UNIVERSITÀ DEGLI STUDI DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Informatica

Elaborato finale

Dandelion e Strephit

Supervisore

Alberto Montresor

Laureando

Edoardo Lenzi

Anno accademico 2018

Ringraziamenti

Prima di addentrarmi nella descrizione del progetto di tesi voglio ricordare coloro che mi sono stati vicini.

Ringrazio la mia famiglia per essermi sempre stata accanto ed avermi supportato (e sopportato) in tutti questi anni, permettendomi di studiare fino a laurearmi; ringrazio in particolare mio fratello Massimo e mia sorella Maria Vittoria che più di chiunque altro mi hanno aiutato nello studio.

Ringrazio i miei zii Chiara e Christian, persone uniche e speciali che da sempre mi sono vicine e mi sostengono.

Ringrazio i miei amici, in particolare Michele, dispensatore dei migliori consigli e delle peggiori critiche, senza il quale non avrei nemmeno scelto questa via.

Ringrazio i miei colleghi di lavoro che mi hanno insegnato moltissimo, facendomi appassionare, di giorno in giorno, sempre più al mio lavoro.

Devo, infine, un ringraziamento particolare ad Alberto Montresor, Davide Setti, Alessio Guerrieri, Ugo Scaiella e Marco Fossati che mi hanno seguito ed aiutato moltissimo in questo percorso di tesi.

A tutti loro va la mia riconoscenza ed i miei più sentiti ringraziamenti.

Contents

Sommario

Il progetto di tesi è il frutto di un tirocinio presso SpazioDati[1] ed una successiva collaborazione al progetto Wikidata StrepHit[2].

La tesi si divide principalmente in tre parti separate, ogniuna delle quali legata al servizio online Dandelion[3], che verrà brevemente introdotto nelle prossime pagine.

Il primo capitolo ha lo scopo di introdurre, al lettore, gli applicativi, le tecnologie ed i servizi usati durante il progetto di tesi, descrivendoli sommariamente; nei tre capitoli successivi, invece, si entrerà maggiormente nel dettaglio del progetto descrivendo i tre ambiti fondamentali su cui si è focalizzata la tesi.

La prima parte del progetto consiste nell'implementazione di un client in C#, finalizzato allo scopo di creare una libreria di metodi "plug&play" per chiamare automaticamente le RESTful-API del servizio Dandelion.

Durante lo sviluppo il codice è stato regolarmente caricato online, sulla piattaforma GitHub, per agevolare la periodica revisione del codice effettuata dal team di SpazioDati; per rendere lo sviluppo più efficiente si è cercato di seguire i principi del TDD (Test Driven Development), affidandosi al servizio di testing automatico Travis per validare automaticamente ogni pull-request su GitHub.

Una volta completata la libreria e l'annessa documentazione è stata compilata la dll e resa pubblica su Nuget; pertanto la libreria è ora facilmente integrabile in qualsiasi progetto C#.

La seconda parte del progetto riguarda l'analisi e il test di un algoritmo presente nel backend di Dandelion al fine di ottimizzarlo. Il task nasce dalla necessità pratica di ridurre la memoria necessaria alle macchine di produzione di SpazioDati per eseguire l'algoritmo senza subire cali prestazionali. Partendo dall'implementazione attualmente in uso si sono studiate delle implementazioni alternative per massimizzare la velocità dell'algoritmo e minimizzare la memoria occupata dalle strutture dati di appoggio dell'algoritmo.

L'ultima parte gravita attorno al progetto StrepHit di Wikidata, un progetto nato un anno fa in FBK ed approvato dalla comunity di Wikidata. Il progetto nasce per arricchire i database di Wikidata con riferimenti a fonti esterne (tendenzialmente altri siti web) in modo da dare all'utente, fruitore dell'enciclopedia, informazioni sempre più corrette, validate anche da un siti esterni e non più solo dalla comunity di Wikipedia/Wikidata.

Il progetto è fortemente legato a Dandelion perchè nelle logiche interne dello script di StrepHit si fa largo uso dei servizi offerti da Dandelion per l'analisi testuale delle pagine dei siti esterni. L'ultima attività consiste nell'implementazione di uno script in Python (versione 2.7) utile ad arricchire un dataset di quickstatements aggiungendo riferimenti a proprietà di Wikidata. Per fare ciò è stato necessario studiare il linguaggio Python ed il funzionamento generale di Wikidata per poi realizzare uno script capace di interfacciarsi con l'endpoint SPARQL[4] di Wikidata.

[1] **SpazioDati** è un'azienda con una sede operativa a Trento ed una a Firenze che si occupa principalmente di tecnologie Big Data e Semantic Web. Come riportato sul loro sito spaziodati.eu stanno costruendo un knowledge-graph di alta qualità, accessibile attraverso delle semplici API su dandelion.eu.

[2] **StrepHit** è un progetto di Wikidata nato l'anno scorso con il fine di implementare uno script capace di analizzare testi e tradurli in Wikidata statements (o QuickStatements).

[3] **Dandelion** è un servizio online fornito da SpazioDati che mette a disposizione dell'utente servizi di analisi semantica testuale.

[4] Wikidata si basa su un knowledge base RDF e mette a disposizione un end-point per eseguire query su quest'ultimo; SPARQL è il linguaggio di query definito da W3C per il framework RDF, adottato da questo end-point.

1 Introduzione

1.1 Dandelion

Dandelion è un servizio online fornito da SpazioDati che mette a disposizione dell'utente servizi di analisi semantica testuale; grazie ad esso è possibile, dato un testo, estrarne le entità semantiche principali (*Entity Extraction*), trovare la lingua in cui è stato scritto (*Language Detection*), classificarlo secondo modelli definiti dall'utente stesso (*Text Classification*) e analizzarne la semantica per capire i sentimenti che l'autore ci vuole trasmettere (*Sentiment Analysis*).

Dandelion ha anche altre due RESTful-API[1] che permettono di confrontare due testi generando un indice di similitudine fra i due (*Text Similarity*) e un motore di ricerca di entità di Wikipedia (*Wikisearch*), nel caso si voglia trovare il titolo di un contenuto senza conoscerlo a priori.

L'enciclopedia alla base di Dandelion è Wikipedia, anche se a volte fra i due si colloca come mediatore dbpedia[2], un progetto italiano per l'estrazione di informazioni semi-strutturate da Wikimedia.

Per poter usare gli end-point di Dandelion basta registrarsi sul portale dedicato e generare un token che andrà inserito come query parameter nelle chiamate https alle API[3].

[**TODO** descrizione dettagliata di Dandelion]

1.2 GitHub e Travis

Per il versioning del codice si è scelto di usare il software git[4], appoggiandosi alla piattaforma GitHub per creare repository pubbliche.

