Autor

Marina López Alet

TREBALL DE FI DE GRAU

MetaResearch: Desenvolupament d'un crawler de conferències i articles científics en Ciències de la Computació

Dirigit per

Pedro García López

Grau d'Enginyeria Informàtica

UNIVERSITAT ROVIRA I VIRGILI

Tarragona, 2024

Resum

Aquest projecte se centra en el desenvolupament d'un crawler web o rastrejador web per extreure dades relacionades amb articles de recerca de diverses conferències de ciències de la computació. L'objectiu principal d'aquest és entendre la recerca científica analitzant conferències, Program Committees i citacions. El crawler, implementat en Python, utilitza les API d'OpenAlex i Semantic Scholar per extreure dades rellevants d'articles acadèmics publicats en les conferències seleccionades. Posteriorment, aquestes dades són emmagatzemades en dos tipus diferents de bases de dades, una de relacional pel Program Committee i les dades generals dels articles, i una de grafs, per les citacions.

En avaluar i identificar aspectes claus com la representació de països i continents, aquest estudi busca determinar si existeixen barreres que limiten l'accés a la recerca científica. L'estudi es basa en la premissa que la recerca ha de ser accessible per a tothom, independentment del seu origen geogràfic o cultural. Una representació desigual en les conferències pot advertir la presència d'aquestes barreres.

En conclusió, aquest projecte ofereix una eina per l'extracció de dades d'articles de conferències i brinda una perspectiva objectiva de la representació de països i continents en les conferències estudiades.

Resumen

Este proyecto se centra en el desarrollo de un crawler web o rastreador web para extraer datos relacionados con artículos de investigación de varias conferencias de ciencias de la computación. El objetivo principal es entender la investigación científica analizando conferencias, Program Committees y citas. El crawler, implementado en Python, utiliza las API de OpenAlex y Semantic Scholar para extraer datos relevantes de artículos académicos publicados en las conferencias seleccionadas. Posteriormente, estos datos se almacenan en dos tipos diferentes de bases de datos, una relacional para el Program Committee y los datos generales de los artículos, y una de grafos, para las citas.

Al evaluar e identificar aspectos clave como la representación de países y continentes, este estudio busca determinar si existen barreras que limitan el acceso a la investigación científica. El estudio se basa en la premisa de que la investigación debe ser accesible para todos, independientemente de su origen geográfico o cultural. Una representación desigual en las conferencias puede advertir la presencia de estas barreras.

En conclusión, este proyecto ofrece una herramienta para la extracción de datos de artículos de conferencias y brinda una perspectiva objetiva de la representación de países y continentes en las conferencias estudiadas.

Abstract

This project focuses on the development of a web crawler to extract data related to research articles from various computer science conferences. The main objective is to understand scientific research by analyzing conferences, Program Committees, and citations. The crawler, implemented in Python, uses the OpenAlex and Semantic Scholar APIs to extract relevant data from academic articles published in the selected conferences. Subsequently, these data are stored in two different types of databases, a relational one for the Program Committee and general article data, and a graph one for citations.

By evaluating and identifying key aspects such as the representation of countries and continents, this study seeks to determine if there are barriers that limit access to scientific research. The study is based on the premise that research should be accessible to everyone, regardless of their geographical or cultural origin. Unequal representation in conferences may indicate the presence of these barriers.

In conclusion, this project provides a tool for extracting data from conference articles and offers an objective perspective on the representation of countries and continents in the studied conferences.

ÍNDEX

Índex de figures

Índex de codi

Índex de taules

1. Introducció

En l'era digital actual, l'accés obert a la informació i la investigació és fonamental per l'avanç científic i tecnològic. No obstant això, aquest accés pot veure's limitat per barreres geogràfiques, econòmiques i culturals. En aquest context, aquest treball se centra en el desenvolupament d'un crawler web, utilitzant el llenguatge de programació Python i les API[1] d'OpenAlex i Semantic Scholar. La finalitat d'aquest crawler és extreure dades d'interès relacionades amb els articles acadèmics i de recerca publicats en certes conferències de computació. A més a més, aquestes s'emmagatzemaran en un format estructurat usant dos tipus de bases de dades. El primer tipus, una base de dades relacional, contindrà de forma estructurada tota la informació relacionada amb els membres del Program Committee i les dades generals dels articles. El segon tipus, una base de dades de grafs, contindrà tota la informació relacionada amb les citacions dels articles.

Es vol identificar i avaluar aspectes rellevants de les conferències, com els països que hi publiquen amb les seves respectives institucions, les citacions que es produeixen en els articles publicats, així com els Program Committee de les conferències. Se sosté la premissa que la recerca ha de ser accessible per a tothom, independentment del seu origen geogràfic o cultural. Per tant, la manca de representació equitativa en les conferències podria indicar l'existència de barreres que limiten l'accessibilitat a la recerca científica.

Aquest projecte s'ha realitzat en el grup d'investigació CloudLab de la Universitat Rovira i Virgili (URV). Agraeixo molt l'ajuda que he rebut per part de Pedro García, que m'ha ajudat en el que he necessitat, i que m'ha presentat a Anwitaman Datta, professor associat a l'Escola de Ciències de la Computació i Enginyeria de la Nanyang Technological University de Singapur. Les reunions que hem realitzat els tres per compartir avenços i resultats d'aquest projecte m'han servit per millorar el meu coneixement en tot aquest àmbit de la investigació.

[1] Interfície de Programació d'Aplicacions (Application Programming Interface). Permeten la comunicació i compartició de dades entre aplicacions utilitzant un conjunt de regles i protocols.

1.1. Contextualització

En les últimes dècades, la tecnologia ha avançat a passos agegantats, gràcies en part a la recerca que es duu a terme els diferents camps de la informàtica. Alguns dels més sonats avui en dia podrien ser el Cloud Computing o la Intel·ligència Artificial, entre d'altres, i cada cop n'hi apareixen més. Aquesta recerca ha fet possible que la tecnologia que abans només era accessible a una certa part de la població, ara està a l'abast de pràcticament qualsevol individu, però, ha succeït el mateix amb aquestes comunitats científiques? Formar part d'aquestes és també accessible per la major part dels individus, o bé continuen estant limitades? Aquesta és la principal pregunta que vol resoldre aquest projecte.

Tanmateix, aquest projecte no és el primer que aborda aquest tema, ja que durant els últims anys, diversos projectes similars s'han anat publicant. Un dels més coneguts és Security Circus, creat per Davide Balzarotti. Aquest és un projecte que mostra diverses dades referents als autors, institucions i països de diferents conferències de Seguretat de Sistemes. Principalment, se centra a determinar quins són els Autors que hi publiquen més articles, o les institucions que hi apareixen més sovint. També hi trobem altres projectes que se centren el l'àmbit de la computació, com per exemple CS-Insights desenvolupat per Jan Philip Whale. Malauradament, aquest projecte encara està en desenvolupament, i pel que sembla, no se centrarà a realitzar un estudi de les dades en si mateixes, sinó que volen crear una pàgina web que sigui capaç de mostrar les dades recopilades de forma senzilla i visual, per tal d'ajudar a futurs projectes de recerca en aquest àmbit.

Així mateix, a part dels dos projectes mencionats, si realitzem una extensa cerca per internet, podem trobar-hi alguns altres projectes que analitzen les dades relacionades amb els articles d'investigació publicats en diverses conferències, però cap d'aquests sembla encarar-lo com ho fem en aquest projecte, ja que pràcticament en la majoria dels projectes on s'estudien conferències en l'àmbit de la investigació en la informàtica, les anàlisis solen mostrar la diferència en el nombre d'articles publicats per conferència, els autors que més articles publiquen, i fins i tot, quins autors solen col·laborar més sovint en la publicació d'articles. El que creiem que manca en aquests projectes, i creiem que podem aportar-ho en el nostre, és una visió des de l'àmbit geogràfic i de comunitats. És a dir, amb aquest projecte s'espera anar més enllà i aconseguir entendre les dinàmiques

dels camps que s'analitzen, creant bases de dades que serviran per a que la comunitat pugui aprendre i analitzar el que passa dins d'aquest camp.

1.2. Objectius del projecte

L'objectiu principal d'aquest projecte és entendre la recerca científica analitzant conferències, els seus respectius Program Committees i les citacions realitzades pels articles de les conferències. Fonamentalment, es vol entendre com funciona el camp de recerca estudiat.

Per tal d'assolir aquest objectiu s'ha de realitzar MetaResearch (investigació sobre la investigació). Això consisteix en estudiar els articles publicats, les conferències, els Program Committees i les cites. Les dades recopilades s'analitzaran per extreure conclusions sobre aquest camp de la recerca.

Per tal d'assolir aquest objectiu es desenvoluparà un sistema d'extracció i recopilació de dades, qui sigui eficient, robust, fàcil d'usar i flexible, per tal d'ajustar-se a les necessitats de l'usuari. Aquest sistema prendrà la forma d'un crawler web, ajudant-se de dues API per tal d'aconseguir totes les dades que, posteriorment, ens seran rellevants.

Així doncs, els objectius plantejats per aquest projecte son els mostrats a continuació:

- Desenvolupar un crawler funcional, que permeti l'extracció de dades de DBLP, i les API d'OpenAlex i Semantic Scholar.
- Comprendre el funcionament bàsic d'un crawler web, per tal d'adaptar-lo a les necessitats d'aquest projecte.
- Investigar les plataformes que s'usaran per a l'extracció de dades, per garantir que totes les operacions que es realitzaran sobre aquestes estan permeses.
- Investigar l'estructura de les API d'OpenAlex i Semantic Scholar, així com les funcions que aquestes ofereixen. D'aquesta forma es podrà desenvolupar un crawler eficient.
- Assegurar la persistència de les dades recopilades mitjançant dos tipus de bases de dades: relacional (Program Committee i dades dels articles) i grafs (per citacions).

- Analitzar les dades per poder identificar aspectes que es considerin rellevants, com els països que participen en les diferents conferències, els membres del Program Committee i les citacions que realitzen els articles de les conferències.

2. Crawling Web

En aquest apartat es detallaran els conceptes teòrics relacionats amb el desenvolupament d'un crawler, així com el seu funcionament bàsic. També es parlarà de les plataformes que s'han usat per realitzar el crawling de les dades necessàries en aquest projecte.

2.1. Què és?

El crawling web és una tècnica que permet recórrer llocs web de manera sistemàtica i recopilar-ne informació rellevant. Per dur-lo a terme s'ha de desenvolupar un programa informàtic que permeti anar accedint de forma recursiva a les diferents pàgines web. Aquest programa es coneix amb el nom de crawler, scrapper o spider.

2.2. Funcionament

El funcionament d'un crawler és bastant senzill, però un dels punts forts que fan que un crawler sigui realment eficaç és que el mateix programador pot dissenyar-lo a mida segons les seves necessitats, i és aquest detall el que el fa realment útil a l'hora de recopilar informació.

En la figura 1 es mostra un esquema que detalla el flux d'execució d'un crawler bàsic. El programa comença amb l'usuari introduint l'URL[2] des de les qual s'iniciarà la cerca. Aquestes URL també s'anomenen *seeds*. Seguidament, el crawler comença a realitzar les peticions web necessàries per obtenir les dades de les pàgines web corresponents a les URL. Corresponent a cada petició, el crawler obté una resposta, que serà l'HTML[3] de la pàgina si no ha ocorregut cap error, o en cas contrari, rebrà un codi d'error. Suposem que no s'ha produït cap error en la petició, així doncs el crawler continua l'execució processant el contingut HTML que ha rebut. A partir d'aquest hi pot extreure pràcticament tota la informació que està visible a la pàgina, com podrien ser imatges, vídeos, altres links o, fins i tot, el mateix text. Més endavant, quan ha processat tota la informació que es considera rellevant, el crawler la indexa de tal forma que sigui fàcilment accessible més endavant. Finalment, depenent de com s'hagi programat el

[2] Uniform Resource Locator

[3] HyperText Markup Language

crawler, continuarà fent peticions usant les noves URL que ha extret del codi HTML, o en cas contrari, acabarà la seva execució. Normalment, les dades recopilades s'emmagatzemaran en algun tipus de memòria, sigui usant alguna mena de base de dades o bé usant fitxers.

