ALMA MATER STUDIORUM - UNIVERSITA' DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI SCIENZE

CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

Aggregazione di dati testuali in MoK: matchmaking basato su ontologie o similarità

Relazione finale in

Sistemi Distibuiti

Relatore:
Chiar.mo Prof. Ing.
Andrea Omicini

Correlatore:
Ing. Stefano Mariani

Presentata da:
Matteo Fattori

III Sessione

Anno Accademico 2016/2017

Indice

Capitolo 1

Introduzione

La conoscenza, espressa in linguaggio naturale, può essere elaborata attraverso approcci che sfruttano le misure di similarità oppure i motori di inferenza semantica basati su ontologie. Entrambi propongono due tipologie differenti di elaborazione: il primo confronta la conoscenza attraverso algoritmi che restituiscono un valore indicativo del livello di **similarità sintattica**, mentre il secondo si basa sul ragionamento formale e rappresenta un'estensione della logica del primo ordine attraverso la quale è possibile quantificare il livello di **similarità semantica**.
Nel Web Semantico, i motori di inferenza semantica basati su ontologie rappresentano una soluzione comune per realizzare applicazioni che gestiscono informazioni testuali. Nonostante i costanti miglioramenti in termini di performance, essi effettuano spesso computazioni costose anche per inferenze semplici, pertanto risulta difficile pensare di utilizzarli in scenari come l'*Internet of Things* o i sistemi pervasivi in generale. Inoltre si basano sulla **completezza** e **correttezza** delle ontologie su cui avviene il ragionamento. Le misure di similarità invece richiedono meno risorse di calcolo e sono più efficienti ma forniscono risultati meno precisi e sono meno potenti a livello espressivo. La sfida principale consiste nell'applicare e mettere a confronto i due approcci per verificare le caratteristiche evindenziate finora nel contesto dell'aggregazione di conoscenza.
Partendo dal contesto e dalle motivazioni appena descritte si vuole determinare lo strumento più adatto, in termini di applicabilità, espressività e prestazioni, per ogni approccio di elaborazione della conoscenza, con l'obiettivo di realizzare meccanismi di matchmaking fra parole, frasi e documenti. Inoltre introducendo il modello di coordinazione *Molecules of Knowledge* (MoK) si prevede la progettazione di un sistema che utilizzi tali strumenti per aggregare testi seguendo il modello MoK. Per ognuno degli approcci, verrà realizzata e testata una versione del suddetto sistema.

Capitolo 2

Calcolo di similarità

In letteratura lo studio di similarità di un insieme costituito da parole, frasi e documenti è stata parte integrante dell'elaborazione del linguaggio naturale. In questo capitolo vengono proposti due tipi di approcci diversi per il calcolo della similarità: quelli basati su ontologie e quelli basati su misure di similarità. I primi utilizzano l'inferenza come processo deduttivo e si basano su una descrizione formale del mondo, mentre i secondi ricavano un valore di similarità a partire dall'insieme in input.

2.1 Approcci basati su ontologie

Nel Web Semantico i motori inferenziali basati su ontologie, anche chiamati *Reasoner*, rappresentano una soluzione comune per fornire la capacità di ragionamento semantico ad applicazioni dedicate alla gestione di informazioni testuali.
I Reasoner sono sistemi in grado di inferire nuova conoscenza a partire dalle informazioni contenute in una *knowledge base* utilizzando una logica di descrizione. Quest'ultima serve a rappresentare, nel web semantico, il mondo attraverso *concepts*, *roles*, *individuals* e le relazioni fra di essi. Il paradigma che sta alla base di questi sistemi viene chiamato **frames paradigm** e concerne il salvataggio di concetti sotto forma di *frame concept* che si possono ricondurre alla rappresentazione di gerarchie di classi nel paradigma *Object Oriented*.
Come nella logica del primo ordine viene definita[1] una sintassi che rappresenta una collezione di simboli e/o di espressioni e una semantica che determina il significato.

[1] `https://en.wikipedia.org/wiki/Description_logic`

Sintassi

La sintassi viene definita da un insieme di concetti simili agli operatori della logica del primo ordine, viene fornito un esempio di notazione di seguito:

Symbol	Description	Example	Read
$\top$	$\top$ is a special concept with every individual as an instance	$\top$	top
$\bot$	empty concept	$\bot$	bottom
$\sqcap$	*intersection* or *conjunction* of concepts	$C \sqcap D$	C and D
$\sqcup$	*union* or *disjunction* of concepts	$C \sqcup D$	C or D
$\neg$	*negation* or *complement* of concepts	$\neg C$	not C
$\forall$	*universal restriction*	$\forall R.C$	all R-successors are in C
$\exists$	*existential restriction*	$\exists R.C$	an R-successor exists in C
$\sqsubseteq$	Concept *inclusion*	$C \sqsubseteq D$	all C are D
$\equiv$	Concept *equivalence*	$C \equiv D$	C is equivalent to D
$\doteq$	Concept *definition*	$C \doteq D$	C is defined to be equal to D
:	Concept *assertion*	$a : C$	a is a C
:	Role *assertion*	$(a, b) : R$	a is R-related to b

Semantica

La semantica della logica di descrizione è definita in base ai concetti interpretati come insieme di *individuals* o *roles* visti come coppie di *individuals*. La semantica di concetti e ruoli non atomici viene quindi definita in modo da descriverli come entità atomiche. Questo procedimento viene effettuato attraverso l'uso di una definizione ricorsiva utile per trasformare anche concetti e ruoli non atomici.

Utilizzando questa rappresentazione formale del mondo è possibile porre domande all'insieme di concetti (o *knowledge base*) in modo da verificare un insieme di proprietà in base ai concetti e le istanze descritte.
Le domande più comuni si possono ricondurre a query effettuate su un database che fondamentalmente controllano:

- **Le classi** : è possibile dedurre le relazioni gerarchiche (esempio il *Sub-Classing*) fra concetti basandosi su quantificatori esistenziali, proprietà e/o istanze.
- **Le istanze** : "se una particolare istanza è membro di un determinato concetto".
- **Le relazioni** : "esiste un ruolo/relazione tra due istanze?" anche espressa come "un ruolo ha una proprietà p?"

- **La subsunzione** : "dato un concetto a, questo è sottoinsieme di un concetto b?"
- **La consistenza** : "c'è una qualche contraddizione fra le definizioni o catena di definizioni?"

In particolare è possibile verificare **le relazioni** fra istanze e concetti attraverso l'utilizzo di *regole di distribuzione*[2] che sfruttano gli operatori sintattici definiti precedentemente, alcuni esempi sono:

Quantificatori Esistenziali

sono regole di distribuzione simili a quelle presenti nella logica proposizionale:

$$\exists p.(A \sqcup B) \equiv (\exists p.A) \sqcup (\exists p.B)$$

Figura 2.1: Unione

$$\exists p.(A \sqcap B) \sqsubseteq (\exists p.A) \sqcap (\exists p.B)$$
$$(\exists p.A) \sqcap (\exists p.B) \sqsubseteq (\exists p.A) \sqcup (\exists p.B)$$
$$(\exists p.A) \sqcap (\exists p.B) \sqsubseteq \exists p.(A \sqcup B)$$

Figura 2.2: Intersezione

Da cui si deduce che l'operatore di *unione* è distributivo, mentre quello di *intersezione* no.

Quantificatori Universali

similari a quelli esistenziali:

$$\forall p.(A \sqcap B) \equiv (\forall p.A) \sqcap (\forall p.B)$$

Figura 2.3: Unione

[2] `http://owl.man.ac.uk/2003/why/20031203/`

```
(∀p.A) ⊓ (∀p.B) ⊑ (∀p.A) ⊔ (∀p.B)
∀p.(A ⊓ B) ⊑ (∀p.A) ⊔ (∀p.B)
(∀p.A) ⊔ (∀p.B) ⊑ (∀p.(A ⊔ B))
```

Figura 2.4: Intersezione

Da cui si deduce che l'operatore di *intersezione* è distributivo, mentre quello di *unione* no.
Risulta quindi immediato tradurre queste regole utilizzando gli operatori di *OWL* come **intersectionOf** e **unionOf**, questo garantisce la possibilità di sfruttare questo tipo di inferenze, tipiche della logica proposizionale, durante il processo di interrogazione dell'ontologia.
Un esempio che mostra il potere espressivo di questi linguaggi, concatenato all'utilizzo di reasoner, è il seguente:
Supponiamo di avere una *knowledge base* di persone che descrive le proprietà di parentela fra di esse, allora è possibile attraverso prolog definire un insieme di fatti che descritti in linguaggio naturale determinano:

- Andrea è una persona.
- Giulio è una persona.
- Carlo è padre di Andrea.
- Carlo è padre di Giulio.

A questo punto è possibile inferire le relazioni di parentela fra Giulio e Andrea utilizzando una regola prolog espressa come clausola di Horn:

```
parent(?x,?z) ∧ parent(?y,?z) ⇒ sibling(?x,?y)
```

Che in linguaggio naturale rappresenta:

```
SE X e Y hanno lo stesso padre
ALLORA X e Y sono fratelli
```

Utilizzando ora un'ontologia di persone, che descrive i fatti citati precedentemente attraverso **concepts**, **individuals** e **properties**, è possibile effettuare la stessa interrogazione per mezzo delle OWL API:

```
// Proprieta
ObjectProperty hasSiblingProp = ontMod
```

```
            .getObjectProperty(person+ "hasSibling");
ObjectProperty hasParentProp = ontMod
            .getObjectProperty(person+ "hasParent");
ObjectProperty isParentOfProp = ontMod
            .getObjectProperty(person+ "isParentOf");
// Istanze
Individual Andrea = ontMod
            .getIndividual(person + "Andrea");
Individual Giulio = ontMod.
            .getIndividual(person + "Giulio");
Individual Carlo = ontMod
            .getIndividual(person + "Carlo");
// VALORI ESPLICITATI
Resource padreAndrea = (Resource) Andrea
            .getPropertyValue(hasParentProp);
Resource padreGiulio = (Resource) Andrea
            .getPropertyValue(hasParentProp);
// VALORI INFERITI
Resource fratelloAndrea = (Resource) Andrea
            .getPropertyValue(hasSiblingProp);
System.out.println("FATTI: ");
System.out.println("- " + Andrea.getLocalName() + " "
        + hasParentProp.getLabel("en") + " "
        + padreAndrea.getLocalName());
System.out.println("- " + Giulio.getLocalName() + " "
        + hasParentProp.getLabel("en") + " "
        + padreGiulio.getLocalName());
System.out.println("
NE SEGUE CHE (inferenza): ");
if (fratelloAndrea != null)
    System.out.println("- " + Andrea.getLocalName()
         + " " + hasSiblingProp.getLabel("en") + " "
        + fratelloAndrea.getLocalName());
else
    System.out.println("ERRORE! Nessun
                fratello inferito!");
}
```

In questo codice di esempio si tralascia tutta la parte di creazione e gestione preliminare dell'ontologia e, una volta eseguito, restituirà in output:

```
FATTI:
```

```
- Andrea has parent Carlo
- Giulio has parent Carlo

NE SEGUE CHE (inferenza):
- Andrea has sibling Giulio
```

2.2 Approcci basati su misure di similarità

In letteratura esistono diversi approcci per individuare un valore di similarità semantica fra parole, frasi e documenti. Di seguito si vuole dare una descrizione formale di queste relazioni in modo da facilitare al lettore la comprensione delle differenze tra i legami concettuali che si possono riscontrare in insiemi costituiti da parole, frasi o documenti.

Relazioni tra lemmi e significati

Le relazioni di tipo semantico fra parole, che verranno elencate di seguito in dettaglio, possono essere utilizzate come verifica del risultato degli algoritmi atti al calcolo del valore di similarità. Esse riguardano il confronto fra due lemmi[3] e solo alcune, come *Iperonimia* e *Iponimia*, sono riconducibili al concetto di relazione gerarchica, mentre altre riguardano relazioni di equivalenza o opposizione.

Sinonimia

Secondo una definizione attribuita a *Leibniz*, due espressioni sono sinonime se la sostituzione di una per l'altra non cambia il valore della frase in cui avviene la sostituzione. Da tale definizione, risulta chiaro che i veri sinonimi sono molto rari. Una definizione meno forte esprime la sinonimia in modo che due espressioni sono sinonime in un contesto C, se la sostituzione di una per l'altra in C non modifica il valore della frase.
Questa definizione rende necessaria la suddivisione in nomi, verbi, aggettivi e avverbi in modo che parole appartenenti a categorie diverse non possano essere sinonime visto che non possono essere intercambiabili.

Antinomia

La relazione di antinomia associa ad un termine il suo contrario. L'antinomo di una parola x, può essere $-x$ in alcuni casi ma non sempre, se consideriamo

[3] `https://it.wikipedia.org/wiki/Lemma_(linguistica)`

white e *black* questi sono antinomi, ma affermare che un oggetto non è bianco, non significa che sia necessariamente nero.
Questo tipo di relazione può essere usato per coppie di parole qualsiasi e rappresenta una relazione **sintattica** tra lemmi. Questa affermazione può essere rappresentata con un esempio: i significati rise, ascend e fall, descend sono concettualmente opposti ma non sono antinomi.

Iponimia e Iperonimia

L'iperonimia[4] mostra la relazione tra un termine generico, detto *iperonimo*, e una specifica istanza di esso, detto *iponimo*, un esempio consiste nella parola *Color* che è iperonimo e può avere *Red* e *Blue* come iponimi. La validità di questi concetti è mantenuta solo per le categorie di nomi e verbi in cui nel secondo caso la relazione viene denominata *troponimia*.
Questo tipo di relazione è asimmetrica e transitiva e genera una struttura gerarchica simile al concetto di generalizzazione presente nei modelli relazionali in cui nella radice sono presenti concetti più generici e alle foglie si hanno concetti specializzati.
Come nel mondo relazionale o nel paradigma ad oggetti, ogni iponimo eredita tutte le caratteristiche del concetto più generale e al più vengono aggiunte caratteristiche utili per contraddistinguere quella particolare istanza dalle altre.
È possibile "testare" l'iponimia fra nomi attraverso la frase *X is a kind of Y* mentre per la troponimia si dice che V1 è troponimo di V2 se vale la frase *V1 is to V2 in some particular manner*.

Iponimia

Oltre alla relazione gerarchica definita da iponimo e iperonimo, è possibile considerare la presenza di una relazione semantica fra due iponimi. Se prendiamo per esempio gli iponimi *Leaf* e *Branch* un'osservatore può inferire che essi condividono lo stesso iperonimo *Tree*. Questo tipo di inferenza è simile a un concetto di **Common Ancestor**[5] descritto nella teoria dei grafi e rappresenta una relazione di similarià semantica.

Meronimia

La Meronimia/olonimia[6] indica una relazione *part of* governata dalla frase *X is a part of Y* ed è applicabile solo alla categoria dei nomi. Questo tipo di

[4] `https://en.wikipedia.org/wiki/Hyponymy_and_hypernymy`
[5] `https://en.wikipedia.org/wiki/Lowest_common_ancestor`
[6] `https://en.wikipedia.org/wiki/Meronymy`

relazione è transitiva e asimmetrica e anch'essa può essere usata per generare una struttura gerarchica tenendo in considerazione che un meronimo può avere più olonimi.

Relazione causale

Anche in questo caso l'applicazione ricade solo sui verbi in cui un verbo X produce una causa C e un verbo Y "riceve" un effetto E. Un esempio è descritto dalla coppia di verbi <*show, see*>.

Relazione di pertinenza

La relazione di pertinenza, anche definita come *pertainym*[7], può essere applicata alla categoria degli aggettivi i quali possono essere definiti dalla frase *of* o *pertaining to* e non possiedono antinomi. Un aggettivo di pertinenza può essere in relazione con un nome o con un altro aggettivo di questo tipo, ad esempio *dental* è in relazione di pertinenza con il sostantivo *tooth*.

Relazione participiale

Anche questa relazione è tipica della categoria degli aggettivi, tipicamente che derivano da un verbo, ad esempio l'aggettivo *burned* deriva dal verbo *burn*.

Attributo

Attributo è il nome per cui uno o più aggettivi esprimono un valore, ad esempio il nome *weight* è un attributo a cui gli aggettivi *light* e *heavy* danno un valore.

Relazioni fra frasi

A differenza delle parole, per le frasi non esiste un insieme di relazioni semantiche ben definite in letteratura pertanto ci si è basati su un test sperimentale volto all'identificazione di frasi similari, considerate tali a seguito di valutazioni fornite da un gruppo selezionato di persone, da parte di una rete neurale [1].
In questo studio viene considerata come frase di partenza la seguente : *"a young woman in front of an old man"* a cui vengono poi applicate le seguenti variazioni:

[7] `https://en.wiktionary.org/wiki/pertainym`

- **Cambio di nome** : *"a young man in front of an old woman"*
- **Cambio di aggettivo** : *"an old woman in front of a young man"*
- **Cambio di preposizione** : *"a young woman behind an old man"*
- **Mantenimento di significato** : *"an old man behind a young woman"*

Come accennato poc'anzi questo tipo di test è stato proposto a 25 persone, ognuna delle quali ha valutato un insieme di 30 frasi, con l'obiettivo di individuare un *ranking* di similarità di ciascuna di esse messa a confronto con quella di partenza. I risultati hanno evidenziato il *Mantenimento di significato* come la trasformazione più similare seguita dal *Cambio di proposizione*, *Cambio di aggettivo* e infine il *Cambio di nome*.
Per completezza sono stati effettuati dei t-test tra i *ranking* delle diverse variazioni che hanno fornito un risultato statisticamente significativo per ognuna di esse. Sebbene nello studio citato finora i t-test abbiano marcato una differenza di similarità fra *Mantenimento di significato* e *Cambio di proposizione* si è deciso di considerarle come significative per i test effettuati in questo elaborato.

Relazioni fra Documenti

L'approccio utilizzato per individuare le relazioni fra documenti è simile a quello visto precedentemente per le frasi, non essendo presente in letteratura nessun riferimento specifico per questo tipo di struttura linguistica. In particolare un documento è costituito da un insieme di paragrafi, a loro volta costituito da un insieme di frasi. Perciò si sono delineate le seguenti variazioni semantiche come riferimento per i documenti:

- Cambio di ordine dei paragrafi.
- Aggiunta o eliminazione di paragrafi a quelli già esistenti.
- Modifica di parte del contenuto dei paragrafi, rimuovendo o aggiungendo frasi non inerenti al contesto.

Queste tre relazioni permettono la generazione di un insieme di documenti che possono essere utilizzati come metro di comparazione con il documento di partenza. Non essendo possibile in questa fase dello studio ottenere risultati comparativi sul calcolo della similarità fra la coppia $<$*documento originale*, *documento modificato*$>$ si rimanda all'applicazione e alla conseguente verifica della validità di queste variazioni nei capitoli successivi.

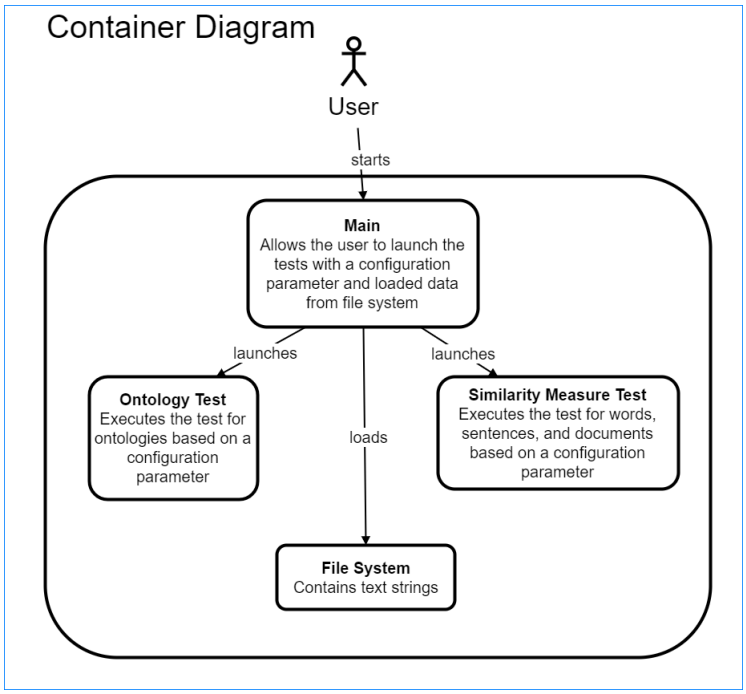
2.3 Progettazione dei test comparativi

Finora sono stati presentati due tipi ti approcci differenti fra loro, utili al raggiungimento di uno scopo comune: effettuare il calcolo di similarità fra parole, frasi e documenti. Durante la descrizione di ogni approccio, sono state messe in evidenza le proprietà semantiche che saranno utilizzate nei capitoli successivi. In particolare nel prossimo capitolo lo scopo è definire formalmente un insieme di strumenti utilizzati per manipolare le ontologie ed effettuare il calcolo del valore di similarità fra parole, frasi e documenti.
In particolare verranno trattati i seguenti strumenti:

- **Basati su ontologie** :
  - *Pellet*[8]
  - *Hermit*[9]
  - *JFact*[10]
- **Basati su misure di similarità** :
  - *DISCO*[11]
  - *Java String Similarity*[12]
  - *Simmetrics*[13]

Ad ognuno di essi verrà applicato un test che prenderà in considerazione i tipi di relazioni semantiche descritte finora al fine di selezionare quello migliore in termini di computabilità e prestazioni.
Di seguito viene indicato il diagramma di *Container* (basato sul modello C4 [2]) dei test, la cui progettazione sarà basata sulle due tipologie di approcci differenti.

[8] `https://github.com/stardog-union/pellet`
[9] `http://www.hermit-reasoner.com`
[10] `http://jfact.sourceforge.net`
[11] `http://www.linguatools.de/disco/disco-download_en.html`
[12] `https://github.com/tdebatty/java-string-similarity`
[13] `https://github.com/Simmetrics/simmetrics`



Da questo diagramma si evince la presenza di due tipi principali di test i quali utilizzano dati che avranno una rappresentazione differente a seconda dell'approccio. In particolare nel prossimo capitolo verranno introdotti i diagrammi che specializzano il *Container* il quale rappresenta il secondo livello di zoom nella terminologia C4.

Capitolo 3

Confronto tra strumenti software

3.1 Motori di inferenza semantica

Di seguito si riporta una descrizione formale dei *Reasoner* considerati in questo elaborato, confrontandoli per capire quanto può essere efficace l'utilizzo della logica di descrizione [2.1] per verificare la similarità fra singole parole, frasi e interi documenti.
Inoltre verranno descritte due ontologie, diverse fra loro, utilizzate successivamente per l'esecuzione dei test comparativi.
Gli aspetti importanti di questa ricerca sono:

- La Computabilità : definire il grado di espressività di questo tipo di rappresentazione.
- Le Prestazioni : scegliere, in base ai tipi di inferenza citati nella sezione 2.2, lo strumento più performante.

Gli strumenti che verranno considerati in questo elaborato sono:

- **Pellet** : rappresenta un reasoner[1] OWL-DL completo in grado di fornire un supporto adeguato per operare con *individuals* e tipi di dato definiti ad-hoc.
- **Hermit** : un reasoner[2] in grado di lavorare con ontologie definite attraverso il *Web Ontology Language* (OWL[3]).

[1] `https://github.com/stardog-union/pellet`
[2] `http://www.hermit-reasoner.com`
[3] `https://it.wikipedia.org/wiki/Web_Ontology_Language`

- **JFact** : deriva dal porting di JFact++ (un reasoner sviluppato per C++) in Java, è un reasoner[4] che opera su linguaggi di descrizione e implementa diverse tecniche di ottimizzazione.

Ognuno di questi strumenti implementa modi differenti di computare un'ontologia scritta attraverso OWL, pertanto risulta interessante cercare di comprendere quale di questi può essere sfruttato per rappresentare ed utilizzare il dominio proposto nella maniera più efficace.

Computabilità

Finora si è parlato della logica del primo ordine in relazione alla logica di descrizione ma esistono delle differenze fondamentali nei termini che sono stati utilizzati. I fondamenti della logica di descrizione mirano a comprendere, in senso metafisico, la natura dell'entità fondamentale *Thing* mentre la logica del primo ordine è un sistema formale che utilizza variabili quantificate per definire gli oggetti o *Thing*. La logica del primo ordine, conosciuta anche come logica dei predicati, si distingue dalla logica proposizionale che non utilizza variabili quantificate e per tale motivo consente una maggiore flessibilità nel definire *Thing*.
La logica di descrizione risulta un sottoinsieme della logica del primo ordine [5] le cui regole possono essere utilizzate per rappresentare un dominio in maniera strutturata e ben definita, in generale questa è più espressiva della logica proposizionale e più efficiente della logica del primo ordine a causa degli operatori descritti nel linguaggio.
Infine nella definizione e sviluppo di un'ontologia è possibile utilizzare le annotazioni sia appartenenti alla logica del primo ordine che alla logica di descrizione.

Prestazioni

Per quanto riguarda le prestazioni, si è deciso di implementare un test sperimentale che sfrutta le Java OWL API[6] utili per creare, manipolare e serializzare ontologie descritte attraverso OWL.
Grazie all'efficacia di questa libreria è stato possibile definire un insieme di funzionalità di base che poi vengono utilizzate pervasivamente dal test.
Lo scopo finale è quindi quello di verificare, data in pasto un'ontologia ai reasoner, le proprietà prestazionali dei tre reasoner descritti in precedenza in

[4] `http://jfact.sourceforge.net`
[5] `https://aic.ai.wu.ac.at/~polleres/teaching/ri2007/alejandro.pdf`
[6] `http://owlapi.sourceforge.net/`

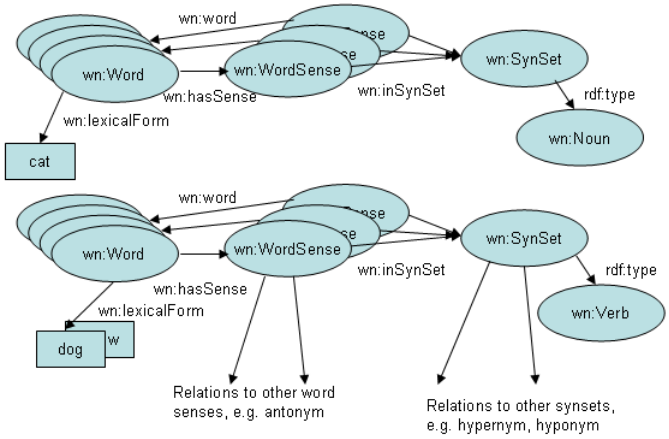
modo da selezionare quello più appropriato a fronte di ontologie di diversa natura e dimensione in input.

Ontologie utilizzate

Una caratteristica fondamentale che non può essere trascurata di questi test è derivante dal fatto che alcuni tipi di inferenza descritti precedentemente possono essere implementati in maniera generica ed eseguiti su una qualsiasi ontologia che viene fornita in input, mentre altri (ad esempio quello basato su relazioni, descritto nella sezione 2.1) richiedono necessariamente una comprensione di più basso livello basata sulla struttura di essa.
Risulta necessario quindi dare una descrizione formale dei concetti contenuti in una data ontologia:

WordNet

WordNet è un'ontologia utilizzata per rappresentare, trasformare e recuperare concetti appartenenti al linguaggio naturale. Può essere sfruttata per annotare e ritrovare concetti appartenenti a diversi domini come i catalogi di prodotti, metadati di foto o più semplicemente per realizzare dei vocabolari. La sua struttura[7] viene rappresentata nel diagramma seguente:



Il concetto fondamentale di *WordNet* è il **Synset** che rappresenta un insieme di sinonimi in cui sono contenute le parole che hanno un significato simile. Come ogni entità **Word**, anche i **Synset** hanno una proprietà che attribuisce ad esso un significato o **WordSense**.

[7] `https://www.w3.org/2001/sw/BestPractices/WNET/wordnet-sw-20040713.html`

WordNet definisce quindi le relazioni semantiche fra risorse in termini di relazioni linguistiche e concettuali fra i termini.

People

People è un'ontologia[8] che adotta come concetto fondamentale la **persona** e può essere utilizzata per descrivere alcuni aspetti fondamentali che rappresentano quest'entità. In particolare è possibile definire alcune proprietà della persona come *è giovane*/*è adulto*, *possiede un veicolo* oppure *ha un animale*. Questo modello permette anche di definire la tipologia del concetto di **animale** fino ad un livello di dettaglio tale da capire cosa esso può mangiare o meno.

Modalità e struttura dei test

I test, che è possibile visionare nel repository online[9], sono strutturati in tre progetti, uno per ogni motore di inferenza semantica considerato. All'interno di ogni progetto sono presenti i seguenti package:

- **[Tool Name]Main** : contiene un'unica classe Main la quale, messa in esecuzione, avvia una sessione di test.
- **[Tool Name]Test** : contiene essenzialmente tre file di cui uno rappresenta l'interfaccia del test il cui contratto viene utilizzato dal Main, una classe di test che implementa l'interfaccia relativa e una classe composta da soli metodi statici in cui vengono wrappate una serie di funzionalità relative all'utilizzo delle OWL API le quali vengono sfruttate pervasivamente dalla classe di test.

Allo scopo di rendere tutti i test più generali possibile, in rispetto dell'integrità concettuale, ho cercato di riutilizzare il più possibile il codice implementato nella classe di test.
L'implementazione del singolo test è stata strutturata in maniera tale da permettere all'osservatore dei risultati di poter configurare l'output a seconda dello specifico test che sta effettuando, in particolare sono state previste sette modalità di esecuzione:

- **PRINTCONTENT** : stampa in standard output tutte le informazioni contenute nell'ontologia considerata, in particolare permette all'osservatore di prendere visione di tutte le classi definite nell'ontologia

[8] `http://owl.man.ac.uk/2005/07/sssw/people.owl`
[9] `https://bitbucket.org/FATTORITEAM/semanticsvssimilarity`

con le relazioni **dirette** di *Subclassing* e delle istanze presenti in essa con le proprietà inferite dal reasoner.

