UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

CORSO DI LAUREA IN INFORMATICA

**Strumenti per l’esplorazione efficiente della letteratura scientifica**

Laureando:

Paolo Crotti

Relatore:

Dott. Mauro Andreolini

Anno Accademico 2018/2019

Sommario

[1 Introduzione 2](#_Toc3975508)

[2 Strumenti e risorse del progetto 3](#_Toc3975509)

[2.1 Google Scholar 3](#_Toc3975514)

[2.2 Microsoft Academic 4](#_Toc3975515)

[2.3 Neo4j 6](#_Toc3975516)

[2.3.1 Database basati su grafi 7](#_Toc3975517)

[2.3.2 Cypher 8](#_Toc3975518)

[2.3.3 Driver Golang 10](#_Toc3975519)

[2.4 Selenium 10](#_Toc3975520)

[2.4.1 WebDriver 11](#_Toc3975521)

[2.4.2 Geckodriver 11](#_Toc3975522)

[2.4.3 Driver Golang 12](#_Toc3975523)

[3 Funzionalità e implementazione del prototipo 12](#_Toc3975524)

[1 Bibliografia 14](#_Toc3975525)

# 1 Introduzione

Questo lavoro di tesi si concentra su un aspetto legato alla scrittura degli articoli scientifici: la ricerca dei “related works”, cioè di tutte quelle pubblicazioni correlate a ciò su cui si sta lavorando. Lo scopo è quello di realizzare un programma che aiuti l’autore a informarsi su un dato argomento, permettendogli di individuare in modo automatico gli articoli, i topic e gli autori correlati.

La scrittura di un articolo ha delle difficoltà: è necessario definire l’obbiettivo e le tappe per raggiungerlo, suddividere il corpo in paragrafi chiari e intuitivi, fornire una moltitudine di esempi e similitudini così da favorire la comprensione da parte del lettore, ma soprattutto richiede una visione completa dell’argomento di cui si sta parlando [1]. La ricerca scientifica viene svolta su un argomento ben preciso attorno al quale si possono individuare delle problematiche ben chiare, inoltre la ricerca deve poter aggiungere novità allo stato dell’arte oppure rivedere in una diversa ottica argomenti già affrontati. Il suo scopo è quello di essere utile agli altri aggiungendo qualcosa di nuovo e tutti gli altri lavori fatti sullo stesso argomento dovranno tenerne conto. La ricerca deve inoltre fornire elementi che permettano di metterla in discussione (verifica delle conclusioni raggiunte) e consentire ad altri di continuarne il lavoro.

Per raggiungere tutti questi obbiettivi è necessario scrivere articoli di qualità ed è per cui fondamentale informarsi approfonditamente sull’argomento, prima di iniziare il lavoro. La prima legge della comunicazione di Wittington dice che “quando qualcuno spiega un argomento che non ha ben capito, sarà compreso solo da chi ne sa più di lui” [2]. Quindi se si scrive un articolo senza aver compreso appieno l’argomento di cui si sta parlando, è probabile che il lettore non riuscirà a capirlo.

Quindi avere una profonda conoscenza dell’argomento aiuta a:

1. Non commettere errori che comprometterebbero l’immagine dell’autore e dell’editore.
2. Scegliere gli aspetti più interessanti da trattare, individuati grazie all’esperienza personale maturata col tempo.
3. In generale a scrivere meglio: usare gli esempi e le parole più efficaci e creare un filo logico più chiaro.

Il problema principale nella scrittura degli articoli è la scrittura dello stato dell’arte definito come “il punto cui sono arrivate le ricerche in una determinata disciplina” [3] perché per avere una visione completa dell’argomento sarebbe necessario che l’autore fosse a conoscenza di qualsiasi pubblicazione mai uscita al riguardo. La soluzione proposta consiste nel cercare sui principali motori di ricerca per le pubblicazioni, quindi Google Scholar e Microsoft Academic gli articoli correlati e costruire un grafo di ricerca. In questo grafo, i nodi sono gli articoli mentre gli archi rappresentano le citazioni. L’idea è quella di partire da una o più pubblicazioni, raccoglierne gli articoli correlati e costruire dei grafi da cui estrarre informazioni come gli articoli e gli autori più importanti o i topic più diffusi.

Per capire come sono correlati tra loro i topic all’interno dei related work creo un secondo grafo in cui mostro i topic (nodi) collegati tra loro in modo da evidenziare, non solo quelli che compaiono più frequentemente (per quello c’è una classifica) ma soprattutto la gerarchia che esiste tra loro, per esempio il topic “rootkit” compare solo in articoli che ne citano altri di “computer security”, questo mi dice che all’interno di questo ambito di ricerca “rootkit” è (secondario) rispetto a “computer security” Questo tipo di grafo lo posso costruire solo quando la ricerca viene svolta su “Microsoft Academic” perché qui sono presenti i topic che sono invece assenti su Google Scholar.