Le repository riguardanti il progetto di tesi sono:

- Repository contenente il testo della tesi scritto in LaTeX[5]
- Repository contenente lo script in Python per il progetto StrepHit[6]
- Repository contenente il codice Java con le varie implementazioni dell'algoritmo per il calcolo della relatedness (la seconda parte del progetto)[7]
- Repository contenente il codice sorgente del client C#[8]

Per quanto riguarda il testing automatico è stata scelta la piattaforma Travis che permette di agganciare una repository GitHub su cui eseguire automaticamente i test ad ogni commit.

Per fare ciò è necessario inserire nella repository un file di configurazione chiamato .travis.yml con i comandi necessari ad eseguire i test.

Per esempio nel caso del client C# il file yml contiene le seguenti istruzioni:

[1] **REST** è un tipo di architettura software per la trasmissione di dati tramite il protocollo HTTP. Fonte

[2] **DBpedia** è un progetto che ha portato alla creazione di un OKG (Open Knowledge Graph) edificato su informazioni estratte da progetti Wikimedia. Fonte

[3] La documentazione delle **API di Dandelion** è disponibile all'indirizzo: https://dandelion.eu/docs/.

[4] **Git** è un software open source gratuito per gestire il versioning del codice. Sito ufficiale

[5] `https://github.com/EdoardoLenzi9/Dandelion.LaTeX`

[6] `https://github.com/EdoardoLenzi9/Wikipedia.StrepHit`

[7] `https://github.com/EdoardoLenzi9/Dandelion.Relatedness`

[8] `https://github.com/EdoardoLenzi9/SpazioDati.Dandelion-eu`

Listing 1.1: File configurazione travis.yml per progetti C#

```
language: csharp
dist: trusty
mono: none
dotnet: 2.0.3

install:
- dotnet restore

script:
- dotnet build
- dotnet test SpazioDati.Dandelion.Test/SpazioDati.Dandelion.Test.csproj
```

Mentre per lo script in Python la sintassi è la seguente:

Listing 1.2: File configurazione travis.yml per progetti Python

```
language: python
python:
  - "2.7"
install:
  - pip install -r requirements.txt
script:
  - python test.py
```

Come si nota dalle righe sopra, basta specificare il linguaggio, la versione ed i comandi[9] per installare e lanciare i test.

1.3 Docker

Per la seconda parte del progetto è stato usato Docker[10] per creare un ambiente "chiuso", ad hoc, dove poter testare al meglio gli algoritmi senza doversi preoccupare di fattori esterni che potrebbero falsare il risultato del test.

Docker permette di creare un'immagine a partire da un file di testo chiamato "Dockerfile" il quale contiene tutte le informazioni per eseguire la build dell'immagine; in pratica il Dockerfile è una sorta di descrittore dello stato di una macchina virtuale che genera, una volta lanciata la build, una macchina virtuale persistente chiamata immagine.

Un container è un istanza di un'immagine che diventa quindi non persistente (qualsiasi modifica allo stato della VM verrà perso una volta chiuso il container).

Una delle caratteristiche più utili di Docker è la semplicità con cui è possibile interfacciare container diversi condividendo aree di memoria sul disco e porte TCP.

L'idea di base per monitorare le prestazioni dell'algoritmo è stata quella di compilare il progetto Java con il tool Ant ed esportare il file Dandelion.jar; è seguita la creazione di un Dockerfile basato sull'immagine openjdk, scaricata da hub.docker, contenente un'implementazione open source di Java SE (Java Platform, Standard Edition) a partire dalla versione 7.

Listing 1.3: Dockerfile

```
FROM openjdk:latest
COPY . /usr/src/dandelion
WORKDIR /usr/src/dandelion
CMD ["java",
    "-javaagent:./lib/jmx_prometheus_javaagent-0.3.1.jar=8080:./lib/configs.yaml",
    "-jar", "/usr/src/dandelion/dist/Dandelion.jar"]
```

[9] Travis esegue i test in ambiente Linux, quindi i comandi devono essere eseguibili in una bash-shell Linux

[10] **Docker** è un progetto open source, cross-platform, che funge da "container platform provider" permette il deployment automatico di applicativi dentro ambienti virtuali chiamati container. Sito ufficiale

La prima direttiva FROM specifica l'immagine di base su cui verrà edificata una nuova immagine, le successive due direttive COPY e WORKDIR servono rispettivamente a copiare tutto il contenuto della cartella corrente dentro l'ambiente virtuale alla path specificata (*/usr/src/dandelion*) e a settare quest'ultima come *working directory* da cui partirà il container docker.

Infine la direttiva CMD specifica il comando che verrà eseguito una volta finito il setup dell'ambiente, in questo caso viene eseguito Dandelion.jar con un particolare javaagent chiamato JMX[11]; questo javaagent è un exporter che si occupa di monitorare l'applicativo e di esportare in tempo reale le relative metrics che vengono esposte in un server locale su una porta a scelta (8080 in questo caso).

Per evitare di effettuare una build del Dockerfile ogni volta che modifichiamo il codice sorgente Java è possibile mappare la cartella corrente con la working directory dell'immagine tramite l'opzione -v che crea un volume condiviso; tramite l'opzione -p invece è possibile mappare una porta tcp interna al docker su una esterna.

Listing 1.4: Run Docker

```
1  docker run -it -p 8080:8080 -v <current-dir>.:<docker-working-dir> \
2  --name dandelion-volume dandelion-container
```

1.4 Prometheus e Grafana

Parallelamente al primo container vengono istanziate altre due immagini, scaricate da hub.docker, chiamate prom/prometheus e grafana/grafana le quali permettono di filtrare le metrics generate dall'exporter e di visualizzarle direttamente su dei grafici.

1.4.1 Prometheus

Prometheus è un applicativo open source rilasciato sotto Apache 2 License su GitHub, permette di salvare in memoria e/o sul disco le metrics raccolte da un exporter e mette a disposizione un linguaggio di query che permette di filtrare le metrics a piacimento.

È anche possibile rappresentare i risultati delle query direttamente su dei grafici statici tuttavia per questo aspetto si è scelto di usare Grafana.

1.4.2 Grafana

Grafana è anch'essa un software open source volto alla creazione di dashboard dinamiche tipicamente consistenti in grafici e widget di monitoraggio di vario tipo, funzionalmente molto completi e gradevoli sotto il profilo estetico.

Grafana è nativamente predisposta per integrarsi con Prometheus, basta indicare la porta tcp su cui è settato Prometheus e la query che si desidera visualizzare.

1.5 Wikidata e StrepHit

Wikidata è un progetto gratuito e open source legato a Wikipedia che mette a disposizione dell'utente un knowledge base accessibile sia dal portale wikidata.org che tramite un end-point SPARQL.