- **CONSISTENCY** : verifica la consistenza dell'ontologia mettendo in evidenza tutti i concetti (classi) che non sono soddisfacibili se presenti.
- **SATISFIABILITY** : per ogni classe descritta nell'ontologia si controlla la soddisfacibilità che concerne un'analisi approfondita degli assiomi inferiti per ogni classe da parte del reasoner, se vengono trovate classi non soddisfacibili, allora se ne stampa la causa sotto forma di un'insieme di assiomi.
- **SUPERCLASS** : viene effettuata un'analisi approfondita sulla struttura gerarchica di tutte le istanze contenute nell'ontologia in cui vengono anche aggregate tutte le istanze appartenenti alla stessa classe.
- **QUERY** : siccome questo test risulta specifico per l'ontologia considerata, ho deciso di implementare in maniera completamente separata, in base appunto all'ontologia, questo tipo di test:

WordNet Query

Concerne l'esecuzione di tre query da me selezionate per questa specifica ontologia che tradotte in linguaggio naturale risultano:

- **Query 1** : dato in input un *WordSense* specificato in linguaggio naturale, recupero tutte le istanze (*individuals*) di classe **Word** che hanno lo stesso significato, specificato attraverso la proprietà *sense*. Tradotto in SPARQL:

```
SELECT ?wordName
WHERE
{
        ?word rdf:type :Word ;
          rdf:about ?wordName;
          :sense ?wsense.
        ?wsense rdf:type :WordSense ;
          rdf:about "Write Once Run EveryWhere"
}
```

- **Query 2** : per ogni istanza di classe *Word*, trovare tutte le istanze di classe *Synset* che condividono lo stesso *WordSense*. In altri termini si vuole sapere per ogni parola, se questa appartiene o

meno ad un insieme di sinonimi che ha lo stesso significato.
Tradotto in SPARQL:

```
SELECT ?wordName ?synsetName
WHERE
{
        ?word rdf:type :Word ;
              rdf:about ?wordName;
              :sense ?wsense.
        ?wsense rdf:type :WordSense ;
              rdf:about ?commonSense
        ?synset rdf:type :Synset ;
              rdf:about ?synsetName;
              :containsWordSense ?commonSense
}
```

– **Query 3** : data una parola in input, verifico se esiste un insieme di sinonimi a cui appartiene (anche se questa non è istanza di classe *Word*), a questo punto recupero tutte le istanze di classe *Word* che hanno lo stesso significato dell'insieme di sinonimi trovato (e che quindi dovrebbero farne parte).
Tradotto in SPARQL:

```
SELECT ?wordName
WHERE
{
        ?word rdf:type :Word ;
              rdf:about ?wordName;
              :sense ?wsense.
        ?wsense rdf:type :WordSense ;
              rdf:about ?commonSense
        ?synset rdf:type :AdverbSynset ;
              rdf:about ?synsetName;
              :containsWordSense ?commonSense
       FILTER ( REGEX(?synsetName, ?wordName ))
}
```

People Query

Anche in questo caso ho selezionato tre query che in linguaggio naturale risultano:

- **Query 1** : voglio conoscete tutte le istanze della classe *person* che hanno un animale, descritto dalla proprietà *has_pet*, la cui istanza è di tipo *cat* ed ha un nome che inizia per "T".
  Tradotto in SPARQL:

```
SELECT ?personName ?petName
WHERE
{
        ?person rdf:type :person ;
                rdfs:label ?personName;
                :has_pet ?pet.
        ?pet rdf:type :cat ;
                rdfs:label ?petName
        FILTER (STRSTARTS(?petName, "T"))
}
```

- **Query 2** : recupero tutte le istanze della classe *person*.
  Tradotto in SPARQL:

```
SELECT ?lcs
WHERE
{
       ?lcs rdfs:subClassOf ns:person
}
```

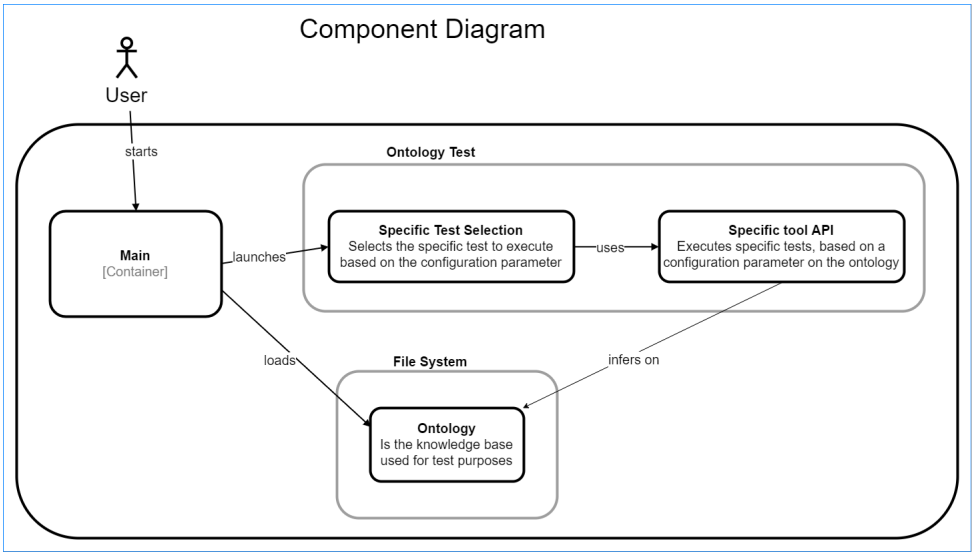
- **Query 3** : voglio conoscere quali tipi di animali piacciono (*likes*) ad ogni istanza di classe *person*. Tradotto in SPARQL:

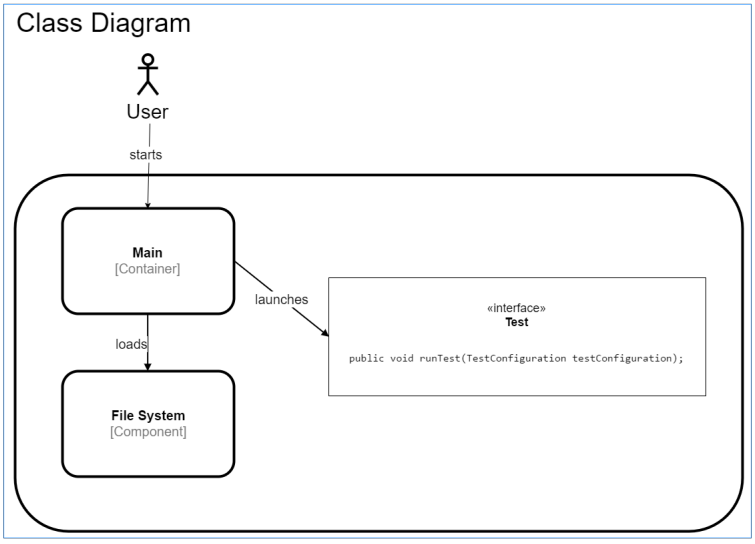
```
SELECT ?personName ?petName
WHERE
{
        ?person rdf:type :person ;
                rdfs:label ?personName;
                :likes ?pet.
        ?pet rdf:type :pet ;
                rdfs:label ?petName
}
```

- **SUBSUMPTION** : data un'ontologia O e due classi A e B, verifica se l'interpretazione di A è un sottoinsieme dell'interpretazione di B in O. Questa operazione viene effettuata per tutte le classi contenute in O.

- **ALL** : Esegue in sequenza tutte le modalità di test citate precedentemente.

Di seguito vengono mostrati i diagrammi C4 [2] di *Component* e *Class* che rappresentano rispettivamente il terzo e il quarto livello di zoom del diagramma *Container* presentato nella sezione 2.3 :





Ognuna delle modalità elencate in questa sezione rappresenta un valore del parametro di *TestConfiguration* presentato nel diagramma delle classi. Dal diagramma dei componenti si deduce la presenza di un'ontologia che viene inizialmente caricata dal main e utilizzata successivamente durante l'esecuzione del test.

3.2 Librerie per il calcolo di similarità

Di seguito vengono presentate e discusse tre librerie open source sviluppate in Java, ognuna delle quali implementa tecniche differenti per il calcolo della similarità fra singole parole, frasi e documenti.
In particolare per effettuare questo tipo di calcolo sono stati identificati i seguenti strumenti:

- **DISCO**[10]
- **Java String Similarity**[11]
- **Simmetrics**[12]

Per ognuno di essi viene fornita una descrizione formale utile per comprendere i concetti di base implementati dagli algoritmi appartenenti alle librerie e successivamente verrà effettuato un confronto per ognuno dei tre casi (parole, frasi e documenti) con lo scopo finale di ricavare una tassonomia.

DISCO

Rappresenta un'applicazione Java[13] che permette di calcolare la similarità semantica fra singole parole e frasi di lunghezza arbitraria. Questo calcolo è basato su un'analisi statistica effettuata su una grande collezione di testi fornita in input.
DISCO consiste in:

- **DISCO API** : Utili per fare query al database contenente le parole simili.
- **DISCO Builder** : Uno strumento aggiuntivo che permette di creare database di parole simili a partire da un testo.

In particolare le *DISCO API* supportano metodi per:

- Trovare le parole più **semanticamente** similari data una parola in input.
- Trovare il valore della **similarità semantica** tra due parole in input.

[10] disco-2.1 `http://www.linguatools.de/disco/disco-download_en.html`
[11] java-string-similarity-0.19 `https://github.com/tdebatty/java-string-similarity`
[12] guava-19.0, simmetrics 1.6.2, simmetrics-core `https://github.com/Simmetrics/simmetrics`
[13] `http://www.linguatools.de/disco/disco_en.html`

- Trovare la **collocazione** testuale di una parola in input, ad esempio la parola "beer" ha una collocazione formata dalle parole "keg brewed brewing lager Pilsener brewers brewery Budweiser mug ale"
- Computare la **similarità semantica** tra testi corti come frasi e paragrafi.
- Computare un **cluster di parole simili**
- Costruire un **grow set** ovvero un insieme contenente tutte le parole simili dato in input un insieme di parole.

Per utilizzare queste funzionalità è necessario effettuare una fase di computazione preliminare in cui viene organizzato il database di parole simili chiamato anche *wordspace*. DISCO 2.0 è in grado di utilizzare due tipi di *wordspace*:

- **COL** : contiene solo un insieme di **word vector**[14] ma non le parole più simili per ogni parola, se si utilizza un *wordspace* di questo tipo non è possibile utilizzare alcune delle API fornite ma si ha un vantaggio di prestazioni in quanto la fase di computazione preliminare è più veloce date le dimensioni ridotte di *wordspace* costruiti utilizzando questa rappresentazione.
- **SIM** : contiene sia un insieme di **word vector** che le parole più simili per ogni parola.

DISCO Builder può essere quindi utilizzato per generare un *wordspace* a partire dal corpo di un testo o da un file contenente una rappresentazione vettoriale di parole.

[14] è il tipo di rappresentazione utilizzata in DISCO per descrivere una distribuzione di parole [4]

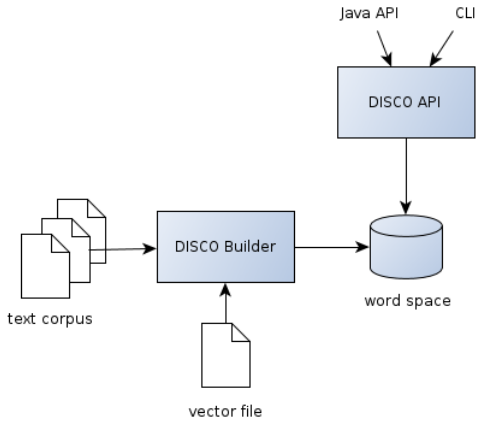


Figura 3.1: Disco diagram

Dopo aver effettuato un'analisi sulle API[15] fornite da questo strumento, è emerso che solo un insieme di esse può essere sfruttato per realizzare il calcolo della similarità fra parole, frasi e documenti, le quali sono :

- **semanticSimilarity**[16] : Date due parole in input e un *Similarity-Measure*[17] restituisce la similarità semantica fra di esse il cui calcolo è basato sugli insiemi di collocazione delle parole. È importante sottolineare che il valore restituito da questa API varia in base all'algoritmo utilizzato di cui ne viene fornita una descrizione in seguito.

- **secondOrderSimilarity**[18] : Computa la *similarità semantica* fra due parole in input calcolata in base all'insieme che rappresenta la distribuzione di probabilità delle parole più simili. Questa API funziona solamente se il *wordspace* utilizzato è di tipo SIM.

- **directedTextSimilarity**[19] : Computa la *similarità semantica* fra due testi corti forniti in input, uno definito come *"da controllare"* ed un altro chiamato *ipotesi*, per mezzo di un algoritmo di riconoscimento testuale[20]. Questa API utilizza l'algoritmo del *Coseno* per il calcolo

[15] `http://www.linguatools.de/disco/disco-api-2.1/index.html`

[16] `http://www.linguatools.de/disco/disco-api-2.1/de/linguatools/disco/DISCO.html`

[17] Rappresenta il tipo di algoritmo utilizzato, attualmente DISCO implementa due algoritmi: *COSINE* e *KOLB*

[18] `https://goo.gl/azC2Er`

[19] `http://www.linguatools.de/disco/disco-api-2.1/de/linguatools/disco/TextSimilarity.html`

[20] `http://u.cs.biu.ac.il/~nlp/RTE1/Proceedings/jijkoun_and_de_rijke.pdf`

della similarità sulle singole parole trovate nelle frasi passate in input e va utilizzata su un *wordspace* importato come file in formato vettoriale.

- **textSimilarity** : Computa la *similarità semantica* fra due testi in input come la media delle similarità semantiche dirette calcolate su entrambi.

- **compositionalSemanticSimilarity**[21] : Computa la *similarità semantica* fra due parole composte, frasi o paragrafi utilizzando la rappresentazione vettoriale delle parole che costituiscono le stringhe in input. Fra i parametri in input a questo metodo si hanno un *composition method* (specificato in seguito) e una serie di valori utili per la composizione dei vettori a seconda del metodo specificato.

In particolare quest'ultima funzionalità ammette una serie di *composition method*[22] :

- **ADDITION** : Addizione di vettori.

- **COMBINED** : È la combinazione di **ADDITION** e **MULTIPLICATION**.

- **DILATION** : Il vettore u viene dilatato verso la direzione del vettore v secondo l'equazione :

$$v' = (u * u)v + (lambda - 1)(u * v)u \tag{3.1}$$

  dove * è il prodotto scalare e *lambda* rappresenta uno dei parametri passati in input al metodo.

- **MULTIPLICATION** : Moltiplicazione fra entry indipendenti le une dalle altre.

- **SUBTRACTION** : Sottrazione di vettori.

[21] `http://www.linguatools.de/disco/disco-api-2.1/de/linguatools/disco/Compositionality.html`

[22] `http://www.linguatools.de/disco/disco-api-2.1/de/linguatools/disco/Compositionality.VectorCompositionMethod.html`

Algoritmi per la misura della similarità

DISCO implementa principalmente due algoritmi per il calcolo della similarità fra singole parole i quali vengono poi utilizzati ricorsivamente anche per il calcolo della similarità fra testi corti:

- **COSINE**[23] : Rappresenta una tecnica euristica per misurare la similarità fra due vettori calcolando il coseno fra di essi. Questo calcolo è descritto dalla formula:

$$similarity = cos(\theta) = \frac{A \cdot B}{\|A\| \, \|B\|} \tag{3.2}$$

  in cui A e B rappresentano due vettori numerici il cui contenuto, nel nostro caso, è la frequenza dei termini contenuti in un testo ossia il numero di volte che una determinata parola ricorre all'interno del testo. In base alla definizione del coseno, dati due vettori in input, il valore di similarità restituito sarà compreso fra [-1,1] dove -1 indica una corrispondenza esatta ma opposta (ad esempio COSINE("casa","asac") = -1) e +1 indica che i due vettori sono identici.

- **KOLB [3]** : Utilizza un approccio basato sul calcolo delle *co-occorrenze* all'interno di una finestra testuale che, a differenza di quello semantico, cerca una relazione sintattica chiamata *feature* data dalle coppie <word,window position>. Una volta ottenuta una rappresentazione di questo tipo viene calcolato il peso di ogni *feature* contenuta nella finestra considerata e successivamente si cercano tutte le parole che condividono il numero massimo di co-occorrenze comuni. In questo modo l'algoritmo è in grado di determinare la similarità fra parole confrontando le distribuzioni di similarità ricavate.

[23] `https://en.wikipedia.org/wiki/Cosine_similarity`

Java String Similarity

È una libreria Java[24] che implementa diverse tecniche e algoritmi per il calcolo della similarità e distanza fra stringhe.
Per il nostro caso di studio il calcolo della distanza non risulta particolarmente utile ma viene alcune volte utilizzato per il calcolo della similarità, pertanto mi soffermerò sugli algoritmi forniti dalla libreria per effettuare questo calcolo di cui viene data una descrizione di seguito :

- **Normalized Levenshtein** : Viene utilizzato il calcolo della distanza fra stringhe secondo *Levenshtein* per ricavare il valore di similarità calcolato come :

$$LevenshteinSimilarity = 1 - LevenshteinDistance \tag{3.3}$$

  Dove *Levenshtein Distance* è il numero minimo di modifiche sui singoli caratteri che sono necessari per rendere una parola uguale ad un'altra. Dato che questo valore è compreso fra [0,1], lo stesso intervallo sarà valido per *Levenshtein Similarity*.

- **Jaro-Winkler** : Computa la similarità fra due stringhe secondo un calcolo che considera la sostituzione di due caratteri vicini meno importante della stessa operazione eseguita su due caratteri lontani fra loro. Il valore restituito da questo algoritmo è compreso fra [0,1].

- **Cosine Similarity** : Computa la similarità fra due stringhe come specificato nella sottosezione 3.2 in cui i due vettori vengono calcolati trasformando le stringhe in input in *n-grammi*.

- **Jaccard Index** : Converte le stringhe in input in insiemi di *n-grammi*[25] chiamati a e b e li utilizza per calcolare la similarità secondo l'equazione :

$$J(a,b) = \frac{|a \cap b|}{|a \cup b|} \tag{3.4}$$

- **Sorensen-Dice** : è simile all'indice di Jaccard ma utilizza la seguente equazione per il calcolo della similarità:

$$SD(a,b) = 2 \cdot \frac{|a \cap b|}{|a| + |b|} \tag{3.5}$$

Fornisco ora una tassonomia preliminare di questi algoritmi con lo scopo di definire alcune proprietà fondamentali :

[24] `https://github.com/tdebatty/java-string-similarity`
[25] Rappresentano una sequenza di n caratteri.

Algorithm	**Normalized**	**Metric**	**Cost**
Normalized Levenshtein	yes	no	O(m*n)
Jaro Winkler	yes	no	O(m*n)
Cosine Similarity	yes	no	O(m+n)
Jaccard Index	yes	yes	O(m+n)
Sorensen-Dice	yes	no	O(m+n)

In cui **Normalized** identifica il fatto che il valore restituito dall'algoritmo è compreso fra [0,1] e **Metric** rappresenta una proprietà sul calcolo della distanza. Come è già stato specificato in precedenza alcuni di questi algoritmi utilizzano la distanza per ricavare il valore di similarità fra stringhe, essendo questi due valori correlati fra loro è possibile definire alcuni assiomi che dovrebbero essere rispettati, uno di questi è la proprietà geometrica definita come **triangle inequality**[26]. Se viene rispettato tale assioma, allora è possibile sfruttare una proprietà transitiva che facilita il calcolo della similarità. Nella tabella sopra mostrata gli algoritmi identificati tramite **Metric** = no non rispettano questa proprieà.

Ognuno degli algoritmi proposti possono essere utilizzati per il calcolo della similarità fra stringhe generiche, pertanto verrà fatta in seguito un'analisi di tipo qualitativo per capire quale di essi è preferibile utilizzare se in input si hanno parole, frasi o documenti.

Simmetrics

È una libreria Java che permette di calcolare la similarità fra stringhe utilizzando delle *metriche*. Ogni metrica rappresenta un algoritmo che sfrutta alcune proprietà *sintattiche* delle stringhe fornite in input per calcolare i valori di similarità e distanza.
Rispetto a *Java String Similarity* questa libreria fornisce una gamma più ampia di algoritmi con la possibilità, per alcuni di essi, di manipolare le stringhe in input prima di effettuare il calcolo della similarità, inoltre ognuno di essi rispetta la **triangle inequality** nel calcolo della distanza e restituisce un valore normalizzato in [0,1].
Gli algoritmi forniti dalla libreria sono :

- **Cosine Similarity** : Il calcolo della similarità avviene come specificato nella sezione 3.2.

[26] `http://www.scholarpedia.org/article/Similarity_measures`

- **BlockDistance** : Calcola la distanza[27] fra due insiemi a e b, costruiti a partire dalle stringhe in input, secondo la seguente equazione:

$$d(a,b) = \|a - b\| = \sum_{i=1}^{n} |a_i - b_i| \tag{3.6}$$

  la quale viene utilizzata per calcolare la similarità s come :

$$s(a,b) = 1 - d(a,b) \tag{3.7}$$

- **Damerau Levenshtein** : Come per il calcolo della distanza di Levenshtein viene misurato il numero minimo di operazioni necessarie per trasformare una stringa s1 in una s2. A differenza di esso però include, oltre alle operazioni di cancellazione, inserimento o sistituzione di un singolo carattere, le trasposizioni tra simboli adiacenti producendo una misurazione della distanza differente. Questa, come descritto nella sezione 3.2, viene poi utilizzata per il calcolo della similarità.

- **Sorensen-Dice** : Il calcolo della similarità avviene come specificato dall'equazione 3.5.

- **Euclidean Distance** : Calcola la similarità s fra due insiemi a e b, costruiti a partire dalle stringhe in input, secondo l'equazione :

$$s(a,b) = 1 - \frac{d(a,b)}{\sqrt{|a|^2 + |b|^2}} \tag{3.8}$$

  dove la distanza viene calcolata come :

$$d(a,b) = \|a - b\| \tag{3.9}$$

- **Generalized Jaccard** : Il calcolo della similarità avviene come specificato dall'equazione 3.4, a differenza del **Jaccard Index** in questo caso si tiene conto dell'occorrenza di una generica entry. Ad esempio se consideriamo le stringhe "hello word" e "hello word hello word" per **Jaccard Index** queste avranno similarità 1, mentre per **Generalized Jaccard** avranno similarità 0.5.

- **Jaccard** : Il calcolo della similarità avviene come specificato dall'equazione 3.4.

[27] `https://en.wikipedia.org/wiki/Taxicab_geometry`

- **Jaro-Winkler** : Il calcolo della similarità avviene come specificato nella sezione 3.2.

- **Levenshtein** : Il calcolo della similarità avviene come specificato dall'equazione 3.3.

- **Monge Elkan** : Effettua una fase preliminare di tokenizzazione delle stringhe in input da cui vengono prodotti due insiemi a e b. Per ognuno dei token ricavati l'algoritmo cerca la migliore corrispondenza ricavando così la similarità *s(a,b)* :

$$s(a,b) = \frac{1}{|a|}\sum_{i=1}^{|a|} \underset{j=1,|b|}{max}\, s'(a[i], b[i]) \tag{3.10}$$

  dove *s'* è una funzione per il calcolo della similarità interna, ad esempio Levenshtein.

- **Needleman Wunch** : Il calcolo della similarità avviene secondo l'algoritmo di Needleman-Wunsch[28].

- **Overlap Coefficient** : In questo caso il calcolo della similarità è definito come la dimensione dell'intersezione fra due insiemi a e b, ricavati dalle stringhe in input, divisa per la dimensione del più piccolo fra i due insiemi :

$$s(a,b) = \frac{|a \cap b|}{min(|a|, |b|)} \tag{3.11}$$

- **g-Gram Distance** : Utilizza l'algoritmo **Block Distance** applicando una suddivisione delle stringhe in input in **trigrammi**.

- **Simon White** : Equivale a **Sorensen-Dice** con la differenza che in questo caso vengono considerate le occorrenze di una generica entry.

- **Smith Waterman** : Risulta essere una variante di **Needleman Wunch** in cui nella fase di costruzione della **Score Matrix**[29] ogni cella contenente uno *score* negativo viene settata a 0.

- **Smith Waterman Gotoh** : Applica l'algoritmo di **Smith Waterman** la cui implementazione è stata ottimizzata secondo le tecniche proposte da *Osamu Gotoh* [4]

[28] `https://goo.gl/2YZnZy`
[29] `https://goo.gl/cEUuDm`

- **Longest Common Subsequence** : Calcola la similarità s fra due stringhe in input *s1* ed *s2* secondo l'equazione :

$$s(s1, s2) = \frac{|lcs(s1, s2)|}{max(|s1|, |s2|)} \tag{3.12}$$

dove la funzione *lcs*[30] e definita come :

$$LCS(X_i, Y_j) = \begin{cases} \emptyset & \text{if } i = 0 \text{ or } j = 0 \\ LCS(X_{i-1}, Y_{j-1}) \frown x_i & \text{if } x_i = y_j \\ \text{longest}(LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)) & \text{if } x_i \neq y_j \end{cases}$$

- **Longest Common Substring** : Effettua il calcolo della similarità s esattamente come descritto dall'equazione 3.12 con una differenza nel calcolo della funzione *lcs*[31].

Struttura dei Test

I test, che è possibile visionare nel repository online[32] sono strutturati in tre progetti, uno per ogni strumento considerato. All'interno di ogni progetto sono presenti i seguenti package :

- **[Tool Name]Main** : contiene un'unica classe Main la quale, messa in esecuzione, avvia una sessione di test.
- **[Tool Name]Test** : contiene tre file di cui uno rappresenta l'interfaccia del test il cui contratto viene utilizzato dal Main e una classe di test che implementa l'interfaccia relativa.
- **[Tool Name]TestData** : contiene una classe di utility che permette di wrappare i dati di input al test istanziati e passati dal main.
- **[Tool Name]WordSpecificTest** : contiene una serie di main, uno per ogni caso specifico considerato, che esegue il test per le singole parole.
- **[Tool Name]SentenceSpecificTest** : contiene una serie di main, uno per ogni caso specifico considerato, che esegue il test per le frasi.
- **[Tool Name]DocumentSpecificTest** : contiene una serie di main, uno per ogni caso specifico considerato, che esegue il test per i documenti.

[30] `https://en.wikipedia.org/wiki/Longest_common_subsequence_problem`
[31] `https://en.wikipedia.org/wiki/Longest_common_substring_problem`
[32] `https://bitbucket.org/FATTORITEAM/semanticsvssimilarity`

L'implementazione dei singoli test è stata strutturata in maniera tale da permettere all'osservatore dei risultati di poter configurare l'output a seconda dello specifica test che sta effettuando, in particolare sono state previste quattro modalità di esecuzione:

- **WORDSIMILARITY** : Effettua il calcolo della similarità per ogni coppia di parole presenti nelle due liste contenute nei *TestData* stampandone il risultato in standard output.
- **SENTENCESIMILARITY** : Effettua il calcolo della similarità per ogni coppia di frasi presenti nelle due liste contenute nei *TestData* stampandone il risultato in standard output.
- **DOCUMENTSIMILARITY** : Effettua il calcolo della similarità per ogni coppia di documenti presenti nelle due liste contenute nei *TestData* stampandone il risultato in standard output. Questa modalità non è prevista nel test concepito per lo strumento **DISCO**.
- **ALL** : Esegue in sequenza tutte le modalità di test citate precedentemente.
- **[Specific Name]TEST** : Consiste in un test specifico che esegue un particolare insieme di algoritmi volta a fornire dei risultati utili per il calcolo della soglia di similarità.

Ognuna di queste modalità effettua il calcolo della similarità per ogni algoritmo presente negli strumenti considerati in modo da ottenere una visione d'insieme che permette di distillare, a seconda dell'input, la validità e consistenza del valore restituito.

Come per i motori di inferenza semantica, anche in questo caso è stato effettuato uno zooming sul diagramma C4 [2] di *Container* riportato nella sezione 2.3. Di seguito vengono mostrati i diagrammi di *Component* e *Class* per i test relativi alle librerie per il calcolo di similarità :

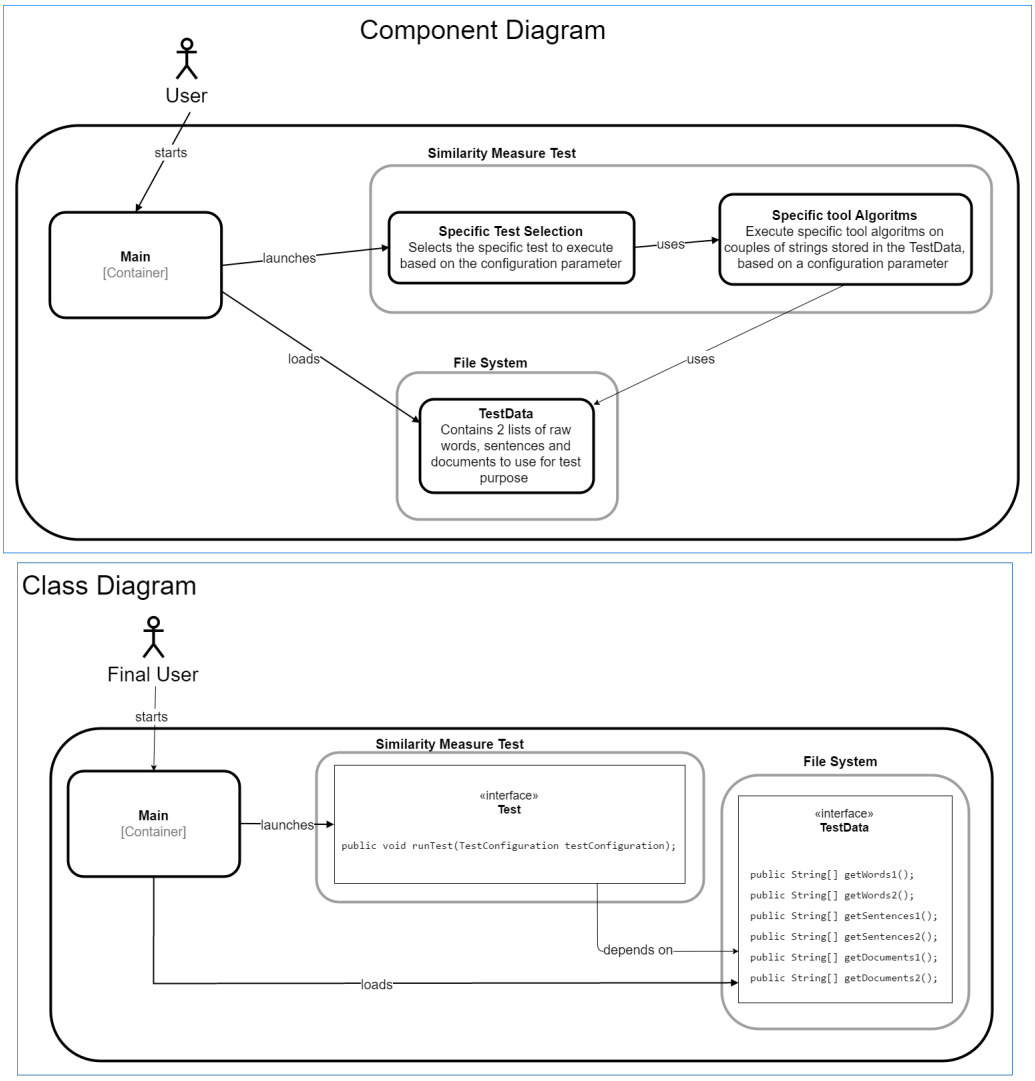


Figura 3.2: Diagrammi C4 di *Component* e *Class*

Dal diagramma dei componenti si deduce la presenza dell'entità *TestData* appositamente introdotta per facilitare il passaggio dei dati elaborati dal *Test*.

