Università degli Studi di Milano - Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di laurea in Informatica

Graph-based Word Sense Disambiguation

Relatore: Prof. Messina Enza

Co-relatore: Dott. Fersini Elisabetta

Relazione della prova finale di:
Demetrio Carrara
Matricola 807894

Anno Accademico 2017-2018

Indice

Capitolo 1

Introduzione

1.1 Il problema della disambiguazione

L'uomo si è sempre espresso in modo ambiguo: molti termini sono riconducibili a significati diversi a seconda del contesto in cui vengono inseriti. Per un umano capire il significato delle parole in una frase è una cosa automatica, semplice e abbastanza precisa. Se questo procedimento di attribuzione di significati alle parole viene fatto inconsapevolmente dagli umani, non si può dire lo stesso delle macchine: esse devono capire come può essere la struttura della frase, quali parole sono più o meno importanti e fra quali significati deve essere scelto quello ritenuto corretto. Si assume che ad ogni senso di una parola corrisponda una definizione diversa nel dizionario di riferimento.

Si considerino le seguenti frasi:

*The american **tank** is full of soldiers*
*That **tank** is full of petrol*

La parola *tank*, scritta ugualmente in entrambe le frasi, assume significato diverso a seconda del contesto: in primo luogo indica un carro armato, poi una tanica o recipiente contenente della benzina.
L'insieme dei significati assumibili dalla parola *tank* è numeroso e associare un senso in base al contesto non è sempre semplice.

Con l'avvento di internet, si è venuta a creare una grossa mole di dati, non strutturati (come pagine web, documenti, articoli scientifici) che hanno bisogno di essere automaticamente catalogati, letti e/o sintetizzati. Le tecniche tradizionali basate sull'analisi lessicale e sintattica mostrano i propri limiti quando vengono applicate a grosse collezioni di dati. Si pensi anche solo al fatto di tradurre una parola da una

lingua ad un'altra: ad esempio il verbo italiano *fare* può essere tradotto in inglese come *do*, *make*, *act* o *produce*.

Un altro passaggio automatico che viene fatto inconsapevolmente dagli umani è quello di categorizzare ogni parola con la corretta parte del discorso (*part of speech*, POS) poichè è necessario capire se, ad esempio, *tank* è un verbo o un sostantivo, al fine di attribuirgli il senso corretto.

Etichettare una parola con il corretto POS può ridurre notevolmente il numero di sensi attribuibili e quindi facilitarne la disambiguazione. Il problema di strutturare la frase può essere separato ed eseguito in una fase di preprocessing: i POS tagger attuali riescono a taggare automaticamente con performance molto alte.

Il problema della disambiguazione è tutt'ora al centro di innumerevoli studi ed è uno dei problemi ritenuti più difficili da risolvere, proprio perchè su una singola frase anche gli umani possono attribuire significati diversi alle stesse parole.

1.2 Descrizione del problema di WSD

L'abilità di attribuire un senso ad una parola in base al suo utilizzo e al contesto viene definita *Word Sense Disambiguation*.

Dato un testo T formato da una sequenza di n parole $(w_1, w_2, ..., w_n)$ delle quali vogliamo conoscere il significato preciso, possiamo descrivere WSD come il problema di assegnare il senso appropriato $S(i)$ fra tutti i possibili sensi $S_D(w_i)$, presenti nel dizionario D, alle parole in T.

$$That\ tank_1\ is\ full_2\ of\ petrol_3$$

Si vuole fare in modo che il sistema attribuisca automaticamente il senso corretto a *tank*, *full* e *petrol*. Il sistema deve quindi scegliere fra le numerose definizioni presenti nel dizionario quella più adatta al contesto.
Dato che le parole assumono significati diversi in base alle altre che le circondano, si deve porre molta attenzione alla frase intera piuttosto che ragionare sui singoli significati: capire in che modo sono correlate fra di loro, che ruolo assumono e quanto i numerosi sensi siano compatibili l'uno con l'altro.
Si può rappresentare quindi come un problema di ottimizzazione: trovare la minor distanza semantica fra i sensi delle parole in oggetto tale per cui venga scelto un solo senso per parola da disambiguare.

Per ottenere le informazioni necessarie a scegliere i sensi corretti come le relazioni che intercorrono fra essi o gli esempi nei quali vengono utilizzati si fa largo uso delle fonti di conoscenza ritenute più autorevoli.

1.3 Fonti di conoscenza

Le fonti di conoscenza sono semplici testi, dizionari leggibili dagli algoritmi di machine learning, ontologie che contengono informazioni essenziali per affrontare il problema del WSD.
La principale fonte di conoscenza utilizzata dallo stato dell'arte è *WordNet*.

WordNet WordNet [1] è un database lessicale in inglese basato su principi psicolinguistici. Racchiude i concetti in termini di gruppi di sinonimi (synset). Possiamo vedere un synset come un gruppo di sensi che esprimono lo stesso significato. Ogni senso identifica univocamente un synset, dal quale si attingono informazioni importanti al fine di capirne il contesto:

- *Gloss*: definizione testuale del synset
- Relazioni semantiche e lessicali: collegano, rispettivamente, coppie di sensi e synset. Alcune relazioni lessicali possono essere:
  - Antonimia: S è antonimo di T se esprime il concetto opposto
  - Meronimia: S è meronimo di T se S è una parte di T
  - Similarità: un aggettivo S è simile a un aggettivo T (*beautiful* è simile a *pretty*)
- Esempi: ogni synset contiene delle frasi utili per capire in che contesto vengono usate le parole

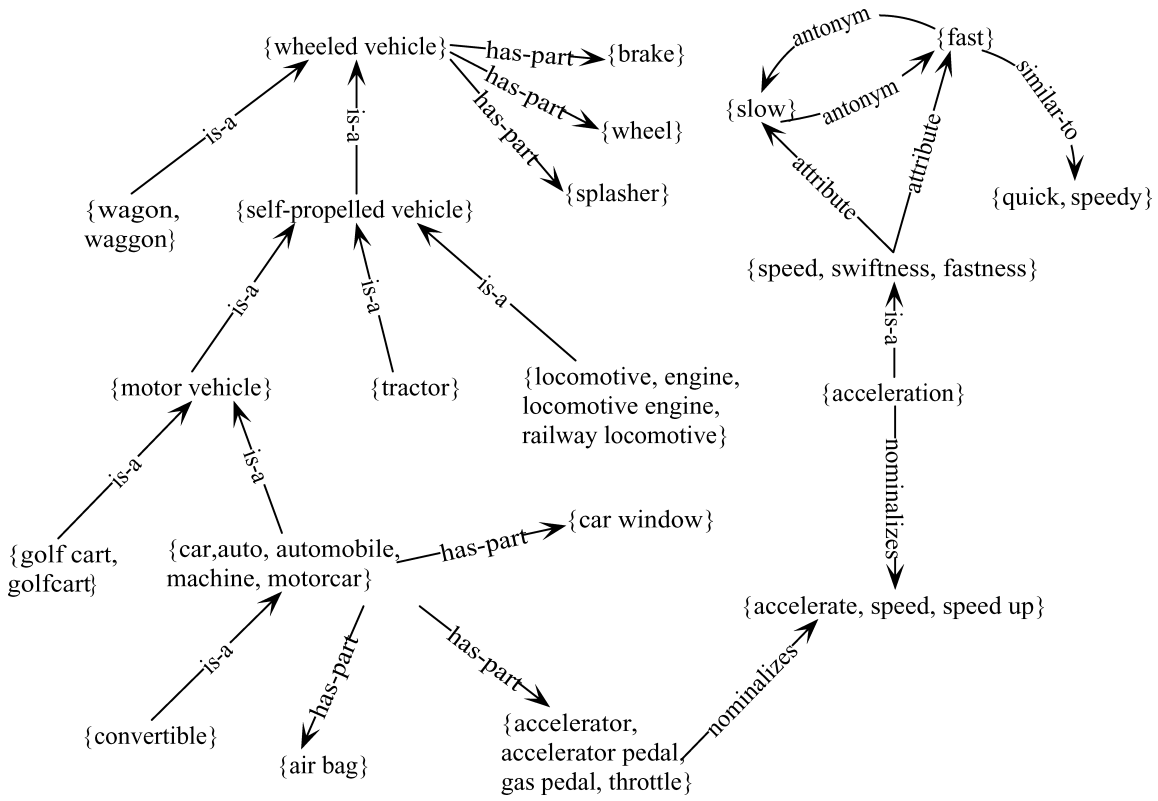


Figura 1.1: Rappresentazione di parte della struttura di WordNet

Per quanto riguarda la frase precedente, in un sistema ideale che utilizza WordNet come fonte di conoscenza, si avrebbe la seguente disambiguazione:

- tank: "a large (usually metallic) vessel for holding gases or liquids"
- full: "containing as much or as many as is possible or normal"
- petrol: "a volatile flammable mixture of hydrocarbons (hexane and heptane and octane etc.) derived from petroleum; used mainly as a fuel in internal-combustion engines"

SemCor SemCor [2] è un'altra fonte di conoscenza molto utilizzata dallo stato dell'arte. E' formato da una parte del Brown Corpus le cui parole sono state annotate manulamente con parte del discorso, lemma e senso esatto preso dal dizionario di WordNet. SemCor è molto utilizzato in ambito supervised perchè comprende 234.000 parole correttamente annotate e rappresenta la più grande risorsa per i classificatori. Sulla base di SemCor poi sono stati creati corpus multilingua, per espandere e aiutare a risolvere il problema della disambiguazione in altre lingue.

