Università degli studi di Perugia

Facoltà di Ingegneria

Tesi di Laurea in

Ingegneria Informatica e dell'Automazione

Generazione automatica di Word Cloud dinamiche

Relatore

Prof. Carla Binucci

Co-relatore

Prof. Walter Didimo

Candidato

Enrico Spataro

Anno Accademico 2014/2015

EVENTUALE DEDICA

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Capitolo 2

Word cloud statiche

Questo capitolo consiste in una breve introduzione al concetto di word cloud.

Nel paragrafo 2.1 verranno introdotte alcune definizioni e si parlerà brevemente di qualche applicazione, mentre nel paragrafo 2.2 si farà il punto sullo stato dell'arte riguardo le word cloud semantiche.

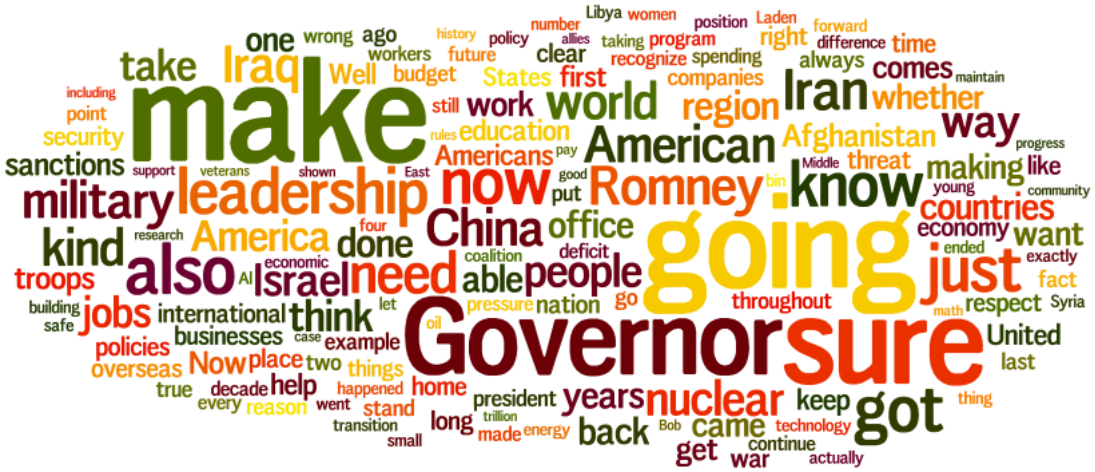
2.1 Definizioni e applicazioni

2.1.1 Cos'è una word cloud?

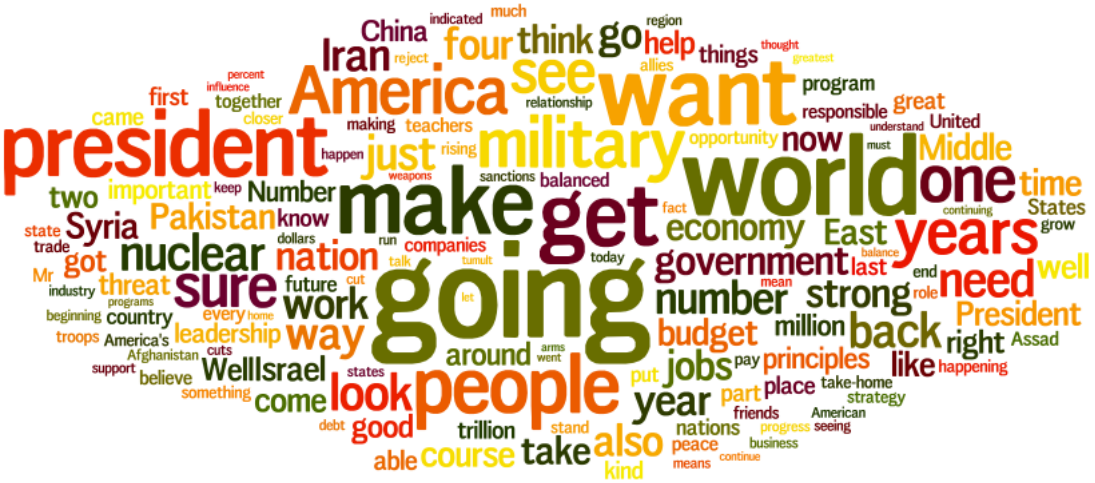
Il recente sviluppo di Internet, con l'avvento del Web 2.0, assieme al grande progresso tecnologico dei calcolatori, ha comportato un'ingente produzione di dati sul web e sulle piattaforme web based, per cui il problema di estrarre, gestire e visualizzare efficacemente tale informazione è diventata, negli ultimi anni, un'area di ricerca piuttosto importante nella visualizzazione dell'informazione.

In generale, una **word cloud** è una rappresentazione visuale di documenti testuali, che utilizza diversi colori, font e dimensioni per raffigurare le parole più rilevanti, dette **keywords**, di un generico documento. Esse sono utilizzate, quindi, per esaminare un testo, in modo da facilitarne la comprensione, o per confrontare più testi. Ad esempio, nelle elezioni presidenziali del 2008 e del 2012 (fig. 2.1), i media americani hanno confrontato le word cloud generate dai dibattiti dei candidati alla presidenza americana, mettendo in risalto le differenze tra i discorsi dei candidati; anche in Italia, in occasione del discorso di insediamento alla Camera da parte del presidente Mattarella, alcune testate giornalistiche hanno fatto uso delle word cloud per analizzare il contenuto del discorso.

In riferimento al web, si parla invece di **tag cloud**, con evidente richiamo ai tag, ovvero ai metadati che riassumono il contenuto di un sito internet. Ogni tag, rappresenta un link ad una specifica risorsa sul web, consentendo agli utenti di accedervi tramite l'utilizzo di keywords. Il loro utilizzo si è diffuso grazie al sito di *photo sharing* Flickr [1], in cui i tag classificano in diverse categorie le foto che vengono condivise tra gli utenti.



(a) Word cloud generata dal discorso di Obama.



(b) Word cloud generata dal discorso di Romney.

Figura 2.1: Due wordcloud relative ai dibattiti tra i candidati alla presidenza statunitense.

Le parole di una word cloud sono tipicamente pesate in base all'importanza che esse ricoprono nel testo: più le parole hanno un font grande, più sono rilevanti. In questo modo, le word cloud permettono immediatamente di evidenziare ciò che è rilevante in un testo. Ci sono anche altri parametri da tenere in considerazione. In [2], Halvey et al. hanno valutato l'effetto di alcuni fattori: la posizione delle parole, la loro disposizione secondo l'ordine alfabetico e, come detto, la dimensione del font, si sono rivelati essere parametri importanti. Inoltre, hanno notato che gli utenti, piuttosto che leggere tutte le parole, danno uno sguardo generale alla word cloud. In un altro lavoro, Lohmann et al. [3] hanno scoperto che parole posizionate vicino al centro catturano di più l'attenzione rispetto a parole vicine ai bordi, così come parole posizionate in alto a sinistra vengono percepite prima delle altre da parte degli utenti.