Dati utilizzati

In riferimento alla sezione 2.2 sono stati selezionati una serie di dati da considerare per capire quali algoritmi forniscono un risultato significativo nel calcolo della similarità. In particolare per le parole sono state selezionate 4 coppie di campioni per ogni tipo di relazione analizzata nella sezione 2.2

da cui ci si aspetta un valore di similarità alto e 4 coppie di parole che non condividono nessun tipo di relazione e pertanto ci si aspetta un valore di similarità basso.
Per le frasi invece sono stati sfruttati i risultati proposti da *Samuel J. Gershman* [1] per selezionare alcuni campioni significativi che riportano le variazioni citate nella sezione 2.2.
In aggiunta a queste, in cui ci si aspetta un valore di similarità alto, viene effettuato lo stesso calcolo su 4 coppie di frasi che non centrano nulla le une con le altre, ad esempio <*"small bug on a large flower", "a black cat in front of a white dog"*>.
Per i documenti invece sono stati selezionati due campioni di base contenenti numeri differenti di paragrafi in maniera tale da consentire un'analisi sulle performance. Partendo da questi sono stati generati due insiemi di documenti derivati seguento le modifiche citate nella sezione 2.2. L'obiettivo in questo caso è quello di effettuare il calcolo della similarità fra il documento originale e ognuna di queste pertubazioni per verificarne il risultato in base al valore atteso (similarità alta). Per ottenere un test completo viene effettuato anche il calcolo della similarità fra i due campioni di partenza, in cui ci si aspetta un risultato pressochè nullo.

3.3 Discussione dei risultati

Motori di inferenza semantica

Una volta implementati i test per ognuno dei tre strumenti presentati nella sezione 3.1, si è deciso di effettuare un test sperimentale in cui vengono fatte eseguire 10 volte le rispettive classi Main di ogni progetto e vengono considerati i tempi totali di esecuzione del test a cui prima viene data in pasto l'ontologia *WordNet* e successivamente *People*.
Si vuole specificare che l'ontologia *People* contiene al suo interno la definizione di 60 *concepts* e 30 *individuals* mentre *WordNet* contiene solamente la definizione di 22 *concepts*. Per tale motivo è stata prevista una fase preliminare, durante l'inizializzazione del test per *WordNet*, in cui vengono sfruttate alcune primitive appartenenti alle OWL API per creare alcuni *individuals* e *properties* che vengono poi sfruttati durante il test. Al termine di questa operazione l'ontologia *WordNet* conterrà 15 *individuals*.
Durante una prima sessione in cui i test sono stati configurati in modalità **ALL**, i risultati riscontrati vengono mostrati in figura 3.3:

Figura 3.3: Esecuzione del test in modalità **ALL**.

In seguito ad una sessione più accurata è stato riscontrato un problema di performance durante l'esecuzione del test **PRINTCONTENT** in cui JFact impiega effettivamente un tempo molto superiore rispetto agli altri due strumenti.
Più in dettaglio il tempo in eccesso è dovuto principalmente all'utilizzo di una primitiva definita nel reasoner *JFact*, la *getObjectPropertyValues* che restituisce tutti i valori di una proprietà dati la definizione della proprietà stessa e l'individuo interrogato. È stato possibile raggiungere questo risultato effettuando un'analisi accurata del test **PRINTCONTENT** di cui i risultati vengono mostrati in figura 3.4:

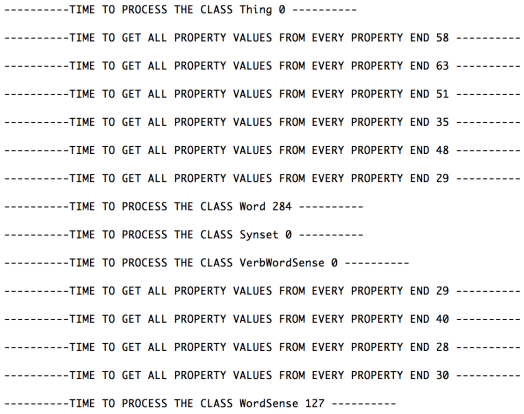


Figura 3.4: Analisi di performance su **PRINTCONTENT**.

Nella figura 3.4 viene mostrato un esempio di esecuzione in cui per ogni *individual* presente nell'ontologia (*WordNet* in questo caso) è stato stampato il tempo necessario per recuperare i valori di tutte le proprietà definite nel modello.

Eliminando l'esecuzione del test **PRINTCONTENT** nella modalità **ALL**, il risultato risulta essere:

Figura 3.5: Analisi di performance senza **PRINTCONTENT**.

Selezione del motore di inferenza semantica

Tutti e tre i motori di inferenza semantica si basano su implementazioni differenti delle interfacce fornite dalle OWL API, pertanto sono equivalenti in termini di computabilità ed espressività.
Dai risultati ottenuti si evince che *Pellet* risulta essere lo strumento più performante in termini di prestazioni anche se di poco rispetto a JFact se consideriamo la seconda sessione di test. Il reasoner *Pellet*, per poter essere utilizzato, deve importare il framework *Jena* mentre *JFact* e *Hermit* possono essere eseguiti in maniera indipendente importando semplicemente le librerie che forniscono le funzionalità specifiche dei reasoner.
Inoltre da non sottovalutare è la problematica che concerne la sessione iniziale di test in cui *JFact* impiega un tempo incredibilmente superiore durante l'esecuzione.
Per queste ragioni si ritiene preferibile l'utilizzo dello strumento **Hermit**.
Per quanto concerne il dominio applicativo del calcolo della similarità fra parole, frasi e documenti si ritiene appropriato l'utilizzo dell'ontologia *WordNet* in cui effettivamente vengono specificati alcuni concetti utili per essere sfruttati in questo dominio.
In particolare nel test **QUERY** relativo a *WordNet* sono stati considerati 3 casi che possono essere sfruttati per ricavare la similarità fra parole, sarebbe

necessario introdurre alcune tecniche per estendere (in maniera ricorsiva per esempio) questi casi per il calcolo della similatità fra frasi e documenti.

Estensione del test specifico per Hermit

Una volta selezionato lo strumento sono stati effettuati una serie di test specifici per verificare le prestazioni di alcuni tipi di inferenze. In questa fase ci si è ispirati al dominio applicativo citato in precedenza in modo da confermare la possibilità di utilizzo di queste tecnologie per il calcolo della similarità fra parole, frasi e documenti.
I tipi di inferenza più rilevanti in questo caso sono:

- **Common Ancestor**: nella teoria dei grafi, il *Common Ancestor* di due nodi v e w in un albero o grafo diretto aciclico è un nodo che ha v e w come discendenti. Questa relazione nel nostro caso può essere sfruttata per capire se due *individuals* possiedono una classe *"parente"* in comune. Una specializzazione di questo concetto è il **Lowest Common Ancestor**[33] il cui significato è differente dal precedente per il fatto che il nodo ricercato deve essere più vicino possibile alle foglie dell'albero.

- **Le istanze**: la definizione di questa relazione semantica equivale a quella fornita nella sezione 2.1.

- **Query Specifica su WordNet**: per definire una relazione di similarità fra singole parole è risultato particolarmente significativo in questi test comprendere un caso di studio già considerato nei test globali. Il caso in questione è rappresentato dalla **Query 2** le cui specifiche sono state riportate nella sezione 3.1. Questa interrogazione mette in evidenza la relazione di similarità semantica fra singole parole nel tentativo di ricercare un insieme composto da parole che condividono lo stesso significato di quella fornita in input, perlustrando le informazioni contenute nella base di conoscenza.

I primi due tipi di inferenze considerate possono essere applicate a qualsiasi base di conoscenza mentre la terza rappresenta un test specifico su *WordNet* la cui struttura, come descritto nella sezione 3.1, specifica i concetti necessari per rappresentare insiemi di parole e singole parole. Si vuole specificare che in questi test sulle prestazioni non vengono considerate frasi e documenti.

[33] `https://en.wikipedia.org/wiki/Lowest_common_ancestor`

Implementazione

In base al principio di riutilizzo del codice si è deciso di implementare questo test specifico estendendo la struttura del test già implementato in precedenza. È stata introdotta una classe *Main* aggiuntiva inserita nell'apposito package (specificato nella sezione 3.1):

- **HermitSpecificTestMain** : classe che implementa l'inizializzazione del test e di tutte le entità necessarie per l'esecuzione. In primo luogo viene effettuato il test sull'ontologia *WordNet* in cui una fase preliminare di inizializzazione ha lo scopo di creare ed inserire gli *individuals* nella base di conoscenza. I dati in questo caso provengono dal database reale di *WordNet 3.1*[34] e vengono letti attraverso l'utilizzo di una libreria Java chiamata *JWI*[35]. Per ottenere un test veritiero si è deciso di introdurre nell'ontologia 100 *individuals* di classe *Synset* e relativi *WordSense*, 50 *individuals* di classe *Word* che condividono lo stesso significato degli insiemi inseriti e 50 *individuals* di classe *Word* che non necessariamente appartengono a un insieme di sinonimi.
  Successivamente viene effettuato il test sull'ontologia *People* che, come specificato nella sezione 3.3, contiene la definizione di 60 *concepts* e 30 *individuals*.

Oltre a questa sono state aggiunte quattro nuove modalità di test al parametro *TestConfiguration* la cui implementazione è stata inserita appositamente nella classe **HermitTest**:

- **COMMON ANCESTOR** : questa modalità implementa il primo tipo di inferenza specificato in precedenza, in questo caso però non si ricerca il nodo più profondo. Questa operazione viene semplificata dal fatto che le API della classe *OWLReasoner* forniscono un metodo *getTypes*[36] in grado di recuperare tutti i concetti che hanno una qualsiasi relazione di parentela (padre, nonno ecc...) con l'*individual* fornito in input. È necessario specificare che queste informazioni vengono fornite sotto forma di insieme in cui si perde il livello di parentela.

- **LOWEST COMMON ANCESTOR** : questa modalità implementa il primo tipo di inferenza specificato in precedenza in cui si ricerca il nodo più profondo. Per i motivi specificati nella descrizione della modalità precedente non è stato possibile utilizzare il metodo *getTypes* per

[34] `https://wordnet.princeton.edu/wordnet/download/current-version/`

[35] `http://projects.csail.mit.edu/jwi/`

[36] `http://owlapi.sourceforge.net/javadoc/org/semanticweb/owlapi/reasoner/OWLReasoner.html`

recuperare tutte le relazioni di parentela ma si è deciso di utilizzarlo unicamente per ottenere quelle **dirette**, ovvero l'insieme di classi che rappresentano il tipo dell'*individual* specificato in input.
Di seguito è riportato lo pseudocodice relativo all'algoritmo implementato:

```
Node lowestCommonAncestor (individual1, individual2,
                                              resoner){
  front = reasoner.getTypes(individual1, true);
  individual2Superclasses = this.reasoner
                                .getTypes(ind2, false);
  while(front.count > 0){
    parentClass = front.get(0);
    for(index = 0; index < individual2Superclasses
                                          .count; index++){
      if(parentClass == individual2Superclasses
                .get(index) && parentClass != root){
        return parentClass;
      }
    }
    front.addAll(reasoner
              .getSuperClasses(parentClass, true));
    front.remove(parentClass);
  }
  return null;
}
```

In questo caso si è deciso di adottare una logica *breadth first* in cui inizialmente vengono aggiunti alla frontiera i nodi che rappresentano i tipi associati al primo individuo. Ad ogni iterazione si preleva il primo nodo dalla frontiera e si cerca una corrispondenza fra tutte le classi "parenti" del secondo individuo, se non c'è alcuna corrispondenza si aggiorna la frontiera inserendo i nodi che corrispondo al prossimo livello gerarchico e si passa alla prossima iterazione.

- **INSTANCE** : in questo caso si vuole trovare l'insieme di classi che corrisponde al tipo dell'*individual* fornito in input. La soluzione adottata prevede l'utilizzo di *getTypes*, come visto in precedenza, per recuperare le classi dirette.

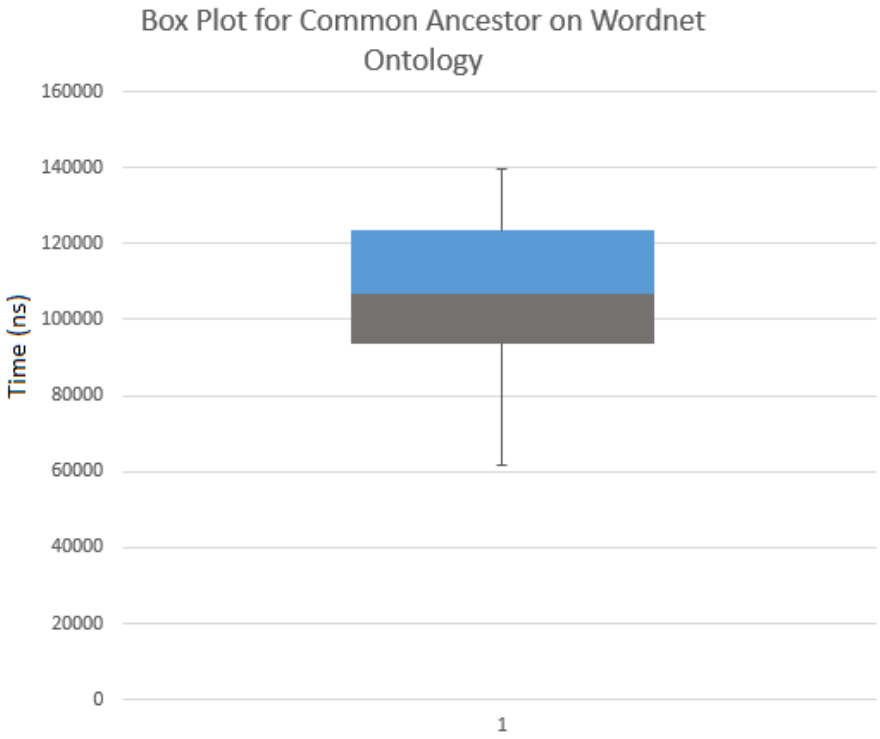
- **WORDNETQUERY** : l'implementazione di questo tipo di inferenza equivale a quella fornita per il test globale.

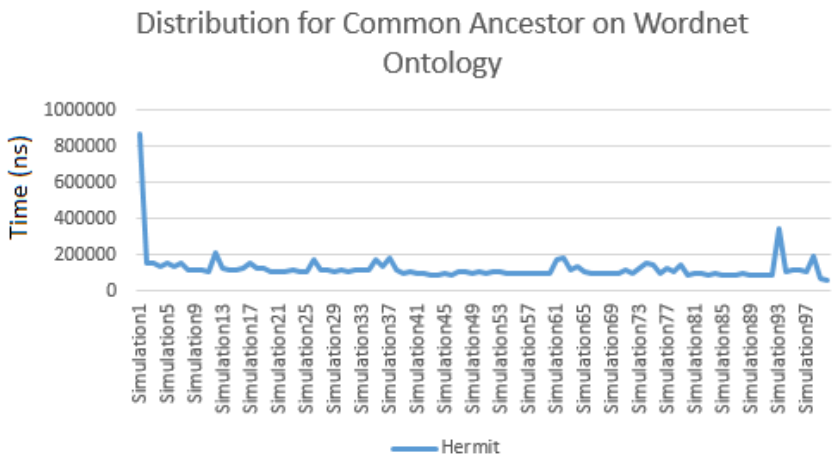
Risultati

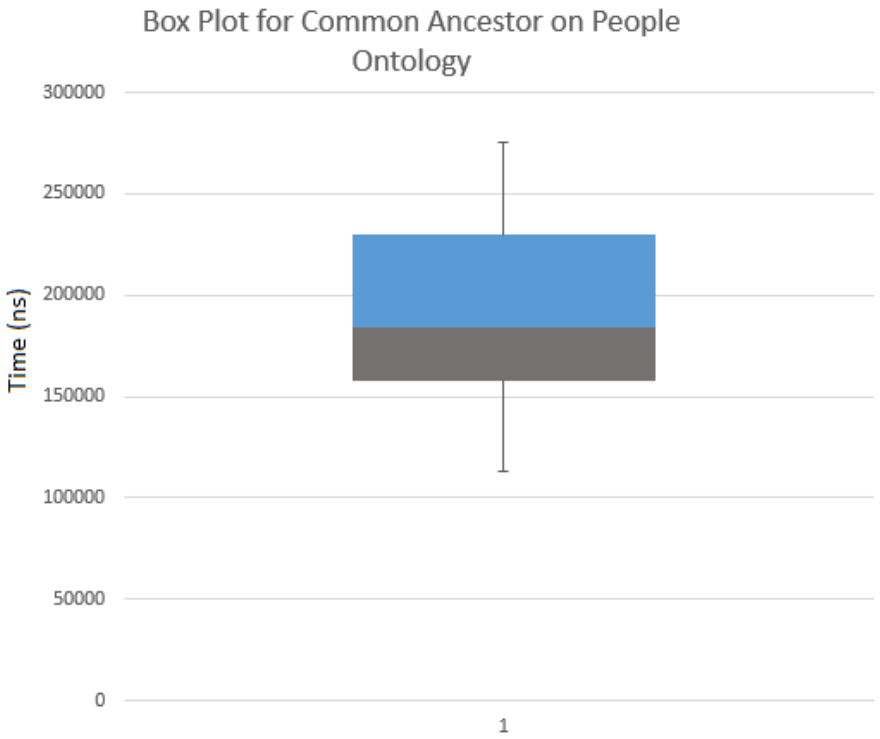
Una volta implementate tutte le modalità descritte in precedenza si è deciso di testarle singolarmente. Ogni test è stato eseguito per un totale di 100 iterazioni dalle quali è stato estratto il tempo di esecuzione calcolato in nanosecondi.

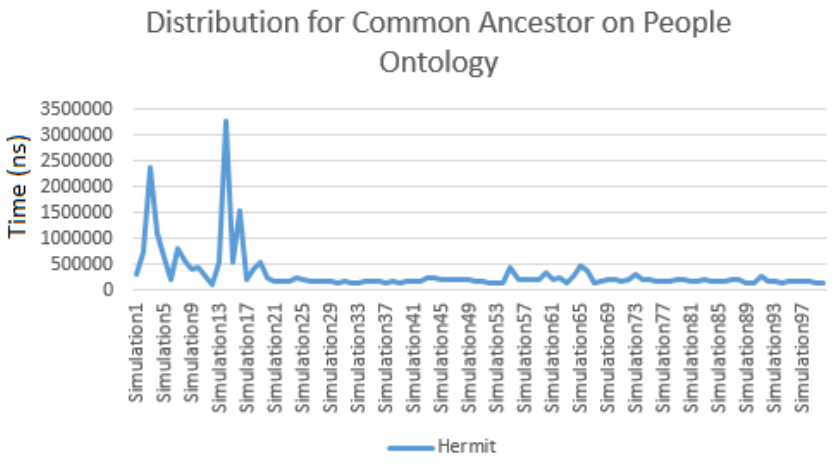
Si riportano di seguito i risultati ottenuti per ogni test:

- **COMMON ANCESTOR** :

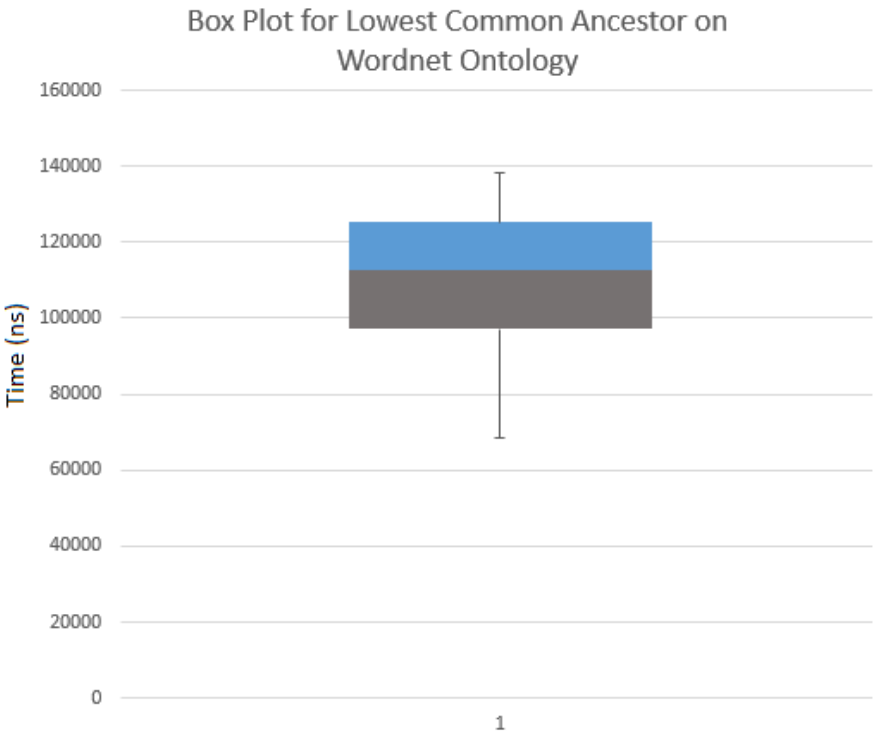


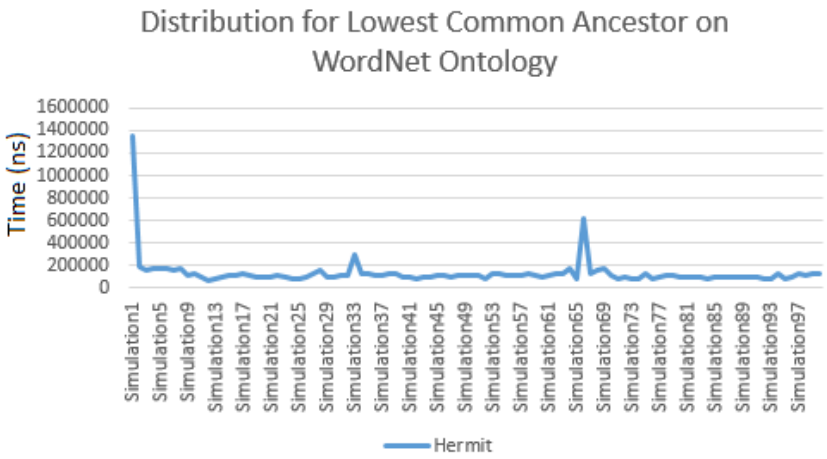


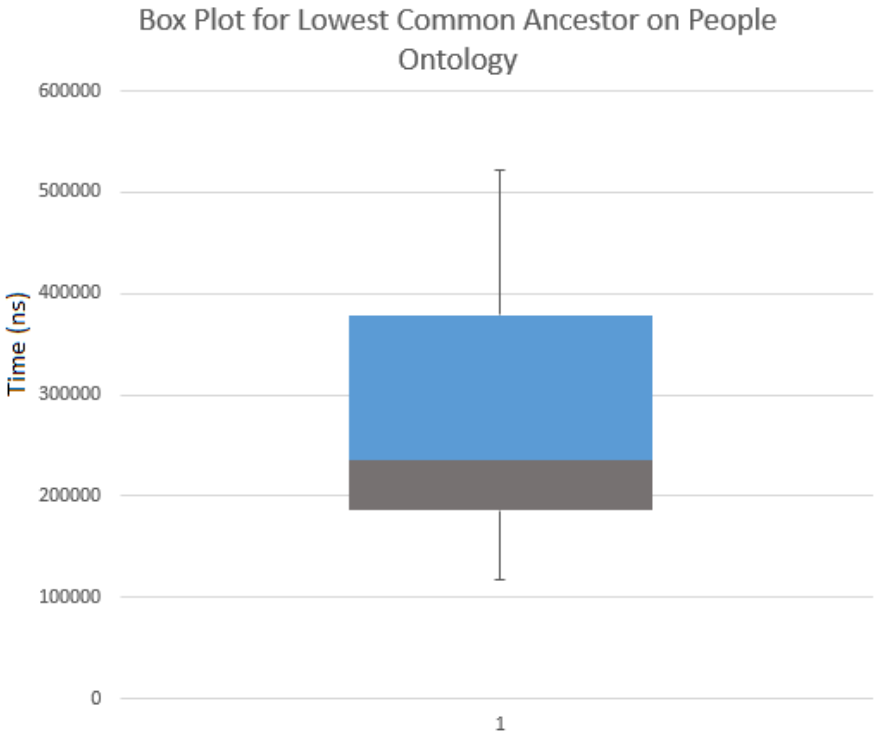

Box Plot for Common Ancestor on People Ontology
Time (ns)
300000
250000
200000
150000
100000
50000
0
1

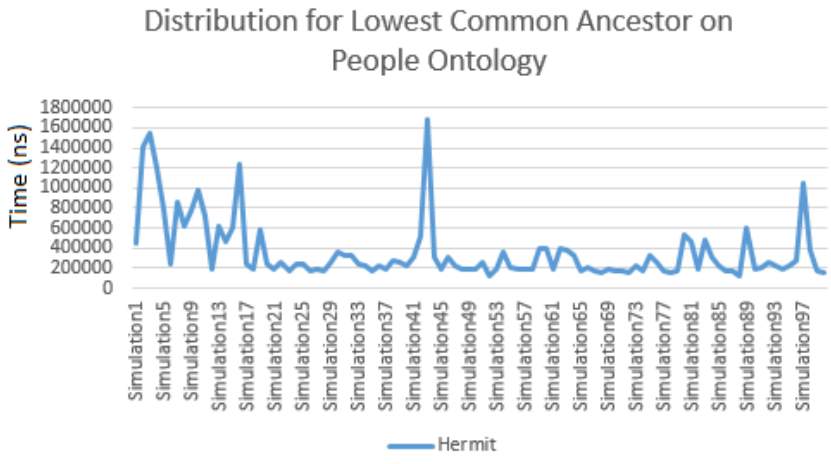

Distribution for Common Ancestor on People Ontology
Time (ns)
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
Simulation1
Simulation5
Simulation9
Simulation13
Simulation17
Simulation21
Simulation25
Simulation29
Simulation33
Simulation37
Simulation41
Simulation45
Simulation49
Simulation53
Simulation57
Simulation61
Simulation65
Simulation69
Simulation73
Simulation77
Simulation81
Simulation85
Simulation89
Simulation93
Simulation97
Hermit

- **LOWEST COMMON ANCESTOR** :

