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.ssa Carla Binucci

Correlatore

Prof. Walter Didimo

Candidato

Enrico Spataro

Anno Accademico 2014/2015

Never say never, because limits, like fears, are often just an illusion.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

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.

L'obiettivo di questo lavoro di tesi consiste nella realizzazione di un sistema che consente di visualizzare, in modo dinamico, l'evoluzione temporale del contenuto di un testo, tramite l'utilizzo di word cloud. Nello specifico, il testo viene elaborato ad intervalli regolari, da cui si estraggono le parole più rilevanti del testo (fino a quell'istante), per poi visualizzarle sullo schermo, tenendo conto delle relazioni semantiche tra le parole. Il tutto avviene in un contesto dinamico, cioè il passaggio da una word cloud alla successiva avviene in modo graduale tramite delle animazioni.

In generale, l'utilizzo delle word cloud consente agli utenti di avere una rapida sintesi sul contenuto di un testo, facilitandone la comprensione. I contesti applicativi sono molteplici, e verranno discussi nei prossimi capitoli.

Questa tesi di laurea è dunque strutturata come segue:

- Nel capitolo 2 verrà introdotto il concetto di word cloud, facendo il punto sulle caratteristiche, sullo stato dell'arte e sulle possibili applicazioni.
- Il capitolo 3, dopo una breve distinzione tra word cloud statiche e dinamiche, tratterà gli aspetti algoritmici del sistema sviluppato. Verranno descritte le varie fasi, composte da diversi algoritmi, che consentono di creare una word cloud statica (sezione 3.2). L'approccio utilizzato per conferire dinamicità al sistema è invece esposto nella sezione 3.3.
- I dettagli implementativi, relativi all'architettura del sistema, e i risultati dell'analisi sperimentale coprono il capitolo 4.

- Infine, il capitolo 5 conclude il lavoro di tesi, riportando le considerazioni finali e delineando possibili sviluppi futuri.

Capitolo 2

Word cloud statiche

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

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

2.1 Definizioni e applicazioni

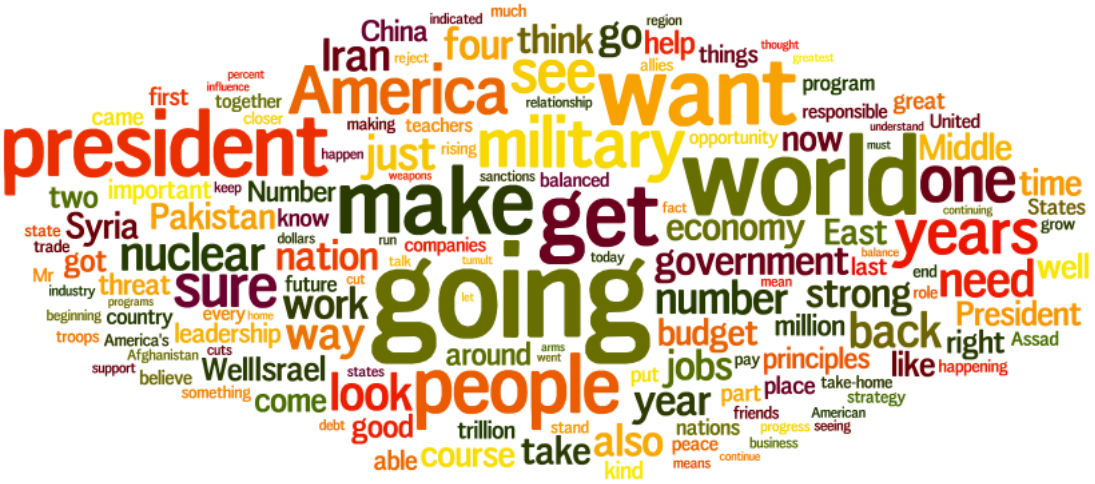
2.1.1 Cos'è una word cloud?

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 visualizzare sinteticamente il contenuto del suo discorso (fig. 2.2).

In riferimento al web, si parla invece di **tag cloud**, con evidente richiamo ai tag. Un tag è un metadato, rappresentato da una parola chiave e riferito ad una specifica risorsa web. Il tag consente la catalogazione dei dati in internet, facilitando la ricerca di una risorsa in base alla sua descrizione. 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 dagli utenti.

La word cloud, dunque, costituisce un potente mezzo che consente agli utenti di avere un'intuizione sul contenuto di un documento. Negli anni, sono state sviluppate varie applicazioni che generano word cloud, ognuna con pregi e difetti. Le word cloud

(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.

Figura 2.2: Word cloud generata dal discorso del presidente Mattarella.

tradizionali, cioè le prime ad essere realizzate, dispongono le parole in maniera randomica o in ordine alfabetico. Tuttavia, esse non sono utili, ad esempio, 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.

Recentemente, la maggior parte dei tool che genera 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. La possibilità di disegnare, vicine nella word cloud, parole correlate semanticamente, può aggiungere informazione utile all'utente, come notato da Deutsch et al. in [2]. Il raggruppamento semantico delle parole, infatti, aiuta l'utente a comprendere più efficientemente il contenuto della word cloud: invece che analizzare le parole singolarmente, l'utente può semplicemente dare un rapido sguardo ai gruppi di parole.

Le word cloud hanno diverse caratteristiche: ciò che viene notato immediatamente dall'utente è la dimensione delle parole. Infatti, 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 di evidenziare ciò che è importante in un testo. Ci sono anche altri parametri da tenere in considerazione. In [3], Halvey e Keane 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 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. [4] 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.

Ad ogni modo, le word cloud non devono mirare troppo ai soli parametri estetici, cioè non devono avere troppi colori, parole con diverse orientazioni ecc... Dovendo fornire all'utente informazione in breve tempo ed in maniera intuitiva, devono essere più semplici e chiare possibile.

2.1.2 Applicazioni

I contesti applicativi delle word cloud sono molteplici. Esempi di applicazioni, come detto, riguardano l'analisi e la comparazione di testi scritti, ma anche:

- analisi statistiche relative ad un sito web: parole chiave utilizzate nei motori di ricerca che hanno condotto al sito web, statistiche relative ai visitatori (ad esempio, il paese di provenienza o le sezioni più cliccate);
- analisi dei *trending topic*, tramite *hashtag*, dei vari social network (Twitter, Facebook, ecc...);
- la visualizzazione dei risultati di un motore di ricerca;
- utilizzo didattico come strumento di apprendimento per gli studenti;
- ecc..

Le word cloud, quindi, costituiscono strumenti utili per fornire all'utente un'informazione sommaria su dei dati. Poi sta all'utente analizzarne il contenuto e ricavarne più informazione possibile.

2.2 Stato dell'arte

Esistono diversi strumenti per la creazione di word cloud. Un tool web-based molto popolare, Wordle [5], 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 [6], 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 [7], tramite l'uso delle *sparklines*, mette in risalto i cambiamenti tra più word

cloud. Collins et al. [8], hanno presentato Parallel Tag Clouds, uno strumento in grado di visualizzare le differenze tra i testi scritti di un ricco dataset. FacetAtlas [9], 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. Tree Cloud [10], è un tool 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 sul concetto di word cloud dinamica (cioè tempo variante), sullo stato dell'arte e alcuni cenni sui possibili contesti d'uso. 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

3.1.1 Word cloud dinamiche e stato dell'arte

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. Un lavoro rilevante, che tiene conto dell'evoluzione semantica e temporale di un insieme di documenti è stato effettuato da Cui et al. [11]: essi hanno proposto un sistema che abbina un grafico di tendenza (*trend chart*) alle word cloud ottenute da documenti appartenenti ad una collezione.

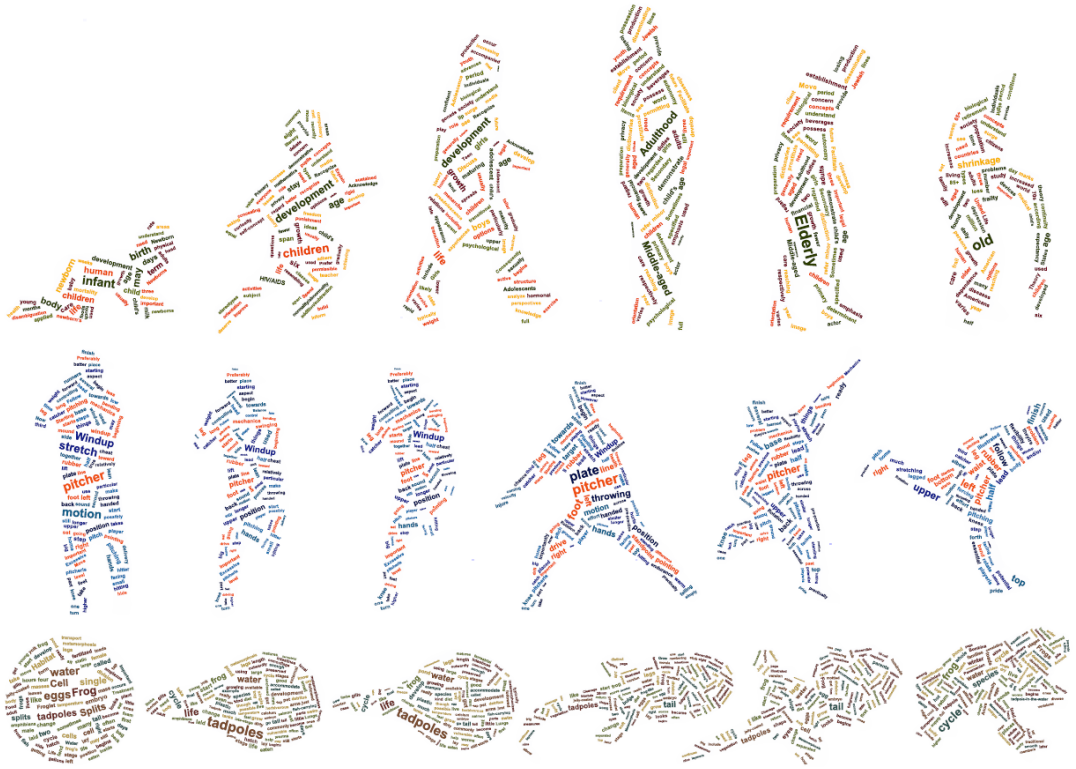


Figura 3.1: Evoluzione di diverse word word cloud tramite tecniche di morphing [17]: in alto, è mostrato il ciclo di vita dell'uomo; in mezzo, sono illustrati i movimenti effettuati dal pitcher nel baseball; in basso, infine, è mostrato il ciclo di vita di una rana.

Sebbene tutti questi lavori abbiano come finalità quella di visualizzare l'andamento 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 il progresso temporale di un 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 (figura 3.1). Tuttavia, come in [11], la visualizzazione è relativa ad un insieme di documenti, quindi il periodo temporale preso in considerazione è piuttosto ampio. Inoltre, 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, si pone come finalità quello di mostrare l'evoluzione di un solo testo, cioè il periodo temporale da cui si genera la word cloud è (relativamente) breve, poichè si prende in considerazione un solo documento. Per questo motivo, diventa significativo avere delle animazioni, che permettono di visualizzare l'evoluzione della word cloud insieme all'avanzamento dell'input. A tal fine, viene utilizzata una tecnica di morphing tra le word cloud generate in diversi istanti di tempo durante l'elaborazione del testo, rispettando, se possibile, il

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

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.

3.1.2 Applicazioni

I possibili utilizzi delle word cloud dinamiche, prodotte a partire da un unico testo di input, riguardano i contesti in cui si vuole visualizzare l'evoluzione di un generico documento relativo ad un breve periodo di tempo. Gli ambiti applicativi sono diversi, tra cui quello:

- didattico, allo scopo, per esempio, di fornire uno strumento di ausilio nella comprensione di un testo;
- divulgativo, per mostrare l'evoluzione di un discorso. Ciò può essere di grande utilità per persone non udenti oppure nel caso di video riprodotti in luoghi con scarsa udibilità;
- statistico, per illustrare l'evoluzione, in un breve periodo di tempo (ad esempio, in una giornata), dei trending topic dei vari social network;
- ecc...

3.2 Algoritmi di generazione di una word cloud semantica

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 3.2. Infine, viene applicato un algoritmo di clustering per assegnare lo stesso colore a parole dello stesso cluster.

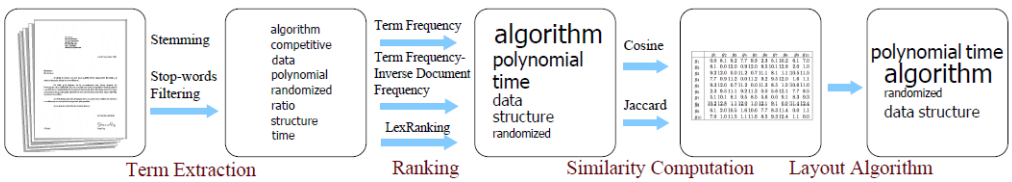


Figura 3.2: Generazione di una word cloud semantica.

Di seguito vengono prima esposti alcuni concetti teorici utilizzati nella definizione degli algoritmi. Successivamente, vengono descritte le varie fasi necessarie alla realizzazione di una word cloud semantica.

3.2.1 Riferimenti teorici

Grafo

Un **grafo** G è una coppia (V, E), dove V è un insieme finito ed E è una relazione binaria in V. L'insieme V è chiamato insieme dei vertici di G, mentre l'insieme E è detto insieme degli archi di G. Un grafo si dice **orientato** se l'insieme degli archi E è formato da coppie di vertici ordinate, altrimenti il grafo si dice **non orientato**. Dati due vertici $u, v \in V$, un arco che collega i due vertici è denotato con (u, v) ed è detto **incidente** nei vertici u e v. In tal caso, u e v sono gli **estremi** di (u, v), e u e v sono **adiacenti**. Se un arco entra ed esce nello stesso vertice, allora si dice **cappio** (**self-loop**).

Dato un grafo $G = (V, E)$, si dice che $G' = (V'E')$ è un **sottografo** di G se $V' \subseteq V$ ed $E' \subseteq E$. Dato un insieme $V' \subseteq V$, il sottografo di G **indotto** da V' è il grafo $G' = (V', E')$, dove $E' = \{(u, v) \in E : u, v \in V'\}$.

Cammini e cicli

Un **cammino** di lunghezza k da un u a v, con $u, v \in V$, in un grafo non orientato $G = (V, E)$, è una sequenza di vertici $(v_0, v_1, ..., v_k)$ tali che $u = v_0, v = v_k$ e $(v_{i-1}, v_i) \in E$, con $0 \leq i \leq k$. La lunghezza del cammino è pari al numero di archi in E.
Un **ciclo** è un cammino $(v_0, v_1, ..., v_k)$ in cui $v_0 = v_k$. Un cammino è **semplice** se tutti i suoi vertici sono distinti. Un ciclo è **semplice** se i vertici $(v_0, v_1, ..., v_{k-1})$ sono distinti. Un cappio è un ciclo di lunghezza unitaria.

Altre proprietà dei grafi

Un grafo non orientato $G = (V, E)$ è **connesso** se ogni coppia di vertici è collegata attraverso un cammino, ovvero se ogni vertice è raggiungibile da ogni altro vertice. Le **componenti connesse** di un grafo G sono i sottografi massimali connessi di G.

Un grafo non orientato è detto **aciclico** se non contiene cicli. Un grafo non orientato e aciclico è una **foresta**; un grafo connesso, non orientato e aciclico è un **albero**. Si noti che ciascuna componente connessa di una foresta è un albero.

Un grafo non orientato si dice **bipartito** se l'insieme dei vertici V può essere partizionato in due insiemi U e V tali che, se $(u, v) \in E$, allora $u \in U$ e $v \in V$ oppure $u \in V$ e $v \in U$. Un **matching** di G è un insieme di archi tali che non ci sono due archi nell'insieme con un vertice in comune.

3.2.2 Estrazione keywords

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

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 *play*, *player*, *played* 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 state utilizzate diverse tecniche di estrazione delle keywords, qui di seguito esposte.

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.

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 devono essere assegnati punteggi più elevati. In particolare, si definisce il parametro **IDF**

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

(**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*, che rappresenta 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 da PageRank [23], algoritmo 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 dell'autovettore definito da 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 (per approfondimenti, consultare [22])

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

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

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

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.

3.2.3 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.

3.2.4 Creazione word cloud

Gli algoritmi che producono word cloud ricevono in input una collezione di n rettangoli (detti *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 sono stati utilizzati tre algoritmi: 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 $sim_{i,j}$ della matrice di similarità, la distanza tra coppie di parole, pari a $d_{i,j} = 1 - sim_{i,j}$. 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* [24]. Ciò crea un layout iniziale con occupazione dello spazio non ottimale. Perciò si applica un algoritmo force directed, che aggiorna le posizione delle parole mantenendo invariata la topologia del grafo, ovvero le relazioni semantiche tra le parole. Tale algoritmo si basa su tre principi:

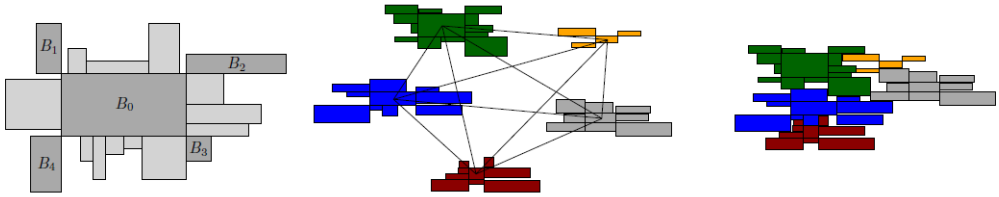
- Principio di compattazione: questo principio mira a rimuovere, per quanto possibile, gli spazi tra le parole, in modo da ottenere un layout compatto;
- Principio di non sovrapposizione: questa condizione richiede che le parole non siano sovrapposte, proprietà fondamentale per la leggibilità della word 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**. Una foresta di stelle è 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:



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

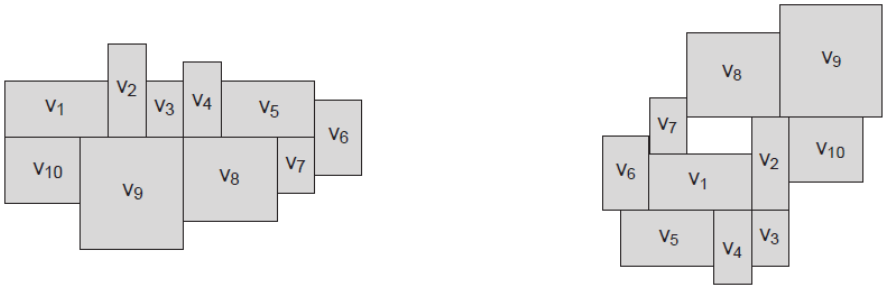
Figura 3.3: Algoritmo Star Forest.

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 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 3.3(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 [25].

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 3.3(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



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

Figura 3.4: Algoritmo Cycle Cover.

$k(1 - sim(s_1, s_2))$, 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.

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 3.4(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.