# 2 Strumenti e risorse del progetto



## Google Scholar

Google Scholarè un motore di ricerca accessibile liberamente che tramite parole chiave specifiche consente di individuare vari tipi di pubblicazioni come articoli scientifici, tesi di laurea e libri. Consente inoltre di reperire articoli da una vasta gamma di case editrici che si rivolgono al mondo dello studio e della ricerca da associazioni scientifiche e professionali e università, oltre che nella galassia di articoli scientifici e culturali distribuiti nel Web.

Uno dei suoi pregi è quello di fornire, tramite una ricerca Web, l’accesso gratuito a una versione full text dell’articolo senza il permesso del giornale che lo ha pubblicato. Questo perché cerca copie degli articoli indicizzati anche su siti considerati non sicuri. Scholar non fornisce una lista dei periodici da cui prende gli articoli perché si limita a fare una lista di tutte le pubblicazioni dalla sua ricerca per il Web. Oltre alla ricerca classica in cui cerca all’interno del proprio indice le parole inserite dall’utente, permette anche una Ricerca Avanzata. Qui è possibile specificare le parole che devono comparire nel titolo e/o nel corpo, gli autori, gli editori e la data di pubblicazione. [4] Tra le informazioni dei risultati compare il numero degli articoli che hanno citato il risultato, questi articoli possono essere recuperati dal link pertinente. Altro link rilevante è quello che porta agli articoli correlati cioè quelli Scholar ritiene più “simili”.

L’algoritmo che Scholar usa per il ranking non è pubblico ma se ne conosco i parametri: il numero di citazioni è quello tenuto maggiormente in considerazione; infatti articoli citati spesso compaiono con molta più frequenza tra i risultati delle ricerche. Di conseguenza Scholar sembra essere più indicato per la ricerca di articoli che seguono la corrente principale piuttosto che quelli che propongono una visione alternativa. [5] Tiene inoltre conto del testo dell’articolo, dell’autore e della rivista.

Non tutti gli articoli mostrati tra i risultati sono accessibili in quanto appartengono a riviste commerciali, in questi casi le uniche informazioni disponibili sono parte dell’abstract e le citazioni. Per accedere all’articolo è necessario pagare. Altra pecca di Scholar è che non tutti i risultati sono articoli accademici, include anche: annunci di notizie, presentazioni in PowerPoint e materiale non pubblicato. Sarebbe quindi necessario filtrare i risultati per discernere le pubblicazioni dal resto.

Scholar attribuisce grande importanza al numero di citazioni quando calcola il ranking e per questo è stato accusato di rafforzare l’effetto Mathew: le pubblicazioni più citate compaiono nelle prime posizioni, mentre quelli più recenti difficilmente appaiono in cima; di conseguenza ricevono meno attenzione da parte degli utenti e quindi meno citazioni.

Google Scholar è particolarmente vulnerabile allo spam. Ricercatori dell’università della California, Berkley e Otto-von-Guericke hanno dimostrato che i conteggi delle citazioni sul motore di ricerca possono essere manipolati ed è possibile creare articoli privi di significato ma che vengono comunque indicizzati. È quindi possibile creare gruppi di articoli falsi che si citano a vicenda e contenenti determinate parole chiave per invalidare il contenuto dell’indice di Scholar. [6]

## 2.2 Microsoft Academic

Microsoft Academic è un motore di ricerca gratuito per articoli scientifici e letteratura accademica, sviluppato dalla Microsoft Research. La raccolta dei documenti avviene tramite strumenti che sfruttano l’intelligenza artificiale per processare i documenti scoperti da Bing (crawling), riconoscere quelli accademici e aggiungerli alla sua base di conoscenza. Le informazioni che riesce a ricavare dalle pubblicazioni sono molteplici: gli URL dei sorgenti differenziandoli tra risorse presenti come pagine Web e scaricabili come PDF, l’abstract, la data, gli autori, il giornale e le citazioni.

È possibile effettuare diverse operazioni sui risultati della ricerca: ordinarli dal più recente, per rilevanza e per numero di citazioni. Academic fornisce ulteriori informazioni oltre ai risultati in se: sulla sinistra mostra autori, affiliazioni, riviste e altre entità presenti nel suo database correlate agli argomenti della ricerca; mentre sulla destra mostra eventuali informazioni sull’autore (nel caso una delle parole facesse match con un autore).

