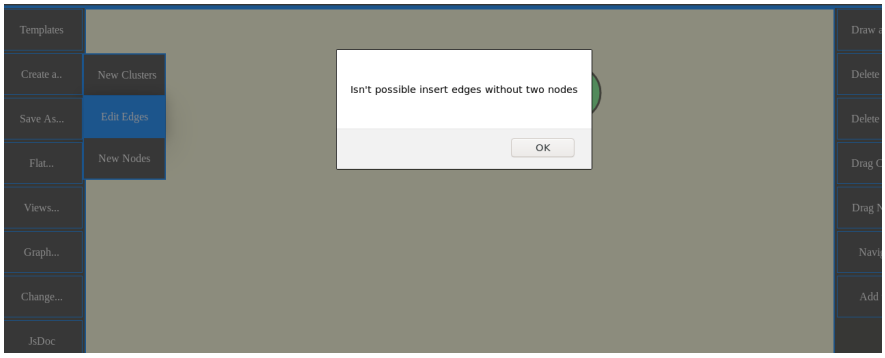

Figura 6.4: Messaggio di errore per il tentativo di creazione di un arco senza due nodi

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 5.4, al cluster genitore. Un cluster inoltre nella graph view ha un raggio r_c dipendente dal numero di oggetti al suo interno. È possibile definire

questo raggio come:

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

$$\forall n >= 5$$

$$\forall f >= 0$$

con k_c definito come un valore di default,n come il numero di nodi all'interno del cluster e f come il numero di figli del cluster. Per quanto riguarda i nodi invece essi saranno rappresentati con un raggio fisso k_n non avendo al loro interno oggetti. 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 6.5 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.

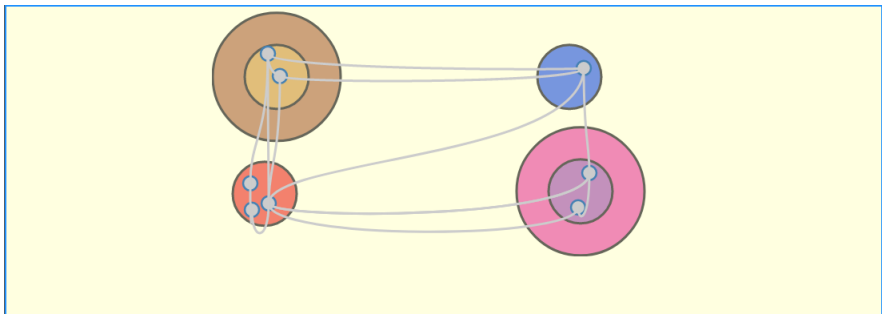

Figura 6.5: Esempio di planarità nella rappresentazione di un grafo clusterizzato

6.3 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.

6.3.1 Spostamento e cancellazione

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. Inoltre il sistema spostando ad esempio un nodo, durante l'operazione successiva provvederà a spostare gli archi ad esso collegati in maniera automatica anche se un nodo può essere mosso solamente all'interno del cluster di appartenenza poiché esso non cambierà la struttura dati alla base ma solamente le sue coordinate nella visualizzazione. Essendo l'utente finale non esente da possibili errori di creazione degli oggetti è stata realizzata una funzione per la cancellazione di oggetti. Al contrario che nelle funzioni di spostamento di un oggetto in questo caso si avranno effetti diversi a seconda dell'oggetto cancellato, che sarà eliminato anche dalla struttura dati alla base della rappresentazione, in sintesi eliminando:

- un nodo: verrà eliminata la sua visualizzazione, quella di tutti gli archi che avevano l'id di quel nodo come valore *source* o come valore *target* e l'id del nodo verrà eliminato dalla lista dei nodi del cluster di appartenenza;
- un cluster: verrà eliminata la sua visualizzazione,tutti i nodi con l'id uguale a quelli appartenenti al cluster e nel caso in cui il cluster fosse di livello $l > 1$ allora sarà eliminato dalla lista dei figli del cluster genitore e ridotto il raggio dello stesso essendo dipendente anche dal numero di figli.

Si riporta, in maniera semplificata, nella Figura 6.6 l'algoritmo di eliminazione di un oggetto.