2.1.2 Applicazioni

Esistono diversi strumenti per la creazione di word cloud. Un tool web based molto popolare, Wordle [4], grazie alle qualità grafiche e alle sue funzionalità, ha permesso la diffusione delle word cloud come potente strumento per riassumere e analizzare un testo. Con Wordle, ad esempio, è possibile impostare alcuni parametri in modo da personalizzare la word cloud finale, come il numero delle parole, il colore, gli angoli di disegno ecc.. Tuttavia, Wordle non riesce a catturare le relazioni semantiche tra le parole, proprietà che può rivelarsi cruciale nell'analisi e nella comprensione di un testo. Per ovviare a ciò, è stato proposto un ulteriore strumento, basato su Wordle, chiamato ManiWordle [5], il quale offre un buon livello di interazione con l'utente, permettendo a quest'ultimo di manipolare il disegno finale e di modificare le parole visualizzate in termini di posizione, colore e orientamento, risultando quindi più flessibile di Wordle. Un altro sistema, SparkClouds [6], tramite l'uso delle *sparklines*, mette in risalto i cambiamenti tra più word cloud. Collins et al. [7], hanno presentato Parallel Tag Clouds, uno strumento in grado di visualizzare le differenze tra i testi scritti di un ricco dataset. FacetAtlas [8], invece, è un'applicazione che, tramite grafici e mappe di densità, visualizza le relazioni che intercorrono tra i documenti di una vasta collezione di testi.

La word cloud, in generale, costituisce un potente mezzo che consente agli utenti di comprendere istantaneamente il contenuto di un documento. Negli anni, sono state sviluppate varie applicazioni che generano word cloud, ognuna con pregi e difetti. Ad esempio, le word cloud tradizionali, che dispongono le parole in maniera randomica o in ordine alfabetico, non sono strumenti utili per comparare più testi. In tal caso, infatti, gli utenti dovrebbero analizzare manualmente le due visualizzazioni per notare eventuali differenze, e ciò va ad annullare i benefici che ne derivano dall'utilizzo delle word cloud. Il nostro lavoro è invece in direzione di quello che è il trend degli ultimi tempi, cioè quello di creare word cloud semantiche, in cui la disposizione delle parole riflette la loro correlazione semantica.

2.2 Word cloud semantiche

Recentemente, la maggior parte dei tool che generano word cloud si è posta come obiettivo quello di raggruppare semanticamente le parole estratte, utilizzando tecniche di elaborazione del linguaggio naturale per correlare parole simili tra loro. Infatti, la possibilità di disegnare, vicine nella word cloud, parole correlate semanticamente, può migliorare l'esperienza dell'utente, come notato da Deutsch et al. in [9].

Tree Cloud [10], ad esempio, è un applicazione in cui le parole vengono disposte secondo un albero, in modo tale da preservare la loro vicinanza semantica. In [11], Cui et al., tramite misure di similarità, mirano a collocare, vicine nel disegno, parole correlate semanticamente, utilizzando poi un metodo force directed per compattare la word cloud. Wu et al. [12], utilizzano una tecnica ispirata al *seam carving*, algoritmo di ridimensionamento dell'immagine in base ai contenuti, per ottenere una word cloud semantica e compatta. Questo lavoro di tesi, invece, prende spunto dal recente lavoro svolto da Kobourov et al. [13], in cui vengono implementati due nuovi algoritmi di visualizzazione, da confrontare con altri algoritmi esistenti, per analizzare la qualità delle word cloud in base a diverse metriche, ovviamente partendo dalla base comune costituita dalla coerenza semantica nella disposizione delle parole.

Capitolo 3

Word cloud dinamiche

Il concetto di word cloud dinamica, che è l'obiettivo di questa tesi, è descritto, dal punto di vista algoritmico, nel seguente capitolo.

La sezione 3.1 offre una visione generale sullo stato dell'arte riguardo le word cloud dinamiche (cioè tempo varianti). Il paragrafo 3.2 espone, fase per fase, il processo che consente di creare una word cloud statica da un testo di input, tenendo ben presente il vincolo di vicinanza semantica tra parole simili. Ogni fase è composta da diversi algoritmi, che vengono descritti progressivamente. Il capitolo quindi si chiude con la sezione 3.3, che presenta i passaggi necessari ad ottenere dinamicità nel layout finale.

3.1 Definizioni e applicazioni

Negli ultimi anni, sono state proposte diverse applicazioni per la creazione di word cloud. Oltre alla distinzione tra word cloud semantiche e non semantiche, è possibile suddividere tali applicazioni sulla base di due ulteriori categorie: word cloud **statiche** e word cloud **dinamiche** (o **tempo varianti**). La principale differenza tra queste due classi è chiaramente costituita dal fattore tempo: le word cloud dinamiche, infatti, hanno come obiettivo quello di illustrare l'evoluzione temporale di un documento o di un set di documenti. I grafici a barre, per esempio, sono tipicamente utilizzati per rappresentare l'andamento temporale di una qualche variabile e consentirne l'analisi visuale [14] [15]; Dubinko et al. [16] hanno sviluppato un tool che mostra l'evoluzione dei tag in Flickr e che permette l'interazione con gli utenti; Cui et al. [11] hanno proposto un sistema che abbina un grafico di tendenza (*trend chart*) alle word cloud di una collezione di documenti, per illustrarne l'evoluzione semantica.

Sebbene tutti questi lavori abbiano come finalità quella di visualizzare il trend temporale di un insieme di testi scritti, le informazioni spaziali e temporali sono rappresentate da immagini statiche. Diversamente, un lavoro interessante è stato svolto di recente da Chi et al. [17], in cui viene mostrato, in modo dinamico, il progresso temporale di un

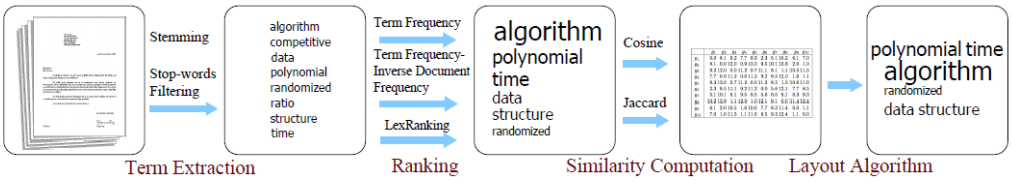


Figura 3.1: Generazione di una word cloud semantica.

set di documenti tramite l'utilizzo di tecniche di *morphing*[1], che permettono di passare gradualmente da una word cloud di un documento ad un'altra di un altro documento, modificandone anche la forma. Tuttavia, in questo studio, non viene affrontato l'aspetto semantico nei layout di ogni word cloud.

Il nostro lavoro, invece, a differenza degli altri citati precedentemente, ha come finalità quello di mostrare lo sviluppo temporale di un solo testo. A tal fine, vengono utilizzate tecniche di morphing tra le word cloud generate in diversi istanti di tempo durante l'elaborazione del testo, rispettando, se possibile, il vincolo di vicinanza geometrica tra parole correlate semanticamente. Inoltre, è anche importante che non ci siano troppe differenze tra word cloud estratte in istanti successivi, ovvero parole che si muovono eccessivamente: l'utente potrebbe disorientarsi e perdersi durante l'evoluzione del testo, per cui la sua mappa mentale non deve cambiare in modo radicale. Da notare che, rispettare il vincolo di vicinanza semantica, insieme alla coerenza della mappa mentale, può costituire un compito impegnativo, dal momento che questi due vincoli costituiscono obiettivi contrastanti fra loro.

3.2 Algoritmi di generazione di una word cloud semantica e statica

Tutti gli algoritmi di visualizzazione di una word cloud ricevono in input un grafo pesato, i cui vertici sono le parole, rappresentate da rettangoli. Tuttavia, sono necessari alcuni passi di preprocessing, che consentono di estrarre questa informazione dall'input. L'algoritmo di visualizzazione disegna, per quanto possibile, le parole più simili vicine tra loro. Il processo di creazione di una word cloud è indicato in figura 3.1. Infine, viene applicato un algoritmo di clustering per assegnare lo stesso colore a parole dello stesso cluster.

