Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management

UN APPROCCIO PER LA DISAMBIGUAZIONE DI AUTORI DI ARTICOLI SCIENTIFICI: ALGORITMO E IMPLEMENTAZIONE

Relatore:
Chiar.mo Prof.
FABIO VITALI

Co-relatore:
SILVIO PERONI

Presentata da:
LEONARDO MONTECCHIARI

Sessione III
Anno Accademico 2013/2014

Al team Padadelli:
Markus il Crucco - Lollo il Pelato - Jacopo jacopino Pino,
giusto per condividere un altro progetto.

Ai miei Genitori,
perché hanno investito dei soldi per farmi laureare,
sono stati i primi a volere che prendessi il foglio di carta,
non io.

A mio fratello,
con la speranza un giorno di vederlo felice come lo sono io
adesso.

Indice

Introduzione

"L'invenzione non è una creazione dal nulla, bensì dal caos."
Mary Wollstonecraft Godwin

Durante la fase di implementazione di un'applicazione web che permetta di annotare articoli scientifici, ho scoperto, con grande stupore, che, ancora oggi, è difficile distinguere in un sistema informatico se due autori che condividono lo stesso nome e cognome sono persone diverse oppure la stessa: questo problema viene chiamato Disambiguazione degli autori.

Dalla necessità di risolvere il problema della disambiguazione di un insieme di autori messo a disposizione dall'Università di Bologna, il Semantic Lancet, è nata l'idea di progettare un algoritmo di disambiguazione in grado di adattarsi, in caso di bisogno, a qualsiasi tipo di lista di autori.

Lo scopo di questa tesi è dimostrare la possibilitá di realizzare un algoritmo di disambiguazione parametrico, cioè capace di impostarsi secondo criteri indicati dall'utente; attualmente esistono giá molti algoritmi per la risoluzione della disambiguazione, ma limitati nell'adattabilità: ogni algoritmo, infatti, è stato progettato per un determinato insieme di informazioni di autori ottenendo, comunque, dei risultati più che soddisfacenti ma non riutilizzabili per altri dataset di informazioni diverse. In tutte le soluzioni illustrate nella sezione 1.2 di questo elaborato sono presenti ancora delle problematiche da risolvere legate principalmente alla quantità di informazioni necessarie per l'ottimizzazzione dell'algoritmo di disambiguazione.

L'algoritmo proposto in questa tesi, a differenza di altri, è stato, quindi, ideato in modo tale da poter disambiguare ogni tipo di struttura dati legata ad autori di qualsiasi genere, e, soprattutto, non necessita di grandi quantità di informazioni. A questo scopo è stato creato Hodor : un programma in grado di leggere una lista di parametri legati alla struttura del dataset da elaborare e capace di leggere i parametri che determinano le opzioni e i pesi di calcolo utilizzati dall'algoritmo. Per semplificare l'uso del tool è stato strutturato un modello di configurazione ovvero una lista di parametri standard in grado di impostare l'algoritmo in modo da poter disambiguare, tendenzialmente, qualsiasi tipologia di dataset con problemi di questo tipo. I casi di disambiguazione trattati dal modello sono focalizzati nell'omonimia tra autori e casi di abbreviazioni nel nome: utilizzando soltanto il numero minimo di informazioni utili per la disambiguazione che in questo caso rispecchiano, oltre al nome e cognome, nella lista di coautori di ogni autore. La scelta di questi dati rispecchia la necessità di utilizzare le uniche informazioni realmente disponibili nei sistemi informatici che trattano documenti scientifici ed i loro autori, dati come "email", "data di nascita", "residenza", ad esempio, non sono facilmente reperibili. Il modello è costituito dal file di configurazione di default fornito dal tool e deve, quindi, essere opportunamente modificato se utilizzato con una struttura dati diversa da quella indicata.

Il funzionamento dell'algoritmo consiste nel prendere le informazioni relative agli autori, associarle ai parametri presi dal file di configurazione per poi confrontare ogni autore con il resto dell'insieme. Il confronto consiste nel calcolare, grazie a delle apposite funzioni impostate con i parametri, la quantità di somiglianza tra le informazioni dello stesso tipo, e se questa supererà delle soglie indicate sempre come parametri, la coppia di autori appena confrontata farà parte della soluzione. Il calcolo della quantità di somiglianza è condizionabile tramite l'utilizzo ad esempio di pesi, soglie, opzioni sempre utilizzando il file di configurazione. Tutto il funzionamento del tool e del

modello ad esso applicato, sono illustrati nel capitolo 2 e la loro struttura nel dettaglio è spiegata, a seguire, nel capitolo 3.

Per la fase di testing dell'algoritmo è stato utilizzato un dataset generato (11724 autori di cui 1295 coppie da disambiguare) dalle informazioni disponibili dal "database systems and logic programming" (DBLP)[1], in modo da essere il più etereogeneo possibile, cioè da contenere il maggior numero di casi di disambiguazione possibile. Per i primi test di sbarramento è stato definito un algoritmo alternativo discusso nella sezione 4.3 ottenendo una misura di esattezza dell'1% ed una di completezza dell'81%. L'algoritmo proposto impostato con il modello di configurazione ha ottenuto invece una misura di esattezza dell'81% ed una di completezza del 70%, test discusso nella sezione 4.4. Successivamente l'algoritmo è stato testato anche su un altro dataset: Semantic Lancet[2](919 autori di cui 34 coppie da disambiguare), ottenendo, grazie alle dovute variazioni del file di configurazione, una misura di esattezza del 84% e una di completezza del 79%, discusso nella sezione 4.5.

L'algoritmo deve essere ancora ottimizzato sia nella complessità che nell'ottimizzazione del risultato, e ci sono ancori numerosi step da compiere prima della sua pubblicazione. Sarà necessaria un'ulteriore fase di testing per migliorare alcune delle sue funzionalità, come ad esempio la semplificazione del file di configurazione per facilitarne l'utilizzo o lo sviluppo di ulteriori opzioni per rendere più affidabile la disambiguazione finale.

[1]http://dblp.uni-trier.de/db/
[2]BCINPV14.

Elenco delle figure

Elenco delle tabelle

Elenco listati di codice

Capitolo 1

Disambiguazione degli Autori

Uno degli obiettivi principali dello studio della letteratura è unire in maniera bidirezionale autori e opere: attribuire a un autore le sue opere e viceversa [ST09].

1.1 Il problema della Disambiguazione

In un ipotetico sistema formato da due insiemi di informazioni, dove ogni elemento della categoria di partenza è in relazione con uno e un solo elemento della categoria di arrivo, è sempre possibile ricavare una funzione; in letteratura, al contrario, non abbiamo corrispondenze univoche di informazioni proprio a causa della mancanza di dipendenze funzionali e si possono, quindi, verificare casi di relazioni ambigue tra i singoli elementi degli insiemi. Prendiamo ad esempio una lista di autori con relative pubblicazioni e proviamo a determinarne l'eventuale unicità:

in questa lista si può notare che ogni autore è unico, ovvero non ne esistono simili o omonimi, da cui si possono individuare le seguenti relazioni:

- Paul Slardar è autore di A
- Sandra Tomero è autore di A
- David Tomero è autore di B

N°	Name	Family Name	Pubblications
0	Paul	Slardar	A
1	Sandra	Tomero	A
2	David	Tomero	B
3	Lewis	Gabà	B

Tabella 1.1: Lista di autori unici

- Lewis Gabà è autore di B

Proviamo ora usando quest'altra lista:

N°	Name	Family Name	Pubblications
0	Paul	Slardar	A
1	Paul	Slardar	C
2	Paul A.	Slardar	B
3	Amid	Slardar	B

Tabella 1.2: Lista di autori NON unici

Eseguendo un'analisi accurata, non riusciamo ad individuare relazioni uniche e possiamo, quindi, fare solo delle supposizioni:

- i 2 autori Paul Slardar potrebbero essere la stessa persona, autrice di A e C.
- Paul A. Slardar e Amid Slardar potrebbero essere la stessa persona autrice di B.
- tutti gli autori in tabella potrebbero essere la stessa persona, autrice di A, B, C.

In questo esempio vi è un chiaro caso di omonimia, alta similarità, e presunta multiregistrazione nel sistema.
Esistono molti altri casi ben noti che, aggiunti a quelli citati, sono la causa del

problema della disambiguazione. È possibile classificare, per semplicità, tale problema in due categorie distinte: *Accorpamento dei Multipli* e *Divisione degli Omonimi*

- **Accorpamento dei Multipli**
  È il tipo più comune, di cui fa parte anche il nostro esempio appena discusso: il medesimo autore è rappresentato più volte nel dataset, ma in modi diversi.

- **Divisione degli Omonimi**
  Più autori sono rappresentati dallo stesso record nel dataset. I casi possibili sono due: multiregistrazione del medesimo autore e/o registrazione di un omonimo già presente nel database. In un database relazionale (DBMS) può avvenire un caso di multiregistrazione, ma il record nel dataset sarà unico rispetto agli altri già presenti tornando, quindi, al primo tipo di disambiguazione; un omonimo invece può iscriversi senza il rischio di sovrapporsi ad un altro record perché nei DBMS la chiave primaria che contraddistingue ogni singola informazione nel database è unica. Questa tipologia di problematiche è strettamente legata al Semantic Web: non esistono, infatti, chiavi univoche, ma solo risorse che rappresentano dei concetti. Le URI che identificano il concetto "Persona" vengono definite, nella maggior parte dei casi, con "prefisso:nome-cognome" e tutte le relative informazioni fanno riferimento alla medesima URI. Prendiamo ad esempio i seguenti autori:

Name	Family Name	Pubblications
Paul	Slardar	A
Paul	Slardar	B
Pàul	Slardar	C

Tabella 1.3: Lista di autori omonimi o simili

In formato RDF verrà rappresentato nel seguente modo:

```
hodor:paul-slardar a foaf:Person .
```

questa URI rappresenta il concetto Persona, nelle URI i caratteri non possono essere accentati. quindi le 3 persone fanno riferimento a questa URI.

```
hodor:paul-slardar foaf:name "Paul" .
hodor:paul-slardar foaf:name "Pául" .
hodor:paul-slardar foaf:familyName "Slardar" .
hodor:paul-slardar pro:holdsRoleInTime "A" .
hodor:paul-slardar pro:holdsRoleInTime "B" .
hodor:paul-slardar pro:holdsRoleInTime "C" .
```

alla stessa URI vengono associati i 2 nomi Paul e Pàul, il cognome Slardar e tutte le pubblicazioni.

I 3 autori inizialmente divisi e facili da individuare sono stati unificati: il risultato è un unico autore che si chiama Paul o Pàul Slardar autore di A, B e C.

È potenzialmente possibile, a livello teorico, creare un registro globale di autori univoci: in un tale sistema ogni autore dovrebbe registrare tutte le opere già pubblicate, per poi continuare ad aggiornare tutte le altre informazioni, come ad esempio stato sociale, residenza, cambio di nome etc. Purtroppo, aggirando il grande problema della disambiguazione, vengono generati molti altri problemi dovuti alla difficoltà di garantire la sicurezza, l'efficienza ed il sostenimento dei costi eccessivi di gestione di un simile database globale[ST09].

1.2 Soluzioni esistenti del problema

Ogni Dataset che raccoglie informazioni di autori avrà sempre un problema di disambiguazione più o meno grave rispetto agli altri e in base al tipo

di informazioni note, è possibile elaborare una soluzione più o meno efficiente: il noto sito IMDb (Internet Movie Database), per esempio, provvede a disambiguare i propri autori utilizzando le informazioni ottenibili tramite i social network[MAC05] e individuando nello specifico le seguenti proprietà:

1. tipi di informazione
   (a) biografica (ad es. *George H.W. Bush* è stato il 41° presidente degli Stati Uniti d'America)
   (b) grammaticale (ad es. *fall* usato come nome piuttosto che verbo)
2. supervisione: i metodi di disambiguazione fanno riferimento ad esempi preimpostati nel sistema.

Il vantaggio sostanziale utilizzato da questo approccio è la capacità di accedere ad una grande quantità di dati, non solo quelli semplici (nome, cognome) ma anche testi, immagini, video.
Non sempre possiamo ottenere una simile quantità di informazioni poiché, il più delle volte, i dataset da disambiguare hanno solo pochi campi a disposizione (vedi sezione 1.1).