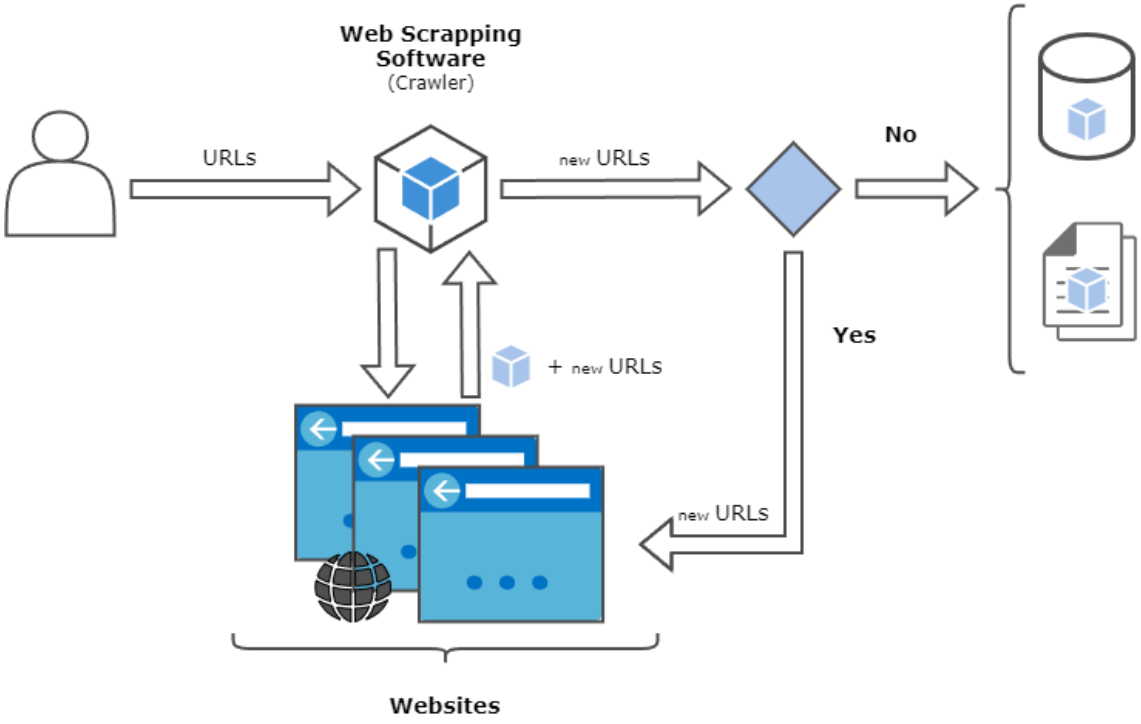


Figura 1. Esquema del funcionament bàsic d'un crawler web

No obstant això, s'han de tenir en consideració diversos aspectes abans de realitzar crawling sobre una pàgina web. Els més importants són els relacionats amb les polítiques de crawling d'un determinat lloc web, i és que realitzar aquest procés no sempre és possible. Hi ha certs llocs web que tenen un fitxer anomenat robots.txt, aquest determina les pàgines que poden ser rastrejades per un crawler i quines no. És important respectar aquestes polítiques per evitar problemes legals o bloquejos per part del lloc web.

2.3. Plataformes Utilitzades

Com s'ha comentat en apartats anteriors, l'objectiu d'aquest projecte és realitzar un crawler que permeti l'obtenció de dades relacionades amb els articles de recerca publicats en diferents conferències, per la qual cosa és fonamental estudiar d'on es poden obtenir aquestes dades. Així doncs, en aquest apartat es detallaran les plataformes i eines que s'han utilitzat per realitzar el crawling de les dades necessàries.

2.3.1. DBLP

DBLP (Digital Bibliography & Library Project), és una base de dades en línia i un servei de recopilació de bibliografia en l'àmbit de la informàtica i la ciència de la computació. Originalment, va ser creat com una base de dades bibliogràfica enllaçada, amb l'objectiu de recopilar informació exhaustiva sobre articles de revistes, actes de conferències, llibres i altres publicacions en aquest àmbit. Amb els anys s'ha convertit en una eina essencial per a investigadors i acadèmics en el camp de la informàtica.

Figura 2. Logotip de DBLP

A continuació s'enumeren alguns dels motius que han convertit a aquesta base de dades en línia en un dels llocs predilectes pel que fa a bibliografies en aquest camp.

- Contingut: DBLP ofereix una extensa col·lecció de registres bibliogràfics d'una àmplia gamma de temes relacionats amb la informàtica i ciència de la computació. Aquestes dades inclouen informació sobre articles, conferències, i altres tipus de publicacions rellevants.
- Accessibilitat: Una de les característiques principals de DBLP, i que presenta un gran avantatge respecte eines similars. DBLP està disponible en línia de forma completament gratuïta, per tant, qualsevol persona interessada a accedir a informació bibliogràfica pot fer-ho a través de la seva pàgina web.
- Interfícies de cerca senzilla: DBLP ofereix un sistema de cerca que permet als usuaris realitzar cerques de publicacions usant diferents camps i característiques d'aquestes, com podria ser el títol, autors, conferència en què s'ha publicat, any de publicació, etc.

En conclusió, DBLP és una de les millors eines que es poden trobar avui en dia pel que fa a articles en el camp de la informàtica i de la ciència de la computació, però hi ha, una última característica que fa destacar DBLP respecte a altres plataformes similars, i que és explotada per aquest projecte, i es tracta de la possibilitat de realitzar crawling sobre la seva pàgina web. Com s'ha comentat en l'apartat 2.1, hi ha llocs web que no permeten realitzar crawling sobre elles, en el cas de DBLP, ells mateixos comenten en

una secció de la seva web que és possible realitzar crawling sobre les seves dades, i fins i tot posen a disposició una petita API per facilitar-ne l'obtenció d'aquestes dades.

2.3.2. OpenAlex

OpenAlex és una plataforma que indexa una gran varietat de dades relacionades amb articles acadèmics. Aquesta plataforma va més enllà de simplement tenir informació sobre els articles en si, si no que proporcionen informació detallada sobre les relacions entre entitats científiques, investigadors, institucions i altres recursos relacionats amb la recerca acadèmica. OpenAlex és l'eina perfecta per a poder obtenir tota la informació relacionada amb les institucions en les quals estan afiliats els autors d'un article, ja que proporcionen informació detallada de més de cent mil institucions d'arreu del món.

Figura 3. Logotip d'OpenAlex

Un altre de les principals característiques que fan d'OpenAlex una perfecta plataforma per l'obtenció de dades relacionades amb la recerca acadèmica, és la possibilitat d'accedir a les dades mitjançant una API. Aquesta permet als desenvolupadors poder accedir a les dades i integrar-les en aplicacions i sistemes externs, com en el crawler en el cas d'aquest projecte. A més cal destacar que l'ús d'aquesta API és gratuït i que no cal cap mena d'autentificació per usar-la, a més a més, el límit de peticions per segon per usuari és bastant elevat, per la qual cosa no presenta un gran inconvenient.

Tanmateix, aquesta API està dissenyada per facilitar el seu ús, distingint entre diferents endpoints pels diferents tipus de dades que proporciones. Aquest fet fa que sigui

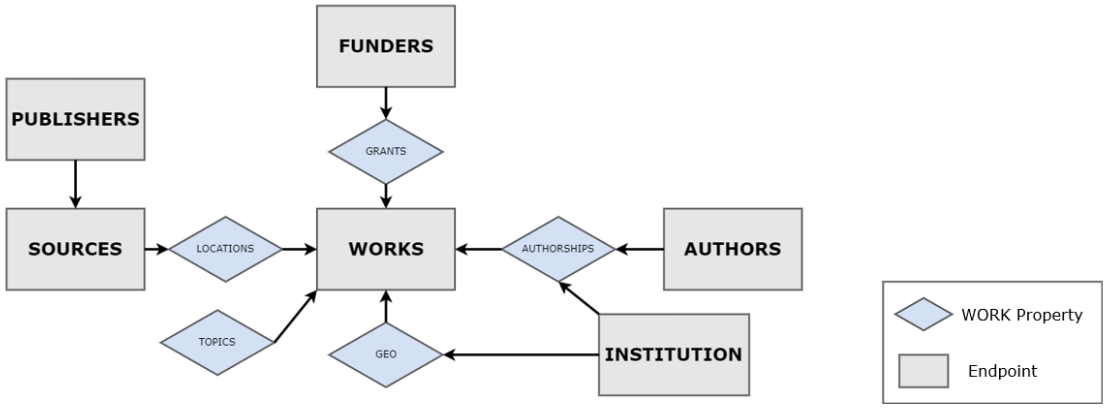


Figura 4. Estructura interna dels endpoints d'OpenAlex

molt més eficient accedir a un tipus determinat de dades, ja que simplement haurien d'accedir a l'endpoint desitjat i fer una consulta per tal d'obtenir les dades desitjades.

En definitiva, OpenAlex és una eina molt completa, que ofereix accés a una gran varietat de dades relacionades amb articles acadèmics, entre aquestes es troba informació respecte als autors i les institucions a les quals aquests estaven afiliats en el moment de la publicació de l'article. Aquestes són les dades que més útils ens seran d'aquesta plataforma, i les que s'obtindran mitjançant el crawler web.

2.3.3. Semantic Scholar

Semantic Scholar és un motor de cerca acadèmica impulsat per la intel·ligència artificial, que proporciona de forma gratuïta eines pel descobriment i recursos adreçats a tothom qui estigui interessat en la recerca.

Figura 5. Logotip de Semantic Scholar

Un dels factors que diferencia a Semantic Scholar respecte d'altres eines similars és l'anàlisi del contingut de les publicacions. I és que Semantic Scholar utilitza tècniques d'intel·ligència artificial i processament del llenguatge natural per analitzar i comprendre el contingut dels articles dels quals disposa. Aquest fet li permet extreure metadades, identificar relacions entre conceptes, detectar les citacions, d'entre altres elements rellevants.

D'altra banda, també utilitza aquestes eines d'intel·ligència artificial i processament del llenguatge natural per fer una indexació semàntica dels continguts dels diferents articles. Això permet que Semantic Scholar pugui fer cerques més precises i rellevants, ja que és capaç d'identificar i relacionar articles que tracten de temes similars, fins i tot si la terminologia que aquests utilitzen és diferent.

D'igual forma que amb OpenAlex, la característica que ens interessa d'aquesta plataforma en relació amb el projecte és l'API que aquesta proporciona. Mitjançant crides a aquesta API, es pot obtenir tota la informació relacionada amb un determinat article o articles. En concret, la informació que més ens interessa pel desenvolupament d'aquest projecte són els abstracts i citacions dels diferents articles. Aprofitant la intel·ligència artificial que proporciona aquesta plataforma, també ens interessa obtenir el que ells

anomenen TLDR. Aquest es tracta d'un resum generat automàticament a partir del contingut de l'article.

Malgrat tot, l'API de Semantic Scholar presenta alguns inconvenients respecte de la d'OpenAlex. El més rellevant és el límit de peticions que es poden realitzar per segon, el qual és bastant baix, i en apartats següents es podrà veure com aquest fet limita l'eficiència i rendiment del crawler. No obstant això, és possible incrementar aquests límits usant una clau d'accés privada per l'API. Aquesta es pot aconseguir de forma gratuïta omplint un formulari en la web de Semantic Scholar.

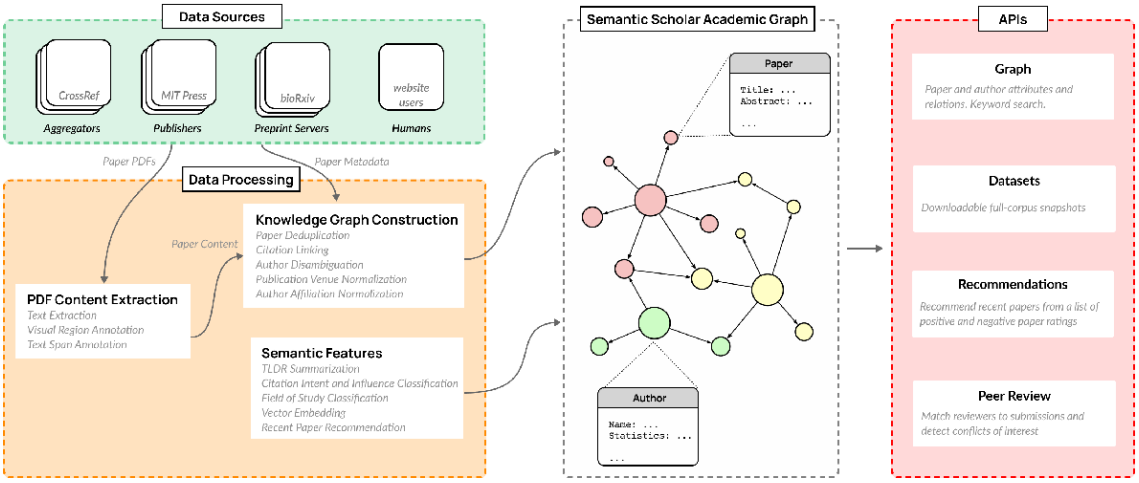


Figura 6. Esquema del funcionament intern de Semantic Scholar

En conclusió, Semantic Scholar és una eina molt potent gràcies, en part, a la intel·ligència artificial que aquesta implementa.

3. Disseny del Crawler

En els següents apartats es detallaran tots els aspectes rellevants que han format part de l'etapa del disseny del crawler.

3.1. Requisits

El crawler ha de complir amb certs requisits, per tal de considerar-lo exitós en la seva tasca d'extracció de dades.

Primerament, aquest ha de ser capaç d'accedir a les pàgines web, i extreure la informació que es considera rellevant per al projecte. A més a més, aquest ha d'intentar ser el més genèric possible, ja que ha de permetre extreure dades de diferents conferències sense la necessitat d'haver de modificar-lo.