3.2.5 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 [26], 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'accuratezza 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

$$\phi = \sum_{i=1}^{K} \sum\nolimits_{x \in C_i} dist(x, c_i), \tag{3.10}$$

dove C_i è l'i-esimo cluster e $dist(x, c_i)$ è una qualche misura di prossimità tra il generico elemento $x \in C_i$ e il centroide c_i dell'i-esimo cluster. In pratica, ciò che si vuole ottenere,

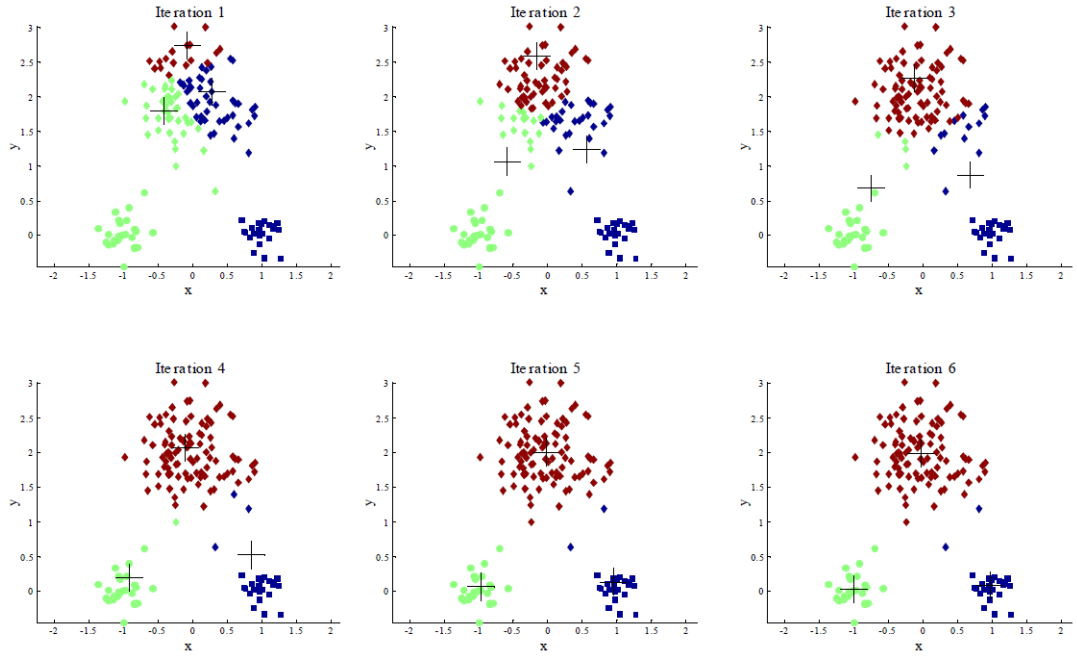


Figura 3.5: K-means: esempio di esecuzione dell'algoritmo [27]. In questo caso si ottengono $K = 3$ cluster.

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 3.5):

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 si osservano ulteriori variazioni nell'insieme dei centroidi.

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 [27].

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.

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:

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 dalla tecnica K-means++, l'algoritmo riesce a trovare 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 semantica 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 keyword, 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

$$\phi = \sum_{i=1}^{K} \sum\nolimits_{w \in C_i} (1 - sim_{w,c_i}) \tag{3.11}$$

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. In realtà, nel lavoro di Cui et al. [11], l'algoritmo di disegno CPWCV, presentato nel paragrafo 3.2.4, è sfruttato in un contesto dinamico: vengono estratte le parole più rilevanti di un insieme di documenti, si crea quindi un layout iniziale, per poi ottenere la word cloud di uno specifico documento filtrando le parole non comuni ai due layout. Tuttavia, l'evoluzione della word cloud è analizzata considerando periodi di tempo piuttosto lunghi.

L'obiettivo del nostro lavoro è invece quello di mostrare l'evoluzione della word cloud di un documento, relativo ad una breve intervallo di tempo. A tal fine, vengono create diverse word cloud del testo, considerando intervalli regolari, dove ogni word cloud è dipendente dalle precedente. 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 layout durante il 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:

1. 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;

2. 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 per il calcolo della similarità descritti in precedenza (vedi paragrafo 3.2.3), così da accoppiare 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 la sezione 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 assegnato a w_i in Γ^k nella fase di estrazione delle parole;

- una volta ottenuto, durante la realizzazione di Γ^k con uno degli algoritmi di disegno descritti precedentemente, un layout iniziale tramite l'applicazione dello scaling multidimensionale, si applica un metodo force directed che, iterativamente, esercita una forza attrattiva da τ_i^k verso τ_i^{k-1}. Questa forza è proporzionale ρ_i^k, cioè $f_{k-1,k} \propto \rho_i^k$. Ciò vale a dire che le parole con peso maggiore in Γ^k tendono a muoversi poco, mentre le parole meno rilevanti sono più soggette a spostamenti tra un disegno e l'altro.

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 (le parole, infatti, possono cambiare cluster tra un disegno e l'altro).

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 e la successiva.

Il procedimento adottato è 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 rispettivamente in Γ^{k-1} e in Γ^k, entrambi aventi dimensione $l, m > 0$.
- 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 nella word cloud 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 una word cloud ed un'altra, il cambiamento di stato delle parole, cioè le variazioni di posizione e dimensione. Un approccio simile è stato usato anche a livello di interfaccia grafica per la gestione delle variazioni di colore delle parole.

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;

Figura 3.6: Esempio di parola che compare in Γ^k ma non in Γ^{k-1}. In figura è illustrato lo stato della parola in 3 istanti consecutivi.

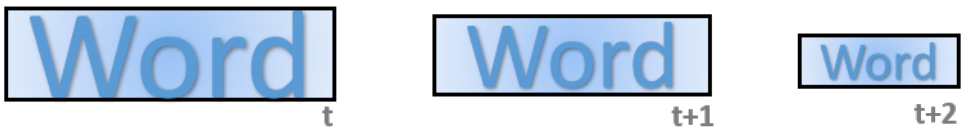


Figura 3.7: Esempio di parola che compare in Γ^{k-1} ma non in Γ^k. In figura è illustrato lo stato della parola in 3 istanti consecutivi.

- $C = \{w_1, ..., w_c\}$ 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, si può assumere che essa avrà un punteggio pari a 0 in Γ^{k-1}. Avendo N frame compresi tra Γ^{k-1} e Γ^k, il punteggio viene incrementato, ad ogni frame, di una quantità pari a $\frac{\rho_i^k}{N+1}$. Di conseguenza, in base al punteggio corrente della parola, anche la dimensione del rispettivo bounding box viene opportunamente modificata ad ogni frame (vedi figura 3.6).

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 si assume che 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 $\frac{\rho_i^{k-1}}{N+1}$. Come nel caso precedente, la dimensione del rispettivo bounding box viene opportunamente modificata ad ogni frame in base al punteggio corrente della parola (figura 3.7).

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 $\Delta_{\rho_i}^{k,k-1} = |\rho_i^k - \rho_i^{k-1}|$ la differenza in valore assoluto tra i due punteggi, se:

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

   In base al punteggio corrente, ad ogni frame la dimensione del rispettivo bounding box di w_i viene opportunamente aggiornata. Ovviamente, il peso rimane invariato nel caso in cui il punteggio non cambia tra un disegno e l'altro.
   Inoltre, in questo caso, le parole vengono anche traslate, quindi le posizioni dei rispettivi bounding box variano. Siano dunque:

   - τ_i^{k-1} il bounding box di w_i in Γ^{k-1}, con centro avente coordinate pari a $\mu_i^{k-1} = (x_i^{k-1}, y_i^{k-1})$;
   - τ_i^k il bounding box di w_i in Γ^k, con centro avente coordinate pari a $\mu_i^k = (x_i^k, y_i^k)$;
   - $d(\mu_i^{k-1}, \mu_i^k)$ la distanza tra τ_i^k e τ_i^{k-1}, cioè $d(\mu_i^{k-1}, \mu_i^k) = ||\mu_i^k - \mu_i^{k-1}|| = \sqrt{(x_i^k - x_i^{k-1})^2 + (y_i^k - y_i^{k-1})^2}$.

   Ad ogni frame, il bounding box viene traslato, in valore assoluto, di una quantità pari a $\dfrac{d(\mu_i^{k-1}, \mu_i^k)}{N+1}$. 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 3.8.

Per quanto riguarda il colore delle parole, si considera il modello di colori RGB (Red, Green, Blue), secondo cui un'immagine può essere scomposta in tre colori principali, cioè rosso, verde e blu. Ogni colore è espresso da un intero compreso in $[0, 255]$, che è l'intervallo che rappresenta lo spettro dei colori dal nero (valore pari a 0) al bianco (valore pari a 255). Ne segue quindi che ogni parola assume un colore dato dalla combinazione di tre interi. La trasformazione da un colore ad un altro viene gestito in modo indipendente per le tre componenti del colore. Dunque, le considerazioni che seguono valgono per ognuna delle componenti del colore.

Come sopra, siano dati due disegni consecutivi Γ^{k-1} e Γ^k, intervallati da N frame, e siano dati due interi x e y, rappresentanti la stessa componente (rosso, verde o blu) di un colore, rispettivamente in Γ^{k-1} e in Γ^k. Volendo raggiungere il valore di y a partire da x, la variazione (incremento o decremento, a seconda dei casi) di colore ad ogni frame è pari a $\Delta(x, y) = \left\lceil \dfrac{|x-y|}{N+1} \right\rceil$.
Dunque, data una parola w_i, si hanno i seguenti tre casi:

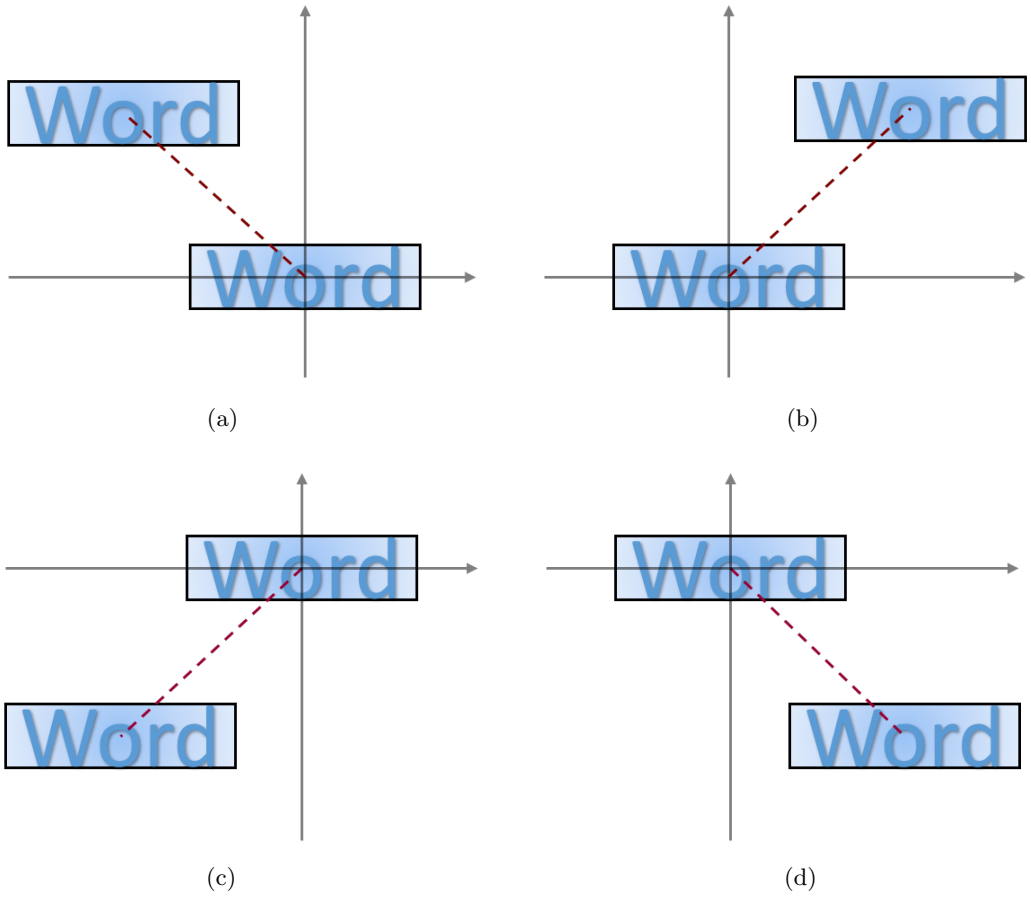


Figura 3.8: Morphing delle parole comuni a Γ_i^{k-1} e Γ_i^k. Al centro c'è Γ_i^{k-1}, mentre Γ_i^k può trovarsi in uno dei quattro quadranti.

1. $w_i \in C$: w_i compare solo in Γ^k, con colore espresso dall'intero y. Poichè w_i non è presente in Γ^{k-1}, si può assumere che, per raggiungere l'intero y, il valore x di w_i sia inizialmente pari a 255, valore che rappresenta il bianco (che è lo sfondo della zona di disegno). In altre parole, ad ogni frame, si effettua un decremento pari a $\Delta(x, y) = \left\lceil \frac{x-y}{N+1} \right\rceil$, dove $x = 255$.

2. $w_i \in S$: w_i compare solo in Γ^{k-1}, con colore espresso dall'intero x. In maniera opposta al caso precedente, si parte da x e si vuole raggiungere l'intero 255. L'incremento, ad ogni frame, è $\Delta(x, y) = \left\lceil \frac{y-x}{N+1} \right\rceil$, dove $y = 255$.

3. $w_i \in P$: w_i compare in Γ^{k-1} con colore dato dall'intero x e in Γ^k con colore dato dall'intero y. Per passare da x a y, si considerano due casi:

   - se $x < y$, allora l'incremento, ad ogni frame, è $\left\lceil \frac{y-x}{N+1} \right\rceil$;
   - se $x > y$, allora il decremento, ad ogni frame, è $\left\lceil \frac{x-y}{N+1} \right\rceil$;

Inoltre, poichè viene usata la parte intera superiore, la variazione minima di colore in ogni frame è pari ad 1. Ciò significa che, una volta raggiunto il valore cercato (cioè y), il colore non varia più.

Capitolo 4

Implementazione e sperimentazione

L'architettura del sistema e la fase di sperimentazione sono esposte in questo capitolo.

La sezione 4.1 spiega, a grandi linee, i dettagli implementativi degli algoritmi presentati nel precedente capitolo, facendo uso di class diagram per descrivere i moduli funzionali che compongono il sistema. La sezione 4.2 riporta invece i test eseguiti: dapprima vengono descritte le metriche utilizzate, poi i risultati ottenuti sono analizzati con l'ausilio di figure e tabelle.

4.1 Architettura del sistema

Il sistema è stato scritto interamente con il linguaggio di programmazione *Java* in ambiente *Eclipse*.

La word cloud viene generata a partire da un file di testo (formato *.txt*) in input. Tuttavia, il software è dotato di una classe, `SrtFileCleaner`, la quale permette di ricevere in input un file di testo in formato *.srt*, che è la rappresentazione di un generico sottotitolo con relativi timestamp, e lo ripulisce, in modo da ottenere un semplice file di testo. Il supporto a questo tipo di file è utile perchè la word cloud dinamica si può creare a partire da video che contengono il relativo sottotitolo (ad esempio, a partire dai video di YouTube).

Come descritto nel capitolo precedente, ogni fase può essere svolta da più algoritmi. L'utilizzo del pattern **Strategy** [28], rende l'architettura parametrica rispetto a tali algoritmi e consente loro di essere intercambiabili. Ciò, inoltre, riduce notevolmente la probabilità di introdurre errori nel caso si vogliano aggiungere nuovi algoritmi o modificare quelli già implementati.

Il file di testo viene elaborato opportunamente in base al numero di campionamenti (istanti di generazione della word cloud) scelto dall'utente e in base al numero di parole

da visualizzare (classe `Document`). L'elaborazione del testo crea le strutture dati (lista delle parole estratte e conteggio delle co-occorrenze tra le parole) che costituiscono l'input dell'algoritmo di calcolo della similarità.

Calcolate le parole e le loro similarità, si crea il grafo delle parole. Ogni grafo viene salvato su una lista di dimensione uguale al numero di campionamenti del testo. Ciascun elemento di tale lista rappresenta il parametro dell'algoritmo di layout che realizza il disegno (implementazione dell'interfaccia `LayoutStrategy`). Inoltre, ogni algoritmo di layout mantiene un riferimento al disegno precedente, in modo da consentire la preservazione della mappa mentale tra layout consecutivi (vedi paragrafo 3.3.1). I disegni (istanze di tipo `LayoutResult`) vengono quindi salvati su una lista di dimensione pari al numero di campionamenti del testo.

L'algoritmo K-means++, per ogni elemento della lista dei layout, esegue il clustering delle parole. La classe `ClusterColorHandler` invoca l'algoritmo di clustering e, inoltre, avendo il riferimento al cluster precedente, gestisce le variazioni dei cluster tra due disegni successivi.

La dinamicità viene gestita dall'algoritmo di morphing descritto nel precedente capitolo: viene creata una lista di oggetti `LayoutResult` di dimensione pari al numero di frame scelti. Il metodo `setFrames(int frames)` permette all'utente di scegliere il numero di frame desiderato tra un layout ed un altro.

La variazione graduale dei colori, tra un frame e l'altro, è realizzata dalla classe `SimpleColorMorphing`, la quale calcola le variazioni dei colori e assegna, ad ogni frame, il giusto colore alle parole.

Terminata la fase di logica applicativa, viene visualizzata la word cloud dinamica con l'utilizzo di una semplice interfaccia grafica.

4.1.1 Estrazione keywords e calcolo similarità

L'estrazione delle parole e il calcolo delle similarità, come abbiamo visto nel paragrafo 3.2.2, prevede una fase iniziale di preprocessing dell'input. Questa fase è stata realizzata con l'ausilio del toolkit open source *Apache OpenNLP* [19], basato su tecniche di Machine Learning per l'elaborazione del linguaggio naturale, che consente l'esecuzione dei diversi passaggi che elaborano in modo adeguato il testo in input, al fine di creare le strutture dati necessarie all'algoritmo di estrazione delle keywords. Dunque, grazie ai servizi forniti dal toolkit, vengono eseguiti il rilevamento delle frasi, la suddivisione del testo in token, il processo di stemming e la rimozione delle stop word. Per ciascuna di queste fasi si utilizzano dei modelli (forniti dalla libreria) per l'elaborazione della lingua inglese, ma anche altre lingue sono supportate.

L'elaborazione del testo, con conseguente estrazione delle keywords, è gestita dalla classe `Document`, la cui generica istanza ha diversi attributi (figura 4.1):

- il testo da elaborare (variabile `text`);

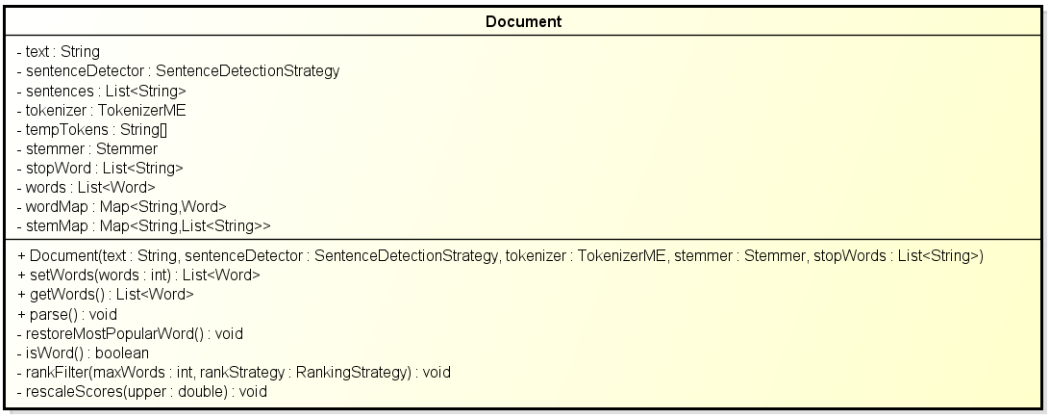


Figura 4.1: Class diagram delle classe `Document`.

- l'algoritmo di rilevamento delle frasi (variabile `sentenceDetector`);
- il tokenizer (variabile `tokenizer`);
- l'algoritmo di stemming (variabile `stemmer`);
- la lista delle stop word (variabile `stopWords`).

Nella nostra implementazione l'algoritmo di stemming utilizzato è il Porter Stemmer, fornito dalla libreria.
Il metodo che elabora il testo è `parse()`, il quale predispone in modo opportuno, tramite i parametri del costruttore, la lista delle parole da cui l'algoritmo di ranking estrarrà le parole più rilevanti. Il ranking e la conseguente estrazione delle parole è quindi gestita dal metodo `rankFilter(int maxWords,RankingStrategy rankStrategy)`, i cui parametri sono il numero di parole da estrarre e l'algoritmo di ranking. Attraverso il parametro `rankStrategy` viene invocato il metodo `rank(Document document)`, che esegue l'effettiva classificazione delle parole. Tale metodo è dichiarato nell'interfaccia `RankingStrategy` ed è realizzato dalle classi che implementano `RankingStrategy`, le quali rappresentano i diversi algoritmi di estrazione delle keywords descritti nel precedente capitolo (vedi figura 4.2).