Un altro approccio discusso in questo articolo [DGA11], è stato elaborato per la disambiguazione di database biometrici ed è articolato in 3 fasi distinte: *Database Integration*, *Mapping Generation* e *Filtering*.

- **Database Integration**
  Fase in cui vengono integrati il database di supporto (CINECA) a quello principale (Bibliometrics), con lo scopo di colmare le informazioni mancanti per arginare gli errori: sono rispettivamente la lista di ricercatori che hanno lavorato nelle università italiane in un determinato periodo e la lista di articoli con le loro affiliazioni a ricercatori italiani.
- **Mapping Generation**
  Fase in cui l'algoritmo genera le relazioni tra gli autori del dataset principale e le identità dei ricercatori dei dataset esterni, ottenendo così una

serie di coppie (autore,identità), andando a creare delle macro-serie tali da rappresentare le relazione tra dataset principale e dataset esterni. La classificazione reale prevede macro-serie formate da una sola coppia (autore,identità). I casi di omonimia dipendono dalla configurazione di una threshold: a una soglia alta corrisponde la completa uguaglianza tra nome e cognome, mentre a una soglia bassa corrispondono solo il cognome e le iniziali. Gli autori esclusi dalla fase di disambiguazione sono coloro che non hanno un corrispondente all'interno del database CINECA, ovvero quelli per cui non è possibile formare una coppia (autore,identità).

- **Filtering**
  Eliminazione delle false coppie create nelle seconda fase; vengono individuati soltanto i casi di omonimia appartenenti ad alcune categorie, definite in base alle strutture dei dataset usati. Nel caso trattato nell'articolo le categorie usate per il filtro sono:

  - **External homonyms**
    Autori senza un ruolo di facoltà oppure affiliati con qualche tipo di organizzazione di ricerca, con l'identità omonima nel ruolo di facoltà.
  - **Inter-address homonyms**
    Autori con un ruolo universitario, con una identità omonima in un'altra università.
  - **Intra-address homonyms**
    Autori con un ruolo di facoltà, con una identità omonima nella stessa università appartenente ad un'altro SDS[3].
  - **Perfect homonym**
    Autori con un ruolo di facoltà, con una identità omonima nella stessa università appartenente allo stesso SSD.

[3]Settori scientifici disciplinari

- **Address Filter**
  Questo filtro elimina tutte le coppie in cui l'affiliazione dell'autore presente su Bibliometrics (estratto dal campo "address") è incompatibile con l'affiliazione dell'identità (l'università indicata nel database CINECA)

- **TheWoS-SDS Filter**
  Tra tutte le coppie rimanenti cancella quelle in cui la categoria WoS dell'articolo non è compatibile con l'SDS dell'identità

- **The shared SDS Filter**
  Filtra le coppie appartenenti a qualsiasi cluster che contenga un'ulteriore coppia la cui identità ha un SDS condiviso. Si tratta di un SDS in comune relativo allo stesso articolo; già precedentemente approvato.

- **The maximum correspondence Filter**
  Utilizzato per processare tutti i cluster rimanenti, relativi ai casi irrisolti di omonimia. A tal fine, per ogni cluster, viene selezionata una sola coppia avente la massima corrispondenza tra SDS e categoria dell'articolo a cui fa riferimento il cluster. Tutte le coppie fin qui ottenute sono disambiguate e corrette, nonostante il set finale possa comunque contenere un certo numero di falsi positivi: ogni cluster con cardinalità non unitaria possiede, ad esempio, almeno un falso positivo. In aggiunta gli eventuali falsi negativi possono compromettere l'accoppiamento di alcuni valori.

Questo schema rappresenta il processo appena descritto:

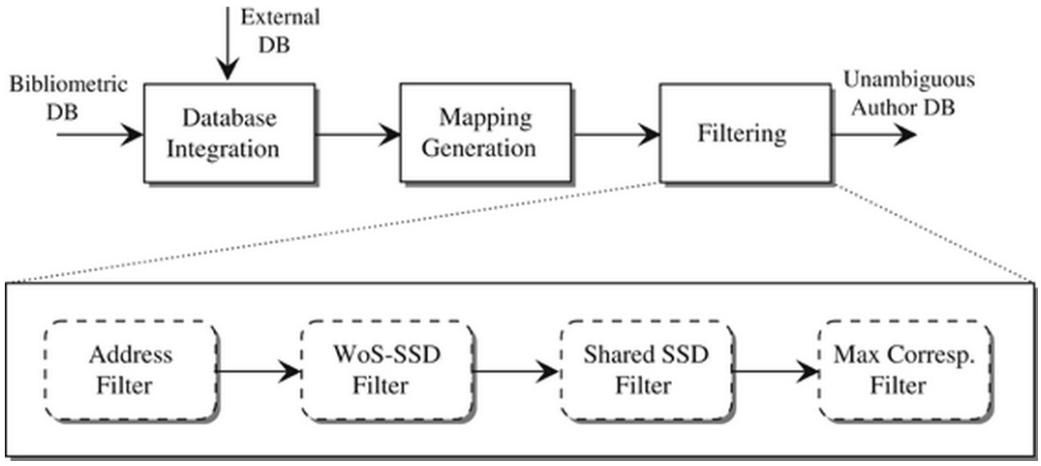

Figura 1.1: Flowchart del processo

Nella soluzione discussa in questo articolo [SMPPH14] viene utilizzato un approccio non ancora trattato ma esistente da diverso tempo; vengono effettuate, inizialmente, delle considerazioni sulle soluzioni già presenti individuandone poi limiti e possibili errori:

- gli algoritmi esistenti per riuscire a disambiguare persone con lo stesso nome hanno bisogno di dati con un maggiore coefficiente di univocità, come ad esempio data e/o luogo di nascita, informazioni non disponibili in certi database di larga scala e datati.
- se i metadati sono incompleti, il risultato dell'algoritmo tenderà ad essere sbagliato.
- non viene utilizzata la rete di citazioni pubbliche Web of Science (Wos[4]) perché non tutti i database la includono.

Questo algoritmo, invece, punta ad utilizzare tutta la rete WoS per ottenere informazioni dei ricercatori che la utilizzano per le loro citazioni.
La prima fase è quella di raggruppamento, suddivisa in due sottofasi:

- **prima fase di clustering** - determinare se due documenti sono stati scritti dallo stesso individuo.

[4]http://thomsonreuters.com/thomson-reuters-web-of-science/

- **seconda fase di clustering** - calcolare un nuovo punteggio di similarità tra tutti i cluster creati nella prima sottofase.

Alla fine di questa fase, ogni insieme di documenti appena definito appartiene ad un solo ricercatore.

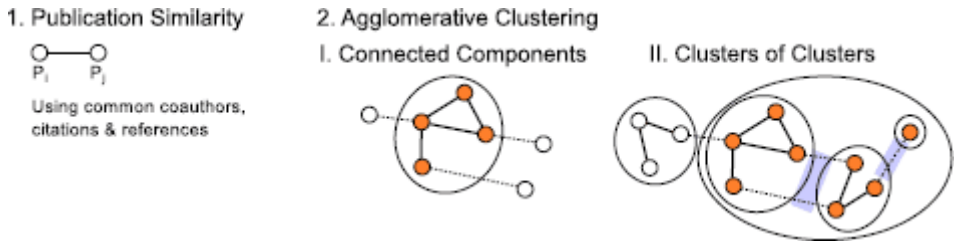

Figura 1.2: Fase di raggruppamento dei documenti

La seconda fase è quella di ottimizzazione: vengono raccolti i parametri trovati nella fase di raggruppamento e vengono testati utilizzando i profili Google Scholar (GSP), calcolando la precisione ed il recupero (precision[5] & recall[6]); verranno utilizzati, quindi, i parametri di input che restituiscono il miglior compromesso tra il valore di precision e quello di recall.

In tutte le soluzioni illustrate in questa sezione sono presenti, ancora, delle problematiche da risolvere legate principalmente alla quantità di informazioni necessarie per l'ottimizzazzione dell'algoritmo di disambiguazione. La soluzione proposta nel prossimo capitolo è stata elaborata partendo proprio da queste problematiche.

[5]misura di esattezza
[6]misura di completezza

Capitolo 2

Descrizione della soluzione proposta per la disambiguazione

In questo capitolo saranno discusse le differenze tra le soluzioni esistenti e quella proposta, ed illustrato il funzionamento dell'applicazione utilizzata per la disambiguazione: Hodor, un tool in grado di eseguire un algoritmo di disambiguazione parametrico e il modello di configurazione.

2.1 Soluzione proposta

Nel precedente capitolo sono state analizzate alcune tipologie di risoluzione del problema della disambiguazione: nell'ultima soluzione della sezione 2.2 soprattutto sono emersi i limiti comuni a tutti gli algoritmi già presenti. In generale si possono riassumere le seguenti problematiche:

- sono necessarie molte informazioni, quanto più uniche possibili.
- in caso di mancanza di informazioni, l'algoritmo tende a falsificare il risultato.

- bisogna tener conto dell'errore umano, cioè che l'informazione non sia salvata in modo corretto.

- nel caso in cui sia previsto un sistema di previsione del risultato atteso, bisogna tener conto dell'errore di calcolo del risultato ottenuto (difficilmente calcolabile).

Non può esistere quindi la soluzione perfetta per tutti i dataset, ma solo soluzioni subottimali, relative cioè a ogni singolo tipo di dataset.
Lo scopo di questa tesi è quello di illustrare come, invece, sia possibile trovare un'unica soluzione soddisfacente per qualsiasi tipo di dataset da disambiguare.
Partendo dalla necessità di disambiguare il dataset Semantic Lancet [BCINPV14] dell'Università di Bologna, un triplestore RDF (ne parleremo nel capitolo 4), è nata l'idea di elaborare Hodor: un tool configurabile in grado di eseguire un algoritmo di disambiguazione di tipo "accorpamento dei multipli" [7] personalizzato. La soluzione proposta è divisa in tre fasi principali: *Analisi*, *Configurazione*, *Elaborazione*.

- **Analisi**
  È la fase di supporto della configurazione in cui vengono individuate le informazioni da attribuire ad ogni autore e come deve essere impostato il dataset da elaborare: il risultato dovrebbe rappresentare la concentrazione dei dati da disambiguare. Applichiamo, quindi, un esempio a questo piccolo dataset:

[7] vedi sezione 2.1

Name	Family_Name	Full_Name	Name_Abbr	Family_Abbr
Lorenzo	Gigli	Lorenzo Gigli	L	G
Markus	Aurich	Markus Aurich	M	A
Jacopo	Carletti	Jacopo Carletti	J	C
L.	Gigli	L. Gigli	L	G

Tabella 2.1: Esempio di Dataset da analizzare

la concentrazione dei dati da disambiguare è nel nome (*Lorenzo Gigli* e *L. Gigli* sono fortemente simili): è necessario, quindi, individuare se *L.* è l'abbreviazione di *Lorenzo*. Ogni analisi è soggettiva e dipendente sempre dall'esito più o meno atteso, che, a causa della scarsità di dati univoci a disposizione, può risultare fortemente falsato. La decisione è, quindi, quella di non dare tendenzialmente molta importanza alle abbreviazioni ottenendo come risultato dell'analisi il seguente dataset:

Name
Lorenzo
Markus
Jacopo
L.

Tabella 2.2: Esempio di Dataset risultante dall'analisi

Con molta probabilità, la concentrazione dei dati da disambiguare non sarà unica, come nel caso dell'esempio; esistono, infatti, dataset con più insiemi di concentrazione ed ognuno di essi rappresenta un caso diverso da elaborare. La difficoltà sostanziale è quella di stimare quanti e quali casi da disambiguare, così da poter determinare, con massima precisione, la concentrazione dei dati contenuti nell'intero dataset.

- **Configurazione**

Fase di configurazione di Hodor: ogni campo (Name, Family_Name..) viene specificato in un file di configurazione prima dell'avvio del programma con indicazione di tutte le concentrazioni trovate e delle relative opzioni, ottenendo, così, i parametri necessari per l'algoritmo che andrà a disambiguare il dataset in ingresso. Ogni set di parametri che rappresenta una determinata concentrazione è definito come un Setting.