E' chiaro come, costruire una fonte di conoscenza molto dettagliata e precisa, richieda molto tempo e sia difficile, WordNet viene preso come riferimento perchè è mantenuto dalla Princeton University e comprende una vastità di parole e significati non indifferente.

Capitolo 2

Stato dell'arte

In questo capitolo si affrontano e valutano i numerosi metodi utilizzati nel campo della disambiguazione dallo stato dell'arte e come si possono combinare per ottenere performance migliori. Si presentano brevemente infine, alcuni degli algoritmi che ottengono attualmente i migliori risultati.

2.1 Disambiguazione supervisionata

Vengono definiti *supervisionati* (supervised) tutti quegli algoritmi che utilizzano delle tecniche di machine-learning per allenare un classificatore. Si differenziano dagli altri approcci perchè la loro esecuzione si può separare facilmente in due parti: fase di training e fase di classificazione.

Il training avviene leggendo vari datasets annotati manualmente con i sensi corretti in modo da "insegnare" ad un classificatore ad assegnare il senso esatto alle parole a seconda del contesto. La scelta dei datasets di training è molto importante per la buona riuscita di un classificatore: essi devono coprire un numero molto alto e molto vasto di istanze per far sì che il classificatore riesca a riconoscere non solo i sensi più comuni. In base alla lettura e valutazione delle istanze di training si generano delle regole con un certo score, utilizzate poi per prendere decisioni nella fase successiva. Si utilizzano strutture ausiliarie per decidere e classificare un'istanza:

- *Decision list*: insieme ordinato di regole trattate come una lista di condizioni binarie [3].
- *Decision tree*: modello predittivo che partiziona l'insieme di training con una struttura ad albero [4].

Il problema del WSD, in ambito supervised, viene spesso risolto combinando varie tecniche e strumenti, diversi classificatori possono essere utilizzati per miglio-

rare l'accuracy complessiva. Esistono numerose strategie di *ensemble* che sfruttano le nature degli algoritmi, combinandoli, per migliorare le performance su specifiche aree di interesse. Questi metodi di ensemble si applicano molto bene al problema della disambiguazione perchè si possono studiare frasi dal punto di vista sintattico, semantico, lessicale e grammaticale. E' più semplice concentarsi e produrre un classificatore molto buono in una singola area applicativa e poi combinarlo con altri che lavorano bene su aree diverse per massimizzare l'accuracy piuttosto che valutare la frase da tutti i punti di vista possibili.
Gli algoritmi più recenti in ambito supervised sono [5, 6] in collaborazione con i maggiori esponenti dell'area WSD (Navigli, Raganato e Lei).

2.2 Disambiguazione non supervisionata

A differenza dei metodi supervisionati, non utilizzano tecniche di machine-learning, riuscendo a superare uno dei problemi principali: la necessità di avere larga disponibilità di datasets etichettati manualmente dai quali apprendere per generare regole e classificare le istanza di input.

I metodi non supervisionati provano a sfruttare tutte le informazioni che si possono ricavare dalle frasi in input senza avere una preconoscenza: una buona idea è che parole vicine fra loro nella frase possano essere utilizzate per capirne il contesto generale e che un senso di una parola abbia vicino parole simili. Possono dedurre il significato delle parole clusterizzandone le occorrenze e classificando le nuove occorrenze nei cluster dedotti. Il grosso svantaggio dei sistemi totalmente non supervisionati è che non sfruttano i dizionari, non avendo così un inventario condiviso di riferimento di sensi.

Mentre WSD è tipicamente identificato come "etichettare le parole con il senso corretto", i sistemi che fanno disambiguazione totalmente non supervisionata eseguono la discriminazione dei sensi, raggruppano cioè le occorrenze della parole in cluster, determinando se due occorrenze rappresentano lo stesso senso o meno, il quale può non avere un vero e proprio corrispondente nel dizionario.
Gli algoritmi più recenti in ambito unsupervised sono [7, 8].

2.3 Disambiguazione knowledge-based

L'obiettivo della disambiguazione knowledge-based è quello di sfruttare al massimo le informazioni ottenibili dalle fonti di conoscenza per dedurre il senso delle parole nel contesto.

Un esempio è quello di scegliere i sensi corretti di parole diverse in base a quanto siano simili le loro definizioni, i loro esempi. Le tecniche che si vanno ad analizzare sono: sovrapposizione delle definizioni dei sensi e approcci strutturali come misure di similarità e metodi graph-based.

Sovrapposizione delle definizioni dei sensi Un approccio semplice si basa sul calcolo di quante parole abbiano in comune le definizioni di due o più sensi. Prende il nome di algoritmo di Lesk [9] e viene così definito: date due parole da disambiguare, i sensi che, nella definizione, hanno il maggior numero di parole in comune sono ritenuti corretti. Formalmente, date due parole w_1 e w_2, si calcola uno score per ogni coppia di sensi $s_1 \in Senses(w1)$ e $s_2 \in Senses(w2)$:

$$score(s_1, s_2) = |gloss(s_1) \cap gloss(s_2)|$$

dove la funzione $gloss(s_i)$ restituisce l'insieme di parole della definizione testuale del senso s_i. Ora i due sensi che hanno il maggiore score sono ritenuti esatti.
L'algoritmo è molto sensibile all'esattezza delle parole nelle definizioni quindi l'assenza di alcune può cambiare totalmente il risultato.

Misure di similarità Sfruttando le informazioni ricavate dalle fonti di conoscenza si vuole di misurare quanto siano simili semanticamente due sensi. Data una misura di similarità semantica m definita come:

$$m : Senses_D \times Senses_D \rightarrow [0, 1]$$

dove $Senses_D$ è l'insieme di tutti i sensi del dizionario D di riferimento, possiamo definire un sistema di disambiguazione che sfrutta la misura di similarità m. Disambiguiamo una parola w_i in un testo $T = (w_1, w_2, ..., w_n)$ scegliendo il senso s che massimizza la seguente formula:

$$s = \underset{s_1 \in Senses_D(w_i)}{\operatorname{argmax}} \sum_{w_j \in T : w_i \neq w_j} \max_{s_2 \in Senses_D(w_j)} m(s_1, s_2)$$

Per ogni parola viene quindi scelto il senso con la somma maggiore.

Approcci graph-based Rappresentando la frase come un grafo si vuole sfruttare la struttura conosciuta e già studiata per applicare algoritmi esistenti che possono fornire informazioni in più.

Si possono vedere le parole (o gruppi di parole) come nodi del grafo e le relazioni (semantiche, lessicali) come archi. Più che al contenuto dei singoli sensi nel dizionario

(significati, esempi, synset) si guarda la forma: come sono collegati fra loro i sensi, che relazione hanno o se esiste un percorso che li unisce, se hanno molte parole in comune.

Gli approcci graph-based considerano le *lexical chains* [10], sequenze di parole $w_1, w_2, ..., w_n$ correlate in un testo, tali per cui tra w_i e w_{i+1} esiste una relazione lessico-semantica. Le catene lessicali sono utili per determinare il contesto della frase e il significato generale. Ad esempio, nella frase *I like to listen to the radio and sing*, la catena *listen* $\rightarrow$ *radio* $\rightarrow$ *sing* aiuta a capire il senso generale e il contesto.

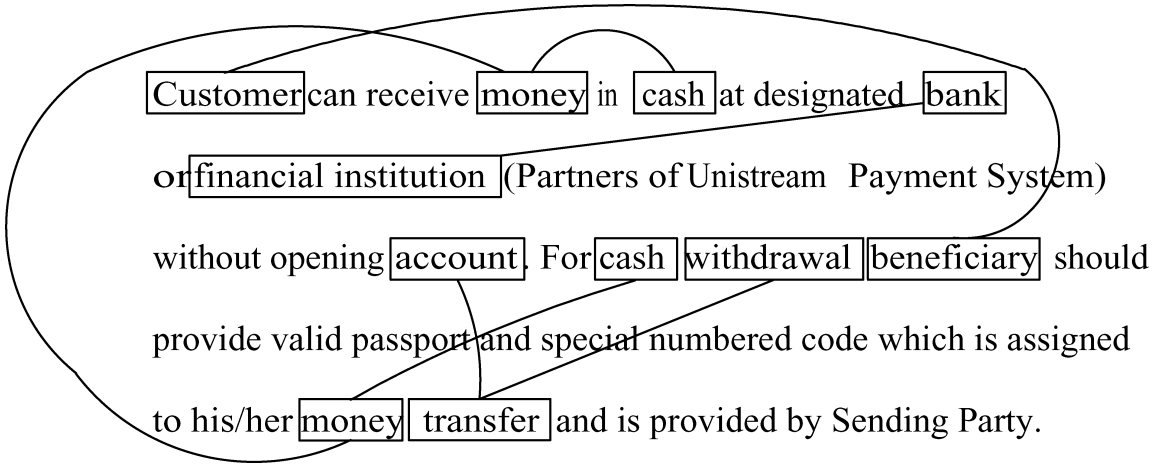


Figura 2.1: Esempio di catene lessicali

Le catene lessicali sono un ottima alternativa alle misure di similarità perchè guardano il contesto globale della frase e provano a dedurne un concetto invece di essere una misura strettamente locale fra due sensi.
Gli algoritmi graph-based più recenti dello stato dell'arte sono [11, 12].

Tra gli algoritmi dello stato dell'arte usati come baseline di riferimento troviamo:

Lesk Lesk è un algoritmo che si basa sulla sovrapposizione delle definizioni dei sensi. E' stato descritto precedentemente e viene scelto perchè disambigua in modo knowledge-based.