Ogni parola è rappresentata da un'istanza della classe `Word`, che gestisce diverse caratteristiche di una parola, tra cui la radice, il punteggio e le sue variazioni, la lista di frasi in cui essa compare ecc... (figura 4.3).

Una volta estratte le keywords, si procede al calcolo della similarità, tramite uno degli algoritmi di similarità definiti in precedenza. Si utilizza un approccio parametrico come per gli algoritmi di ranking (figura 4.4).

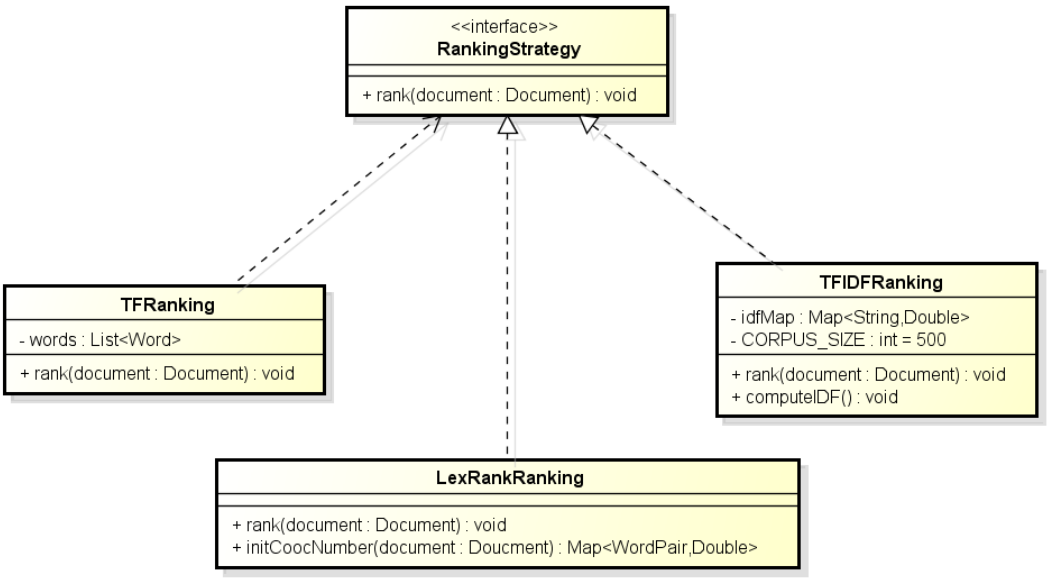


Figura 4.2: Classi coinvolte nell'estrazione delle parole. Nell'implementazione dell'algoritmo TF-IDF, è stato usato il corpus Brown [29].

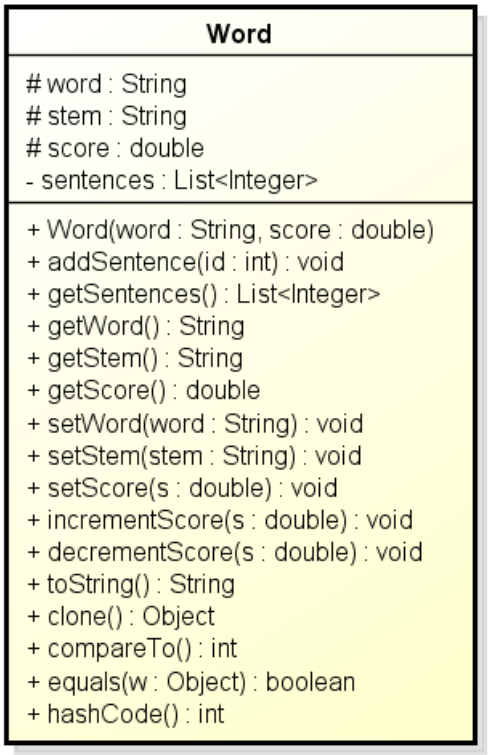


Figura 4.3: Class diagram delle classe `Word`.

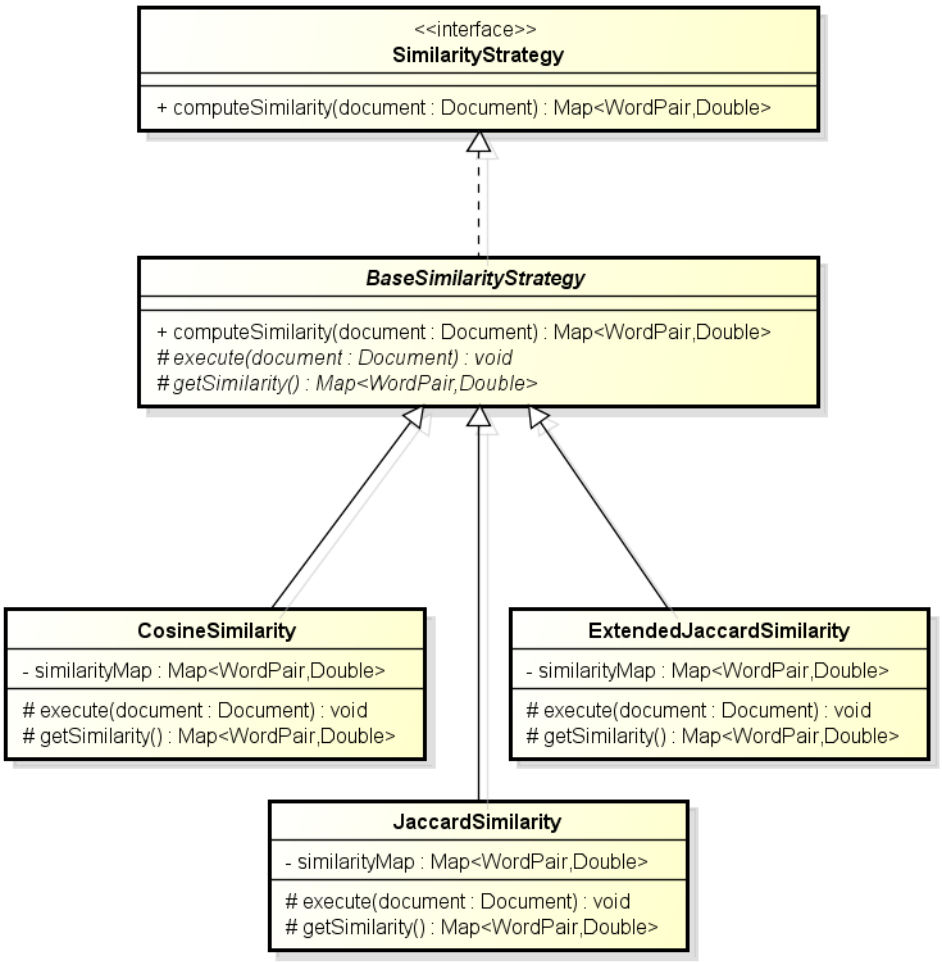


Figura 4.4: Classi coinvolte nel calcolo della similarità tra parole.

4.1.2 Creazione word cloud dinamica

Layout

Ottenute la lista delle parole più rilevanti e la similarità a coppie, si procede con la costruzione del grafo delle parole tramite la classe `WordGraph`, il cui costruttore ha come parametri proprio la lista delle parole e la mappa delle similarità tra coppie di parole. Gli algoritmi di layout creano il disegno a partire da un oggetto `WordGraph` e salvano tutte le relative informazioni in un'istanza di `LayoutResult`. I servizi comuni tra le classi che effettivamente realizzano gli algoritmi sono esposti nella classe astratta `BaseLayoutStrategy`. In particolare, si hanno i seguenti metodi:

- `layout(WordGraph wordGraph)`, che inizializza alcune variabili di utilità, per poi chiamare il metodo astratto `execute()`, il quale a tempo di esecuzione sarà ovviamente concreto e verrà eseguito a seconda del tipo effettivo di `LayoutStrategy`;
- `createResult()`, il quale crea un oggetto di tipo `LayoutResult` e lo salva nella variabile `lastResult`. Questo riferimento sarà la chiave per il mantenimento della mappa mentale nel passaggio da una word cloud ad un'altra;
- `createBoundingBoxes()` associa, ad ogni parola contenuta nella lista `words` (contenente oggetti di tipo `Word`), il relativo bounding box, che non è altro che un oggetto `Rectangle`. Questa mappatura viene salvata nella variabile `wordPositionsMap`. Per generare il bounding box, Java consente di estrarre il bounding box di una sequenza di caratteri tramite il toolkit Graphics2D. La dimensione del bounding box viene quindi opportunamente scalata in base al relativo punteggio di ciascuna parola.

La coerenza della mappa mentale, come detto, viene rispettata grazie al riferimento all'ultimo disegno realizzato, denotato dalla variabile `lastResult`. Tale variabile viene ereditata dallo specifico algoritmo di layout ed è gestita in modo opportuno nell'implementazione della procedura force directed descritta nel paragrafo 3.3.1. Tale procedura viene eseguita subito dopo l'applicazione dello scaling multidimensionale al grafo delle parole, che colloca sul piano le parole in base alla loro similarità. Il metodo force directed, grazie al riferimento all'ultimo layout, riesce a ricavare le parole in comune tra i due layout. Per ognuna di queste parole, viene calcolata la distanza tra la vecchia e la nuova posizione, dunque si applica una forza attrattiva proporzionale al punteggio della parola nel layout corrente. Il numero massimo di iterazioni è 1000.
Di seguito è riportato lo pseudocodice della procedura force directed utilizzata.

Clustering

L'esecuzione dell'algoritmo di clustering K-means++ avviene per ciascun oggetto di tipo `WordGraph`. Per calcolare il corretto numero di cluster, viene avviato più volte l'algoritmo

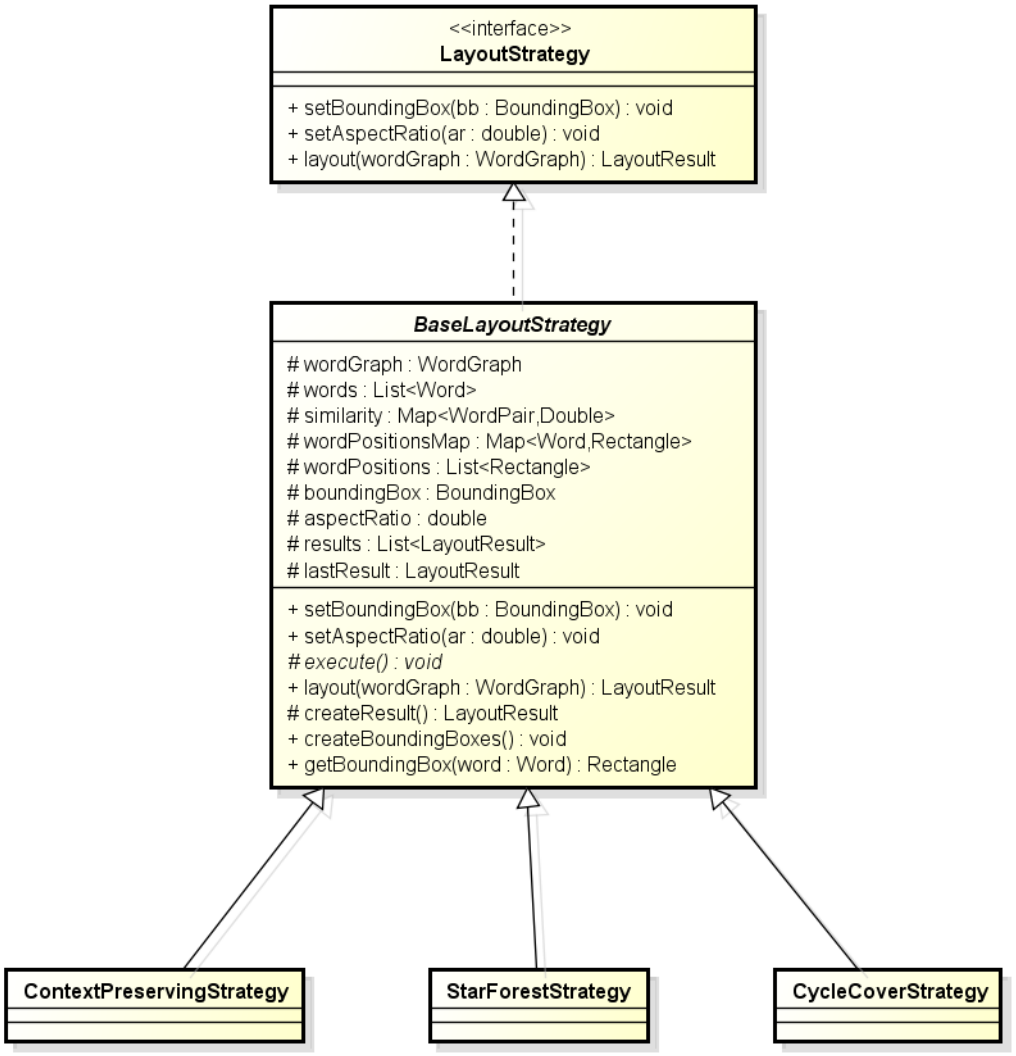


Figura 4.5: Classi coinvolte nella creazione della word cloud.

Algorithm 1 Avvio iterazione.

1: **procedure** ForceDirected–Iteration
2: siano Γ_{i-1} il layout precedente, Γ_i il layout corrente e sia W l'insieme delle parole comuni a Γ_{i-1} e Γ_i;
3: **repeat**
4: **for each** parola $w \in W$ **do**
5: sia $\tau_{i-1} \leftarrow$ posizione di w in Γ_{i-1};
6: sia $\tau_i \leftarrow$ posizione di w in Γ_i;
7: $\rho_i \leftarrow$ punteggio di w Γ_i;
8: $\delta \leftarrow$ attractiveforce$(\tau_{i-1},\tau_i,\rho_i)$;
9: muovi r_i in direzione di r_{i-1} di una quantità pari a δ;
10: **end for**
11: **until** equilibrio raggiunto OR iterazioni massime effettuate;
12: **end procedure**

Algorithm 2 Calcolo forza attrattiva.

1: **procedure** AttractiveForce(τ_{i-1},τ_i,ρ_i)
2: $P = (x, y) \leftarrow$ punto di coordinate x e y corrispondenti alle distanze dei centri di τ_{i-1} e τ_i;
3: $\bar{P} \leftarrow$ normalizzazione di P;
4: **return** $\bar{P}$ scalato di una quantità proporzionale a ρ_i;
5: **end procedure**

a partire da $k = \sqrt{n/2}$, incrementando k di una unità ad ogni iterazione. Poi si esegue il confronto tra clustering successivi per selezionare quello migliore.

L'algoritmo viene eseguito dalla classe `ClusterColorHandler` (vedi figura 4.6), mediante l'invocazione del metodo `initialize(WordGraph wordGraph, ClusterResult prevResult)`. Il cluster prodotto è referenziato dalla variabile `clustering`, il cui tipo è `ClusterResult`, struttura dati che contiene tutte le informazioni relative al cluster. Ad ogni cluster è associato un intero, il quale corrisponde all'indice di un elemento dell'array dei colori da assegnare alle parole del cluster (variabile `colorSequence`). Il metodo `getColor(Word w)`, quindi, consente di restituire il colore della parola `w` tramite l'indice associato al cluster di `w`. Il metodo `updateClusters(ClusterResult prevResult)`, gestisce invece la variazione dei cluster tra due layout successivi, per mezzo dei seguenti passaggi:

- vengono calcolati i valori di similarità tra i cluster del layout precedente (variabile `prevResult`) e di quello corrente (variabile `clustering`) tramite l'algoritmo di similarità che implementa l'interfaccia `ClusterSimilarityStrategy`;
- viene invocato il metodo `computeBestPairs(List<ClusterPair> bestPairs)`, il quale salva su una lista le coppie (di interi) con i valori più alti;
- se il numero di cluster, nel layout corrente, è aumentato, allora vengono aggiunte nuove coppie alla lista `bestPairs`, le quali sono calcolate in base agli interi non ancora assegnati ad alcun cluster. Il numero di coppie aggiunto è uguale a $m - l$, dove l è il numero di cluster del disegno precedente ed m il numero di cluster del disegno attuale;
- infine, il cluster corrente viene modificato invocando `updateClusters(bestPairs)`, metodo offerto della classe `ClusterResult`, che esegue l'aggiornamento degli indici dei vari cluster;

Al termine, per ciascun oggetto `WordGraph`, si ottiene un oggetto `ClusterResult` aggiornato.

Morphing

La dinamicità è conferita dallo specifico algoritmo di morphing che implementa l'interfaccia `MorphingStrategy`. Ovviamente, anche in questo caso è possibile estendere

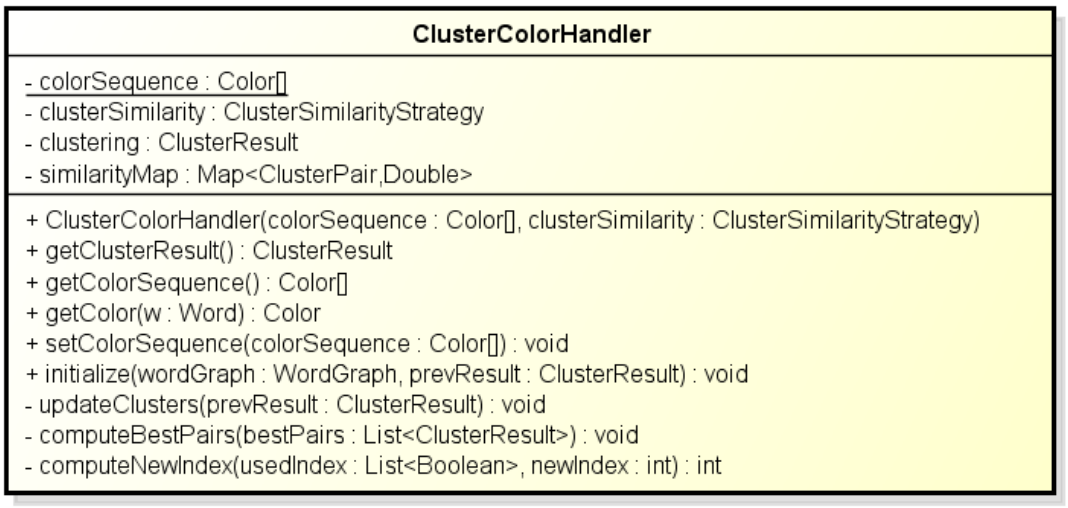


Figura 4.6: Class diagram della classe `ClusterColorHandler`.

facilmente l'architettura con nuovi algoritmi. L'interfaccia definita prevede quindi due metodi:

- il primo, `morph(LayoutResulta resultA)`, prende come parametro un solo layout ed esegue il morphing iniziale, ovvero quello in cui, nella word cloud, compaiono solo nuove parole;
- il secondo, `morph(LayoutResulta resultA, LayoutResult resultB)`, prende come parametri due layout consecutivi e ne esegue il morphing.