Queste informazioni sono conservate nel Microsoft Academic Graph (MAG), un grafo contenente più di 210 milioni di pubblicazioni e 256 milioni di autori. L’essere costruito in modo automatico comporta degli svantaggi come perdersi delle citazioni (solo 60 milioni di articoli hanno delle citazioni) e la mancanza di disponibilità delle informazioni delle affiliazioni. D’altra parte, vanta una vasta base di pubblicazioni ricca di metadati sugli articoli (al contrario di Google Scholar) che copre bene diversi campi di ricerca, in particolare per le discipline scientifiche. Anche la disambiguazione delle affiliazioni e degli autori ha un buon livello di correttezza.

Al contrario di Google Scholar, Academic memorizza anche i campi di studio (o i topic) di un articolo. Stando alla definizione data da Microsoft:” Topics are organized in a non-mutually exclusive hierarchy with 19 top-level fields of study” [7]. In pratica i campi sono organizzati in una gerarchia divisa su 4 livelli: in cima risiedono pochi (19) campi più generici mentre scendendo nei livelli inferiori si ha via via una maggiore granularità. I topic più diffusi (ordinati per numero di occorrenze) sono: medicina, biologia, scienza dei materiali, chimica, ingegneria, computer science. È importante notare che il dataset viene costantemente aggiornato e con esso la classifica, anche se sono sempre gli argomenti a carattere scientifico a dominare le prime posizioni. I nuovi topic vengono scoperti da sistemi che sfruttano l’intelligenza artificiale per effettuare una comprensione semantica del contenuto delle pubblicazioni; questa analisi ha anche lo scopo di aggiornare la gerarchia in modo corretto, disponendo i topic secondo un ordine logico di specificità (es. biologia freccia botanica freccia coltivazione)

Nel Microsoft Academic Graph sono presenti più di 1,9 miliardi di coppie di citazioni e in media ogni entità (articoli, libri, conferenze, …) viene citata 9,33 volte. Tuttavia, una parte significativa del grafo è disconnessa dal resto: sono presenti molti articoli che non citano, né vengono citati; è infatti comune trovare articoli privi di citazioni anche quando si cercano argomenti molto popolari. Con i vari aggiornamenti del grafo, il numero delle citazioni è aumentato notevolmente; più del numero delle pubblicazioni. Se da un lato questo suggerisce un costante miglioramento dei contenuti, può anche portare alla raccolta di dati non affidabili. [8]

Una recente feature offerta da Microsoft Academic permette di ottenere diverse informazioni statistiche e classifiche. Nella sezione “preview” del sito, è disponibile una pagina dedicata alle pubblicazioni in cui è possibile specificare un campo di studio e ricevere delle informazioni al riguardo. Tra queste ci sono un istogramma con il numero di pubblicazioni divise per anno e diverse top 10: migliori autori, riviste, istituti e conferenze in cui è stato trattato quell’argomento. Purtroppo, non è possibile specificare direttamente il topic ma, partendo da quelli top-level, è necessario cliccare su quello che sembra più legato a quello interessato. In questo modo ne vengono proposti nuovi di più specifici o correlati a quello precedente. Da un lato questo tipo di ricerca aiuta l’utente inesperto o che non sa di preciso cosa cercare (o ne ha solo una vaga idea), dall’altro permette di scoprire nuovi argomenti legati a quello che si sta cercando. [9]



<https://preview.academic.microsoft.com/publications/>

## 2.3 Neo4j

Neo4j è un database managment system basato sui grafi sviluppato da Neo4 Technology e stando alla classifica stilata da DB-Engines è il database su grafi più popolare al mondo. [10]

Il software è completamente scritto in java ed è disponibile open-source la versione “community” tramite licenza GPL3; fornisce backup online e una vasta gamma di estensioni, sia gratuite che commerciali. La prima release è avvenuta nel 2010, mentre la stable attuale (3.0) è stata rilasciata nel 2016.

Una volta lanciato il programma, è possibile accedere alle funzionalità del dbms tramite un qualsiasi browser web collegandosi a localhost sulla porta del protocollo scelto: http (7474), https (7473) oppure bolt (7687). Tramite l’interfaccia è possibile esplorare il database tramite query scritte in Cypher, un linguaggio che come lo Sparql, è stato pensato per database non relazionali.