3.2.1 Estrazione keywords

Il processo di estrazione delle keywords prevede una serie di passaggi preliminari, con tecniche di elaborazione del linguaggio naturale, le quali predispongono, in maniera appropriata, il testo in ingresso all'algoritmo di estrazione delle parole.

[1]Il morphing consiste nella trasformazione fluida, graduale e senza soluzione di continuità tra due immagini di forma diversa [18].

Innanzitutto, il testo viene suddiviso in frasi. Poi, ogni sequenza di caratteri viene scomposta in *token* (insiemi di caratteri, ad esempio parole, punteggiatura, numeri, simboli ecc...): ciò può essere eseguito mediante l'ausilio di librerie di elaborazione del linguaggio naturale (e.g. *Apache OpenNLP* [19]). Successivamente, dal testo vengono eliminate le *stop words*, cioè articoli, congiunzioni, parole di uso comune che sono poco rilevanti dal punto di vista informativo. Le parole rimanenti vengono quindi raggruppate in base alle rispettive radici (in inglese, *stem*), tramite un algoritmo di *stemming*: in questo modo, ad esempio, parole come *player*, *play* e *playing* vengono raggruppate secondo la radice comune *play*. Nella nostra implementazione, è stato utilizzato il noto algoritmo *Porter Stemmer* [20]. Alla fine, nella word cloud finale, viene visualizzata la variante più frequente della parola.

Una volta eseguiti questi passaggi, si procede all'estrazione delle parole e al loro ranking in modo da trovare quelle più rilevanti, utilizzando tecniche di Information Retrieval. Ogni algoritmo assegna alle parole un punteggio e ne seleziona le n più frequenti, dove n è il numero di parole da visualizzare nella word cloud.

In questo lavoro di tesi sono stati utilizzati diverse tecniche di estrazione delle keywords, qui di seguito esposti.

Term Frequency (TF)

Il modo più intuitivo di assegnare un peso alle parole consiste nel contare le loro singole occorrenze. Questo è ciò che viene fatto dall'algoritmo Term Frequency. Tuttavia, la rilevanza di un termine non aumenta linearmente con il numero delle occorrenze, in quanto documenti di una certa lunghezza potrebbero contribuire di più rispetto a documenti più corti, cioè potrebbero avere un peso maggiore nel conteggio delle occorrenze. Il calcolo della frequenza potrebbe quindi essere influenzato da questo fattore, per cui il punteggio di ogni parola spesso viene scalato tramite una qualche funzione. In tabella 3.1, sono riportate le tipiche funzioni che vengono utilizzate per pesare tale contributo (come indicato in [21]), dove l'argomento $tf_{t,d}$ indica la frequenza del termine t nel documento d.

Tipo funzione	**Peso**
Binaria	0,1
Lineare	$tf_{t,d}$
Radice quadrata	$\sqrt{tf_{t,d}}$
Logaritmo	$1 + \log tf_{t,d}$
Doppia normalizzazione con parametro K	$K + (1 - K) * \frac{tf_{t,d}}{\max_{t' \in d} tf_{t',d}}$

Tabella 3.1: Term Frequency ranking: funzioni

Ad ogni modo, pur dopo aver rimosso le stop words, l'algoritmo Term Frequency tende ad assegnare punteggi troppo alti a termini poco rilevanti. Termini rari, invece, hanno contenuto informativo più alto rispetto a termini frequenti, per cui ad essi vanno assegnati punteggi più elevati. In particolare, si definisce il parametro **IDF** (**Inverse Document Frequency**) di un termine t la quantità:

$$idf_t = \log \frac{N}{df_t}, \tag{3.1}$$

dove N è la dimensione di una collezione di documenti, mentre la quantità df_t è detta *document frequency*, cioè il numero totale di documenti in cui il termine t compare. Per termini frequenti in una collezione, tale valore tende a zero, mentre per termini meno frequenti il punteggio sarà più alto. Lo scaling che viene applicato è solitamente logaritmico, con qualche variante.

TF-IDF

Ora si possono combinare le due definizioni di TF e IDF per produrre un ulteriore algoritmo, noto come TF-IDF, il quale assegna, ad ogni termine t di un documento d, la quantità

$$tf\text{-}idf_{t,d} = tf_{t,d} \times idf_t. \tag{3.2}$$

Ne segue che:

- se t è un termine comune nella collezione, avrà un $tf_{t,d}$ alto, ma un idf_t vicino a zero, per cui $tf\text{-}idf_{t,d}$ sarà tendente a zero;
- se t è un termine raro nella collezione, ma frequente nel documento d, allora avrà entrambi i contributi elevati, da cui ne deriva che $tf\text{-}idf_{t,d}$ sarà alto.

In pratica, con questo approccio, vengono filtrati i termini molto comuni, mentre quelli davvero rilevanti per il documento vengono estratti.

Uno degli schemi più noti in letteratura per calcolare la $tf\text{-}idf_t$, come suggerito in [21] ed adottato in questa tesi, è il seguente:

$$tf\text{-}idf_{t,d} = (1 + \log tf_{t,d}) \times \log \frac{N}{df_t}. \tag{3.3}$$

LexRank

Il terzo algoritmo di ranking è LexRank [22], già usato in [12] per la creazione di word cloud semantiche. Tale algoritmo prende spunto dall'algoritmo PageRank [23], utilizzato da Google per assegnare un punteggio alle pagine web e quindi migliorare le ricerche che si effettuano con il noto motore di ricerca.

LexRank è un algoritmo basato su un grafo $G = (V, E)$, dove i vertici sono le parole, collegati da archi che rappresentano le co-occorrenze di due parole all'interno di una

frase. Ogni arco (i, j) ha infatti un peso w_{ij}, pari al numero di occorrenze della parola i e della parola j all'interno di una stessa frase. I punteggi vengono poi calcolati sfruttando il concetto di centralità degli autovettori definiti in G. Tale valore di centralità viene distribuito, da ogni vertice, ai suoi vicini. Sia quindi R il vettore di ranking, di dimensione $1 \times |V|$, dove $|V|$ è il numero dei nodi di G. Possiamo definire:

$$R = dM \cdot R + (1 - d)p \tag{3.4}$$

$$P \cdot R = \lambda R \tag{3.5}$$

$$P = dM + (1 - d)p \cdot 1^T \tag{3.6}$$

dove:

- d è il *damping factor*, tipicamente scelto nell'intervallo $[0.1, 0.2]$, come suggerito in [22];
- M è la matrice delle co-occorrenze normalizzata, avente dimensioni $|V| \times |V|$ e tale che la somma di ogni colonna sia pari a 1;
- p è il vettore delle probabilità, di dimensione $1 \times |V|$, con ogni elemento pari a $1/|V|$;
- R è l'autovettore corrispondente al più grande autovalore di M e può essere ricavato tramite l'algoritmo *Power Method*, usato in [22].

Alla fine, le parole estratte saranno costituite dai primi n valori di R.

3.2.2 Calcolo similarità

Il passo successivo è quello di calcolare la similarità tra le keywords, ovvero quanto esse sono correlate tra loro. Data la lista delle n parole estratte, viene calcolata la matrice $n \times n$ delle similarità tra coppie di parole. Ogni valore è compreso tra 0 (nessuna correlazione) e 1 (massima correlazione). Esistono diversi algoritmi per il calcolo delle similarità. Tutti usano uno spazio vettoriale di dimensione n (pari al numero di parole estratte), dove il generico vettore $w_i = \{w_{i1}, w_{i2}, ..., w_{in}\}$ rappresenta la co-occorrenza della i-esima parola con le altre $n - 1$ parole.