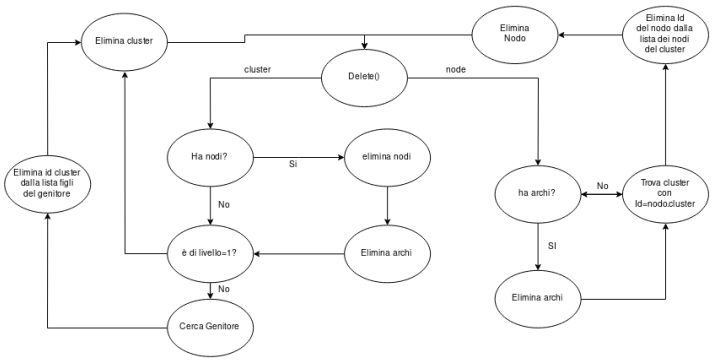

Figura 6.6: algoritmo di cancellazione di un oggetto

6.3.2 raggio e colore degli oggetti

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 soprattutto a poter avere un maggiore spazio a disposizione nel caso in cui il grafo raggiunge dimensioni considerevoli ma anche di avere un maggior dettaglio nel caso inverso in cui un grafo presenta

ad esempio un numero limitato di cluster ed un significativo numero di nodi per ognuno di essi. Ad esempio lo stesso grafo clusterizzato riportato nella Figura 5.7 andando a dimezzare la dimensione solamente dei suoi cluster e eseguendo qualche piccolo accorgimento per quanto concerne lo spostamento degli oggetti come è riportato nella figura Figura 6.7 come sia possibile avere a disposizione molto più spazio anche lavorando su un piano di lavoro delle stesse dimensioni di quello utilizzato in precedenza. Come già accennato in precedenza ai cluster è assegnato

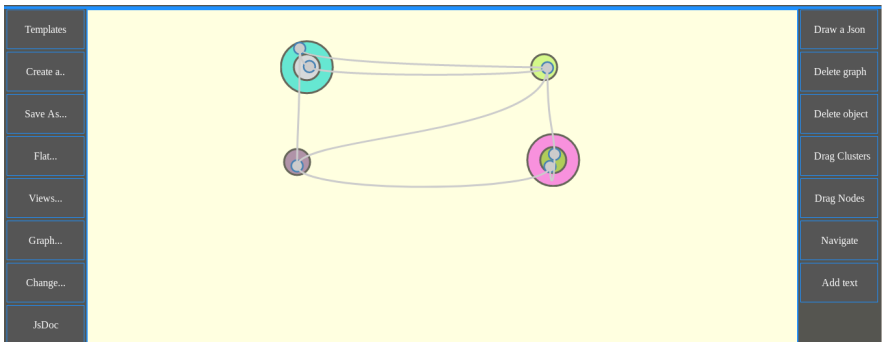

Figura 6.7: operazione di cambiamento del raggio dei cluster da parte dell'utente

un colore. Questo è di notevole importanza essendo un principio fondamentale per la rappresentazione come espresso anche nel capitolo sulle primitive della visualizzazione. Al colore di ogni cluster di default 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 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 esadecimale fissi per poter avere una scala di colori che richiami solo la tonalità richiesta

dall'utente. Riprendendo nuovamente il grafo mostrato nella figura Figura 5.7 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 6.8. In questo modo è possibile anche catalogare e

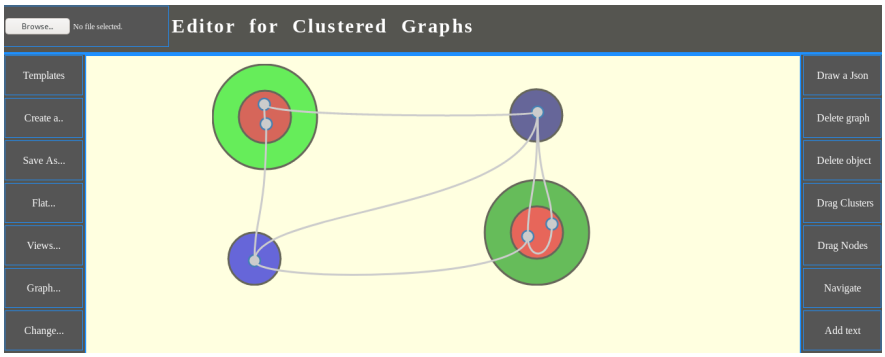

Figura 6.8: impiego di Palette diverse da quella di default da parte dell'utente

categorizzare i cluster in base alla loro colorazione. Per concludere la sezione sulle possibili operazioni di modifica inerenti al raggio e alla colorazione degli oggetti 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. Si ricorda che il colore può essere definito anche con un valore esadecimale come visto anche nel paragrafo inerenti alle primitive per la visualizzazione.