Nonostante la semplicità della sua interfaccia, si rivela intuitivo funzionale. La entry in cui è possibile scrivere la query è provvista di completamento automatico, inoltre mostra dei warning quando si cerca di eseguire query troppo onerose dal punto di vista computazionale; per esempio quando, nella selezione dei parametri (nodi e relazioni) su cui si andrà a lavorare, si effettua un prodotto vettoriale, ritrovandosi con una lista molto lunga di parametri da controllare. I risultati vengono mostrati in diversi formati:

1. Testuale: esattamente come in sql i risultati sono inseriti in una tabella che ha come nome delle colonne, quelli delle variabili in output definite nella query; mentre i risultati sono le entry.
2. Json: questo formato è particolarmente utile quando si vogliono esportare i risultati per passarli ad un’altra applicazione in modo che gli elabori.
3. GUI: il formato più intuitivo e utile per farsi un’idea di come sono distribuiti i nodi, quali relazioni li legano, quali hanno un grado maggiore, e così via; un limite di questa visualizzazione è il numero massimo dei nodi visualizzabili che è molto limitato (circa 300), se si imposta una soglia superiore sarà necessaria una quantità non indifferente di memoria RAM per evitare cali di prestazioni. Una caratteristica utile è la possibilità di colorare i nodi e le relazioni in base alle label assegnate; questo permette di evidenziare percorsi e entità.



Qui dovrei inserire un’immagine della main di neo4j sul browser.

Consente anche di definire degli schemi per le entità e le relazioni che permettono poi di creare indici e vincoli che migliorano le prestazioni durante la ricerca. I principali modi per esportare i risultati sono in formato json, se si vogliono passare come input per un’analisi più approfondita e png, se è necessaria solo l’immagine del grafo in output. Esiste inoltre un plugin che permette di esportare nel formato GraphML che può essere letto da applicazioni come Gephi e yEd. [11]

Le unità fondamentali di questo modello sono:

* Nodi: rappresentano le entità e possono possedere delle proprietà, può anche essere assegnata loro un’etichetta (label).
* Relazioni: servono per indicare un qualche tipo di connessione tra due nodi, anche loro possono avere proprietà e etichette. È possibile assegnarle una direzione, se si ha bisogno di un grafo diretto; ma è importante notare che è possibile avere due relazioni dello stesso tipo (stessa label) una diretta, l’altra no. Una relazione può essere attraversata in entrambi i versi, è quindi possibile specificare prima la destinazione e poi la sorgente o anche solo una delle due.
* Proprietà: sono coppie chiave valore dove la chiave è una stringa (il nome della proprietà), mentre il valore può essere di diversi tipi come string e int. Sono supportati anche gli array di tipi primitivi.
* Label: serve per raggruppare i nodi/relazioni: gli elementi con la stessa label appartengono allo stesso sottoinsieme; questo rende le interrogazioni più facili da scrivere. Le label possono essere modificate a runtime, questo permette di segnare temporaneamente alcuni elementi, magari per indicarne lo stato.
* Path: il percorso indica il sotto grafo che separa il nodo sorgente da quella destinazione e la sua lunghezza è data dal numero di salti necessari per raggiungere la destinazione. Il percorso più corto possibile è costituito da un singolo nodo privo di relazioni e ha lunghezza 0.

### 2.3.1 Database basati su grafi

Questo tipo di database utilizza gli elementi tipici della teoria dei grafi per memorizzare le informazioni: nodi, proprietà e relazioni. Quelli che in database relazionale sarebbero i record delle tabelle, qui vengono rappresentati come entità con proprietà (campi); le relazioni tra tabelle diventano relazioni tra singole entità. La qualità più apprezzata di questo modello è l’intuitività con cui si riescono ad individuare il rapporto tra i dati, particolarmente utile tra dati fortemente interconnessi. Alcuni dei linguaggi maggiormente utilizzati sono Sparql e Cypher.

Esistono principalmente due modelli per conservare i dati:

* Labeled: ai nodi e alle relazioni vengono assegnate delle etichette così da poterli raggruppare con maggiore semplicità; è il sistema più elementare ed è quello adottato da Neo4j.
* Resource Description Framework (RDF): i dati vengono conservati sotto forma di triplette:

nodo (soggetto) – relazione (predicato) – nodo (oggetto)

anche qui nodi e relazioni possono avere delle proprietà. I singoli nodi possono essere identificati da stringe o URI (stringhe che identificano in modo non ambiguo una risorsa).

Sono particolarmente utili quando si ha necessità di eseguire query graph-like, come la ricerca del percorso minimo tra due nodi. Inoltre, non richiedono un grande lavoro a livello di progettazione del database, questo rende più semplice l’aggiunta di nuovi dati senza paura di perdere funzionalità.