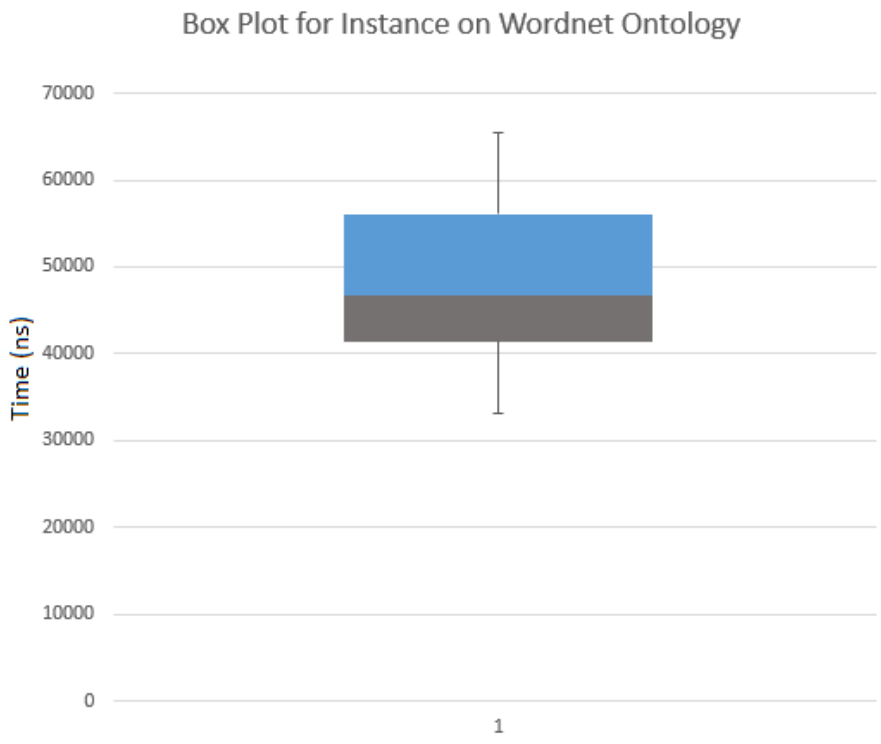


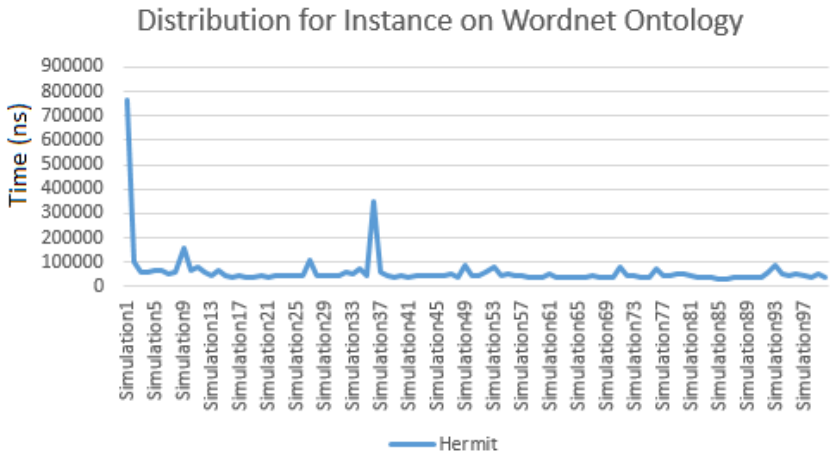


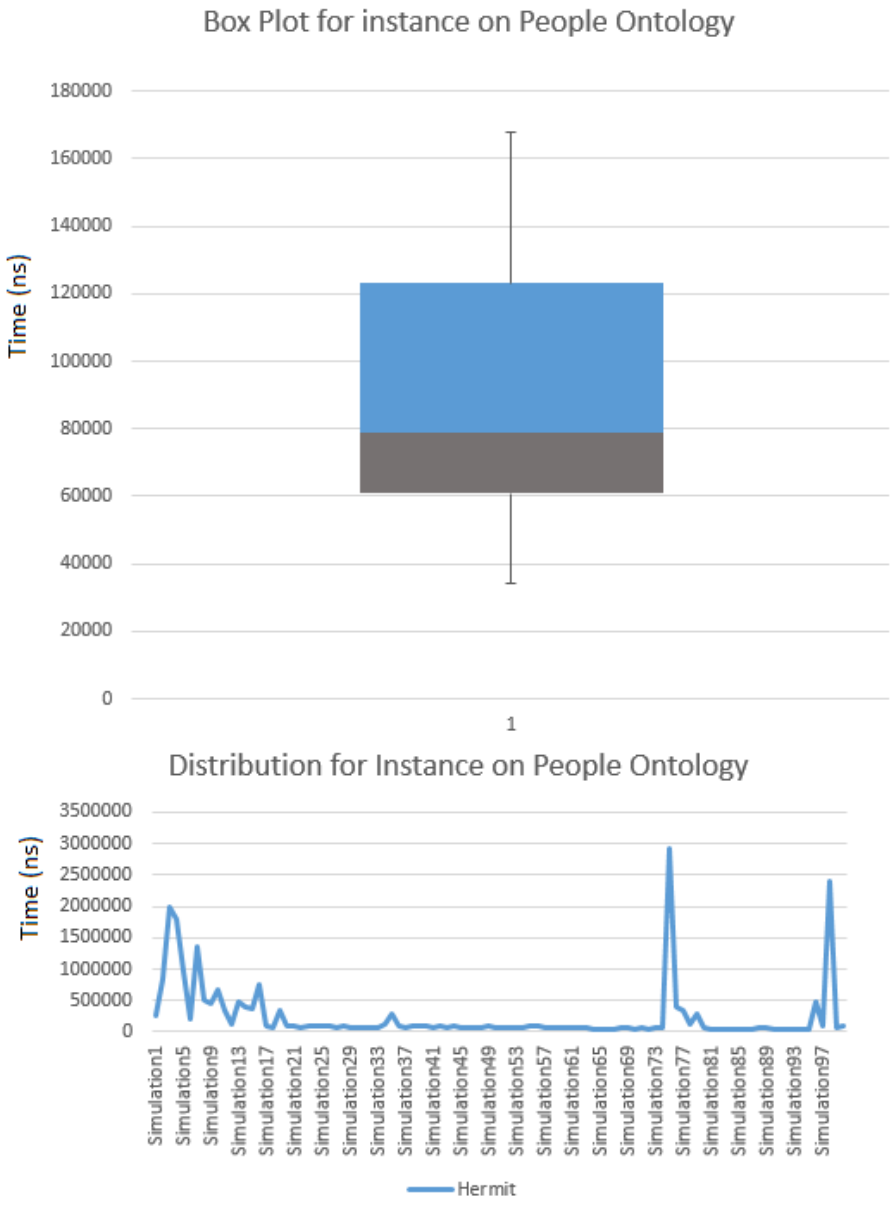

Box Plot for Lowest Common Ancestor on People Ontology
Time (ns)
600000
500000
400000
300000
200000
100000
0
1


Distribution for Lowest Common Ancestor on People Ontology
Time (ns)
1800000
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
Simulation1
Simulation5
Simulation9
Simulation13
Simulation17
Simulation21
Simulation25
Simulation29
Simulation33
Simulation37
Simulation41
Simulation45
Simulation49
Simulation53
Simulation57
Simulation61
Simulation65
Simulation69
Simulation73
Simulation77
Simulation81
Simulation85
Simulation89
Simulation93
Simulation97
Hermit

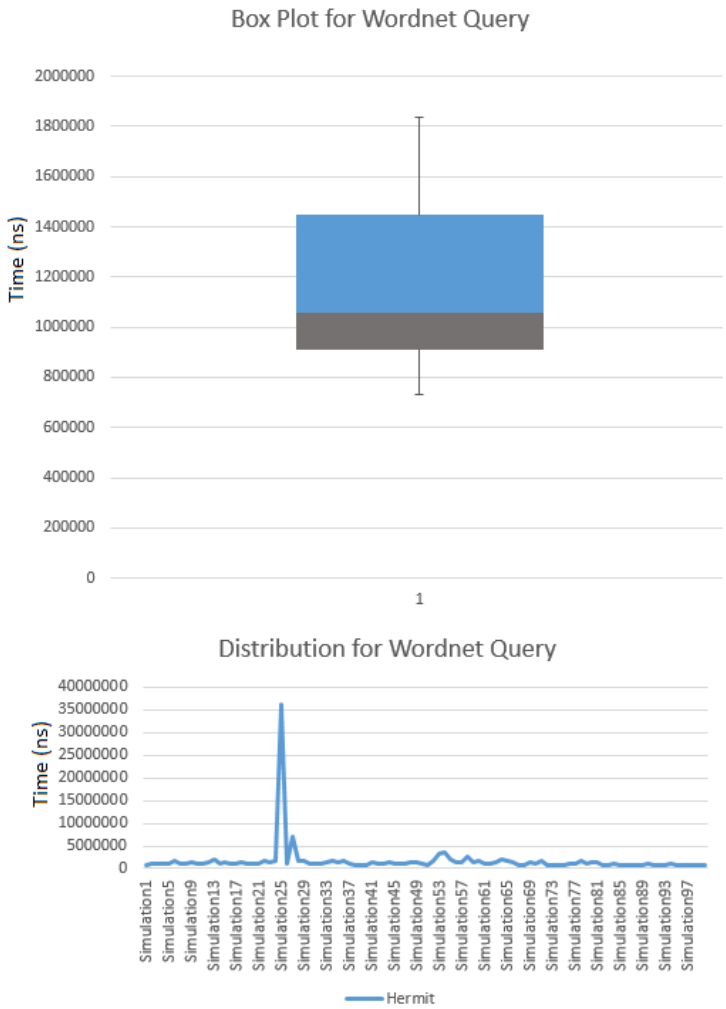
- **INSTANCE** :


Box Plot for Instance on Wordnet Ontology
70000
60000
50000
40000
30000
20000
10000
0
Time (ns)
1


Distribution for Instance on Wordnet Ontology
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
Time (ns)
Simulation1
Simulation5
Simulation9
Simulation13
Simulation17
Simulation21
Simulation25
Simulation29
Simulation33
Simulation37
Simulation41
Simulation45
Simulation49
Simulation53
Simulation57
Simulation61
Simulation65
Simulation69
Simulation73
Simulation77
Simulation81
Simulation85
Simulation89
Simulation93
Simulation97
Hermit


Box Plot for instance on People Ontology
Time (ns)
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
1
Distribution for Instance on People Ontology
Time (ns)
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
Simulation1
Simulation5
Simulation9
Simulation13
Simulation17
Simulation21
Simulation25
Simulation29
Simulation33
Simulation37
Simulation41
Simulation45
Simulation49
Simulation53
Simulation57
Simulation61
Simulation65
Simulation69
Simulation73
Simulation77
Simulation81
Simulation85
Simulation89
Simulation93
Simulation97
Hermit

- **WORDNETQUERY** :



Dai grafici appare evidente che i risultati dei test hanno una deviazione standard bassa, questa è un'ulteriore conferma del fatto che *Hermit* risulta essere stabile e perfomante.

Considerazioni sui Risultati

Consideriamo uno scenario di sistemi distribuiti in cui ogni nodo ha il compito di aggregare un elevato numero di parole in poco tempo posizionando ogni singola parola in insiemi separati, ciascuno dei quali contiene tutte le parole che condividono un concetto. Per esempio potrebbe essere analizzato un

insieme di commenti ricavati dal noto social network *Facebook* per individuare tutte le parole che sono in relazione con la parola *"barca"*. Prendendo come riferimento il tipo di inferenza **Common Ancestor** è possibile stimare il tempo impiegato mediamente per aggregare 10^5 parole sarà:

$$T = 110000 \times 10^5 = 11 \times 10^9 ns = 11s \tag{3.13}$$

dove 110000 equivale al valore di mediana in nanosecondi rappresentata nel grafico relativo al box-plot di **Common Ancestor** per l'ontologia **WordNet**.
Stimando che mediamente una pagina di un libro contiene 300 parole[37] si deduce che il calcolo descritto in precedenza equivale all'aggregazione delle parole contenute in un libro di circa 300 pagine.
Un altro esempio di questo tipo consiste nel considerare che in una frase, specificata nella lingua inglese, sono presenti circa 15-20 parole[38]. In questo caso è possibile effettuare una stima analoga del tempo impiegato in media per aggregare le parole più significative contenute in una singola frase. Nel caso peggiore possiamo pensare che tutte le parole contenute in essa siano significative, pertanto:

$$T = 110000 \times 20 = 22 \times 10^5 ns = 2,2ms \tag{3.14}$$

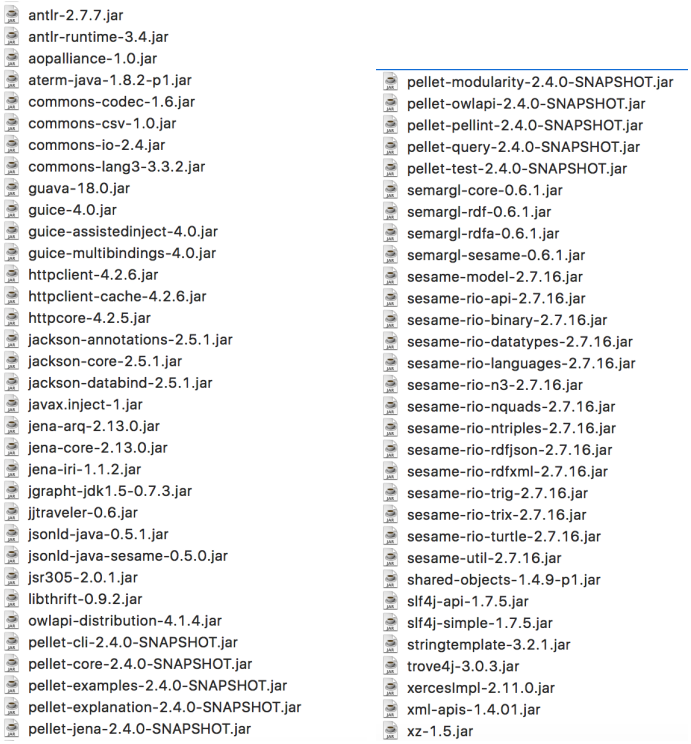
Attraverso questi esempi si è dimostrata l'efficacia dello strumento *Hermit* in un contesto applicativo in cui è necessario processare enormi quantità di dati in breve tempo.

[37] `https://goo.gl/BzCixg`
[38] `https://strainindex.wordpress.com/2008/07/28/the-average-sentence-length/`

Librerie necessarie per riprodurre i test

Pellet

richiede la presenza del framework *Jena* per poter operare e di tutta una serie di librerie fra cui le owl-api. Le librerie utilizzare nei test presentati precedentemente sono state estratte dal progetto *Maven* di *Pellet*[39] i cui riferimenti sono stati poi aggiunti al progetto *Java* di Test:



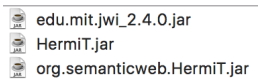
JFact

richiede le librerie contenenti le funzionalità del reasoner in aggiunta alle owl-api:

[39] `https://github.com/stardog-union/pellet`

Hermit

richiede le librerie contenenti le funzionalità del reasoner e *JWI* utilizzata per leggere i dati contenuti nel database *Wordnet*:



Librerie per il calcolo di similarità

L'esecuzione dei test per le librerie adebite al calcolo di similarità ha inizialmente l'obiettivo di generare una tassonomia degli algoritmi presentati nel capitolo precedente in modo tale da capire quali di essi possono essere applicati a parole, frasi e documenti.

Tassonomia degli Algoritmi

Per eliminare a priori tutti quegli algoritmi che non restituiscono un valore di similarità che si avvicina al valore reale (1 se le stringhe sono uguali e 0 se sono completamente diverse) è stato effettuato un test in cui vengono messi in esecuzione tutti gli algoritmi di ciascun tool con i dati descritti nella sezione 3.2. L'aggregazione dei risultati sotto forma tabellare è stata effettuata considerando una soglia di similarità a priori:

$$priorThreshold = 0.5 \tag{3.15}$$

Se il valore di similarità restituito da ciascun algoritmo è maggiore o uguale a *priorThreshold* allora nella cella corrispondente a <Algoritmo, Relazione o Perturbazione> sarà presente il valore *"yes"*, *"no"* altrimenti. Nel caso in cui le stringhe in input non condividano alcuna relazione semantica nella cella <Algoritmo, Relazione o Perturbazione> sarà presente *"yes"* se il valore restituito dall'algoritmo è minore o uguale a *priorThreshold*, *"no"* altrimenti.

I risultati di questa aggregazione sono visibili di seguito:

Parole

Algoritmo/Relazione	Sinonimia	Antinomia	Iponimia/Iperonimia	Iponimia/Iponimia	Meronimia	Participiale	Parole differenti
DISCO							
semanticSimilarity.COSINE Workspace Sim	no	no	no	no	no	no	yes
semanticSimilarity.COSINE Workspace Sim word2vec	no	no	no	yes	yes	yes	yes
semanticSimilarity.KOLB Workspace Sim	no	no	no	no	no	no	yes
secondOrderSimilarity Workspace Sim	yes	no	yes	yes	yes	yes	yes
secondOrderSimilarity Workspace Sim word2vec	no	yes	no	yes	no	no	yes
Java String Similarity							
Normalized Levenshtein	no	no	no	no	no	yes	yes
Jaro-Winkler	yes	yes	yes	no	yes	yes	no
Cosine Similarity	no	no	no	no	no	yes	no
Jaccard Index	no	no	no	no	no	yes	yes
Sorensen-Dice	no	no	no	no	no	yes	no
Simmetrics							
Cosine Similarity	no	no	no	no	no	no	no
BlockDistance	no	no	no	no	no	no	no
Damerau Levenshtein	no	no	no	no	no	yes	yes
Sorensen-Dice	no	no	no	no	no	no	no
Euclidean Distance	no	no	no	no	no	no	no
Generalized Jaccard	no	no	no	no	no	no	no
Jaccard	no	no	no	no	no	no	no
Jaro-Winkler	yes	yes	yes	no	yes	yes	no
Levenshtein	no	no	no	no	no	yes	yes
Monge Elkan	no	no	no	no	no	yes	yes
Needleman Wunch	yes	yes	yes	yes	yes	yes	no
Overlap Coefficient	no	no	no	no	no	no	no
q-Gram Distance	no	no	no	no	no	yes	yes
Simon White	no	no	no	no	no	yes	yes
Smith Waterman	no	no	no	no	no	yes	yes
Smith Waterman Gotoh	no	no	no	no	no	yes	yes
Longest Common Subsequence	no	no	no	no	no	yes	yes
Longest Common Substring	no	no	no	no	no	no	yes

La tabella mostra il risultato dell'esecuzione dei test per ogni tipo di relazione considerata e, per ogni coppia <*algoritmo, relazione*>, indica se il risultato del calcolo della similarità per i campioni di **singole parole** considerati si avvicina al valore atteso o meno.

Frasi

Riporto ora i risultati del confronto fra algoritmi per il calcolo della similarità fra frasi:

Algoritmo/Perturbazione	Cambio di nome	Cambio di aggettivo	Cambio di preposizione	Mantenimento di significato	Frasi differenti
DISCO					
directedTextSimilarity Workspace Sim	yes	yes	yes	yes	no
textSimilarity Workspace Sim	yes	yes	yes	yes	no
compositionalSemanticSimilarity .ADDITION Workspace Sim word2vec	yes	yes	yes	yes	no
compositionalSemanticSimilarity .SUBTRACTION Workspace Sim word2vec	yes	yes	yes	yes	no
compositionalSemanticSimilarity .MULTIPLICATION Workspace Sim word2vec	yes	yes	yes	yes	no
compositionalSemanticSimilarity .COMBINED Workspace Sim word2vec	yes	yes	yes	yes	no
compositionalSemanticSimilarity .DILATATION Workspace Sim word2vec	yes	yes	yes	yes	no

Java String Similarity					
Normalized Levenshtein	yes	no	yes	no	yes
Jaro-Winkler	yes	yes	yes	yes	no
Cosine Similarity	yes	yes	yes	yes	no
Jaccard Index	yes	yes	yes	yes	no
Sorensen-Dice	yes	yes	yes	yes	no
Simmetrics					
Cosine Similarity	yes	yes	yes	yes	yes
BlockDistance	yes	yes	yes	yes	yes
Damerau Levenshtein	yes	no	yes	no	yes
Sorensen-Dice	yes	yes	yes	yes	yes
Euclidean Distance	yes	yes	yes	yes	no
Generalized Jaccard	yes	yes	yes	yes	yes
Jaccard	yes	yes	yes	yes	yes
Jaro-Winkler	yes	yes	yes	yes	no
Levenshtein	yes	no	yes	no	yes
Monge Elkan	yes	yes	yes	yes	no
Needleman Wunch	yes	yes	yes	yes	no
Overlap Coefficient	yes	yes	yes	yes	yes
g-Gram Distance	yes	yes	yes	yes	yes
Simon White	yes	yes	yes	yes	yes
Smith Waterman	yes	no	yes	no	yes
Smith Waterman Gotoh	yes	no	yes	no	yes
Longest Common Subsequence	yes	yes	yes	yes	yes
Longest Common Substring	yes	no	no	no	yes

Le prime quattro colonne di queste tabelle rappresentano il caso in cui la coppia di frasi considerate sono : <frase originale, frase modificata>.
In questo caso si può notare come alcuni degli algoritmi appartenenti a *Simmetrics* abbiano fornito un risultato simile al valore atteso in tutti i casi considerati.

Documenti

Algoritmo/Perturbazione	Cambiare l'ordine dei paragrafi	Eliminare/aggiungere paragrafi	Modifica paragrafi	Documenti differenti
Java String Similarity				
Normalized Levenshtein	no	yes	yes	yes
Jaro-Winkler	yes	yes	yes	no
Cosine Similarity	yes	yes	yes	no
Jaccard Index	yes	yes	yes	no
Sorensen-Dice	yes	yes	yes	no
Simmetrics				
Cosine Similarity	yes	yes	yes	no
BlockDistance	yes	yes	yes	yes
Damerau Levenshtein	no	yes	yes	yes
Sorensen-Dice	yes	yes	yes	yes
Euclidean Distance	yes	yes	yes	no
Generalized Jaccard	yes	yes	yes	yes
Jaccard	yes	yes	yes	yes
Jaro-Winkler	yes	yes	yes	no
Levenshtein	no	yes	yes	yes
Monge Elkan	yes	yes	yes	no
Needleman Wunch	yes	yes	yes	no
Overlap Coefficient	yes	yes	yes	yes
g-Gram Distance	yes	yes	yes	yes
Simon White	yes	yes	yes	no
Smith Waterman	X	X	X	X
Smith Waterman Gotoh	no	no	yes	yes
Longest Common Subsequence	yes	yes	yes	yes
Longest Common Substring	no	no	no	yes

Anche nel caso dei documenti sono emersi determinati algoritmi che forniscono un risultato che si avvicina particolarmente al valore atteso di similarità. In particolare *Smith Waterman* presenta evidenti problemi di performance pertanto non è stato considerato valido, già in questa fase, per il calcolo della similarità fra documenti.

Questi risultati preliminari sono stati ottenuti considerando un insieme di dati ridotto per facilitarne la comprensione. Successivamente verrà effettuata un'analisi più approfondita volta a considerare solamente gli algoritmi che in questa fase hanno fornito i risultati migliori.

Scelta della Soglia di Similarità

Per effettuare il calcolo della soglia di similarità sono stati considerati un insieme di 10 coppie di parole, frasi e documenti. I dati sono strutturati esattamente come specificato nella sezione 3.2 e sono stati considerati unicamente gli algoritmi migliori di ogni tool a seconda del caso specifico.
Con lo scopo di ottenere un test coerente, si è deciso di confrontare solo i casi in cui ci si aspetta valori di similarità che superano la soglia scelta a priori di 0,5. In particolare dopo aver eseguito un test specifico per ogni caso di studio, considerato in maniera separata per parole, frasi e documenti, è stata calcolata una media aritmetica dei valori risultanti da ogni algoritmo così da ottenere un valore di soglia preliminare.

Parole

	Sinonimia	Antinomia	Iponimia/Iperonimia	Iponimia/Iponimia	Meronimia	Participiale
Soglia di Similarità	0,5722	0,5088	0,4356	0,5931	0,4339	0,5488

Frasi

	Cambio di nome	Cambio di aggettivo	Cambio di preposizione	Mantenimento di significato
Soglia di Similarità	0,9261	0,9415	0,8411	0,8079

Documenti

	Cambiare l'ordine dei paragrafi	Eliminare/aggiungere paragrafi	Modifica paragrafi
Soglia di Similarità	0,9078	0,8905	0,8619

Da questi risultati si deduce che la modifica anche solo di una lettera in una parola ha molto più peso sul calcolo della soglia di similarità rispetto alla modifica per esempio di un'intera parola in una frase o un documento. Come conseguenza diretta di ciò la soglia di similarità totale si abbassa inducendo così l'utilizzatore a considerare simili, in maniera erronea, anche parole che non condividono alcuna relazione di similarià fra loro.

Grazie ai risultati mostrati finora è stato possibile trovare la soglia di similarità totale per parole, frasi e documenti calcolata come la media aritmetica delle soglie trovate in precedenza :

	Parole	**Frasi**	**Documenti**
Soglia di Similarità	0,5154	0,8792	0,8867

Infine si vuole ricordare che il calcolo della soglia di similarità non comprende tutti i casi considerati nella sezione 3.3, bensì solo quelli che riportano *yes* nelle tabelle mostrate in precedenza.
L'unico algoritmo che presenta evidenti problemi di prestazioni è *SmithWaterman* in quanto nel calcolo della similarità fra documenti i tempi di calcolo per questo algoritmo risultano essere eccessivi, pertanto non è stato considerato nel calcolo della soglia.

Misurazioni Simili

Visti i risultati ottenuti dagli algoritmi per il calcolo della similarità è bene sottolineare il fatto che alcuni di essi riescano con successo (vedi sezione 3.3) a inferire se due frasi o documenti sono simili o meno in tutti i casi considerati. Per le parole invece non è stato trovato alcun algoritmo che rispecchia tale proprietà, pertanto reputo difficile l'applicazione degli strumenti considerati in un caso reale.
In questa sezione l'obiettivo è quello di selezionare l'algoritmo più performante prendendo in considerazione unicamente quelli che presentano tutti *yes* nelle tabelle mostrate nella sezione 3.3 e quindi solamente gli algoritmi che danno un risultato che si avvicina particolarmente al valore di similarità atteso per **ogni** perturbazione considerata.
Si può dedurre quindi che l'unica libreria considerata per i test prestazionali sarà *Simmetrics* sia per le frasi che i documenti :

Sentences Simmetrics Algorithms					
Cosine Similarity	yes	yes	yes	yes	yes
BlockDistance	yes	yes	yes	yes	yes
Sorensen-Dice	yes	yes	yes	yes	yes
Generalized Jaccard	yes	yes	yes	yes	yes
Jaccard	yes	yes	yes	yes	yes
Overlap Coefficient	yes	yes	yes	yes	yes
g-Gram Distance	yes	yes	yes	yes	yes
Simon White	yes	yes	yes	yes	yes
Longest Common Subsequence	yes	yes	yes	yes	yes

Documents Simmetrics Algorithms				
BlockDistance	yes	yes	yes	yes
Sorensen-Dice	yes	yes	yes	yes
Generalized Jaccard	yes	yes	yes	yes
Jaccard	yes	yes	yes	yes
Overlap Coefficient	yes	yes	yes	yes
g-Gram Distance	yes	yes	yes	yes
Longest Common Subsequence	yes	yes	yes	yes

Implementazione

L'implementazione ha previsto l'estensione dei test svolti finora per *Simmetrics* in cui sono state introdotte due nuove modalità, opportunamente aggiunte al parametro *TestConfiguration*:

- **SENTENCEPERFORMANCETEST** : Esegue tutti gli algoritmi utili per le frasi le cui istanze sono contenute nei *SIMTestData* riportando in standard output i tempi di ogni esecuzione in nanosecondi.
- **DOCUMENTSPERFORMANCETEST** : Esegue tutti gli algoritmi utili per i documenti le cui istanze sono contenute nei *SIMTestData* riportando in standard output i tempi di ogni esecuzione in nanosecondi.

Per sfruttare le due nuove modalità con l'intento di eseguire un test più inerente ad un caso di utilizzo reale, si è deciso di recuperare un insieme di frasi e documenti contenuti nel *Legal Case Reports Data Set*[40] il quale rappresenta un insieme di casi legali provenienti dalla corte federale australiana. La

[40] `https://archive.ics.uci.edu/ml/datasets/Legal+Case+Reports`

particolarità di questo dataset è che fornisce non solo i documenti ma anche le frasi e citazioni riguardanti tutti i casi in formato testuale.
Con l'obiettivo di recuperare tali informazioni e fornirle opportunamente in input al test, è stata implementata una classe di utility chiamata *SIMUtil* e contenuta in un nuovo package che presenta lo stesso nome, la quale fornisce un insieme di metodi statici utili per raggiungere lo scopo.
Grazie all'appoggio di questa classe è stato possibile inserire due classi di main aggiuntive rispettivamente nei package **SIMDocumentSpecificTest** e **SIMSentenceSpecificTest** :

- **SIMSentencePerformanceMain** : Carica 2000 frasi diverse fra loro e le organizza in due liste contenenti 1000 frasi ciascuno, successivamente avvia il test in modalità *SENTENCEPERFORMANCETEST*.
- **SIMDocumentPerformanceMain** : Carica 2000 documenti diversi fra loro e li organizza in due liste contenenti 1000 documenti ciascuno, successivamente avvia il test in modalità *DOCUMENTSPERFORMANCETEST*.

Risultati

Per ognuna delle modalità presentate in precedenza è stato condotto un test sperimentale di cui riporto i risultati in seguito :

Frasi

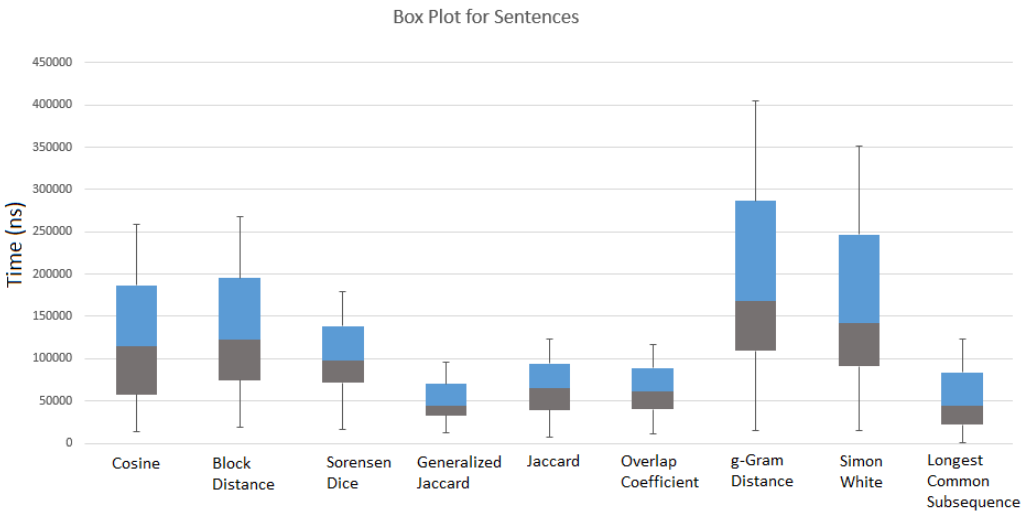


Figura 3.6: Analisi di performance sul test specifico per le frasi.

Dall'immagine 3.6 emerge che la scelta dell'algoritmo più performante per il calcolo della similarità fra frasi ricade su *Generalized Jaccard* il quale rappresenta, come specificato nella sezione 3.2, la versione estesa di *Jaccard* nel quale non viene tenuta in considerazione l'occorrenza di una generica entry dell'insieme formato dalle stringhe in input. In aggiunta a queste considerazioni il fatto che la formula per il calcolo del *Jaccard Index* richiede la trasformazione dell'input in insiemi di *n-grammi* porta ad ottenere in pratica un guadagno in termini di precisione sul calcolo della similarità a discapito delle prestazioni. Infine la tabella che mostra la tassonomia di algoritmi nella sezione 3.2 evidenzia che il calcolo del *Jaccard Index* è *Normalizzato* e rispetta la proprietà geometrica chiamata *triangle inequality*, il che introduce il vantaggio di poter applicare questo algoritmo a qualsiasi tipo di frase espressa in linguaggio naturale.