Babelfy Babelfy è attualmente l'algoritmo non supervisionato, graph-based che ha le migliori performance nell'ambito WSD. Sfrutta dei cammini casuali [13] per determinare le connessioni fra gli insiemi dei sensi. Utilizza BabelNet come fonte di

conoscenza e ottiene i migliori risultati quando considera l'intero documento, anzichè la singola frase, come contesto.

Capitolo 3

Graph-based Word Sense Disambiguation

In questo capitolo si introduce il metodo utilizzato, disambiguando i sensi delle parole in una frase, tramite la creazione di due grafi: uno utilizzato per il calcolo delle centralità ed uno utilizzato per la disambiguazione mediante TSP.

Il metodo si può definire:

- *unsupervised* perchè non utilizza nessuna tecnica di training e nessun algoritmo di machine-learning
- *graph-based* perchè costruisce una struttura a grafo a partire da una frase e ne utilizza gli algoritmi conosciuti per estrarre informazioni riguardanti il contesto
- *token-based* [14] perchè associa uno specifico significato ad ogni occorrenza di una parola in base al contesto in cui appare, a differenza degli approcci *type-based* che si basano sull'assunzione che una stessa parola assuma lo stesso significato all'interno dell'intero testo piuttosto che nel singolo contesto. Questi ultimi tendono a dedurre un senso (chiamato predominante) per una parola e ad attribuirlo ad ogni occorrenza all'interno dell'intero testo. Si nota come gli approcci *token-based* sono sempre adattabili ad assegnare poi il senso più comune ad ogni occorrenza della parola nel testo.

L'idea alla base dell'approccio è quella di sfruttare la struttura a grafo, ricavata dai sensi delle parole nella frase e dalle loro relazioni, eseguendo algoritmi di centralità che mettano in evidenza i nodi principali dai quali si può capire il contesto. Sulla base dei risultati degli algoritmi poi si vogliono applicare altre soluzioni generiche sui grafi, riconducibili al nostro problema, come l'esecuzione di TSP.

Per poter pesare in modo sensato i collegamenti fra i sensi di diverse parole, si è deciso di valutare l'importanza di un nodo all'interno del grafo in base a diversi

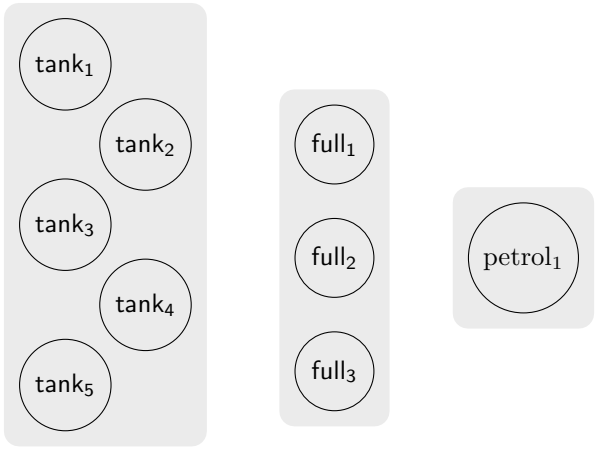
algoritmi di centralità. I nodi centrali saranno ritenuti quindi importanti e per poter eseguire TSP è necessario che la loro centralità venga distribuita come peso degli archi che li connette agli altri nodi.

Si vuole quindi costruire un grafo, che rappresenti la frase intera, avente come nodi, tutti i sensi attribuibili alle parole nella frase, e come archi i possibili collegamenti fra questi sensi. Sulla base della struttura del grafo poi si vogliono applicare diversi algoritmi di centralità in modo che i sensi ritenuti corretti nel contesto risultino "centrali".

3.1 Approcci proposti

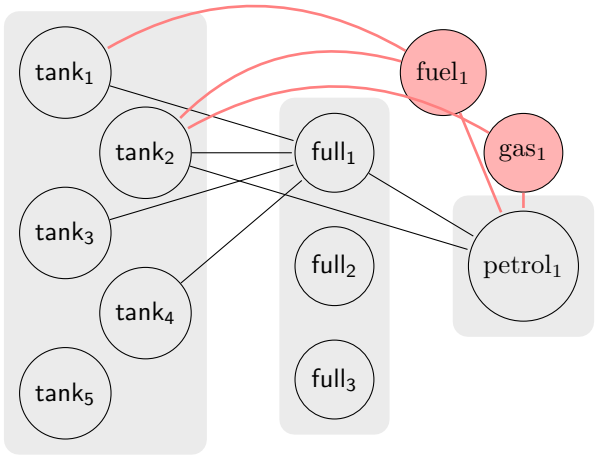
Il grafo per il calcolo delle centralità viene costruito a partire dal grafo di WordNet. Si vuole fare in modo che i sensi ritenuti corretti nel contesto siano quelli con maggiore centralità. Il grafo $G = (V, E)$ che rappresenta la frase viene così costruito:

- Per ogni parola w_i contenuta nella frase e utile alla disambiguazione (congiunzioni, articoli e punteggiatura non vengono presi in considerazione), vengono ricercati i vari sensi s_{ij} attribuibili nel dizionario di WordNet. Ogni senso viene poi rappresentato nel grafo di partenza come nodo.
  Riprendendo la frase *That tank₁ is full₂ of petrol₃*, la si rappresenta in questo modo:



- Per capire in che modo i sensi delle diverse parole possano essere collegati tra loro, si esegue una ricerca DFS sul grafo di WordNet: per ogni senso s_{ij} si cercano tutti i cammini (con al massimo n nodi intermedi) che lo colleghino ad un altro senso s_{kl} rappresentante un'altra parola (non ha senso cercare

connessioni fra sensi che rappresentano uno stessa parola). Per ogni percorso trovato si aggiungono al grafo i nodi e gli archi intermedi (tutti equipesati).



Il grafo è composto quindi da cluster di nodi: ogni cluster rappresenta una parola e ogni nodo un senso. L'esperimento prende in considerazione tutti i tipi di relazioni, perchè se volessimo scegliere solo le relazioni più significative (come sinonimi e derivati) non si avrebbero abbastanza nodi nel grafo per calcolare in modo efficace le centralità. Sulla base di queste relazioni viene costruito il grafo sul quale vengono calcolate le centralità.

In questo modo si avrà che sensi fortemente connessi ad altri sensi saranno molto importanti per determinare il contesto e saranno quindi candidati ad essere i sensi corretti.

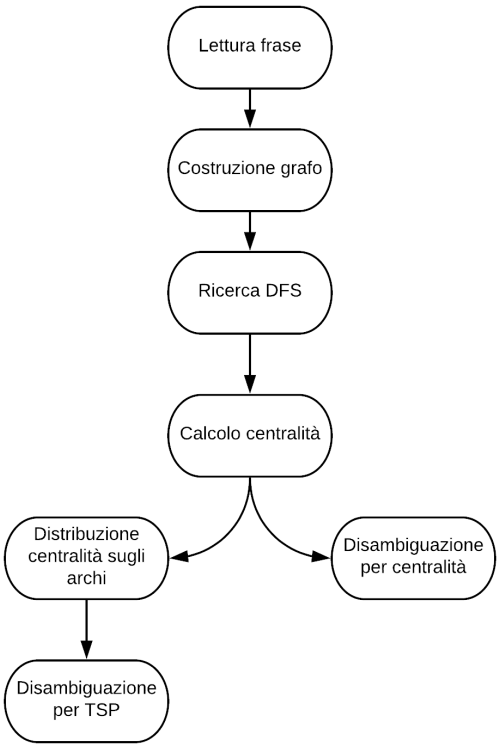


Figura 3.1: Pipeline dell'algoritmo per WSD

3.1.1 Grafo delle centralità

Ora che il grafo ha preso forma, si è deciso di applicare diversi algoritmi di centralità e confrontare i risultati ottenuti in modo da valutare un ampia gamma di configurazioni per risolvere il problema presentato.

Introduciamo il concetto di distanza $d(a, b)$ tra un nodo a e un nodo b come il cammino minimo (numero di archi) che congiunge i due nodi, se esiste ($a \rightsquigarrow b$):

$$d(a,b) = \begin{cases} \text{lunghezza cammino minimo} & \text{se } a \rightsquigarrow b \\ \infty & \text{altrimenti} \end{cases}$$

Definiamo il concetto di misura locale come il grado di importanza di un nodo v all'interno del grafo G. Formalmente una misura locale l è definita come una funzione che associa ad ogni nodo v un numero compreso tra 0 e 1:

$$l : v \to [0, 1]$$

Un valore vicino ad 1 indica che un vertice è relativamente importante, vicino a 0 indica invece che è un vertice periferico.

Le misure locali utilizzate sono quelle dei più famosi algoritmi di centralità: *closeness*, *Page Rank*, *eigenvector* e *in-degree*.

Closeness L'algoritmo di *closeness centrality* [15] si basa sul principio di "vicinanza" di un nodo v a tutti gli altri. Si utilizzano tutte le distanze da v agli altri nodi della rete per ottenere una misura di quanto sia importante quel nodo rispetto agli altri. Si calcola come:

$$Closeness(v) = \frac{\sum_{u \in V : u \neq v} \frac{1}{d(u,v)}}{|V| - 1}$$

Page Rank L'algoritmo di *Page Rank* [16] determina la centralità di un nodo nella rete in base a quanti nodi importanti è direttamente collegato. L'algoritmo viene calcolato iterativamente utilizzando ogni volta i valori dell'iterazione precedente fino a convergenza (o fino a quando il cambiamento è irrilevante) con $\alpha = 0.85$ valore di damping factor [17]:

$$PageRank(v) = \frac{(1 - \alpha)}{|V|} + \alpha \cdot \sum_{(u,v) \in E} \frac{PageRank(u)}{\text{outdegree(u)}}$$

La centralità iniziale è $\frac{1}{|V|}$ per ogni nodo.