Come nei database relazionali, per mantenere la consistenza dei dati si utilizza il modello ACID; tuttavia la sua implementazione è molto diversa. ACID garantisce al database le seguenti caratteristiche:

* Atomic: tutte le operazioni in una transazione devono avere successo o il sistema torna allo stato precedente all’esecuzione della prima operazione.
* Consistent: al termine di ogni transazione, il database deve trovarsi in uno stato coerente, quindi non devono esserci contraddizioni tra i dati archiviati (inconsistenza).
* Isolated: ogni transazione deve essere indipendente dalle altre, così il fallimento di una transazione non influisce sulle altre.
* Durable: i cambiamenti apportati da una transazione avvenuta con successo non potranno andare persi nel tempo (persistenza).

Queste proprietà garantiscono che i dati prodotti da una transazione completata con successo siano consistenti e conservati in modo permanente su disco.

### 2.3.2 Cypher

Cypher è un linguaggio dichiarativo creato per interrogare database basati su grafo. È il linguaggio nativo di neo4j e punta sull’espressività e sull’efficienza nell’esecuzione delle query. Quello che l’utente fa è chiedere al database di trovargli “tutti gli oggetti simili o che gli assomigliano” a un dato pattern.

La sintassi di Cypher permette di descrivere con caratteri ASCII il pattern del sotto grafo che ci interessa esaminare, per esempio:

(a)-[:KNOWS]->(b)-[:KNOWS]->(c) , (a)-[:KNOWS]->(c)

Questa sintassi permette di individuare una struttura di questo tipo:



In Cypher è anche possibile dei punti da cui iniziare la ricerca, invece che controllare tutti i nodi del grafo per controllare se corrispondono al pattern. Questi nodi sono chiamati punti di ancoraggio e possono essere definiti tramite la clausola START. Tuttavia, dalla versione 3.2 questa possibilità è stata rimossa; al suo posto è possibile sfruttare gli indici per migliorare le normali prestazioni che si avrebbero con MATCH. Questo è possibile grazie alle ottimizzazioni della nuova versione del compilatore che permette 3 ambienti di runtime (interpreted, slotted e compiled) che sfruttano una sempre migliore gestione delle singole operazioni al fine di migliorare le performance e l’utilizzo della memoria. La modalità più performante (compiled) permette di raggruppare in modo intelligente le operazioni così da creare un nuovo piano di esecuzione che migliora notevolmente le prestazioni e il consumo di risorse; tuttavia è ancora in fase di sviluppo e non supporta tutti gli operatori e le query. Da notare che gli ultimi due ambienti sono disponibili solo per la versione Enterprise.

Come in sql, le query sono composte da clausole, alcune delle principali sono:

* MATCH serve per specificare uno o più pattern all’interno del grafo, permette di definire su quali dati lavorare.
* WHERE aggiunge dei vincoli per filtrare i risultati, ad esempio specificando il valore di certe proprietà.
* RETURN cosa restituire.

Esistono anche altre clausole che permetto di creare nuovi nodi e relazioni (CREATE) e di aggiungerli se non sono già presenti (MERGE). Sono anche presenti le clausole classiche del sql come LIMIT per limitare il numero dei risultati restituiti e ORDER BY che permette di ordinare i risultati secondo una qualche proprietà.

Le label sono anche utilizzate per la creazione di indici, che velocizzano di molto la ricerca, e la definizione dei vincoli. È importante notare che, in assenza di vincoli, le singole entità non sono identificate dal valore di una loro proprietà, come poteva essere per i database relazionali. Tipicamente esiste un identificatore comune a tutte le entità e relazioni (indipendente dalla label) che viene incrementato a ogni creazione; è infatti possibile creare molteplici entità con gli stessi valori.

In questo linguaggio “null” è usato per rappresentare un valore mancate o non definito.

### 2.3.3 Driver Golang

Questo driver è disponibile si Github ed è stato sviluppato da “johnnadratowski”. Questo driver, scritto in Go, realizza un’implementazione del protocollo Bolt a basso livello; questo permette di inviare comandi a Neo4j e ricevere i risultati in risposta. Tra le caratteristiche principali si ha il pipelining dei messaggi, per migliorare la concorrenza e la possibilità di creare una pool di connessioni, così da svolgere compiti diversi contemporaneamente.

Di default le procedure si aspettano che le query in input siano scritte in Cypher, ma supporta anche le funzionalità richieste dall’interfaccia di Go “database/sql/driver”. Tuttavia, è caldamente consigliato l’utilizzo di Cypher in quanto fornisce maggiori funzionalità e preastazioni. [12]

Recentemente l’autore ha comunicato che non manterrà più il codice; l’ultima versione disponibile è la 3.1.0-M02. [13]

## 2.4 Selenium

Selenium è uno dei software per l’automazione di browser più diffusi al mondo. La sua principale funzione è svolgere dei “test”, permette quindi di inviare al browser una serie di operazioni da fargli eseguire in modo automatico. Questo si rivela molto utile quando si vogliono testare le funzionalità del proprio sito web in fase di sviluppo: la reazione a diversi tipi di input, misurare i tempi di risposta e così via. [14]

Il software è disponibile per tutte le maggiori piattaforme: Winows, Linux e macOS ed è disponibile in formato open-source con licenza Apache 2.0.