Documenti

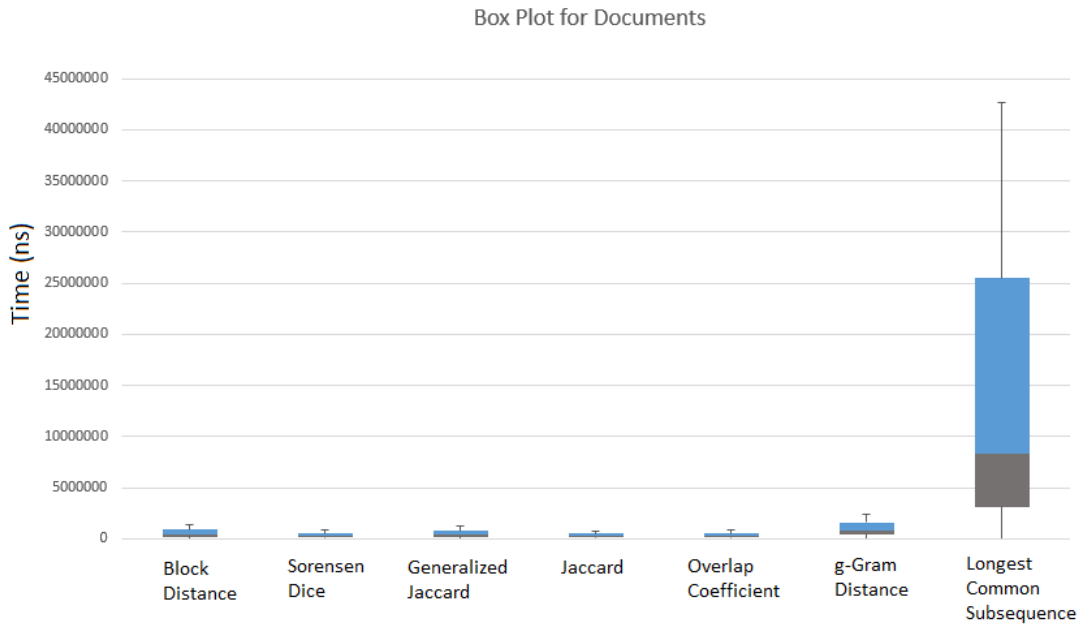


Figura 3.7: Analisi di performance sul test specifico per i documenti.

L'analisi riportata in figura 3.7, i cui risultati hanno granularità del nanosecondo, ha messo in evidenza che le performance dell'algoritmo *Longest Common Subsequence* non sono comparabili rispetto a quelle riscontrate in tutti gli altri casi. In seguito a ciò si è deciso di scartare a priori l'ipotesi che questo algoritmo possa essere utilizzato in futuro per il calcolo della simila-

rità fra documenti. In seguito viene riportato lo stesso test in cui *Longest Common Subsequence* non viene considerato.

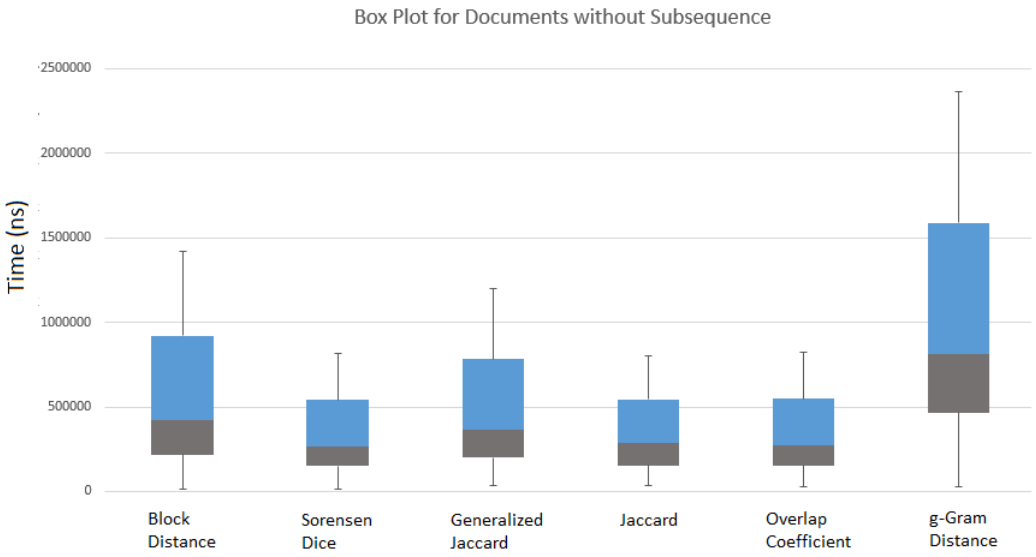


Figura 3.8: Analisi di performance sul test specifico per i documenti senza *Longest Common Subsequence*.

I test di performance, mostrati in figura 3.8, in questo caso non sono risultati sufficienti a selezionare un'unico algoritmo per il calcolo della similarità fra documenti in quanto sono presenti tre algoritmi che sono equivalenti: *Sorensen Dice*, *Jaccard* e *Overlap Coefficient*. Verrà ora affrontata un'analisi più in dettaglio sugli aspetti matematici di ognuno di essi.
Dalle formule evidenziate nella sezione 3.2 si evince che ognuno dei tre algoritmi presenta al numeratore un calcolo basato sull'intersezione degli insiemi costituiti dalle stringhe in input :

$$A \cap B = \sum_{n}^{i=1} w_i : w_i \in A \wedge w_i \in B \tag{3.16}$$

Se prendiamo in considerazione documenti testuali, ognuno dei tre algoritmi controlla se una parola w_i è presente in entrambi i documenti A e B senza tener conto dell'occorrenza di ogni parola, pertanto il calcolo del numeratore risulta identico se in input si presentano le stringhe "boats" e "boats boats boats".
Per questo motivo si è deciso di scartare questi tre algoritmi e di selezionare il quarto in classifica, ovvero *Generalized Jaccard*. I vantaggi di utilizzo di questo algoritmo sono già stati elencati in precedenza insieme alla risoluzione del problema dell'occorrenza.

3.4 Progettazione dei test di fattibilità e prestazioni

Finora sono stati presentati tre motori di inferenza semantica : *Pellet*, *Hermit* e *JFact* e tre librerie per il calcolo di similarità : *DISCO*, *Java String Similarity* e *Simmetrics*. Per ognuno dei motori di inferenza semantica è stato progettato un test mirato ad individuare quello più adatto per essere utilizzato nel contesto applicativo di aggregazione di testi che verrà presentato nel capitolo successivo.
Una volta selezionato *Hermit* come strumento di riferimento sono stati effettuati una serie di test più specifici che hanno messo in discussione sia le prestazioni che la stabilità dello strumento stesso, i quali hanno confermato da un lato che esso rappresenta la scelta ottimale e dall'altro l'applicabilità di questo tipo di strumenti in un task di aggregazione di testi.
Parallelamente è stato progettato un test basato sulle librerie per il calcolo della similarità con l'obiettivo iniziale di ottenere una tassonomia degli algoritmi, la quale ha portato alla scelta di una *soglia di similarità* per le parole, frasi e documenti. Per poter selezionare anche in questo caso l'algoritmo, o set di algoritmi, più adatto per affrontare il task di aggregazione è stata svolta un'analisi sulle prestazioni e proprietà caratteristiche, per esempio il fatto di rispettare la *triangle inequality*, di ognuno di essi.
I due strumenti selezionati : *Hermit* e *Simmetrics*, in particolare l'algoritmo *GeneralizedJaccard*, assieme alla soglia di similarità citata poc'anzi verranno sfruttati come punto di partenza nel prossimo capitolo per realizzare un sistema il cui scopo è aggregare testi. In particolare verranno sviluppate due versioni di tale sistema, una che sfrutta l'approccio basato su ontologie e un'altra quello basato su misure di similarità.

Di seguito si mostrano i diagrammi C4 [2] di *Container* e *Component* utili a fornire al lettore una prima descrizione progettuale del sistema di aggregazione :

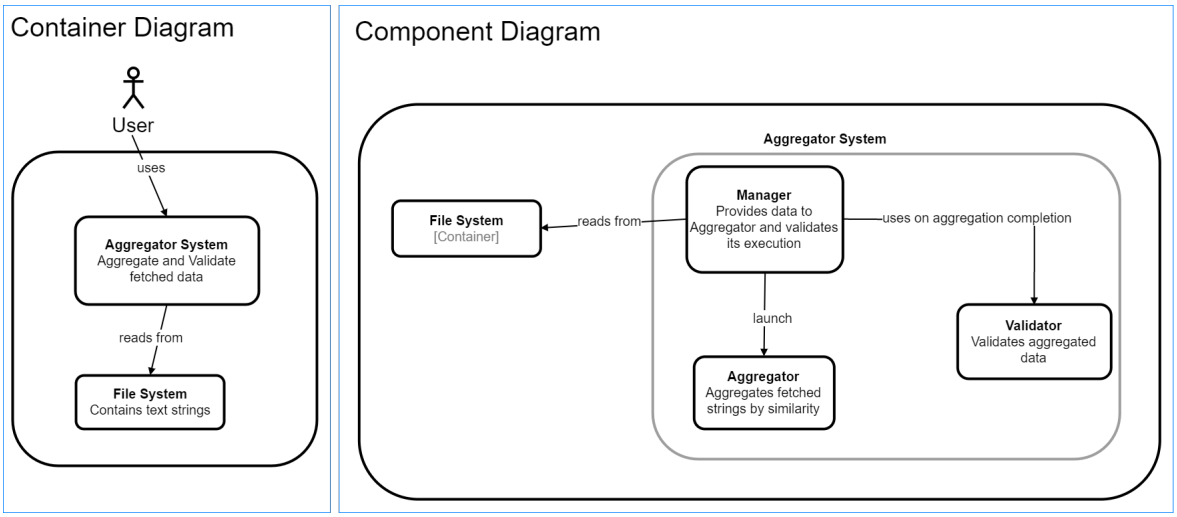


Figura 3.9: Diagrammi C4 di *Container* e *Component* del sistema di aggregazione.

Dal diagramma *Component* si evidenzia una prima struttura formale delle entità che compongono il sistema di aggregazione. In particolare il *Manager* rappresenta colui che gestisce la sequenza di passi logici di esecuzione : caricare i dati da *File System*, avviare l'*Aggregator* con i dati caricati, avviare il *Validator* una volta terminata l'aggregazione. Si vuole evidenziare che attualmente il sistema è composto da componenti aggiuntivi, come ad esempio il *Validator*, progettati in quanto essenziali in questo studio per consentire un'analisi approfondita una volta terminata l'aggregazione.
Il *Validator* permetterà di verificare i risultati dell'*Aggregator*, basandosi su diversi parametri fra i quali la soglia di similarità, con l'obiettivo di capire se i testi aggregati sono simili implementando un metodo di analisi che si avvicina il più possibile a quello di un utente umano.

Capitolo 4

Aggregazione di testi basata su similarità

4.1 Il modello MoK

MoK è un modello di coordinazione ispirato dalla biochimica che permette l'auto-organizzazione della conoscenza [5]. La motivazione alla base del modello MoK è l'idea che la conoscenza si deve autonomamente aggregare e diffondere fino a raggiungere il suo consumatore senza essere ricercata esplicitamente. Il modello si ispira alla biochimica dato che ormai si riconosce ai sistemi biochimici, la capacita di raggiungere un "ordine partendo dal caos", grazie alla loro adattabilità e all'auto-organizzazione.

Concetti principali alla base del modello

Il modello MoK si basa su diversi concetti presi dalla biochimica [6] come:

- gli atomi: sono la parte di conoscenza più piccola; un atomo contiene informazioni provenienti da una sorgente ed è situato in un compartimento dove è soggetto a date leggi;

- le molecole: sono le unità dove vengono aggregate le informazioni, all'interno delle molecole si trovano informazioni (atomi) tra loro correlate;

- gli enzimi: emessi dai catalizzatori, gli enzimi influenzano le relazioni all'interno di MoK in maniera tale da modificare le dinamiche di evoluzione della conoscenza del compartimento per andare incontro a ciò che interessa all'utente;

- le reazioni : sono le leggi biochimiche che regolano il compartimento; hanno un determinato rate con cui vengono applicate. Attraverso le reazioni vengono specificate regole per l'aggregazione, la diffusione e il decadimento delle informazioni;

- i compartimenti: rappresentano il luogo concettuale che contiene tutte le entità di MoK, è sulla base di questo concetto che si possono definire i concetti di località e vicinanza;

- le sorgenti: sono le origini della conoscenza che viene continuamente iniettata ad un determinato rate sotto forma di atomo all'interno del relativo compartimento;

- i catalizzatori: rappresentano i consumatori-utilizzatori o prosumer di conoscenza, emettono enzimi che rappresentano le loro azioni, le quali andranno ad impattare sulle dinamiche della conoscenza all'interno del proprio compartimento in modo tale da incrementare la probabilità che giungano informazioni rilevanti per l'utente.

I concetti principali del modello sono gli atomi, le molecole, gli enzimi e le reazioni che verranno analizzati in dettaglio di seguito.

Gli atomi

Gli atomi, oltre a contenere al loro interno la conoscenza devono avere anche delle informazioni rispetto al contesto come l'origine del contenuto in maniera tale da mantenere il significato originale. Un atomo all'interno del modello MoK è una tripla definita come $atom(src,val,attr)_c$ dove *src* identifica in modo non ambiguo la sorgente, *val* è il corrente pezzo di conoscenza e *attr* è un attributo relativo alla conoscenza contenuta che permette una sua migliore comprensione. Questo attributo generalmente si basa su una ontologia ben definita o su un vocabolario. Infine la c a pedice rappresenta la corrente concentrazione dell'atomo, cioè il numero di atomi dello stesso tipo all'interno del compartimento.

Le molecole

Le molecole sono degli aggregatori stocastici guidati dall'ambiente di atomi, il loro obiettivo è quello di estrapolare relazioni semantiche fra gli atomi e, in base ad esse, aggregarli, creando così nuova conoscenza. Ogni molecola è semplicemente un insieme di atomi non ordinati tra loro semanticamente correlati. La struttura di una molecola nel modello MoK è la seguen-

te *molecule(Atoms)*$_c$ dove c rappresenta la concentrazione della molecola e *Atoms* la collezione di atomi correlati dalla molecola.

Gli enzimi

Gli enzimi sono una delle entità chiave del modello MoK perchè interpretano le azioni sulla conoscenza da parte del prosumer come feedback positivi in modo tale da incrementare, all'interno del compartimento, la concentrazione delle molecole correlate a quelle di interesse.

La rappresentazione degli enzimi all'interno di MoK è la seguente *enzyme(Species, s, Reactant, Context)*$_c$ dove c è la concentrazione dell'enzima, *Species* denota la natura epistemica dell'azione reificata da esso aiutando il sistema a determinare le attività da intraprendere, *Context* rappresenta la traccia delle informazioni contestuali relative all'azione, s denota la forza dell'enzima e *Reactant* è l'entità del modello MoK coinvolta nell'azione reificata.

Le reazioni

All'interno del modello MoK sono presenti diverse reazioni di base. In MoK la correlazione semantica di informazioni viene effettuata attraverso la F_{MoK} che rappresenta il meccanismo di *matchmaking* utilizzato dalle reazioni per assegnare uno *score*, ossia un parametro di correlazione, a due entità possibilmente eterogenee del modello MoK.
Un'altra reazione alla base del modello è quella di aggregazione, la quale date due molecole le aggrega ed è definita come:

$$\begin{aligned} molecule(Atoms_1) + molecule(Atoms_2) \xrightarrow{r-agg} \\ molecule(Atoms_1 \cup Atoms_2) + residual(Atoms_1, Atoms_2) \end{aligned} \tag{4.1}$$

Nella relazione, *r-agg* rappresenta la frequenza con la quale la reazione viene applicata, inoltre è necessario che esista almeno una coppia di atomi $<$*atom1, atom2*$>$ che appartengono ai rispettivi insiemi $<Atoms_1, Atoms_2>$ con una determinata correlazione sulla base della regola semantica sopra citata. In *residual* vengono messi tutti gli atomi di $Atoms_1$ e $Atoms_2$ che non sono stati aggregati.
Grazie alla reazione di rinforzo viene attuato il feedback positivo, la reazione consuma un enzima e produce una molecola/atomo relativa:

$$enzyme(Atoms_1) + molecule(Atoms_2)_c \xrightarrow{r-reinf} molecule(Atoms_2)_{c+1} \tag{4.2}$$

dove *r-reinf* rappresenta la frequenza con la quale la reazione di rinforzo viene applicata. L'enzima *enzyme(Atoms*$_1$*)* e la molecola *molecule(Atoms*$_2$*)*$_c$ devono essere all'interno dello stesso compartimento, con $c \neq 0$ e *mok(atom*$_1$*, atom*$_2$*)* deve essere valida per $atom_1 \in Atoms_1$, $atom_2 \in Atoms_2$.
Per seguire la metafora biochimica, le molecole devono svanire con il passare del tempo, diminuendo pian piano la loro concentrazione secondo una legge di decadimento riportata di seguito:

$$molecule(Atoms)_c \xrightarrow{r-decay} molecule(Atoms)_{c-1} \tag{4.3}$$

È stato necessario prevedere anche un meccanismo per la diffusione delle molecole fra i vari compartimento vicini. La relativa reazione viene rappresentata come segue:

$$\begin{aligned} \|Molecules_1 \cup molecule_1\|_{\sigma'} + \|Molecule_2\|_{\sigma''} \xrightarrow{r_{diff}} \\ \|Molecules_1\|_{\sigma'} + \|Molecule_2 \cup molecule_1\|_{\sigma''} \end{aligned} \tag{4.4}$$

dove la struttura $\|\|_\sigma$ significa *all'interno del compartimento* σ. Come nelle reazioni precedenti *r-diff* rappresenta la frequenza della reazione in esami e σ' e σ'' sono compartimenti vicini.

Definizione del modello per l'*Aggregator System*

Il modello proposto finora mette in evidenza diverse entità concettuali utili per definire le condizioni al contorno dell'*Aggregator System*. Lo scopo in questo capitolo è quello di sviluppare la reazione di aggregazione descritta dall'equazione 4.1 producendo un sistema che rispetti determinate proprietà prestazionali per poter essere integrato, con il minimo sforzo, nel sistema MoK.
Per semplificare lo sviluppo dell'*Aggregator System* ci si concentra ora sulla reazione di aggregazione e le entità concettuali necessarie per la sua realizzazione. Una particolarità che non può essere trascurata in fase di sviluppo è la testabilità del sistema che viene effettuata post-aggregazione dal *Validator*, il quale ha il compito di sfruttare le informazioni contenute nelle entità del modello MoK per poter determinare l'esito di aggregazione.
Partendo dal concetto di reazione di aggregazione, definito dall'equazione 4.1, è possibile pensare ad una suddivisione di essa in tre passi logici differenti:

- **Aggregare atomi** : Rappresenta una semplificazione del concetto di aggregazione in cui, in un dato compartimento, si prova ad aggregare

tutti e soli gli atomi a coppie. Il risultato finale in questo caso concerne la creazione di molecole contenenti al più due atomi al loro interno;

- **Aggregare atomi in molecole** : Dato un compartimento di partenza in cui possono essere presenti sia atomi che molecole, per ogni atomo si prova ad effettuare l'aggregazione con ognuna delle molecole esistenti, in caso di successo si prevede l'aggiunta dell'atomo all'interno della molecola simile identificata;

- **Aggregare molecole** : Rappresenta l'estensione del passo precedente in cui, partendo da un compartimento in cui sono presenti sia atomi che molecole, prova ad aggregare a coppie le molecole esistenti generando, se tutte le condizioni vengono soddisfatte, una molecola contentente tutti gli atomi appartenenti a ciascuna delle molecole considerate;

Seguendo questa suddivisione dei compiti risulta più semplice passare dalla progettazione all'implementazione in cui ci si può concentrare sul perfezionamento del singolo passo in termini di prestazioni e correttezza. Inoltre si vuole ricordare che MoK non esegue l'aggregazione in modo lineare considerando un compartimento stazionario in un dato istante, bensì tenterà di aggregare due entità del modello seguendo le leggi della biochimica.
Di seguito viene mostrata, nella figura 4.1, la versione modificata del diagramma C4[2] *Component* dell'*Aggregator System* presentato nella sezione 3.4:

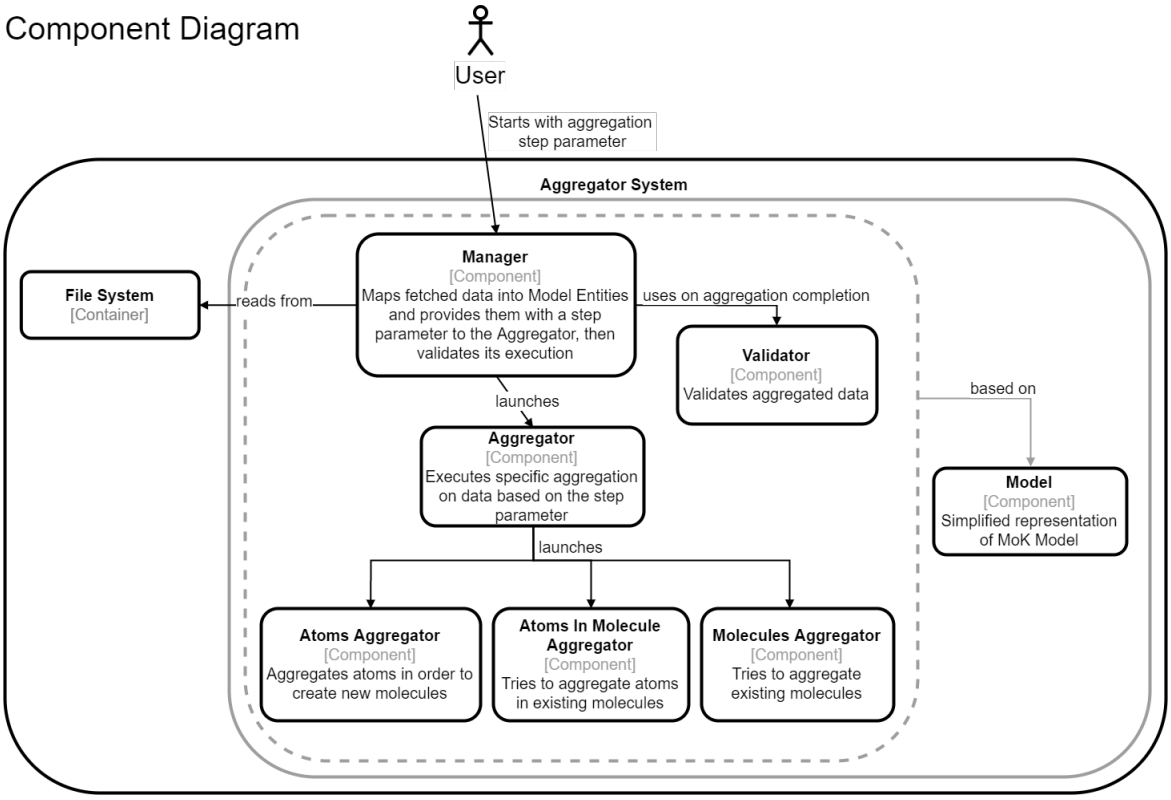


Figura 4.1: Estensione del diagramma C4 *Component* del sistema di aggregazione.

L'entità *Model*, rappresentata in figura 4.1, è l'insieme dei concetti semplificati ottenuti dal modello MoK che l'*Aggregator System* sfrutta per l'aggregazione e soprattutto per effettuare la validazione a posteriori. Il *Manager* che si occupa di leggere i dati da *File System* avrà l'onere di presentare all'*Aggregator* i dati mappati utilizzando la rappresentazione del *Model*.
Prendendo come i riferimento le entità concettuali definite finora nel modello MoK è stata definita una versione del modello semplificata in cui vengono mantenute unicamente le entità necessarie ai fini specifici determinati dalla reazione di aggregazione, di seguito si riporta il diagramma C4[2] *Class* (figura 4.4) del suddetto modello:

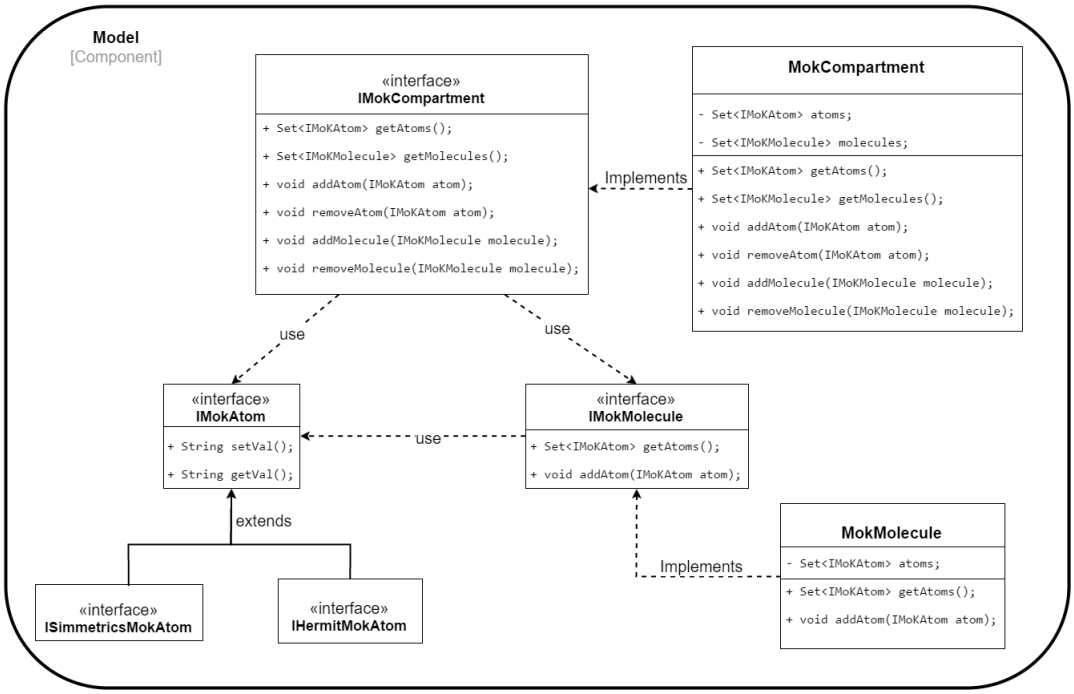


Figura 4.2: Diagramma C4 *Class*, zooming sull'entità *Model*.

In questa versione semplificata del modello, rappresentata in figura 4.2, sono state mantenute le entità *Atomi*, *Molecole* e *Compartimento* dove in particolare l'entità *IMoKAtom* contiene le informazioni che generalmente caratterizzano l'atomo MoK tralasciando le proprietà *src*, *attr* e *c* in quanto esse non risultano necessarie per realizzare l'aggregazione. Inoltre vengono fornite già in questa fase progettuale due entità aggiuntive: *ISimmetricsMoKAtom* e *IHermitMoKAtom* che estendono *IMoKAtom* aggiungendo le informazioni necessarie, a seconda dello strumento utilizzato, per realizzare la validazione a posteriori. La struttura UML di queste due entità verrà specificata in seguito.
Le entità *Compartimento* e *Molecola* sono state anch'esse ridotte ai minimi termini ed in questa rappresentazione sono semplicemente contenitori di in un caso atomi e molecole e nell'altro solamente di atomi.
Nelle sezioni successive per semplicità vengono considerati due casi: **aggregazione omogenea di frasi** per l'approccio *non-semantico* e **aggregazione omogenea di parole** per l'approccio *semantico*.

4.2 Approccio non-semantico: Simmetrics

Basandosi sui modelli di *Aggregator System* e *Model*, proposti nella sezione 4.1, viene introdotto di seguito il dataset utilizzato durante lo sviluppo del sistema in cui la reazione di aggregazione viene implementata utilizzando *Simmetrics*, questo permetterà di sintetizzare l'interfaccia *ISimmetricsMokAtom*.

Scelta del Dataset

Il dataset, come citato nella sezione 4.1, dovrà contenere frasi che verranno mappate opportunamente dall'entità *Manager* dell'*Aggregator System* in entità *IMoKAtom* e *IMoKMolecule* a seconda delle necessità derivanti dallo specifico passo implementativo.
In aggiunta il dataset dovrà necessariamente contenere informazioni relative al ranking di similarità fra frasi in modo tale da facilitare la validazione a posteriori.
La scelta, dovendo rispettare queste specifiche, è ricaduta su insieme di file testuali ottenuti da *Semantic Text Similarity Dataset Hub*[1] che contengono le informazioni seguendo il seguente formato specificato per una singola riga:

Similarity Sentence1 Sentence2

Dove ogni componente della riga è separato da tab ed è da interpretarsi come segue:

- *Sentence1* : rappresenta la prima frase, scritta in lingua inglese.
- *Sentence2* : rappresenta la seconda frase, scritta in lingua inglese.
- *Similarity* : rappresenta il valore di similarità determinato secondo la **Gold Standard**[2] compreso tra 0 e 5 individuato tra *Sentense1* e *Sentence2* dove :
  - il valore 5 determina che le due frasi sono completamente equivalenti, ovvero significano la stessa cosa. Un esempio è dato dalla coppia <*The bird is bathing in the sink.* , *Birdie is washing itself in the water basin.*>;
  - il valore 4 determina che le due frasi sono equivalenti in gran parte ma qualche dettaglio non importante differisce. Un esempio è dato dalla coppia <*In May 2010, the troops attempted to invade Kabul.* , *The US army invaded Kabul on May 7th last year, 2010.*>;

[1] `https://github.com/brmson/dataset-sts`
[2] `https://goo.gl/QrvoGf`

- il valore 3 determina che le due frasi sono più o meno equivalenti ma qualche informazione importante manca o differisce. Un esempio è dato dalla coppia <*John said he is considered a witness but not a suspect.* , *"He is not a suspect anymore." John said.*>;
- il valore 2 determina che le due frasi non sono equivalenti ma condividono gli stessi dettagli. Un esempio è dato dalla coppia <*They flew out of the nest in groups.* , *They flew into the nest together.*>;
- il valore 1 determina che le due frasi non sono equivalenti ma condividono lo stesso argomento. Un esempio è dato dalla coppia <*The woman is playing the violin.* , *The young lady enjoys listening to the guitar.*>;
- il valore 0 determina che le due frasi riguardano argomenti differenti. Un esempio è dato dalla coppia <*John went horse back riding at dawn with a whole group of friends.* , *Sunrise at dawn is a magnificent view to take in if you wake up early enough for it.*>;

Il dataset è stato assemblato utilizzando il metodo *Mechanical Turk*[3], raccogliendo diversi punteggi per ogni coppia di frasi e, per ogni riga, il punteggio *Similarity* è stato attribuito come la media di essi.