I dati presenti nel knowledge base sono dati strutturati secondo le logiche del semantic web e adottano, quindi, il framework RDF (Resource Description Framework) proposto da W3C.

Questo servizio rende possibile l'interazione con il knowledge base di Wikidata direttamente tramite query veicolate da chiamate http ad un end-point SPARQL ed è facilmente integrabile con qualsiasi altro knowledge base di terze parti che segua gli stessi schemi e modelli RDF.

StrepHit è un progetto nato l'anno scorso, seguito da un team in FBK, coordinato da Marco Fossati; il progetto prevedeva la creazione di un applicativo scritto prevalentemente in python capace di analizzare testi e tradurli in Wikidata statements (o QuickStatements).

QuickStatemens è un tool scritto da Magnus Manske che definisce un linguaggio volto a descrivere/-modificare elementi Wikidata. Questo linguaggio è ormai diventato uno standard per quanto riguarda

[11] **JMX Exporter** è un progetto GitHub open source che fornisce un javaagent capace di esportare ed esporre metrics per il software Prometheus.

Wikidata tant'è che la nomenclatura degli elementi del knowledge base seguono gli standard dettati da Quickstatements.

1.5.1 SPARQL

SPARQL è il linguaggio di query definito da W3C per RDF. Le query sono molto affini al SQL, infatti ne condividono molte key-word, la differenza sta nel fatto che i dati ora seguono i modelli RDF pertanto avranno sempre un item, una property e un value.

Wikidata mette a disposizione un'interfaccia che permette di lanciare manualmente query SPARQL sul knowledge base e un end-point per contattare lo stesso servizio con chiamate http.

2 Client C#

2.1 Struttura Generale

Per l'implementazione si è cercato di seguire le best practices, dettate dalle linee guida Microsoft, per la stesura del codice. Sono stati adottati, come pattern di programmazione, dependency injection, MVC (per quanto possibile) e TDD, appoggiandosi a librerie esterne Nuget[1] quali Newtonsoft.Json e SimpleInjector. I metodi sono asincroni e le property thread-safe.

La solution è stata partizionata in vari progetti; il progetto Business contiene la business-logic (comprendente servizi, metodi estensione, l'implementazione del client e un wrapper del Container di SimpleInjector) e utilizza i modelli di Domain.

In Test sono contenute le classi di testing e le fixture XUnit[2], mentre in Documentation sono presenti i file generati dal tool Wyam[3] per la documentazione.

2.2 Il Client

Dalla documentazione ufficale delle API di Dandelion si evince che ogni end-point richiede uno o più testi da analizzare ed una serie di parametri, alcuni obbligatori ed altri opzionali; pertanto sono stati creati dei modelli, coerenti con tali definizioni, che, passati in argomento a dei servizi, ritornano i DTO delle risposte degli end-point di Dandelion.

Ogni servizio controlla che i parametri inseriti dall'utente rispettino i constraints definiti nella documentazione e costruisce, tramite i metodi `ContentBuilder()` e `UriBuilder()`, un dizionario di parametri e l'URI che identifica l'end-point.

Infine il servizio chiama il metodo generico `CallApiAsync()`, passandogli il content, la URI ed il metodo HTTP; nel metodo di chiamata si è scelto di gestire separatamente i metodi HTTP GET e DELETE dato che essi non permettono il passaggio parametri nel boby della chiamata. Pertanto è stato necessario inviare il content come query parameter facendone il percent-encoding.

Dandelion ammette sia chiamate GET che POST ai servizi, tuttavia nel caso del metodo GET non è garantito il corretto funzionamento del servizio per testi che superano i 2000 caratteri; pertanto nel caso in cui l'utente non specifichi il metodo HTTP, di default verrà usato il metodo POST.

Se la chiamata è andata a buon fine il JSON di ritorno viene mappato in un DTO specifico, a seconda del servizio scelto, e restituito direttamente all'utente.

Listing 2.1: Metodo generico del client per le chiamate HTTP

```
1  public Task<T> CallApiAsync<T>(string uri, List<KeyValuePair<string, string>> content,
       HttpMethod method = null)
2  {
3      var result = new HttpResponseMessage();
4      if (method == null)
5      {
6          method = HttpMethod.Post;
7      }
8
9      if (_client.BaseAddress == null)
10     {
11         _client.BaseAddress = new Uri(Localizations.BaseUrl);
12     }
13
```

[1] **Nuget** è un package manager per .NET. Sito ufficiale

[2] **XUnit** è una libreria Nuget che mette a disposizione un tool per eseguire unit test. Repository GitHub

[3] **Wyam** è un tool che permette l'esportazione della documentazione del codice C# (sottoforma di tag xml) in pagine html. Sito ufficiale

```
14      return Task.Run(async () =>
15      {
16          if (method == HttpMethod.Get)
17          {
18              string query;
19              using (var encodedContent = new FormUrlEncodedContent(content))
20              {
21                  query = encodedContent.ReadAsStringAsync().Result;
22              }
23              result = await _client.GetAsync($"{uri}/?{query}");
24          }
25          else if (method == HttpMethod.Delete)
26          {
27              string query;
28              using (var encodedContent = new FormUrlEncodedContent(content))
29              {
30                  query = encodedContent.ReadAsStringAsync().Result;
31              }
32              result = await _client.DeleteAsync($"{uri}/?{query}");
33          }
34          else
35          {
36              var httpContent = new HttpRequestMessage(method, uri);
37              if (content != null)
38              {
39                  httpContent.Content = new FormUrlEncodedContent(content.ToArray());
40              }
41              result = await _client.SendAsync(httpContent);
42          }
43          string resultContent = await result.Content.ReadAsStringAsync();
44          if (result.StatusCode == System.Net.HttpStatusCode.RequestUriTooLong)
45          {
46              throw new ArgumentException(ErrorMessages.UriTooLong);
47          }
48          if (!result.IsSuccessStatusCode)
49          {
50              throw new Exception(resultContent);
51          }
52          return JsonConvert.DeserializeObject<T>(resultContent);
53      });
54  }
```

2.3 Testing

La maggior parte dei servizi è stata testata tramite il tool XUnit; principalmente sono stati eseguiti test di validazione, data la natura della libreria, appoggiandosi ad una fixture comune a tutti i test per l'inizializzazione dei servizi.

è stato usato fin da subito il servizio online di testing automatico Travis, tramite il quale è stato possibile validare ogni rilascio facendo partire automaticamente i test con l'evento di push di Git.

2.4 Documentazione

Infine le classi e i metodi più rilevanti sono stati commentati tramite i tag XML specificati nella documentazione Microsoft e la documentazione in formato HTML è stata generata automaticamente tramite il tool Wyam.