Aquest últim punt, encara que sembla trivial i obvi, és més complicat del que pot semblar, perquè encara que les dades s'extreuen dels mateixos llocs web, no és estrany que cada conferència tingui les dades estructurades de formes diferents. És per això que un s'ha de realitzar un estudi exhaustiu de cada una de les plataformes que s'han d'utilitzar per al crawling.

3.2. Dades a extreure

Com s'ha anat comentant al llarg d'aquest document, les dades a obtenir estan relacionades amb articles de recerca publicats en diverses conferències, però encara no s'han comentat de forma detallada quines són les dades concretes que s'han d'extreure. És per això que en aquest apartat s'enumeraran les dades que es consideren rellevants per poder realitzar el posterior estudi i, per tant, les que s'han d'extreure de DBLP, OpenAlex i Semantic Scholar.

Per cadascuna de les conferències que es volen estudiar s'han d'aconseguir els articles que han estat publicats en un rang d'anys determinat. Aquest rang d'anys pot ser variable i se li ha d'indicar al crawler. En la taula que es troba a continuació es pot observar les diferents dades que volem obtenir per cadascun dels articles, així com el lloc del qual s'obtindrà i si aquestes dades estan sempre disponibles. Aquest últim punt fa referència a si s'hi poden trobar casos on la dada en qüestió no té valor o no està disponible a cap de les fonts que s'utilitzen.

Dada	Lloc d'on s'extreu	Està sempre disponible?
Títol de l'article	DBLP	SI
Any de publicació	DBLP	SI
Tema (topic) de l'article	DBLP	NO
Número DOI	OpenAlex, Semantic Scholar	NO
Link de l'article en OpenAlex	DBLP	NO
ID de l'article en Semantic Scholar	Semantic Scholar	NO
Noms del Autors	DBLP, OpenAlex	SI
Institucions dels autors	OpenAlex	NO
País de les respectives institucions	OpenAlex	NO
Treballs Citats	OpenAlex, Semantic Scholar	NO
Abstract de l'article	Semantic Scholar	NO
TLDR (Resum de l'article)	Semantic Scholar	NO

Taula 1. Dades que s'extreuen amb el crawler

Com s'observa, es tracta d'un nombre elevat de dades per cada article, encara que s'ha de tenir en compte que no tota serà utilitzada en l'estudi que es realitzarà posteriorment. De fet, algunes de les dades no s'obtenen amb l'objectiu de ser estudiades posteriorment, sinó amb l'objectiu de poder accelerar el procés d'obtenció de la resta de dades.

En relació amb l'acceleració de dades, s'ha de tenir en compte que l'obtenció d'aquestes està limitada pel límit en el nombre de peticions que tant OpenAlex i Semantic Scholar accepten en el seu sistema. Aquest és un fet clau, ja que determina en gran manera el temps que triga el crawler a obtenir totes les dades.

3.3. Funcionament

Per tal de mitigar el temps que triga el crawler a extreure totes les dades que es volen, s'han utilitzat dues tècniques.

Primerament, es fa ús de programació paral·lela, creant diferents fils de programació que realitzen el crawling de forma concurrent. S'ha d'anar amb compte en la utilització d'aquesta, ja que és molt senzill sobrepassar el límit de peticions a una pàgina web o API usant-la.

En segon lloc, s'ha dividit el crawler en tres parts diferents, per tal de no saturar els servidors d'OpenAlex i Semantic Scholar amb el nombre de peticions tan elevades que realitza el crawler. Fent aquesta divisió, cadascuna de les tres parts es pot executar en instants separats temporalment, encara que aquestes no són independents entre si.

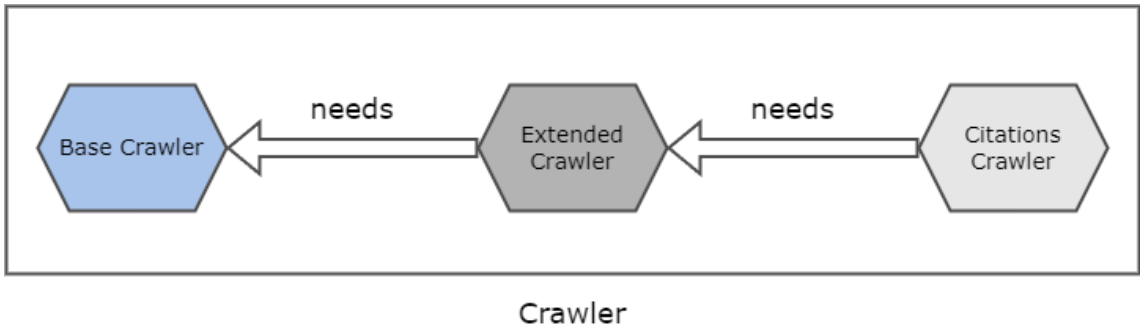


Figura 7. Esquema dels tres crawlers

En la figura 7 es mostra un esquema del crawler amb les tres parts diferenciades. Com es pot veure, només la primera part anomenada *Base Crawler* és independent de la resta, el que vol dir que es pot executar de forma independent. La segona part, anomenada *Extended Crawler* requereix la primera per poder executar-se, i l'última part, anomenada *Citations Crawler* requereix les dues parts anteriors.

A continuació es mostra un petit diagrama de classes que pretén il·lustrar de forma senzilla el funcionament general dels tres crawlers dissenyats i implementats.

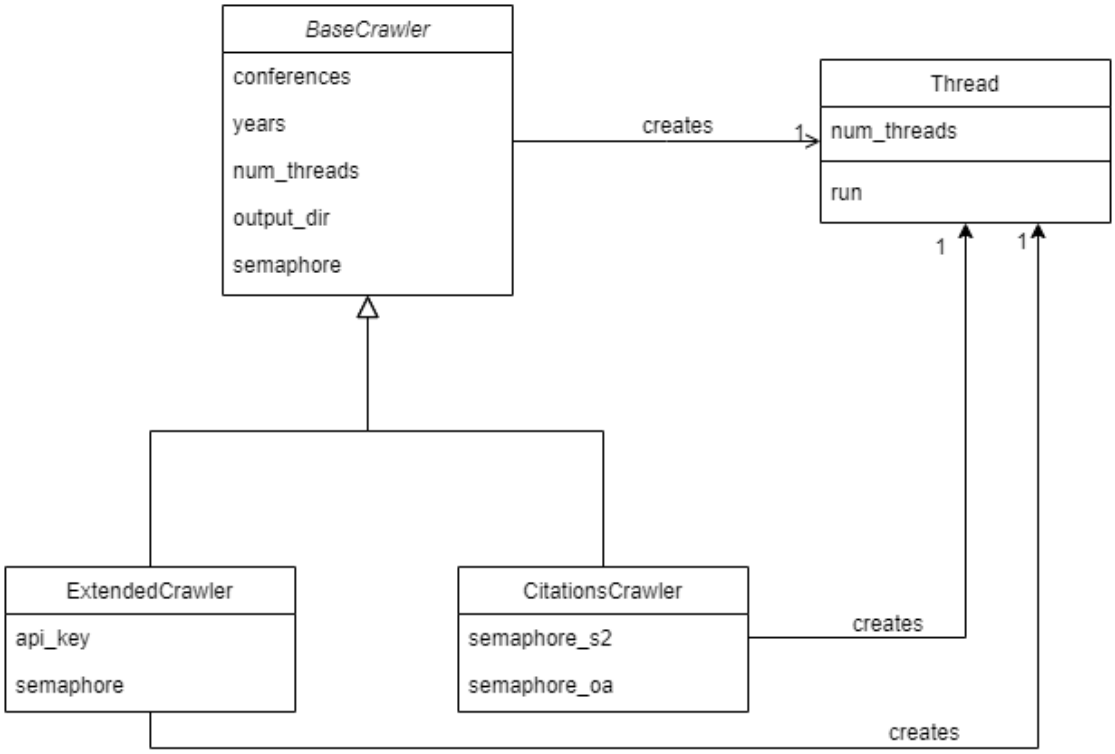


Figura 8. Diagrama de classes dels crawlers

En la figura anterior es pot observar l'esquema de classes que hi ha en aquest projecte. Convé ressaltar que no tots els fitxers de codi que utilitza aquest projecte segueixen un esquema de POO[4], sinó que s'ha aprofitat el fet que Python permet utilitzar diferents esquemes de programació en el mateix projecte. És per aquest motiu que no tots els fitxers del projecte es troben en el diagrama de classes de la figura anterior.

Continuant amb el diagrama de classes de la figura 8, podem veure com hi ha tres classes pels crawlers, una per cadascun dels que s'ha detallat en l'apartat de disseny, i també trobem una classe anomenada Thread. Aquesta se n'encarrega de crear els fils d'execució necessaris. En apartats posteriors, es mostrarà més en detall aquesta classe. Pel que fa a les classes dels crawlers, podem veure com tenim una classe principal anomenada *BaseCrawler*, de la qual hereten les altres dues classes de crawlers, *ExtendedCrawler* i *CitationsCrawler*. En realitzar aquesta herència, les classes filles aprofiten els atributs de la classe pare, així com alguns dels seus mètodes.

[4] Programació Orientada a Objectes

3.3.1. Base Crawler

Aquest crawler és el més senzill de tots, i és el que obté les que podríem considerar les dades principals dels articles, com serien el títol, l'any de publicació i els autors i les seves respectives institucions. Tanmateix, també s'extreuen dades addicionals que podrien ser útils per l'extracció d'altres dades o bé pel seu posterior estudi.

A continuació es mostra el diagrama on es pot veure el flux d'execució d'aquest crawler. Seguidament, es detallarà cadascun dels punts que es troben assenyalats.

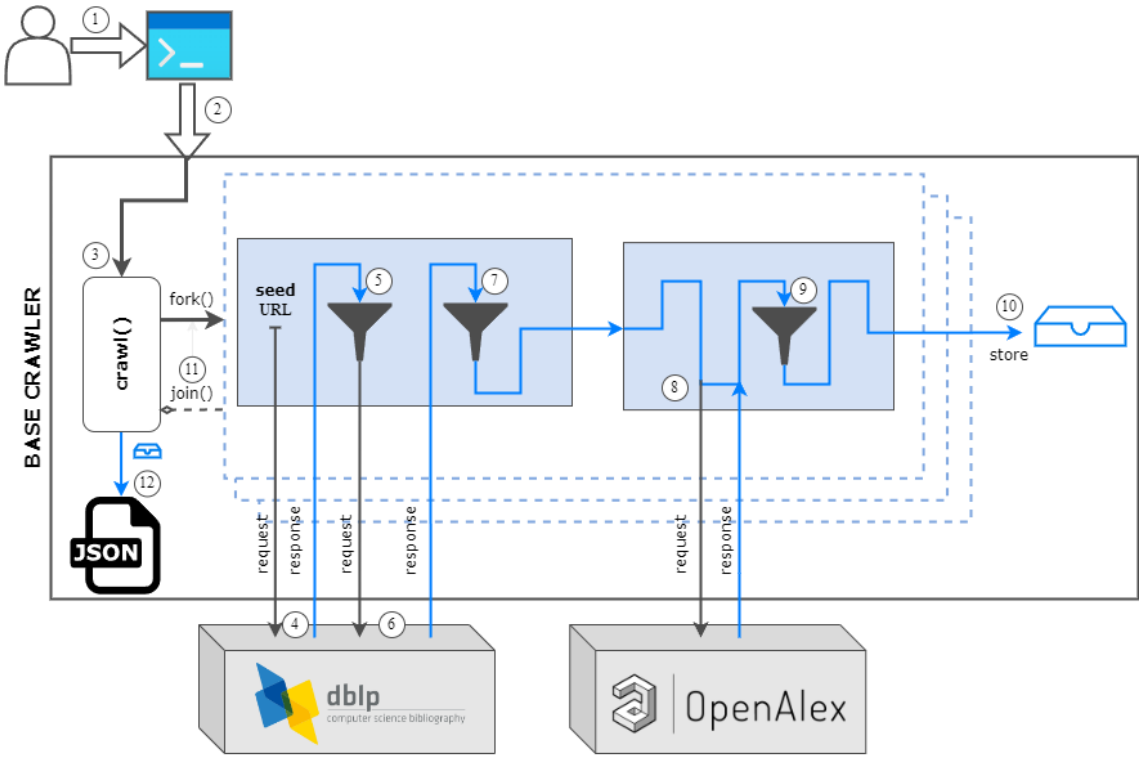


Figura 9. Diagrama del funcionament del Base Crawler

1. L'usuari introdueix la comanda per executar el crawler mitjançant un terminal.
2. El programa processa els arguments introduïts per l'usuari. Els arguments introduïts podrien modificar el funcionament del crawler, ja que, per exemple, l'usuari podria modificar el nombre de fils d'execució que es creen.
3. Els arguments arriben a la funció *crawl()* que s'encarrega d'ajustar el funcionament del crawler respecte als paràmetres que ha rebut. Crea tants fils d'execució com s'hagi indicat, o en cas contrari, no es crearà cap fil d'execució addicional (execució seqüencial). En el cas que es realitzés paral·lelització, els següents punts fins al 10 (inclòs) serien executats en paral·lel per tots els fils d'execució.