Eigenvector L'algoritmo di *eigenvector centrality* [15] misura l'influenza di un nodo v rispetto alla rete. Come descritto in [18] la centralità di v è proporzionale alla somma delle centralità dei vertici ai quali è connesso. A differenza di PageRank, la centralità iniziale di v è random, creando così maggiore granularità.

In Degree L'algoritmo di *in-degree centrality* [15] misura l'importanza di un nodo in base al numero di archi in entrata (in-degree). Seppur semplice, può essere molto efficace su grafi non pesati e soprattutto nel problema presentato, perchè creando omogeneità l'algoritmo poi, a parità di centralità, sceglierà il senso più comune. Il numero di archi in entrata viene poi normalizzato, per avere un valore compreso tra 0 e 1:

$$InDegree(v) = \frac{|\{(u,v) \in E : u \in V\}|}{|V-1|}$$

Si vuole quindi associare ad ogni parola w_i un senso s_{ij}. Una delle possibili soluzioni è quella di assegnare un rank ad ogni senso e scegliere il senso con rank maggiore.

Come viene attribuito il rank? Per ogni parola w_i si elencano in ordine di maggiore centralità i sensi s_{ij}. Si sceglie il primo senso, ovvero il senso rappresentato dal nodo più "centrale". Per sfruttare il *bias* del senso più comune, a parità di centralità si sceglie il primo senso preso da WordNet (corrispondente al senso più comune).

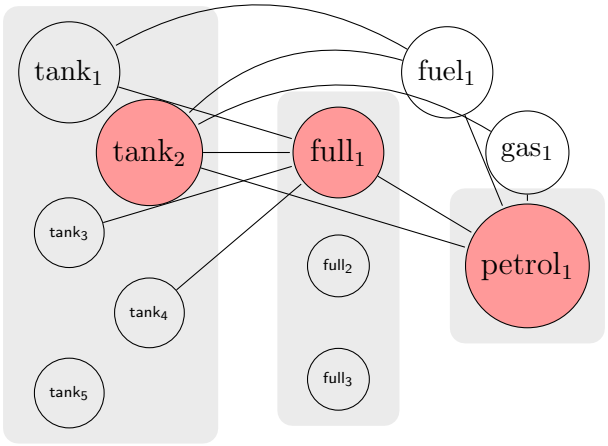


Figura 3.2: Grafo risultante dopo il calcolo delle centralità

3.1.2 Grafo di disambiguazione mediante Traveling Salesman Problem

WSD come Traveling Salesman Problem [19] Il problema del commesso viaggiatore (o TSP) è un problema molto conosciuto fra i problemi NP-Hard, dei quali non si conosce una soluzione che abbia al massimo tempo polinomiale. Si tratta di un problema in cui, un ipotetico commesso deve visitare n città e tornare alla città di partenza nel minore tempo possibile.

Possiamo applicare il problema e le euristiche utilizzate per risolvere TSP anche al problema del WSD in quanto è possibile vedere le n parole della frase come delle città. Il collegamento fra le parole è rappresentato da un numero che può indicare una misura di similarità, o una distanza in base a qualche altro criterio. Allargando

di più il problema, ogni parola è un insieme di sensi, si vuole trovare uno e un solo senso per ogni parola.

A differenza del grafo delle centralità, questo grafo non deve avere i nodi ausiliari creati dalla ricerca DFS. Il problema è rappresentabile come *Equality Generalized Traveling Salesman Problem* [20], una generalizzazione di TSP basata sui grafi non diretti e clusterizzati, in cui per ogni cluster è necessario che il solver visiti solamente un nodo (il senso corretto).

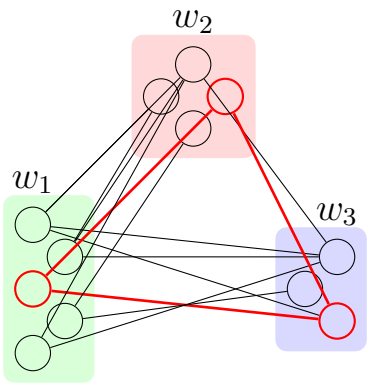


Figura 3.3: Rappresentazione Word Sense Disambiguation come Traveling Salesman Problem

Formalmente il problema *Equality Generalized Traveling Salesman Problem* è così definito da [21]:

Dato un grafo $G = (V, E)$ clusterizzato con n partizioni, dove $P = \{P_1, P_2, ..., P_n\}$, $\bigcap_{i=1}^{n} P_i = \varnothing$ e $\bigcup_{i=1}^{n} P_i = V$, l'obiettivo è quello di trovare il cammino hamiltoniano con minimo costo di k vertici, visitando ogni partizione una sola volta e solo un vertice per partizione.

Si definiscono le variabili per rappresentare il problema di ottimizzazione:

- x_e: variabile booleana uguale a 1 se l'arco $e \in E$ è stato scelto, 0 altrimenti.
- y_v: variabile booleana uguale a 1 se il vertice $v \in V$ appartiene al cammino, 0 altrimenti.
- c_e: costo associato all'arco e.

Per ogni sottoinsieme $S \subseteq V$, definiamo:

- $E(S) = \{[i, j] \in E : i \in S, j \in S\}$ insieme di archi i cui nodi appartengono a S
- $\delta(S) = \{[i, j] \in E : i \in S, j \notin S\}$ insieme di archi che hanno solo un nodo appartenente a S

Ora si può definirie E-GTSP come:

$$\min \sum_{e=1}^{|E|} c_e x_e$$

s.t.

$$\sum_{e \in \delta(S)} x_e = 2y_v \text{ per } v \in V$$

$$\sum_{v \in P_h} y_v = 1 \text{ per } h = 1, ..., n$$

$$\sum_{e \in \delta(S)} x_e \geq 2(y_i + y_j - 1) \text{ per } S \subset V, 2 \leq |S| \leq n-2, i \in S, j \in V \backslash S$$

Il grafo sul quale viene eseguito il solver è così composto: per ogni parola w_i si aggiunge un nodo per ogni senso s_{ij} della parola. Ogni parola w_i forma quindi un cluster di nodi, non connessi fra loro, fra i quali verrà scelto il senso corretto.

Per ogni coppia di sensi s_{ij}, s_{kl}, che disambiguano diverse parole, si vuole creare un arco e pesarlo in modo da avere minor distanza su archi che congiungono due nodi "centrali". Sfruttiamo le centralità precedentemente calcolate e le distibuiamo sugli archi in modo che il peso dell'arco e_{ijkl} sia la media delle centralità dei nodi. Definiamo quindi come $c(i, j)$ la centralità del nodo rappresentante il senso s_{ij}:

$$w(e_{ijkl}) = avg(c(i, j), c(k, l))$$

I pesi degli archi non sono delle distanze, ma maggiore è il peso di un arco, più i due nodi alle estremità sono importanti (perchè il peso deriva dalle centralità). Il solver cerca il cammino minimo che coinvolga un solo nodo per ogni cluster, quindi la distanza degli archi va invertita e convertita in un numero intero.

Il grafo viene sottoposto al solver GLKH [20] che tramite l'euristica di Lin-Kerninghan-Helsgaun restituisce il cammino minimo che attraversa un nodo per ogni cluster.

Nel caso di frasi con una sola parola da disambiguare viene preso il senso più comune perchè non è possibile, e non ha molto significato, eseguire il solver su un unico cluster.

3.2 Combinare similarità sintattica e algoritmo graph-based

Partendo da un esperimento già esistente, si è voluto aggiungere e combinare la conoscenza ottenuta dall'esperimento graph-based confrontando poi i risultati.

3.2.1 Descrizione algoritmo

L'esperimento sfrutta una misura di similarità calcolata per ogni coppia di sensi. Si pesano gli archi fra due nodi rappresentanti due sensi con il valore restituito dalla misura di similarità e si esegue TSP per trovare il cammino minimo che comprenda un solo senso per parola da disambiguare.

Similarità sintattica tramite Tree Kernel La misura di similarità è ottenuta tramite l'utilizzo di Tree Kernel [22] che rappresentano in strutture ad albero le dipendenze grammaticali e lessicali dei sensi presi in considerazione. I Tree Kernel si occupano poi di confrontare le due strutture e ricavare quanto siano simili fra loro.

In questo caso i Tree Kernel (in particolare valutando i subTree) sono stati utilizzati per confrontare la similarità sintattica fra le coppie di sensi, ottenendo così un valore con il quale pesare gli archi del grafo di disambiguazione.

3.2.2 Aggiunta sensi ausiliari per calcolo centralità

Sulla base di quanto sperimentato precedentemente, si è voluto aggiungere la conoscenza fornita da WordNet: trovare tutti i sensi che sono in relazione con quelli della frase e aggiungerli al grafo della disambiguazione, per dare una forma diversa e centralizzare i sensi strettamente connessi fra loro.

Si effettua una ricerca DFS di WordNet se si trova un cammino con al massimo n archi che congiunge due sensi già presenti nella frase allora i nodi e gli archi intermedi vengono aggiunti al grafo della disambiguazione. In questo modo si aggiungono sensi molto utili alla disambiguazione perchè strettamente correlati con gli altri già presenti (massimo 3 archi di profondità per non sbilanciare troppo il grafo).

A partire dal grafo arricchito con i sensi ausiliari si calcolano le centralità dei nodi e si esegue la disambiguazione.