È possibile anche effettuare un'analisi più accurata tramite l'utilizzo del tool stesso, cioè utilizzando un'apposita configurazione che permette un risultato in grado di evidenziare una concentrazione di dati più precisa, utile nell'elaborazione vera e propria del dataset (vedi capitolo 4)

- **Elaborazione**
  Fase in cui viene impostato ed eseguito l'algoritmo di disambiguazione grazie ai parametri indicati nel file di configurazione.

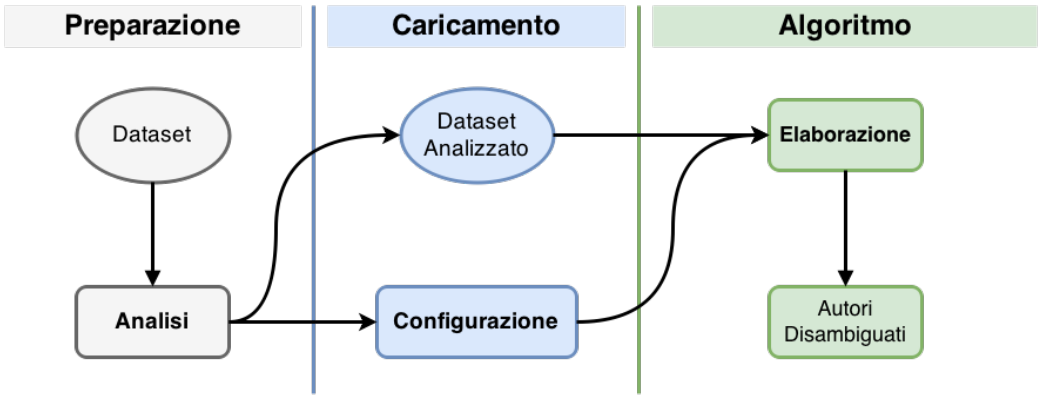

Figura 2.1: Fasi della soluzione proposta

2.2 Hodor

Hodor è un tool in grado di eseguire un algoritmo di disambiguazione degli autori parametrico; pensato inizialmente come algoritmo statico per il

Semantic Lancet [BCINPV14], cioè capace di disambiguare solo i dataset con la medesima struttura, in seguito completamente ripensato sia nella forma che nella struttura, con l'unico scopo di renderlo completamente riutilizzabile e configurabile sotto ogni aspetto. Le parti principali di Hodor sono: *Sistema di Caricamento*, *Sistema delle Similarità* e *Sistema dei Setting*.

- **Sistema di Caricamento**
  Dato che il programma è completamente configurabile, è necessario indicare nell'apposito file la struttura del dataset in input specificando per ogni campo il tipo:

  - **Identifier** è un campo che rappresenta dati unici, come ad esempio: URI, ID, Matricola, Codice Fiscale.
  - **Literal** è un campo che rappresenta dati non unici come ad esempio: nome, cognome, nickname ed in generale tutte le stringhe e numeri.
  - **Date** è un campo che rappresenta delle date come ad esempio: anno di pubblicazione, periodo lavorativo.

  Tutti i campi possono rappresentare anche delle liste di dati, l'importante è sia specificamente indicato nella struttura del dataset. Prendiamo, ad esempio, i seguenti campi: *Full_Name* - *Pubblication_Year_List* - *ID* i tipi relativi saranno

  - $Full\_name \Rightarrow Literal$.
  - $Pubblication\_Year\_List \Rightarrow Date/Identifier$ (indicando come dividere il dato precedentemente unificato).
  - $ID \Rightarrow Identifier$.

- **Sistema delle Similarità**
  La Similarità o grado di Similarità è la quantità di somiglianza tra due elementi espressa in termini percentuali: ogni Setting definisce distintamente una serie di funzioni che calcola la Similarità tra Autori

utilizzando i parametri indicati. Durante il calcolo finale viene presa in considerazione la coppia di autori che riesce a superare almeno uno dei limiti di Similarità presenti in ognuna delle impostazioni definite.

- **Sistema dei Setting**
  Nella configurazione possono essere definiti i Setting tramite cui il tool dovrà impostare l'algoritmo. Ogni Setting deve avere i valori globali e lista dei parametri.
  - i valori globali rappresentano il limite minimo di Similarità da raggiungere per ogni coppia di autori elaborata nel sistema.
  - all'interno della lista dei parametri vengono indicati i campi da includere nell'impostazione con i relativi sotto-parametri e opzioni, nello specifico:
    * limite minimo di Similarità
    * limite massimo di Similarità
    * peso di Similarità
    * lista di opzioni di Calcolo di Similarità

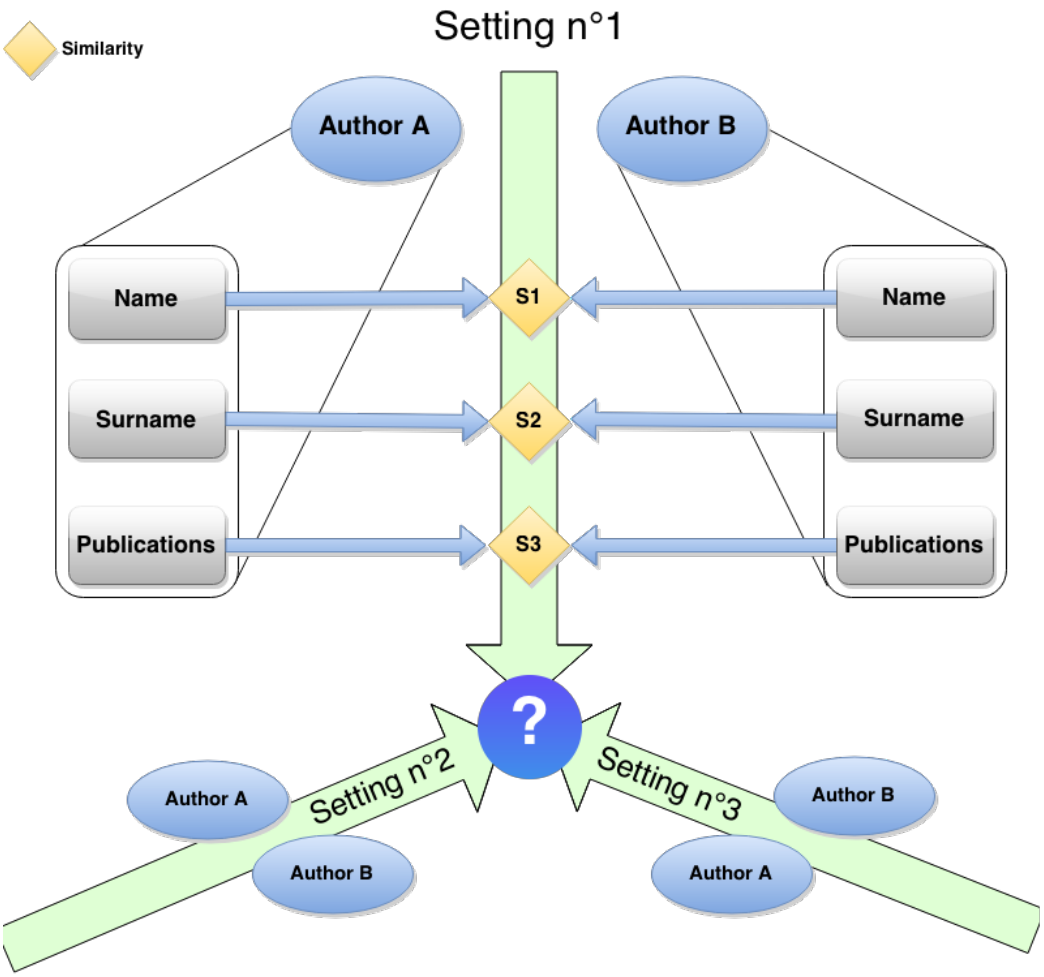

Figura 2.2: Esempio di Similarità a 3 Setting, con il primo nel dettaglio

2.3 Algoritmo di Disambiguazione

L'algoritmo di disambiguazione prende le informazioni di ciascun autore, le associa ai parametri indicati in un singolo Setting, ed esegue un confronto univoco di tutti gli autori del dataset per poi avviare un calcolo di similarità utilizzando le apposite funzioni parametrizzate. Questo procedimento viene ripetuto per ogni Setting, finché almeno uno di questi confronti non supera le

soglie di calcolo impostate nel file di configurazione. È relativamente semplice l'algoritmo, perché avvia soltanto le funzioni di calcolo delle similarità tra tutti gli autori e le rispettive informazioni; la disambiguazione avviene fondamentalmente per mezzo dei parametri presenti nel file di configurazione, perché condizionano le funzioni di calcolo delle similarità e quindi scelgono indirettamente le coppie di autori che faranno parte della lista di autori disambiguati.

Trovare il giusto set di parametri di disambiguazione è la vera grande difficoltà della soluzione proposta; è fondamentale, quindi, analizzare bene il dataset in input per decidere al meglio quanti e quali Setting utilizzare nella configurazione del tool. Senza un file di configurazione ben curato, Hodor non riuscirà ad eseguire l'algoritmo in modo efficiente, l'errore di conseguenza dipenderà sempre e solo dal soggetto che utilizza il programma.

Si è reso necessario creare un modello di configurazione di partenza (o file di configurazione di default) già testato su un dataset generato, verrà discusso nel capitolo 4, solo ed esclusivamente a questo scopo. Il dataset in questione ha una struttura molto comune:

- **person**: campo di identificazione, rappresenta un codice univoco come ad esempio "1029103239103" e utilizzato in tutte le impostazioni poiché in grado di distinguere i record del dataset senza far parte del sistema di similarità.

- **family_name**: rappresenta il cognome dell'autore e viene utilizzato in tutte le impostazioni in quanto costituisce il campo di partenza per i casi di "accorpamento dei multipli"; ha un peso elevato ed il requisito minimo accettabile è una Similarità del 100%.

- **given_name**: rappresenta il nome dell'autore e viene utilizzato in tutte le impostazioni ma con pesi e soglie di Similarità differenti, trattandosi

dell'unico campo in grado di distinguere le concentrazioni dei dati da disambiguare.

- **full_name**: rappresenta sia il nome che il cognome, non ha soglie di Similarità, ma viene utilizzato per diminuire l'errore di calcolo ottenuto dal nome.

- **coauthor_list**: una volta individuate le concentrazioni, la lista di coautori funge da filtro poiché scarta le coppie che hanno il nome e cognome ai limiti consentiti senza un riscontro minimo nel suddetto campo; il peso è più basso dei precedenti e non vi sono soglie di Similarità.

- **publications**: la lista delle pubblicazioni viene usata come ulteriore filtro in alcune delle concentrazioni, non ha soglie ed ha un peso dimezzato rispetto a quello della lista dei coautori.

- **publications_pub_years**: la lista degli anni di pubblicazione è usata in una delle impostazioni e viene utilizzata come terzo filtro, non ha soglie ed ha un peso minore del campo publications.

Grazie all'analisi iniziale, sono stati scartati i campi inutili, così da poter migliorare l'affidabilità dell'algoritmo:

- *family_name_abbr*: abbreviazione del cognome.

- *given_name_abbr*: abbreviazione del nome, utile come campo di riserva (esiste come opzione).

- *publications_titles*: lista dei titoli delle pubblicazioni.

- *publications_venues*: lista dei luoghi di pubblicazione.

- *publications_affiliations*: la lista delle affiliazioni è un campo generalmente molto sensibile, ma raro nella maggior parte dei record, tendendo a falsare il risultato, se utilizzato.

2.3.1 Esempio di Disambiguazione

Per aiutare il lettore nella comprensione della soluzione proposta, di seguito viene riportato un caso di disambiguazione estrapolato dal risultato ottenuto utilizzando il modello di configurazione.

Dati i seguenti due record A e B, essi sono salvati nel dataset come due autori distinti.