2.5 Nuget

La dll generata dalla compilazione della libreria è stata infine documentata sotto il profilo delle dipendenze, generando il file .nuspec, ed inclusa nel file .nupkg tramite l'apposito tool fornito da Nuget. Il tutto è stato caricato sul portale online Nuget ed è ora possibile includerlo in un progetto tramite il comando cli:

```
$ dotnet add package SpazioDati.Dandelion
```

3 Calcolo della Relatedness

La seconda parte del progetto riguarda l'analisi di due metodi critici (`readDump()` e `rel(int a, int b)`) che servono a calcolare, dati gli identificativi di due entità semantiche, il loro valore di correlazione (relatedness).

Per fare ciò si ha a disposizione un dump dove sono salvati tutti i valori di correlazione (sopra una certa soglia minima) per ogni coppia di entità.

Il metodo `readDump()` serve a caricare in memoria i valori del dump sottoforma di "matrice" mentre il metodo `rel(int a, int b)` serve a fare una ricerca nella struttura dati per poi ritornare il valore di correlazione fra le entità con identificativi `a` e `b`.

Questa scelta implementativa, attualmente adottata, è sicuramente molto efficiente in termini di prestazioni ma sicuramente molto onerosa in termini di memoria; pertanto il fine dell'analisi sarebbe quello di studiare/testare implementazioni alternative per ottimizzare le prestazioni e/o diminuire lo spreco di memoria.

3.1 Implementazione Iniziale

3.1.1 Il Dump

Listing 3.1: Estratto del dump

```
53676192
4922289
0.01
2
1.0
53676158 7 null
53676164 5 NnAwQT/ZvTVJ05MeSfcdI0rMrg1LBmUH
53676016 13 RDsFRUkyASRLFf3T
53675811 16 null
...
```

Leggendo le prime righe di un dump si nota che le prime cinque righe sono valori di inizializzazione mentre dalla sesta riga in poi troviamo le righe della matrice. Il primo numero intero, denominato *max_id*, indica il limite massimo che gli id delle entità possono assumere; dato che gli id delle entità non sono necessariamente sequenziali (fra due di essi potrei avere dei "buchi", degli intervalli in cui gli identificativi non sono associati ad alcuna entità) esiste una funzione *map* che mappa ("compatta") gli id delle entità su altri id univoci e sequenziali.

Il secondo intero, denominato *nodesSize* è il valore massimo che la funzione *map* può assumere (limita il codominio della funzione).

$$\begin{aligned} &map() : \ \mathbb{N} \to \mathbb{N} \\ &[0, max\_id] \to [0, nodeSize], max\_id \geq nodeSize \end{aligned} \tag{3.1}$$

Continuando a leggere il dump troviamo la relatedness minima considerata *minRel* (sotto la quale i valori di correlazione non vengono salvati) e altri valori di configurazione quali *minIntersection* e *threshold* che tuttavia non ci interessano particolarmente.

La funzione map in un certo senso mappa "al contrario" gli id; infatti l'id massimo verrà mappato su 0, il penultimo id verrà mappato su 1 e così via. Nel caso del dump in questione abbiamo:

Listing 3.2: Esempio di funzione map e postingList

```
MaxId = 53676192
NodeSize = 4922289

   ID           MAP(ID)   POSTINGLIST
...
53676158   ->      7          null
53676159   ->      6          FW/I...
53676164   ->      5          NnAw...
53676174   ->      4          null
53676176   ->      3          null
53676177   ->      2          null
53676180   ->      1          null
53676190   ->      0          null
```

Gli id presenti nel dump non sono ordinati ma le *postingList* ad essi associate lo sono.

Dalla sesta riga del dump in poi inizia la definizione della matrice, ogni riga è composta da tre valori separati da uno spazio. Ad esempio se consideriamo la riga:

Listing 3.3: Riga del dump

```
53676164 5 NnAwQT/ZvTVJ05MeSfcdI0rMrg1LBmUH
```

Il primo valore $a = 53676164$ è l'id di un entità, il secondo valore $b = 5 = map(a)$ è il risultato della funzione map, infine il terzo valore $c = NnAwQT/ZvTVJ05MeSfcdI0rMrg1LBmUH$ è una stringa di dimensione variabile oppure *null*; quest'ultima stringa, denominata *postingList*, è la codifica in *Base64* di un array di byte che rappresenta un dizionario chiave valore.

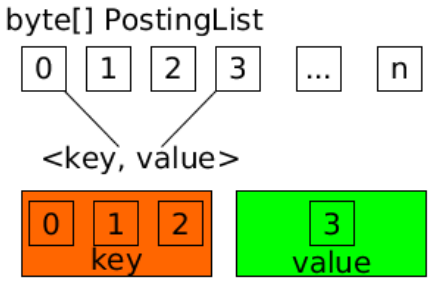


Figure 3.1: Encoding del dizionario nella postingList

In pratica una tupla $< chiave,\ valore >$ è codificata sull'array da una sequenza di quattro byte; quindi partizionando l'array in gruppi da 4 byte ottengo, per ogni gruppo, la chiave (è un id "mappato") codificata sui primi tre byte ed il valore (la relatedness) definito sul quarto byte (il byte più a destra).

Questa codifica, pertanto, permette di gestire set di entità con id "mappato" massimo $2^{3 \cdot 8} \simeq 1,7 \cdot 10^7$ ed una scala di valori di correlazione appartenenti all'intervallo $[0,\ 255]$.

Bisogna precisare che l'array di byte, denominato *postingList*, è ordinato per valore crescente delle chiavi e, dato che rel(a, b) = rel(b, a), si è deciso di salvare una sola volta il valore di correlazione imponendo la relazione: $a > b$.

Esempio

Nell'ipotesi in cui abbia una riga nel dump costituita dalla tupla $(a,\ b,\ c)$ e volessi trovare $relatedness(a, d)$ con $map(a) > map(d)$.

Devo cercare nella *postingList* c, un gruppo di quattro byte in cui i primi tre byte corrispondano a *map(d)* ed il quarto byte sarà il valore di correlazione voluto.

Ovviamente è possibile che c sia *null* (l'entità con id a non ha nessuna correlazione con entità i cui id "mappati" siano inferiori a b) oppure che nella posting non sia presente nessuna chiave corrispondente a *map(d)*, in questi casi $relatedness(a, d) = 0$.