Selenium è un insieme di diversi componenti:

* Selenium IDE: è un plugin disponibile per Firefox e Chrome e offre un’interfaccia grafica che aiuta lo sviluppatore a realizzare i test. Tra le principali funzionalità che offre ci sono: strumenti per la registrazione delle sessioni e per la scrittura/debugging dei test. Solo a partire dal 2018, Selenium IDE iniziò ad essere mantenuto in modo attivo dalla comunità.
* Selenese: è il linguaggio in cui sono scritti i test, permette di inviare comandi al browser e esaminare il codice delle pagine. Sono disponibili svariate API per utilizzare Selenium senza utilizzare necessariamente Selense; per i maggiori linguaggi esistono delle versioni ufficiali aggiornate periodicamente (come per il C++, Java e python). Esistono però API non ufficiali per i linguaggi meno diffusi.
* Selenium Webdriver: è il cuore dell’applicazione, accetta comandi (inviati tramite Selenese o API) e li invia al browser. Per poter interagire con i vari browser ha bisogno di driver specifici (es. per Firefox e necessario geckodriver), questi gli permettono di tradurre le sue istruzioni generiche in una sintassi specifica per quel browser. Dalla versione 2.0, si è deciso di utilizzare funzionalità al livello del sistema operativo per interagire col browser, invece che utilizzare dei comandi in JavaScript; questo ha ridotto notevolmente il numero delle funzioni disponibili ma a favorito la creazione di versioni di Selenium ad och per svolgere compiti specifici.
* Selenium Remote Control (RC): è un server che acetta comandi via http. Scritto in Java, permette di scrivere test automatizzati per un’applicazione web in qualsiasi linguaggio di programmazione.

Per lo sviluppo di questo progetto è stato sufficiente utilizzare il WebDriver, rinominato Selenium Standalone Server, che dalla versione 3.0 in poi non necessita più del Remote Control, ma è in grado svolgere tutte le operazioni in modo autonomo. Sono state utilizzate diverse versioni nel corso del tirocinio ma quella definitiva è la 3.141.59 e consiste di una singola applicazione java. [15]

### 2.4.1 WebDriver

Il webdriver è un’interfaccia di controllo remota che permette di interagire con lo user agent; fornisce una piattaforma e un linguaggio che permettono ai programmi di inviare istruzioni al browser web. Permette di interfacciarsi e modificare gli elementi della pagina web (DOM), consente così di eseguire test automatizzati e script. [16]

Implementa un protocollo che permette la comunicazione tra:

* Local end: lato client, gestito tramite librerie che implementano le API.
* Remote end: server side, si dividono in nodi intermedi (che agiscono come proxy) e endpoint (destinazione).

Tra le principali funzionalità ci sono:

* Non essendo legato ad uno specifico framework, può essere facilmente integrato con framework per il testing come JUnit e TestNG.
* È in grado di gestire gli alert di Javascript, più frame e più finestre contemporaneamente.
* Permette di trovare le coordinate degli elementi nella pagina.
* Simulare il comportamento del mouse e la pressione dei tasti della keyboard.
* Supporta il testing per l’ambiente Android e iOS, il drag-and-drop, gli elementi AJAX e la navigazione tra le pagine.

### 2.4.2 Geckodriver

Questo programma si comporta come un proxy: fornisce le API http descritte dallo standard W3C del protocollo dei WebDriver; il suo compito è tradurre queste chiamate in altre comprensibili a Firefox.

È possibile specificare la posizione dell’eseguibile di Firefox che si vuole usare, eventuali parametri che gli si vogliono passare e la verbosità dei file di log. [17]

### 2.4.3 Driver Golang

Questo driver fornisce un client WebDriver per programmi scritti in Go; è disponibile su Github ed è mantenuto da “Miki Tebeka”. [18]

Questo package fornisce funzioni che permettono di avviare il server di Selenium (supporta sia Firefox che Chrome) e di inviargli dei comandi. Sarà quindi possibile navigare all’interno dei siti web, interagire con le pagine e gestire i cookie. Offre anche la possibilità di ridimensionare la finestra del browser e impostare un tempo massimo di attesa per il caricamento di una pagina.