Tabella 2.3: Esempio reale di un caso di disambiguazione - parte 1

	person (Identifier)	family_name (Literal)	given_name (Literal)	full_name (Literal)
A	xy1	Liu	Feng	Feng Liu
B	ze3	Liu	Feng	Feng Liu

Tabella 2.4: Esempio reale di un caso di disambiguazione - parte 2

	coauthors_list (Identifier)	publications (Identifier)	publications_pub_years (Identifier)
A	xy2, xy3, xy4, ze4, ze5	001,002,003	Empty
B	ze4,ze5	004,005,006	Empty

Il risultato ottenuto dall'algoritmo di disambiguazione preimpostato con il modello di configurazione è:

- *person* grado = 0.0 con peso = 0
- *family_name* grado = 100.0 con peso = 4 massima somiglianza trovata in: "Liu" & "Liu"
- *given_name* grado = 100.0 con peso = 3 massima somiglianza trovata in: "Feng" & "Feng"
- *full_name* grado = 100.0 con peso = 3 massima somiglianza trovata in: "Feng Liu" & "Feng Liu"

- *coauthors_list* grado = 22.6 con peso = 2
- *publications* grado = 0.0 con peso = 1
- *publications_pub_years* grado = 0.0 con peso = 0.5

La similarità pesata è di 77.4. Il calcolo della similarità di ogni campo e quella totale sono illustrati nel dettaglio nella sezione 3.5 e, invece, il modello proposto viene discusso nel dettaglio nella sezione 3.7.

Questo modello può essere utilizzato solo se il dataset rispecchia la struttura utilizzata: se le concentrazioni sono simili a quelle individuate nel dataset di test verranno trovati i record da accorpare; in caso contrario invece il risultato dell'elaborazione sarà pari a 0 autori disambiguati. Non esiste, al momento, un sistema di analisi guidato all'interno del tool: resta, quindi, fondamentale eseguire ripetutamente la fase di analisi per riuscire a modificare al meglio la configurazione di default.

Capitolo 3

Struttura dettagliata del Sistema: Hodor e Algoritmo

Vedremo ora nel dettaglio la struttura del tool Hodor e quella del modello di configurazione utilizzato per la disambiguazione del Dataset di testing.

3.1 Tecnologia utilizzata

Hodor è un tool Stand Alone scritto in Java 8, con la possibilità di eseguire query verso un endpoint per ottenere un dataset in runtime, invece che leggerlo localmente: l'unico formato accettato è Json sia per il dataset che per il file di configurazione, rispettando la struttura predefinita. Le librerie usate sono le seguenti:

- **Gson** in grado di convertire Json in oggetti Java e viceversa.
- **json** di supporto per la gestione dei Json.
- **javassist** in grado di creare classi, metodi, variabili a runtime facilitando l'uso della "reflection".
- **okhttp e okio** usate per la gestione delle richieste http.

Ogni dataset in input utilizzato per le fasi di testing è il risultato di una query SPARQL di tipo "SELECT" sul triplestore Semantic Lancet, mentre il dataset generato per effettuare i vari i test di collaudo del tool, è stato ricavato dal database DBLP e convertito grazie ad uno script in Node.js.

3.2 Struttura generale e Flussi

La struttura generale è composta da due parti principali: *Builder* e *Core*.

- **Builder**
  Il builder ha la funzione principale di generare i Setting definiti nel file di configurazione seguendo gli appositi modelli *Configuration* e *Params*.
  - **Configuration** raccoglie la configurazione globale di cui fanno parte:
    * l'eventuale query da eseguire
    * l'endpoint su cui eseguire la query
    * la struttura del dataset
    * l'identificativo univoco
  - **Params** rappresenta la lista di parametri di un singolo Setting, generali e di ogni campo campo:
    * parametri generali: le soglie da rispettare per il calcolo di similarità
    * parametri di campo: peso, soglie e lista di opzioni

  In questa parte della struttura, vengono anche definiti i tipi e le relative similarità che ogni campo può assumere: *Identifier*, *Literal* e *Date*.

- **core**
  Nel Core viene caricata la struttura che rappresenta la lista di autori grazie al modello *Author* e viene avviato l'algoritmo.

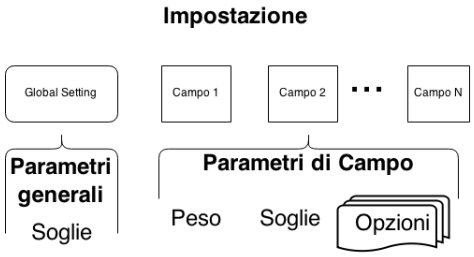

Figura 3.1: Dettaglio di un Setting

– **Author** definisce il modello di ogni singolo autore:
  * HashMap Identifiers: contiene tutti i campi Identifier.
  * HashMap Literals: contiene tutti i campi Literal.
  * HashMap Dates: contiene tutti i campi Date.

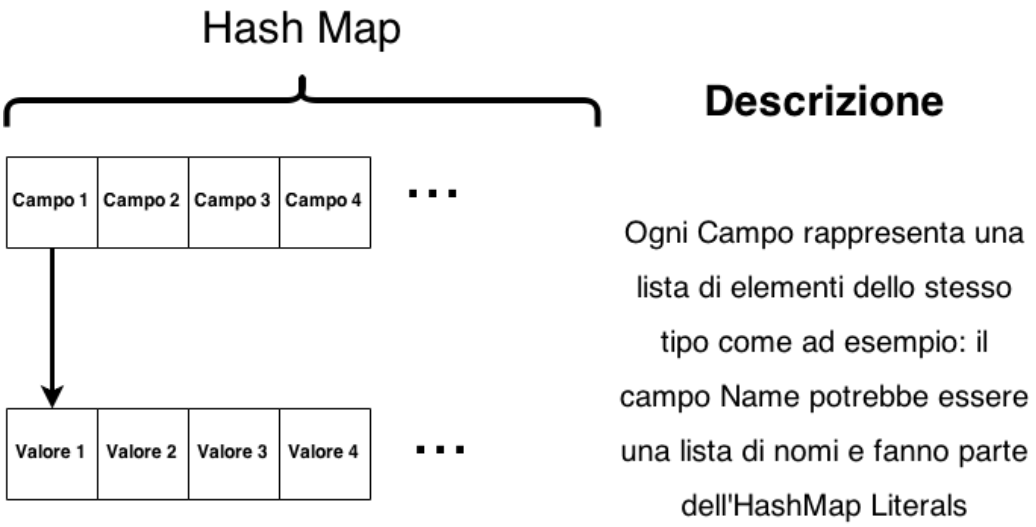

Figura 3.2: Dettaglio di un Hash Map presente nel modello di un autore

Ogni autore ha una funzione Similarity, in grado di richiamare tutte le sottofunzioni dei campi di un autore.

La struttura generale del tool può essere rappresentata secondo il seguente schema:

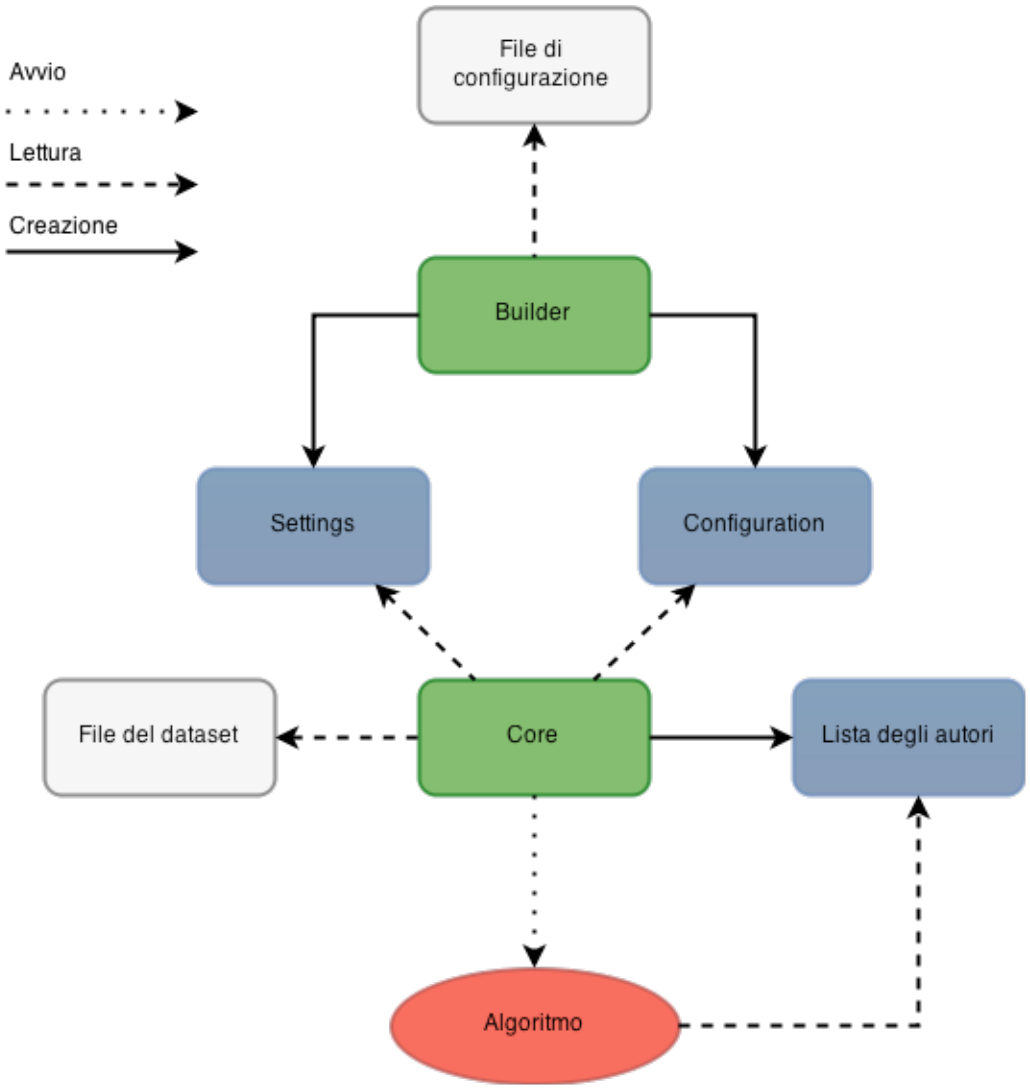

Figura 3.3: Schema funzionamento di Hodor

3.3 Struttura input

I due input *Dataset* e *file di configurazione* devono avere una struttura predefinita in formato json. Il Dataset dipende sempre dai campi trattati, ottenibile ad esempio eseguendo una query SPARQL ad un Fuseki senza dover applicare delle eventuali conversioni.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

```
PREFIX pro: <http://purl.org/spar/pro/>

SELECT DISTINCT
 ?person
 (GROUP_CONCAT(DISTINCT ?role ; separator = " ; ") AS
    ?rolelist)
WHERE {
 ?person a foaf:Person ;
  pro:holdsRoleInTime ?role ;

}
GROUP BY ?person
LIMIT 1
```

Listing 3.1: SPARQL query

Listing 3.2: json ottenuto eseguendo la query SPARQL

```
{
  "head": {
    "vars": ["person" ,"rolelist" ]
  } ,
  "results": {
    "bindings": [
      {
        "person": {"type": "uri" ,"value":
           "http://www.hodor.com/resource/person/paul-slardar"
           },
        "rolelist": {"type": "literal" ,"value":
           "http://www.hodor.com/author/paul-slardar-as-author"
           }
      }
    ]
```

```
  }
}
```

3.3.1 File di Configurazione

Il file di configurazione è stato strutturato in modo da poter accogliere le informazioni generali del dataset e tutti i parametri relativi ai campi.