4. A partir de la Seed URL que s'obté mitjançant el nom de la conferència de la qual es vol obtenir les dades, es realitza una petició a DBLP amb aquesta URL. DBLP rep la petició, la processa i envia una resposta. Si no ha ocorregut cap error, la resposta conté el codi HTML de la pàgina web corresponent a l'URL. En cas contrari, rebem un codi d'error i es pararà l'execució del crawler.
5. El codi HTML rebut es processa per tal d'obtenir tots els links necessaris. En aquest cas es tracten dels links de la pàgina on es troben els articles d'un determinat any. El nombre de links que s'obtindrà finalment dependrà de diversos factors, com l'estructura en què la conferència ha distribuït els articles en DBLP o el nombre d'anys dels quals es vulguin obtenir dades.
6. Es realitzen peticions a DBLP per cadascun dels links obtinguts en el pas anterior, per cada petició rebrem una resposta. En cas que no hi hagi error en la resposta enviada per DBLP, obtindrem l'HTML de la pàgina on hi ha tota la informació relacionada amb els articles publicats. En cas d'error, rebrem un codi d'error i es passarà a realitzar la petició del següent link.
7. El codi HTML rebut en la resposta del pas anterior es processa per tal d'aconseguir tan sols un tros del codi, aquest serà processat a continuació per les funcions encarregades del processament de les dades.
8. Es comencen a extreure les dades dels articles, com el títol, l'any de publicació i els autors. A més, si està disponible, s'extreu un link que algun dels articles de DBLP poden tenir associats juntament amb la resta de dades. Aquest es tracta d'un link directe a l'API d'OpenAlex. Si aquest link s'ha pogut extreure, es realitza una petició a l'API mitjançant el link. El servidor d'OpenAlex processa la petició i retorna una resposta. En cas que no s'hi produeixi cap error, la resposta estarà en format JSON.
9. Es processa la resposta rebuda en el pas anterior. En cas que hagi retornat un codi d'error se segueix amb l'extracció de la resta de dades. Contràriament, la resposta rebuda en format JSON serà processada per extreure les dades relacionades amb els autors, les institucions i el país en el qual estan localitzades.
10. Finalitzant amb les etapes d'extracció de dades, totes les dades dels articles que han estat recopilades es guarden en una estructura de dades. Aquesta estructura és global i tots els threads que s'han creat hi poden accedir.
11. Quan tots els threads creats acaben la seva execució, es realitza un join i es torna a la funció *crawl()*. En el cas que l'execució no hagi estat paral·lela, es retorna a

la funció *crawl()* quan l'únic thread emmagatzema les dades a l'estructura anterior.

12. Finalment, les dades que han estat emmagatzemades en l'estructura de dades s'escriuen a un fitxer en format JSON.

3.3.2. Extended Crawler

Aquest crawler extreu dades que complementen les dades aconseguides amb el crawler explicat en el punt anterior. Aquestes dades s'extreuen completament usant Semantic Scholar i estan relacionades amb els articles que cita un article determinat, així com els abstracts i resums dels diferents articles d'una conferència.

Així mateix, és important recordar que per poder fer ús d'aquest crawler, primerament s'ha d'haver executat el *Base Crawler*.

A continuació es mostra el diagrama on es pot veure el flux d'execució d'aquest crawler. Seguidament, es detallarà cadascun dels punts que es troben assenyalats.

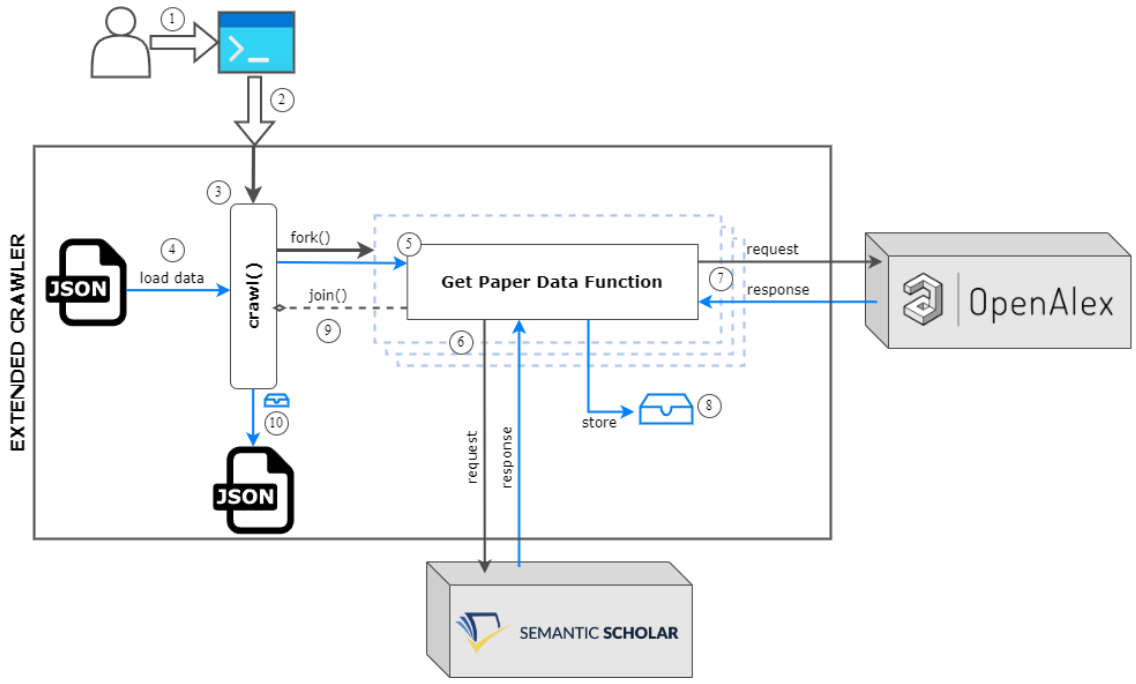


Figura 10. Esquema del funcionament de l'Extended Crawler

1. L'usuari introdueix la comanda per executar el crawler mitjançant un terminal.
2. El programa processa els arguments introduïts per l'usuari. Els arguments introduïts podrien modificar el funcionament del crawler, ja que, per exemple, l'usuari podria modificar el nombre de fils d'execució que es creen.

3. Els arguments arriben a la funció *crawl()* que s'encarrega d'ajustar el funcionament del crawler respecte als paràmetres que ha rebut. Crea tants fils d'execució com s'hagi indicat, o en cas contrari, no es crearà cap fil d'execució addicional (execució seqüencial). En el cas que es realitzés paral·lelització, els punts 5 fins al 8 (inclòs) es realitzarien concurrentment per tots els threads.
4. Es llegeixen les dades que s'han extret amb el *Base Crawler*, en cas que no es trobin, es mostra un missatge d'error i es demana a l'usuari que executi el *Base Crawler* per, posteriorment, executar aquest crawler.
5. Si les dades anteriors s'han pogut carregar de forma satisfactòria, aquestes són passades a les funcions encarregades de realitzar les peticions, per posteriorment extreure les dades que es consideren necessàries.
6. Mitjançant les dades anteriors, es realitzen peticions a l'API de Semantic Scholar. El servidor pertinent processa les dades i envia una resposta. En cas que no hagi ocorregut cap error, la resposta conté les dades sol·licitades en format JSON[5]. En cas contrari, depenent del tipus d'error rebut s'esperen uns segons i es torna a realitzar una petició al servidor. Finalment, si es trona a trobar un error, les dades es guarden amb valors nuls i es prossegueix a realitzar la següent petició.
7. Les dades rebudes en la resposta del punt anterior són processades, i si es dona un cas determinat, s'efectua una petició a l'API d'OpenAlex. Aquesta petició només es realitza si l'article del qual s'està obtenint la informació té valors nuls en les dades d'OpenAlex obtingudes amb el *Base Crawler*, i si a més, amb les dades aconseguides de Semantic Scholar, s'ha pogut extreure el número DOI[6] de l'article. És mitjançant aquest número que es realitza la petició a OpenAlex per intentar aconseguir les dades que no s'han pogut extreure amb el *Base Crawler*.
8. Després d'haver obtingut totes les dades necessàries, aquestes són emmagatzemades a una estructura de dades. Aquesta és global i pot ser accedida pels diferents fils d'execució.
9. Quan tots els threads creats acaben la seva execució, es realitza un join i es torna a la funció *crawl()*. En el cas que l'execució no hagi estat paral·lela, es retorna a la funció *crawl()* quan l'únic thread emmagatzema les dades a l'estructura anterior.

[5] Java Script Object Notation

[6] Digital Object Identifier

10. Finalment, les dades que han estat emmagatzemades en l'estructura de dades s'escriuen a un fitxer en format JSON.

3.3.3. Citations Crawler

Aquest és l'últim dels crawlers que es proporcionen. Les dades que aquest obté no pertanyen directament als articles publicats en una conferència, però hi tenen relació. En concret, es tracten de les dades pertanyents als articles que són citats per articles d'una conferència.

Aquestes dades s'aconsegueixen mitjançant algunes de les dades obtingudes en els crawlers anteriors, i es fa ús de les API de Semantic Scholar i OpenAlex.

Pel que fa a les dades en si, es podria considerar que són les dades principals dels articles citats, com el títol, els autors i les seves respectives institucions, l'any de publicació i en quina conferència (si n'hi ha) va ser publicat aquest article.

A continuació es mostra el diagrama on es pot veure el flux d'execució d'aquest crawler. Seguidament, es detallarà cadascun dels punts que es troben assenyalats.

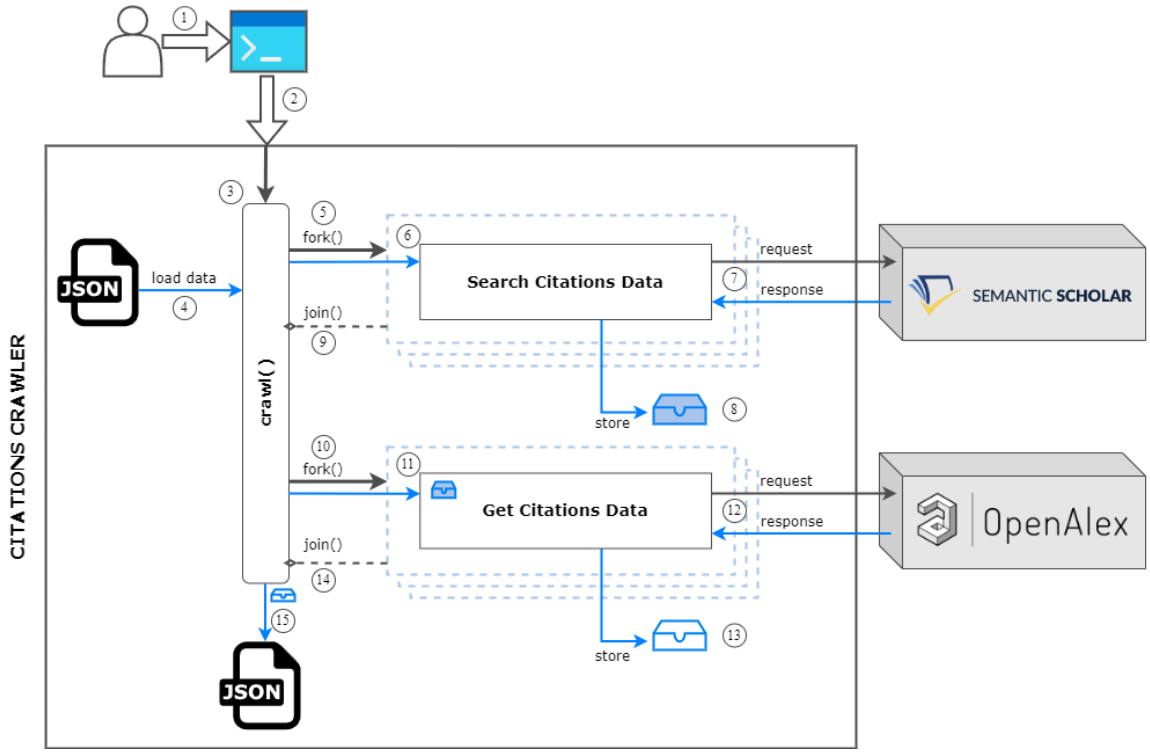


Figura 11. Esquema del funcionament del Citations Crawler

1. L’usuari introdueix la comanda per executar el crawler mitjançant un terminal.
2. El programa processa els arguments introduïts per l’usuari. Els arguments introduïts podrien modificar el funcionament del crawler, ja que, per exemple, l’usuari podria modificar el nombre de fils d’execució que es creen.
3. Els arguments arriben a la funció *crawl()* que s’encarrega d’ajustar el funcionament del crawler respecte als paràmetres que ha rebut. És important notar que en aquest crawler es crearan dos *pools* de threads diferents i cadascun d’aquests realitzarà funcions diferents.
4. Es llegeixen les dades que s’han extret amb l’*Extended Crawler*, en cas que no es trobin, es mostra un missatge d’error i es demana a l’usuari que executi l’*Extended Crawler* per, posteriorment, executar aquest crawler.
5. Es crea el primer *pool* de threads, creant-ne tants com l’usuari hagi especificat amb els arguments. En cas que no s’hagi especificat cap, no es crearà cap fil d’execució addicional i l’execució serà seqüencial. En cas que es creïn diversos fils d’execució, els punts del 6 fins al 8 (inclòs) es realitzaran de forma concurrent pels diferents threads.
6. Les dades llegides anteriorment de fitxer són passades a les funcions de cerca de citacions. Aquestes se n’encarreguen d’agrupar les dades relacionades amb els articles que són citats i en descarten aquelles que no hi estan disponibles (valors nuls).
7. Un cop les dades anteriors són agrupades per cadascun dels articles que s’han publicat en una determinada conferència, es realitza una petició a Semantic Scholar. Seguidament, el servidor processa la petició i retorna una resposta. En cas d’error, s’informa l’usuari i es realitza la petició pel següent article disponible. En cas contrari, les dades demanades es troben a la resposta en format JSON.
8. Les dades rebudes en el punt anterior es processen per extreure les que es consideren rellevants. En aquest cas, no s’obté informació relacionada amb l’article en si, sinó que s’obté el número DOI per tal d’usar-lo posteriorment amb OpenAlex. Finalment, aquestes dades s’emmagatzemen en una estructura de dades. Aquesta estructura és global i pot ser accedida per tots els fils d’execució que s’han creat.
9. Quan tots els threads creats acaben la seva execució, es realitza un join i es torna a la funció *crawl()*. En el cas que l’execució no hagi estat paral·lela, es retorna a

la funció *crawl()* quan l'únic thread emmagatzema les dades a l'estructura anterior.