6.3.3 aggiungere descrizione agli oggetti

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 quell'oggetto per l'utente che andrà a realizzarlo o ancora bisogna lasciare dei brevi commenti er 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 o aiuti specifici per un particolare oggetto. 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 o sarà eliminato. Un esempio di utilizzo, facendo ancora una volta riferimento alla Figura 5.7 del capitolo precedente, potrebbe essere quello riportato nella Figura 6.9. Nella figura

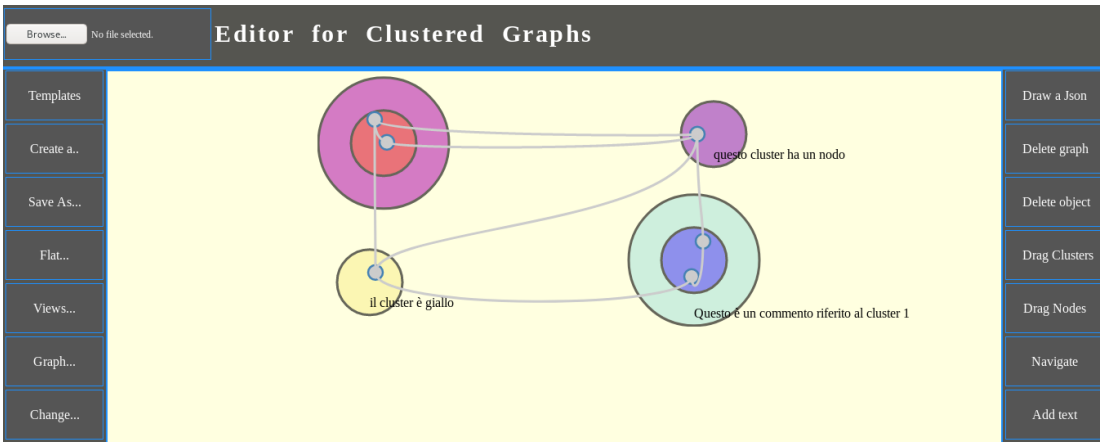

Figura 6.9: aggiunta di commenti da parte dell'utente

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.

6.4 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 esempio 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 di focus su un particolare elemento del grafo dovrà prima richiedere al sistema di entrare in una modalità che può essere definita "*navigate-view*" in cui ad un primo sguardo non si avranno grandi differenze. Entrando per la prima volta durante una stessa sessione all'interno della *navigate-view* sarà pero visualizzato un messaggio di aiuto per l'utente come mostrato nella figura Figura 6.10 che indicherà le operazioni eseguibili in questa modalità per poter dare anche se in minima parte un consiglio ed un aiuto su come agire.

Spostandosi sul piano di lavoro sarà possibile eseguire le primitive di traslazione e di zooming in/out dell'intera rappresentazione del grafo clusterizzato come mostrato nella Figura 6.11 in cui si sono eseguite le operazioni di traslazione e zoom-In rispetto alla rappresentazione della Figura 5.7. Inoltre in questa modalità di visualizzazione passando sopra un oggetto sarà evidenziato a schermo per dare più attenzione e focalizzazione su di esso. In particolare questo focus dipenderà dal tipo di oggetto evidenziato:

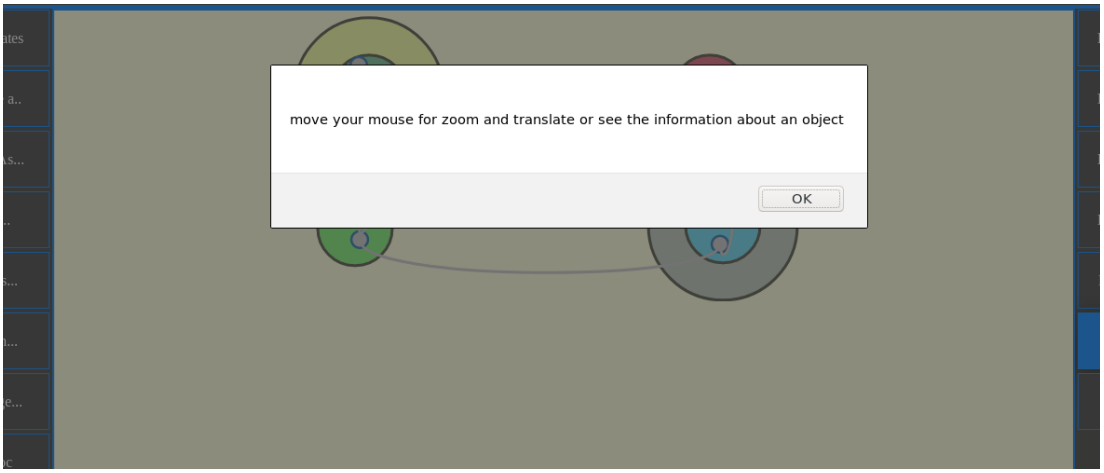

Figura 6.10: Messaggio di aiuto al primo accesso dell'utente alla navigate-view

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

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

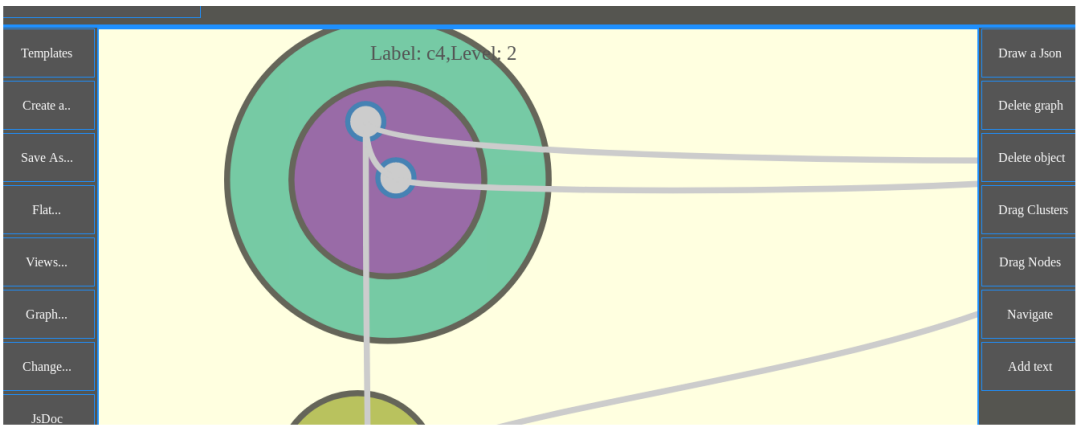

Figura 6.11: zooming di un grafo nella navigate-view

Capitolo 7

Riduzione Flat del grafo clusterizzato

Di seguito, come anticipato nel capitolo precedente, sarà analizzata l'operazione di riduzione eseguibile mediante una interazione dell'utente con il sistema. Data la sua importanza si è scelto di trattarla in capitolo separato rispetto alle altre operazioni eseguibili. Per quanto riguarda i concetti introduttivi e le definizioni varie che saranno applicate in questo capitolo si rimanda al capitolo relativo allo stato dell'arte.

7.1 metodo

Ritornando a quanto visto nello stato dell'arte una istanza $C(G,T)$ di un c-graph, che nel nostro caso è rappresentata dall'oggetto *clusteredGraph* di cui l'utente ne gestisce dati e visualizzazione, con n vertici e c cluster può essere ridotta in tempo $O(n+c)$ in una istanza equivalente $C_f(G_f,T_F)$ in cui T è omogeneo, la$r(T)$ definita come la radice dell'albero ha almeno due figli e $h(T) <= n-1$.
In altre parole l'obiettivo è quello di poter rappresentare un qualunque albero