Questi due metodi estraggono una lista di disegni (oggetti `LayoutResult`) con le posizioni delle parole via via aggiornate. La dimensione di tale lista è pari al numero di frame scelto tra una word cloud e la successiva, definito dal parametro `frames` del costruttore dello specifico algoritmo di morphing che estende `MorphingStrategy`.
L'algoritmo di morphing utilizzato è rappresentato dalla classe `SimpleMorphing`, che implementa diversi metodi, tra cui:

- `morphNewWords(int iter)`, il quale, per ogni elemento della lista `newWords`, incrementa il punteggio ad ogni frame e restituisce una mappa in cui, a ciascuna parola, è associata un rettangolo con le dimensioni aggiornate;
- `morphDisappearingWords(int iter)`, il quale, per ogni elemento della lista `disappearingWords`, decrementa il punteggio ad ogni frame e restituisce una mappa in cui, a ciascuna parola, è associata un rettangolo con le dimensioni aggiornate;
- `morphCommonWords(int iter)`, il quale, per ogni elemento della lista `commonWords`, incrementa o decrementa il punteggio ad ogni frame. Inoltre, viene chiamato il metodo `handleRects(...)` che, a seconda dei casi (vedi paragrafo 3.3.3), aggiorna le

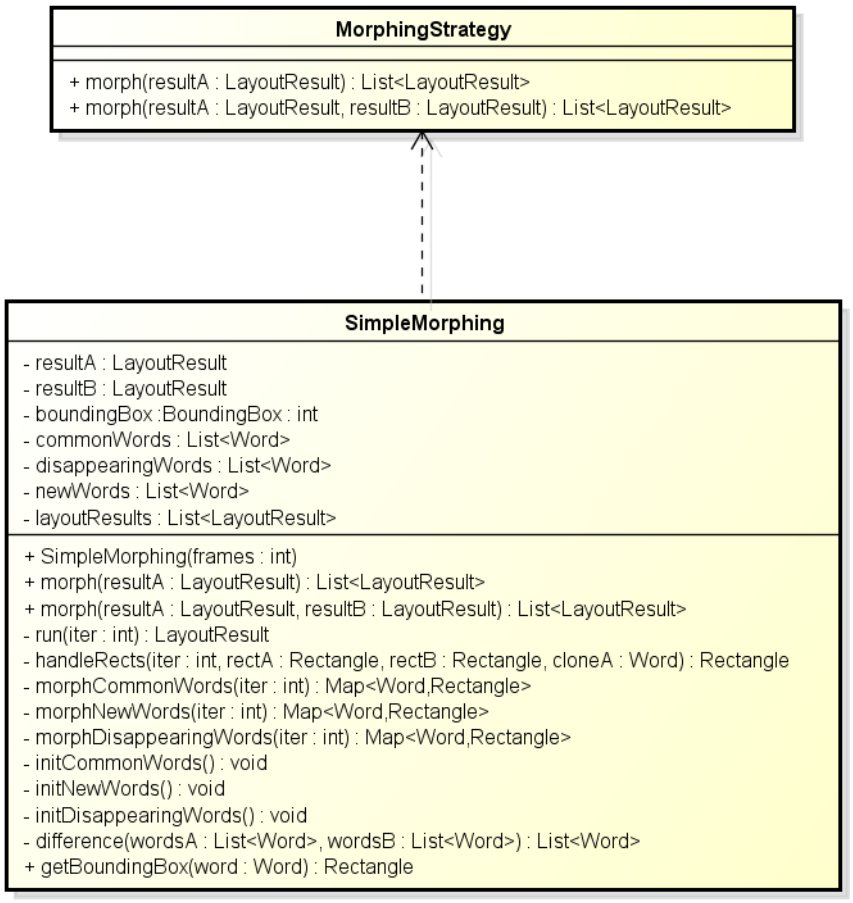


Figura 4.7: Classi coinvolte nell'operazione di morphing.

coordinate delle parole. Il metodo restituisce dunque una mappa in cui, a ciascuna parola, è associato un rettangolo con le coordinate e le dimensioni aggiornate.

Quindi, una volta terminato il morphing, se n è il numero di frame tra un layout ed un altro, vengono creati n oggetti `LayoutResult`, che rappresentano i layout aggiornati frame per frame.

La variazione del colore da un layout al successivo è invece trattata, con un approccio simile alla classe `SimpleMorphing`, dalla classe `SimpleColorMorphing`. La classe `Color` di Java codifica i colori, nel modello RGB, con interi compresi tra 0 a 255. Come sopra, i metodi `morphNewWords(...)`, `morphDisappearingWords(...)` e `morphCommonWords(...)` applicano il morphing dei colori tra due disegni consecutivi, con la differenza che il tipo restituito è un array di oggetti `Color` (nel caso dei primi due metodi) o una matrice di oggetti `Color` (nel caso del terzo metodo). Queste strutture dati vengono restituite alla classe `DynamicColorHandler`, che si occupa, ad ogni frame, di assegnare il colore esatto a ciascuna parola, tramite il metodo `getColor(Word w)`, così come nella classe `ClusterColorHandler`.

4.1.3 Interfaccia grafica

La visualizzazione dinamica della word cloud avviene per mezzo di una semplice interfaccia grafica sviluppata con il framework *Swing* di Java. L'interfaccia contiene i bottoni tipici di un qualsiasi player, i quali permettono di avviare la visualizzazione, metterla in pausa, riavviarla, passare direttamente alla successiva word cloud o alla precedente (vedi figura 4.8).

Sempre in figura 4.8 si nota, in basso a sinistra, un grafico a barre (la cui visibilità si può impostare dal menù) contenente le 10 parole più importanti ad ogni layout, ordinate in base al relativo punteggio. Tale finestra è stata realizzata tramite la libreria Java *JFreeChart* [30].

(a)

(b)

Figura 4.8: Esempio di visualizzazione dinamica di una word cloud in due layout successivi.

4.2 Risultati sperimentali

Il sistema è stato testato su un dataset di 200 testi, corrispondenti a discorsi estratti da video della conferenza *TED* [31]. Le trascrizioni dei video (file di testo *.txt*) sono state ottenute grazie al tool online TED2SRT [32]. Ogni discorso tratta temi di attualità e la durata media è di circa 17 minuti. La sperimentazione è stata infine effettuata su un calcolatore con processore Intel i7 2.40Ghz e memoria RAM 8GB.

4.2.1 Metriche adottate

Gli algoritmi di layout sono stati testati sulla base di diverse metriche. Le metriche tengono conto sia di aspetti quantitativi (compattazione, distanza tra parole correlate semanticamente) che qualitativi (mantenimento della mappa mentale) ed assumono valori compresi nell'intervallo $[0, 1]$.

Distortion metric

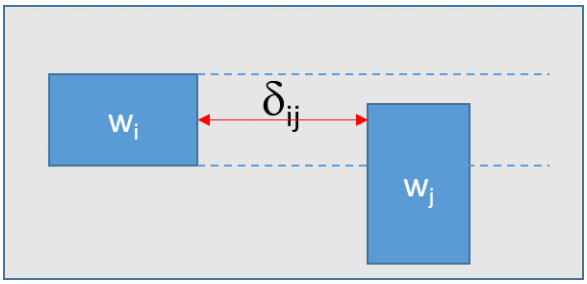
Data una word cloud Γ^k, $k \in \{1, ..., K\}$, tale metrica misura la vicinanza geometrica tra parole correlate semanticamente nel disegno. In particolare, siano:

- s_{ij} il valore di similarità, compreso tra 0 e 1, tra la parola w_i e la parola w_j ed estratto dalla matrice di similarità calcolata con uno degli algoritmi descritti in precedenza;
- d_{ij} la distanza normalizzata (cioè compresa tra 0 e 1) tra la parola w_i e la parola w_j. Essa è calcolata come $d_{ij} = \frac{\delta_{ij}}{D}$, dove δ_{ij} è la distanza minima tra due bounding box, mentre D è la massima distanza possibile nel disegno tra due punti, cioè la diagonale del più grande bounding box che contiene la word cloud. In generale, nel calcolo di δ_{ij}, si hanno tre casi (vedi figura 4.9). Se b_i, h_i e c_i rappresentano la base, l'altezza e il centro del bounding box di w_i, allora $\delta_{ij} = \sqrt{\Delta_{x_{ij}}^2 + \Delta_{y_{ij}}^2}$, dove:
  - la quantità $\Delta_{x_{ij}} = \max\{0, |c_{x_i} - c_{x_j}| - \frac{b_i + b_j}{2}\}$ rappresenta la distanza minima sull'asse x;
  - la quantità $\Delta_{y_{ij}} = \max\{0, |c_{y_i} - c_{y_j}| - \frac{h_i + h_j}{2}\}$ rappresenta la distanza minima sull'asse y.

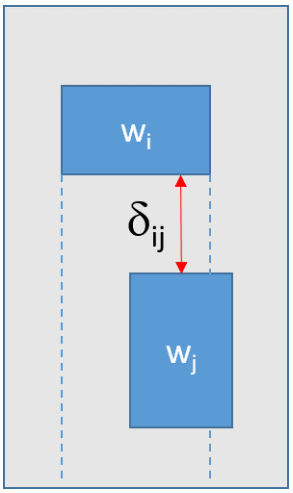
  Per ogni coppia di parole w_i e w_j, dunque, si calcola il coefficiente $c_{ij} = (1 - d_{ij})^2$.

La misura S^k della vicinanza geometrica fra parole correlate semanticamente nel disegno Γ_k viene quindi calcolata combinando c_{ij} con s_{ij} mediante la seguente formula:

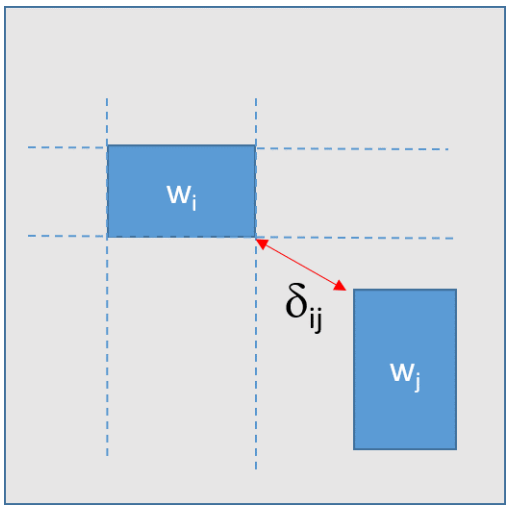
$$S^k = \frac{\sum_{ij} c_{ij} s_{ij}}{\sum_{ij} s_{ij}}, \tag{4.1}$$



(a) Distanza solo sulle ascisse.



(b) Distanza solo sulle ordinate.



(c) Distanza su entrambi gli assi.

Figura 4.9: Distanza minima tra due parole.

dove $S^k \in [0,1]$. Se tale valore è vicino a 1, allora la distanza geometrica rispecchia la similarità, altrimenti se tale valore tende a 0 vale il contrario.

Coherence metric

Il mantenimento della mappa mentale tra due disegni consecutivi è calcolato con questa metrica. 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 p parole comuni a Γ^{k-1} e Γ^k. Denotiamo con $\sigma(w_i^k, w_i^{k-1}, w_i \in P$, la variazione di posizione della parola w_i nel disegno Γ^k. Sia inoltre D la diagonale del più grande bounding box che contiene la word cloud. La metrica è definita come:

$$\vartheta^{k,k-1} = 1 - \frac{\sum_{i=1}^{p} \sigma(w_i^k, w_i^{k-1})}{pD} \tag{4.2}$$

Ne segue che:

1. $\vartheta^{k,k-1} \in [0,1]$;
2. essendo $k \in \{2, ..., K\}$, tale misura non è definita per il primo layout. Si può dunque assumere che $\vartheta^{1,0} = 1$;

Combination metric

Per ciascun disegno $\Gamma^k, k \in \{1, ..., K\}$, sono state calcolate S^k e $\vartheta^{k,k-1}$. A questo punto si può ottenere una nuova metrica, combinazione lineare delle due metriche appena definite:

$$\nu = \frac{1}{K} \sum_{k=1}^{K} (\alpha S^k + \beta \vartheta^{k,k-1}) \tag{4.3}$$

dove:

- K è il numero di word cloud realizzate;
- α e β sono due coefficienti entrambi positivi e tali che $\alpha + \beta = 1$;

La definizione di questa metrica è utile poichè, come già espresso in precedenza, la vicinanza geometrica tra parole maggiormente correlate semanticamente e la preservazione della mappa mentale cosituiscono due obiettivi contrastanti. Con i valori ottenuti da questa metrica, è possibile analizzare gli algoritmi di disegno sulla base di tali parametri e scegliere quindi l'algoritmo adatto in base alle proprie esigenze.

Space metric

L'uso efficiente dell'area di disegno viene racchiusa da questa semplice misura. Si calcola l'area utilizzata come la somma delle aree dei bounding box di ciascuna parola (μ); poi si calcola l'area del bounding box (o dell'inviluppo convesso, detto *convex hull*) che

contiene tutte le parole (φ). La **compattezza** è definita come $\gamma = 1 - \frac{\mu}{\varphi}$. Se tale valore tende ad 1, allora si ha un disegno poco compatto, mentre un valore tendente a zero denota un disegno troppo compatto.

Running time

Il calcolo del tempo d'esecuzione consiste nel conteggio del tempo necessario ad eseguire i diversi passaggi nella creazione della word cloud, a partire dall'elaborazione del testo, fino ad arrivare alla generazione dell'interfaccia grafica. Il tempo di esecuzione è stato valutato dunque per:

- l'elaborazione del testo ed estrazione delle keywords;
- il calcolo delle similarità;
- la creazione della word cloud;
- l'applicazione della procedura di morphing tra le parole;
- il clustering e la gestione delle variazioni dei cluster tra una word cloud e l'altra;
- la gestione delle variazioni dei colori delle parole;
- la generazione dell'interfaccia grafica;
- tempo totale di esecuzione.

Eccetto per gli ultimi due valori, tutti i tempi sono stati mediati sul numero di campionamenti del testo (pari a 4).

4.2.2 Risultati

La durata media dei discorsi che compongono il dataset è di circa 17 minuti, per cui si è scelto di creare le word cloud in 4 istanti, in modo da ottenere una prima finestra temporale tra i 4 e i 5 minuti e avere abbastanza parole da estrarre. Inoltre, poichè l'evoluzione della word cloud è dinamica, è bene non visualizzare troppe parole, in modo da non creare confusione all'utente. Per questi motivi, sono state realizzate, per ogni combinazione degli algoritmi utilizzati, tre word cloud da 20, 40 e 60 parole ciascuna.

Combination metric

Sono stati calcolati i valori dei tre algoritmi di layout per tutte le combinazioni possibili tra il numero di parole, l'algoritmo di ranking e l'algoritmo di similarità. Gli andamenti degli algoritmi sono riportati nelle figure delle pagine successive. I valori assunti giacciono su una retta che congiunge due punti, cioè $\alpha = 0$, che coincide con la misura della Coherence, e $\alpha = 1$, che coincide con la misura della Distortion. In entrambe le metriche,

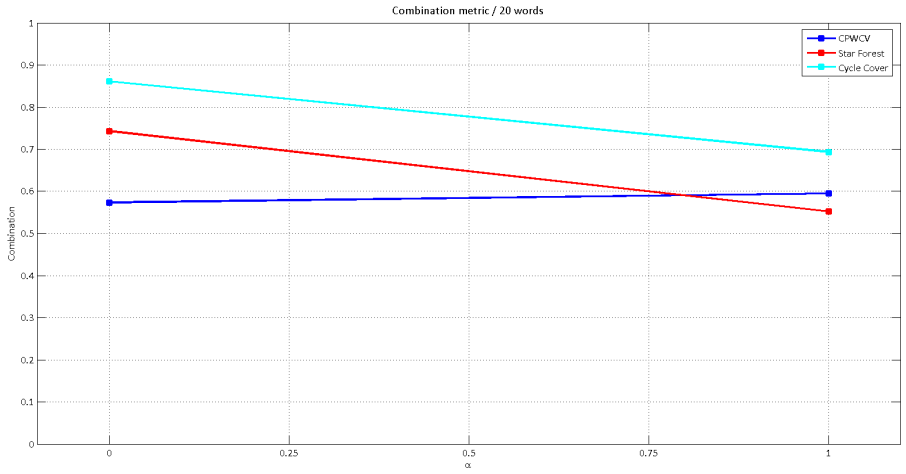


Figura 4.10: Combination metric: Term Frequency + Jaccard Similarity con 20 parole estratte.

la quantità D, cioè la diagonale del bounding box che contiene la word cloud, è presa come il massimo per ogni coppia {numero di parole,algoritmo di layout}.

L'algoritmo che offre le performance migliori è Cycle Cover, superando quasi sempre Star Forest e CPWCV. In particolare, preserva piuttosto bene la mappa mentale, mentre per quanto riguarda la Distortion, i valori assunti dai vari algoritmi tendono ad essere molto vicini e in ogni caso sono più che discreti. Com'era lecito aspettarsi, i valori più alti si osservano nel caso di 20 parole estratte, mentre all'aumentare delle parole i valori decrescono. Star Forest ha un comportamento simile a Cycle Cover, assumendo valori leggermente inferiori sulla Coherence. CPWCV è invece l'algoritmo che, il più delle volte, si comporta peggio nella misura della coerenza della mappa mentale. Ciò fa si che le parole tendano a muoversi di più, migliorando d'altro canto la Distortion. La retta che rappresenta l'andamento della CPWCV, infatti, ha solitamente un andamento non decrescente.

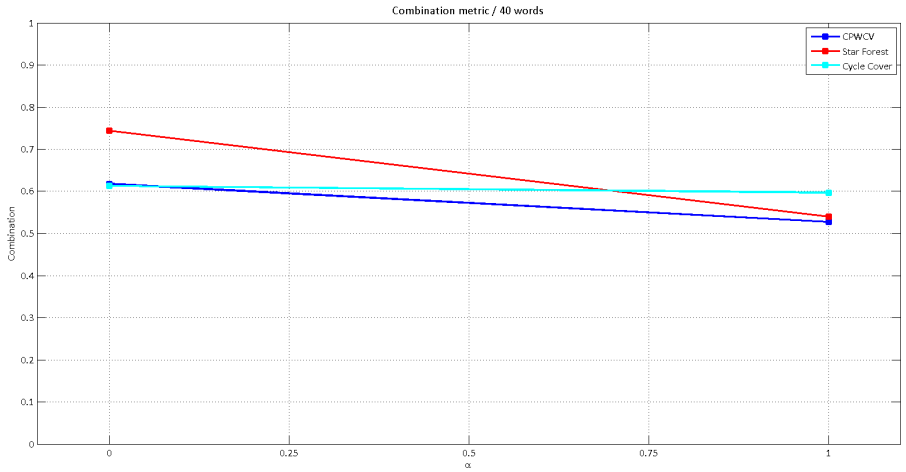


Figura 4.11: Combination metric: Term Frequency + Jaccard Similarity con 40 parole estratte.

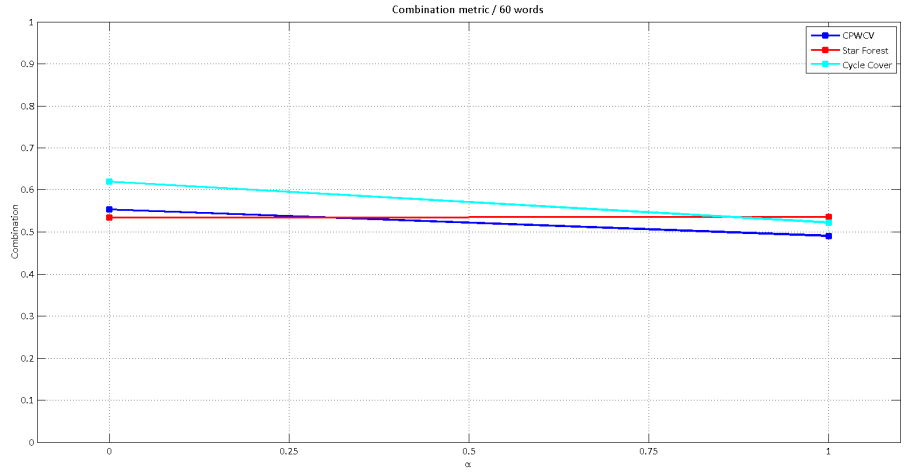


Figura 4.12: Combination metric: Term Frequency + Jaccard Similarity con 60 parole estratte.

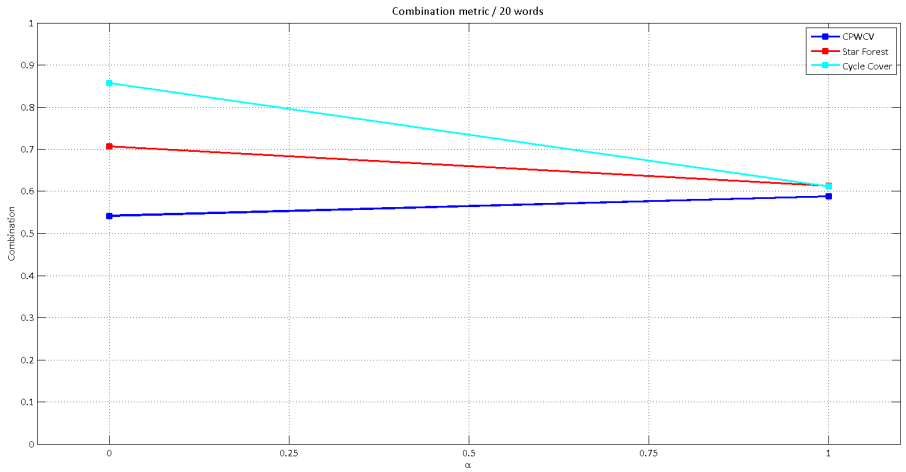


Figura 4.13: Combination metric: Term Frequency + Cosine Similarity con 20 parole estratte.