Di seguito sono esposte le tecniche di calcolo da noi implementate.

Cosine Similarity

La cosine similarity tra due vettori w_i e w_j viene calcolata come:

$$sim_{ij} = \frac{w_i \cdot w_j}{||w_i||\,||w_j||} \tag{3.7}$$

In pratica, tale quantità corrisponde alla misura dell'angolo formato tra i vettori w_i e w_j. Se la similarità è 1, allora l'angolo formato è pari 0°, mentre se la similarità è 0, allora i due vettori sono perpendicolari (angolo di intersezione 90°) e non condividono alcuna frase.

Jaccard Similarity

La Jaccard similarity è definita come il rapporto tra il numero delle frasi condivise tra due parole e il numero totale delle frasi in cui esse compaiono. In formule:

$$sim_{ij} = \frac{|S_i \cap S_j|}{|S_i \cup S_j|}, \tag{3.8}$$

dove S_i e S_j sono, rispettivamente, l'insieme delle frasi in cui compare la parola i e l'insieme delle frasi in cui compare la parola j.

Jaccard Similarity estesa

Un ulteriore modo per il calcolo della similarità è rappresentato dalla Jaccard similarity estesa (anche nota come **coefficiente di Tanimoto**), definita come:

$$sim_{ij} = \frac{w_i \cdot w_j}{||w_i||^2 + ||w_j||^2 - w_i \cdot w_j} \tag{3.9}$$

Tale misura si riduce alla Jaccard Similarity nel caso di vettori binari.

3.2.3 Creazione word cloud

Gli algoritmi che producono word cloud ricevono in input una collezione di n rettangoli (chiamati *bounding box*), corrispondenti alle n parole estratte, ognuno dei quali avente dimensioni proporzionali alla rilevanza della parola, e la matrice delle similarità (di dimensione $n \times n$), dove ogni elemento è in $[0, 1]$. L'output è costituito da un insieme di rettangoli sul piano e non sovrapposti. All'interno dei rettangoli sono disegnate le parole. Ovviamente, nel layout finale, saranno visibili solo le parole.

In questo lavoro di tesi ne sono stati utilizzati tre: Context-Preserving Word Cloud Visualization [11], Star Forest [13] e Cycle Cover [13]. Tutti e tre gli algoritmi sono stati pensati per una visualizzazione statica della word cloud, per cui essi sono stati modificati per tener conto dell'aspetto dinamico (più tardi vedremo come).

Context-Preserving Word Cloud Visualization (CPWCV)

Questo algoritmo, introdotto da Cui et al. in [11], mira a soddisfare il vincolo di vicinanza geometrica tra parole correlate semanticamente in due passi: prima di tutto, viene calcolata, a partire dal contributo di ogni elemento della matrice di similarità, la

distanza tra coppie di parole. Tale valore corrisponde alla distanza ideale tra la generica coppia di parole (i, j) in uno spazio n-dimensionale. Viene quindi applicato uno scaling multidimensionale (MDS), in modo da ottenere, su uno spazio bidimensionale, una prima collocazione delle parole tale da rispettare, approssimativamente, le distanze calcolate. Per preservare il posizionamento relativo tra le parole, si utilizza la triangolazione di Delaunay. Ciò crea un layout iniziale con occupazione dello spazio non ottimale. Perciò si applica un algoritmo force directed, che permette di riposizionare le parole mantenendo invariata la topologia del grafo, ovvero le relazioni semantiche tra le parole. Tale algoritmo si basa su tre principi:

- Principio di compattazione: questo principio mira a rimuovere, per quanto possibile, gli spazi tra le parole, in modo da ottenere un layout compatto;
- Principio di non sovrapposizione: questa condizione richiede che le parole non siano sovrapposte, proprietà fondamentale per la leggibilità della word cloud;
- Principio di planarità: per mantenere le relazioni semantiche tra le parole, il grafo è bene che sia planare, anche se ciò non è strettamente necessario. Inoltre imporre questa condizione può portare ad uno spreco di spazio.

Seguendo tali principi, la word cloud che si ottiene è compatta, facilmente leggibile e semanticamente coerente. Ognuna di queste proprietà è ottenibile applicando, rispettivamente, una forza elastica, una forza repulsiva e una forza attrattiva.

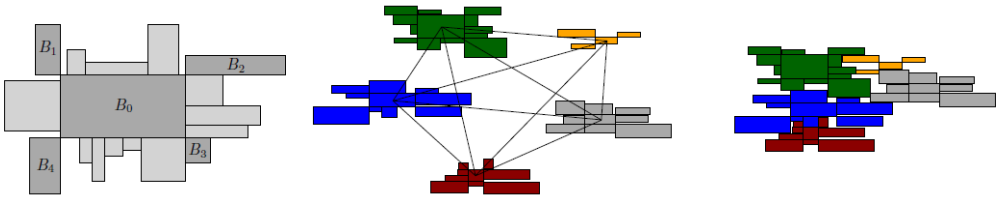
Star Forest

Introdotto in [13] da Kobourov et. al, Star Forest consiste in una foresta di stelle, ovvero una foresta dove le componenti connesse sono costituite da stelle. Una stella è un albero di profondità massima pari a 1.

L'algoritmo è composto da una sequenza di tre passaggi fondamentali:

1. Partizionamento del grafo, in modo da ottenere una foresta di stelle.
2. Applicazione dello scaling multidimensionale alla matrice delle distanze, ottenuta da quella delle similarità tra coppie di parole, con conseguente creazione della word cloud per ogni stella.
3. Compattazione delle singole word cloud realizzate, da cui ne deriva il risultato finale.

Le stelle vengono estratte dal grafo in modo *greedy*. Si cerca un vertice v i cui vertici adiacenti abbiano peso massimo, cioè tali che la somma $\sum_{u \in V} sim(u, v)$ sia massima. Si assume dunque che il vertice v sia il centro della stella, mentre l'insieme dei vertici $V - \{v\}$ forma l'insieme delle foglie. Vengono scelte le parole adiacenti a v e tali parole



(a) Solving a star (b) Packing star solutions

Figura 3.2: Algoritmo Star Forest.

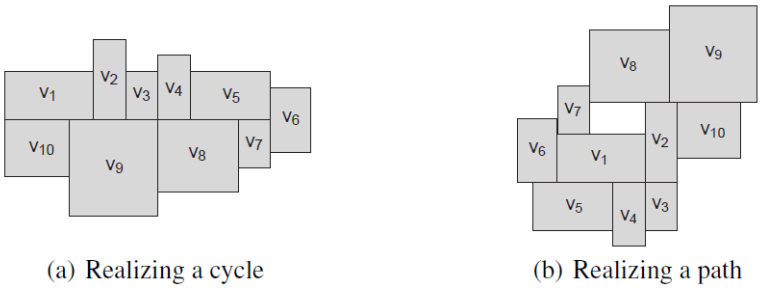
vengono rimosse dal grafo. Si ripete la procedura con grafi via via più piccoli, finchè non ci saranno più vertici.