Mapping del Dataset in entità del Modello

Come specificato nel diagramma *Component* dell'*Aggregator System*, l'entità *Manager* deve occuparsi di leggere il dataset e mapparlo opportunamente in entità del modello MoK semplificato.
A tale scopo si è deciso di generare due entità *ISimmetricsMoKAtom* per ogni riga contenuta nel dataset e, per permettere la validazione a posteriori, la struttura di tale entità sarà come descritto di seguito (figura 4.3):

[3] `https://www.researchgate.net/figure/239939742_fig1_Figure-1-Example-of-Amazon-Mechanical-Turk-HIT-used-for-creating-the-Gold-Standard`

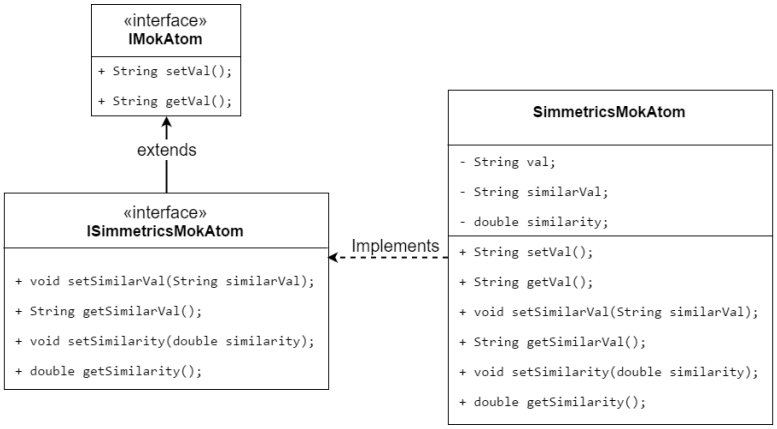


Figura 4.3: Diagramma C4 *Class*, zooming sull'entità *Model*.

In figura 4.3, rispetto all'entità *IMoKAtom*, vengono aggiunte le proprietà *similarVal* e *similarity* che sono da interpretarsi come segue:

- *val* : rappresenta il vero contenuto dell'atomo, ovvero una frase in formato testuale;
- *similarVal* : rappresenta la frase che il dataset reputa simile a quella contenuta in *Val*;
- *similarity* : corrisponde al valore di similarità riportato nel dataset che lega *Val* a *similarVal*;

Implementazione dell'*Aggregator System* per Simmetrics

Sulla base della descrizione del modello dei *Component*, descritta in figura 4.1, si è deciso di implementare l'*Aggregator* come entità attiva, il quale si occupa di aggregare il compartimento fornitogli dal *Manager* una volta completata la fase iniziale di traduzione del dataset in entità del modello MoK semplificato. Anche se attualmente il sistema progettato non presenta tale necessità, in Mok risulta essenziale la suddivisione dei compiti in diversi flussi di controllo per poter gestire al meglio il concetto di reazione biochimica.
In figura 4.4 viene mostrato il diagramma C4 *Class* che riporta una descrizione formale dell'implementazione dell'*Aggregator System* per *Simmetrics*:

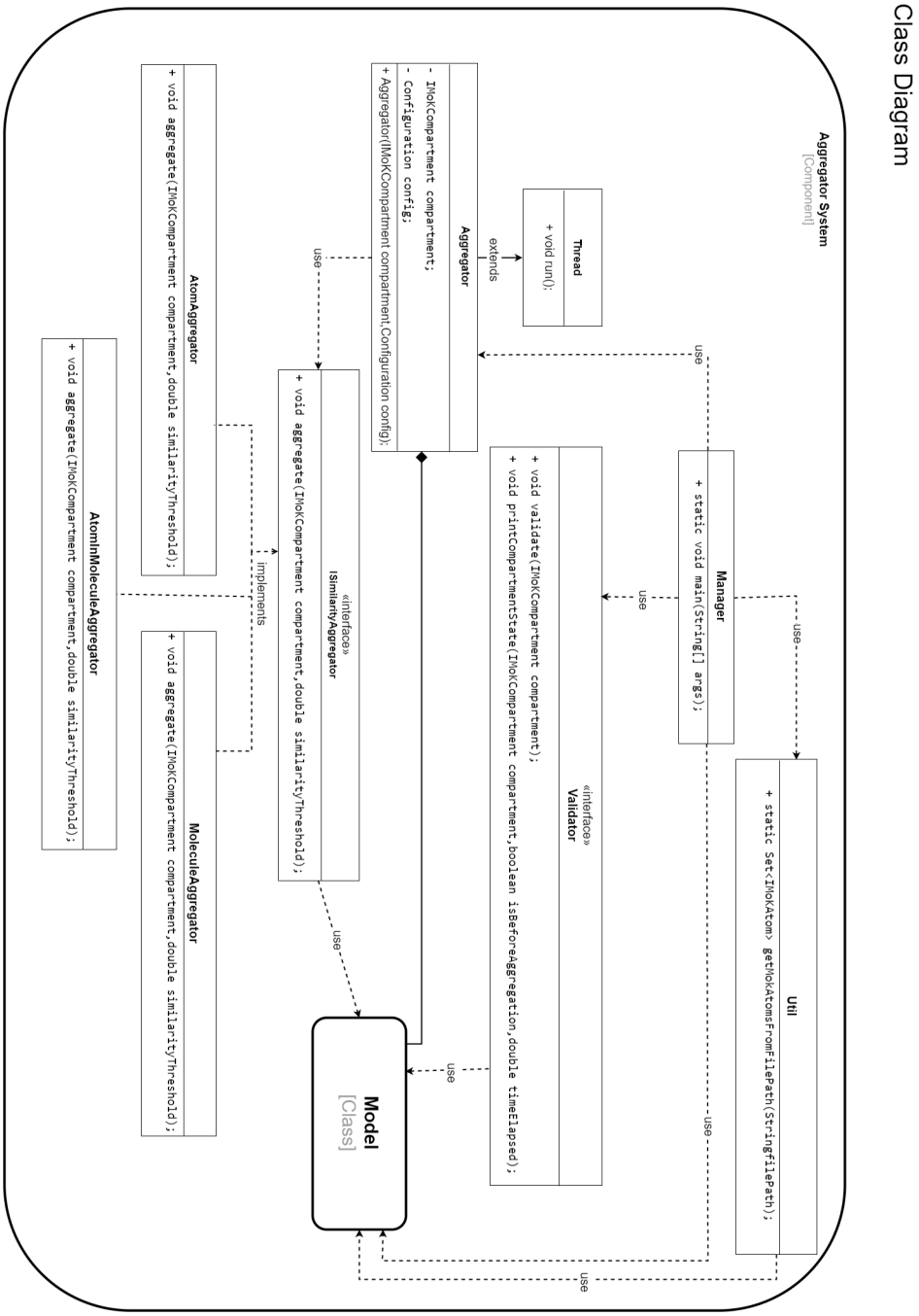


Figura 4.4: Diagramma C4 *Class* del sistema di aggregazione.

Dalla figura 4.4 emerge la definizione UML delle entità che compongono il sistema di aggregazione di cui viene data, per le principali, una descrizione in linguaggio naturale per facilitarne la comprensione:

- **Manager** : come descritto in figura 4.1, questa entità rappresenta il coordinatore del sistema. Per agevolare la riusabilità del codice si è deciso di implementare un'entità di supporto *Util* che si occupa di leggere le informazioni dal dataset e di tradurle in entità del modello semplificato. Grazie a questa separazione concettuale risulta più semplice immaginare la realizzazione di diverse entità *Manager*. Inoltre questa entità ha l'onere di avviare l'*Aggregator*, aspettare la sua terminazione ed iniziare una sessione di validazione sull'*IMoKCompartment* risultante tramite il *Validator*;

- **Aggregator** : è un'entità attiva che esiste con l'unico scopo di aggregare la conoscenza contenuta nell'*IMoKCompartment*. In aggiunta, quest'ultimo è composto da un parametro di configurazione che definisce lo specifico passo di aggregazione da eseguire e contiene la soglia di similarità determinata nella sezione 3.3 per le frasi che comunicherà alle entità che implementano gli specifici passi di aggregazione;

Implementazione dei passi di Aggregazione

Dal diagramma mostrato in figura 4.1 emergono tre passi di aggregazione distinti ciascuno dei quali è stato implementato in concomitanza con il metodo di validazione.
Si specifica che ognuno di essi utilizza i risultati conseguiti nel capitolo precedente che, nel caso di *Simmetrics*, prevede l'uso dell'algoritmo *Generalized Jaccard* e di una soglia di similarità pari a 0,8792.

Aggregazione fra Atomi

L'implementazione di questo specifico passo di aggregazione prevede il tentativo di aggregare ogni singola entità *ISimmetricsMoKAtom* con un'altra dello stesso tipo che risulta abbastaza simile. Anche se in ogni atomo sono presenti le proprietà *similarVal* e *similarity* che riportano le informazioni contenute nel dataset, lo scopo di questo specifico aggregatore è quello di confrontare, utilizzando *Generalized Jaccard*, la proprietà *Val* di ogni atomo presente nell'*IMoKCompartment* con quella di tutti gli altri finchè il valore di similarità restituito da *Generalized Jaccard* supera la soglia, in quel caso viene generata e aggiunta al compartimento una nuova *IMoKMolecule* contenente i due *ISimmetricsMokAtom* confrontati. Se l'aggregatore non trova

alcuna coppia di atomi da aggregare, essi rimangono invariati.
Al termine di questo processo l'*IMoKCompartment* conterrà sia le nuove molecole generate che gli atomi non aggregati, a questo punto egli deve occuparsi di validare l'aggregazione.

Validare l'Aggregazione fra Atomi

Come mostrato nel diagramma C4 *Class* in figura 4.4, l'entità *Validator* offre due metodi a chi lo utilizza, uno per validare un *IMoKCompartment* e l'altro per stamparne lo stato. In particolare il metodo *validate* permette di verificare se ogni molecola presente nel compartimento è composta da atomi che sono simili o meno. Per implementare questo concetto si è pensato inizialmente di sfruttare le due proprietà *similarVal* e *similarity* contenute in ogni *IMoKAtom* per effettuare un controllo congiunto sui due atomi, i quali presumibilmente sono stati aggregati, che compongono la molecola. Per facilitarne la comprensione, si riporta in figura 4.5 il procedimento introdotto poc'anzi:

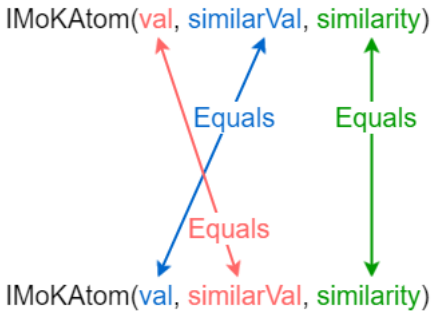


Figura 4.5: Esempio di validazione fra atomi.

In questo modo è possibile determinare se due atomi, contenuti in una molecola, sono stati aggregati in maniera parzialmente corretta. Per ottenere la correttezza completa, in base alle informazioni contenute nel dataset, è necessario verificare che *similarity* sia maggiore o uguale al valore 3 in quanto esso rappresenta la soglia oltre il quale una coppia di frasi, secondo *Gold Standard*, viene considerata equivalente.
Questa prima versione del *Validator* è stata testata richiamando due volte il metodo *printCompartmentState* a cui viene passato il compartimento sia prima che dopo l'aggregazione per verificare l'avvenuta perturbazione degli atomi contenuti in esso. In particolare questo metodo permette di ottenere, oltre alle informazioni sullo stato, una classifica degli atomi e molecole in

cui vengono conteggiati il numero di atomi che hanno valore di similarità (*similarity*) compreso entro una determinata fascia.
Questo ha permesso di analizzare sia la quantità che la qualità, in termini di similarità, degli atomi aggregati ed ha portato alla conclusione che il metodo di validazione introdotto finora risulta troppo restrittivo per il nostro caso di studio.
Dalla descrizione del dataset fornita all'inizio della sezione 4.2 si evince che la definizione del valore di similarità associato ad ogni coppia di frasi è stato distillato considerando la relazione **semantica** che esse condividono, ovvero se i loro significati combaciano, secondo determinati criteri dettati dal *Gold Standard*.
L'entità *Aggregator* utilizza l'algoritmo *Generalized Jaccard* che, come citato nella sezione 3.2, effettua un calcolo basato sulla rappresentazione in *n-grammi* delle frasi in input, il quale restituisce un valore di similarità **sintattica** e non considera in alcun modo la semantica delle due frasi.

Metodo alternativo di Validazione

Secondo i ragionamenti descritti poc'anzi si è pensato di perseguire un metodo di validazione alternativo che non consideri unicamente la similarità semantica di due frasi ma che permetta di effettuare una validazione più verosimile rispetto alle proprietà dell'algoritmo utilizzato.
In figura 4.6 viene mostrato un esempio di esecuzione del metodo alternativo di validazione in cui, partendo da una molecola contenente due atomi, viene effettuato il controllo atto a verificare se essi sono simili o meno.

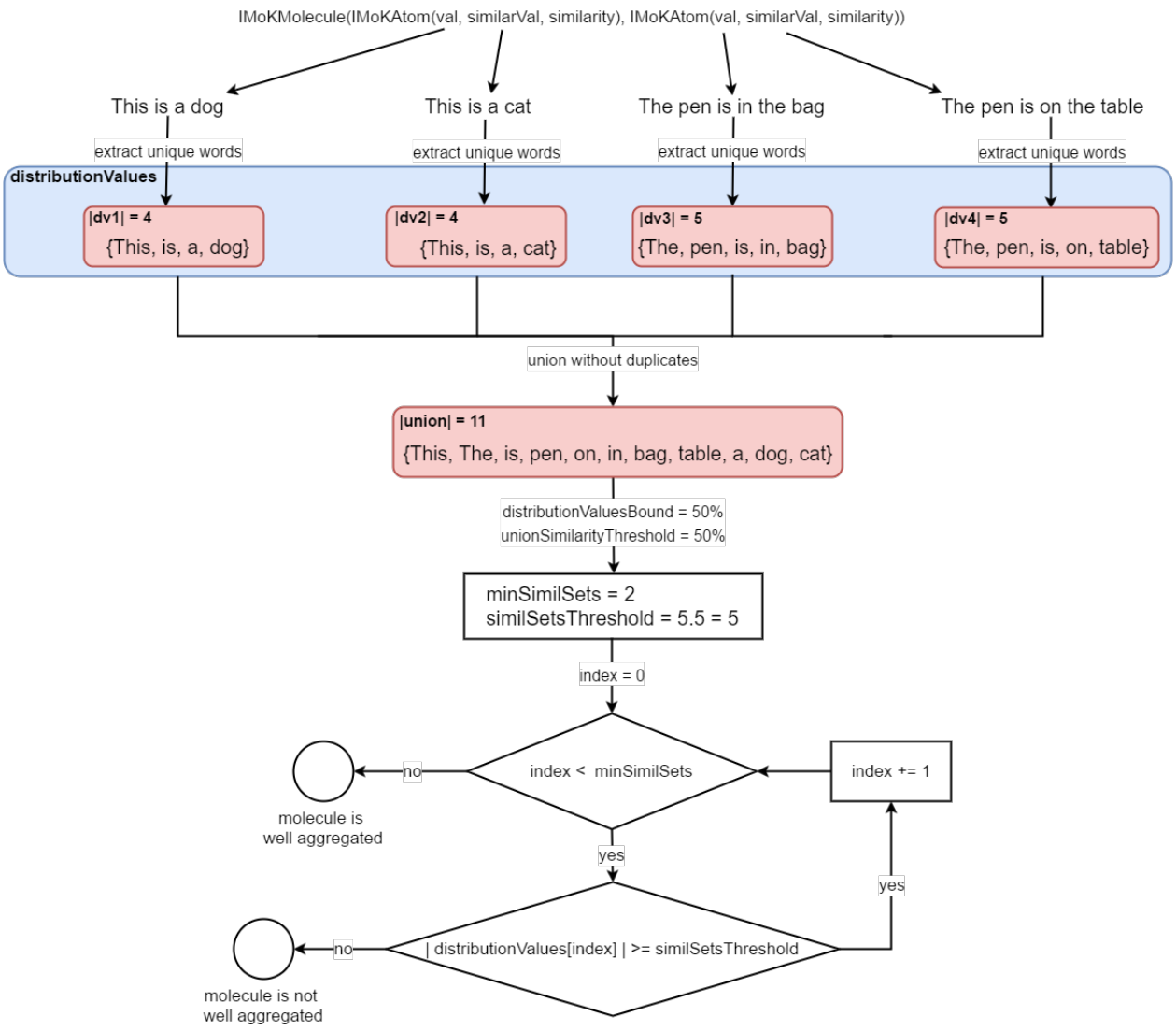


Figura 4.6: Esempio di validazione alternativa fra atomi.

Il metodo descritto in figura 4.6 sfrutta le proprietà *similarVal* e *val* di ogni atomo contenuto nella molecola in oggetto e due parametri caratteristici dell'algoritmo di validazione: *distributionValuesBound* e *unionSimilarityThreshold*.
Inizialmente viene estratto l'insieme di parole da *val* e *similarVal* di ogni atomo, senza duplicati. Partendo da questi viene poi costruito un insieme che ne contiene l'**unione**, senza duplicati.
La rimozione dei duplicati da questi insiemi permette di ottenere un concetto di similarità **sintattica** tra frasi, in quanto se le frasi di partenza contengono le stesse parole, allora gli insiemi di parole costituiti da esse e la loro unione

saranno circa equivalenti.
A questo punto risulta evidente che, se l'insieme unione contiene all'incirca gli stessi elementi di ciascuno dei *distributionValues* preso singolarmente, allora le frasi considerate si possono definire sintatticamente simili.
Per definire un algoritmo di validazione più flessibile rispetto al primo metodo, sono stati introdotti due parametri che vengono descritti nel dettaglio di seguito:

- *distributionValuesBound* : limita in percentuale il numero di elementi di *distributionValues* da considerare durante il controllo del *Validator*, determinando il *minSimilSets*;
- *unionSimilarityThreshold* : limita in percentuale la lunghezza dell'insieme unione da considerare durante il controllo del *Validator*, determinando la *similSetsThreshold*;

Una volta determinati questi due parametri il *Validator* verifica che la lunghezza di ciascuno dei *distributionValues* sia maggiore o uguale della *unionSimilarity*. Appena questo controllo fallisce il sistema etichetta la molecola in esame come **"non aggregata correttamente"**, se invece tutti i controlli vanno a buon fine e il ciclo termina, la molecola sarà etichettata con **"aggregata correttamente"**.
Un aspetto da non sottovalutare in questi controlli è il seguente: dato che viene limitato il numero di *distributionValues*, è necessario calibrare il loro ordinamento in modo da non considerare sempre i primi n elementi. Questo rappresenta un fattore importante che può cambiare completamente la scelta dell'etichetta da assegnare alla molecola. Se consideriamo l'esempio riportato in figura 4.6 risulta evidente che nel caso in cui i *distributionValues* scelti siano i primi due, l'algoritmo assegnerà l'etichetta **"non aggregata correttamente"** alla molecola, in quanto il primo elemento {*This, is, a, dog*} ha quattro elementi mentre la *unionSimilarity* è pari a cinque. Se invece i *distributionValues* scelti sono gli ultimi due, l'algoritmo assegnerà l'etichetta **"aggregata correttamente"** alla molecola. Per semplicità nei test che verranno presentati in seguito, è stato utilizzato il metodo *shuffle* della classe *Collections*[4] di Java che permette di mischiare casualmente gli insiemi contenuti in *distributionValues*.

[4] `https://docs.oracle.com/javase/7/docs/api/java/util/Collections.html`

Tuning dei parametri di validazione

L'obiettivo distillato dall'analisi sulla reazione di aggregazione MoK riguarda la massimizzazione del numero di atomi aggregati, anche se questi non sono stati aggregati correttamente. Per capire il perchè di questa dichiarazione basti pensare che, quando si ricerca un'informazione specifica attraverso *Google* vengono forniti anche risultati non inerenti, che generalmente non interessano all'utente. Per MoK vale lo stesso ragionamento, in quanto è stato pensato per presentare all'utente le informazioni contenute in un compartimento.
Questi concetti sono esprimibili attraverso le definizioni di **exploration** ed **exploitation**. Per **exploration** si intende quanti atomi l'*Aggregator System* riesce ad aggregare in base al totale di partenza, in sostanza si vuole aggregare il più possibile anche se gli atomi non risultano abbastanza simili. Per **exploitation** si intende quanti atomi l'*Aggregator System* riesce ad aggregare in modo corretto, in questo caso ci si aspetta un numero minore di aggregazioni rispetto all'**exploration** ma queste verrano effettuate per la maggior parte fra atomi simili.
Per individuare delle coppie di valori per i parametri <*distributionValuesBound, unionSimilarityThreshold*> che permettano di incentivare **exploration** o **exploitation** è necessario effettuare un tuning dei parametri stessi.
Inizialmente è stato messo in esecuzione l'*Aggregator System* con 15 file contenenti un numero variabile di righe, dalle 400 alle 700, formattate come descritto nella sottosezione **Scelta del Dataset**.
Per ogni esecuzione, il *Validator* ha generato un file da identificarsi come *nomefile_results.txt* che al suo interno riporta i risultati di aggregazione come ad esempio il numero di atomi aggregati, la percentuale di atomi aggregati correttamente, lo stato del compartimento prima e dopo l'aggregazione e tutte le ulteriori informazioni.
Analizzando i file generati è stato costruito un primo grafico, mostrato in figura 4.7, che mostra la percentuale di atomi aggregati rispetto al totale di quelli presenti inizialmente nel compartimento:

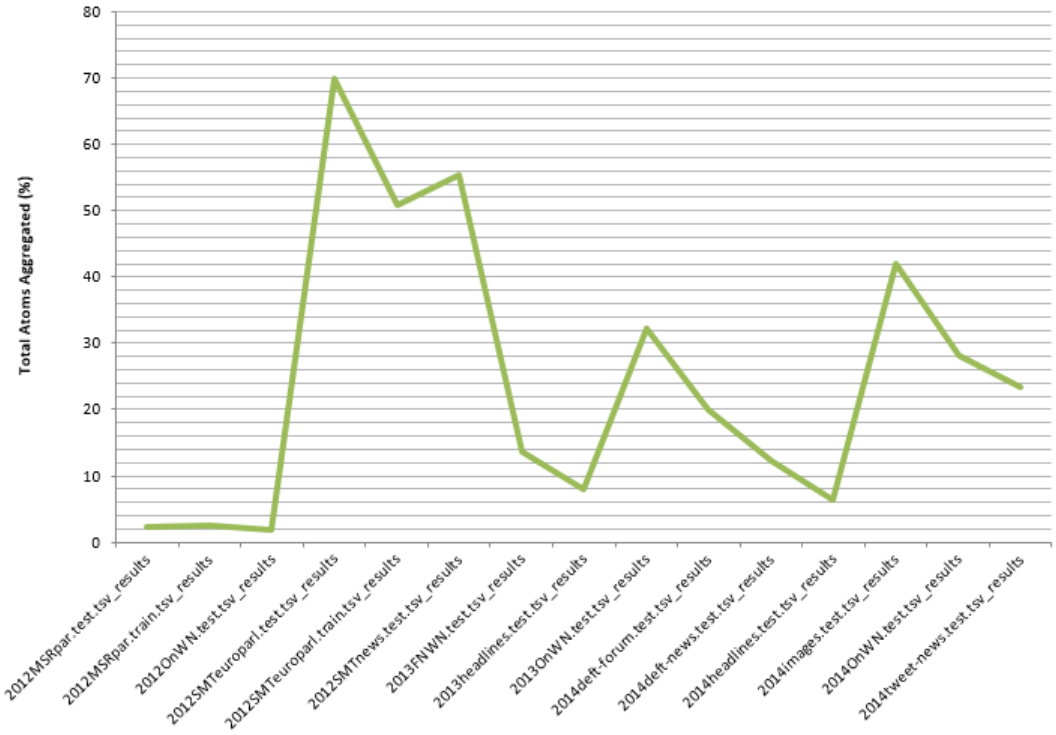


Figura 4.7: Percentuale di atomi aggregati per ogni file del dataset.

Da questa prima analisi risulta evidende la presenza di casi in cui l'aggregazione **ha aggregato molti atomi** oppure **ha aggregato pochi atomi** rispetto al totale di quelli contenuti nel compartimento. Si vuole ricordare che l'attuale implementazione dell'*AtomAggregator* utilizza l'algoritmo *Generalized Jaccard* e la soglia di similarità pari a 0,8792 il cui valore è stato determinato valido per le frasi.
Basandosi sui concetti appena introdotti viene mostrato in figura 4.8 un grafico che riporta i risultati di un'ulteriore test, calcolati al termine del processo di validazione, in cui è stato messo in esecuzione il sistema con i 15 file del dataset considerando diverse coppie di valori per i parametri *distributionValuesBound* e *unionSimilarityThreshold*.

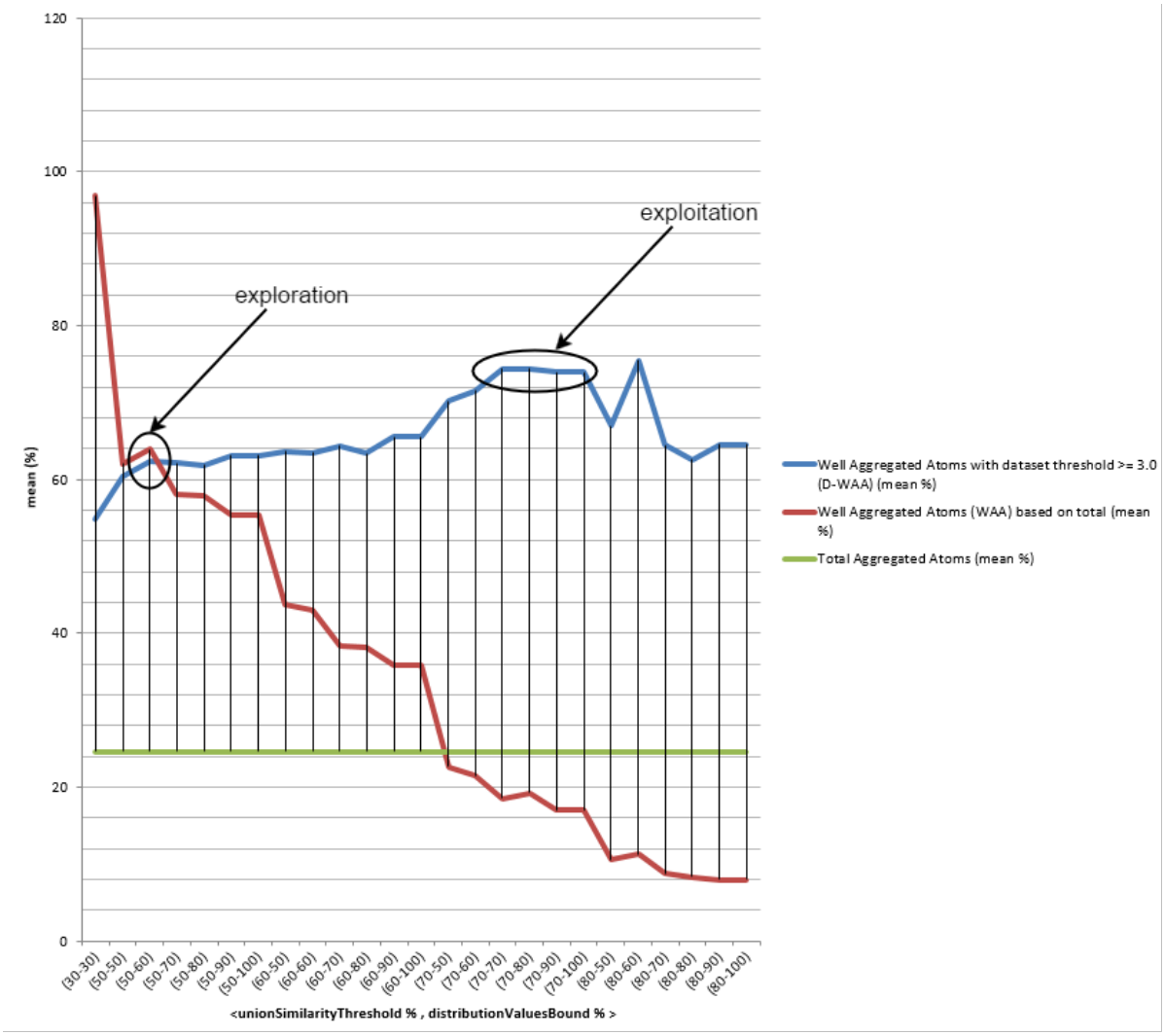


Figura 4.8: Test per il tuning dei parametri, exploration vs. exploitation.

Nel grafico mostrato in figura 4.8 sono presenti i seguenti dati:

- Gli atomi correttamente aggregati con valore di *similarity* $>= 3$: rappresenta la percentuale media di atomi che hanno valore di *similarity* $>= 3$ calcolata sul totale di atomi aggregati correttamente (successivamente in questo documento ci si riferirà a questa misura con *D-WAA*);
- Gli atomi correttamente aggregati: rappresenta la percentuale media di atomi aggregati correttamente rispetto a quelli aggregati in totale (successivamente in questo documento ci si riferirà a questa misura con *WAA*);

- Gli atomi aggregati: rappresenta la percentuale media di atomi aggregati rispetto al totale contenuto nel compartimento;

Si vuole sottolineare che nelle ascisse del grafico in figura 4.8 sono riportate tutte le coppie di valori in percentuale di <*unionSimilarityThreshold, distributionValuesBound*>, le quali sono state utilizzate per ricavare il risultato di ogni singola esecuzione.
Inoltre sono state marcate due parti in cui risulta evidente la distinzione fra **exploration** ed **exploitation**, nel primo caso il grafico mostra una percentuale più bassa di *D-WAA* rispetto agli *WAA*, mentre nel caso di **exploitation** vengono evidenziate le coppie di valori che permettono di ottenere una percentuale bassa di *WAA* e una percentuale molto alta di *D-WAA*.
Riassumendo, è evidente che al diminuire in percentuale dei due parametri di validazione, l'**exploration** viene accentuata, al contrario al loro aumentare si accentua l'**exploitation**.
Tali conclusioni non sono da ritenersi come definitive, in quanto analizzando il grafico emerge che i risultati ottenuti dipendono fortemente dalla natura del dataset considerato. Per ridurre queste incertezze verrà affrontata di seguito una breve analisi su due coppie di valori estrapolate dal grafico in figura 4.8: *exploration : <unionSimilarityThreshold = 50%, distributionValuesBound = 60%>* e *exploitation : <unionSimilarityThreshold = 70%, distributionValuesBound = 80%>*.