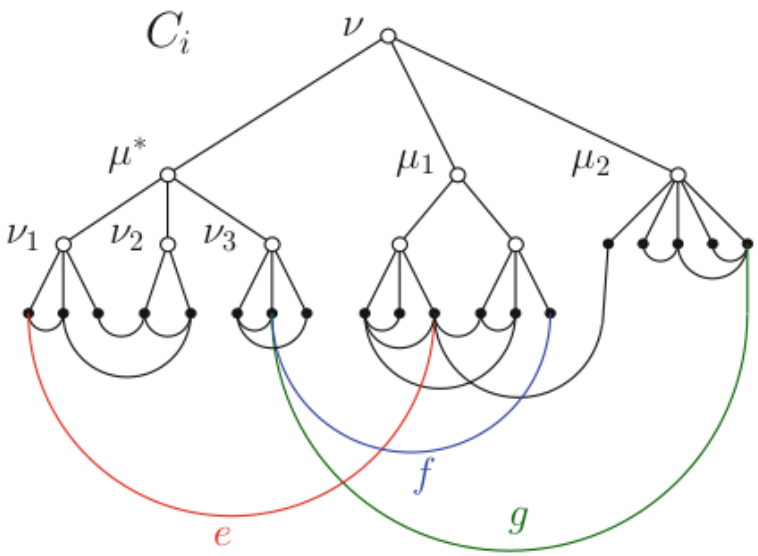

Figura 7.1: C_i con albero di inclusione T omogeneo

di inclusione con una profondità d in un albero in cui la profondità massima è uguale a due, tutti i nodi interni sono cluster e tutte le foglie sono archi senza avere una perdita di informazioni intesa come numero di nodi o collegamenti tra di essi. 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 omogeneo T_i,ogni trasformazione richiede $O(n)$ tempo e $S(T)$ è definito come la dimenzione dell'albero di inclusione T, ovvero il numero di nodi superiori di T diversi dalla radice.

Prendendo come mostrato in figura Figura 7.1 un grafo clusterizzato nella sua rappresentazione "layered" è possibile vedere come esso rispetti le condizioni

imposte in quanto l'albero di inclusione risulta omogeneo. È possibile dunque costruire $C_{i+1} =< G_{i+1}, T_{i+1} >$ come segue ottenendo la rappresentazione mostrata nella figura Figura 7.2:

- 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.

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. È inoltre possibile visualizzare la riduzione senza dover necessariamente utilizzare l'albero di inclusione. Partendo da una rappresentazione node-link con il metodo Spring-embedding del grafo cluterizzato ed in particolare soffermandosi solamente sul dettaglio del cluster μ^* di livello $l > 2$ visualizzato in Figura 7.3 è possibile eseguire una visualizzazione post-riduzione, 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. In questa rappresentazione che è mostrata nella Figura 7.4 però

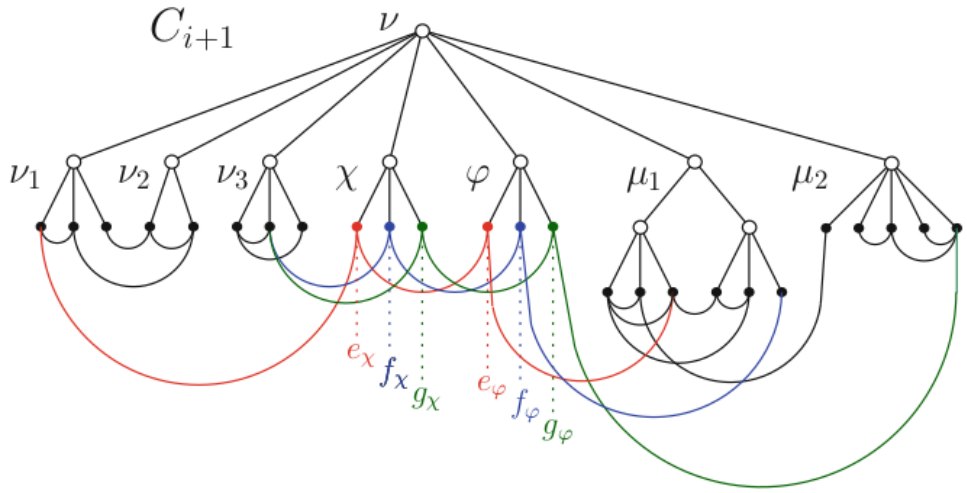

Figura 7.2: C_{i+1} ottenuto dopo parte del processo di riduzione

i cluster aggiunti, 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^* 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.
- $S(Ti + 1 = S(T_i) - 1$, 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