# Funzionalità e implementazione del prototipo

## 3.1 Struttura Dati

Le strutture utilizzate per conservare le informazioni sugli autori e sugli articoli, sono conservate nel file “structures/structures.go”.

## 3.2 Ricerca su Google Scholar

### 3.2.1 Esplorazione su 1 livello (firstN)

### 3.2.2 Esplorazione in profondità (everFirst)

### 3.2.3 Esplorazione in parallelo (concurency)

#### 3.2.3.1 Multitasking in Go

## 3.3 Ricerca su Microsoft Academic

## 3.4 Prerequisiti

Il codice è stato sviluppato su Debian 9 ed è compatibile con tutte le distribuzioni Linux. Per poter eseguire il programma è necessario avere java 8 come default e il pacchetto “xvfb” [19] che fornisce un server X, che può essere eseguito su sistemi senza schermo e senza dispositivi fisici di input; questo pacchetto emula un framebuffer muto usando la memoria virtuale. Queste due dipendenze sono necessarie a Selenium.

È anche necessario avere l’ambiente di Go installato; con tutte le librerie base e le variabili d’ambiente settate in modo corretto (GOPATH e GOBIN). [19]

Dal momento che il codice è disponibile solo su Github, è necessario avere il comando sul terminale o l’applicazione desktop installata.

3.1 Fasi della ricerca

1. Cerca delle parole chiave su Google Scholar.
2. Top n dei risultati.
   1. Risultati Rilevanti: di questi articoli guardo quelli che li citano (ordinati anche questi per rilevanza) e prendo i top n (Sistema Classico). Costruisco il grafo delle citazioni da cui estrarrò quello dei related works.
   2. Risultati Più Recenti: prendo i più recenti e guardo chi citano, se ci sono degli articoli citati da molti allora sono importanti. Pb: GS non mi dice chi cito io mi cita ma solo chi mi cita. Alternativa: guardo i related di questi articoli per vedere se ce ne sono in comune, questi saranno importanti.
   3. Idea: potrei disporre gli elementi del grafo per data per avere un’idea di come si è evoluto l’argomento. Potrei filtrare i risultati per un certo range (es. 1980-1990) e ordinarli per rilevanza per ottenere i risultati più rilevanti per quel periodo. Quelli che compaiono anche in altre ricerche saranno quelli che hanno contribuito maggiormente allo state of art.
   4. Nota: gli articoli più giovani di una settimana difficilmente hanno dei lavori correlati. Obb: trovare i lavori correlati a quelli più recenti.
   5. Idea: trovare i top autori per un dato argomento. Come? Prendo gli autori degli articoli più importanti, cioè quelli che compaiono più spesso tra i risultati.
   6. Autori e Date potrei farlo solo per le sottocategorie che mi interessano (solo su MA).

IL PB È CHE SU GS NON CI SONO I TOPIC E NON È NOTO COME GS FACCIA IL RANKING E TROVI I RELATED.

3.2 Obbiettivo su Google Scholar

Trovare gli articoli principali che contribuiscono allo “stato dell’arte” di un argomento di ricerca.

Pb. Su Google Scholar non ci sono gli argomenti (i topic)

Approccio: Cerco delle parole chiave, trovo degli articoli e gli leggo, cerco sia tra quelli più rilevanti che tra quelli più recenti.

Se ora volessi fare un lavoro più approfondito dovrei vedere: i correlati e le citazioni di tutti gli articoli che ritengo più importanti tra quelli che ho appena letto. Così potrei trovare altri articoli rilevanti.

Idea: automatizzare questo processo: creo di grafi con la top degli articoli risultati da ogni ricerca (utilizzo la firstN). Poi per raffinare il processo posso considerare come articoli che appartengono allo stato dell’arte quelli che compaiono più spesso tra i risultati e costruisco un grafo solo con21q quelli.

3.2 Normalizzare il numero di citazioni su MA

Utilizzare gli articoli che hanno vinto il premio di Turing non funziona perché:

1. Spesso il premio viene conferito alla carriera di un ricercatore piuttosto che alla scrittura di un articolo particolarmente influente nel suo ambito di ricerca.
2. Spesso in MA (meno frequentemente anche in GS) gli articoli sono presenti più volte con date di pubblicazione diverse e tutte queste occorrenze hanno un numero di citazioni diverso. Come dovrei gestire questo problema? Considero il numero di citazioni come la somma di tutti quelli presenti, come solo quello della data di pubblicazione effettiva (che a volte non è neppure presente, in questo caso dovrei usare quello meno recente).

L’obbiettivo è quello di normalizzare il numero di citazioni di un articolo con quello dell’articolo che nello stesso anno di pubblicazione ha massimo il numero di citazioni. Per questo inizialmente si è pensato di usare il premio di Turing, in quanto è molto importante nell’ambito dell’informatica.