Exploration vs. Exploitation

Considerando le due coppie di valori identificate poc'anzi, una per l'*exploration* e l'altra per l'*exploitation*, è stata condotta un'indagine più in dettaglio su tre dataset per coppia che hanno generato risultati incoerenti in fase di validazione:

- **<unionSimilarityThreshold = 50%, distributionValuesBound = 60%>** : per questa coppia sono stati considerati i risultati di validazione contenuti nei file: *2012SMTnews.test.tsv_results*, *2012SMTeuroparl.test.tsv_results* e *2012SMTeuroparl.train.tsv_results*. Come mostrato nel grafico in figura 4.8 la percentuale media di *WAA* coincide circa con quella relativa agli *D-WAA*. Per i tre specifici casi considerati è stata riscontrata un'anomalia che non segue la regola determinata dal grafico, bensì si hanno entrambe le percentuali di aggregazione con valore superiore al 90%, in sostanza si riscontra che l'*Aggregator* ha aggregato **molti atomi correttamente** mentre ci si aspettava di avere **molti atomi aggregati in maniera errata**.

Analizzando le informazioni contenute nel compartimento prima e dopo l'aggregazione, in particolare le proprietà *val* e *similarVal* di ogni atomo, è emersa la presenza di molte coppie di frasi (righe del dataset) che sono sintatticamente equivalenti e di conseguenza risulta più semplice aggregarle in maniera corretta;

- <**unionSimilarityThreshold = 70%, distributionValuesBound = 80%**> : per questa coppia sono stati considerati i risultati di validazione contenuti nei file: *2012MSRpar.test.tsv_results*, *2013OnWN.test.tsv_results* e *2014deft-forum.test.tsv_results*. Come mostrato nel grafico in figura 4.8 la percentuale media di *D-WAA* supera di molto quella degli *WAA*. Analizzando i risultati di validazione, contenuti nei tre file precedentemente citati, è emersa la presenza di casi in cui entrambe le percentuali hanno un valore poco superiore al 30%, in sostanza l'*Aggregator* ha aggregato **pochi atomi e in maniera errata** mentre ci si aspettava di avere **pochi atomi aggregati correttamente**. Anche in questo caso l'anomalia è dovuta alle frasi contenute nel dataset, la cui analisi ha evidenziato l'esistenza di atomi con *val* duplicati che vengono associati a frasi (*similarVal*) diverse. Per tale motivo l'*Aggregator* aggrega gli atomi con *val* equivalente ma il *Validator* etichetta, in maniera erronea, la molecola come *"non aggregata correttamente"*. Di seguito viene riportata una molecola che, dopo l'aggregazione, presenta l'anomalia:

```
MoKMolecule[
  MokAtom[
    Similarity(1.0) ,
    Val(i don't want a president who 'cares'.) ,
    SimilarVal(i don't want a president who is charasmatic.)],
  MokAtom[
    Similarity(4.6) ,
    Val(i don't want a president who 'cares'.) ,
    SimilarVal(don't want a president that is compassionate.)]
]
```

In questo caso il *Validator* effettua una validazione pessimistica in quanto i *val* corrispondenti ai due atomi sono uguali, mentre i *similarVal* differiscono. Una possibile soluzione è escludere dall'insieme di *distributionValues* tutti gli elementi derivati da frasi contenute in atomi che hanno *similarity*< 3. Potrebbe essere parte di uno sviluppo futuro migliorare il *Validator* introducendo i controlli appena specificati;

Dalle considerazioni emerse finora si evince che i risultati ottenuti dal nuovo metodo di validazione dipendono fortemente dalla natura dello specifico

dataset. Una possibile soluzione per ridurre al minimo la presenza di queste anomalie è quella di considerare un numero maggiore possibile di dataset. Per coerenza di analisi tra i test effettuati finora e quelli successivi, in questo elaborato verrà considerato lo stesso insieme di file appartenenti al dataset.

Aggregare atomi in molecole

Questo passo di aggregazione prevede l'eventuale aggiunta di atomi a molecole già esistenti in un dato compartimento. Dato un atomo non aggregato si cerca una molecola fra quelle esistenti nel compartimento che risulti abbastanza simile da poter aggregare l'atomo in essa formando una molecola più grande. Il concetto di similarità in questo caso può essere interpretato in diversi modi, ad esempio si può pensare di aggregare un atomo in una molecola se il valore di similarità, calcolato con *Generalized Jaccard*, fra esso ed **ognuno** degli atomi contenuti nella molecola supera la soglia.
Dall'analisi affrontata precedentemente è emerso che la reazione di aggregazione in MoK necessita di incentivare l'**exploration** mentre il metodo per definire la similarità fra un atomo ed una molecola appena introdotto risulta più adatto ad incentivare l'**exploitation**.
Un metodo più flessibile che può portare ad ottenere risultati meno precisi consiste nel cercare, dato un atomo X da aggregare, almeno un atomo Y appartenente alla molecola in esame che rispetti la seguente condizione : *similarity(X,Y) >= similarityThreshold*. I risultati ottenuti dall'implementazione di quest'ultimo verranno analizzati nel dettaglio in seguito dopo aver effettuato la verifica dei parametri di validazione.

Verifica dei parametri di validazione

Per verificare la validità dei valori scelti durante il tuning dei parametri di validazione è stato eseguito l'*Aggregator System* con gli stessi 15 file considerati per il primo passo di aggregazione, in particolare le due coppie di valori scelte sono : *exploration : <unionSimilarityThreshold = 50%, distributionValuesBound = 60%>* e *exploitation : <unionSimilarityThreshold = 70%, distributionValuesBound = 80%>*. Inoltre si è deciso di configurare il sistema in modo tale da eseguire una iniziale aggregazione fra atomi e far si che l'aggregazione di atomi in molecole venga effettuata su un compartimento con alcune molecole già presenti.
Di seguito, in figura 4.9, vengono mostrati i risultati di esecuzione del sistema dove il *Validator* è stato configurato con diverse coppie di valori per i parametri di validazione scelti nell'intorno delle due coppie identificate per **exploration** e **exploitation**:

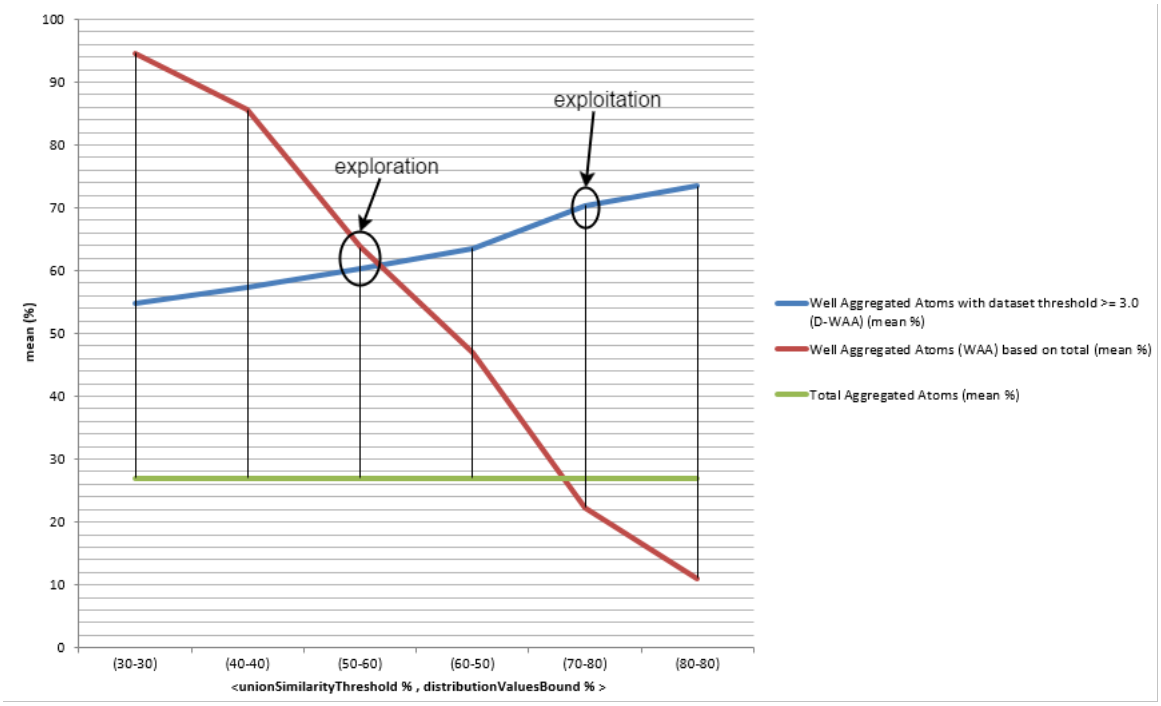


Figura 4.9: Test per la verifica dei parametri, exploration vs. exploitation.

Si evince una somiglianza tra il grafico mostrato in figura 4.9 e quello relativo al passo di aggregazione illustrato in precedenza 4.8, e di conseguenza tra i rispettivi risultati di validazione.
Come previsto emerge l'aumentare della percentuale media di atomi aggregati che in questo caso si aggira intorno al 27%, mentre per il passo precedente risultava intorno al 25%.

Aggregare molecole

L'implementazione del terzo ed ultimo passo, prevede l'aggregazione fra le sole molecole presenti in un dato compartimento. Questa operazione porta alla costruzione di una molecola contenente tutti gli atomi appartenenti alle due molecole considerate simili dal sistema.
Anche in questo caso, si è deciso di adottare un metodo che favorisce l'**exploration** in cui date due molecole M1 e M2, queste vengono aggregate se esistono due atomi A1 $\in$ M1 e A2 $\in$ M2 che rispettano la seguente condizione: *similarity(A1,A2) >= similarityThreshold*.
Questa verifica in fase di aggregazione porta a ridurre notevolmente i tempi di esecuzione, in media se le due molecole in oggetto sono simili dovranno essere controllati solo metà degli atomi contenuti in ciascuna di esse per poterle aggregare. Inoltre la validità di questo metodo viene confermata dalla proprietà transitiva la quale può essere applicata ai singoli atomi:

$$\begin{aligned} similarity(atomA, atomB) >= similarityThreshold \wedge \\ similarity(atomB, atomC) >= similarityThreshold \rightarrow \\ similarity(atomA, atomC) >= similarityThreshold \end{aligned} \tag{4.5}$$

considerando che le molecole sono insiemi di atomi, risulta immediata l'estensione di tale proprietà anche ad esse.

Verifica dei parametri di validazione

Anche in questo caso, come descritto nella verifica dei parametri di validazione per il passo precedente, è stato eseguito l'*Aggregator System* con i 15 file appartenenti al dataset considerati per il primo passo di aggregazione. Inoltre si è deciso di configurare il sistema in modo tale da eseguire i primi due passi in sequenza e far si che l'aggregazione fra molecole venga effettuata su un compartimento con alcune molecole già presenti.
Di seguito, in figura 4.10, vengono mostrati i risultati di esecuzione del sistema, il quale è stato configurato con le stesse coppie di valori individuate nel grafico relativo al passo precedente 4.9 :

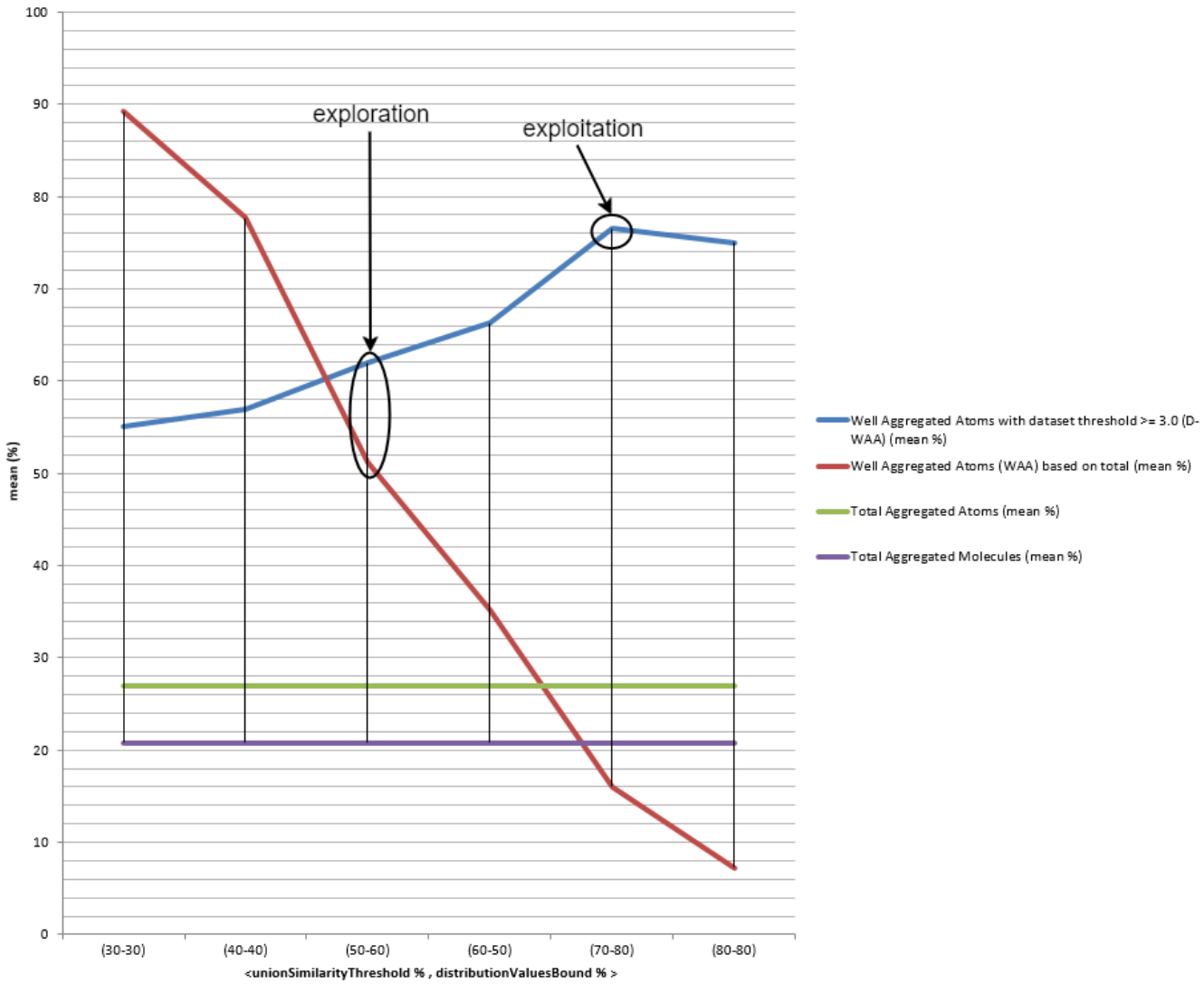


Figura 4.10: Test per la verifica dei parametri, exploration vs. exploitation.

In questo caso i risultati di validazione, mostrati in figura 4.10, differiscono leggermente rispetto ai passi precedenti. Nell'intervallo corrispondente alla coppia di valori $<unionSimilarityThreshold = 50\%, distributionValuesBound = 60\%>$ si riscontra una percentuale media di *D-WAA* maggiore rispetto a quella di *WAA*. Da ciò si deduce che, a differenza degli altri passi, il grado di **exploration** diminuisce in quanto la percentuale media di *WAA* si riduce di circa il 10%.
Un'informazione aggiuntiva che si evince dal grafico è il fatto che la percentuale degli atomi aggregati in totale risulta stazionaria rispetto al passo

precedente.

4.3 Approccio semantico: Hermit

Partendo dai modelli di *Aggregator System* e *Model* mostrati nella sezione 4.1 viene di seguito specializzato il modello MoK semplificato, in particolare l'interfaccia *IHermitMoKAtom*, con l'obiettivo di implementare una versione del sistema che sfrutta *Hermit* per aggregare la conoscenza.

Scelta del Dataset

A differenza dell'approccio *non-semantico*, non essendoci vincoli particolari nella scelta del tipo di dato da aggregare (parole, frasi o documenti), si è deciso di riutilizzare buona parte del lavoro svolto nel capitolo 3 riguardante i test specifici su *Hermit*, in cui il dataset utilizzato è il database di *WordNet*[5]. Si vuole specificare che all'interno del database sono presenti diverse *Class* appartenenti all'ontologia *WordNet* ma in questo caso, per implementare l'aggregazione ed effettuare dei test di validazione a posteriori, verrà considerata solamente una relazione semantica: la *Sinonimia*. Partendo da quest'idea risulta necessaria la presenza dell'ontologia *WordNet*, la quale verrà creata e comunicata alle altre entità del sistema attraverso il *Manager*.
Il caricamento dell'ontologia avviene come specificato nella sezione 3.3, ma dato che questa operazione prevede solo la creazione della struttura, emerge la necessità di introdurre degli *Individuals* creati a partire dal database citato in precedenza.

Mapping del Dataset in Entità del Modello

Come specificato poc'anzi si è pensato di utilizzare le classi *Word* e *Synset* dell'ontologia *WordNet* per effettuare l'aggregazione e permettere una validazione a posteriori.
A tale scopo si è deciso di generare un'entità *IHermitMoKAtom* per ogni *Word* contenuta nell'ontologia e di conseguenza modellare la struttura di tale entità come rappresentato in figura 4.11:

[5] `https://wordnet.princeton.edu/wordnet/download/current-version/`

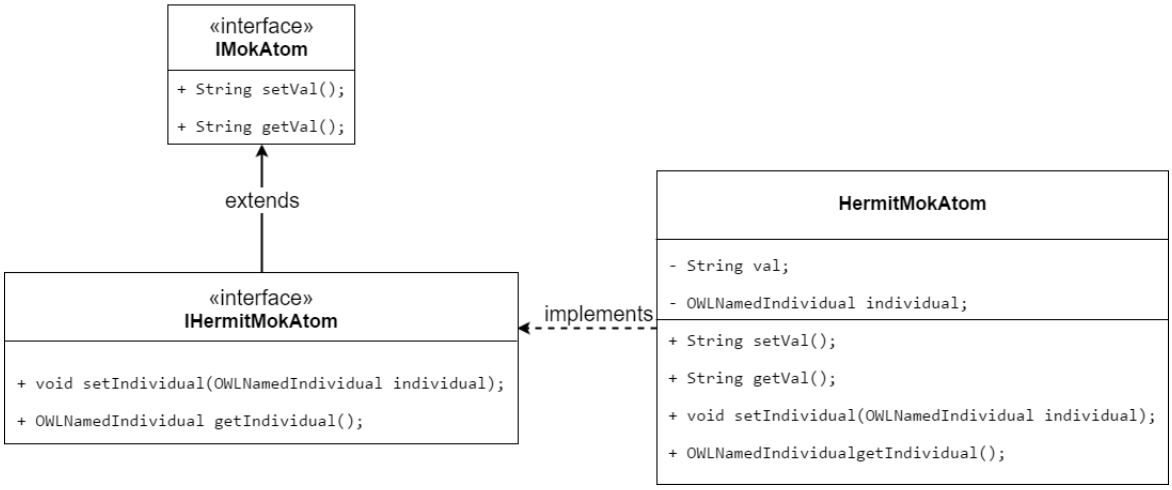


Figura 4.11: Diagramma C4 *Class*, zooming sull'entità *Model*.

In figura 4.11 rispetto all'entità IMokAtom, viene aggiunta la proprietà *individual* il cui significato è specificato di seguito:

- *Val* : rappresenta la conoscenza contenuta nell'atomo, che in questo caso è data da una singola parola in formato testuale;
- *individual* : è la rappresentazione di *val* nel contesto dell'ontologia *WordNet*, ovvero un'instanza della classe *Word*.

Implementazione dell'*Aggregator System* per Hermit

In seguito all'analisi effettuata finora è emerso che l'unica entità a dover essere specializzata è l'atomo, pertanto si è deciso di riutilizzare totalmente la struttura del sistema di aggregazione presentato nella sezione 4.2 in figura 4.4 e definire implementazioni differenti per ogni specifico passo di aggregazione, che di seguito saranno descritti in dettaglio.

Aggregazione fra Atomi

L'implementazione del primo passo di aggregazione prevede il tentativo di aggregare coppie di entità *IHermitMokAtom* in cui le parole, contenute nelle rispettive proprietà *val*, siano *sinonimi*. La relazione di *Sinonimia* in *WordNet* è rappresentata dal concetto di *Synset*, il quale non è altro che un insieme di parole che sono sinonimi tra loro. Come specificato in figura 4.11 un *IHermitMokAtom* contiene sia la parola in formato testuale che l'*individual* corrispondente ad essa, il quale ha una proprietà *sense* che ne determina il

significato.
Grazie ad un'analisi sui dati presenti nel database *WordNet*, caricati nell'ontologia dal *Manager*, è emerso che un *individual* di classe *Word* ha lo stesso significato dell'*individual* di classe *Synset* che lo contiene.
Pertanto si è deciso di aggregare una coppia atomi se e solo se i loro significati sono equivalenti, di conseguenza si presume che nel database *WordNet* sia presente un insieme di sinonimi che contiene entrambe le parole associate alle proprietà *val* degli stessi.
Un metodo alternativo a quello appena citato consiste nel trovare un insieme di sinonimi che contiene entrambe le parole relative ad una coppia di atomi. Questa operazione tuttavia risulta essere più onerosa in termini di perfomance.

Aggregare atomi in molecole

Come descritto per l'approccio *non-semantico* questo passo di aggregazione prevede l'eventuale aggiunta di atomi a molecole già esistenti in un dato compartimento. Dato un atomo non aggregato si cerca una molecola fra quelle esistenti nel compartimento i cui i atomi rappresentino parole che sono sinonimi della proprietà *val* relativa all'atomo da aggregare.
Avendo come obiettivo quello di incentivare l'*exploration*, ovvero massimizzare la quantità di atomi aggregati a discapito della precisione, si è deciso di implementare questo passo di aggregazione seguendo la stessa logica descritta nella sezione 4.2, in cui un atomo viene aggregato ad una molecola se esiste almeno una coppia <atomToAggregate, moleculeAtom>, dove moleculeAtom è un atomo appartenente alla molecola, che rispetti la seguente condizione:

$$\mathrm{sense}(\mathrm{atomToAggregate}) = \mathrm{sense}(\mathrm{moleculeAtom})$$

I *sense* di ogni atomo sono in formato testuale e vengono recuperati attraverso le proprietà *individual* relative a ciascun *IHermitMokAtom*.

Aggregare molecole

Il terzo e ultimo passo di aggregazione, come descritto nella sezione 4.2, prevede l'aggregazione fra le molecole presenti in un dato compartimento. Tale operazione, a partire da due molecole M1 e M2, ne crea una nuova contenente tutti gli atomi appartenenti a M1 e M2 se esiste almeno una coppia di atomi *<A1, A2>* dove $A1 \in M1$ e $A2 \in M2$ che soddisfa la seguente condizione:

$$\mathrm{sense}(A1) = \mathrm{sense}(A2)$$

Quella appena descritta rappresenta l'unica condizione necessaria e sufficiente affinchè avvenga l'aggregazione fra molecole. Si vuole sottolineare che anche in questo caso, come nel passo precedente, tale condizione permette di incentivare l'*exploration*.

Validazione dei passi di Aggregazione

La validazione dei passi di aggregazione prevede il controllo di tutte le molecole contenute nel compartimento per verificare che siano state aggregate correttamente o meno. Per raggiungere tale obiettivo si utilizza la relazione semantica di sinonimia messa a disposizione dell'ontologia *WordNet*. In particolare data una molecola si cerca un *Synset* come descritto di seguito:

$$\exists \quad S : \left(\bigcup_{i=1}^{n} \quad a_i \in M \quad | \quad val(a_i) \right) \subseteq S \tag{4.6}$$

Dove S è un insieme di sinonimi e a_i rappresenta l'atomo i-esimo appartenente alla molecola M. Se viene individuato nell'ontologia un *individual* di classe *Synset* che rispetta tali proprietà allora la molecola viene etichettata come **"aggregata correttamente"**, se invece non viene individuato alcun *Synset* verrà etichettata come **"non aggregata correttamente"**.
Si vuole sottolineare che, come descritto in precedenza, il sistema si basa solo sulla relazione di *Sinonimia* e nel database di *WordNet* possono essere presenti *individual* di classe *Word* che non appartengono ad un *individual* di classe *Synset*. Pertanto al termine dell'aggregazione è possibile che nel compartimento siano presenti parole aggregate correttamente non appartenenti ad alcun insieme di sinonimi. Di conseguenza il *Validator* etichetterà, in maniera erronea, come **"non aggregata correttamente"** la molecola in cui tali parole sono contenute. Per evitare questa anomalia si è pensato di caricare nell'ontologia unicamente parole appartenenti ad un insieme di sinonimi, in questo modo se il *Validator* etichetta una molecola come **"aggregata correttamente"** significa che tutte le parole contenute in essa appartengono allo stesso *Synset*.
Una volta individuato il metodo di validazione descritto dall'equazione 4.6 si è deciso di implementare l'entità *Validator* come definito in figura 4.4 e di utilizzarla per visualizzare lo stato del compartimento prima e dopo l'aggregazione ed effettuare la validazione di quest'ultima.
Durante i test di validazione, effettuati eseguendo il sistema, è stato riscontrato che il *Validator* etichetta tutte le molecole presenti nel compartimento con **"aggregata correttamente"**. Questi risultati sono dovuti alla relazione stretta tra il metodo di aggregazione e quello di validazione, nonchè

alla natura dell'ontologia che fornisce direttamente la rappresentazione della relazione di equivalenza **semantica** necessaria per aggregare le parole.

Aggregazione di frasi con Hermit

Dai risultati ottenuti finora è emerso che la validazione non permette di verificare in modo significativo la correttezza del processo di aggregazione dell'*Aggregator System* per *Hermit*, inoltre non rende possibile confrontare le implementazioni dei due approcci *semantico* e *non-semantico*.
Di conseguenza si è deciso di implementare un'ulteriore versione di tale sistema per aggregare frasi, potendo così utilizzare le considerazioni fatte nella sezione 4.2. In tale contesto si è scelto un dataset composto da file testuali ottenuti da *Semantic Text Similarity Dataset Hub* e successivamente è stata modellata la struttura dell'*IMoKAtom*, mostrata in figura 4.3, che è stato possibile riutilizzare anche per l'implementazione dell'approccio *semantico*. Da queste considerazioni emerge la possibilità di riutilizzare lo stesso metodo di validazione progettato per l'approccio *non-semantico* mostrato in figura 4.6.

Algoritmo di Aggregazione di frasi per Hermit

Diversamente dalle parole, per le frasi non è stata individuata alcuna ontologia che possa rappresentare in maniera completa le relazioni semantiche descritte nel capitolo 3. Pertanto si è deciso di sfruttare l'ontologia *WordNet* e il dataset individuato in precedenza per realizzare un algoritmo in grado di aggregare frasi.
In figura 4.12 viene mostrato il comportamento di tale algoritmo attraverso un esempio di esecuzione che, partendo da una coppia di atomi, si occupa di verificare se questi possono essere aggregati o meno.

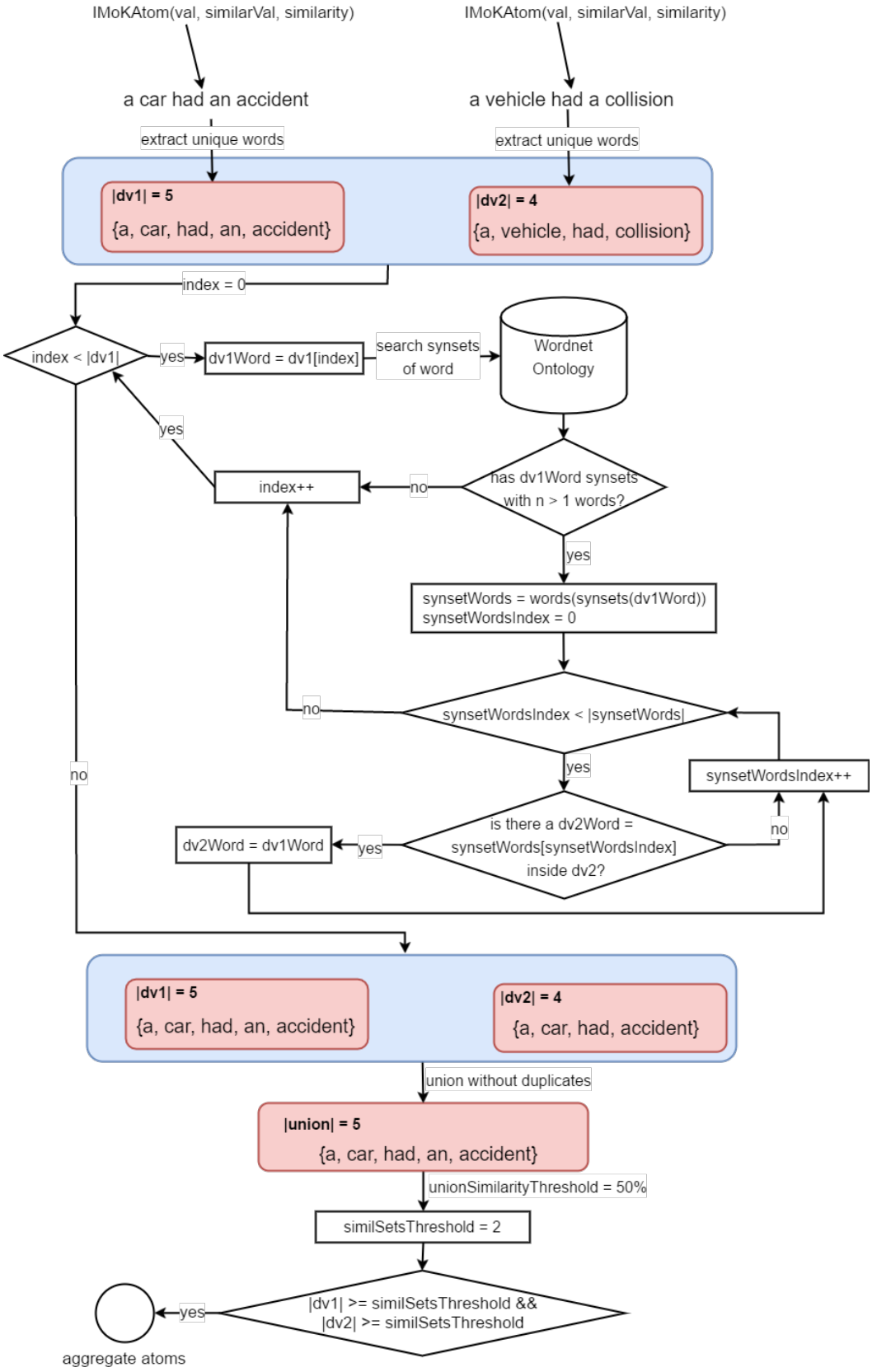


Figura 4.12: Algoritmo di Aggregazione di frasi per *Hermit*.