Selezionare il miglior insieme di parole adiacenti al centro v della stella è un problema equivalente al noto problema dello zaino (*Knapsack Problem*): dati N oggetti, ognuno avente un peso e un valore, si vuole scegliere l'insieme di oggetti di maggior valore da inserire in uno zaino avente peso massimo sopportato pari a W. Sia dunque B_0 il bounding box corrispondente al centro della stella. In ogni soluzione ottima, ci sono quattro bounding box B_1, B_2, B_3, B_4, ognuno avente un lato che contiene uno degli angoli di B_0 (figura 3.2(a)). Dati B_1, B_2, B_3, B_4, il problema si riduce ad assegnare ad ogni box B_i uno dei quattro lati di B_0. Questo problema è equivalente ad un'istanza del problema dello zaino: il peso del bounding box B_i è proporzionale alle sue dimensioni, dove la sua base è moltiplicata per la base di B_0 e l'altezza è moltiplicata per l'altezza di B_0, mentre il valore è pari al peso dell'arco che collega B_0 a B_i. L'algoritmo viene eseguito, in senso orario, a partire dal lato superiore di B_0. Per risolvere le istanze del problema, si utilizza l'algoritmo di approssimazione in tempo polinomiale descritto in [24].

Le soluzione ottenute per le singole stelle vengono dunque messe insieme in un layout compatto, senza sovrapposizioni tra le parole e nel quale le relazioni semantiche tra le parole sono preservate (figura 3.2(b)). Per ogni coppia di stelle s_1, s_2, si ottiene la similarità media tra le parole di s_1 e s_2 come $sim(s_1, s_2) = \frac{\sum_{u \in s_1} \sum_{v \in s_2} sim(u,v)}{|s_1||s_2|}$. Si utilizza dunque l'MDS, con la distanza ideale tra le coppie di stelle posta uguale a $k(1 - sim(s_1, s_2))$, dove k è un fattore di scala, ottenendo così un primo layout. Poi, per compattare il disegno, si applica un algoritmo force directed. Da notare che esso viene applicato sulle stelle, non sulle singole parole. Tale algoritmo fa uso di due forze:

- attrattiva, per rimuovere gli spazi vuoti, uguale a $k_a(1 - sim(s_1, s_2))\Delta l$, con Δl pari alla minima distanza tra i centri delle due stelle;
- repulsiva, per evitare sovrapposizioni tra le varie parole, pari a $k_r min(\Delta x, \Delta y)$, dove Δx (Δy) corrisponde alla larghezza (altezza) della regione di spazio in sovrapposizione.

Come ogni algoritmo force directed, questo metodo aggiorna le posizioni delle stelle iterativamente.



(a) Realizing a cycle (b) Realizing a path

Figura 3.3: Algoritmo Cycle Cover.

Cycle Cover

Questo algoritmo è stato proposto, come Star Forest, da Kobourov et. al in [13]. Esso si basa sull'estrazione di un sottografo planare dal grafo $G = (V, E)$ definito dalla matrice di similarità. Tale sottografo è composto da un insieme disgiunto di cicli di peso massimo ed è denominato, appunto, *cycle cover*.

L'algoritmo che calcola il cycle cover estrae, dal grafo G, un grafo bipartito H. Si inizializza infatti H con i vertici di G. Poi, per ogni $v \in V(G)$, si aggiunge un vertice $v' \in V(H)$, e per ogni arco $(u, v) \in E(G)$, vengono creati due archi, (u', v) e $(u, v') \in E(H)$, aventi peso pari a $sim(u, v)$. Il grafo H risultante è bipartito per costruzione ed è facile ricavare un matching di peso massimo. Tale matching consiste in un insieme di cammini e cicli disgiunti di G, in quanto ogni u è accoppiato a v' e ogni u' è accoppiato a v.

Fissato dunque un ciclo $(v_1, v_2, ..., v_n)$, sia t il massimo indice tale che $\sum_{i \leq t} w_i \leq \sum_{i \leq n} w_i/2$, dove w_i è la base del bounding box corrispondente alla parola i-esima. I vertici $v_1, v_2, ..., v_t$ vengono collocati orizzontalmente da sinistra verso destra, mentre i vertici $v_n, v_{n-1}, ..., v_{t+2}$ vengono collocati da destra verso sinistra. In entrambi i casi, i vertici sono allineati su una stessa linea (figura 3.3(a)). Rimane da piazzare il vertice v_{t+1}, in contatto tra v_t e v_{t+2}. Si sceglie il gruppo di vertici (superiore o inferiore) i cui rettangoli hanno la minima larghezza oppure lo si pone a metà tra v_t e v_{t+2}. I cicli vengono convertiti in cammini se sono composti da più di 10 vertici. In tal caso, dopo aver posizionato i vertici v_1 e v_2 vicini l'un l'altro, si procede a piazzare il generico vertice v_i in modo tale che tocchi v_{i-1} nel primo spazio disponibile in senso orario, ottenendo così un layout a spirale.

Successivamente, così come nell'algoritmo Star Forest, le soluzioni individuali realizzate vengono messe insieme. Poi si applica lo stesso algoritmo force directed, descritto precedentemente, per compattare il disegno e rimuovere eventuali intersezioni tra le parole.

3.2.4 Clustering

Una volta ottenuto il layout, si prosegue con l'applicazione di un algoritmo di *clustering*, il quale suddivide le parole in più gruppi (*cluster*). Ad ogni cluster viene assegnato un colore diverso e ogni colore identifica un diverso raggruppamento semantico. In generale, l'obiettivo del clustering è quello di ottenere che gli oggetti di ogni cluster siano il più possibile simili tra loro, o comunque che la loro correlazione sia più alta rispetto a quella con oggetti di cluster diversi.

L'algoritmo di clustering utilizzato in questa tesi è K-means++, proposto da Arthur et. al in [25], ed è una versione migliorata di uno dei più noti algoritmi in letteratura, K-means. Prima verrà quindi presentato K-means, per poi passare alla variante K-means++.

K-means

K-means è un tipo di algoritmo *prototype-based*, cioè basato sul concetto di prototipo, che è l'elemento più rappresentativo di un cluster. Infatti, un cluster è una collezione di oggetti in cui ogni oggetto è più simile al prototipo del rispettivo cluster piuttosto che ai prototipi di altri cluster. K-means definisce il prototipo in termini di centroide, tipicamente inteso come elemento medio di un insieme di punti.

La tecnica di clustering definita dal K-means è piuttosto semplice e intuitiva, oltre ad essere veloce, sebbene non ci sono garanzie riguardo l'accuratezzza del risultato. In generale, dato un intero $K > 0$ e un insieme di n oggetti di un dataset χ, si vogliono scegliere K centroidi, tali da minimizzare (o massimizzare) la funzione obiettivo $\phi = \sum_{i=1}^{K} \sum_{x \in C_i} dist(c_i, x)$, dove C_i è l'i-esimo cluster e $dist(c_i, x)$ è una qualche misura di prossimità tra il generico elemento $x \in C_i$ e il centroide dell'i-esimo cluster C_i. In pratica, ciò che si vuole ottenere, sono k cluster in cui gli oggetti appartenenti ad ogni cluster siano il più possibile simili tra loro.

L'ottimizzazione della funzione obiettivo è un problema NP-completo, per cui l'approccio comunemente adottato prevede le seguenti fasi, che fanno convergere l'algoritmo ad una soluzione locale (figura 3.4):

1. Si scelgono K centroidi $C = \{c_1, ..., c_k\}$.
2. Ogni elemento $x \in \chi$ viene associato al centroide più vicino. Ogni collezione di elementi associata a $c_i, i \in \{1, ..., k\}$, forma un cluster.
3. Si aggiorna il centroide di ogni cluster.
4. I passi 2 e 3 vengono eseguiti iterativamente finchè non ci sono ulteriori variazioni nell'insieme dei centroidi.