10. A continuació, es crea un altre *pool* de threads, per tal de processar les dades emmagatzemades en l'estructura anterior. Es creen tants threads com l'usuari hagi especificat amb els arguments. En cas que no s'hagi especificat cap, no es crearà cap fil d'execució addicional i l'execució serà seqüencial. En cas que es creïn diversos fils d'execució, els punts de l'11 fins al 13 (inclòs) es realitzaran de forma concurrent pels diferents threads.
11. Les dades de l'estructura es passen a les funcions encarregades d'obtenir les dades dels articles citats (títol, any de publicació, autors, etc.).
12. Amb les dades anteriors, es realitza una petició a OpenAlex. La petició és processada pel servidor, i es rep una resposta. En cas d'error s'obté un codi d'error, s'informa l'usuari, i es prossegueix a processar les dades del següent article. En cas contrari, les dades de la resposta rebuda estaran en format JSON.
13. Un cop les dades han estat processades, i s'han obtingut les que es consideren rellevants, aquestes són emmagatzemades a una estructura de dades. Aquesta estructura és global i tots els fils d'execució que s'han creat hi tenen accés.
14. Quan tots els threads creats acaben la seva execució, es realitza un join i es torna a la funció *crawl()*. En el cas que l'execució no hagi estat paral·lela, es retorna a la funció *crawl()* quan l'únic thread emmagatzema les dades a l'estructura anterior.
15. Finalment, les dades emmagatzemades en l'estructura anterior són escrites a un fitxer en format JSON.

4. Implementació

En aquest apartat es detallarà com es van implementar totes les parts que comprenen el crawler.

Primerament, s'ha de comentar que aquest crawler ha estat implementat mitjançant el llenguatge de programació Python, i utilitzant diverses llibreries que estan disponibles de forma gratuïta. A continuació s'enumeraran les llibreries més usades juntament amb una petita explicació de l'ús que se'ls ha donat en aquest projecte.

- threading: serveix per crear tot l'entorn paral·lel de threads, juntament amb la creació de semàfors per les seccions crítiques.
- requests: permet fer diversos tipus de peticions HTTP[7], com GET, POST, PUT, etc.
- BeautifulSoup (bs4): permet treballar amb codi HTML d'una forma més senzilla, convertint els seus elements en objectes bs4. Posteriorment, es poden cridar mètodes sobre els objectes creats.
- logging: permet configurar tots els aspectes relacionats amb la sortida de log. És una forma fàcil de mostrar els missatges d'error a l'usuari.
- time: permet parar els fils d'execució durant un cert període de temps.
- re: permet implementar expressions regulars per, posteriorment, utilitzar-les per trobar patrons.
- tqdm: afegeix un component visual que permet a l'usuari veure una barra de càrrega pel terminal. Aquesta barra marca els threads que han finalitzat la seva execució.
- dotenv: permet treballar amb fitxers .env, per tal d'emmagatzemar claus i altres valors delicats.

En segon lloc, cal remarcar que en l'apartat de disseny ja s'ha explicat de forma detallada el flux de dades que segueixen els tres crawler, és per això que en els següents apartats es detallarà com s'han implementat els punts claus, i perquè s'han pres certes decisions de disseny. Malgrat tot, en les següents seccions no es mostraran els codis complets dels fitxers, tanmateix, aquests es poden trobar sencers en els annexos.

[7] HyperText Transfer Protocol

4.1. Paral·lelització Mitjançant Threads

Com s'ha comentat en apartats anteriors, el crawler dissenyat pot fer ús de la paral·lelització per tal de millorar el seu temps d'execució. Aquesta paral·lelització s'ha implementat mitjançant la llibreria threading de Python. Un altre punt a tenir en compte d'aquesta paral·lelització, és que s'han de crear fils d'execució en les tres parts del crawler o, el que seria dir el mateix, crear threads al *Base Crawler*, *Extended Crawler* i *Citations Crawler*. Per tal de reutilitzar codi, s'ha creat una classe anomenada Thread, la qual té implementat el constructor i un mètode anomenat *run().* Aquest és l'encarregat de crear els threads, posar-los en marxa i esperar a la seva finalització, per finalment, fer un join().

4.1.1. Funció run()

```
def run(self, target, args):
    first_year, last_year = args[1:3]
    data = args[0]

    total_years = last_year - first_year + 1
    average_chunk_size = total_years // self.num_threads
    remainder = total_years % self.num_threads
    start_year = first_year

    if self.num_threads == 1:
        target(*args)
    else:
        threads = []
        for i in range(self.num_threads):
            chunk_size = average_chunk_size
            if i < remainder:
                chunk_size += 1
            end_year = start_year + chunk_size - 1
            t = threading.Thread(target=target, args=(data,
                start_year, end_year))
            threads.append(t)
            t.start()
            start_year = end_year + 1

        for t in tqdm(threads):
            t.join()
```

Codi 1. Funció Thread.run()

En pertànyer a una classe, el primer paràmetre que rep el mètode és *self*, que és un paràmetre predeterminat en tots els mètodes que corresponen a una classe en Python. El segon paràmetre que rep correspon a la funció que s'ha de paral·lelitzar. El tercer paràmetre que trobem correspon als arguments (paràmetres) que s'han de passar a la funció que es paral·lelitzarà. Podem veure com aquest últim paràmetre és descompost en tres paràmetres diferents, les dades a tractar per la funció a paral·lelitzar, i els anys a tractar. Seguidament, la funció *run()* calcula quina porció de les dades haurà de tractar cadascun dels threads que es creïn. Cal notar que aquesta descomposició només s'efectua quan el nombre de fils d'execució a crear és major a u. A continuació es troba un diagrama que mostra el que es fa en aquesta descomposició de les dades.

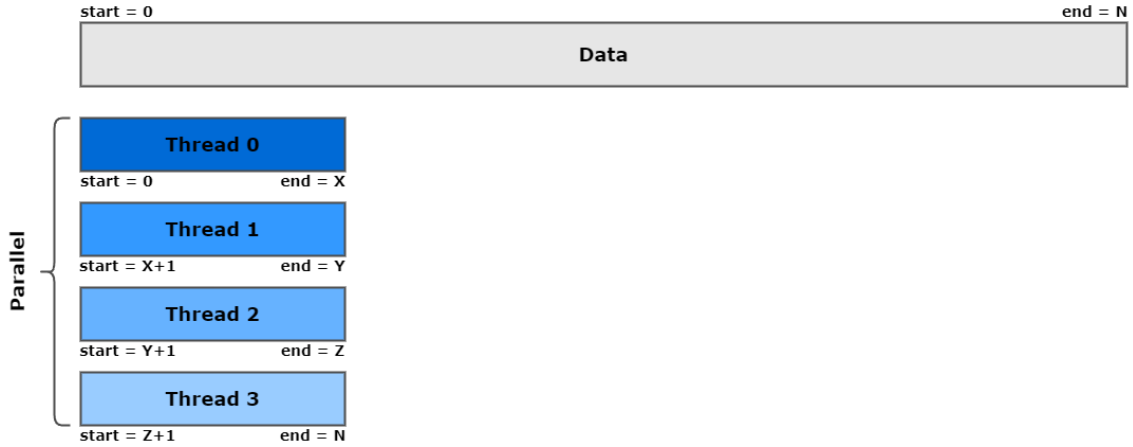


Figura 12. Diagrama de la paral·lelització de les dades

Finalment, si observem el fragment de codi 1, podem veure com el mètode run() espera que tots els fils d'execució acabin, realitzant un *join()* quan això succeeix.

4.2. Funcions de Fitxers

Al llarg de l'apartat de disseny, s'ha pogut observar com tot aquest projecte fa ús de fitxers en format JSON per tal d'anar emmagatzemant les dades. A conseqüència d'això, hi ha moltes parts del codi on necessitem llegir/carregar les dades d'un fitxer o bé, escriure les dades emmagatzemades a un fitxer. Per tal de reutilitzar codi, s'ha creat un petit fitxer de Python que conté diverses funcions relacionades amb les diverses operacions que es poden realitzar sobre un fitxer. A continuació es mostrarà el codi de les més rellevants, incloent-hi una breu explicació del que realitzen. Cal notar que una de les funcions que es mostrarà no està relacionada amb els fitxers JSON, sinó amb els fitxers *.env*. Es detallarà més en l'apartat corresponent.

4.2.1. Funció save_json()

```
def save_json(file_path, data):
    file_path = f'{file_path}.json'
    with open(file_path, 'w', encoding='utf-8') as f:
        json.dump(data, f, ensure_ascii=False, indent=4)
```

Codi 2. Funció save_json()

En el codi anterior es mostra la funció *save_json()*, que s'ha utilitzat cada cop que s'ha requerit emmagatzemar les dades a un fitxer JSON. Aquesta funció rep dos paràmetres, el primer correspon a la ruta de memòria on s'emmagatzemarà el fitxer creat, el segon correspon a les dades que es volen escriure en aquest fitxer. Com es pot veure, el codi d'aquesta funció és relativament senzill, ja que simplement s'obre el fitxer que es vol escriure (si no existeix el crea), i seguidament s'escriuen les dades en format JSON.

4.2.2. Funció exists_file()

```
def exists_file(file_path):
    return os.path.exists(f'{file_path}.json')
```

Codi 3. Funció exists_file()

Aquesta funció té com a objectiu retornar un valor booleà depenent de si el fitxer cercat existeix en la ruta especificada o no. Si el fitxer existeix es retorna *True*, en cas contrari, es retorna *False*.

4.2.3. Funció load_json()

```
def load_json(file_path):
    if exists_file(file_path):
        file_path = f'{file_path}.json'
        with open(file_path, 'r', encoding='utf-8') as f:
            data = json.load(f)
            return data
    else:
        return None
```

Codi 4. Funció load_json()

El codi anterior mostra el funcionament de la funció *load_json(),* que s'encarrega de carregar les dades que es troben emmagatzemades en un fitxer format JSON. Com es pot observar, aquesta funció fa ús de la funció mostrada en el codi 3, per tal de determinar si aquest fitxer existeix o no. En cas afirmatiu, s'obre el fitxer i es llegeixen les dades. En cas contrari, es retorna un valor nul.

4.2.4. Funció api_key_in_env()

```
def api_key_in_env():
    load_dotenv()
    api_key = os.getenv("S2_API_KEY", None)
    return api_key
```

Codi 5. Funció api_key_in_env() per carregar una clau d'un fitxer .env

Aquesta funció s'utilitza per comprovar si existeix un valor anomenat api_key en un fitxer d'entorn (*.env*). Aquests fitxers són utilitzats per emmagatzemar valors delicats, com ara claus personals. En relació amb aquest projecte, s'utilitza una clau per l'API de Semantic Scholar (es detallarà en apartats posteriors). Per tal de comprovar si existeix aquesta clau s'utilitza la funció anterior. Primerament, es carrega el fitxer d'entorn, i s'intenta obtenir el valor anomenat S2_API_KEY, en cas que aquesta clau existeixi, es retornarà el seu valor. En cas contrari es retornarà un valor nul.

4.3. Implementació Base Crawler

En la següent secció s'explicarà la implementació seguida per la classe *BaseCrawler*. Es mostraran alguns dels mètodes que aquesta classe implementa, així com una breu explicació de la funció que aquests realitzen.