Come mostrato in figura 4.12, data una coppia di atomi, l'algoritmo di aggregazione procede estraendo inizialmente due insiemi di parole dalle proprietà *val* di ogni atomo. Poichè tali insiemi sono ricavati da frasi, è possibile che tra gli elementi contenuti in essi vi siano dei sinonimi. Non conoscendo a priori la posizione di questi ultimi, l'algoritmo cerca, per ogni parola *dv1Word* contenuta in *dv1*, tutti gli insiemi di sinonimi in cui essa compare. Questo passo viene effettuato sfruttando l'ontologia *WordNet* in cui sono stati caricati in precedenza un insieme di *Synset*.
In seguito a questa ricerca si possono avere tre casi:

- Caso 1: Non viene trovato alcun insieme di sinonimi in cui *dv1Word* è contenuta;
- Caso 2: Vengono trovati uno o più insiemi di sinonimi in cui *dv1Word* è l'unico elemento presente;
- Caso 3: Vengono trovati uno o più insiemi di sinonimi con più elementi tra i quali è presente *dv1Word*;

Se si presenta il primo o il secondo caso l'algoritmo non effettua alcuna operazione e passa ad esaminare la parola successiva contenuta nell'insieme *dv1*, altrimenti se a presentarsi è il terzo caso viene cercata una corrispondenza fra una qualsiasi parola dell'insieme *dv2*, chiamata *dv2Word*, e una qualsiasi parola dell'insieme *synsetWords*, il quale contiene tutti i sinonimi di *dv1Word*, compresa essa stessa, trovati nell'ontologia. Per ogni match di *dv2Word* e *synsetWords[synsetWordsIndex]* l'algoritmo sostituisce nell'insieme *dv2* la parola *dv2Word* con il sinonimo *dv1Word* rendendo gli insiemi *dv1* e *dv2* più simili fra loro. Tale operazione, eseguita per ogni parola dell'insieme *dv1*, permette di ottenere un'insieme *unione* che contiene solo le parole uniche, ricavate dalle due frasi di partenza, senza duplicati e sinonimi.
Una volta calcolato l'insieme *unione* il parametro *unionSimilarityThreshold* ne limita in percentuale la lunghezza determinando *similSetsThreshold*, il quale rappresenta la soglia di lunghezza minima che gli insiemi *dv1* e *dv2* devono rispettare affinchè i due atomi possano essere aggregati.
Ci si aspetta quindi che all'aumentare del parametro *unionSimilarityThreshold* l'algoritmo aggreghi due atomi solo se tra le frasi, contenute nelle proprietà *val* degli stessi, vi è un numero alto di parole uguali o sinonime. Questo comporterà la diminuzione del numero di aggregazioni effettuate dal sistema.

Risultati dell'aggregazione di frasi

L'algoritmo di aggregazione introdotto poc'anzi sfrutta la relazione di sinonimia, rappresentata nell'ontologia *WordNet*, per aumentare l'efficacia del controllo **sintattico** fra frasi, specificato in figura 4.12.
Si è deciso di affrontare l'implementazione dei tre passi, adottando la stessa logica di aggregazione di atomi, atomi in molecole e molecole introdotta nella sezione 4.2, in cui però viene utilizzato l'algoritmo sopra descritto per decidere se due atomi, un atomo e una molecola o due molecole devono essere aggregate o meno. Rispetto all'approccio *non-semantico*, questa implementazione è più onerosa computazionalmente in quanto richiede la costruzione di strutture dati, che rappresentino gli insiemi di parole *dv1* e *dv2*, e la ricerca di insiemi di sinonimi nell'ontologia.
Per ottimizzare le performance si è deciso pertanto di introdurre due cache: la prima contenente gli insiemi di parole corrispondenti alle frasi già analizzate e la seconda contenente gli insiemi di sinonimi corrispondenti alle parole già analizzate. In tal modo è possibile evitare di generare più volte le stesse strutture dati a fronte di atomi già visitati dall'algoritmo.
Di seguito vengono riportati i risultati di validazione dell'*Aggregator System* per aggregare frasi, implementato utilizzando *Hermit*, in cui il *Validator* è stato configurato con i parametri per l'**exploration**: *<unionSimilarityThreshold = 50%, distributionValuesBound = 60%>*, individuati nella sezione 4.2. In questi test sono stati caricati 10000 *Synset* nell'ontologia, ricavati dal database *Wordnet*, ed è stato eseguito il sistema utilizzando gli stessi file mostrati in figura 4.7 per ognuno dei passi di aggregazione.

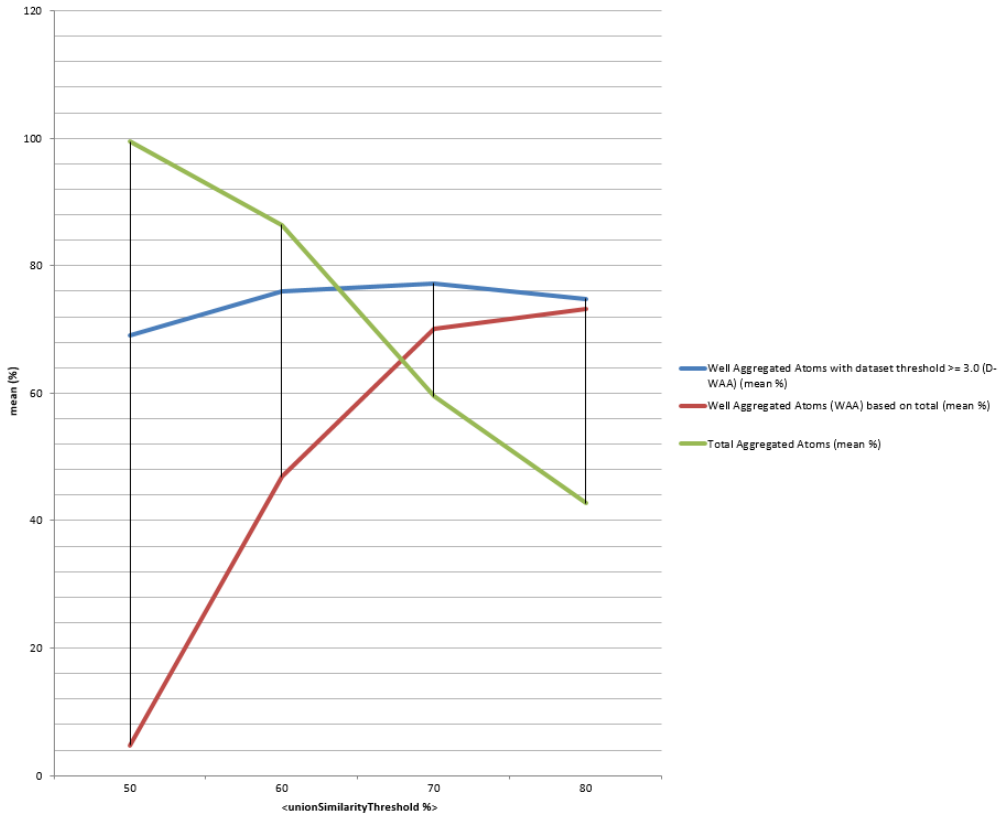


Figura 4.13: Primo passo: Percentuale di atomi aggregati in media e test di validazione.

Il grafico in figura 4.13 mostra l'esecuzione del primo passo di aggregazione e, per completezza, si riporta la descrizione dei dati, ottenuti in seguito all'esecuzione del *Validator* progettato per l'approccio *non-semantico*:

- Gli atomi correttamente aggregati con valore di *similarity* $>= 3$: rappresenta la percentuale media di atomi che hanno valore di *similarity* $>= 3$ calcolata sul totale di atomi aggregati correttamente (successivamente in questo documento ci si riferirà a questa misura con *D-WAA*);
- Gli atomi correttamente aggregati: rappresenta la percentuale media di atomi aggregati correttamente rispetto a quelli aggregati in totale (successivamente in questo documento ci si riferirà a questa misura con *WAA*);
- Gli atomi aggregati: rappresenta la percentuale media di atomi aggregati rispetto al totale contenuto nel compartimento;

In particolare nell'asse delle ascisse vengono riportati i valori in percentuale del parametro *unionSimilarityThreshold* considerati in questo test e, come previsto, all'aumentare del suo valore corrisponde una diminuzione della percentuale di atomi aggregati.
Si vuole sottolineare che nel caso in cui venga scelto un valore basso per il parametro *unionSimilarityThreshold*, ad esempio il 50%, il sistema aggrega una percentuale molto elevata di atomi in totale, ma la percentuale dei *WAA* risulta molto bassa. In caso contrario all'aumentare di tale valore diminuisce il numero di atomi aggregati in totale e la percentuale dei *WAA* aumenta, convergendo con quella dei *D-WAA*. Da queste considerazioni emerge che nel primo caso viene incentivata l'**exploration**, mentre nel secondo viene incentivata l'**exploitation**.

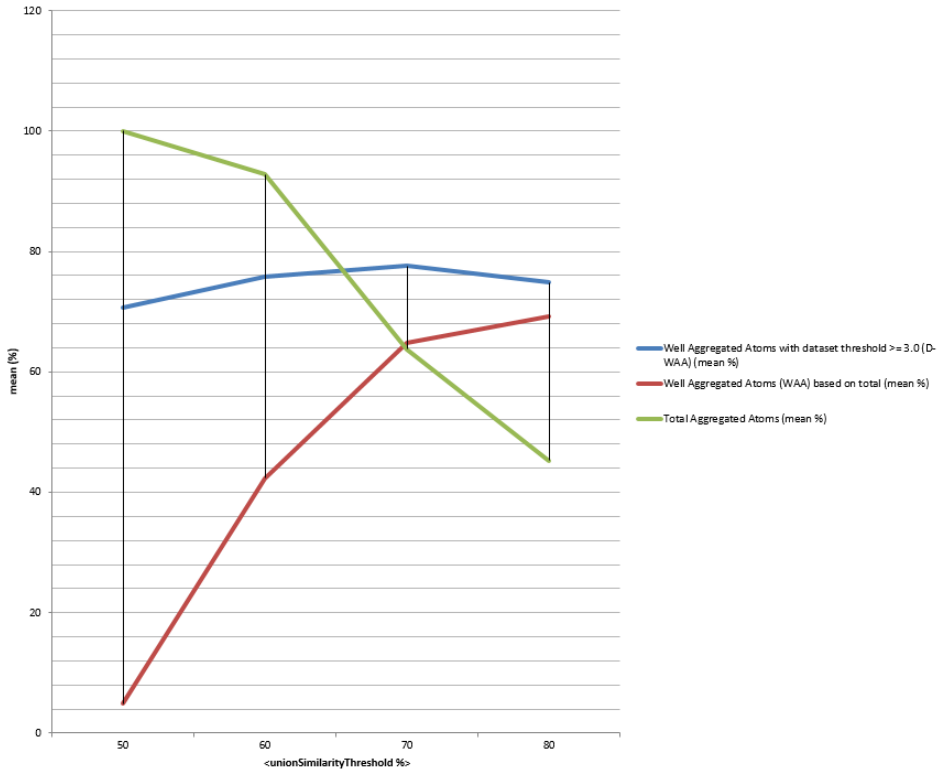


Figura 4.14: Secondo passo: Percentuale di atomi aggregati in media e test di validazione.

Dal grafico in figura 4.14 si evince, come previsto, un leggero aumento sull'andamento della percentuale di atomi aggregati rispetto al grafico in figura 4.13.

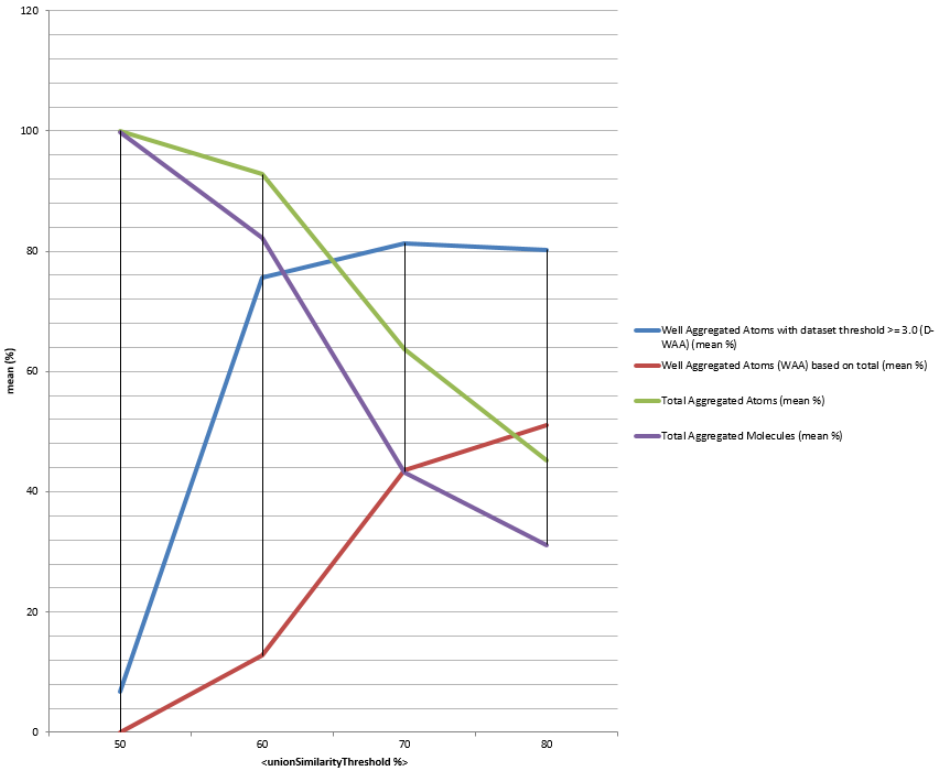


Figura 4.15: Terzo passo: Percentuale di atomi e molecole aggregati in media e test di validazione.

In figura 4.15 viene mostrato il grafico che riporta i risultati di validazione del terzo ed ultimo passo di aggregazione, in cui si evidenzia l'andamento della percentuale di atomi e molecole aggregate al variare del parametro *unionSimilarityThreshold*.
Dai risultati riportati finora si evince che, come previsto, all'aumentare del valore di *unionSimilarityThreshold* diminuisce il numero di aggregazioni effettuate dall'*Aggregator System*.

Accuratezza dei Risultati

Essendo l'approccio *non-semantico* e *semantico* differenti, in quanto nel primo viene considerata la similarità *sintattica* e nel secondo quella *semantica*, sarebbe corretto utilizzare un'entità *Validator* progettata appositamente per l'approccio *semantico*. Tuttavia essendo assente un'ontologia che rappresenti le relazioni semantiche tra frasi, si è dovuto introdurre un algoritmo ibrido per l'aggregazione, mostrato in figura 4.12. Si è reso così possibile utilizzare un metodo **sintattico** per la validazione e di conseguenza si è deciso di riutilizzare la stessa entità *Validator* progettata per l'approccio *non-semantico*. In questo modo i dati raccolti sui test di validazione effettuati, pur basandosi su un metodo di validazione puramente **sintattico**, rappresentano un'approssimazione sufficiente su cui si può basare il confronto che verrà effettuato nella discussione dei risultati.
Si vuole ricordare al lettore che il riutilizzo dell'entità *Validator* comporta la possibile presenza di anomalie di validazione dovute alla natura del dataset, inoltre i risultati presentati finora dipendono fortemente dai *Synset* caricati nell'ontologia in quanto data una coppia di frasi, in fase di aggregazione, il sistema si comporta in maniera differente a seconda del numero di sinonimi trovati fra le parole che le compongono. Inoltre il numero di controlli da effettuare per poter aggregare due frasi è direttamente proporzionale al numero di insiemi di sinonimi caricati nell'ontologia. Pertanto tale numero incide sia sulle performance che sulla qualità dell'aggregazione.

4.4 Discussione dei risultati

L'introduzione del sistema MoK ha permesso di definire un contesto applicativo in cui è stato possibile applicare i due approcci individuati nel capitolo 3: il *Reasoner Semantico Hermit* e l'algoritmo *Generalized Jaccard* implementato in *Simmetrics*. È stato ideato un sistema in grado di realizzare la reazione di aggregazione definita nel modello MoK per il cui sviluppo si è resa necessaria l'introduzione di un modello semplificato rispetto a quello di origine.
In questo contesto sono emersi requisiti quali la necessità di avere performance di aggregazione elevate e incentivare la quantità di molecole e atomi aggregati a discapito della precisione. Tali requisiti hanno portato alla definizione di un modello del sistema di aggregazione che è stato man mano raffinato a seconda dello specifico approccio da utilizzare.
Con l'obiettivo di individuare l'approccio migliore tra quelli sopra descritti, sono stati effettuati degli ulteriori studi che hanno permesso di analizzare il sistema sotto due aspetti fondamentali: l'efficienza e l'efficacia dell'*Aggregator System* applicato al dominio di MoK. Per avere un'ulteriore verifica sull'efficacia dell'aggregazione è stata inoltre prevista un'entità atta alla validazione di tale processo.
La valutazione di tali aspetti ha reso necessario ricavare lo stato iniziale e finale del compartimento in esame e i tempi di esecuzione per ogni passo del processo di aggregazione. Si vuole ricordare che è stato eseguito il sistema di aggregazione su un compartimento per ogni file del dataset considerato, per un totale di quindici processi di aggregazione seguiti da altrettante validazioni, per le quali l'entità *Validator* è stata configurata con *<unionSimilarityThreshold = 50%, distributionValuesBound = 60%>*. Pertanto si è individuato lo stato iniziale come la media di tutti gli atomi e molecole presenti in ciascun compartimento prima dell'aggregazione:

averageCompartmentAtoms = 1212
averageCompartmentMolecules = 0

Tale stato iniziale è quello su cui si basa solamente il primo passo di aggregazione, in quanto gli stati iniziali dei passi successivi sono determinati dall'effetto che esso ha su tale stato.
In particolare per l'approccio *non-semantico* a seguito dell'esecuzione del primo passo di aggregazione, sono stati riscontrati i seguenti risultati di validazione:

averageWAA = 63 %
averageD–WAA = 62 %
averageTotalAggregatedAtoms = 25%

averageAtomsAggregationExecutionTime = 5508 ms

Per tale processo, come per i successivi, vengono riportate le percentuali medie di aggregazione assieme al tempo di esecuzione medio. Inoltre è stato ricavato lo stato iniziale del secondo passo di aggregazione costituito come segue:

averageCompartmentAtoms = 931
averageCompartmentMolecules = 140

A seguito dell'esecuzione del secondo passo di aggregazione, sono stati riscontrati i seguenti risultati di validazione:

averageWAA = 63 %
averageD–WAA = 60 %
averageTotalAggregatedAtoms = 27%
averageAtomsInMoleculeAggregationExecutionTime = 903 ms

Inoltre è stato ricavato lo stato iniziale del terzo e ultimo passo di aggregazione, costituito come segue:

averageCompartmentAtoms = 903
averageCompartmentMolecules = 140

L'aggregazione di tale stato ha riportato i seguenti risultati di validazione:

averageWAA = 51 %
averageD–WAA = 61 %
averageTotalAggregatedAtoms = 27 %
averageTotalAggregatedMolecules = 21 %
averageMoleculeAggregationExecutionTime = 1275 ms

Da queste informazioni si evince la compatibilità del sistema di aggregazione, implementato secondo un approccio *non-semantico*, con i requisiti individuati per il sistema MoK.
Grazie all'introduzione dell'entità *Validator* è stato possibile effettuare dei test sui risultati di aggregazione, i quali hanno mostrato l'efficacia dell'applicazione di questo approccio al dominio di MoK, in cui un'azione di aggregazione viene pianificata ed eseguita secondo leggi probabilistiche e le informazioni contenute in un compartimento subiscono la legge di decadimento ed hanno pertanto una durata vitale limitata.
Inoltre si vuole ricordare che in tutti i test effettuati per questo approccio è stato utilizzato un dataset composto da file testuali la cui natura è risultata determinante per la scelta dei parametri di configurazione del *Validator*. Una volta scelte due coppie di valori per tali parametri, un'analisi dettagliata sui risultati di esecuzione ha evidenziato la presenza di casi in cui la natura

del dataset introduce un'anomalia sulla validazione del compartimento. Tale anomalia si è manifestata in tre dei quindici casi di esecuzione, pertanto si ritiene che la natura del dataset abbia un impatto ridotto sui risultati di aggregazione.
Per l'approccio *semantico* invece è stata realizzata una versione iniziale dell'*Aggregator System* per aggregare parole utilizzando l'ontologia *WordNet*. Con l'obiettivo di effettuare un confronto tra questo approccio e quello *non-semantico*, si è deciso di realizzare una seconda versione del sistema per aggregare frasi e di effettuare una serie di test di esecuzione utilizzando lo stesso dataset individuato per l'approccio *non-semantico*. In questo caso è stato possibile ricavare l'efficienza del sistema considerando il parametro del processo di aggregazione *unionSimilarityThreshold* pari al 60%. Partendo dallo stato iniziale individuato precedentemente, a seguito dell'esecuzione del primo passo di aggregazione, sono stati riscontrati i seguenti risultati di validazione:

averageWAA = 46 %
averageD–WAA = 75 %
averageTotalAggregatedAtoms = 86 %
averageAtomsAggregationExecutionTime = 44622 ms

Inoltre è stato ricavato lo stato iniziale del secondo passo di aggregazione, costituito come segue:

averageCompartmentAtoms = 156
averageCompartmentMolecules = 528

A seguito dell'esecuzione del secondo passo di aggregazione, sono stati riscontrati i seguenti risultati di validazione:

averageWAA = 42 %
averageD–WAA = 75 %
averageTotalAggregatedAtoms = 93 %
averageAtomsInMoleculeAggregationExecutionTime = 20376 ms

Da cui è stato anche ricavato lo stato iniziale del terzo e ultimo passo di aggregazione, costituito come segue:

averageCompartmentAtoms = 81
averageCompartmentMolecules = 528

L'aggregazione di tale stato ha riportato i seguenti risultati di validazione:

averageWAA = 12 %
averageD–WAA = 75 %
averageTotalAggregatedAtoms = 93 %
averageTotalAggregatedMolecules = 82 %
averageMoleculeAggregationExecutionTime = 89103 ms

Da queste informazioni si evince un calo di performance rispetto all'approccio *non-semantico* dovuto all'algoritmo introdotto per il riconoscimento di sinonimi all'interno di frasi, descritto nella sezione 4.3. Considerando i requisiti descritti in precedenza risulta evidente che l'approccio *non-semantico* rispetta quelli di performance mentre quello *semantico* è stato in grado di aggregare un numero molto maggiore di atomi incentivando l'**exploration**. Inoltre rispetto all'approccio *non-semantico* emerge una percentuale più bassa di *WAA* dovuta al valore scelto per il parametro *unionSimilarityThreshold* e, di conseguenza, alla quantità di aggregazioni effettuate.

Capitolo 5

Conclusioni e sviluppi futuri

In questo elaborato sono stati inizialmente analizzati due approcci per il calcolo di similarità: uno basato su ontologie e l'altro basato su misure di similarità. Con l'obiettivo di realizzare un sistema di aggregazione di conoscenza sono state illustrate, attraverso uno studio sullo stato dell'arte, le proprietà computazionali di tali approcci. Per quello basato su ontologie sono stati individuati e classificati i tipi di inferenza semantica, mentre per quello basato su misure di similarità sono state analizzate le relazioni semantiche tra parole, frasi e documenti.
Successivamente è stato effettuato uno studio che ha permesso di individuare tre strumenti per ogni approccio. Tra questi si è cercato di individuare quello ottimale per ogni approccio che rispettasse determinati requisiti di espressività computazionale e di performance. A tale scopo è stato necessario introdurre dei test incentrati sui tipi di inferenza semantica e sulle relazioni individuate in precedenza per i due approcci. Durante i test effettuati per l'approccio basato su misure di similarità è emersa la necessità di determinare una tassonomia degli algoritmi per il calcolo di similarità fra parole, frasi e documenti. Questo studio ha portato alla definizione di una soglia di similarità che ha permesso di determinare se una coppia di parole, frasi o documenti fosse simile o meno. D'altro canto gli strumenti considerati per l'approccio basato su ontologie sono risultati essere equivalenti dal punto di vista dell'espressività computazionale, pertanto durante la scelta di quello migliore ci si è basati sulle performance e la stabilità.
Si è potuti così giungere alla scelta del motore di inferenza **semantica** *Hermit*, definito come approccio *semantico*, e dell'algoritmo *Generalized Jaccard* per il calcolo di similarità **sintattica**, definito come approccio *non-semantico*, come strumenti ottimali. Una volta individuati tali strumenti, sono stati realizzati dei test specifici per entrambi gli approcci in cui sono emerse principalmente le rispettive performance. I risultati ottenuti da tali test hanno

permesso di affermare che i due strumenti ottimali sono utilizzabili nel contesto dell'aggregazione di conoscenza individuato per il modello MoK, preso come contesto applicativo di riferimento per gli approcci considerati. È stato pertanto progettato un sistema di aggregazione di cui sono state realizzate due versioni per aggregare frasi, una per l'approccio *semantico* e l'altra per quello *non-semantico*. Entrambe si compongono di tre passi realizzati ispirandosi al modello MoK: aggregazione di soli atomi, aggregazione di atomi in molecole già esistenti e aggregazione di sole molecole. Per l'approccio *non-semantico* è risultato immediato, una volta scelto il dataset, implementare il comportamento di ognuno dei tre passi, mentre per quello *semantico* è stato necessario introdurre un'apposito algoritmo di aggregazione che utilizza l'ontologia *WordNet* per riconoscere i sinonimi tra due frasi. Riguardo a tale algoritmo si specifica che non è stato possibile utilizzare totalmente l'approccio *semantico* in quanto non è stata individuata un'ontologia che rappresentasse il dominio delle frasi in modo generale. Per verificare la validità di ciascun approccio è stato introdotto un metodo di validazione rappresentato dall'entità *Validator*. Questo ha consentito il confronto dei due approcci dal quale è emerso che quello *semantico* impiega un tempo di elaborazione notevolmente più alto rispetto a quello *non-semantico*, ma risulta più efficace in termini di **exploration**. Si vuole però sottolineare che tali risultati dipendono anche dalla natura del dataset considerato, la quale può introdurre delle anomalie nel processo di validazione.
Con l'obiettivo di ottimizzare il trade-off tra **exploration** e **exploitation** si potrebbero in futuro effettuare degli ulteriori test per individuare i parametri di validazione ottimali per ognuno dei tre passi di aggregazione implementati. In questo modo se si decidesse di utilizzare l'approccio *non-semantico* per implementare la reazione di aggregazione MoK, sarebbe possibile utilizzare l'entità *Validator* per verificare ed eventualmente correggere lo stato del compartimento nel caso in cui l'aggregazione non fosse avvenuta correttamente. Se invece si decidesse di utilizzare l'approccio *semantico* occorrerebbe valutare per ogni passo di aggregazione il valore del parametro *unionSimilarity-Threshold* utilizzato dal metodo di aggregazione progettato per *Hermit*, per incentivare l'**exploration** in modo da ottenere i risultati desiderati. Inoltre per questo approccio si potrebbero utilizzare ontologie apposite che rappresentino un determinato dominio applicativo per le frasi, in questo modo sarebbe possibile implementare un algoritmo di aggregazione totalmente basato sulla **semantica**. Dalle considerazioni sulle performance dei singoli approcci emerge infine la possibilità di progettare un sistema che li fonda entrambi in un singolo processo di aggregazione. In tal modo si riuscirebbero ad ottenere le performance dell'approccio *non-semantico* e l'**exploration** di quello *semantico*.

Bibliografia

[1] Samuel J. Gershman, Joshua B. Tenenbaum. Phrase similarity in humans and machines, Department of Brain and Cognitive Sciences, MIT, Cambridge, MA 02139, USA.

[2] Simon Brown, The Art of visualising Software Architecture, 16, February 2016

[3] Peter Kolb, DISCO: A Multilingual Database of Distributionally Similar Words

[4] Osamu Gotoh, An Improved Algorithm for Matching Biological Sequences, J. Mol. Bid. (1982) 162, 705-708

[5] Stefano Mariani and Andrea Omicini. Molecules of Knowledge: A novel perspective over knowledge management. In Paolo Liberatore, Michele Lombardi, and Floriano Scioscia, editors, *Proceedings of the Doctoral Consortium of the 12th Symposium of the Italian Association for Artificial Intelligence*, volume 926 of *CEUR Workshop Proceedings*, pages 23-27, Rome, Italy, 15 June 2012, AI*IA, Sun Site Central Europe, RWTH Aechen University.

[6] Stefano Mariani and Andrea Omicini. Self-organizing news management: The *Molecules of Knowledge* approach. In Jeremy Pitt, editor, *Self-Adaptive and Self-Organizing Systems Workshop (SASOW)*, pages 235-240. IEEE CS, April 2013. 2012 IEEE Sixth International Conference (SASOW 2012), Lyon, France 10-14 September 2012. Proceedings.