3.1.2 Il Codice Nativo

Listing 3.4: Implentazione nativa

```
// STRUTTURA DATI
public class RelatednessMatrix {
    public int[] map;
    public byte[][] matrix;
    public float configuredMinRel;
    public int configuredMinIntersection;
    public float threshold;
    public static int EL_SIZE = 4;
}

// CODICE UTILIZZO
public float rel(int a, int b){
    if(a==b) return 1f;

    int nodeA = data.map[a];
    int nodeB = data.map[b];
    if (nodeA < 0 || nodeB < 0) return 0f;

    int key = a>b ? nodeB : nodeA;
    byte[] array = a>b ? data.matrix[nodeA] : data.matrix[nodeB];

    if (array == null) return 0f;

    int start = 0;
    int end = array.length - RelatednessMatrix.EL_SIZE;
    int pos = -1;
    while(pos == -1 && start <= end)
    {
        int idx = ((start+end)/RelatednessMatrix.EL_SIZE)/2;
        int idx_value = ((array[idx*RelatednessMatrix.EL_SIZE] & 0xFF) << 16)
                  + ((array[idx*RelatednessMatrix.EL_SIZE+1] & 0xFF) << 8)
                  + ( array[idx*RelatednessMatrix.EL_SIZE+2] & 0xFF);

        if(idx_value == key) pos = idx;
        else{
            if(key > idx_value)
                start = (idx + 1) * RelatednessMatrix.EL_SIZE;
            else
                end = (idx - 1) * RelatednessMatrix.EL_SIZE;
        }
    }

    if(pos == -1) return 0f;
    else
    {
        byte by = array[pos * RelatednessMatrix.EL_SIZE + 3];
        int byint = by + 128;
        float byteRel = byint / 255f;
        return data.configuredMinRel + byteRel * (1 - data.configuredMinRel);
    }
}

// CODICE LETTURA
public String END_OF_FILE = "" + '\0';

public RelatednessMatrix readDump() throws Exception {
    URL url = getClass().getResource("dump.txt");
```

```
58      File file = new File(url.getPath());
59      BufferedReader fbr = new BufferedReader(new FileReader(file));
60      String line = new String();
61
62      try {
63          int max_id = Integer.parseInt(fbr.readLine().trim()); // Trims all leading and
                trailing whitespace from this string
64          int nodesSize = Integer.parseInt(fbr.readLine().trim());
65          float minRel = Float.parseFloat(fbr.readLine().trim());
66          int minIntersection = Integer.parseInt(fbr.readLine().trim());
67          float threshold = Float.parseFloat(fbr.readLine().trim());
68
69          RelatednessMatrix data = new RelatednessMatrix();
70          data.map = new int[max_id + 1];
71          data.matrix = new byte[nodesSize][];
72          data.configuredMinRel = minRel;
73          data.configuredMinIntersection = minIntersection;
74          data.threshold = threshold;
75
76          int idx = 0;
77
78          while ((line = fbr.readLine()) != null) {
79              if (line.trim().equals(END_OF_FILE.trim())) {
80                  break;
81              }
82              System.out.println(line);
83              String[] splittedLine = line.split("\\s+"); // split(" ")
84              if (splittedLine.length != 3) {
85                  throw new Exception("Wrong format relatedness file for the line: " + line);
86              }
87
88              int wid = Integer.parseInt(splittedLine[0]);
89              int node = Integer.parseInt(splittedLine[1]);
90              data.map[wid] = node;
91
92              if (splittedLine[2].equals("null")) {
93                  data.matrix[node] = null;
94              } else {
95                  byte[] a = Base64.getDecoder().decode(splittedLine[2].toString());
96                  data.matrix[node] = Base64.getDecoder().decode(splittedLine[2].toString());
97              }
98              idx++;
99          }
100
101         if (idx != nodesSize) {
102             throw new Exception("Wrong format relatedness file the number of line do not
                    match with size of matrix");
103         }
104
105         return data;
106     } catch (Exception e) {
107         System.out.println(e.getMessage());
108         return null;
109     }
110 }
```

Il metodo `readDump()` legge riga per riga il dump e restituisce un'istanza di RelatednessMatrix valorizzata. La funzione map viene salvata in un array di interi (`data.map`) mentre la matrice con le correlazioni viene salvata in un array bidimensionale di byte (`data.matrix`).

Il metodo `rel(int a, int b)` invece calcola la funzione `map` sugli id `a` e `b` dopodichè fa una binary

search sulla postingList ottenuta da matrix[map(a)] (assumendo $map(a) > map(b)$). Infine se la ricerca binaria ha successo il valore di relatedness ottenuto va riscalato su una scala di valori $[0, 255]$, convertito in float e riscalato nuovamente in base al valore di relatedness minima considerata (per escludere i valori al di sotto della relatedness minima, che non verranno mai usati).

Listing 3.5: Conversione della relatedness da byte a float

```
byte by = array[pos * RelatednessMatrix.EL_SIZE + 3]; //considero il quarto byte
int byint = by + 128; //scalo di 128 valori per ottenere una relatedness in [0, 255]

//cast in float, escludendo i valori sotto la relatedness minima (configuredMinRel)
float byteRel = byint / 255f;
return data.configuredMinRel + byteRel * (1 - data.configuredMinRel);
```

3.1.3 Osservazioni

Considerato che il metodo readDump() viene chiamato una sola volta per allocare in memoria la struttura dati nel server, non ci sono particolari vincoli temporali per la sua esecuzione; per quanto riguarda il calcolo della relatedness invece l'algoritmo dev'essere il più performante possibile.

Il codice nativo per il calcolo della relatedness costa $O(lg_2(n))$ (con $n = nodeSize$) per la ricerca binaria e O(1) per quanto riguarda le istruzioni precedenti e successive a quest'ultima.

Tolta una piccola modifica al codice che portebbe ad evitare tre moltiplicazione per ciclo nella binary search (non è necessario dividere e poi moltiplicare per RelatednessMatrix.EL_SIZE), non sembra si possono migliorare di molto le prestazioni dell'algoritmo senza cambiare struttura dati.

Una via percorribile per non "pagare" $O(lg_2(n))$ per la binary search potrebbe essere quella di allocare direttamente in memoria la matrice completa (in questo caso avrei un algoritmo di tempo costante O(1)). Questa soluzione però andrebbe a peggiorare drasticamente lo spreco di memoria. Infatti nell'implementazione attuale la matrice ha dimensione massima $nodeSize \otimes nodeSize$ ma le righe hanno dimensione variabile e non occupano quindi sempre $(nodeSize - 1) \cdot 4\ Byte$.

Stima della Memoria Allocata

Se consideriamo le prime righe del dump in questione possiamo stimare che mediamente il 57% degli id hanno postingList vuota (null) ed il restante 43% ha un numero medio di relatedness per id pari allo $0,000101\%$ di nodeSize.

L'implementazione attuale alloca in memoria per l'array di interi map circa:

$$nodeSize \cdot 4\ Byte = 4^2 \cdot 10^6 Byte = 16MB \tag{3.2}$$

(arrotondando e senza tenere conto di fattori secondari come i 32 byte occupati da overhead e puntatore).