3.3 Strumenti e librerie

La fonte di conoscenza utilizzata per questo esperimento è il dizionario di WordNet. Esso può essere visto come un grosso grafo: i nodi del grafo sono i significati semantici (synset) delle parole e i loro sensi più stretti, gli archi rappresentano le varie relazioni lessicali e semantiche.

Per evitare di dover interagire direttamente con i file di WordNet si è deciso di utilizzare una libreria già esistente e collaudata, JWI [23].

JWI Java Wordnet Interface (JWI) è la libreria Java utilizzata per interfacciarsi con il dizionario Wordnet. Consente di fare scraping delle informazioni in modo rapido, strutturato ed efficiente. Mette a disposizione delle funzioni per ottenere parole indice ("parola - POS", utili al fine della ricerca) e significati semantici in modo efficiente. Carica in memoria completamente il dizionario in modo tale da poterlo interrogare molto velocemente. Il grafo rappresentante WordNet permette di essere esplorato anche seguendo le relazioni lessicali e semantiche che collegano le parole, e possono essere sfruttate delle classi built-in per una agile manipolazione dei dati.

Ogni senso viene etichettato con la relativa *part of speech*, in modo da avere una maggiore efficacia nella ricerca. Ogni coppia di valori "parola - POS" viene definita *parola indice*. Ad ogni parola indice viene associato un numero finito di sensi, ordinati per senso più comune.

Se nella frase è presente la parola *keep* che può essere sia nome che verbo, nei dataset utilizzati viene annotata con la corretta *part of speech* in modo che si ricerchino i sensi per parola indice avendo una maggiore accuratezza.

Ad esempio, in uno stesso synset sono presenti sensi di parole diverse che hanno lo stesso significato e che possono essere interscambiati fra di loro (sinonimi). Le relazioni sono etichettate, per distinguere *ipernomi*, *participi*, *attributi*, *antonimi*, *sinonimi* e così via.

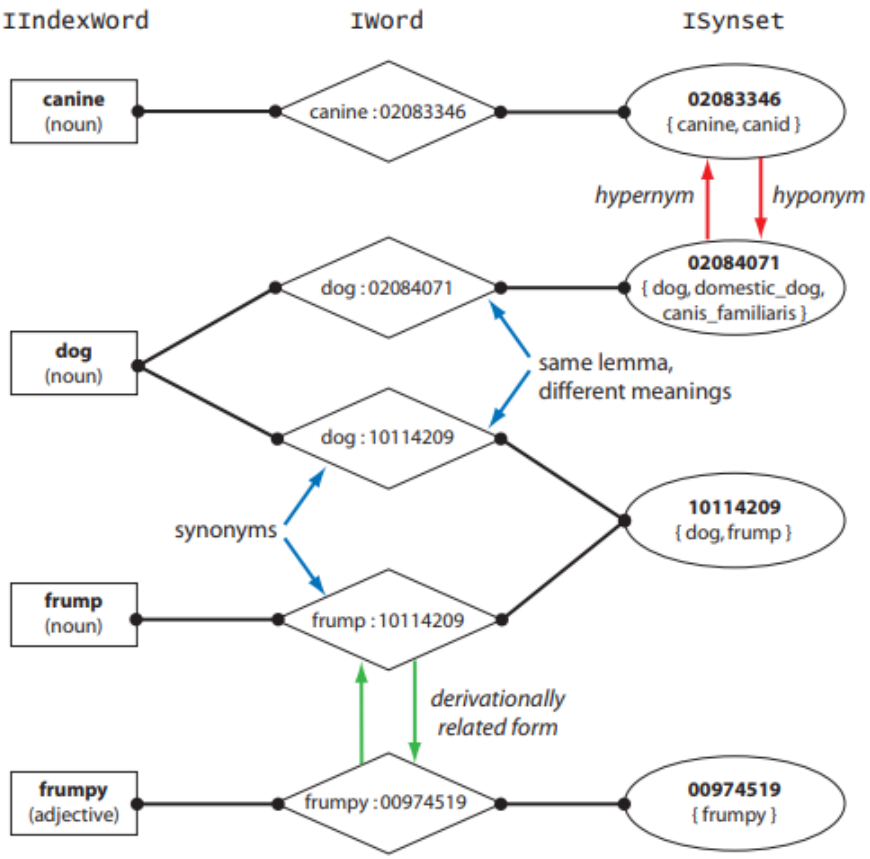


Figura 3.4: Struttura di WordNet tramite JWI: "parole indice", sensi, significati semantici, sinonimi, relazioni lessicali (frecce verdi) e semantiche (frecce rosse)

Ogni senso può essere quello corretto nel contesto rappresentato dalla frase.

Capitolo 4

Ottimizzazione dei tempi di esecuzione

In questo capitolo vengono descritte le ottimizzazioni applicate per migliorare i tempi di esecuzione dell'algoritmo sui vari dataset. Sono state effettuate prove su datasets che, in totale, contengono più di 1000 frasi, di diversa grandezza e quindi computazionalmente più o meno impegnative.

4.1 Analisi dei bottlenecks

La prima valutazione è stata fatta su quali fossero gli eventuali bottlenecks dell'algoritmo, e in che modo si potessero superare. In base alla diversa fase in cui l'algoritmo si trova vengono più o meno utilizzate le risorse hardware disponibili. Le fasi dell'algoritmo che sono computazionalmente pesanti sono:

- Esplorazione del grafo di WordNet: Per ogni parola vanno recuperati tutti i sensi e per ogni senso va fatta poi una ricerca DFS a profondità variabile, in base al numero massimo archi di profondità e al numero di sensi dai quali iniziare la ricerca si hanno prestazioni molto differenti.
- Esecuzione del solver TSP: E' una fase computazionalmente impegnativa perchè nonostante il solver sia ottimizzato, su grandi grafi aventi nodi nell'ordine delle centinaia e archi nell'ordine delle migliaia, anche le migliori euristiche impiegano molto la CPU.

4.1.1 Fase di test

La fase di test è molto lunga: per ogni versione dell'algoritmo, si vogliono verificare le performance di tutte le possibili configurazioni sui singoli datasets. Le performance dipendono strettamente dai parametri di configurazione dell'algoritmo stesso: numero di archi massimo della ricerca DFS, algoritmo di centralità scelto, esecuzione del solver.

Per non mischiare i risultati di ogni configurazione si è creato un file con tutti i possibili parametri: la computazione viene avviata indipendentemente per ogni combinazione su ogni frase da analizzare di ogni dataset.

Si viene a creare un problema per il quale alcune fasi dell'algoritmo vengono eseguite più volte con lo stesso input e quindi con lo stesso output; possiamo pensare ad esempio alla ricerca DFS su WordNet dei sensi di una frase: parametri come l'algoritmo scelto di centralità non influiscono sul grafo risultante dalla ricerca, è importante quindi non ricomputare più volte se non necessario.

Durante i test, l'algoritmo proseguiva ma le risorse non venivano sfruttate pienamente. Se concettualmente ogni frase è a sè stante, praticamente ci sono delle risorse comuni che vengono utilizzate da ogni frase. Modificare l'algoritmo in modo da parallelizzare l'esecuzione della specifica configurazione su una singola frase sarebbe stato il primo obiettivo che avrebbe portato sicuramente notevoli miglioramenti. Si è iniziato quindi un processo di refactor tale per cui, ogni thread potesse eseguire tutto l'algoritmo in modo atomico su una frase, senza aspettare la fine degli altri threads.

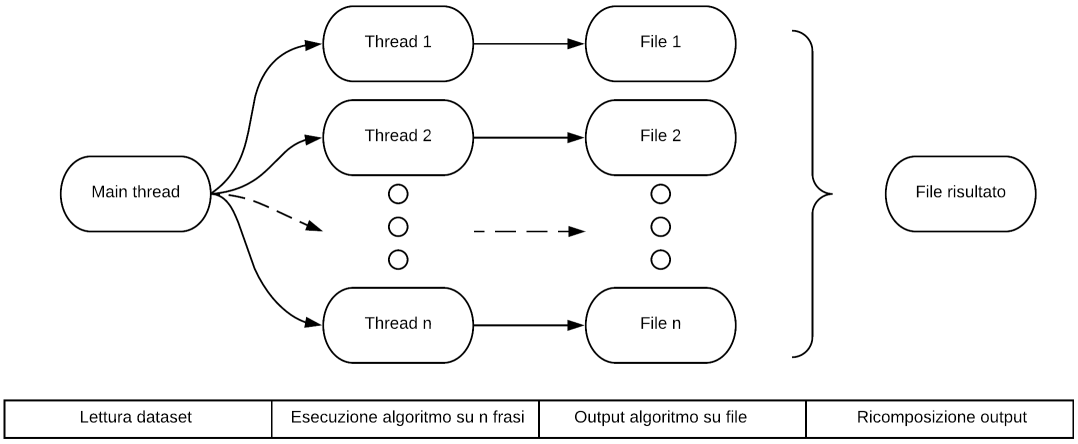


Figura 4.1: Ottimizzazione dei tempi di esecuzione grazie ai threads

In base alla macchina sulla quale andremo a eseguire l'algoritmo quindi si avranno benefici derivanti dalla potenza di calcolo della CPU e dal numero di thread in esecuzione in parallelo.

Ogni thread completa una singola configurazione dell'algoritmo su una singola frase, questo comporta uno spreco di risorse: la ricerca dei sensi su una stessa frase nel dizionario, ad esempio, è uguale per tutte le configurazioni, non ha senso eseguirla ogni volta.