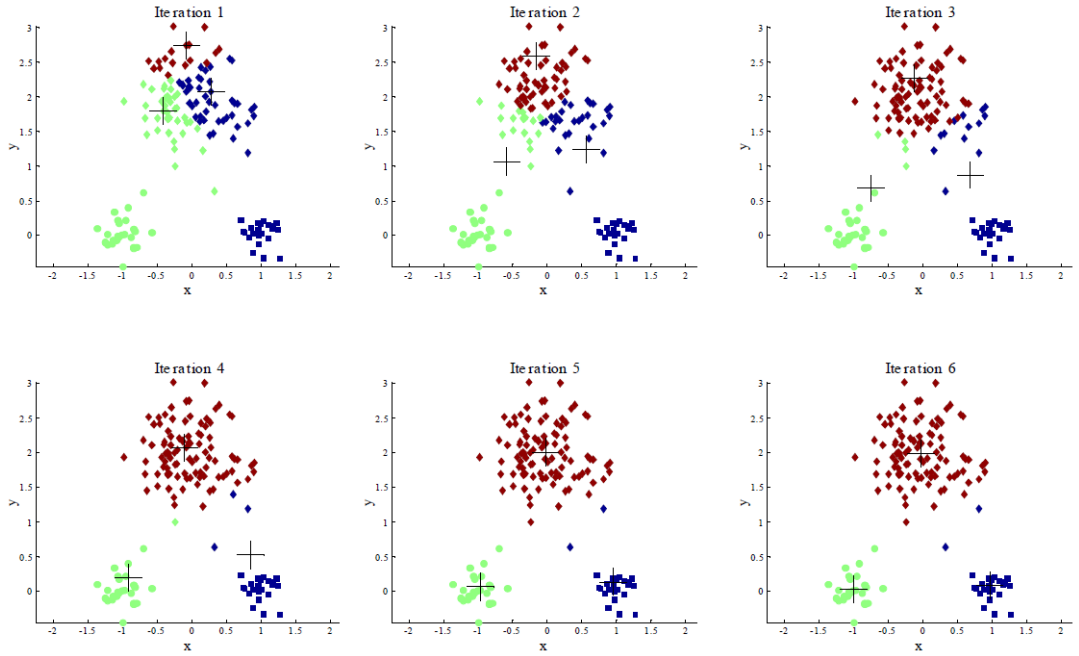


Figura 3.4: K-means: esempio di esecuzione dell'algoritmo [26]. In questo caso si ottengono $K = 3$ cluster.

Spesso i centroidi iniziali vengono selezionati in modo randomico, ma è possibile seguire diversi approcci. In ogni caso, supponendo di voler minimizzare la funzione obiettivo, l'idea dell'algoritmo è quella di decrementare, ad ogni esecuzione, il valore della funzione ϕ, mediante i passi 2 e 3, finchè non vengono osservate variazioni nei centroidi. Infatti, ϕ è una funzione monotona decrescente, per cui ad ogni esecuzione il suo valore al massimo rimane invariato. Dal momento che ci sono k^n raggruppamenti possibili, l'esecuzione dell'algoritmo termina sempre.

L'obiettivo dell'algoritmo, come detto, è quello di minimizzare (massimizzare) la funzione obiettivo ϕ. Tale funzione, misura la qualità del clustering tramite il calcolo della prossimità (similarità) tra il centroide c_i e gli elementi del cluster C_i. Solitamente, i punti fanno parte di uno spazio n-dimensionale e come misura di prossimità si utilizza la distanza euclidea. Ad ogni modo, ci sono diverse possibilità per la scelta della misura di prossimità, come indicato in tabella 3.2 [26].

K-means++

La differenza principale tra il K-means++ e il K-means classico consiste nell'inizializzazione dell'algoritmo, ovvero nella scelta dei K centroidi iniziali, da cui dipende la qualità del clustering finale. La fase di inizializzazione, infatti, consiste di diversi passaggi di seguito esposti:

Funzione di prossimità	Centroide	Funzione obiettivo
Manhattan	mediana	Minimizzare la somma delle distanze di ogni oggetto dal centroide del suo cluster
Euclidea	media	Minimizzare la somma delle distanze (al quadrato) di ogni oggetto dal centroide del suo cluster
Similarità (coseno, Jaccard...)	media	Massimizzare (minimizzare) la similarità (dissimilarità) di ogni oggetto con il centroide del cluster di appartenenza
Divergenza di Bregman	media	Minimizzare la somma della divergenza di Bregman di ogni oggetto con il centroide del cluster di appartenenza

Tabella 3.2: K-means: scelte comuni per le misure di prossimità.

1. Si sceglie il primo centroide c_1 in modo casuale tra tutti i punti (nel nostro caso le parole).

2. Per ogni altro elemento del dataset, $x \in \chi$, si calcola la distanza $dist(x, c_i)$ tra x e il centroide più vicino c_i tra quelli già selezionati.

3. Un elemento generico x può quindi diventare un centroide con probabilità pari a $\frac{dist(x,c_i)^2}{\sum_{x \in \chi} dist(x,c_i)^2}$.

4. Si ripetono gli step 2 e 3 finchè non si hanno K centroidi.

5. Avendo ottenuto un insieme di K centroidi, si può procedere con il K-means classico.

Con l'inizializzazione suggerita dal K-means++, l'algoritmo trova una soluzione che è $O(log\ k)$-competitiva rispetto alla soluzione ottima fornita dal K-means classico.

Per quanto riguarda l'applicazione dell'algoritmo al nostro lavoro, si parte da un dataset costituito dall'insieme delle n parole $W = \{w_1, ..., w_n\}$ estratte in fase di pre-processing. Come misura di prossimità è stata utilizzata la distanza tra le parole, ricavata dalla matrice di similarità. Il clustering quindi non viene eseguito sul disegno finale, cioè in base al posizionamento sul piano delle parole, ma a partire dalla loro similarità. Ne segue che il raggruppamento delle parole è logico, non geometrico (ovvero tramite la distanza euclidea, come spesso avviene). La funzione obiettivo da minimizzare diventa dunque $\phi = \sum_{i=1}^{K} \sum_{w \in C_i} (1 - sim_{w,c_i})$.

3.3 Algoritmi di generazione di una word cloud semantica e dinamica

Nel paragrafo precedente, è stata presentata la procedura, insieme agli algoritmi, necessaria a realizzare una word cloud semantica a partire da un testo in input. Tuttavia, tali algoritmi sono stati pensati per un singolo layout statico, cioè invariante nel tempo. L'obiettivo del nostro lavoro è invece quello di mostrare l'evoluzione della word cloud di un documento. A tal fine, vengono create diverse word cloud del testo ad intervalli regolari. Per passare da un layout al successivo in modo dinamico, è stata applicata una semplice tecnica di *morphing*. La dinamicità, come vedremo meglio dopo, dipende dallo stato delle parole, poichè esse possono apparire, scomparire o rimanere nel passaggio da una word cloud ad un'altra. Ciò significa che bisogna tener conto di diversi aspetti: variazione della posizione delle parole, variazione delle dimensioni delle parole in base al punteggio via via ottenuto e variazione dei colori. Tuttavia, conferire dinamicità comporta alcune problematiche:

- le parole comuni a due successive word cloud potrebbero variare di molto le rispettive posizioni, disorientando l'utente e portando confusione alla sua mappa mentale;
- i cluster più simili di due layout consecutivi devono poter mantenere lo stesso colore (quanti cluster manterranno il colore lo vedremo dopo).