Per la matrice invece possiamo assumere che, su una macchina a 64 bit, avrò il 57% di righe a null ($0.57 \cdot 4 \cdot 10^6 \cdot 8\ Byte = 18\ MB$) ed il restante 43% di righe con mediamente 404 valori di relatedness codificati su 4 Byte ($0.43 \cdot 4 \cdot 10^6 \cdot 404 \cdot 4\ Byte = 2.8\ GB$).

Quindi in totale potremmo stimare che solo questo dump occupa quasi 3 GB (notare che esiste un dump per ogni lingua supportata da Dandelion e che questo è uno dei dump più piccoli).

Stima della Matrice Completa

Se allocassimo in memoria una matrice di Byte completa, di dimensioni $nodeSize \otimes nodeSize$, essa occuperebbe $(4 \cdot 10^6)^2 = 1.6 \cdot 10^{13}\ Byte = 16\ TB$. Pur considerando che esistono strutture dati particolarmente ottimizzate per gestire array di grandi dimensioni con pochi valori al loro interno (teniamo presente che più della metà dei valori nella matrice non saranno valorizzati), come SparseArray e SuperArrayList, bisogna tener presente che spesso tali strutture dati si basano su liste; in questo caso

però difficilmente si otterrebbero prestazioni migliori di quelle dell'implementazione attuale dato che il tempo di lookup sarebbe sempre $O(lg_2(noseSize))$.

In conclusione, a meno di uno spreco di memoria ulteriore, difficilmente otterremo buoni risultati mantenendo come struttura dati di riferimento la matrice.

Stima del problema inverso

Una via praticabile potrebbe essere quella di vedere il problema al contrario, considerando che esistono solo 256 valori possibili di relatedness, per minimizzare lo spreco di memoria potrei pensare di allocare 256 array in cui storicizzo tutte le coppie di id che hanno la stessa relatedness. In questo modo eliminerei la ridondanza costituita da valori uguali di relatedness salvati nel dump migliaia di volte.

Ci si accorge subito che anche questa via non è praticabile dato che per risparmiare 1 Byte devo storicizzare 8 Byte per le coppie di id di tipo intero. Andrei ad occupare $0.43 \cdot 4 \cdot 10^6 \cdot 404 \cdot 2 \cdot 4 = 5.6GB$, il doppio rispetto all'implementazione nativa, senza contare che le performance peggiorerebbero.

Stima implementazione di un grafo

Si potrebbe pensare di implementare un grafo orientato, salvando per ogni nodo l'id e la lista di archi uscenti e su ogni arco la relatedness fra nodo di partenza e nodo di arrivo. Dovrei quindi usare una struttura dati del tipo:

```
1    public class Node{
2        public int Id;
3        public List<Arrow> Arrows;
4    }
5
6    public class Arrow{
7        public Node Node;
8        public byte Relatedness;
9    }
```

Tuttavia in questo modo sprecherei memoria perchè il puntatore al nodo successivo (ipotizzando di lavorare su una macchina a 64 bit) peserebbe più dell'identificativo del nodo stesso. Al che potrei sostituire il puntatore con un intero ma anche in questo caso avrei il dizionario $< Id,\ Relatedness >$ che pesa 5 Byte mentre nella postingList pesa 4 Byte perchè chiave e valore sono accorpati. Infine se seguissi la stessa politica di codifica del dizionario della postingList di fatto sarei tornato all'implementazione iniziale senza trarre alcun giovamento, se non lo svantaggio agguintivo di non poter accedere in O(1) ad un nodo arbitrario.

3.2 Implementazione con Hash Map

Sembra che l'unica via praticabile per mantenere tutti i dati in memoria, diminuendo la memoria occupata e massimizzando le prestazioni sia una funzione di Hash Map, che mappi gli id concatenati sul valore della relatedness; se la funzione fosse ben ottimizzata otterrei in O(1) la relatedness (guadagno prestazionale) e potenzialmente potrebbe occupare meno memoria della matrice iniziale.

Purtroppo è molto difficile stimare a priori lo spazio occupato da un Hash Map, il modo più rapido è confrontare con dei benchmark l'implementazione nativa contro quella basata su Hash Map.

3.2.1 Codice

Listing 3.6: Implentazione con HashMap

```
1  public float Relatedness(int a, int b) throws Exception {
2      if (a == b) {
3          return 1f;
4      }
5
6      int nodeA = data.map[a];
```

```
    int nodeB = data.map[b];

    if (nodeA < 0 || nodeB < 0) {
        return 0f;
    }

    String key = (nodeA > nodeB) ? (nodeA + "." + nodeB) : (nodeB + "." + nodeA);
    try {
        byte value = (byte) data.hm.get(key);

        if (value == 0) {
            return 0f;
        } else {
            int byint = value + 128;
            float byteRel = byint / 255f;
            return data.configuredMinRel + byteRel * (1 - data.configuredMinRel);
        }
    } catch (Exception e) {
        return 0f;
    }
}

public RelatednessHashMap ReadDump(String dump) throws Exception {
    URL url = getClass().getResource(dump);
    File file = new File(url.getPath());
    BufferedReader fbr = new BufferedReader(new FileReader(file));
    String line = new String();

    try {
        RelatednessHashMap data = new RelatednessHashMap();
        int max_id = Integer.parseInt(fbr.readLine().trim());
        int nodesSize = Integer.parseInt(fbr.readLine().trim());
        data.configuredMinRel = Float.parseFloat(fbr.readLine().trim());
        data.configuredMinIntersection = Integer.parseInt(fbr.readLine().trim());
        data.threshold = Float.parseFloat(fbr.readLine().trim());

        data.map = new int[max_id + 1];
        data.hm = new HashMap();

        String END_OF_FILE = "" + '\0';

        while ((line = fbr.readLine()) != null) {
            if (line.trim().equals(END_OF_FILE.trim())) {
                break;
            }

            String[] splittedLine = line.split("\\s+");
            if (splittedLine.length != 3) {
                throw new Exception("Wrong format relatedness file for the line: " + line);
            }

            int wid = Integer.parseInt(splittedLine[0]);
            int node = Integer.parseInt(splittedLine[1]);
            data.map[wid] = node;

            if (!splittedLine[2].equals("null")) {
                byte[] postingList = Base64.getDecoder().decode(splittedLine[2].toString());

                int idx_value = ((postingList[0] & 0xFF) << 16)
                          + ((postingList[1] & 0xFF) << 8)
                          + ((postingList[2] & 0xFF) << 0);
```

```
68
69                  String key = idx_value + "." + node;
70                  data.hm.put(key, postingList[3]);
71              }
72          }
73
74          return data;
75      } catch (Exception e) {
76          System.out.println(e.getMessage());
77          return null;
78      }
79  }
```

[**TODO** implementazione alternativa fastutils e metrics di Prometheus e Grafana]

4 StrepHit

La terza parte del progetto consiste nell'implementazione di uno script in Python (versione 2.7) che, dato in input un dataset di QuickStatements, deve restituire in output un nuovo dataset di QuickStatements arricchito di nuovi riferimenti.