Una prima ottimizzazione è stata quindi riutilizzare l'output di funzioni specifiche: il grafo, costruito a partire dai sensi delle parole presenti nella frase viene creato una volta sola, e poi utilizzato più volte con diversi algoritmi di centralità, e diverse esecuzioni del solver.

L'intero dizionario di WordNet viene caricato in memoria, in modo da essere leggibile molto più velocemente, ed è consultabile da tutti i threads.

Il solver TSP è un eseguibile che viene richiamato dall'interno del programma. Esso è configurabile tramite un file di input nel quale vengono specificati i file e i parametri che deve utilizzare. Ogni thread deve attendere che il solver finisca l'esecuzione precedente per poi modificare il file di configurazione ed eseguire TSP. Questo comporta uno spreco di tempo CPU (i threads devono stare in *wait* per troppo tempo, il solver TSP può impiegare anche diversi minuti) e introduce un nuovo bottleneck.

La soluzione applicata a questo problema è stata: ogni thread crea i propri file di input al solver, identificati dall'id della frase e dalle configurazioni dell'algoritmo, crea dinamicamente uno script che avvia TSP con i riferimenti ai file appena creati, avvia il processo e ne attende il risultato.

L'output di TSP viene letto e trasformato in un formato leggibile dallo *Scorer* che ne valuta la precision.

4.2 Tempi di esecuzione

Per confrontare i tempi di esecuzione dell'algoritmo con diverse configurazioni è necessario avere una build del progetto riproducibile e tentare di avere sempre le stesse condizioni nello stesso ambiente. Per questo motivo si è pensato di costruire un'immagine Docker in cui vengono scaricate le librerie e gli strumenti necessari per il funzionamento dell'algoritmo.

Macchina di test L'immagine Docker è stata caricata poi su una macchina virtuale hostata su Google Cloud Engine. La macchina è configurata con 8 vCPU, 8

GB RAM e disco SSD.

Test eseguiti Sono state lanciate in esecuzione tutte le configurazioni possibili sul dataset *All* (concatenazione di tutti i dataset, in totale 1172 frasi, 7253 parole da disambiguare): da 2 a 4 archi massimi della ricerca DFS, 4 algoritmi di centralità, disambiguazione per centralità e per TSP.

Si è testato l'algoritmo in 3 fasi della sua ottimizzazione per mostrare come si è riusciti ad abbattere drasticamente i tempi di esecuzione:

- Single-thread: il main thread esegue tutte le configurazioni una dopo l'altra in modo sequenziale, senza parallelismi di alcun tipo
- Multi-thread (no ottimizzazione): il main thread crea un pool di threads ausiliari, ciascun thread esegue tutte le configurazioni di una singola frase
- Multi-thread: i threads ausiliari riutilizzano l'output di funzioni ripetute (ricerca DFS e costruzione grafo) senza ricomputarle ogni volta

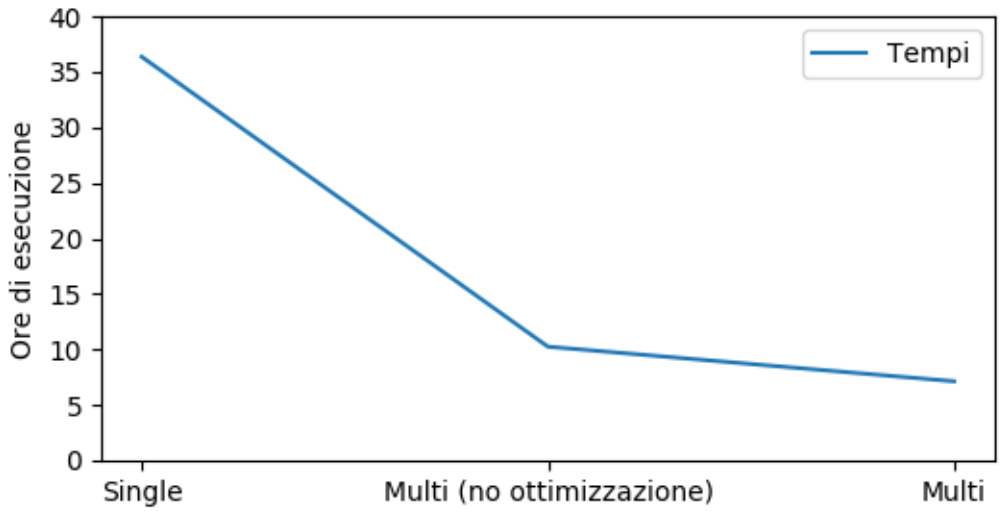


Figura 4.2: Ottimizzazione tempi

Capitolo 5

Risultati e conclusioni

Presentiamo qui le misure di valutazione utilizzate per confrontare i vari approcci, i risultati ottenuti e le considerazioni che possono essere fatte per un possibile miglioramento.

5.1 Misure di valutazione

Le misure di valutazione utilizzate sono le stesse del campo dell'information retrieval. Si riportano le definizioni come in [14].

Sia $T = (w_1, w_2, ..., w_n)$ un testo composto da n parole da disambiguare e A una funzione "risposta" che associa ad ogni parola $w_i \in T$ un insieme di sensi corretti dal dizionario D. Dato un assegnamento $A'(i) \in Senses_D(w_i) \cup \{\epsilon\}$ fornito dal sistema WSD, possiamo definire *coverage* C la percentuale di risposte fornite dal sistema per il numero totale di risposte da fornire dal testo, cioè:

$$C = \frac{\#\text{ risposte fornite}}{\#\text{ risposte totali da fornire}} = \frac{|\{i \in \{1, ..., n\} : A'(i) \neq \epsilon\}|}{n}$$

Indichiamo con ϵ il caso in cui il sistema non fornisca una risposta per una specifica parola w_i, $(A'(i) = \epsilon)$. Il numero totale di risposte da fornire è il numero n di parole da disambiguare nel testo.

Indichiamo con P la *precision*, calcolata come la percentuale di risposte corrette fornite dal sistema sul numero totale di risposte fornite, cioè:

$$P = \frac{\#\text{ risposte corrette fornite}}{\#\text{ risposte fornite}} = \frac{|\{i \in \{1, ..., n\} : A'(i) \in A(i)\}|}{|\{i \in \{1, ..., n\} : A'(i) \neq \epsilon\}|}$$

La *precision* determina quanto siano buone le risposte fornite dal sistema che stiamo valutando.

La *recall* R è la percentuale di risposte corrette fornite dal sistema sul totale di risposte da fornire:

$$P = \frac{\text{\# risposte corrette fornite}}{\text{\# risposte totali da fornire}} = \frac{|\{i \in \{1, ..., n\} : A'(i) \in A(i)\}|}{n}$$

Abbiamo sempre che $R \leq P$. E nel caso la coverage sia al 100%, abbiamo $P = R$. Per prevenire al problema di quale indicatore di score utilizzare per i confronti è stata introdotta la misura chiamata F_1-*measure* o anche chiamata *F-score*, definita come la media armonica pesata di *precision* e *recall*:

$$F_1 = \frac{2PR}{P + R}$$

Nel nostro approccio, avente *coverage* = 100%, $F_1 = P = R$, confrontiamo quindi le nostre performance con le *F-measure* degli altri sistemi.

5.2 Baselines

Si descrivono le baselines di riferimento, standard con i quali vengono poi confrontate le performance: quella casuale e quella del senso più comune.

Baseline casuale Consideriamo D il dizionario di riferimento e $T = (w_1, w_2, ..., w_n)$ il testo contenente le istanze di test w_i. La baseline casuale consiste nella scelta casuale di un senso fra tutti quelli disponibili nel dizionario per ogni parola w_i. Per ogni parola quindi la probabilità di scegliere il senso corretto è $\frac{1}{|Senses_D(w_i)|}$.

L'accuracy si calcola facendo la media di tutte le probabilità in T:

$$A = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{|Senses_D(w_i)|}$$

Baseline senso più comune Considerato il dizionario D si sceglie sempre il senso più comune per ogni parola w_i. A seconda del dizionario esso può essere diverso o calcolato in vari modi. Prendendo come riferimento WordNet, il senso più comune è scelto sulla base della frequenza delle occorrenze di ogni senso nel corpus SemCor: i sensi vengono salvati in ordine di frequenza nel dizionario in modo che il primo sia quello più comune.

Si farà la scelta corretta se il senso più comune di ogni parola w_i^1 sarà fra quelli corretti $C(i)$, ottenendo così l'accuracy:

$$A = \frac{|\{i \in \{1, ..., n\} : w_i^1 \in C(i)\}|}{n}$$

5.3 Dataset

Per confrontare le performance con altri sistemi WSD è necessario avere le stesse istanze di input: sono stati creati dataset annotati manualmente per alcune competizioni internazionali, vengono presi in considerazione alcuni di essi perchè sono utilizzati molto dallo stato dell'arte e si suppone siano stati etichettati correttamente con piccolissimi margini di errore. I 5 dataset utilizzati provengono tutti dalla stessa competizione (Senseval, poi rinominata Semeval), le istanze sono etichettate con: senso corretto di WordNet, parte del discorso e lemma. Ad ogni istanza vengono date queste informazioni per restringere il campo dei possibili sensi, in modo che l'algoritmo si possa concentrare esclusivamente sulla disambiguazione.

Precisamente l'algoritmo è stato testato su: *Senseval-2*, *Senseval-3*, *Semeval-2007*, *Semeval-2013* e *Semeval-2015*. Complessivamente si tratta di 1172 frasi, con 7253 parole da disambiguare

Senseval-2 Utilizza un dizionario di sensi più dettagliato (WordNet 1.7) rispetto a quello della precedente competizione (Senseval-1). Consiste in 242 frasi, 2282 parole annotate ed ha la minore ambiguità (5.4) fra i dataset considerati.