Di seguito viene riportata la struttura di un file di configurazione ad un singolo Setting:

Listing 3.3: Esempio di file di configurazione

```
{
  "configuration": {
    "endpoint": "an-end-point",
    "query": "a-query",
    "data_structure": [
      {
        "key": "campo",
        "type": "Tipo",
        "delimiter": ";",
        "identifier": true
      }
    ]
  },
  "settings": [
    {
      "global_setting": {
        "global_threshold": "75",
        "global_checks": "0"
      },
      "param": [
```

```
        {
          "key": "campo1",
          "type": "Tipo",
          "weight": "0",
          "threshold": "0",
          "limit": "100",
          "reserve": ["campo_di_riserva"],
          "options": {
            "option1": true,
            "option2": true
          }
        }
      ]
    }
  ]
}
```

Il json Object "*configuration*" raccoglie tutti i campi relativi al dataset da elaborare: "*endpoint e query*" sono facoltativi e permettono di accedere ad un dataset remoto eseguendo una query, mentre "*data_structure*" è un json-object obbligatorio rappresentante la struttura dei campi del dataset in input. Prendiamo ad esempio il seguente dataset:

Tabella 3.1: Dataset da strutturare

Identity	**Full_Name**	**Publications**
000001	Stefania Storani	A;B
000002	Giacomo Broglia	C;D;E

La struttura che lo rappresenta è la seguente:

Listing 3.4: Esempio di file di configurazione: data_structure

```
1
2    "data_structure": [
3      {
4        "key": "Identity",
5        "type": "Identifier",
6        "delimiter": "",
7        "identifier": true
8      },
9      {
10       "key": "Full_Name",
11       "type": "Literal",
12       "delimiter": "",
13       "identifier": false
14     },{
15       "key": "Publications",
16       "type": "Identifier",
17       "delimiter": ";",
18       "identifier": false
19     }
20   ]
```

Il Json-Array *settings* rappresenta la lista dei Setting. Ogni elemento al suo interno è diviso in *global_setting* e *param*.

- **global_setting** è un json-object che accoglie la soglia generale *global_threshold*, da raggiungere in percentuale ed il numero di funzioni di similarità *global_checks*, da superare sempre in percentuale.
- **param** è un json-array che accoglie i parametri attribuiti ad ogni campo del *data_structure*:
  - *key* - nome del campo
  - *type* - tipo del campo

- *weight* - peso di similarità non in percentuale
- *threshold* - soglia Similarità di campo da raggiungere in percentuale
- *limit* - limite Similarità di campo da non superare in percentuale
- *reserve* - campo di riserva, usato nel caso in cui la Similarità calcolata in questo campo non sia sufficiente a raggiungere la threshold oppure supera il limit.
- *options* - lista delle opzioni, specificate per cambiare quelle di default (come ad esempio *spaces* e *dots* che corrispondono rispettivamente al trattare il campo senza spazi e senza punti.)

Facendo riferimento al Dataset rappresentato nella tabella 3.1, definiamo il seguente Setting:

Listing 3.5: Esempio di Setting nel file di configurazione

```
{
      "global_setting": {
        "global_threshold": "100",
        "global_checks": "0"
      },
      "param": [
        {
          "key": "Full_Name",
          "type": "Literal",
          "weight": "1",
          "threshold": "100",
          "limit": "100"
        }
    ]
}
```

L'algoritmo creato andrà a disambiguare soltanto gli autori che presentano nel campo *Full_Name* lo stesso valore di un altro autore.

3.4 Sistema di Caricamento

La struttura del dataset in input, indicata nel file di configurazione, viene utilizzata per la creazione della lista di autori da disambiguare. Il sistema di caricamento legge la struttura ed il dataset impostando ogni campo di ciascun autore con il relativo tipo - *Identifier, Literal o Date* definito grazie a due classi Builder e BuilderHandler:

- **Builder** definisce il tipo vero e proprio con la relativa funzione di calcolo.
- **BuilderHandler** racchiude una lista di elementi di un determinato tipo gestendo il calcolo di similarità tra di esse.

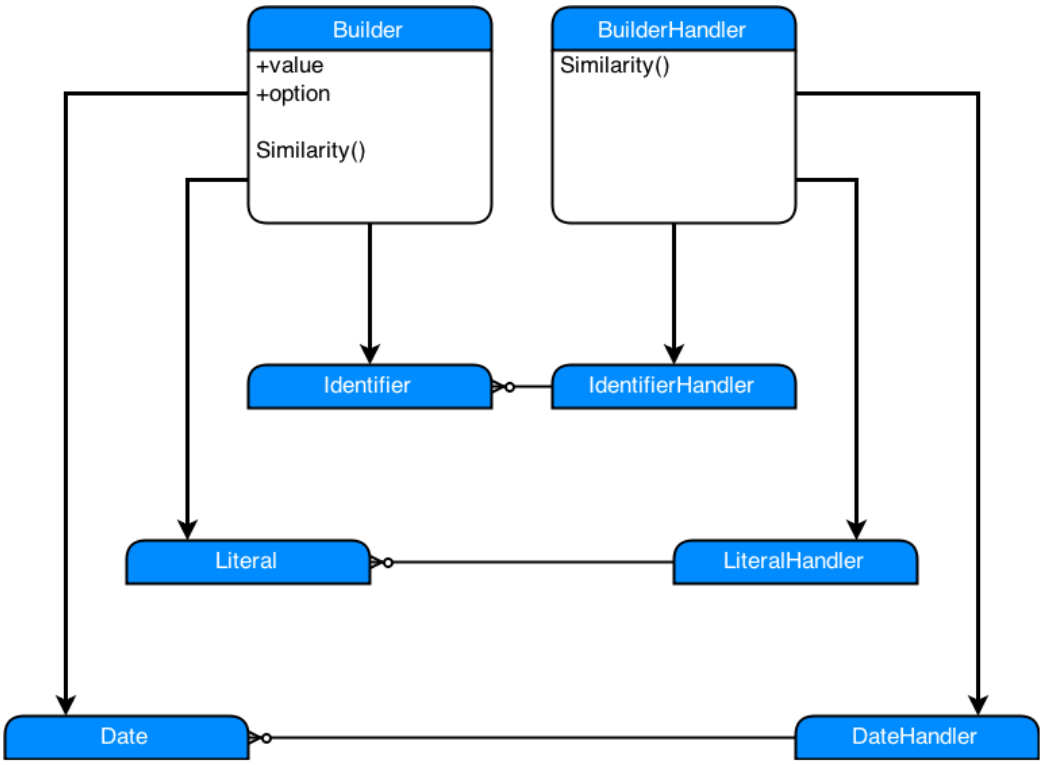

Figura 3.4: Diagramma Entità/relazione dei tipi di input

3.5 Sistema delle Similarità

Ogni elemento utilizzato nell'algoritmo di disambiguazione ha una funzione *Similarity* che permette di ottenere un valore di Similarità in percentuale calcolato tramite apposite funzioni precedentemente impostate nel file di configurazione; è stata definita un'interfaccia *Similarity* ed stata implementata nelle seguenti classi: *Author*, *Builder* e *BuilderHandler*

Listing 3.6: Interface Similarity

```
public interface Similarity<T> {
    public <T> T Similarity(T t);
}
```

L'algoritmo si limita a richiamare la funzione Similarity di ogni autore con ogni altro autore della lista in modo univoco. La funzione Similarity di ogni

autore richiama a sua volta le funzioni Similarity dei BuilderHandler delle 3 HasMap e che loro volta richiameranno le Similarity dei Builder ottenendo così tre diversi livelli di profondità:

1. **Similarità tra Autori:** funzione Similarity di *Author*. Viene richiamata la funzione di ogni BuilderHandler delle tre HashMap calcolando poi la Similarità utilizzando i pesi specificati nel file di configurazione. Il Calcolo della Similarità segue la seguente formula:

$$AuthorSimilarity = \frac{\sum_{i=0}^{n} g_i \cdot w_i}{\sum_{i=0}^{n} w_i} \tag{3.1}$$

   n è il numero delle similarità tra campi, g è il grado di similarità del campo e w è il peso di similarità del campo

Listing 3.7: Le tre *HashMap* della classe *Author*

```
public Map<String, IdentifiersHandler> Identifiers;
public Map<String, LiteralsHandler> Literals;
public Map<String, DatesHandler> Dates;
```

2. **Similarità tra Campi:** funzione Similarity di *BuilderHandler*. Vengono richiamate le funzioni di Similarità di tutti i Builder appartenenti al BuilderHandler: il calcolo della similarità a questo livello di profondità varia in base al tipo trattato.

3. **Similarità tra Valori:** funzione Similarity di *Builder*. Vengono messi a confronto i valori di ogni coppia di Builder; la funzione di similarità è diversa per ogni tipo.

3.5.1 Calcolo di Similarità: Literal

Le liste di valori di questo tipo vengono messe a confronto cercando la coppia di Literal con la più alta similarità possibile.
Prendiamo, ad esempio, le seguenti LiteralHandler

Tabella 3.2: Lista di 2 LiteralHandler

Literal_1	**Literal_2**
Paul	Amid
F.	Paul

La combinazione migliore di Literal calcolata è *Paul* con Similarità del 100%.

Distanza di Levenshtein

La funzione per il calcolo della Similarità tra Literal, utilizza la distanza di Levenshtein [Lev66] e può essere definita come il numero di modifiche elementari (cancellazione, sotituzione, inserimento) necessarie per trasformare una stringa A in una stringa B [RY98]. Per trasfomare per esempio *elogio* in *pelosi* occorrono le seguenti modifiche:

1. *elogio* $\Rightarrow$ *pelogio*, inserimento di *p*
2. *pelogio* $\Rightarrow$ *pelosio*, sostituzione di *g* con *s*
3. *pelosio* $\Rightarrow$ *pelosi*, cancellazione di *o*

quindi la distanza tra le due stringhe è 3.

La funzione Similarity di un Literal utilizza l'algoritmo di Levenshtein per trovare la distanza tra le due stringhe calcolando il grado di Similarità secondo la formula

$$g = 100 - (\frac{100}{l} \cdot (d + c)) \tag{3.2}$$

dove l è la lunghezza massima tra le due stringhe, d è la distanza di Levenshtein e c è il numero di sostituzioni

Il numero di sostituzioni fa parte del gruppo delle opzioni disponibili, quindi può essere escluso dalla formula.

3.5.2 Calcolo di Similarità: Identifier

Il calcolo della Similarità di questo tipo viene eseguito nel IdentifierHandler e non in Identifier: vengono messe a confronto le 2 liste di valori e viene contato esattamente il numero delle coppie di autori uguali. Il calcolo della similarità di Identifier si limita ad applicare l'algoritmo di Levenshtein e a controllare che la distanza sia uguale a 0, e cioè che le due stringhe abbiano gli stessi caratteri. Il calcolo finale segue la seguente formula:

$$g = \frac{c \cdot 100}{c_{max} + (c_{min} - c)} \tag{3.3}$$

dove c è il numero di coppie di valori con distanza di Levenshtein pari a 0

3.5.3 Calcolo di Similarità: Date

Il calcolo della Similarità di tipo Date consiste nel determinare periodi di tempo e di cercare eventualmente delle sovrapposizioni. Il calcolo può avvenire in tre diversi modi:

1. se non esiste sovrapposizione di periodi (nessun periodo della prima lista è uguale alla seconda) viene attribuito un grado prefissato nel file di configurazione; in caso contrario sarà 0. Ad esempio A{2006,2007} e B{2002,2004} non hanno sovrapposizione.

2. se l'intervallo di Date di una lista appartiene all'intervallo dell'altra lista, viene attribuito un grado 100: ad esempio l'intervallo 2003-2004 appartiene all'intervallo 2000-2010.

3. viene contato il periodo consecutivo più lungo possibile includendo sia le date adiacenti che le stesse. Siano A = {2001-2004-2006} e B = {2003-2005} il risultato del conteggio segue la seguente procedura:

   (a) $A + B = \{2001 - 2003 - 2004 - 2005 - 2006\}$ fusione dei dati ed ordinati in grandezza.

   (b) $C_1 = \{2001\}$ conteggio 1 su 5

   (c) $C_2 = \{2003 - 2004 - 2005 - 2006\}$ conteggio 4 su 5

   il grado viene calcolato seguendo la seguente formula:

$$g = \frac{100}{n} * c_{max} \tag{3.4}$$

   dove n è il numero valori totali, c_{max} è il conteggio più alto ottenuto.

3.6 Sistema dei Setting

Tutte le informazioni contenute nel file di configurazione sono presenti sotto forma di classi Java. Nella sezione 3.3.1 è stata descritta nel dettaglio la struttura del del json mentre di seguito viene presentata la classe che rappresenta la lista di Setting

Listing 3.8: Classe che definisce un'impostazione

```
public class Settings {

    public GlobalSetting global_setting;
    public JsonObject configuration;
    public List<Param> params;
```

```
}
```

GlobalSetting ha i parametri generali dell'impostazione.