Ogni riga del dataset di input presenta una proprietà reference URL (P854) con l'URL di riferimento della risorsa, lo script deve eseguire una query SPARQL per trovare l'item nel knowledge base che corrisponde al dominio della URL e la proprietà di tale item che definisce lo schema della URL in questione.

4.0.2 Esempio di funzionamento

Per esempio dato il seguente quickstatement in input:

Listing 4.1: Riga del dump

```
1    Q193660 P106 Q207628 S854 "http://www.nndb.com/people/031/000097737/"
```

Notiamo una prima relazione semantica Q193660 (Ramon Llull) P106 (occupation) Q207628 (musical composition) che ci dice semplicemente che "*Ramon Llull lavora come compositore musicale*".

Segue la proprietà che interessa maggiormente S854 "http://www.nndb.com/people/031/000097737/" che indica la provenienza dell'informazione.

Lo script procede estrapolando il dominio dalla reference url (www.nndb.com) e con una query sparql trova l'item di riferimento a quel dominio in Wikidata, sempre che esista.

In questo caso l'item di riferimento è Q1373513 che fra le varie proprietà presenta anche una Wikidata property (P1687) chiamata "*NNDB people ID*" (P1263).

Se ora analizziamo la proprietà "*NNDB people ID*" (P1263) notiamo che presenta una proprietà formatter URL (P1630) il cui valore è http://www.nndb.com/people/$1/. Il valore finale $1 nella formatter URL è un placeholder che sta ad indicare la parte della URL che corrisponde al valore della proprietà prescelta.

Lo script andrà quindi ad estrapolare dalla URL di partenza il valore 031/000097737 che corrisponde al valore della proprietà P1263 e andrà ad arricchire il dump con questa informazione aggiuntiva.

Listing 4.2: Risultato dello script

```
1    Q193660 P106 Q207628 S854 "http://www.nndb.com/people/031/000097737/" S248 Q1373513
         S1263 "031/000097737" S813 2018-06-04T02:19:10Z/14
```

I riferimenti aggiuntivi sono: S248 (stated in) Q1373513 (NNDB) S1263 (NNDB people ID) "031/000097737" (il valore estrapolato, people ID) S813 (retrieved) 2018-06-04T02:19:10Z/14 (timestamp).

4.1 SPARQL Query

Per risolvere ogni dominio e ogni formatter URL sconosciuta si usa una sola query; si è cercato di limitare il più possibile il numero di query effettuate (dato che alcune query possono impiegare svariati secondi ad essere eseguite), salvando in memoria e su disco i risultati di quelle già lanciate in precedenza per minimizzare il tempo di computazione dello script.

Listing 4.3: SPARQL query per cercare item e proprietà relativi al dominio "www.nndb.com"

```
1    select Distinct ?subjects ?wikidataProperty ?formatterUrlLabel ?sitelinkLabel
2    where {
```

```
        {
            BIND("www.nndb.com" AS ?domain).
            SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en".
                }
            ?subjects wdt:P856 ?sitelink ;
                      wdt:P1687 ?wikidataProperty.
            ?wikidataProperty wdt:P1630 ?formatterUrl
            FILTER (REGEX(str(?formatterUrl), ?domain) || REGEX(str(?sitelink), ?domain)).
        }
        union
        {
            BIND("www.nndb.com" AS ?domain).
            SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en".
                }
            ?subjects wdt:P856 ?sitelink ;
            FILTER REGEX(str(?sitelink), ?domain).
        }
    }
```

Va premesso che esistono molti modi di ottenere lo stesso risultato, con costrutti molto meno verbosi, tuttavia questa è l'unica query che attualmente non manda in timeout l'endpoint.

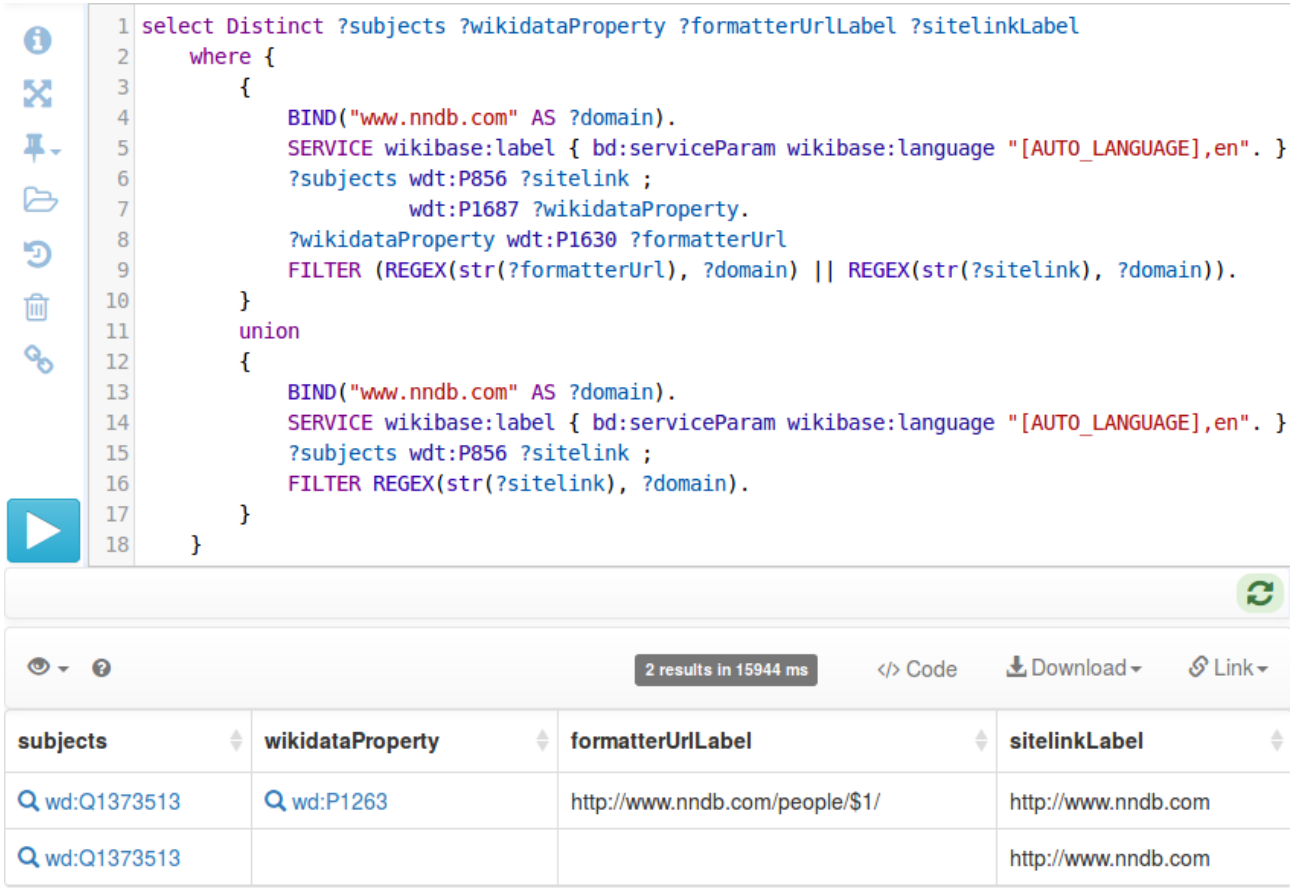
Il superamento del timeout è sicuramente dovuto al fatto che internamente si usano delle regular expressions che appesantiscono molto l'esecuzione della query, soprattutto su grandi knowledge base come quello di Wikidata.