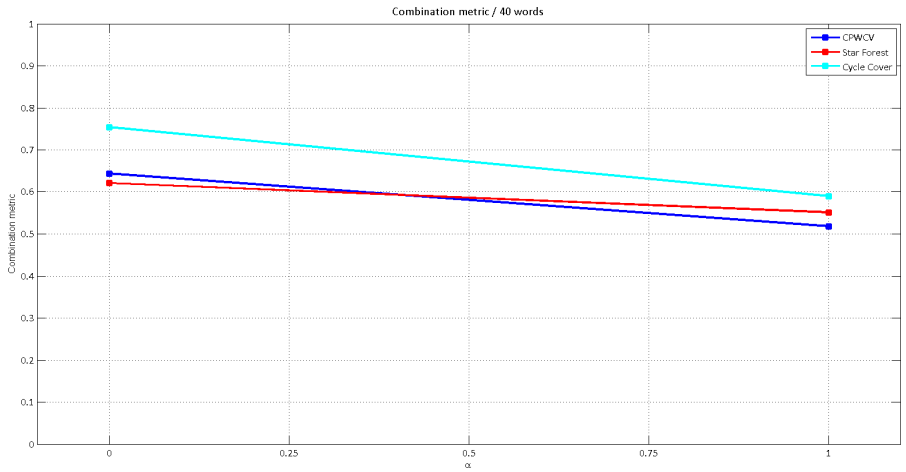


Figura 4.14: Combination metric: Term Frequency + Cosine Similarity con 40 parole estratte.

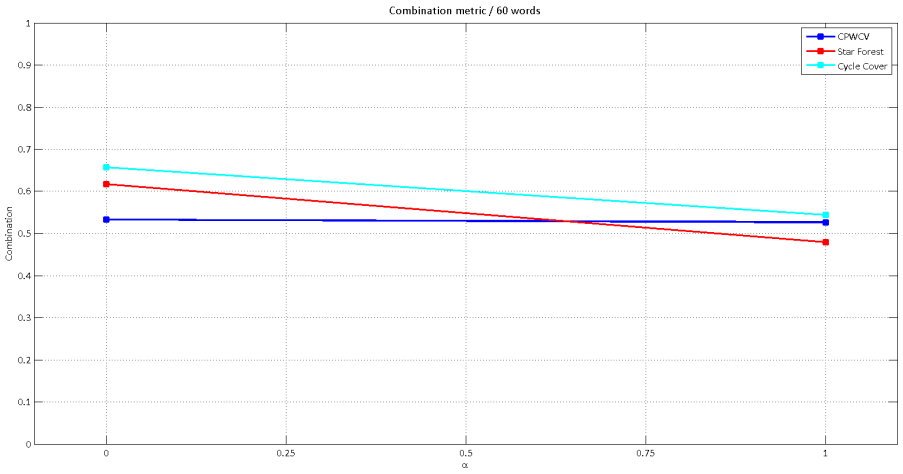


Figura 4.15: Combination metric: Term Frequency + Cosine Similarity con 60 parole estratte.

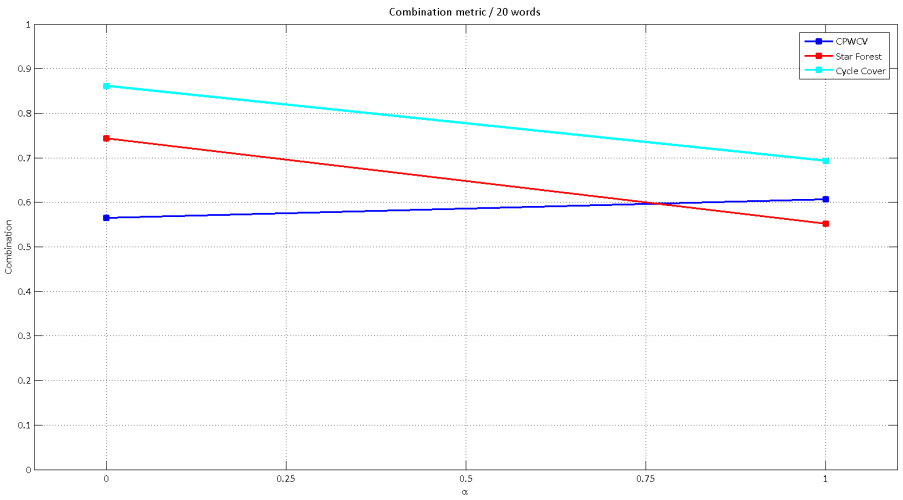


Figura 4.16: Combination metric: Term Frequency + Extended Jaccard Similarity con 20 parole estratte.

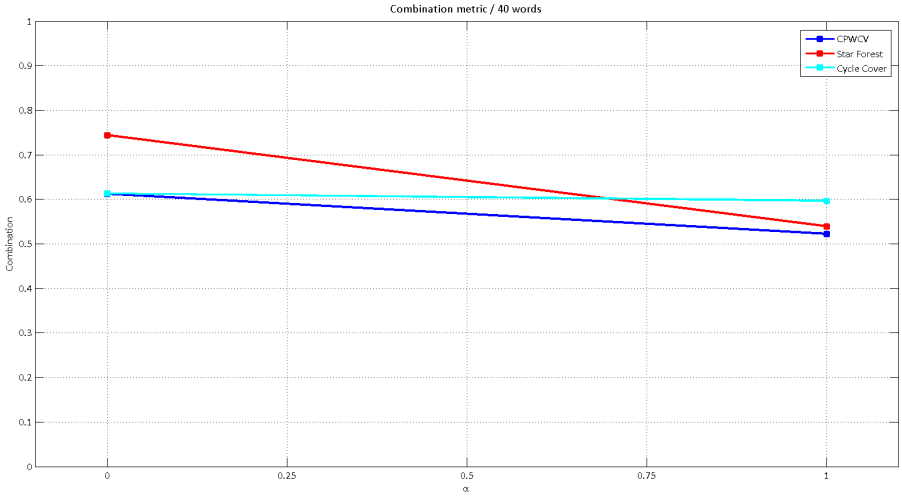


Figura 4.17: Combination metric: Term Frequency + Extended Jaccard Similarity con 40 parole estratte.

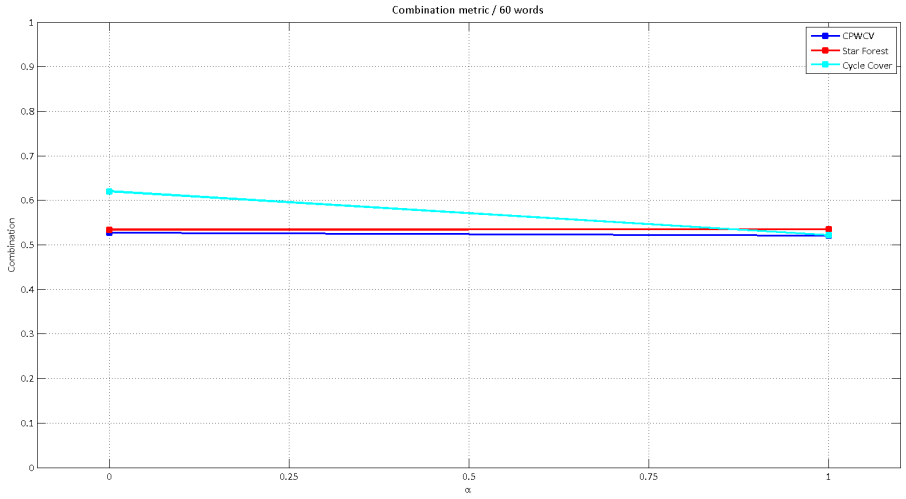


Figura 4.18: Combination metric: Term Frequency + Extended Jaccard Similarity con 60 parole estratte.

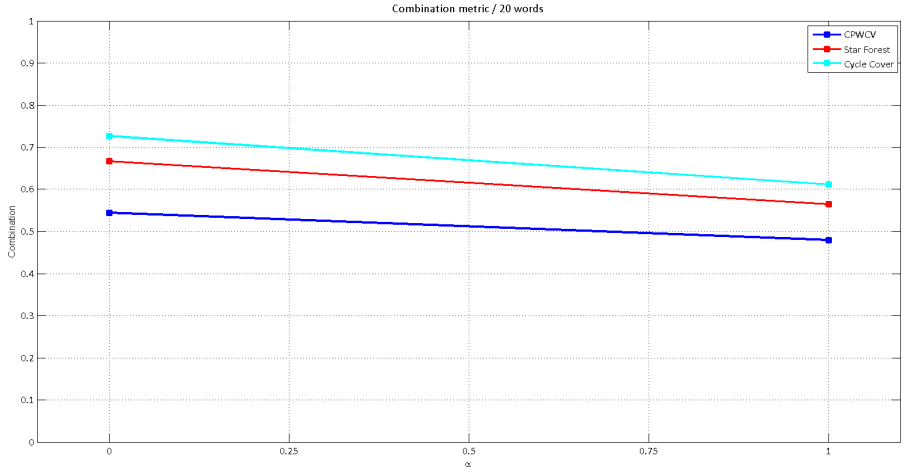


Figura 4.19: Combination metric: TFIDF + Jaccard Similarity con 20 parole estratte.

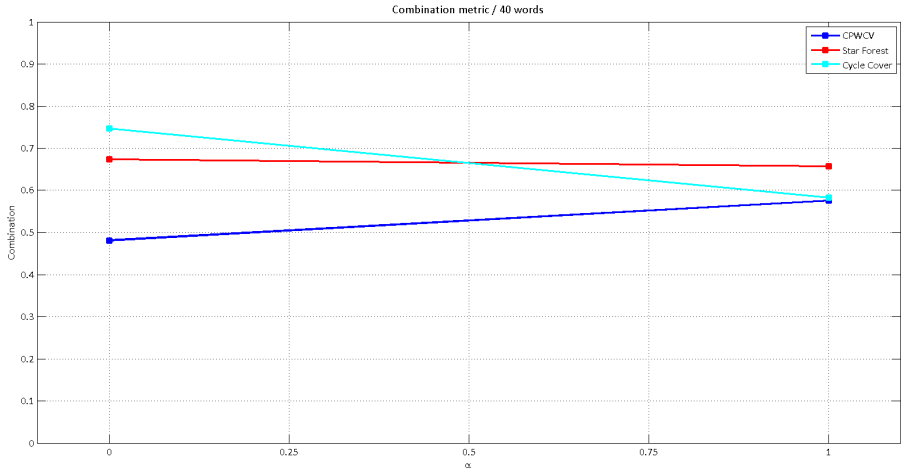


Figura 4.20: Combination metric: TFIDF + Jaccard Similarity con 40 parole estratte.

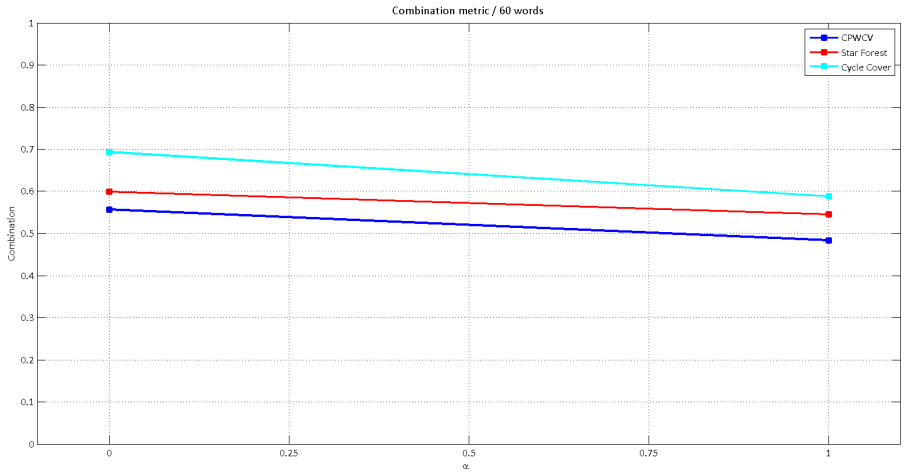


Figura 4.21: Combination metric: TFIDF + Jaccard Similarity con 60 parole estratte.

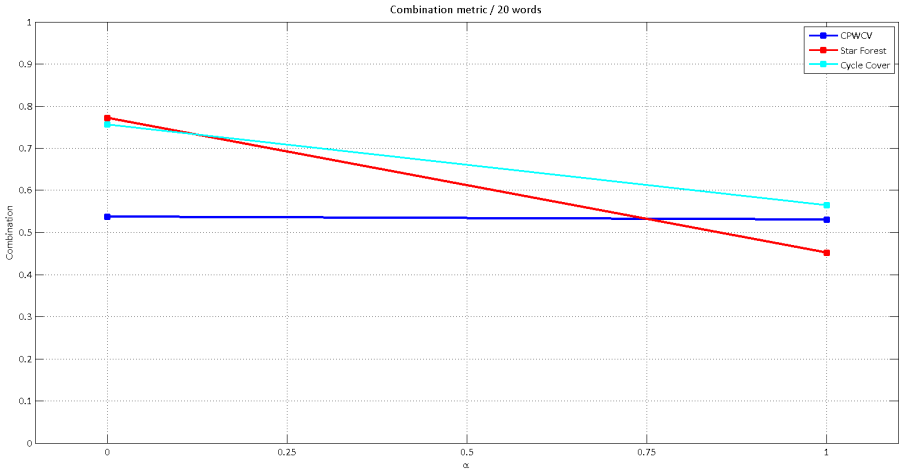


Figura 4.22: Combination metric: TFIDF + Cosine Similarity con 20 parole estratte.

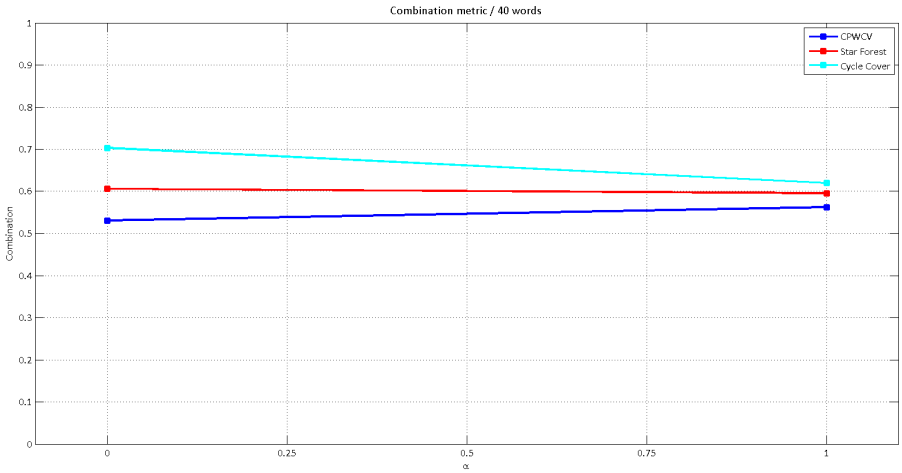


Figura 4.23: Combination metric: TFIDF + Cosine Similarity con 40 parole estratte.

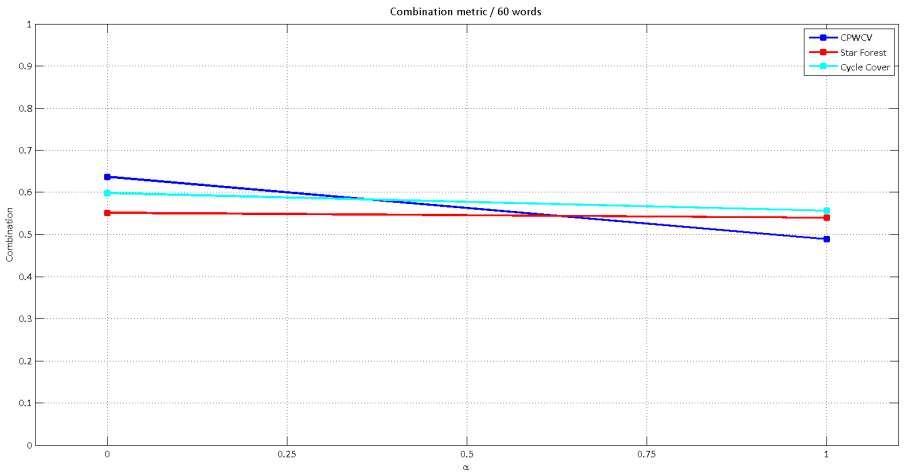


Figura 4.24: Combination metric: TFIDF + Cosine Similarity con 60 parole estratte.

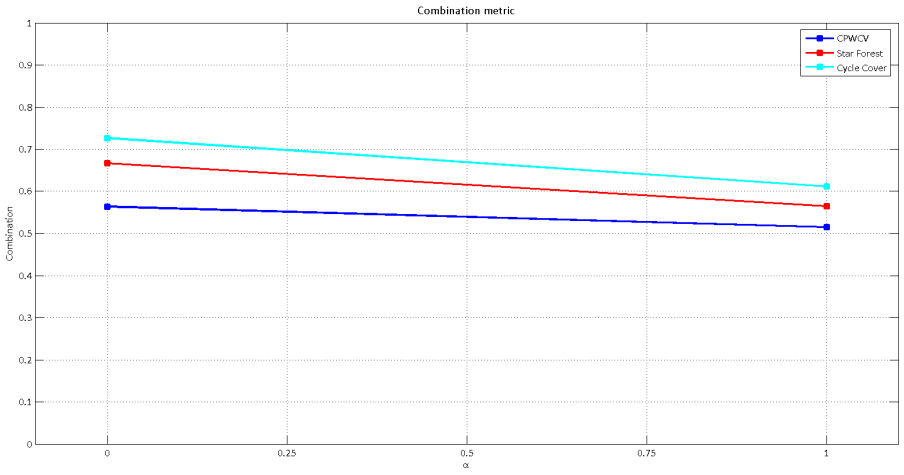


Figura 4.25: Combination metric: TFIDF + Extended Jaccard Similarity con 20 parole estratte.

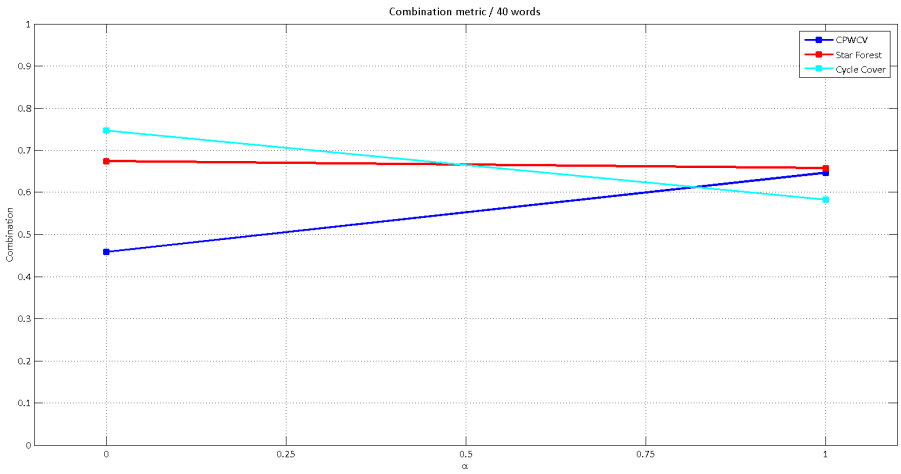


Figura 4.26: Combination metric: TFIDF + Extended Jaccard Similarity con 40 parole estratte.