Listing 3.9: Classe GlobalSetting

```
public class GlobalSetting {

    public String global_threshold;
    public String global_checks;

}
```

il JsonObject *configuration* è la lista di Setting letta dal file di configurazione che funge da backup e fa parte del json in output mentre

la terza variabile è la lista di Param *params*, ovvero i parametri di ogni campo.

Listing 3.10: Classe Setting

```
public class Param {

    public String key;
    public String type;
    public String weight;
    public String threshold;
    public String limit;

    public Option options = new Option();

}
```

La classe Option rappresenta la lista di opzioni disponibili per cambiare la funzione di Similarità di quel determinato campo.

Listing 3.11: Classe Option

```
public class Option {

    public boolean mwa = false; // Multiple Words Analyzer
    public double cor = 2.0; // the Cost Of Replacement for each
        char
    public boolean type_error = false; // consider typing errors:
        jack almost similar to jacks ALPHA
    public boolean spaces = false; //Remove the spaces between words
    public boolean dots = false; // remove all dots
    public String abbreviation = 50; // Grade of abbreviation string
    public boolean arw = false; // Auto Recalibrating Weight, (Must
        be unique in the config. file)
    public String dct = "a50" // Date calculate type
}
```

Alcune di queste Opzioni sono state appena pianificate oppure non testate:

- **mwa** (Multiple Words Analyzer) può essere utilizzata nell'analisi di testi o campi con parole multiple: la similarità tiene conto del numero di parole esattamente uguali.
- **cor** (Cost of Replacement) definisce quanto deve pesare l'operazione di sostituzione nell'algoritmo di Levenshtein
- **type_error** (Errore di battitura) determina se una stringa è stata scritta digitando dei caratteri sbagliati adiacenti a quelli corretti, come ad esempio *Slardar* e *Skardar* la *l* e la *k* nella tastiera sono adiacenti. Uno dei limiti di questa opzione deriva dal tipo di tastiera utilizzata. Per adesso vengono analizzate 3 layout: Inglese, Italiano, Tedesco incidendo sulla performance dell'algoritmo in modo considerevole.
- **spaces** rimuove gli spazi dalla stringa compattando tutti i caratteri: *Paul Amid Slardar* diventa *PaulAmidSlardar*

- **dots** rimuove i punti dalla stringa: *Paul A. Slardar* diventa *Paul A Slardar*

- **abbreviation** nel caso di una delle due stringhe formata da un carattere e questo sia esattamente l'iniziale dell'altra stringa, la similarità sarà del grado indicato nel file di configurazione

- **arw** (Auto Recalibrating Weight) è una Opzione nata principalmente per la fase di Analisi di un Dataset, aiutando a determinare il peso di un campo indipendentemente da tutti i parametri e gradi ottenuti. Nel momento in cui viene calcolata la similarità di autore viene ridefinito il peso di questo determinato campo risolvendo la seguente equazione:

$$x = \frac{\sum_{i=0}^{n-1} g_i w_i - T_{global} \sum_{i=0}^{n-1} w_i}{T_{global}} \tag{3.5}$$

  dove n è il numero dei campi, T_{global} è la threshold globale dell'impostazione, g è il grado di similarità di campo e w è il peso di similarità

- **dct** (Date Calculate Type) serve per determinare quale funzione di similarità utilizzare nel tipo Date, ed, eventualmente, decidere quanto debba essere il grado di similarità in caso di successo.

3.7 Descrizione del Modello di configurazione

Nel modello di configurazione i Setting fanno riferimento ad un numero significativo di concentrazioni ricavate dall'analisi; ognuna di esse ha delle proprietà uniche, il cui unico scopo è quello di disambiguare il maggior numero di autori possibili senza commettere errori. Le concentrazioni sono rappresentate dai seguenti Setting:

- **stesso nome e stesso cognome**: l'omonimia è la concentrazione più comune che si possa trovare in un qualsiasi dataset. Il Setting utilizzato è il seguente:

Listing 3.12: Setting: stesso nome e stesso cognome

```
"global_setting": {
  "global_threshold": "75",
  "global_checks": "0"
},
"param": [
  {
    "key": "person",
    "type": "Identifier",
    "weight": "0",
    "threshold": "0",
    "limit": "100"
  },
  {
    "key": "family_name",
    "type": "Literal",
    "weight": "4",
    "threshold": "100",
    "limit": "100",
    "options": {
      "spaces": true,
      "dots": true
    }
  },
  {
    "key": "given_name",
    "type": "Literal",
    "weight": "3",
    "threshold": "100",
    "limit": "100",
    "options": {
```

```
            "abbreviation":"50",
            "spaces": true,
            "dots": true
          }
        },
        {
          "key": "full_name",
          "type": "Literal",
          "weight": "3",
          "threshold": "0",
          "limit": "100",
          "options": {
            "spaces": true,
            "dots": true
          }
        },
        {
          "key": "coauthors_list",
          "type": "Identifier",
          "weight": "2",
          "threshold": "0",
          "limit": "100"
        },
        {
          "key": "publications",
          "type": "Identifier",
          "weight": "1",
          "threshold": "0",
          "limit": "100"
        },
        {
```

```
62          "key": "publications_pub_years",
63          "type": "Identifier",
64          "weight": "0.5",
65          "threshold": "0",
66          "limit": "100"
67        }
68      ]
```

È l'unico Setting che utilizza tutti i campi della struttura così da permettere di trovare le differenze tra gli autori omonimi. Viene sempre riconosciuta con più importanza al nome e cognome come bilanciamento finale di similarità.

- **stesso cognome e nome simile al 50%** questo Setting va a disambiguare le persone che hanno nel nome un'abbreviazione:

Listing 3.13: Setting: stesso cognome e nome simile al 50%

```
1       "global_setting": {
2       "global_threshold": "61.5",
3       "global_checks": "0"
4     },
5     "param": [
6       {
7         "key": "person",
8         "type": "Identifier",
9         "weight": "0",
10        "threshold": "0",
11        "limit": "100"
12      },
13      {
14        "key": "family_name",
15        "type": "Literal",
```

```
        "weight": "4",
        "threshold": "100",
        "limit": "100",
        "options": {
          "spaces": true,
          "dots": true
        }
      },
      {
        "key": "given_name",
        "type": "Literal",
        "weight": "3",
        "threshold": "50",
        "limit": "50",
        "options": {
          "abbreviation":"50",
          "spaces": true,
          "dots": true
        }
      },
      {
        "key": "coauthors_list",
        "type": "Identifier",
        "weight": "2",
        "threshold": "0",
        "limit": "100",
        "options": {
          "arw": false
        }
      }
    ]
```

Viene impostato inizialmente il campo del nome, limitando inferiormente e superiormente la similarità al 50%, specificando come opzioni l'utilizzo di *abbreviation* con valore 50%. In altre parole: se il campo nome è un'abbreviazione viene controllata la prima lettera dell'altro nome e se queste coincidono la similarità non viene calcolata sull'unica lettera in comune ma viene riconosciuta importanza maggiore fino ad arrivare al 50%. I nomi *A.* e *Amida*, ad esempio, potrebbero essere uno l'abbreviazione dell'altro viene quindi attribuito un valore di incertezza che è esattamente la metà della massima Similarità ottenibile invece del 20% calcolato normalmente.

- **nome non significativo**: questo Setting ha lo scopo di trovare gli autori che hanno lo stesso cognome ma con il nome completamente diverso o, comunque, poco simile:

Listing 3.14: Setting: nome non significativo

```
1     "global_setting": {
2       "global_threshold": "75",
3       "global_checks": "0"
4     },
5     "param": [
6       {
7         "key": "person",
8         "type": "Identifier",
9         "weight": "0",
10        "threshold": "0",
11        "limit": "100",
12        "delimiter": ";",
13        "identifier": true
14      },
15      {
16        "key": "family_name",
```

```
        "type": "Literal",
        "weight": "4",
        "threshold": "100",
        "limit": "100",
        "delimiter": "",
        "identifier": false,
        "options": {
          "spaces": true,
          "dots": true
        }
      },
      {
        "key": "given_name",
        "type": "Literal",
        "weight": "3",
        "threshold": "0",
        "limit": "50",
        "options": {
          "abbreviation":"50",
          "spaces": true,
          "dots": true
        }
      },
      {
        "key": "full_name",
        "type": "Literal",
        "weight": "3",
        "threshold": "0",
        "limit": "100",
        "options": {
          "spaces": true,
```

```
48            "dots": true
49          }
50        },
51        {
52          "key": "coauthors_list",
53          "type": "Identifier",
54          "weight": "2",
55          "threshold": "0",
56          "limit": "100",
57          "options": {
58            "arw": true
59          }
60        }
61      ]
```

La similarità del nome non deve superare il 50% mentre la lista di coautori deve superare lo 0% di similarità. Per facilitare l'impostazione generale e trovare i giusti parametri di campo e globali, viene usata nel campo della lista di coautori l'Opzione *arw* così da automatizzare il processo.

L'algoritmo disambiguerà in 3 modi diversi il dataset unendo, alla fine, i risultati ottenuti; nel modello non ci sono Setting relativi agli errori nei valori come, ad esempio, quelli di battitura o accenti mancanti poiché comporterebbe la disambiguazione di autori senza questa tipologia di problematica. Il modello proposto deve rimanere una solida partenza per rendere l'algoritmo efficace: inizialmente è sempre necessario osservare come il modello si adatti ad un nuovo input, variandone i parametri con estrema precisione per poi cominciare ad aggiungere e/o diminuire Setting, così da mettere a punto il file di configurazione.

In questo capitolo è stato spiegato come il tool Hodor riesca a disambiguare utilizzando i parametri del file di configurazione e come, grazie al modello di configurazione, venga eseguito l'algoritmo di disambiguazione in grado di disambiguare dataset con problematiche molto comuni. Nel prossimo capitolo verranno, invece, illustrati i risultati ottenuti dai test effettuati con un dataset generato e l'algoritmo parametrizzato dal modello.

Capitolo 4

Valutazione di Hodor

In questo capitolo verranno discussi i risultati dei test effettuati con l'algoritmo parametrizzato dal modello di configurazione.

4.1 Origine dell'algoritmo

Come illustrato nei precedenti capitoli, l'algoritmo in questione è stato parametrizzato dal modello di configurazione: inizialmente Hodor non aveva un file di configurazione ma l'algoritmo era fissato con dei parametri scelti da me; una volta deciso di parametrizzare il problema la difficoltà maggiore si è riscontrata nella fase di ricerca di un algoritmo di partenza al fine di utilizzare correttamente il tool. Durante lo sviluppo si è passati da un dataset di prova molto piccolo, (circa 700 autori) ad un dataset contenente almeno 10000 autori con ovvi problemi di disambiguazione da risolvere. Completata la parte significativa del tool mi sono impagnato nella ricerca dell'algoritmo di disambiguazione: il file di configurazione è servito quindi per effettuare l' analisi sul dataset di prova che, come ricordo, è la fase più difficile. Ho avviato il tool con un numero molto elevato di combinazioni di parametri sempre seguendo gli stessi principi: la svolta è avvenuta utilizzando una configurazione apparentemente senza senso: volevo ottenere le coppie di autori che avessero il cognome simile almeno al 90% e che la lista di coautori fosse

il vero dato sensibile, utilizzando l'opzione *awr*. Il risultato è stato sorprendente: 0.68 precision[8] e 0.51 recall[9]. Pur non essendo il set di parametri che cercavo mi ha permesso di capire cosa non andasse nelle precedenti configurazioni: avevo molte prove con una precision dell'80% ma una recall inutile per le mie analisi, oppure l'esatto contrario.