Senseval-3 E' composto da 3 documenti appartenenti a diversi ambiti (editoria, news e fantascienza), consistenti di 353 frasi, 1850 significati ed è molto più ambiguo di Senseval-2 (6.8).

Semeval-2007 Si utilizza WordNet 2.1 come dizionario, è il dataset più piccolo preso in considerazione (solo 135 frasi) ma è quello con il maggior livello di ambiguità (8.5). I sensi annotati sono 485.

Semeval-2013 Utilizza WordNet 3.0 come dizionario ed è il più grande in tema di numero di argomenti (14 documenti di diverso genere). E' composto da 6 lingue.

Semeval-2015 Il dizionario di riferimento è stato BabelNet 2.1.5 [24] e consiste in 138 frasi e 1022 significati annotati. L'ambiguità stimata è di 5.5.

5.4 Analisi risultati ottenuti

I risultati ottenuti con l'approccio utilizzato sono molto diversi in base alle configurazioni scelte: in primis, si devono separare quelli ottenuti tramite l'euristica TSP applicata al grafo di disambiguazione e quelli ottenuti scegliendo il senso di ogni parola con centralità massima nel cluster.

La costruzione del grafo delle centralità si basa sulla ricerca DFS del grafo di WordNet, e produce grafi molto diversi a seconda del numero massimo di archi di profondità. Più archi vengono presi in considerazione, più il grafo è grande e sbilanciato, meno è possibile che due nodi abbiano lo stessa centralità nello stesso cluster. Disambiguando per centralità, si sceglie il nodo che rappresenta il senso più comune fra quelli con valore massimo. Disambiguando tramite TSP invece non è detto che l'euristica scelga il nodo con centralità massima nel cluster poichè i nodi degli altri cluster influiscono nella scelta del cammino ottimo.

Ad esempio si può notare come, su 1536 parole disambiguate con centralità uguale a 0 (tutti i sensi hanno la stessa centralità, si sceglie il più comune), 691 siano state le scelte del senso più comune. Nel 55% dei casi infatti l'euristica TSP decide un altro senso.

Passando ai risultati veri e propri della disambiguazione TSP, si può vedere in dettaglio come, all'aumentare degli archi di profondità aumenti la precisione dell'algoritmo:

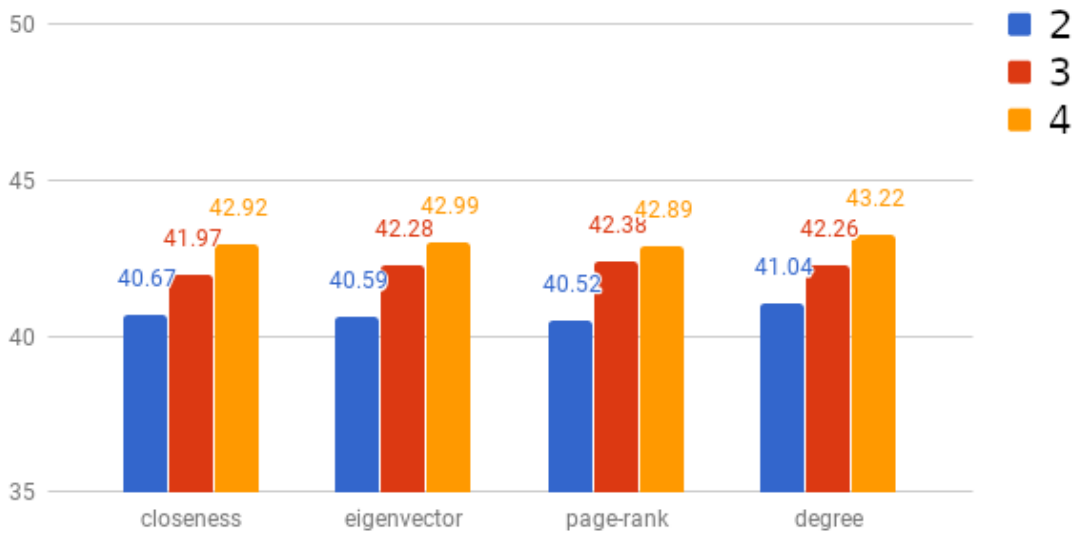


Figura 5.1: Disambiguazione per TSP suddivisi per il numero di archi massimo della DFS

Se confrontati con lo stato dell'arte però notiamo che siamo al di sotto della concorrenza:

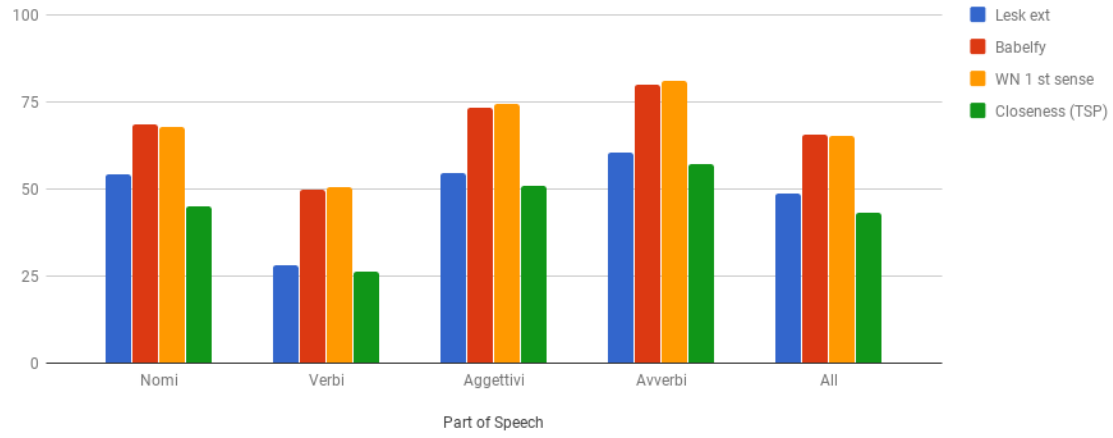


Figura 5.2: Confronto disambiguazione TSP con lo stato dell'arte, risultati divisi per *part of speech*

Lo stato dell'arte (Babelfy e WordNet) fa risultati migliori, ma l'approccio proposto ha dei notevoli margini di miglioramento: basti pensare al fatto che non viene utilizzata nessuna misura di similarità diretta fra i sensi, i pesi degli archi sono uniformi e le centralità si basano unicamente sul grafo derivante dalla ricerca DFS del grafo di WordNet.

Se notiamo invece i risultati ottenuti con la disambiguazione tramite centralità, ci avviciniamo molto di più allo stato dell'arte, arrivando quasi al 60% di *performance*.

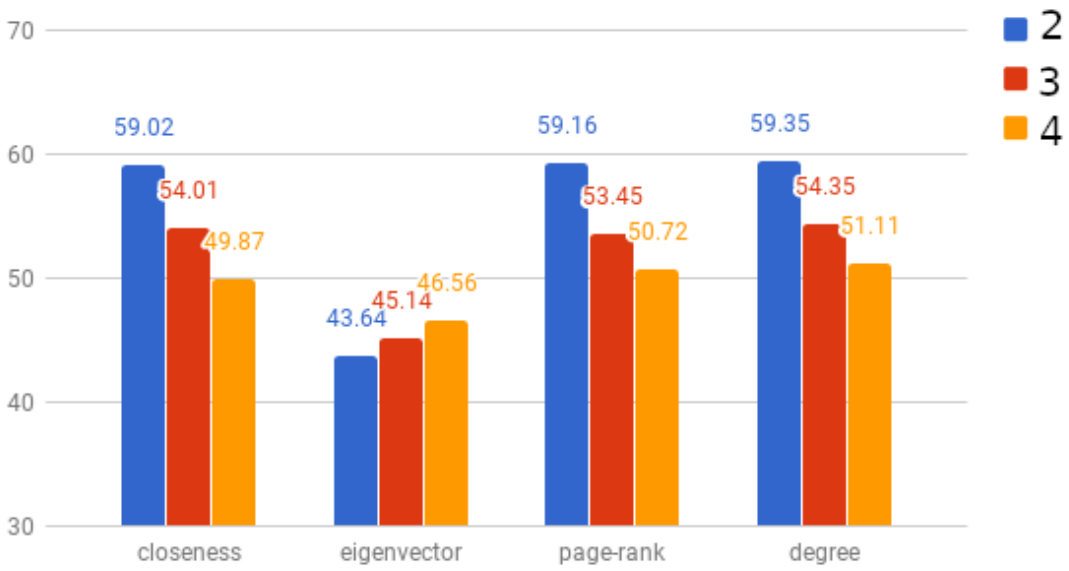


Figura 5.3: Disambiguazione per centralità suddivisi per il numero di archi massimo della DFS

Si potrebbe notare una controtendenza: se prima, disambiguando con TSP, all'aumentare degli archi di profondità aumentavano le *performance*, ora diminuiscono notevolmente. Questo è dovuto al fatto che la disambiguazione per centralità è molto *biased* verso il *Most Common Sense* perchè la ricerca DFS con pochi archi di profondità genera poco squilibrio nel grafo. Questo fa sì che, calcolando le centralità si ottengono molti nodi "centrali", che vengono disambiguati scegliendo il senso più comune.

Come si può notare dai dati raccolti, solo nel caso si utilizzino 2 archi di profondità della DFS si arriva al 77% di *bias* verso il *Most Common Sense*. A differenza degli algoritmi supervised che disambiguano i sensi più comuni nel 71% e 75% [25] , utilizzando TSP si ha il 43% di disambiguazioni secondo il most common sense (con

DFS profonda 4 archi), mentre per centralità il 54%. Meno profonda è la DFS più sensi comuni vengono disambiguati.