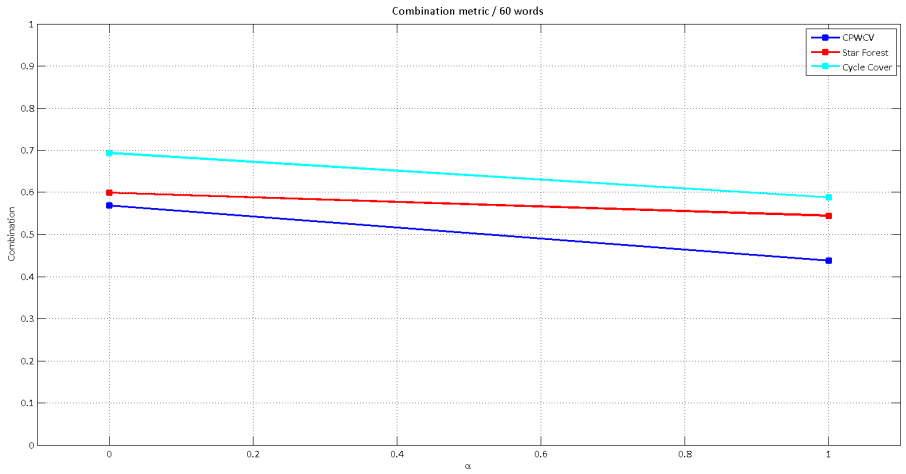


Figura 4.27: Combination metric: TFIDF + Extended Jaccard Similarity con 60 parole estratte.

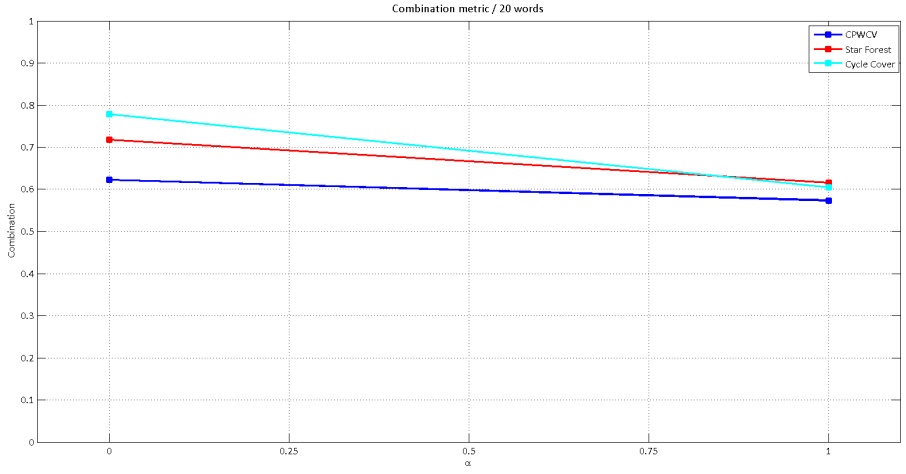


Figura 4.28: Combination metric: LexRank + Jaccard Similarity con 20 parole estratte.

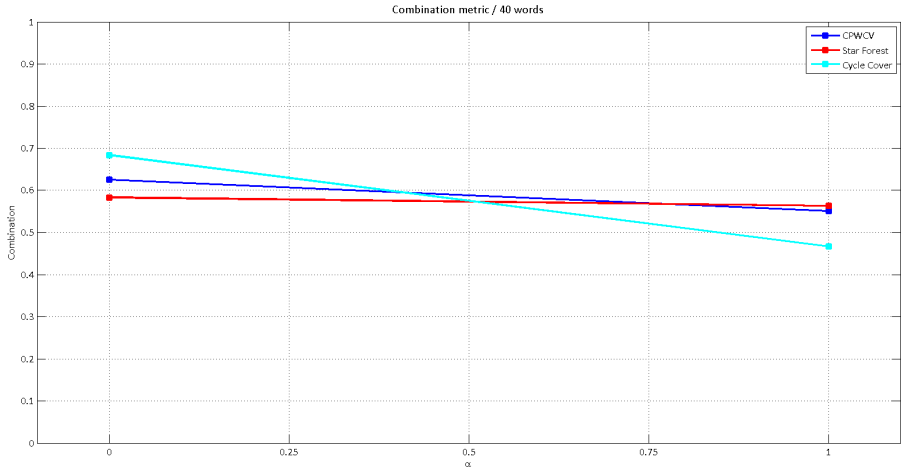


Figura 4.29: Combination metric: LexRank + Jaccard Similarity con 40 parole estratte.

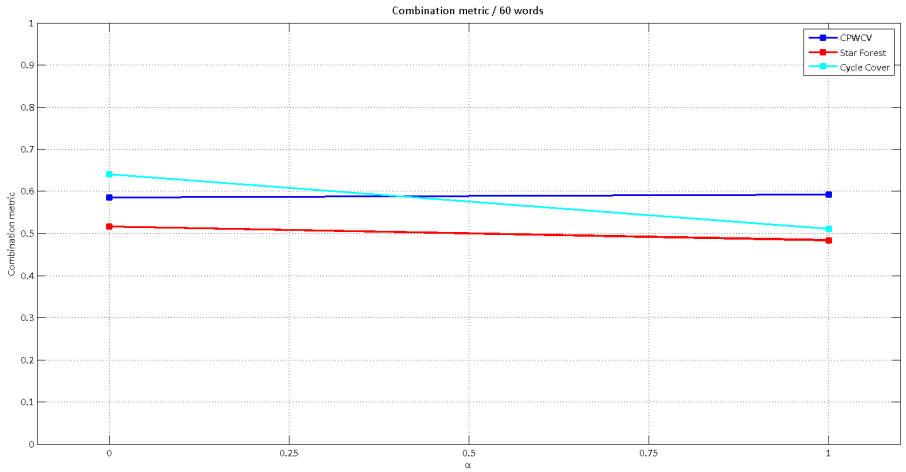


Figura 4.30: Combination metric: LexRank + Jaccard Similarity con 60 parole estratte.

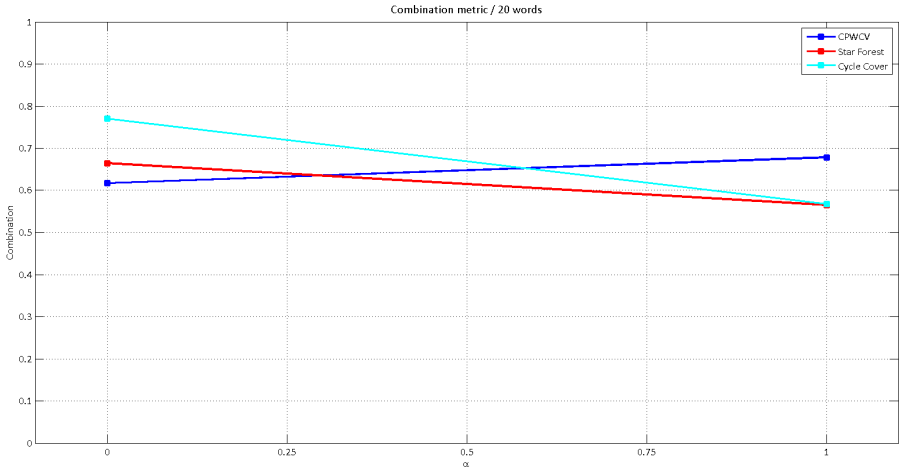


Figura 4.31: Combination metric: LexRank + Cosine Similarity con 20 parole estratte.

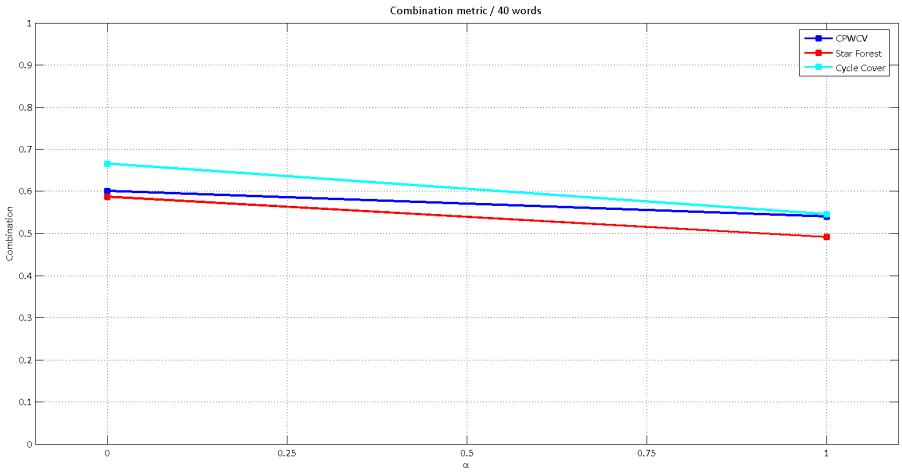


Figura 4.32: Combination metric: LexRank + Cosine Similarity con 40 parole estratte.

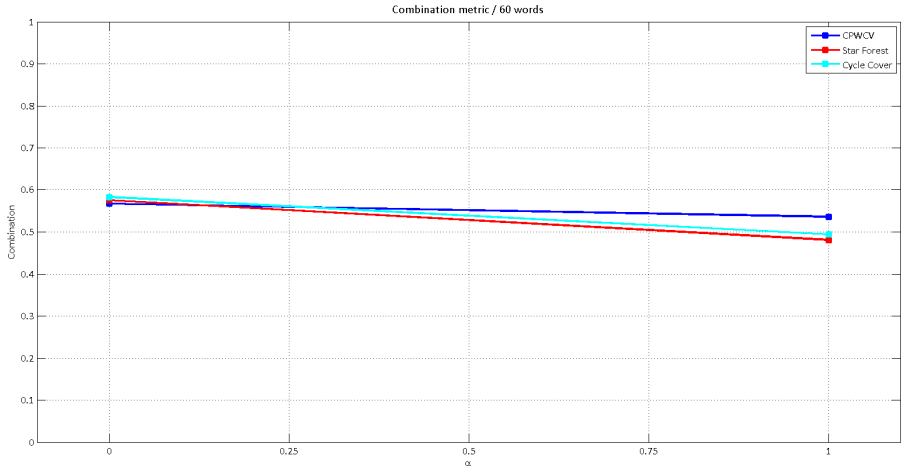


Figura 4.33: Combination metric: LexRank + Cosine Similarity con 60 parole estratte.

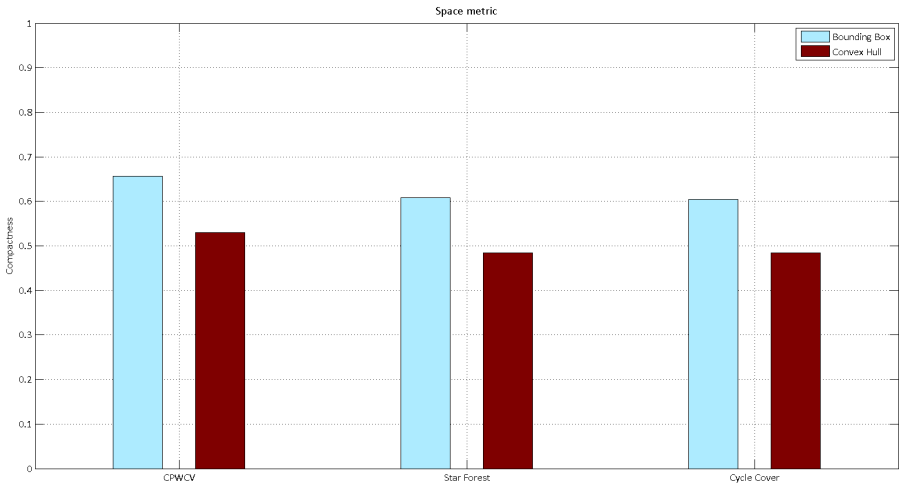


Figura 4.34: Space metric: Term Frequency + Jaccard Similarity con 20 parole estratte.

Space metric

Nei risultati ottenuti, l'algoritmo CPWCV risulta essere l'algoritmo che produce disegni meno compatti, anche se la differenza con Star Forest e Cycle Cover non è eccessiva. Ovviamente, si ottengono valori diversi a seconda dell'utilizzo del bounding box o del convex hull. Quest'ultimo, infatti, ha un'area più piccola del bounding box, abbassando quindi il valore $1 - \frac{\mu}{\varphi}$. In generale, comunque, si ottengono buoni risultati: i valori estremi possono infatti essere dannosi, poichè disegni troppo compatti comportano una difficoltà nella lettura di parole adiacenti, mentre disegni poco compatti possono risultare dispersivi.

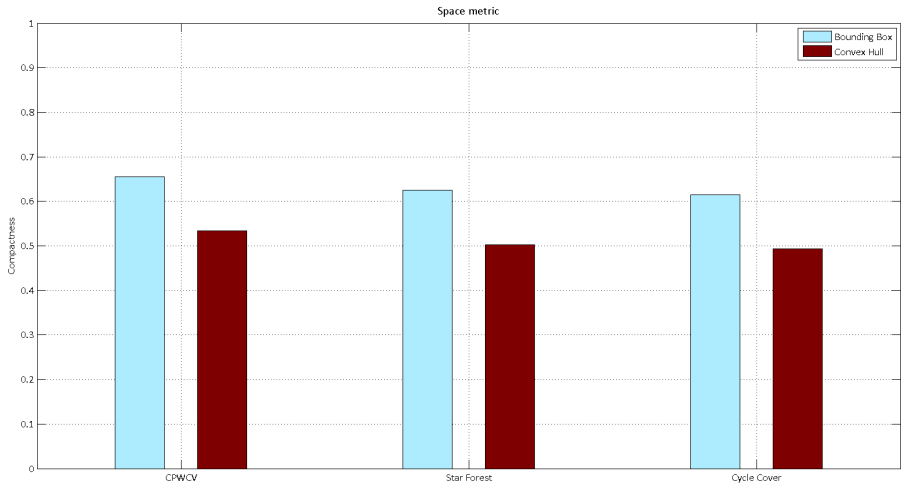


Figura 4.35: Space metric: Term Frequency + Jaccard Similarity con 40 parole estratte.

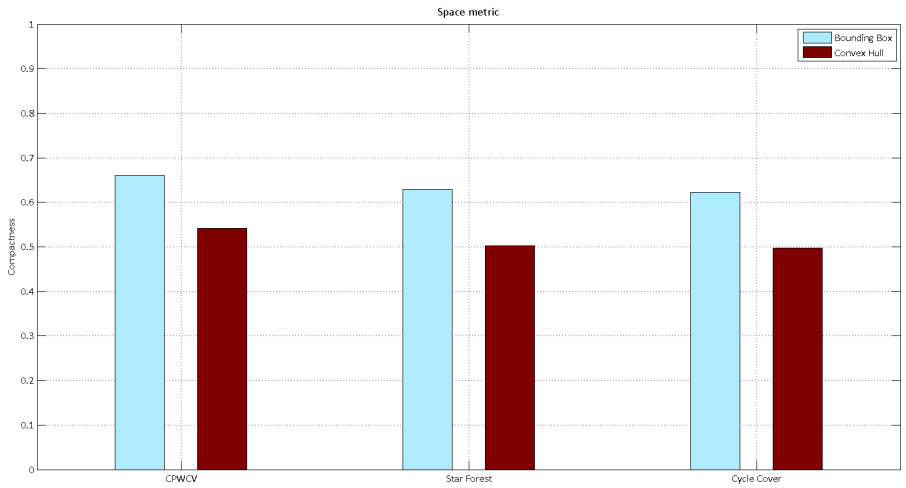


Figura 4.36: Space metric: Term Frequency + Jaccard Similarity con 60 parole estratte.

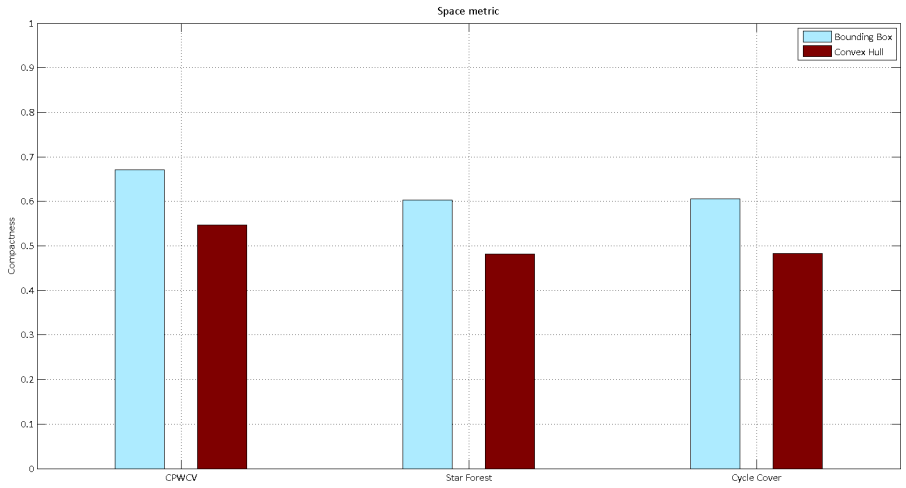


Figura 4.37: Space metric: Term Frequency + Cosine Similarity con 20 parole estratte.

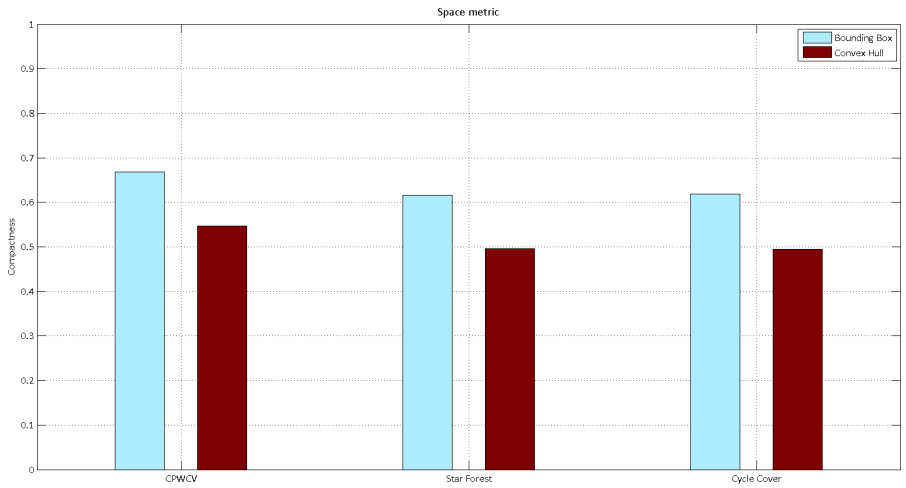


Figura 4.38: Space metric: Term Frequency + Cosine Similarity con 40 parole estratte.

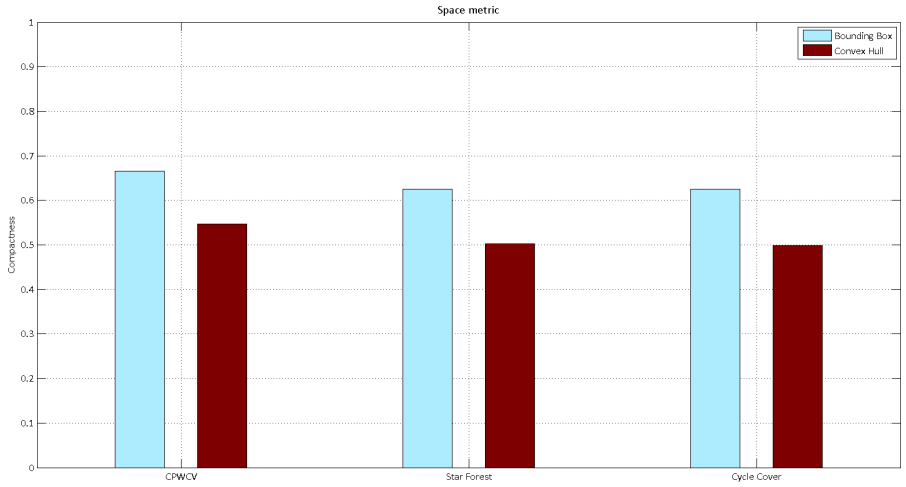


Figura 4.39: Space metric: Term Frequency + Cosine Similarity con 60 parole estratte.