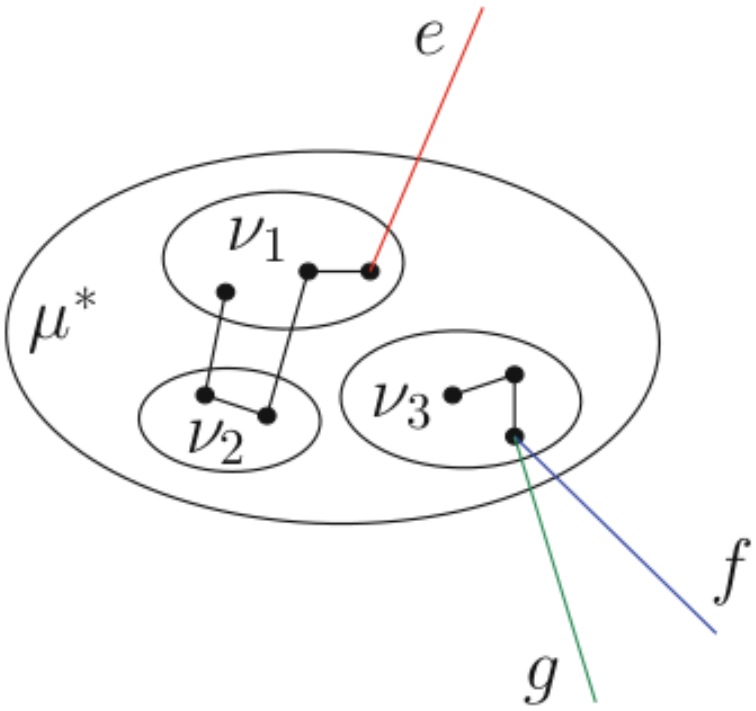

Figura 7.3: Dettaglio del cluster μ^* con rappresentazione Spring-enbedding

$C_i(G_i, T_i)$ 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

$$Ci(Gi, Ti)isc - planarifandonlyifCi + 1(Gi + 1, Ti + 1)isc - planar$$

7.2 algoritmo

Definita e chiarita la riduzione polinomiale ideata dal professor Patrignani dell'università Roma Tre si passa ora all'operazione di trasformazione che l'utente, mediante una interazione su un bottone On/Off, potrà richiedere. 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

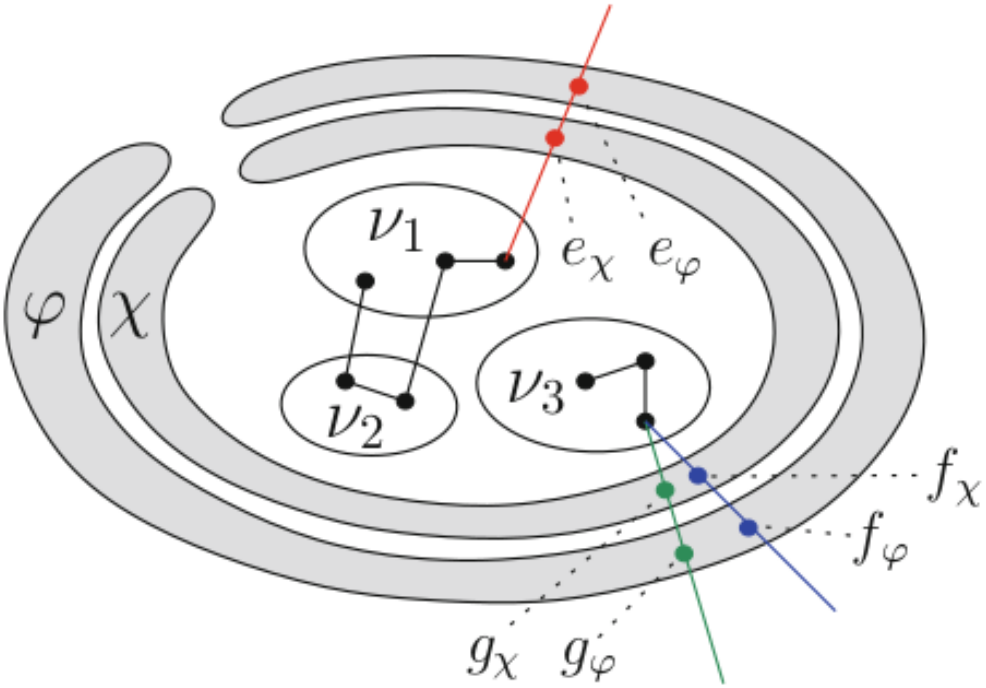

Figura 7.4: Cluster μ^* dopo la riduzione con rappresentazione semi Spring-enbedding

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 7.5 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;