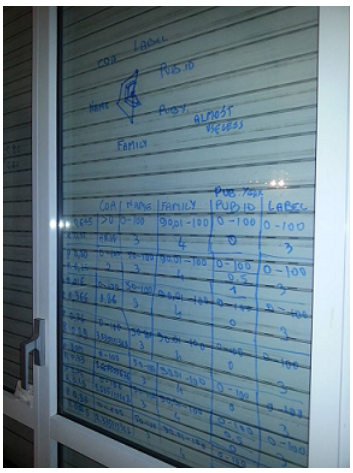

Figura 4.1: Serie di prove effettuate: foto

Tutte le nuove prove hanno determinato quali fossero le concentrazioni dei dati nel dataset: la precision era sostanzialmente nulla ma la cosa non era al momento rilevante. Il seguente grafico rappresenta la concentrazione delle prove significative effettuate:

[8] misura di esattezza

[9] misura di completezza

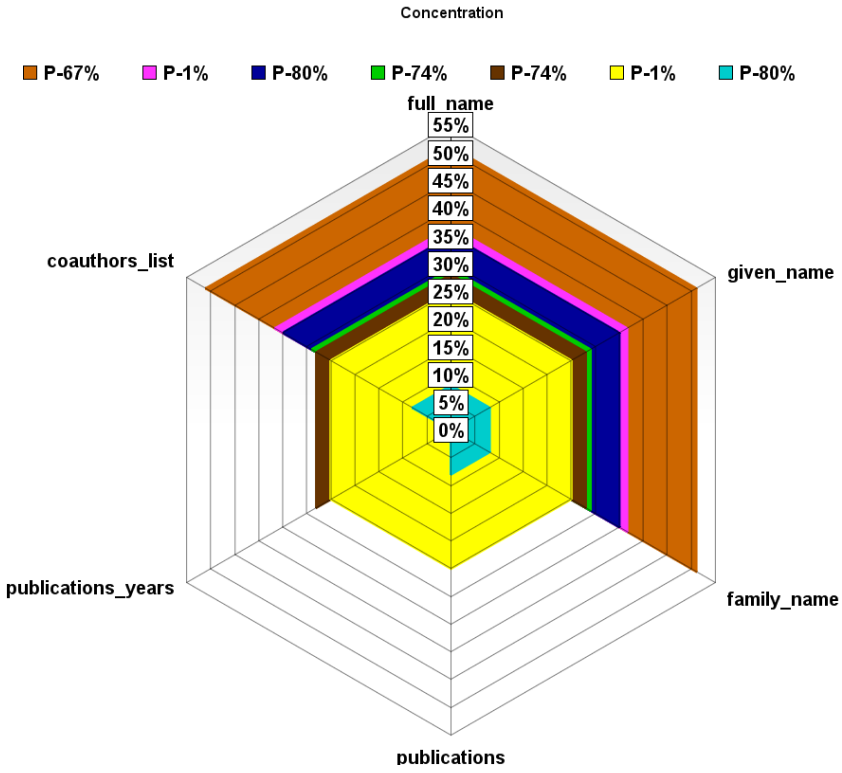

Figura 4.2: Grafico Serie di prove effettuate

In legenda viene riportato il valore di precision P ed il colore relativo all'area che rappresenta il valore di recall: i campi più sensibili alla recall sono *coauthors_list*, *full_name*, *given_name* e *family_name*; ma esistono prove in cui vi e un alto valore di precision grazie all'uso di campi meno sensibili come, ad esempio, *publications* e *publications_years*. Come si può notare dal grafico, quindi, la disambiguazione in questo dataset e frammentata: si è reso necessario trovare delle configurazioni personalizzate per ogni frammento, per cui il risultato finale prende le configurazioni di ogni frammento unendole.

4.2 Complessità e costo dell'algoritmo

L'algoritmo proposto utilizza tre Setting per disambiguare un dataset: ognuna di queste ha un costo di computazione diverso poiché dipende dal

numero di campi che vengono inclusi. Nella fase di elaborazione ogni autore della lista viene confrontato univocamente con tutti gli altri per cui l'algorimo cicla N elementi, per ognuno dei quali verrà effettuato un numero di cicli pari ai campi realmente calcolati. Se ogni autore ad esempio s ha 3 campi, effettueremo un ciclo annidato iniziale con il costo di $O(n^2)$ proseguendo, poi, con la Similarità dei 3 campi, che, nel caso di liste composte da un solo, elemento hanno un costo pari a $O(1)$. Se uno dei campi non soddisfa i requisiti minimi di similarità si passa direttamente all'autore successivo interrompendo il processo di similarità. Definiamo quindi con M i sotto elementi di n: essendo il numero di elementi superiormente limitato il costo finale migliore in ogni caso sarà $O(Mn^2)$. Essendo M una lista con elementi definiti con p, le operazioni di confronto a questo livello hanno un costo di $O(1)$; il costo in caso di campi di liste di valori sarà di $O(pMn^2)$. Esplicitando completamente il costo dell'algoritmo, definiamo con s il numero di Setting nel file di configurazione che, nel nostro caso, è pari a 3 mentre il valore finale sarà quindi $O(pMn^2s)$. L'utilizzo dell'algoritmo di Levenshtein ha inevitabilmente appesantito l'elaborazione soprattutto nel caso in cui i campi contengano testi molto lunghi. In generale le opzioni non sono pesanti a livello computazionale, tranne per l'opzione *type_error* che, se usato in stringhe troppo lunghe, aumenta il costo maggiormente rispetto all'algoritmo di Levenshtein.

4.3 BaseLine effettuate

Ogni algoritmo di disambiguazione deve assolutamente superare dei test preliminari. Tenuto conto della difficoltà nella ricerca di un algoritmo di partenza sono state fissate delle BaseLine in grado di ottenere una disambiguazione parzialmente casuale. Le BaseLine hanno una fase di raccolta degli omonimi o presunti omonimi e una di decisione casuale. La fase di raccolta può essere classificata come *SimilarityBaseLineA* e *SimilarityBaseLineB*.

- **SimilarityBaseLineA** consiste nel prendere gli autori che hanno esattamente lo stesso cognome e che hanno la prima lettera del nome in

comune: ad esempio *Pablo Slardar* e *Paul Slardar*.

- **SimilarityBaseLineB** consiste nel prendere gli autori che hanno esattamente lo stesso cognome e che hanno lo stesso primo nome: ad esempio *Alex De Fabri* e *Alex Paul De Fabri*.

La fase di decisione casuale consta nel decidere se una determinata coppia trovata nella fase precedente fa parte o meno della soluzione. Questo viene stabilito utilizzando una funzione Random, ovvero simulando il lancio di una moneta pesata. Sono state eseguite le due baseline utilizzando i seguenti pesi di insuccesso: 80, 50 e 20 con i seguenti risultati.

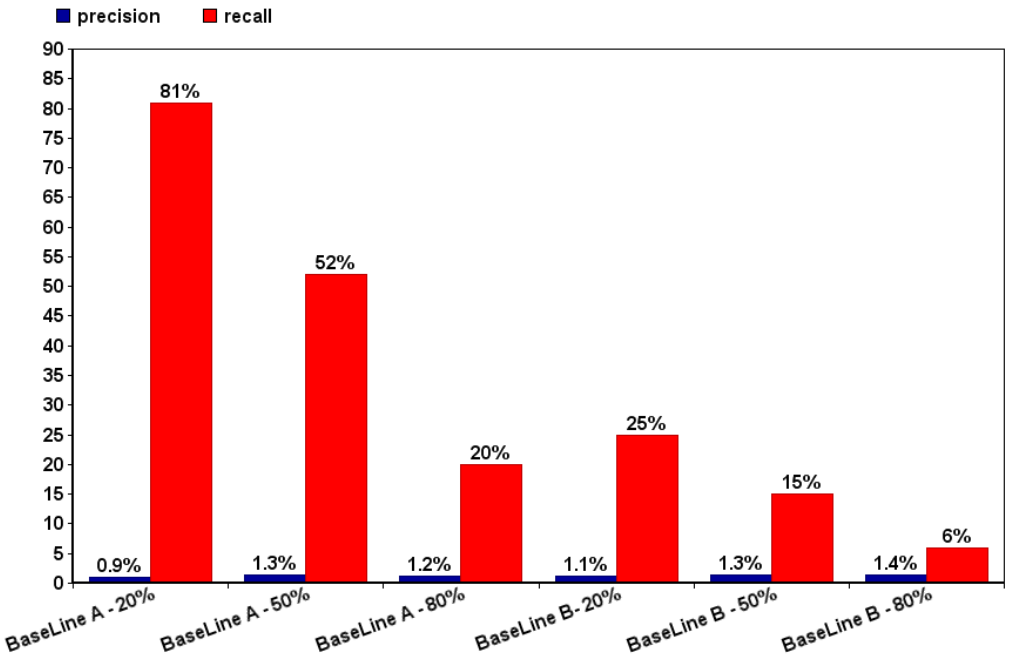

Figura 4.3: Grafico delle precision / recall delle BaseLine

Gli algoritmi delle BaseLine riescono ad ottenere una recall significativa dell'81% ma con precisione completamente inesistente.

4.4 Dataset DBLP Test

Il dataset principale su cui è stato creato il modello di configurazione è stato ricavato da un altro dataset utilizzato nell'algoritmo discusso in questo articolo [TFWZ12]. Tutti i dati provengono da *DBLP computer science bibliography*[10], riadattati dal Dott. Silvio Peroni[11] in modo tale da rendere il dataset più etereogeneo possibile.

Utilizzando, poi, uno script scritto in Node.js è stato convertito a livello strutturale per permettere al tool Hodor di leggerlo correttamente. Il dataset accoglie al suo interno 7447 pubblicazioni e 11724 autori: di questi sono presenti 1295 coppie uniche di autori da disambiguare. L'algoritmo riesce a disambiguare il dataset ottenendo una precision dell'81% e una recall del 70%, il calcolo impiega circa 45min su un computer con un processore Intel Core 2 Quad Q9950 2.83Hz e 8GB DDR3 di RAM. L'impiego di una quarto Setting, che dipendesse essenzialmente dalla minima variazione della similarità della lista di coautori, ha reimpostato l'algoritmo di disambiguazione ottenendo una precision del 71% e una recall del 75% in circa un' ora sempre sulla medesima macchina. Nel seguente grafico sono riportati i dati più significativi ottenuti durante la fase i ottimizzazione dell'algoritmo.

[10] http://dblp.uni-trier.de/db/
[11] http://www.essepuntato.it/

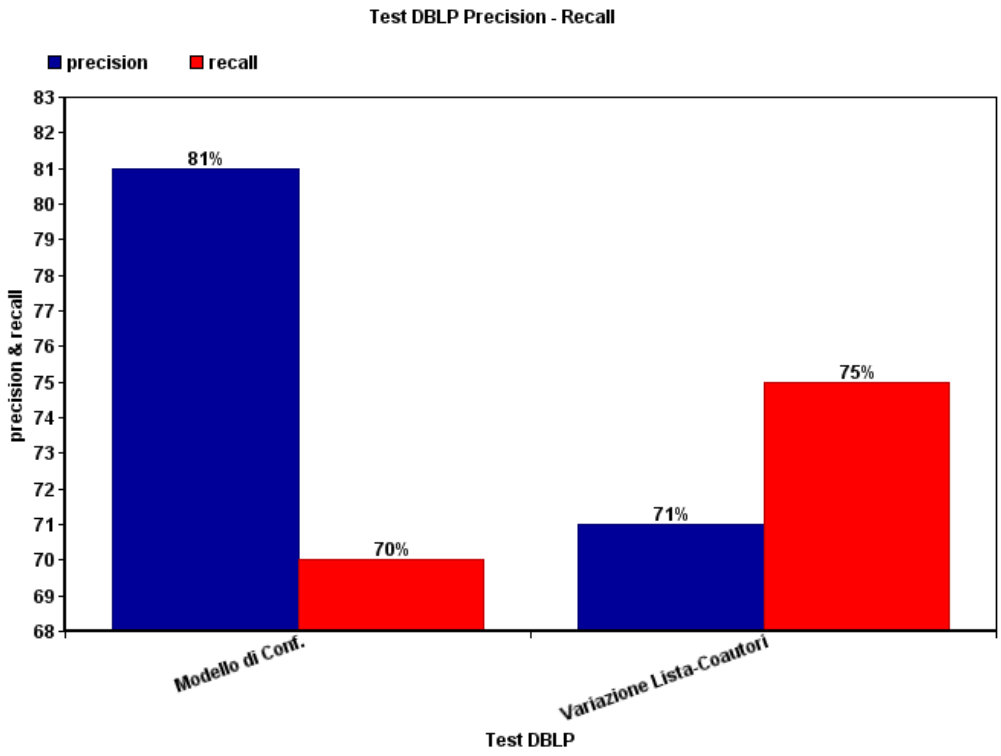

Figura 4.4: Test sul dataset DBLP: precision e recall

4.5 Semantic Lancet Test

Il Semantic Lancet [BCINPV14] è un triplestore RDF supportato da un endpoint SPARQL che permette l'accesso a dati scientifici per scopi di ricerca. Al momento il dataset contiene i metadati dei documenti pubblicati in *Journal of Web Semantics*[12] di Elsevier ed ha 919 autori; essendo un triplestore non ha problemi di accorpamento dei multipli evidenti come nel Dataset DBLP generato, cioè non esistono omonimi perfetti perché essi sono fusi insieme negli stessi record (vedi sezione 1.1 Divisione degli Omonimi).