4.3.1. Mètodes principals

El primer mètode que trobem s'anomena *crawl()*, aquest és el que controla de forma general tot el procés de crawling. El codi que es troba a continuació pertany a aquest mètode.

```
def crawl(self):
    initial_time = time.time()
    first_year, last_year = self.years
    for conf in self.conferences:
        global data_per_year
        data_per_year = {}

        threads = thread.Thread(self.num_threads)
        threads.run(self._search, (conf, first_year, last_year))
        file.save_json(f"{self.output_dir}/{conf}_basic_data",\
        data_per_year)

    final_time = time.time()
```

```
    minutes = (final_time - initial_time) / 60
    print(f"(BASE) - Done in {minutes:.3f} minutes")
```

Codi 6. Funció crawl() del Base Crawler

En primer lloc, podem veure com s'inicialitzen algunes variables necessàries per al procés de crawling, com el rang d'anys dels quals s'ha d'obtenir informació. En segon lloc, es troba un bucle for, el qual va iterant sobre les diferents conferències que l'usuari ha introduït com argument en la comanda. Dins d'aquest bucle podem observar com s'inicialitza l'estructura de dades global *data_per_year*, que serà l'estructura en la qual els diferents threads escriuran les dades que extreuen. En tercer lloc, trobem la inicialització dels threads. Es pot observar com es crea una instància de la classe Thread, la qual s'ha detallat en apartats anteriors. En el constructor de la instància es passen per paràmetre el nombre de threads que es volen crear, i seguidament, es crida al mètode *run()*. Com a paràmetres es passen els anys i el nom de la conferència. Aquest últim paràmetre serveix per poder crear l'URL des de la qual es començarà a realitzar peticions a DBLP. Conjuntament amb aquests paràmetres, també s'ha de passar el nom de la funció que es vol paral·lelitzar, en aquest cas la funció *search()*.

Un cop els fils d'execució creats han acabat la seva execució, les dades obtingudes s'hauran emmagatzemat en l'estructura *data_per_yer*, i per emmagatzemar aquestes dades a fitxer simplement s'haurà d'escriure l'estructura a fitxer. Per fer-ho s'usa la funció *save_json()* explicada en l'apartat de funcions de fitxers. Per acabar, la funció *crawl()* escriu per consola el temps que ha tardat aquest procés de crawling de les dades.

El segon mètode important que trobem en la classe *BaseCrawler* és el mètode *search()*. Com s'acaba d'explicar, aquest és el mètode que és paral·lelitzat amb els threads. Tot seguit es troba el codi corresponent a aquest mètode.

```
def _search(self, conf, first_year, last_year):
    links = self._get_links(conf)
    valid_links = []
    for link in links:
        if self._filter_dblp_links(conf, link) and any(str(year) in\
        link for year in range(first_year, last_year + 1)):
            valid_links.append(link)

    for link in valid_links:
        pub_list_raw = self._get_pub_list(link)
```

```
        for pub in pub_list_raw:
            article_items = pub.find_all('li', {'itemtype': \
            'http://schema.org/ScholarlyArticle'})
            header = pub.find_previous('h2')
            if self._filter_section(header.text):
                continue
            for child in article_items:
                pub_data = self._get_dblp_paper_data(child)
                if pub_data is None:
                    continue
                if pub_data['Year'] not in data_per_year:
                    self.semaphore.acquire()
                    data_per_year[pub_data['Year']] = []
                    self.semaphore.release()

                self.semaphore.acquire()
                data_per_year[pub_data['Year']].append(pub_data)
                self.semaphore.release()
```

Codi 7. Funció search() del Base Crawler

Si es mira el que va fent la funció *search()* de forma seqüencial, s'observarà que és l'encarregada d'anar indicant les peticions que s'han de fer, i posteriorment, processa les dades d'interès que s'han extret de la resposta rebuda. Els següents passos expliquen de forma breu què és el que realitza la funció *search()*.

1. A partir del nom de la conferència s'obté la seed URL, i es realitzen la petició corresponent. Com a resultat es reben els links que pertanyen a cadascun dels anys que s'ha dut a terme la conferència.
2. Es filtren els links anteriors per tal d'obtenir els links corresponents als anys dels quals volem obtenir les dades.
3. S'itera sobre els links filtrats i es van obtenint les dades dels articles publicats en cadascun dels anys.
4. Es filtren els articles, per eliminar els que no són articles de recerca, per exemple els que s'anomenen Proceedings, que serien articles que donen la benvinguda a la conferència, però que no corresponen a material d'investigació.
5. Per cadascun dels articles s'extreu tota la informació que es vol obtenir, i es guarda en l'estructura *data_per_year*. Com tots els threads han de poder escriure i llegir

de l'estructura a la vegada, s'ha d'assegurar una coherència de dades, per tant, sorgeix una secció crítica. Per protegir aquesta secció crítica s'usen semàfors.

Si ens fixem en detall, podem veure que el codi de les peticions en si no es troba directament en el mètode *search(),* ja que en aquest cas el mètode resultant tindria centenars de línies de codi i no seria llegible. Seguint les bones pràctiques de la programació orientada a objectes, s'han descompost els punts principals en mètodes més petits, d'aquesta forma és molt més llegible i fàcil de depurar.

A més, fer aquesta descomposició permet que algunes de les classes filles aprofitin els mètodes de peticions a les API de la classe pare.

A manera d'exemple, el següent fragment de codi mostra com es realitzen les peticions. El funcionament és pràcticament igual en tots els casos, només canvia el link al qual es fa la petició. En aquest cas, es tracta d'una petició a DBLP, on es pot veure com s'usa la llibreria BeautifulSoup per tractar els elements HTML que es reben com a resposta.

```
def _get_pub_list(self, link):
    resp = requests.get(link, timeout=10)
    soup = BeautifulSoup(resp.content, features="lxml")
    return soup.findAll("ul", attrs={"class": "publ-list"})
```

Codi 8. Exemple de codi d'una petició

4.4. Implementació Extended Crawler

La implementació de l'*Extended Crawler* és similar a la del *Base Crawler*, encara que les dades que s'obtenen són diferents, el funcionament general és pràcticament idèntic.

Cal tenir en compte que la classe *ExtendedCrawler* hereta de la classe *BaseCrawler*, per tant, aquesta classe té accés a tots els atributs de *Base Crawler*, així com els mètodes.

4.4.1. Mètodes Principals

D'igual forma que trobem en el *Base Crawler,* el primer mètode que trobem és el *crawl().* El codi següent mostra com s'ha implementat.

```
def crawl(self):
    initial_time = time.time()
    first_year, last_year = self.years
```

```
    for conf in self.conferences:
        global data_per_year
        data_per_year = {}

        data_dir = f"./data/base_crawler_data/{conf}_basic_data"
        if file.exists_file(data_dir):
            basic_data = file.load_json(data_dir)
        else:
            sys.exit(f"Error: The basic data for the conference \
            {conf} does not exist. Please run the base crawler first.")

        for year in range(first_year, last_year + 1):
            if not file.year_exists_in_file(year, data_per_year):
               logging.info(f"(EXTENDED) - {year} data not found.")

        threads = thread.Thread(self.num_threads)
        threads.run(self._get_paper_data, (basic_data, \
        first_year, last_year))

        file.save_json(f"{self.output_dir}/{conf}_extended_data",\
        data_per_year)

    final_time = time.time()
    minutes = (final_time - initial_time) / 60
    print(f"(EXTENDED) - Done in {minutes:.3f} minutes")
```

Codi 9. Funció crawl() de l'Extended Crawler

El funcionament d'aquesta funció és pràcticament idèntic al que es troba implementa en la classe *BaseCrawler*, tenint en compte que en aquest cas la funció a paral·lelitzar és una altra. La principal diferència entre aquest *crawl()* i el de la classe pare, és que aquest ha de realitzar una comprovació abans de crear tot l'entorn dels threads. S'ha de comprovar que existeixi el fitxer generat pel *Base Crawler* per les mateixes dades que es vol realitzar el crawling actual, ja que aquestes s'usen per aconseguir-ne de noves. També es comprova si els anys dels quals es vol obtenir les dades es troben al fitxer, el cas que algun dels anys no es trobi s'escriurà per fitxer de log indicant-ho. El crawler no pararà la seva execució, perquè si la resta d'anys estan disponibles, podrà obtenir les dades corresponents.

El segon mètode important que trobem en aquest crawler, de forma similar que en la classe pare, és el mètode que es paral·lelitza. A continuació es troba el codi corresponent a aquest.

```
def _get_paper_data(self, data, start_year, end_year):
    for year in range(start_year, end_year + 1):
        paper_data = []
        if str(year) not in data:
            continue
        for elem in data[str(year)]:
            paper_title = elem['Title']
            paper_doi_num = elem['DOI Number']
            paper_pub_year = elem['Year']
            paper_openalex_link = elem['OpenAlex Link']
            authors_institutions = elem['Authors and Institutions']
            referenced_works = elem['OpenAlex Referenced Works']

            s2_data = self._get_s2_paper_data(paper_title,\
            paper_doi_num)
            paper_id, abstract, tldr, citations_s2, \
            doi_s2 = s2_data if s2_data is not None else \
            (None, None, None, None, None)

            if doi_s2 is not None and paper_openalex_link is None:
                openalex_data = super()._get_openalex_data \
                (f"https://api.openalex.org/works/https:// \
                 doi.org/{doi_s2}")
                paper_doi_num, authors_institutions, \
                referenced_works = openalex_data \
                if openalex_data is not None else (None, None, None)

            paper_data.append ({
                'Title': paper_title,
                'Year': paper_pub_year,
                'DOI Number': paper_doi_num,
                'OpenAlex Link': paper_openalex_link,
                'S2 Paper ID': paper_id,
                'Authors and Institutions': authors_institutions,
                'OpenAlex Referenced Works': referenced_works,
                'Citations S2': citations_s2,
                'Abstract': abstract,
```

```
                'TLDR': tldr
            })

        self.semaphore.acquire()
        data_per_year[year] = paper_data
        self.semaphore.release()
```

Codi 10. Funció _get_paper_data de l'Extended Crawler

D'igual forma que en el *Base Crawler*, aquesta funció és la que coordina totes les peticions que s'han de realitzar, i posteriorment, emmagatzema les dades. Tal com es pot observar, la implementació és molt senzilla. La part més complexa la trobem en el moment d'escriptura en l'estructura de dades. D'igual manera que en el *Base Crawler*, l'accés a aquesta estructura representa una secció crítica, per tant, s'ha protegit mitjançant semàfors.

Finalment, és interessant observar com s'han implementat les peticions a l'API de Semantic Scholar, i, tot i que el funcionament és idèntic al mostrat en el codi 10, amb una petita diferència. Aquesta diferència té a veure amb el fet que Semantic Scholar ofereix la possibilitat d'usar una clau per incrementar el límit de peticions. Si l'usuari disposa d'aquesta clau, la qual ha d'estar emmagatzemada en un fitxer de tipus *.env*, aquesta és llegida del fitxer i s'utilitza per realitzar les peticions. Com es pot donar el cas en què l'usuari no disposa d'aquesta clau, el crawler també es pot usar sense, però s'ha de tenir en compte que aquest fet presenta limitacions. Tornant a la implementació de les peticions, bàsicament, depenent de si l'usuari disposa d'aquesta clau o no, la capçalera de la petició que s'enviarà a Semantic Scholar serà diferent. A continuació es mostra el codi referent al mètode implementat per controlar els dos casos diferents. Cal aclarir que aquest mètode és cridat cada cop que es vol realitzar una petició a Semantic Scholar.

```
    def _make_request_func(self, url, params, api_key):
        if api_key is not None:
            response = requests.get(url, params=params, headers={'x-
            api-key': api_key})
        else:
            response = requests.get(url, params=params)
        return response
```

Codi 11. Mètode make_request_func() de l'Extended crawler

4.5. Implementació Citations Crawler

Aquest crawler, és sens dubte, el més complex pel que fa a la implementació. Això es deu al gran volum de peticions que es realitzen, juntament amb el gran volum de dades que es tracten. Així doncs, aquest és el crawler amb un temps d'execució més elevat respecte als altres dos que s'han vist.

D'altra banda, un dels problemes principals amb els quals es troba aquest crawler, és que en realitzar tantes peticions, els errors que sorgeixen per les limitacions imposades per les API ocorren més sovint del que es desitjaria, encara que s'ha intentat limitar-los el màxim possible.

En conclusió, s'ha intentat implementar aquest crawler d'una forma en la qual es minimitzin errors per límits de peticions, però a la vegada aquest pugui ser eficient i fiable.