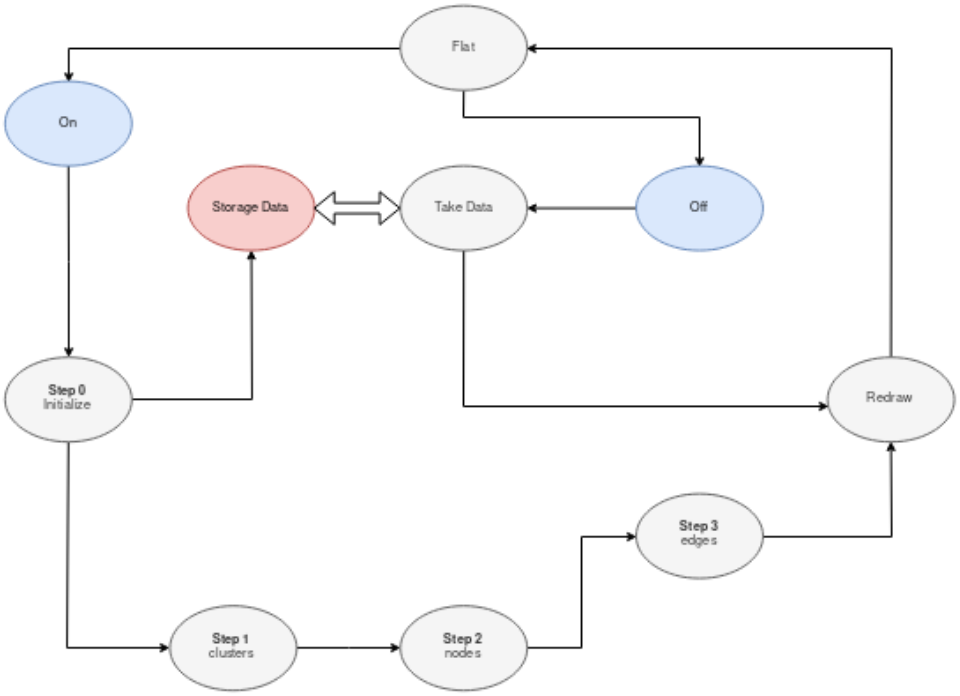

Figura 7.5: Struttura semplificata dell'algoritmo di flattizzazione

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

Nella fase di inizializzazione il sistema prima di eseguire qualunque cambiamento sugli oggetti creati e/o importati dall'utente provveder ad eseguire una copia di questi dati per poter eseguire operazioni di ritorno ai valori pre-riduzione e continuarne la modifica. Successivamente vengono creati delle liste contenenti solo ed esclusivamente gli oggetti del grafo su cui eseguire la riduzione, quali nodi con archi uscenti e cluster con livello $l > 2$.

Superata la fase di inizializzazione si passa alla trasformazione dei dati. Nello step 1 $\forall$cluster $\mu_i, \forall i = 0, ..., l.length$ con l uguale alla lista dei cluster da cambiare inizializzata in precendenza, questo viene eliminato ed al suo posto vengono

inseriti due oggetti cluster che possiederanno come attributo label l'etichetta del cluster μ_i a cui sarà unita la lettera X o Y. Creati i cluster si passa poi al secondo step ovvero all'aggiunta dei nodi aggiuntivi. In particolare $\forall$nodo $n_i, \forall i = 0, ..., nodes.length$ con nodes uguale alla lista dei nodi da cambiare inizializzata,si creeranno, $\forall$ arco esterno del loro *rotationScheme* e_i due nodi n_iX ed n_iY rispetivamente interni ai cluster creati prima che hanno sostituito quello in cui il nodo risiedeva. Nel terzo step creati cluster e nodi si può passare alla rimozione degli archi intercluster appartenenti alla lista degli archi da cambiare 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 7.6. 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