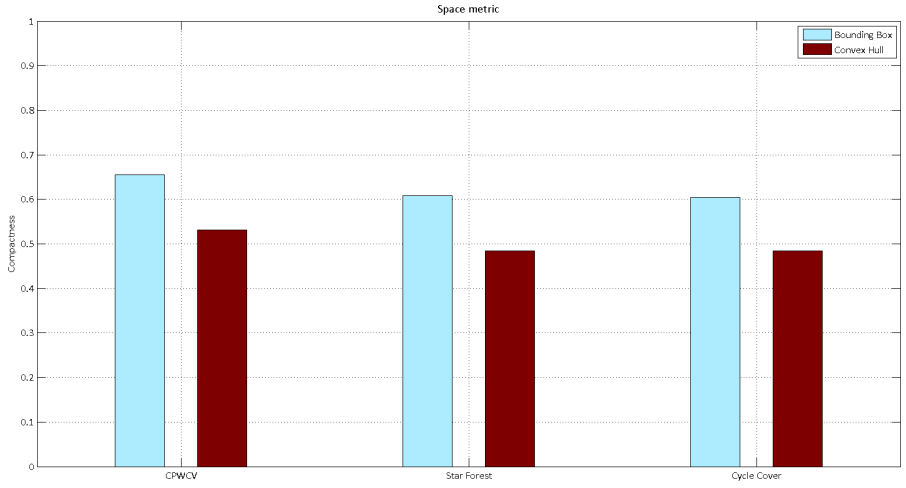


Figura 4.40: Space metric: Term Frequency + Extended Jaccard Similarity con 20 parole estratte.

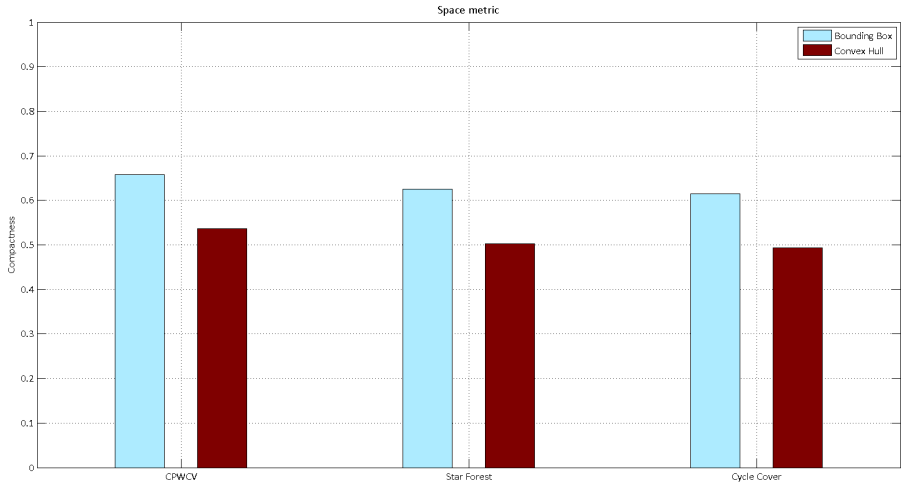


Figura 4.41: Space metric: Term Frequency + Extended Jaccard Similarity con 40 parole estratte.

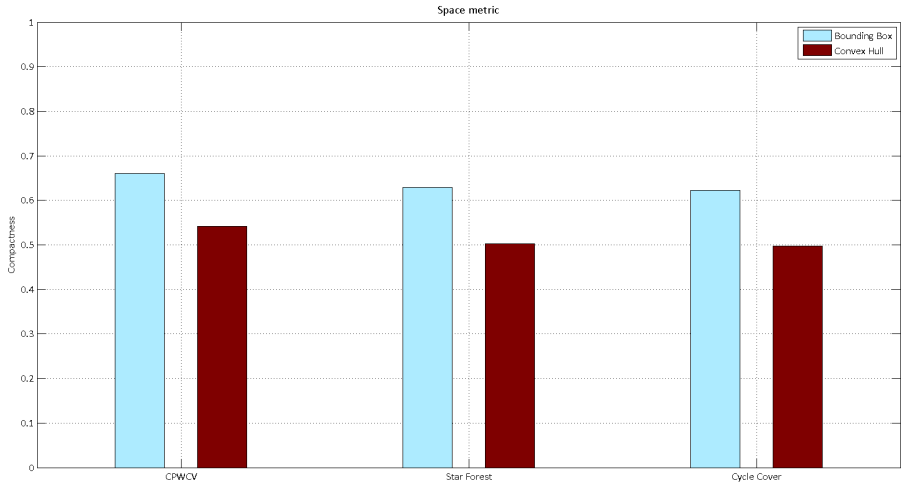


Figura 4.42: Space metric: Term Frequency + Extended Jaccard Similarity con 60 parole estratte.

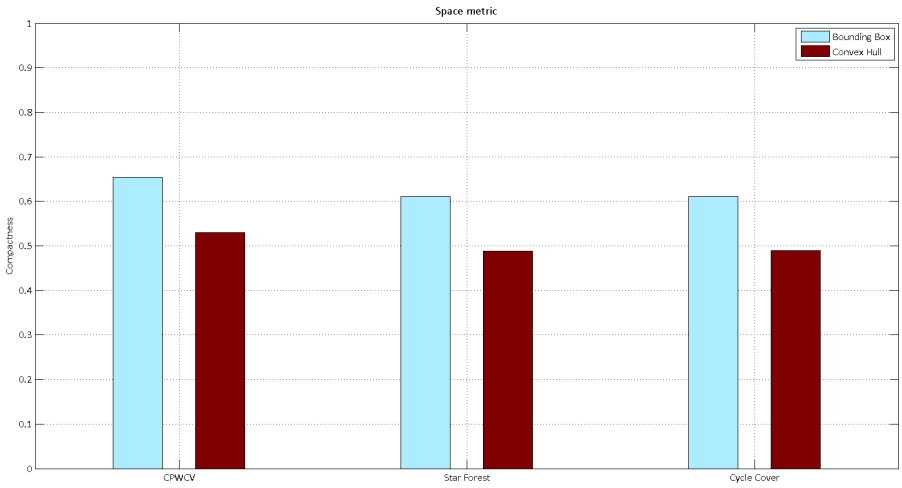


Figura 4.43: Space metric: TFIDF + Jaccard Similarity con 20 parole estratte.

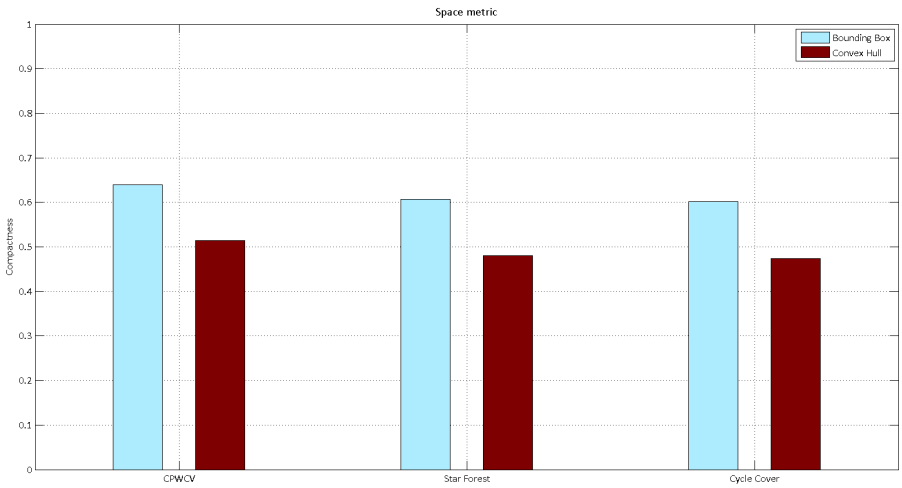


Figura 4.44: Space metric: TFIDF + Jaccard Similarity con 40 parole estratte.

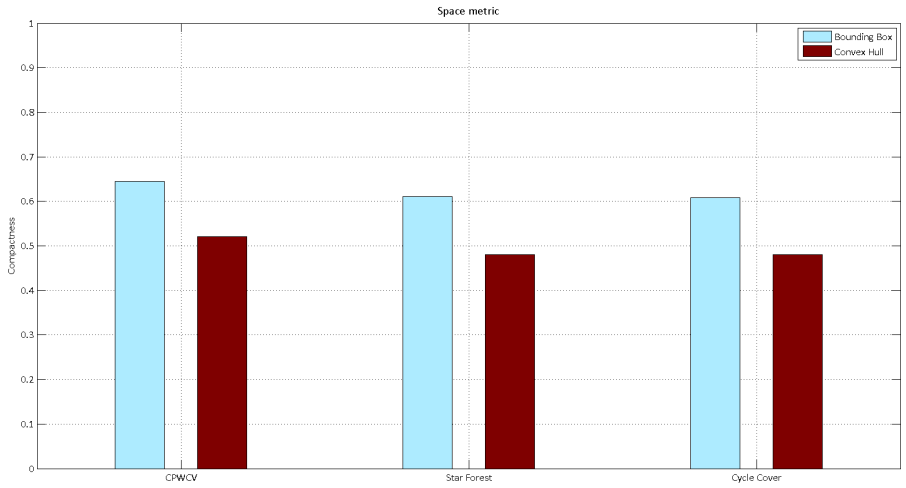


Figura 4.45: Space metric: TFIDF + Jaccard Similarity con 60 parole estratte.

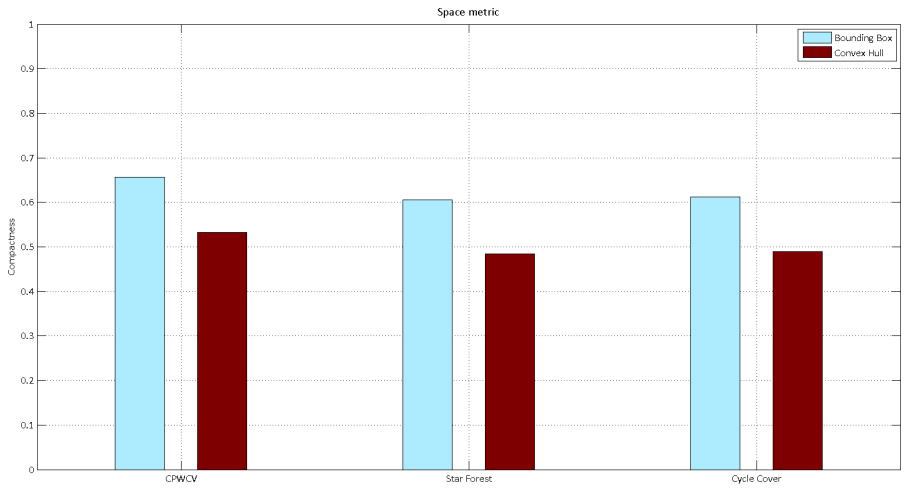


Figura 4.46: Space metric: TFIDF + Cosine Similarity con 20 parole estratte.

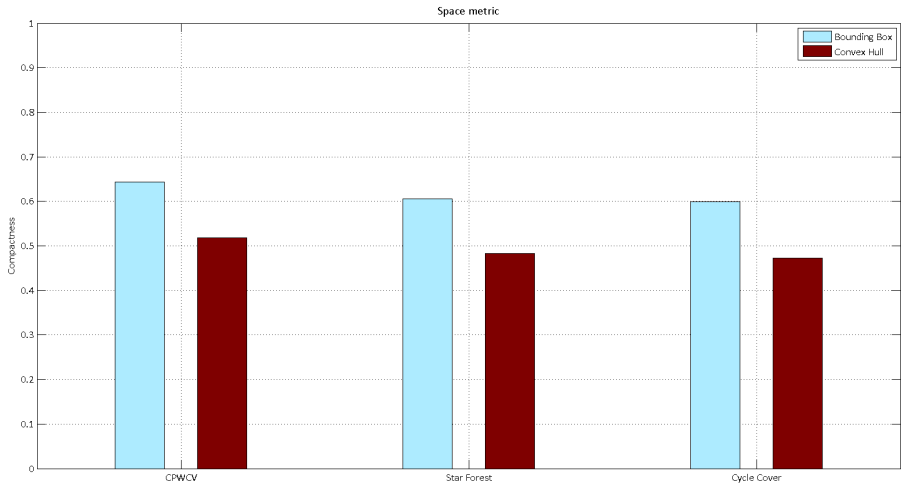


Figura 4.47: Space metric: TFIDF + Cosine Similarity con 40 parole estratte.

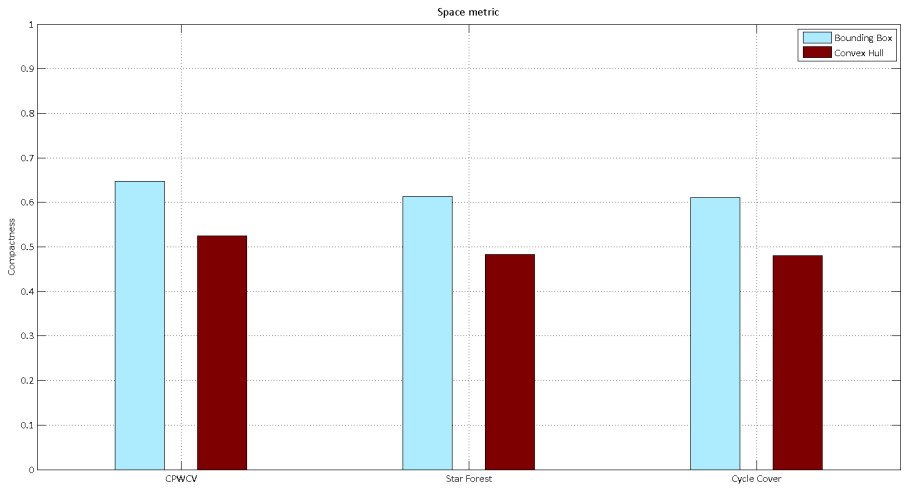


Figura 4.48: Space metric: TFIDF + Cosine Similarity con 60 parole estratte.

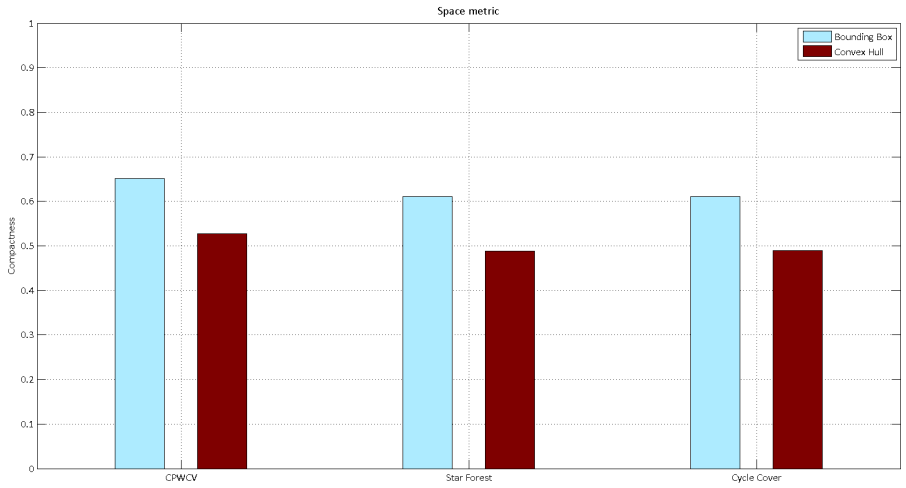


Figura 4.49: Space metric: TFIDF + Extended Jaccard Similarity con 20 parole estratte.

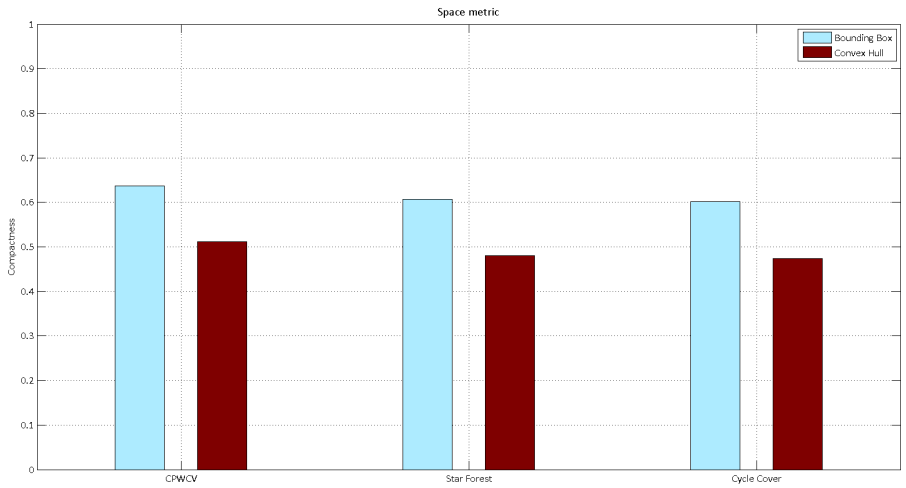


Figura 4.50: Space metric: TFIDF + Extended Jaccard Similarity con 40 parole estratte.

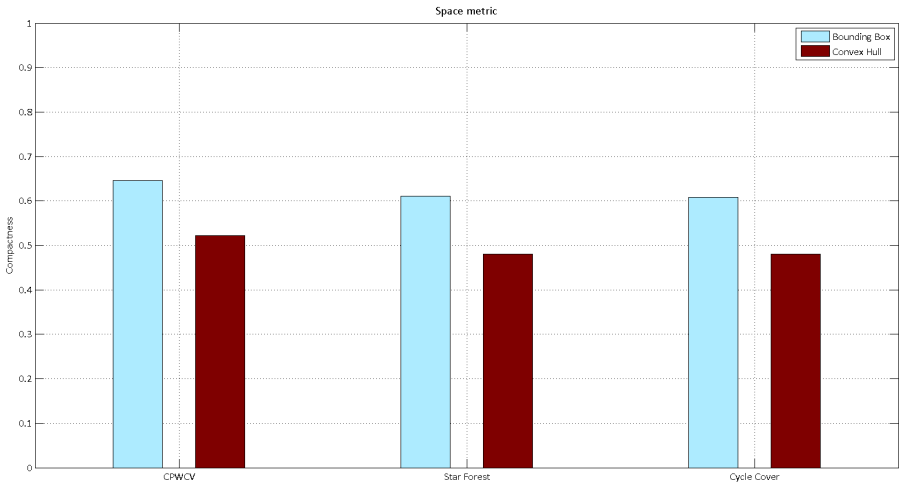


Figura 4.51: Space metric: TFIDF + Extended Jaccard Similarity con 60 parole estratte.

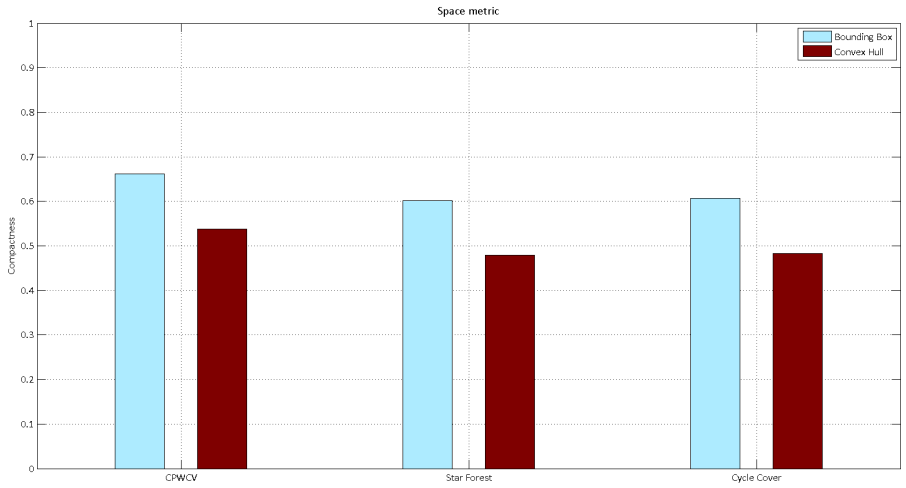


Figura 4.52: Space metric: LexRank + Jaccard Similarity con 20 parole estratte.