4.5.1. Mètodes Principals

D'igual manera que en els altres dos crawlers, hi trobem el mètode *crawl(),* que és l'encarregat de coordinar tot el procés de crawling. Tot i això, aquest és una mica diferent respecte als que es troben en les implementacions dels crawlers vistos amb anterioritat. A continuació es mostra el codi corresponent a aquest.

```
def crawl(self):
    initial_time = time.time()
    first_year, last_year = self.years
    for conf in self.conferences:
        global papers_data
        papers_data = []
        global all_citation_data
        all_citation_data = {}

        data_dir=f"./data/extended_crawler_data/{conf}_extended_data"
        if file.exists_file(data_dir):
           extended_data = file.load_json(data_dir)
        else:
           sys.exit(f"Error: The extended data for the conference\
            {conf} does not exist. Please run the extended crawler\
            first.")

        self._get_all_paper_data(self.conferences)
```

```
        threads = thread.Thread(self.num_threads)
        threads.run(self._search_citations_data, (extended_data,\
        first_year, last_year))

        threads.run(self._get_citation_data, (papers_data, 0,\
        len(papers_data)))

        file.save_json(f"{self.output_dir}/{conf}_extended_data",\
        all_citation_data)

    final_time = time.time()
    minutes = (final_time - initial_time) / 60
    print(f"(CITATIONS) - Done in {minutes:.3f} minutes")
```

Codi 12. Funció crawl() del Citations Crawler

Encara que a simple vista sembla similar al mètode *crawl()* que trobem en l'*Extended Crawler*, hi ha una petita diferència, i és que aquest crawler crea dos pools de therads com s'ha comentat en l'apartat de disseny. Això ho podem veure en les línies que es crea la instància de la classe thread. Podem veure com seguit d'aquesta línia hi trobem dues línies que criden al mètode *run()*. Gràcies a com s'ha implementat la classe Thread, es pot reaprofitar la instància ja creada per fer que es creïn dos pools diferents, això sí, tenint en compte que aquesta creació és seqüencial. Això vol dir que primer es creen els primers threads, i un cop tots aquests han finalitzat la seva execució, llavors es creen els altres fils d'execució.

La dedició d'implementar dos pools diferents es va realitzar perquè a causa del nombre tan elevat de peticions, si no es paral·lelitzaven tant les peticions a Semàntic Scholar com les peticions a OpenAlex, el temps d'execució augmentava de forma considerable. Per tal de trobar la implementació paral·lela més eficient es van provar diferents implementacions, i finalment aquesta va ser la que va donar millors resultats.

Un altre dels mètodes importants que hi trobem en aquest crawler és el mètode *_search_citations_data()*. Aquest se n'encarrega de filtrar els articles citats per obtenir les dades amb les quals es realitzarà la petició posterior a Semantic Scholar. El fragment de codi que es troba a continuació mostra aquest mètode.

```
def _search_citations_data(self, data, start_year, end_year):
    papers = {}
    for year in range(start_year, end_year + 1):

        # obtain all the papers ids from the citations
        try:
            for paper in data.get(str(year), []):
                citations = paper.get("Citations S2", [])

                if citations:
                    paper_ids = [citation["paperId"] for citation in\
                    citations if citation.get("paperId")]
                    papers[paper["Title"]] = paper_ids
        except KeyError:
            pass

    self._batch_request_s2(papers)
```

Codi 13. Mètode _search_citations_data() del Citations Crawler

Es pot veure com la implementació d'aquest mètode és molt senzilla, ja que simplement es basa en condicionals i bucles, però la part interessant és la que es troba a l'última línia. En aquesta hi podem veure una crida a un mètode de la classe anomenat *_batch_request_s2()*. Aquest és el conté la lògica de les peticions a Semantic Scholar. El codi corresponent a aquest mètode es troba a continuació.

```
def _batch_request_s2(self, papers):
    url_s2 = "https://api.semanticscholar.org/graph/v1/paper/batch"
    for title, citations in papers.items():
        responses = []
        citations_len = len(citations)
        num_iterations = citations_len // 500
        rest = num_iterations + 1 if citations_len % 500 > 0 else
        num_iterations
        for j in range(rest):
            ini = j * 500
            fin = min((j + 1) * 500, citations_len)
            c = citations[ini:fin]
            r = requests.post(url_s2,
                params={'fields':
                'title,year,venue,externalIds,authors.name'},
                json={"ids": c})
```

```
            time.sleep(1.25)
            if r.status_code == 200:
                responses.append(r.json())
            else:
                logging.error(f"(CITATIONS) - {r.status_code} in
                request for paper {title}")

        self.semaphore_s2.acquire()
        papers_data.append({"Title": title, "Response": responses})
        self.semaphore_s2.release()
```

Codi 14. Mètode _batch_request_s2 del Citations Crawler

La petició que es realitza a Semàntic Scholar és un tipus especial, ja que s'anomena petició *batch*. Aquest tipus de petició permet obtenir les dades de fins a cinc-cents articles diferents fent només una petició a l'API de Semàntic Scholar.

Així doncs, el primer que es fa en aquest mètode és dividir articles dels quals es voldrà obtenir informació. Si hi ha més de cinc-cents, es divideixen usant el mètode de partició de dades que s'usa en la creació dels fils d'execució, i que s'ha explicat anteriorment. El segon que fa és posar els paràmetres necessaris a la petició. Cal fixar-se que aquesta petició no és del tipus GET, la qual és la més comuna, i la que s'ha usat en la resta de peticions fins ara. Aquesta petició en ser un tipus especial usa el tipus POST. Un cop s'han configurat tots els camps de la petició, aquesta es realitza. La resposta rebuda en la petició s'emmagatzema en una estructura, juntament amb el títol de l'article que realitza totes aquestes citacions, és a dir, un article de la conferència que s'ha obtingut anteriorment amb l'*Extended Crawler*. Aquesta resposta serà processada posteriorment pels altres threads d'execució que es creïn.

Un cop s'han aconseguit totes les respostes relacionades amb les peticions a Semantic Scholar, els fils d'execució encarregats d'aquestes finalitzen, i seguidament es posen en marxa els següents threads, que se n'encarregaran de realitzar les peticions a OpenAlex de forma paral·lela. A continuació es mostra el codi de la funció que es paral·lelitza en aquest segon pool de threads.

```
    def _get_citation_data(self, data, start, end):
        for elem in data[start:end]:
            cited_data = []
            main_paper_title = elem.get("Title", None)
            response = elem.get("Response", None)
```

```
            if response == []: continue

            for cited_paper in response:
                for c in cited_paper:
                    data = self._get_cited_paper_data(c)
                    cited_data.append(data)
            self.semaphore_oa.acquire()
            all_citation_data[main_paper_title] = cited_data
            self.semaphore_oa.release()
```

Codi 15. Mètode _get_citation_data del Citations Crawler

De nou, la implementació d'aquesta funció és molt simple, ja que delega la major part del treball a altres funcions, i són aquestes les que realitzen les peticions i processen les dades rebudes pel servidor. De forma general, el que fa és anar agafant totes les respostes de Semantic Scholar que han estat emmagatzemades, i per cadascuna de les respostes passa les dades com a paràmetre a un altre mètode el qual extreurà les dades necessàries per realitzar les peticions a l'API d'OpenAlex. El mètode encarregat de coordinar les peticions s'anomena *_get_cited_paper_data()*. Finalment, aquest retorna totes les dades que s'escriuran posteriorment al fitxer JSON. Cal notar que aquestes dades, d'igual manera que en la resta de crawlers, primer són escrites a una estructura, la qual està protegida per semàfors en tractar-se d'una secció crítica.

4.6. Peticions a Semantic Scholar

Com s'ha comentat en apartats anteriors, el principal problema de Semantic Scholar és el límit de peticions per segon que es poden realitzar a l'API. Aquest fet va complicar de forma significativa l'extracció de les dades, ja que hi va haver un moment en el qual l'empresa propietària de Semantic Scholar va veure el gran tràfic de peticions que generaven els diferents crawlers, i va decidir reduir encara més el límit de peticions, i fins i tot va bloquejar l'adreça des de la qual es realitzava el crawling per intentar reduir el tràfic generat als servidors. Això, evidentment, va alentir molt el procés d'extracció.

Per poder solucionar aquest petit inconvenient temporal, es van crear màquines virtuals localitzades arreu del món. Aquestes VM[8] contenien el codi font dels diferents crawlers, per la qual cosa es va començar a realitzar el crawling de dades des de cadascuna

[8] Virtual Machine

de les diferents VM. Com les peticions arribaven a Semantic Scholar des de diferents parts del món, el servidor no podia identificar que aquestes es tractaven de la mateixa persona, per la qual cosa no les podia bloquejar.

La creació d'aquestes màquines virtuals es va realitzant mitjançant el servei de GitHub Codespaces, el qual permet crear-les de forma senzilla a partir d'un repositori de GitHub. Aquest fet va facilitar en gran manera l'extracció de les dades mitjançant les VM, ja que l'entorn que es creava dins de la VM era un entorn de desenvolupament Python usant Visual Studio Code, per la qual cosa l'execució es realitzava d'igual manera que es faria en una màquina local.

Finalment, es van crear màquines virtuals en les regions de US East, US West, Southeast Asia i Australia. Aquestes VM van ser utilitzades per realitzar el crawling de les dades restants.

5. Execució

Aquest projecte es troba localitzat en un repositori de GitHub. Dins d'aquest, hi trobem diversos directoris i fitxers, els quals fan possible l'execució dels diferents crawlers, així com l'emmagatzemament de les dades que aquests obtenen. Primerament, hi trobem el directori "auxiliar". Dins d'aquest s'hi troben els fitxers que contenen mètodes i classes auxiliars pels crawlers. En segon lloc, trobem el directori "cli" que conté la lògica i processament dels arguments que l'usuari introdueix per consola. El tercer directori que trobem és "crawler", aquest conté tota la lògica dels crawlers. En quart lloc, trobem el directori "data" que és on es guardaran els fitxers de dades de forma predeterminada. I, finalment, trobem el directori "log" que conté la configuració del fitxer de log, així com el mateix fitxer.

Pel que fa als fitxers, els més importants són "requirements.txt" que conté totes les llibreries necessàries per executar el projecte i "use_crawler.py" que és el codi principal que posa en marxa tot el crawler. A continuació es mostraran els passos que s'han de seguir per executar els crawlers.

1. Descarregar o clonar el repositori de GitHub.
2. Obrir un terminal amb Python en la ubicació on s'hagi descarregat el repositori.
3. Executar la següent comanda per instal·lar les llibreries necessàries: "pip install -r requirements.txt"
4. En el mateix terminal, executar la comanda necessària per començar el crawler. A continuació es troba el format que ha de tenir la comanda "python ./use_crawler.py --c {conference} --y {[from to] / year} [--extended] [--citations] [--t int] [--no_key]". Per exemple, "python ./use_crawler.py --c nsdi --y 2023"

Dins del mateix repositori de GitHub es troba un fitxer anomenat "README.md". En el seu interior s'hi troba informació detallada de tot el projecte, així com dels arguments que es poden usar en la comanda del crawler, i que s'hi pot fer amb cadascun d'aquests.

A manera de resum, a continuació es llisten els arguments que accepta la comanda, així com una breu explicació del que fan o modifiquen.

- --c: nom de la conferència o conferències a estudiar
- --y: any o anys dels quals es vol obtenir informació

- --extended: flag que indica que es vol utilitzar l'*Extended Crawler*
- --t: nombre de fils d'execució que es volen crear
- --no_key: flag que indica que no es vol usar una clau personalitzada per l'API de Semantic Scholar
- --citations: flag que indica que es vol utilitzar el *Citations Crawler*

Per concloure, cal aclarir que el nom de les conferències que se li ha de passar al crawler per paràmetre no és a l'atzar, és a dir, no se'n pot ficar un de qualsevol. Per saber quin nom correspon a cada conferència, primerament s'ha de buscar la conferència a DBLP i després s'ha d'observar el nom que hi ha en l'URL de la pàgina principal de la conferència. El nom que hi hagi correspondrà al nom identificatiu de la conferència dins del crawler, i cada cop que es faci referència a aquesta conferència, s'usarà aquest nom. Per a més informació d'aquest tema, es recomana consultar el fitxer "README.md" del repositori.

6. Estudi de les Dades

Tots els apartats anteriors detallen com s'ha dissenyat i implementat el crawler, però no ens hem d'oblidar que l'objectiu principal del crawler era extreure les dades relacionades amb els diferents articles publicats en diverses conferències, per posteriorment, estudiar-les. Dit d'una altra manera, l'objectiu principal d'aquest projecte és estudiar les dades obtingudes per determinar si la premissa realitzada en un inici es podria considerar certa, i el crawler seria un element que ens permet apropar-nos a aquest objectiu.

D'aquesta forma, en els següents apartats es parlarà sobre les conferències estudiades, els anys dels quals s'han extret les dades, a més d'explicar com s'han tractat i processat les dades pel posterior anàlisi. Finalment, mostraran alguns dels resultats obtinguts fins al moment.

6.1. Conferències Estudiades

Per tal de tenir una representació tan diversa com sigui possible, vam decidir estudiar els articles d'algunes de les conferències de ciències de la computació més conegudes internacionalment. D'aquesta forma, es van acabant seleccionant deu conferències diferents. A continuació s'enumeren.

- Symposium on Networked Systems Design and Implementation (NSDI/nsdi)
- ACM Symposium on Cloud Computing (SoCC/cloud)
- International Middleware Conference (Middleware/middleware)
- European Conference on Computer Systems (EuroSys/eurosys)
- IEEE International Conference on Distributed Computing Systems (ICDCS/icdcs)
- IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID/ccgrid)
- European Confeence on Parallel Processing (Euro-Par/europar)
- ACM SIGCOMM Conference (SIGCOMM/sigcomm)
- IEEE International Conference on Cloud Computiong (IEEE Cloud/IEEEcloud)
- International Conference on Cloud Engineering (IC2E/ic2e)

Com s'observa, al costat del nom complet hi trobem dos acrònims entre parèntesis. El primer correspon a l'acrònim oficial de la conferència, i el que s'usarà en les següents seccions per referir-se a les conferències. El segon acrònim és el que s'hauria de posar en el crawler si es volgués extreure les dades d'una determinada conferència. Aquest acrònim és definit per DBLP tal com s'ha explicat en l'apartat d'execució.