Per la prima criticità, sono stati modificati gli algoritmi di disegno: lo spostamento di una parola dipende dalla sua rilevanza, per cui parole importanti variano poco la propria posizione. Questo è un obiettivo discordante con il mantenimento delle relazioni semantiche tra le parole, per cui è necessario calibrare i parametri in base alle proprie esigenze.

Per il secondo problema, invece, sono stati applicati gli algoritmi di calcolo della similarità descritti in sezione 3.2.2, così da trovare i cluster più simili in layout consecutivi (la similarità è intesa come numero di parole in comune tra i vari cluster).

Nel seguito, verranno esposti i passaggi necessari a risolvere le due criticità sopra elencate. Chiude il paragrafo la descrizione della tecnica di morphing utilizzata nel passaggio da una word cloud alla successiva.

3.3.1 Creazione word cloud

Gli algoritmi di disegno sono stati modificati come segue (eccetto ovviamente nella creazione del primo disegno):

- fissate K word cloud, siano Γ^{k-1} e Γ^k due disegni consecutivi, con l'intero $k \in [2, K]$;

- consideriamo inoltre le p parole che compaiono sia in Γ^{k-1} che in Γ^k, cioè l'insieme $P = \{w_1, ..., w_p\}$. Per ogni $w_i \in P$, si denotano con τ_i^{k-1} la posizione di w_i in Γ^{k-1}, con τ_i^k la posizione di w_i in Γ^k e con ρ_i^k il punteggio di w_i in Γ^k;
- una volta ottenuto, durante la realizzazione di Γ^k, un layout iniziale con l'utilizzo dell'MDS, si applica un metodo force directed che, iterativamente, trasforma τ_i^{k-1} in τ_i^k. Questa traslazione è però contrastata da ρ_i^k. La forza applicata dall'algoritmo durante la traslazione è dunque inversamente proporzionale a ρ_i^k, cioè $f_{k-1,k} \propto 1/\rho_i^k$.

Così facendo, le parole più importanti tendono a non cambiare continuamente posizione e quindi la mappa mentale dell'utente non varia di molto. Ciò è significativo, poichè la word cloud dinamica di un solo testo deve essere intuitiva e non deve portare confusione all'utente, sebbene questo possa significare una perdita di prestazioni dal punto di vista della vicinanza semantica delle parole.

3.3.2 Calcolo similarità tra cluster

Per quanto riguarda la seconda criticità, si procede al calcolo della similarità tra cluster con uno degli algoritmi descritti in precedenza. La similarità, praticamente, viene valutata sulla base di quante parole i cluster hanno in comune tra una word cloud ed un'altra.

Il procedimento adottato il seguente:

- fissate K word cloud, siano Γ^{k-1} e Γ^k due disegni consecutivi, con l'intero $k \in [2, K]$.
- denotiamo con $C^{k-1} = \{c_0^{k-1}, c_1^{k-1}, ..., c_l^{k-1}, \}$ e con $C^k = \{c_0^k, c_1^k, ..., c_m^k, \}$ gli insiemi dei cluster presenti in Γ^{k-1} e Γ^k, aventi dimensione $l, m > 0$ rispettivamente.
- si esegue il calcolo della similarità a coppie tra i due insiemi di cluster e la si indica con $sim_{c_i^{k-1}, c_j^k}$, dove $i \in \{1, ..., l\}$ e $j \in \{1, ..., m\}$. Le varie coppie ottenute vengono ordinate in ordine decrescente su una lista di dimensione $l \times m$. L'ordine è dato dal valore di similarità di ciascuna coppia.
- data la prima coppia nella lista, ad esempio (c_a^{k-1}, c_b^k), si associa b ad a. In questo modo, il cluster b avrà lo stesso colore di a. Vengono quindi eliminate tutte le occorrenze di c_a^{k-1} e c_b^k nella lista. Si procede così per tutte le altre coppie.

Si noti che il numero totale di cluster che tra Γ^{k-1} e Γ^k manterrà il colore è uguale a $min(l, m)$.

3.3.3 Morphing tra successive word cloud

Il morphing è stato applicato per gestire in modo fluido e continuo, tra un istante ed un altro, lo stato delle parole, cioè il loro movimento e dimensioni.

Dati due disegni consecutivi Γ^{k-1} e Γ^k, con $k \in [2, K]$, indichiamo con:

- $P = \{w_1, ..., w_p\}$ l'insieme delle parole comuni a Γ^{k-1} e Γ^k;
- $S = \{w_1, ..., w_s\}$ l'insieme delle parole in Γ^{k-1} che non compaiono in Γ^k;
- $C = \{w_1, ..., w_c\}$ l'insieme delle parole in Γ^k che non compaiono in Γ^{k-1};
- N il numero di frame tra un layout ed un altro (più N è alto, più l'animazione è fluida).

La parola i-esima w_i può far parte di tre diversi insiemi. A seconda dei casi, il comportamento può variare. Analizziamo ogni possibile situazione:

1. $w_i \in C$: poichè w_i compare solo in Γ^k con un punteggio pari a ρ_i^k, essa avrà un punteggio pari a 0 in Γ^{k-1}. Avendo N frame tra Γ^{k-1} e Γ^k, il punteggio viene incrementato, ad ogni frame, di una quantità pari a $\rho_i^k/(N+1)$. Di conseguenza, in base al punteggio corrente della parola, anche la dimensione del rettangolo viene opportunamente modificata ad ogni frame (figura 3.5).

2. $w_i \in S$: caso opposto al precedente, ovvero w_i compare solo in Γ^{k-1} con un punteggio pari a ρ_i^{k-1}, per cui essa avrà un punteggio pari a 0 in Γ^k. Poichè tra Γ^{k-1} e Γ^k ci sono N frame, il punteggio viene decrementato, ad ogni frame, di una quantità pari a $\rho_i^{k-1}/(N+1)$. Come nel caso precedente, la dimensione del rettangolo viene opportunamente modificata ad ogni frame in base al punteggio corrente della parola (figura 3.6).

3. $w_i \in P$: è il caso più complesso dei tre. Il termine w_i compare in entrambi i layout. Indicati con ρ_i^{k-1} il punteggio di w_i in Γ^{k-1}, con ρ_i^k il punteggio di w_i in Γ^k e con $\Delta_{\rho_i}^{k,k-1} = |\rho_i^k - \rho_i^{k-1}|$, se:

   - $\rho_i^k < \rho_i^{k-1}$, il peso di w_i viene decrementato di $\Delta/(N+1)$ ad ogni frame;
   - $\rho_i^k > \rho_i^{k-1}$, il peso di w_i viene incrementato di $\Delta/(N+1)$ ad ogni frame;