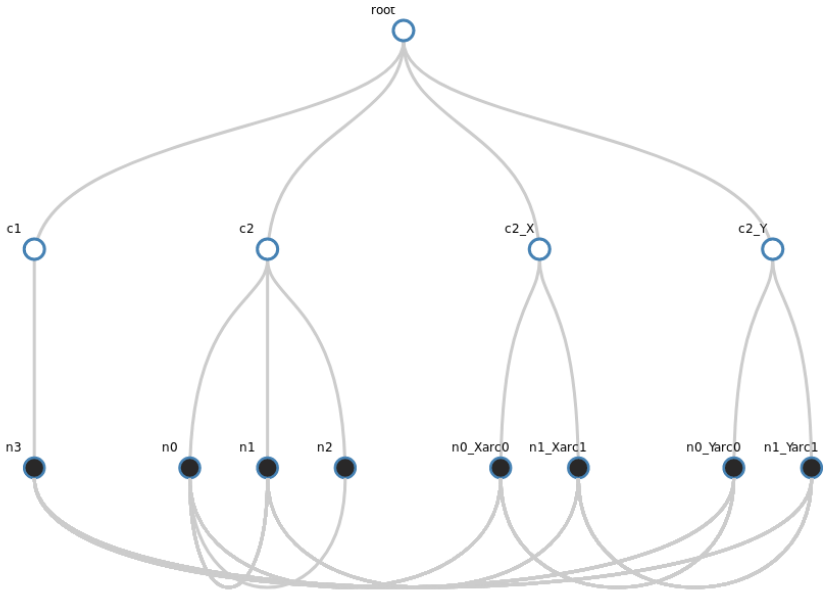

Figura 7.6: Esempio della riduzione di un grafo clusterizzato nella Tree-View

riduzione saranno visti nel dettaglio nel capitolo successivo in cui verranno utilizzate la maggior parte delle interazione ed operazioni dell'utente in cui saranno eseguite simulazioni reali di utilizzo e di test non solo sulle connessioni tra gli oggetti che compongono il grafo clusterizzato ma anche sulla loro visualizzazione. Il sistema inoltre non pone alcun messaggio di errore nel caso in cui l'utente decida di effettuare di riduzione del grafo clusterizzato 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 8

Esempi reali di utilizzo

Di seguito saranno viste le varie possibilitá prese in considerazione e non ancora implementate. L'idea di base è quindi quella di estendere non le funzionalità ma le tipologie di implementazioni, tramite l'utilizzo di strutture dati diverse, in modo da avere come obiettivo finale quello di poter dare all'utente utilizzatore la possibilità di scegliere l'implementazione desiderata.

8.1 Quadtree e Octree

8.1.1 Definizione delle strutture

Conclusioni e sviluppi futuri

Il metodo *Lattice Boltzmann* come visto è tuttora utilizzato in tutti gli ambiti inerenti la fluidodinamica proprio per l'algoritmo che, a contrario delle equazioni di Navier-Stokes viste nel capitolo 1, è facilmente realizzabile a livello di computazione e ciò ha spinto l'attività di tirocinio formativo a trovare ed analizzare altri aspetti, che potessero affiancare l'utilizzo di questo metodo, da implementare e realizzare.

All'inizio dell'attività di tirocinio una volta preso nota l'importanza nell'ambito della computer grafica e delle simulazioni dell'utilizzo del metodo Lattice Bolzmann sono stati quindi individuati due obiettivi chiave su cui porre un'attenzione particolare. Primo dei quali, non per importanza, è l'aumento della velocità di esecuzione che potesse comunque adattarsi ed essere utilizzato su qualunque tipologia di macchina. Il secondo si concentra invece sulla facilità di utilizzo da parte dell'utente finale, obiettivo messo in primo piano da molti sviluppatori di librerie.

Al termine del tirocinio formativo e con il risultato della progettazione della libreria sono stati raggiunti entrambi gli obiettivi che furono posti tre mesi prima. Il primo, la realizzazione di un software veloce è stato raggiunto mediante l'introduzione di due paradigmi di programmazione che puntano proprio alla velocità di esecuzione come visto nei paragrafi 2.2 e 2.3: la programmazione concorrente e parallela, che fa uso di più thread di esecuzione per suddividere un problema in più sottoproblemi

da svolgere in parallelo introducendo quindi il concetto di multi-tasking e la programmazione generica, che mediante l'utilizzo delle classi e dei metodi templetizzati o "modelli" risolti a tempo di compilazione anzichè a tempo di esecuzione migliora anche il riutilizzo del codice e il refactoring.

I risultati ottenuti in questa tesi rappresentano un primo passo esplorativo nella direzione dello studio della velocità computazionale e del confronto di essa a parità di prestazioni della macchina. La velocità computazione infatti tiene conto: delle strutture dati utilizzate, per gli storage e per il reticolo discreto e della quantità di celle, particelle e distribuzioni di velocità di cui è composta la griglia, ovvero dalla tipologia di reticolo scelto dall'utente.

Bibliografia

[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.