In SPARQL usare una ricerca per stringa è certamente una forzatura perchè solitamente si conoscono già a priori item e propery che interessano tuttavia in questo caso è stato necessario adottare la ricerca per regular expression dato che lo script deve proprio affrontare il problema inverso.

La query è il risultato dell'unione di due sub-query; la prima sub-query cerca tutte le proprietà che posseggono una proprietà formatter URL (P1630) il cui valore ha come dominio il dominio del quickstatement che lo script sta analizzando (in questo caso "www.nndb.com"); in oltre controlla che tale priprietà sia legata ad un item il cui official website (P856) sia coerente con il dominio in questione.

La seconda sub-query invece cerca solo tutti gli item il cui official website (P856) abbia lo stesso dominio di quello del quickstatement che lo script sta analizzando. Questa query è necessaria perchè non sempre esiste una proprietà la cui formatter URL matcha correttamente con il link che si sta analizzando; può succedere che esista solo l'item relativo al database in questione ma non la proprietà specifica e in pochi casi non esiste nemmeno tale item. In alcuni casi invece può succedere che l'item relativo al database esista ma abbia un dominio completamente differente da quello della proprietà la cui formatter URL presenta un match con la URL in analisi.

La difficoltà principale nella realizzazione dello script è stata proprio quella di gestire una moltitudine di casi particolari derivanti dal fatto che il dataset era molto grande (più di 500.000 quickstatements) con una svariati domini differenti e qualche URL completamente sbagliata o deprecata.



```
select Distinct ?subjects ?wikidataProperty ?formatterUrlLabel ?sitelinkLabel
    where {
        {
            BIND("www.nndb.com" AS ?domain).
            SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en". }
            ?subjects wdt:P856 ?sitelink ;
                      wdt:P1687 ?wikidataProperty.
            ?wikidataProperty wdt:P1630 ?formatterUrl
            FILTER (REGEX(str(?formatterUrl), ?domain) || REGEX(str(?sitelink), ?domain)).
        }
        union
        {
            BIND("www.nndb.com" AS ?domain).
            SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en". }
            ?subjects wdt:P856 ?sitelink ;
            FILTER REGEX(str(?sitelink), ?domain).
        }
    }
```

2 results in 15944 ms Code Download Link

subjects	wikidataProperty	formatterUrlLabel	sitelinkLabel
wd:Q1373513	wd:P1263	http://www.nndb.com/people/$1/	http://www.nndb.com
wd:Q1373513			http://www.nndb.com

Figure 4.1: Risultato della query per il dominio "www.nndb.com"

4.2 Lo script

4.2.1 Struttura del progetto

Il progetto presenta una cartella assets contenente tutti i file di input e output, i json di configurazione e i log degli errori; nella cartella business sono contenuti i servizi, i metodi di utilità e le query.

La cartella domain contiene i modelli e le localizations (nel file localizations.py sono definite anche una serie di costanti da settare a seconda delle esigenze per configurare lo script).

In tests abbiamo tutte le classi di test basate sulla libreria unittest, come per il Client *C#* anche in questo caso è stato configurato Travis per far eseguire i test automaticamente ad ogni pull-request.

Infine nella root del progetto abbiamo l'entry point dello script (main.py) e l'entry point dei test (test.py), i requirement per il package manager e il file di configurazione strephit.py nel caso si voglia lanciare lo script in un virtualenv tramite la libreria click.

4.2.2 Algoritmo

Istanziando l'oggetto QuickStatementsService vengono caricati automaticamente in memoria i mappings presenti nella cartella assets; i mappings sono dei file json di salvataggio in cui lo script salva i risultati delle le chiamate, all'end-point SPARQL, già effettuate.

Listing 4.4: Some Code

```
    "www.nndb.com": [
        {
            "db_id": "Q1373513",
            "db_property": "S1263",
            "to_upper_case": false,
            "url_pattern": "http://www.nndb.com/people/$1/"
        }
    ],
```

Il metodo add_db_references_async() cicla su ogni riga del dataset e per ogniuna di esse chiama un handler (add_db_references_async_handler()), passandogli un oggetto QuickStatement contenente tutte le informazioni della riga, che procede con l'analisi della URL e la generazione dei riferimenti mancanti.

A questo punto, in modo sincrono o asincrono, a seconda del settaggio delle costanti in localizations.py (IS_ASYNC _MODE = True/False), viene chiamato il metodo generate_db_reference() che provvede a controllare che nei mappings ci sia già una entry con formatter URL compatibile con la URL del quickstatement e a generare i riferimenti mancanti.

Se nei mappings non esiste nessuna entry compatibile con la URL allora viene chiamato il metdo new_mapping() che provvede a chiamare l'end-point SPARQL e a selezionare il risultato migliore per aggiungerlo ai mappings.

Se la costante MAP_ALL_RESPONSES viene settata a True, ogni risposta del endpoint SPARQL viene salvata interamente nei mappings, questo previene qualsiasi altra chiamata all'endpoint per un determinato dominio ma accresce la dimensione dei mappings a dismisura e rende di conseguenza la ricerca più lenta.

4.2.3 Refresh delle URL

Nel dataset analizzato alcuni domini erano deprecati tant'è che tentando di accedervi da un browser si veniva reindirizzati ad altri domini oppure era avvenuto il passaggio da http ad https.

Per non perdere questi riferimenti si è deciso di implementare anche una funzionalità che data una lista di domini va a refreshare tutte le URL aventi tali domini e ad eliminare le righe contenenti URL inesistenti.

A fine procedura vengono salvati i log con la lista di url modificate e con la lista di righe eliminate dal dataset.

Al termine dell'esperienza si è concluso che ... [**TODO**]