Profondità DFS	TSP	Centralità
2	39%	77%
3	41%	63%
4	43%	54%

Tabella 5.1: Percentuale di most common sense disambiguati

Confrontiamo ora i risultati ottenuti per centralità con quelli dello stato dell'arte:

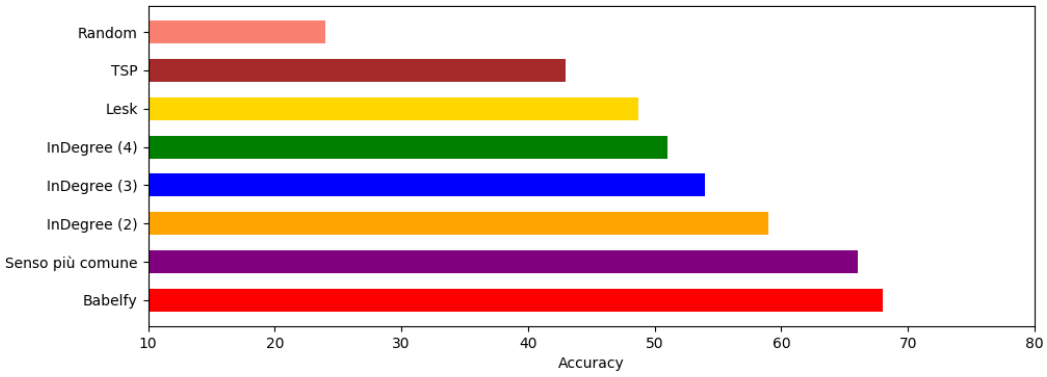


Figura 5.4: Confronto disambiguazione per centralità con lo stato dell'arte

Confrontando la sola disambiguazione per centralità con quelle del paper Graph Connectivity (cita paper qui) si nota come si abbiano dei risultati molto simili. In particolare, sul dataset Senseval-3, suddivisi per *part of speech*:

Part of speech	Paper	Tesi		
Profondità DFS		2	3	4
Nomi	61.9%	62.4%	54.9%	46.9%
Aggettivi	62.8%	68.8%	64.5%	53.1%
Verbi	36.1%	39.8%	37.4%	34.0%

Tabella 5.2: Confronto con lo stato dell'arte (Graph Connectivity), Senseval-3

Se si analizzano le performance ottenute con provando a combinare l'algoritmo di similarità si può notare come ci sia un miglioramento di qualche punto percentuale rispetto all'esperimento originale:

Dataset	Similarità	Similarità + DFS
Profondità DFS		3
Senseval-3	38.2%	40.7%
Semeval-2015	48.6%	51.3%
All	41.3%	43.2%

Tabella 5.3: Confronto con disambiguazione per similarità

Questo è sicuramente dovuto al fatto che introdurre sensi correlati, ma non presenti nella frase, aiuta a capire il senso generale, perchè spesso alcune parole significative vengono sottintese.

5.5 Conclusioni

Da tutti gli esperimenti eseguiti si apprende come scegliere le giuste configurazioni cambi tantissimo i risultati, l'aggiunta dei nodi ausiliari (i sensi correlati a più di un senso nella frase) è importantissima: come si può vedere nell'ultimo esperimento, alza le performance di quasi 3 punti percentuali su un algoritmo già esistente e collaudato.
La rappresentazione del problema come problema di ottimizzazione, se in primo luogo sembra non avere precision molto alta, è importantissima perchè molto flessibile. Si può utilizzare come peso degli archi una qualsiasi misura e poi confrontare i risultati. La disambiguazione per centralità invece ha il vantaggio di essere molto veloce ma è molto condizionata dal *bias* del senso più comune.
Sperimentando si è capito come non bisogna andare troppo a fondo (non aumentare troppo il numero di archi) nella ricera per evitare che si aggiungano sensi troppo distanti che sbilancino molto il grafo pur essendo non così importanti.
Un possibile sviluppo è quello di esplorare maggiormente le fonti di conoscenza ricavando più informazioni dagli insiemi di significati semantici, si possono sfruttare maggiormente le strutture delle frasi di esempio, le definizioni e scegliere con maggiore accuratezza le relazioni in WordNet.

Bibliografia

[1] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[2] George A. Miller, Claudia Leacock, Randee Tengi, and Ross T. Bunker. A semantic concordance. In *Proceedings of the Workshop on Human Language Technology*, HLT '93, pages 303–308, Stroudsburg, PA, USA, 1993. Association for Computational Linguistics. ISBN 1-55860-324-7. doi: 10.3115/1075671.1075742. URL `https://doi.org/10.3115/1075671.1075742`.

[3] Ronald L Rivest. Learning decision lists. *Machine learning*, 2(3):229–246, 1987.

[4] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[5] Alessandro Raganato, Claudio Delli Bovi, and Roberto Navigli. Neural sequence learning models for word sense disambiguation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1156–1167, 2017.

[6] Xue Lei, Yi Cai, Qing Li, Haoran Xie, Ho-fung Leung, and Fu Lee Wang. Combining local and global features in supervised word sense disambiguation. In *International Conference on Web Information Systems Engineering*, pages 117–131. Springer, 2017.

[7] Dmitry Ustalov, Denis Teslenko, Alexander Panchenko, Mikhail Chernoskutov, Chris Biemann, and Simone Paolo Ponzetto. An unsupervised word sense disambiguation system for under-resourced languages. *arXiv preprint arXiv:1804.10686*, 2018.

[8] Alexander Panchenko, Fide Marten, Eugen Ruppert, Stefano Faralli, Dmitry Ustalov, Simone Paolo Ponzetto, and Chris Biemann. Unsupervised,

knowledge-free, and interpretable word sense disambiguation. *arXiv preprint arXiv:1707.06878*, 2017.

[9] Michael Lesk. Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In *Proceedings of the 5th Annual International Conference on Systems Documentation*, SIGDOC '86, pages 24–26, New York, NY, USA, 1986. ACM. ISBN 0-89791-224-1. doi: 10.1145/318723.318728. URL http://doi.acm.org/10.1145/318723.318728.

[10] Jane Morris and Graeme Hirst. Lexical cohesion computed by thesaural relations as an indicator of the structure of text. *Computational linguistics*, 17(1):21–48, 1991.

[11] Neeraja Koppula, B Padmaja Rani, and Koppula Srinivas Rao. Graph based word sense disambiguation. In *Proceedings of the First International Conference on Computational Intelligence and Informatics*, pages 665–670. Springer, 2017.

[12] Haw-Shiuan Chang, Amol Agrawal, Ananya Ganesh, Anirudha Desai, Vinayak Mathur, Alfred Hough, and Andrew McCallum. Efficient graph-based word sense induction by distributional inclusion vector embeddings. *arXiv preprint arXiv:1804.03257*, 2018.

[13] Andrea Moro, Alessandro Raganato, and Roberto Navigli. Entity linking meets word sense disambiguation: A unified approach. *Transactions of the Association for Computational Linguistics*, 2:231–244, 2014. ISSN 2307-387X. URL https://transacl.org/ojs/index.php/tacl/article/view/291.

[14] Roberto Navigli. Word sense disambiguation: a survey. *ACM COMPUTING SURVEYS*, 41(2):1–69, 2009.

[15] Roberto Navigli and Mirella Lapata. Graph connectivity measures for unsupervised word sense disambiguation. In *IJCAI*, pages 1683–1688, 2007.

[16] Larry Page, Sergey Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web, 1998.

[17] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30 (1):107 – 117, 1998. ISSN 0169-7552. doi: https://doi.org/10.1016/S0169-7552(98)00110-X. URL http://www.sciencedirect.com/science/article/pii/S016975529800110X. Proceedings of the Seventh International World Wide Web Conference.

[18] Phillip Bonacich. Some unique properties of eigenvector centrality. *Social Networks*, 29(4):555 – 564, 2007. ISSN 0378-8733. doi: https://doi.org/10.1016/j.socnet.2007.04.002. URL `http://www.sciencedirect.com/science/article/pii/S0378873307000342`.

[19] Gregory Gutin and Abraham P Punnen. *The traveling salesman problem and its variations*, volume 12. Springer Science & Business Media, 2006.

[20] Keld Helsgaun. Solving the equality generalized traveling salesman problem using the lin–kernighan–helsgaun algorithm. *Mathematical Programming Computation*, 7(3):269–287, Sep 2015. ISSN 1867-2957. doi: 10.1007/s12532-015-0080-8. URL `https://doi.org/10.1007/s12532-015-0080-8`.

[21] Matteo Fischetti, Juan José Salazar González, and Paolo Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394, 1997.

[22] Castellucci G. Basili R. Croce D., Filice S. Kelp: Kernel-based learning platform official guide of the version 2.0.2 - draft version. *SAG - Semantic Analytics Group University of Roma - Tor Vergata.*

[23] Mark Finlayson. Java libraries for accessing the princeton wordnet: Comparison and evaluation. In *Proceedings of the Seventh Global Wordnet Conference*, pages 78–85, 2014.

[24] Andrea Moro and Roberto Navigli. Semeval-2015 task 13: Multilingual all-words sense disambiguation and entity linking. In *Proceedings of the 9th international workshop on semantic evaluation (SemEval 2015)*, pages 288–297, 2015.

[25] Alessandro Raganato, Jose Camacho-Collados, and Roberto Navigli. Word sense disambiguation: A unified evaluation framework and empirical comparison. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, volume 1, pages 99–110, 2017.