Sono stati effettuati dei test iniziali sul semantic lancet lasciando il modello di configurazione intatto senza ottenere alcun risultato: si tratta ovviamente di un esito corretto e ampiamente previsto poiché l'algoritmo, così impostato, è destinato a dataset con problemi di accorpamento dei multipli.

[12]http://www.journals.elsevier.com/journal-of-web-semantics/

Abbassando i valori delle threshold globali dei Setting, l'algoritmo tenta di disambiguare il dataset trovando come soluzione circa 25 coppie tutte con nome particolarmente diverso ma con la lista di coautori molto simile e relativa precision del 16% e recall del 11%.

Il test successivo è stato effettuato utilizzando solamente il secondo Setting del modello di configurazione, ma senza tenere conto delle informazioni relative alla lista di coautori e delle informazioni riguardanti le pubblicazioni: i risultati mostrano una precision del 100% e una recall del 46%. Il più delle volte il problema si concentra intorno a casi di multiregistrazione con un solo articolo pubblicato senza alcun riscontro di similarità nella lista di coautori o all'interno di altri campi relativi alle pubblicazioni. Studiando, ancora, il dataset si è riscontrato che il precedente algoritmo ha scartato gli autori che nel nome presentano più parole, come ad esempio per *Paul A.* e *Paul Amid* si è reso necessario l'utilizzo dell'opzione *mwa* permettendo così un calcolo di similarità più accurato con esito più che positivo: precision 84% e recall 79%.

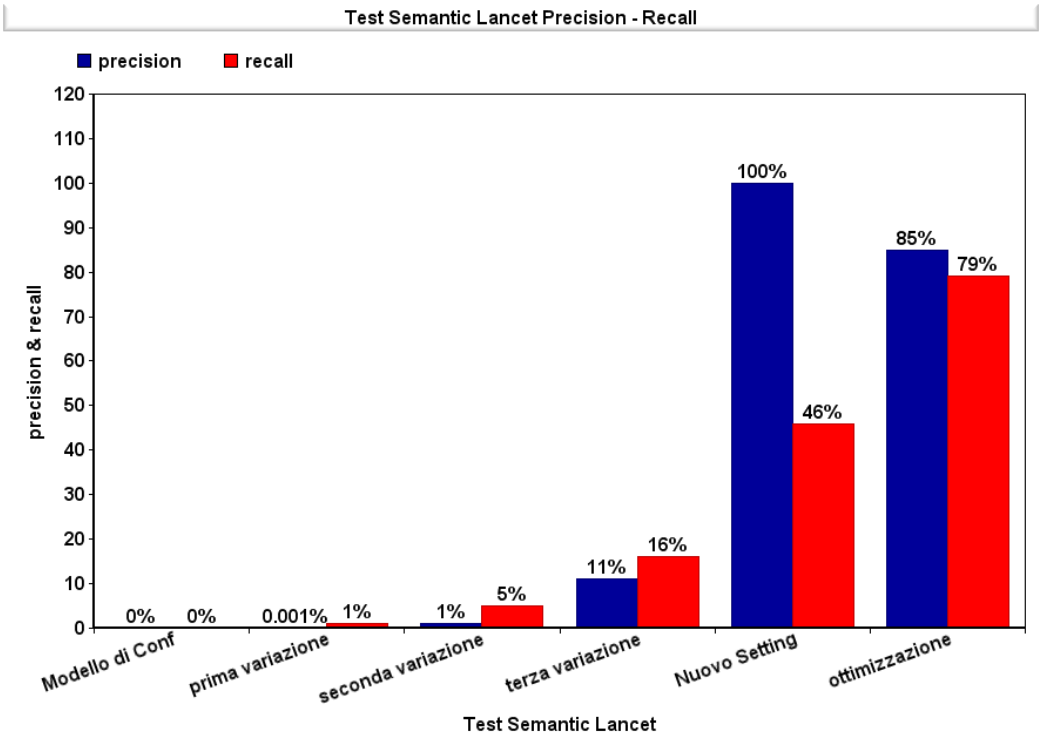

Figura 4.5: Test sul dataset Semantic Lancet: precision e recall

Il grafico rappresenta tutti i risultati ottenuti dai test precedentemente descritti: la prima metà identifica i valori di precision e recall ricavati dal modello di configurazione mentre la seconda i valori che fanno riferimento al file di configurazione modificato. A differenza dei dati riscontrati nei test del dataset DBLP si può notare come la recall e la precision rimangano sempre molto vicine: questo significa che nel Semantic Lancet le concentrazioni dei dati da disambiguare non sono frammentate.

Listing 4.1: Setting ottimale per il Semantic Lancet

```
1     "global_setting": {
2       "global_threshold": "0",
3       "global_checks": "0"
4     },
5     "param": [
```

```
6       {
7         "key": "person",
8         "type": "Resource",
9         "weight": "0",
10        "threshold": "0",
11        "limit": "100"
12      },
13      {
14        "key": "family_name",
15        "type": "Information",
16        "weight": "4",
17        "threshold": "100",
18        "limit": "100",
19        "options": {
20          "spaces": true,
21          "dots": true
22        }
23      },
24      {
25        "key": "given_name",
26        "type": "Information",
27        "weight": "3",
28        "threshold": "50",
29        "limit": "50",
30        "options": {
31          "mwa": true,
32          "spaces": true,
33          "dots": true
34        }
35      }
36    ]
```

Questo Setting infine, è stato aggiunto al modello di configurazione ed stato rieseguito un test sul dataset DBLP: i nuovi parametri hanno permesso all'algoritmo di disambiguare la lista di autori ottenendo il 75% di precision e il 74.5% di recall. L'esito ha dimostrato che esistono ancora molti casi di disambiguazione non parametrizzati nel modello di configurazione, e che quest'ultimo è ancora lontano dal riuscire a impostare l'algoritmo nel modo ottimale così da poter ottenere i valori di precision e recall sopra del 90%.

In questo capitolo sono state illustrate le difficoltà affrontate nella configurazione dell'algoritmo di disambiguazione; soprattutto nella fase di test con il Semantic Lancet sono emersi dei problemi di adattabilità, risolti successivamente grazie alla possibilità di personalizzare l'algoritmo a seconda delle necessità e delle esigenze.

Conclusioni

In questa tesi è stato presentato Hodor, un tool in grado di eseguire un algoritmo di disambiguazione parametrico degli autori: un set di parametri specifico è un modello generale di disambiguazione e discusso come possibile soluzione al problema della disambiguazione.

Con la necessità di disambiguare il triplestore Semantic Lancet è nata l'idea di implementare un sistema parametrico che permettesse la creazione di algoritmi sempre diversi, con lo scopo di seguire l'eventuale cambiamento delle informazioni del dataset.

Il tool in questione è completamente configurabile, in grado di creare algoritmi in base a parametri input opportunamente scelti dall'utente: il punto cardine è proprio l'adattabilità permettendo l-uso di un solo tool per qualsiasi dataset da disambiguare.

Per facilitare l'uso di Hodor, è stato elaborato un file di configurazione usato come punto di partenza nella creazione di un nuovo algoritmo: dopo aver analizzato accuratamente il dataset da disambiguare, sarà possibile rappresentare sotto forma di parametri tutti i casi di disambiguazione che affliggono il set di dati; il tool creerà successivamente l'algoritmo e lo eseguirà elaborando il dataset in input.

L'algoritmo creato dal file di configurazione di default è stato testato su un dataset con evidenti problemi di disambiguazione da risolvere, generato utilizzando le informazioni di 11724 autori (1295 coppie da disambiguare), ottenute da DBLP: i risultati dei test hanno ottenuto una precision dell'81% e una recall del 70%. Come ulteriore prova l'algoritmo è stato testato sul

triplestore Semantic Lancet da 919 autori (34 coppie da disambiguare) non ottenendo disambiguazioni: questo risultato dimostra che l'algoritmo riesce a disambiguare in modo unico soltanto il dataset per cui è stato impostato; dopo aver variato dei parametri del file di configurazione, l'esito è stato infatti 84% di precision e 79% di recall migliorando i valori ottenuti dai test del dataset DBLP.

La difficoltà principale è riuscire ad analizzare nel modo corretto il database da elaborare così da poter creare il giusto algoritmo di disambiguazione: non esiste al momento una procedura guidata per aiutare l'utente nella fase dell'analisi; è una feature purtroppo molto difficile da progettare in quanto il sistema è in grado di accettare qualsiasi tipo di dataset. Per aumentare ulteriormente l'usabilità del tool la prima cosa da migliorare è la lista di opzioni di calcolo disponibili come parametri nel file di configurazione, facilitando di molto la scelta dei parametri, migliorando anche le prestazioni dell'algoritmo.

Per permettere l'uso del tool più ad alto livello, sarà necessario, infine, implementare una GUI chiara ed essenziale in grado di assistere l'utente in tutte le fasi: configurazione, caricamento del dataset (sia remoto che in locale) e scelta del tipo di output.

Bibliografia

[Lev66] Vladimir I. Levenshtein. «Binary codes capable of correcting deletions, insertions, and reversals». In: *Soviet Physics-Doklady* 10.8 (1966).

[RY98] E.S. Ristad e P.N. Yianilos. «Learning string-edit distance». In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 20.5 (mag. 1998), pp. 522–532. ISSN: 0162-8828. DOI: 10.1109/34.682181.

[MAC05] Bradley Malin, Edoardo Airoldi e KathleenM. Carley. «A Network Analysis Model for Disambiguation of Names in Lists». English. In: *Computational & Mathematical Organization Theory* 11.2 (2005), pp. 119–139. ISSN: 1381-298X. DOI: 10.1007/s10588-005-3940-3. URL: http://dx.doi.org/10.1007/s10588-005-3940-3.

[ST09] Neil R. Smalheiser e Vetle I. Torvik. «Author name disambiguation». In: *Annual Review of Information Science and Technology* 43.1 (2009), pp. 1–43. ISSN: 1550-8382. DOI: 10.1002/aris.2009.1440430113. URL: http://dx.doi.org/10.1002/aris.2009.1440430113.

[DGA11] Ciriaco Andrea D'Angelo, Cristiano Giuffrida e Giovanni Abramo. «A heuristic approach to author name disambiguation in bibliometrics databases for large-scale research assessments». In: *Journal of the American Society for Information Science and Technology* 62.2 (2011), pp. 257–269. ISSN: 1532-2890.

DOI: 10.1002/asi.21460. URL: http://dx.doi.org/10.1002/asi.21460.

[TFWZ12] Jie Tang, A.C.M. Fong, Bo Wang e Jing Zhang. «A Unified Probabilistic Framework for Name Disambiguation in Digital Library». In: *Knowledge and Data Engineering, IEEE Transactions on* 24.6 (giu. 2012), pp. 975–987. ISSN: 1041-4347. DOI: 10.1109/TKDE.2011.13.

[BCINPV14] Andrea Bagnacani, Paolo Ciancarini, Angelo Di Iorio, Andrea Giovanni Nuzzolese, Silvio Peroni e Fabio Vitali. *The Semantic Lancet Project: a Linked Open Dataset for Scholarly Publishing*. To appear in Proceedings of Satellite Events of EKAW 2014, Lecture Notes in Artificial Intelligence. 2014. URL: http://speroni.web.cs.unibo.it/publications/bagnacani-in-press-semantic-lancet-project.pdf.

[SMPPH14] Christian Schulz, Amin Mazloumian, AlexanderM Petersen, Orion Penner e Dirk Helbing. «Exploiting citation networks for large-scale author name disambiguation». English. In: *EPJ Data Science* 3.1, 11 (2014). DOI: 10.1140/epjds/s13688-014-0011-3. URL: http://dx.doi.org/10.1140/epjds/s13688-014-0011-3.