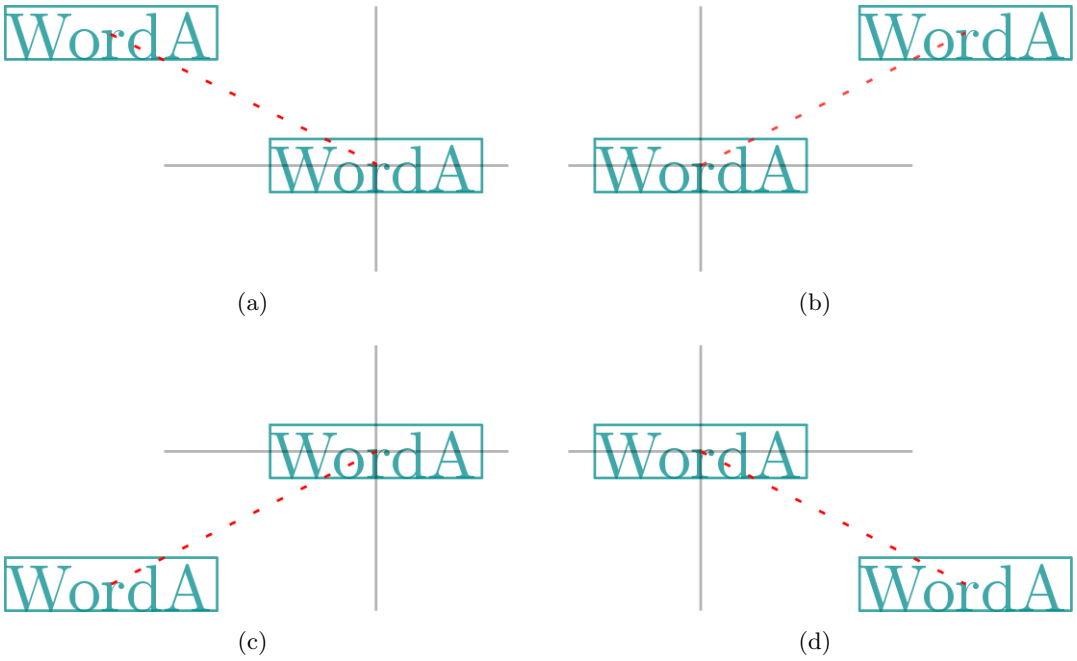
   In base al punteggio corrente, ad ogni frame la dimensione del bounding box di w_i viene opportunamente aggiornata.
   Inoltre, in questo caso, le parole vengono anche traslate, quindi le posizioni dei rispettivi rettangoli variano. Siano dunque $\tau_i^k = (x_i^k, y_i^k)$ il centro di w_i in Γ^k, $\tau_i^{k-1} = (x_i^{k-1}, y_i^{k-1})$ il centro di w_i in Γ^{k-1} e $\delta_{\tau_i}^{k,k-1} = ||\tau_i^k - \tau_i^{k-1}||$, ovvero la distanza tra τ_i^k e τ_i^{k-1}. Ad ogni frame, il rettangolo viene traslato, in valore assoluto, di una quantità pari a $\delta_{\tau_i}^{k,k-1}/(N+1)$. La direzione dello spostamento dipende dal relativo posizionamento di τ_i^{k-1} in Γ^{k-1} e di τ_i^k in Γ^k. Abbiamo in generale quattro casi diversi, come indicato in figura 3.7.

(a) (b) (c)

Figura 3.5: Esempio di parola che compare in Γ^k ma non in Γ^{k-1}. In figura sono riportati 3 frame consecutivi.

(a) (b) (c)

Figura 3.6: Esempio di parola che compare in Γ^{k-1} ma non in Γ^k. In figura sono riportati 3 frame consecutivi.



(a) (b) (c) (d)

Figura 3.7: Morphing delle parole comuni a τ_i^{k-1} e τ_i^{k-1}. Al centro c'è τ_i^{k-1}, mentre τ_i^k può trovarsi in uno dei quattro quadranti.

Bibliografia

[1] Flickr: photo sharing website, `http://www.flickr.com/photos/tags`.

[2] Martin J. Halvey and Mark T. Keane. An assessment of tag presentation techniques. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 1313–1314, New York, NY, USA, 2007. ACM.

[3] Steffen Lohmann, Jürgen Ziegler, and Lena Tetzlaff. Comparison of tag cloud layouts: Task-related performance and visual exploration. In *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part I*, INTERACT '09, pages 392–404, Berlin, Heidelberg, 2009. Springer-Verlag.

[4] Fernanda B. Viegas, Martin Wattenberg, and Jonathan Feinberg. Participatory visualization with wordle. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1137–1144, November 2009.

[5] Kyle Koh, Bongshin Lee, Bo Hyoung Kim, and Jinwook Seo. Maniwordle: Providing flexible control over wordle. *IEEE Trans. Vis. Comput. Graph.*, 16(6):1190–1197, 2010.

[6] Bongshin Lee, Nathalie Henry Riche, Amy K. Karlson, and Sheelash Carpendale. Sparkclouds: Visualizing trends in tag clouds. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1182–1189, November 2010.

[7] Christopher Collins, Fernanda B. Viegas, and Martin Wattenberg. Parallel tag clouds to explore and analyze faceted text corpora. In *IEEE VAST*, pages 91–98. IEEE Computer Society, 2009.

[8] Nan Cao, Jimeng Sun, Yu-Ru Lin, David Gotz, Shixia Liu, and Huamin Qu. Facetatlas: Multifaceted visualization for rich text corpora. *IEEE Trans. Vis. Comput. Graph.*, 16(6):1172–1181, 2010.

[9] Stephanie Deutsch, Johann Schrammel, and Manfred Tscheligi. Comparing different layouts of tag clouds: Findings on visual perception.

[10] Philippe Gambette and Jean Véronis. Visualising a Text with a Tree Cloud. In *IFCS'09: International Federation of Classification Societies Conference*, Studies in

Classification, Data Analysis, and Knowledge Organization, pages 561–569, Dresde, Germany, March 2009. Springer Berlin / Heidelberg.

[11] Weiwei Cui, Yingcai Wu, Shixia Liu, Furu Wei, Michelle X. Zhou, and Huamin Qu. Context-preserving, dynamic word cloud visualization. *IEEE Computer Graphics and Applications*, 30(6):42–53, 2010.

[12] Yingcai Wu, Thomas Provan, Furu Wei, Shixia Liu, and Kwan-Liu Ma. Semantic-Preserving Word Clouds by Seam Carving. *Computer Graphics Forum*, 2011.

[13] Lukas Barth, Stephen G. Kobourov, and Sergey Pupyrev. Experimental comparison of semantic word clouds. In Joachim Gudmundsson and Jyrki Katajainen, editors, *Experimental Algorithms*, volume 8504 of *Lecture Notes in Computer Science*, pages 247–258. Springer International Publishing, 2014.

[14] Lee Byron and Martin Wattenberg. Stacked graphs – geometry & aesthetics. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1245–1252, November 2008.

[15] Susan Havre, Beth Hetzler, and Lucy Nowell. Themeriver: Visualizing theme changes over time. In *Proc. IEEE Symposium on Information Visualization*, pages 115–123, 2000.

[16] Micah Dubinko, Ravi Kumar, Joseph Magnani, Jasmine Novak, Prabhakar Raghavan, and Andrew Tomkins. Visualizing tags over time. *ACM Trans. Web*, 1(2), August 2007.

[17] Ming-Te Chi, Shih-Syun Lin, Shiang-Yi Chen, Chao-Hung Lin, and Tong-Yee Lee. Morphable word clouds for time-varying text data visualization. *IEEE Trans. Vis. Comput. Graph.*, 21(12):1415–1426, 2015.

[18] Wikipedia, Morphing, consultato a Gennaio 2016, `http://it.wikipedia.org/wiki/morphing`.

[19] . Apache OpenNLP, `http://opennlp.apache.org`, .

[20] Martin F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[21] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[22] Dragomir R. Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, 22:2004, 2004.

[23] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. In *Proceedings of the 7th International World Wide Web Conference*, pages 161–172, Brisbane, Australia, 1998.

[24] Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, October 1975.

[25] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[26] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.