Alternativa: Per ogni anno, per ogni motore di ricerca: trovo l’articolo con max numero di citazioni e plotto come si evolve l’andamento.

Pb: il numero di citazioni varia molto da un argomento all’altro, è quindi necessario discriminare in base al topic (almeno in modo generale) (posso farlo solo su MA)

Pb: MA è difficile cercare un anno specifico e non è possibile ordinare i risultati perché sono troppi. Alternativa: potrei farlo solo per una certa categoria, una delle 19 top-level (computer science) ma non riesco comunque a ordinarli.

Pb: GS non permette di ordinare i risultati per numero di citazioni, sarebbe quindi necessario controllarli tutti per trovare il max.

METRICHE UTILIZZATE

Quanti articoli per firstN? OBB: anche se ne aumento il numero, non aggiungo articoli rilevanti che quindi appartengono al related work.

Quanti thread? COME: ripeto più volte la stessa ricerca con numero di thread diverso e trovo la massima efficienza (buttaci anche il grafo) (argmax(E)), caso mai mettici anche la tabella dello SpeedUp.

Sviluppi futuri

Incrociare i risultati dei documenti rilevanti tra Google Scholar e Microsoft Academic, confrontando i titoli degli articoli in modo intelligente (perché a volte i titoli differiscono tra i due motori di ricerca). Una soluzione potrebbe essere confrontare anche gli autori (che però non sono completamente presenti su Scholar) e la data (che non riesco a ricavare da Academic). Come ulteriore problema le date sono spesso diverse tra i due e lo stesso articolo è presente più volte con date diverse.

Riuscire a raccogliere più informazioni dagli articoli su Microsoft Academic, come la data e l’abstract.

# Bibliografia

|  |  |
| --- | --- |
| [1] | «Mestiere di Scrivere,» [Online]. Available: http://www.mestierediscrivere.com/articolo/articolotecnico.html. |
| [2] | U. d. V. D. d. C. e. Civiltà. [Online]. Available: http://www.dcuci.univr.it/documenti/OccorrenzaIns/matdid/matdid007359.pdf. |
| [3] | «Wikipedia: Stato dell'Arte,» [Online]. Available: https://it.wikipedia.org/wiki/Stato\_dell%27arte. |
| [4] | «Comparison Search Engine,» [Online]. Available: https://dspace3-labs.atmire.com/bitstream/handle/123456789/7634/338.pdf. |
| [5] | «Google Scholar's Ranking Algorithm,» [Online]. Available: https://www.gipp.com/wp-content/papercite-data/pdf/beel09.pdf. |
| [6] | «Ike Antkare,» [Online]. Available: http://rr.liglab.fr/research\_report/RR-LIG-008.pdf. |
| [7] | M. A. Topics. [Online]. Available: https://academic.microsoft.com/#/topics/0/. |
| [8] | P. K. Drahomira Herrmannova, «An Analysis of the Microsoft Academic Graph,» [Online]. Available: http://www.dlib.org/dlib/september16/herrmannova/09herrmannova.html. |
| [9] | P. M. Academic. [Online]. Available: https://preview.academic.microsoft.com/home. |
| [10] | «DB-Engines Ranking of DBMS,» [Online]. Available: https://db-engines.com/en/ranking/graph+dbms. |
| [11] | «Apoc Documentazione GraphML,» [Online]. Available: https://neo4j-contrib.github.io/neo4j-apoc-procedures/#graphml. |
| [12] | «Doc Golang Neo4j Driver,» [Online]. Available: https://godoc.org/github.com/johnnadratowski/golang-neo4j-bolt-driver. |
| [13] | johnnadratowski, «Golang Neo4j Driver,» [Online]. Available: https://github.com/johnnadratowski/golang-neo4j-bolt-driver. |
| [14] | «Selenium,» [Online]. Available: https://en.wikipedia.org/wiki/Selenium\_(software). |
| [15] | «Selenium Standalone Server Download,» [Online]. Available: https://www.seleniumhq.org/download/. |
| [16] | W3C, «WebDriver,» [Online]. Available: https://www.w3.org/TR/webdriver1/. |
| [17] | «Geckkodriver Releases,» [Online]. Available: https://github.com/mozilla/geckodriver/releases. |
| [18] | «Driver Selenium Golang,» [Online]. Available: https://github.com/tebeka/selenium. |
| [19] | «WebDriver Advantages,» [Online]. Available: https://www.softwaretestingclass.com/what-is-selenium-webdriver-selenium-training-series/. |