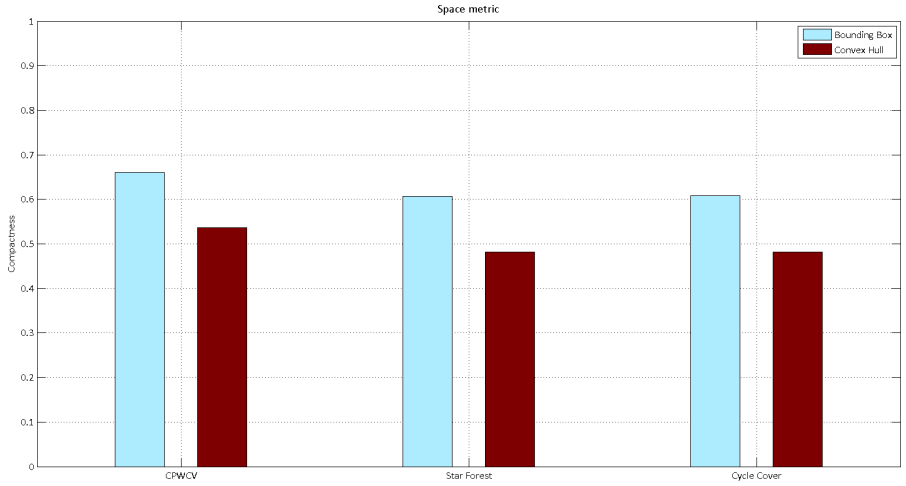


Figura 4.53: Space metric: LexRank + Jaccard Similarity con 40 parole estratte.

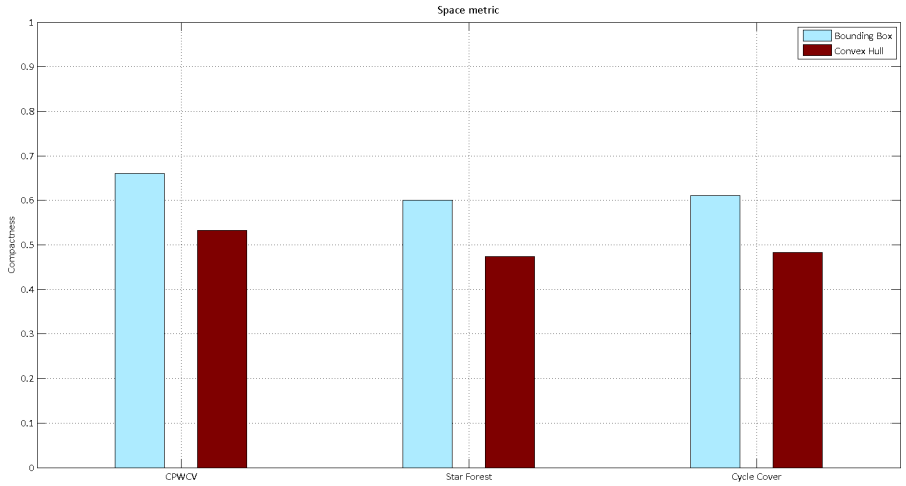


Figura 4.54: Space metric: LexRank + Jaccard Similarity con 60 parole estratte.

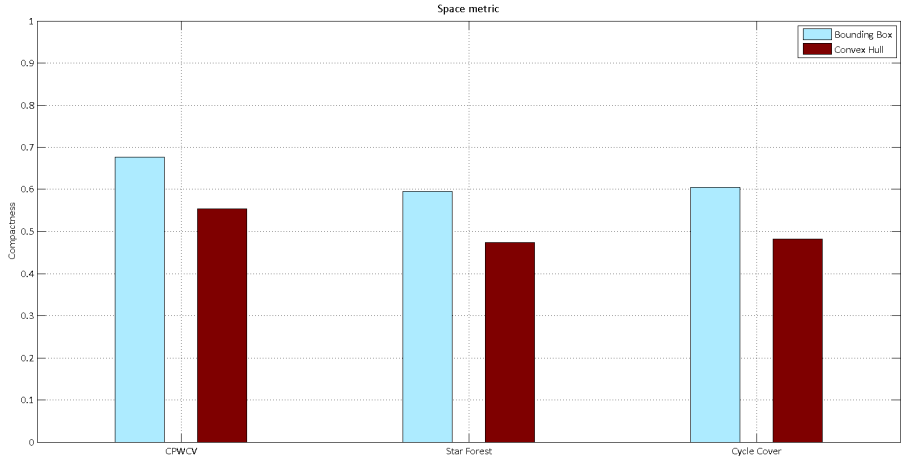


Figura 4.55: Space metric: LexRank + Cosine Similarity con 20 parole estratte.

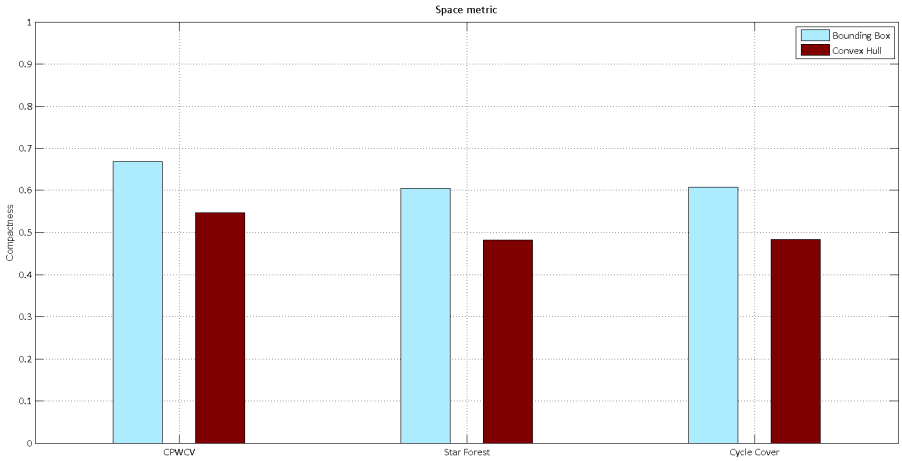


Figura 4.56: Space metric: LexRank + Cosine Similarity con 40 parole estratte.

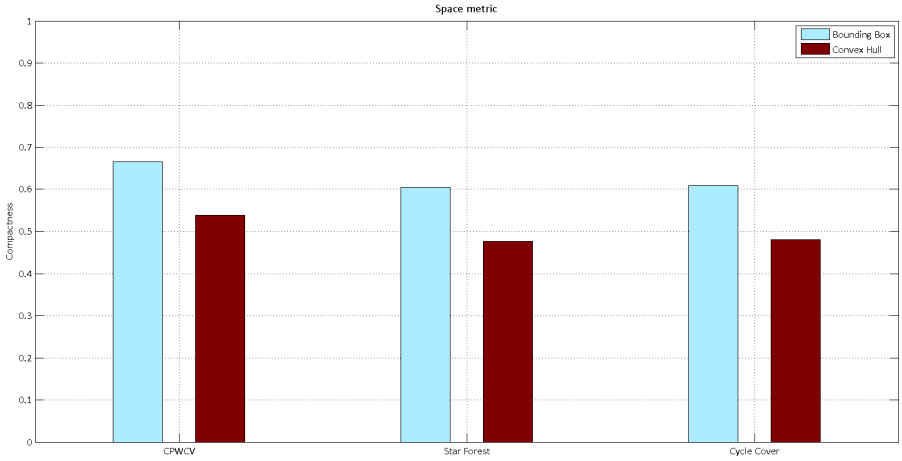


Figura 4.57: Space metric: LexRank + Cosine Similarity con 60 parole estratte.

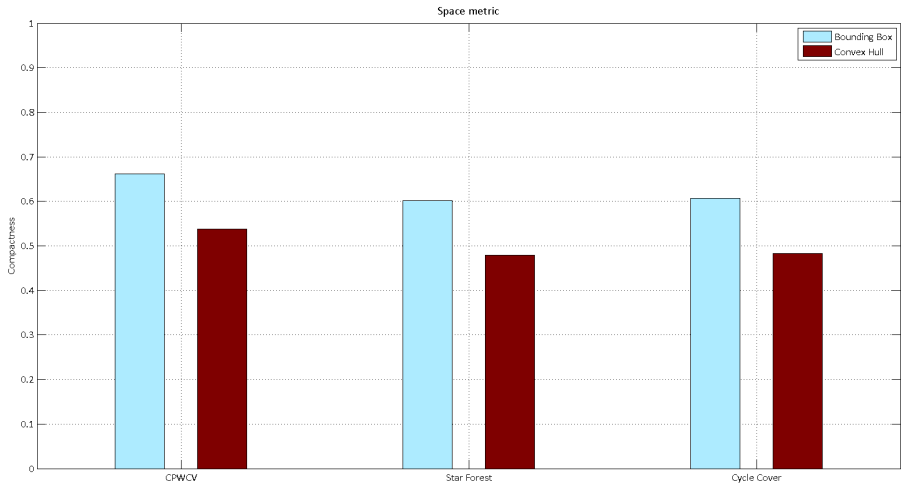


Figura 4.58: Space metric: LexRank + Extended Jaccard Similarity con 20 parole estratte.

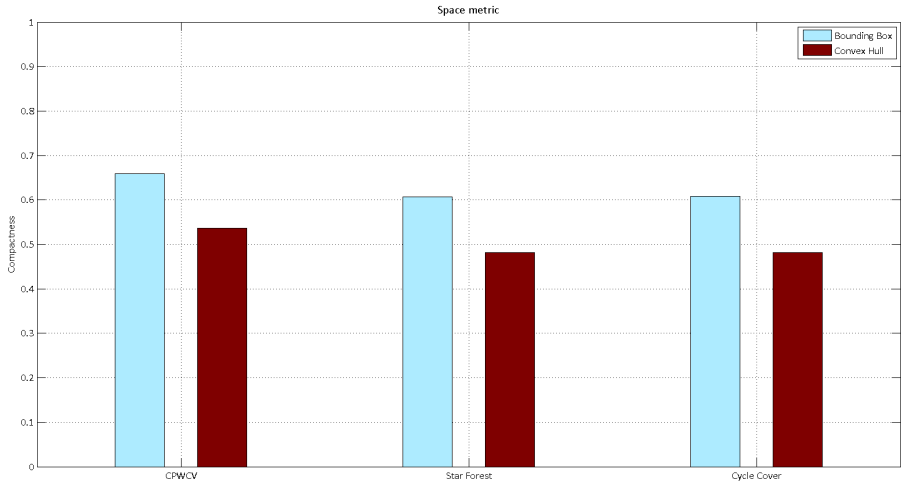


Figura 4.59: Space metric: LexRank + Extended Jaccard Similarity con 40 parole estratte.

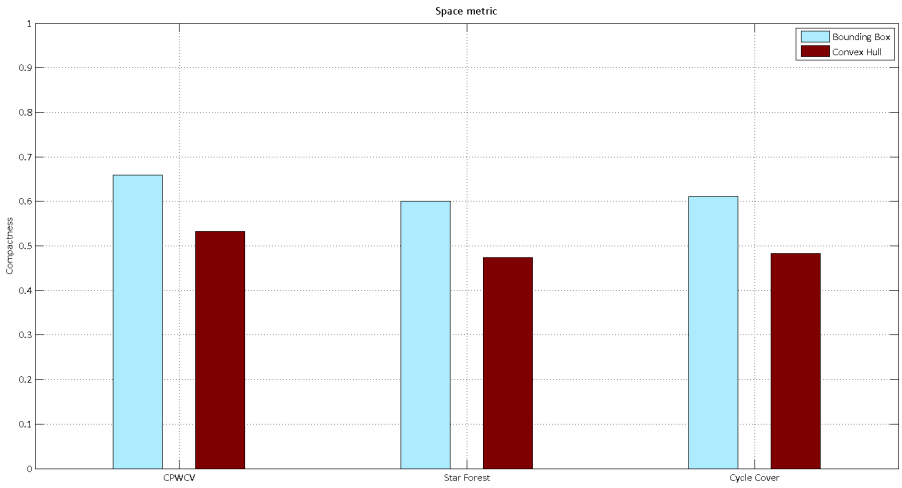


Figura 4.60: Space metric: LexRank + Extended Jaccard Similarity con 60 parole estratte.

Running time

Il tempo medio per l'elaborazione del testo e l'estrazione delle parole è espresso in tabella 4.1. Si nota che l'algoritmo che impiega di più è TF-IDF, seguito da LexRank e Term Frequency, piuttosto veloce. TF-IDF risulta il meno veloce poichè per ogni termine t, si calcola il parametro idf_t, estratto dal corpus Brown, però rispetto a Term Frequency estrae parole più rilevanti.

Il tempo impiegato è lo stesso indipendentemente dal numero di parole estratte: le parole vengono comunque tutte classificate e inserite su una lista ordinata per punteggio. Poi vengono selezionate le prime n parole, dove n è il numero di parole scelto da visualizzare.

Algoritmo	Tempo [sec]
Term Frequency	0.05
TF-IDF	0.41
LexRank	0.23

Tabella 4.1: Tempo d'esecuzione medio (espresso in secondi) di elaborazione del testo e classificazione delle parole.

Il tempo utilizzato per il calcolo delle similarità tra coppie di parole è indicato in tabella 4.2. Sono stati valutati i tempi di calcolo (espressi in μs) per 20,40 e 60 parole estratte. Gli algoritmi Cosine similarity ed Extended Jaccard similarity hanno quasi gli stessi tempi (in effetti i due algoritmi sono molto simili). La Jaccard similarity impiega un pò di più.

Parole estratte	Algoritmo		
	Jaccard	Cosine	Extended Jaccard
20	0.55	0.34	0.34
40	1.61	0.98	1.1
60	3.02	1.86	1.88

Tabella 4.2: Tempo d'esecuzione medio (espresso in μs) per il calcolo della similarità per 20,40 e 60 parole estratte.

Parole estratte	Algoritmo		
	CPWCV	Star Forest	Cycle Cover
20	$3.93\mu s$	$0.095s$	$6.43\mu s$
40	$0.016s$	$0.151s$	$0.024s$
60	$0.048s$	$0.27s$	$0.067s$

Tabella 4.3: Tempo d'esecuzione medio per la creazione della word cloud con 20,40 e 60 parole.

In tabella 4.3 sono riportati i tempi degli algoritmi di disegno. L'algoritmo più lento è Star Forest, mentre CPWCV e Cycle Cover sono paragonabili. In ogni caso, gli algoritmi di disegno sono piuttosto veloci e offrono buone prestazioni.

La procedura di morphing è stata testata per un numero elevato di frame tra una word cloud ed un'altra, ovvero 150. L'algoritmo risulta comunque piuttosto veloce (tabella 4.4). I valori sono quasi uguali per 20 e 40 parole estratte, mentre il tempo aumenta per 60 parole estratte.

Parole estratte	Tempo [μs]
20	0.55
40	0.57
60	1.86

Tabella 4.4: Tempo d'esecuzione medio (espresso in μs) dell'algoritmo di morphing delle parole.

Nella tabella 4.5, a pagina successiva, sono riportati i tempi d'esecuzione dell'algoritmo di clustering, che include anche la gestione delle variazioni tra cluster di layout successivi.

L'algoritmo di morphing che esegue l'aggiornamento dei colori frame per frame è leggermente più lento del morphing delle parole. Questo perchè nell'implementazione vengono create ed accedute strutture dati più complesse (vedi tabella 4.6).

La generazione dell'interfaccia grafica è, insieme all'elaborazione del testo e all'esecuzione dell'algoritmo di layout, la fase che consuma più tempo. Ovviamente, al crescere del numero di parole, anche il tempo impiegato aumenta (vedi tabella 4.7).

Parole estratte	Tempo [μs]
20	2.86
40	9.32
60	18.8

Tabella 4.5: Tempo d'esecuzione medio (espresso in μs) dell'algoritmo di clustering per 20,40 e 60 parole estratte.

Parole estratte	Tempo [μs]
20	3.41
40	7.8
60	13.16

Tabella 4.6: Tempo d'esecuzione medio (espresso in μs) dell'algoritmo di morphing dei colori delle parole.

Parole estratte	Tempo [s]
20	0.398
40	0.478
60	0.614

Tabella 4.7: Tempo medio (espresso in secondi) di creazione dell'interfaccia grafica per 20,40 e 60 parole.

Parole estratte	Caso peggiore	Caso peggiore
20	$2.65s$	$0.77s$
40	$3s$	$0.94s$
60	$3.66s$	$1.25s$

Tabella 4.8: Tempo d'esecuzione medio (espresso in secondi) per la creazione della word cloud con 20,40 e 60 parole.

Infine, per valutare il tempo totale necessario a creare la word cloud dinamica, il sistema è stato avviato, rispettivamente per 20,40 e 60 parole estratte, con le configurazioni degli algoritmi che sulla base delle tabelle precedenti hanno riportato le prestazioni peggiori e migliori. Ciò vale ovviamente solo per gli algoritmi di estrazione delle parole, calcolo delle similarità e disegno delle word cloud. Il numero di frame tra una word cloud e la successiva è pari a 150.

La configurazione peggiore è data dagli algoritmi TFIDF, Jaccard Similarity e Star Forest, mentre la configurazione migliore è data da Term Frequency, Cosine (o Extended Jaccard) Similarity e CPWCV (o Cycle Cover). I risultati ottenuti sono del tutto ragionevoli e sono riportati in tabella 4.8.

Capitolo 5

Conclusioni e sviluppi futuri

5.1 Risultati

Questa tesi di laurea presenta un nuovo approccio alla visualizzazione dinamica di una word cloud semantica. Sono stati valutati quantitativamente diversi algoritmi sulla base di varie metriche. Tali algoritmi, già esistenti in letteratura, sono stati arricchiti con un metodo force directed in modo da gestire al meglio, se possibile, la dinamicità della word cloud, tenendo conto della correlazione semantica tra le parole. L'algoritmo Cycle Cover sembra essere il migliore da questo punto di vista: la distanza geometrica tra le parole riflette la loro correlazione semantica, pur mantenendo abbastanza discretamente la mappa mentale dell'utente.

Il calcolo della similarità tra cluster di word cloud successive, ha consentito una gestione ottimale delle variazioni dei cluster delle parole.

Inoltre, l'applicazione delle procedure di morphing ha permesso di ottenere una visualizzazione dinamica e gradevole della word cloud.

Il tutto è corredato da un'interfaccia grafica di semplice utilizzo, che permette all'utente di interagire con la word cloud e di modificare a proprio piacimento il layout, portando avanti o indietro la visualizzazione, così come un qualsiasi media player.

5.2 Sviluppi futuri

In futuro, si può intervenire su diversi aspetti. Innanzitutto, si potrebbe testare tale lavoro da un punto di vista qualitativo, definendo nuove metriche che tengano conto dell'interfaccia grafica e affiancando magari uno user study, in modo da verificare l'efficacia e l'utilità del nostro lavoro sotto il profilo della quantità di informazione fornita all'utente e, possibilmente, di migliorare il sistema realizzato.

Per quanto riguarda gli aspetti algoritmici, una parte fondamentale è costituita dal preprocessing del testo: l'utilizzo di tecniche più sofisticate di text mining (e.g. *Part-of-Speech Tagging*, cioè l'assegnazione di categorie linguistiche alle parole), potrebbe meglio classificare le parole e catturare in modo più preciso le loro relazioni semantiche. Un'altra

modifica da introdurre, è quella di considerare, tra parole in comune di word cloud consecutive, parole aventi la stessa radice. In tal caso, però, è necessario intervenire anche a livello di interfaccia grafica, poichè con la procedura di morphing, la trasformazione di una parola in un'altra dovrebbe avvenire in modo graduale.

Anche l'interfaccia grafica è da migliorare. Ad esempio, potrebbe essere interessante sviluppare un'applicazione web, arricchendo il layout con statistiche riguardanti le parole e i cluster.

Bibliografia

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

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[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] B. Delaunay. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7:793–800, 1934.

[25] 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.

[26] 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.

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

[28] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[29] Wikipedia, Brown Corpus, consultato a Gennaio 2016, `http://it.wikipedia.org/wiki/brown_corpus`.

[30] JFreeChart. http://sourceforge.net/projects/jfreechart.

[31] TED, Technology Entertainment Design, `http://www.ted.com`.

[32] TED2SRT, `http://ted2srt.org`.