Per acabar, cal comentar que per cadascuna d'aquestes conferències s'han extret dades des del 2012 fins al 2023, ambdós inclosos.

6.1.1. Dades adicionals

Encara que gràcies al crawler s'han pogut extreure totes les dades relacionades amb els articles de les conferències, hi ha un tipus de dades que no es poden obtenir mitjançant el crawler. Aquestes dades no tenen relació amb els articles en si, però sí que tenen relació amb l'organització de la conferència. Les dades en qüestió fan referència als membres del Program Committee de cadascuna de les conferències. Aquestes ens interessen, ja que podem mesurar el percentatge de membres d'aquest comitè que també publiquen articles a la vegada que exerceixen de jurat. Aquesta mesura ens pot indicar si el fet de pertànyer a la conferència de forma interna repercuteix al fet de tenir més possibilitats de publicar un article en la conferència.

Malauradament, l'obtenció d'aquestes dades és bastant més complicada que les que s'han vist fins ara. Això es deu al fet que no es pot crear un crawler, almenys senzill, que sigui capaç d'extreure les dades. El principal problema és que aquestes dades només es troben en format PDF o en alguns casos són les mateixes conferències les que les mostren en la seva pàgina web. Extreure aquestes dades de les pàgines web és complicat pel fet que cadascuna té un format diferent, i si recorden, el crawling es basa en obtenir les dades mitjançant el codi HTML. Pàgines web diferents representa un codi HTML diferent, la qual cosa implica un crawler diferent.

Així doncs, aquestes dades s'han hagut d'extreure de forma manual de les diverses fonts. S'ha de tenir en compte que hi poden haver errors en les dades, i que fins i tot, hi poden haver-hi anys de les conferències on no hi hagi dades relacionades amb aquest comitè.

6.2. Processament de les Dades

En aquesta secció es detallarà com s'han processat les dades de les conferències obtingudes amb el crawler pel seu posterior estudi.

Primerament, cal recordar que les dades estan emmagatzemades en fitxers JSON, i, encara que es pot treballar de forma directa amb les dades dels fitxers, s'ha optat per emmagatzemar-les en una base de dades relacional. Tanmateix, en aquesta base de dades no es guarden totes les dades, sinó només les que estan relacionades amb els articles publicats en les conferències, així com els membres del Program Committee.

Per totes les dades relacionades amb els articles citats s'ha optat per usar un tipus de base de dades en concret, en aquest cas una Graph Data Base, que com el seu nom indica, representa les dades en grafs.

6.2.1. Construcció de la Base de Dades Relacional

La implementació de la base de dades relacional va ser portada a terme amb l'eina SQLlite3, la qual és una llibreria de Pyhton que permet crear bases de dades relacionals SQL de forma local.

L'estructura de la base de dades dissenyada per poder emmagatzemar totes les dades dels articles és bastant simple. Primerament, cal dir que cada conferència té la seva base de dades pròpia, fet que permet que sigui més senzill extreure les dades i estudiar-les per cadascuna de les conferències. A més a més, com a decisió de disseny, es va optar per crear taules separades per les dades de cadascun dels anys recopilats, d'aquesta manera el seu posterior estudi per anys és molt més senzill.

A continuació es mostren les taules que es creen en cadascuna de les bases de dades, juntament amb les propietats que aquestes tenen.

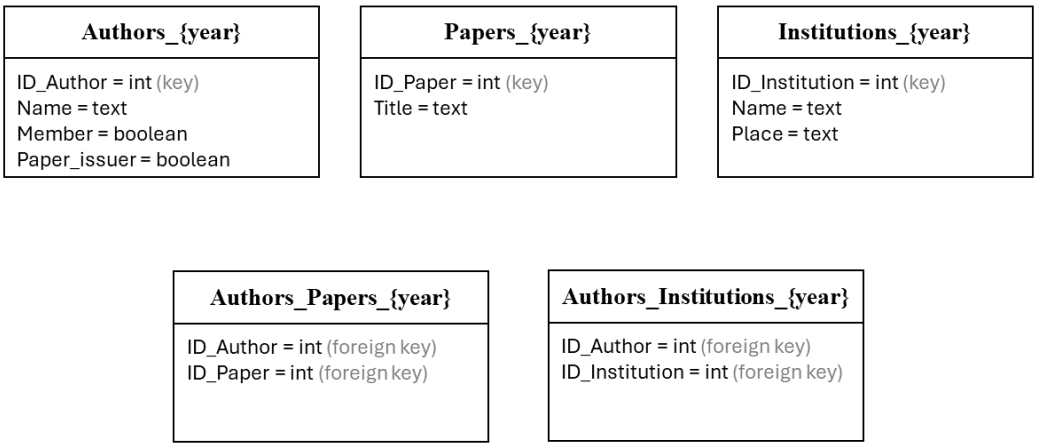


Figura 13. Taules de la base de dades relacional

6.2.2. Construcció de la Graph DB

Com s'ha comentat, les dades relacionades amb les citacions s'han emmagatzemat formant grafs. Aquest tipus de bases de dades són molt útils per dades que presenten moltes relacions entre elles, ja que permeten representar aquestes relacions mitjançant nodes i arestes en un graf. Els nodes del graf representen entitats, mentre que les arestes representen les relacions entre aquestes entitats.

En les dades de les citacions, si pensem en les relacions que hi ha, podem observar com la relació més important que trobem és la relació cita, amb la qual s'indica que un article ha citat a un altre. També podem veure com apareix l'entitat "article", per la qual cosa, representar aquestes dades amb una base de dades de graf presenta un gran avantatge davant de representar-la com una base de dades relacional o qualsevol altre model.

Per realitzar la implementació d'aquest tipus de base de dades s'ha usat Neo4j, una de les plataformes més conegudes en aquest camp. Neo4j és una base de dades de graf que està dissenyada per emmagatzemar, gestionar i analitzar aquest tipus de dades. El model que utilitza Neo4j està especialment adequat per representar relacions complexes i interconnectades.

A més a més, Neo4j treballa amb transaccions de tipus ACID[9], assegurant que les operacions en la base de dades siguin segures i consistents.

Figura 14. Logotip de Neo4j

Neo4j permet crear nodes que representen entitats, relacions que representen connexions entre els nodes i propietats que són parells clau-valor. Tant els nodes com les relacions poden tenir propietats per emmagatzemar dades addicionals.

A més a més, Neo4j té el seu propi llenguatge per les consultes a la base de dades, anomenat Cypher. Està especialment dissenyat per treballar amb bases de dades de graf, la qual cosa el fa molt intuïtiu. Cypher permet realitzar consultes complexes de manera senzilla i expressiva.

```
MATCH (p:Paper{title:"Title1"})-[:CITES]->(cited:Paper)
RETURN cited.title
```

Codi 16. Exemple d'una consulta Cypher

En el codi anterior es mostra un exemple d'una consulta Cypher, en la qual obtindríem tots els títols dels articles que són citats per l'article que té el títol "Title1".

Ara que ja es tenen nocions del que és una base de dades de graf, i què és Neo4j, s'explicarà com s'ha dissenyat la base de dades per emmagatzemar totes les dades relacionades amb les citacions dels articles.

En primer lloc, es va dissenyar el model de dades, el qual representa les entitats, relacions i les seves respectives propietats. Per l'estudi d'aquestes dades, ens interessa saber les institucions que estan relacionades amb cadascun dels articles i alhora, ens interessa el país en el qual es localitza la institució. Aquestes dades ens resulten les més interessants, ja que volem poder determinar si els articles d'un determinat país o continent segueixen un patró citant a articles del mateix país o continent, o en cas contrari, citant articles pertanyents a altres països. Així doncs, dades com els autors no s'han representat com a entitats, sinó que s'han representat com a propietats dins de l'entitat "Paper". En

[9] Atomicity, Consistency, Isolation and Durability

la figura que es troba a continuació es mostra el model de dades utilitzat per construir la base de dades final.

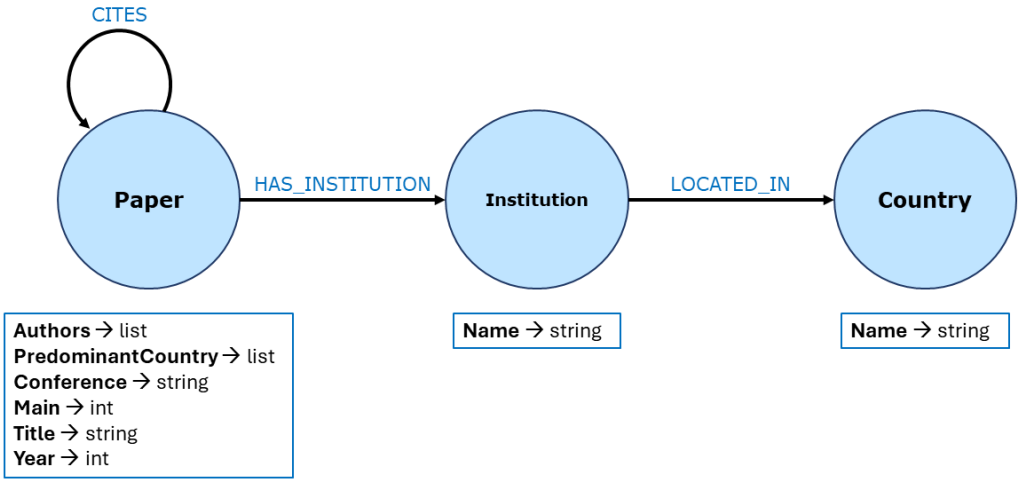


Figura 15. Disseny base de dades Neo4j

En segon lloc, es va configurar tot l'entorn Neo4j per poder crear la base de dades. Per facilitat d'ús i eficiència en les consultes, es va crear una base de dades diferent per cada conferència. Configurar l'entorn Neo4j és molt senzill, sobretot si s'usa l'aplicació d'escriptori que ells mateixos ofereixen. El més complicat d'aquesta fase de configuració és usar el driver oficial de Neo4j per poder accedir a les bases de dades des d'un fitxer Python. Afortunadament, la mateixa empresa Neo4j disposa de manuals i exemples que expliquen de forma detallada com fer-ho. El següent codi mostra un exemple de connexió de Python a la base de dades de Neo4j mitjançant el driver.

```
from neo4j import GraphDatabase

graph_db = GraphDatabase.driver("bolt://localhost:7687",
auth=("neo4j","passwordDB"))
session = graph_db.session()
```

Codi 17. Connexió a la base de dades de Neo4j mitjançant el driver

Com es pot observar en el codi anterior, primerament s'importa la classe GraphDatabase, seguidament es crida al mètode driver d'aquesta classe, al qual li has de passar l'URI on es troba la base de dades, en aquest cas en el port *bolt* de la màquina local, i el parell clau-valor que correspon a l'usuari de la base de dades i la contrasenya que se li ha posat. L'usuari normalment sol ser "neo4j", però es pot modificar en crear la base de dades.

Per finalitzar, un cop la base de dades està configurada, es pot començar a crear els nodes i relacions. A continuació es mostraran els mètodes que es van crear per crear els nodes “Paper” i les relacions “CITES”.

```
def create_paper_node(self, paper_title, publication_year, conference,
main):
    with self.driver.session() as session:
       if self.exists_paper_node(paper_title):
           return
       query = ("CREATE (p:Paper {title: $paper_title, year:
       $publication_year, conference: $conference, main: $main})")
       self.driver.execute_query(query, paper_title=paper_title,
       publication_year=publication_year, conference=conference,
       main=main,  database_=self.database)
```

Codi 18. Mètode create_paper_node()

```
def create_paper_citation_relationship(self, citing_paper_title,
cited_paper_title):
    with self.driver.session() as session:
       if not self.exists_paper_node(citing_paper_title):
           #print(f"{citing_paper_title} node does not exist")
           return
       if not self.exists_paper_node(cited_paper_title):
           #print(f"{cited_paper_title} node does not exist")
           return
      query = ("MATCH (c:Paper {title: $citing_paper_title}) MATCH
      (d:Paper {title: $cited_paper_title}) CREATE (c)-[:CITES]->(d)")
      self.driver.execute_query(query,
      citing_paper_title=citing_paper_title,
      cited_paper_title=cited_paper_title, database_=self.database)
```

Codi 19. Mètode create_paper_citation_relationship()

6.3. Resultats

En aquesta secció es trobaran els resultats obtinguts després de processar i analitzar les dades aconseguides mitjançant el crawler. Convé ressaltar que els resultats són orientatius, ja que malauradament les dades poden contenir errors o estar incompletes.

Abans de començar a mostrar les diferents anàlisis elaborades juntament amb els seus resultats, s'han de definir de forma detallada les variables que usarem per determinar la representació dels països i continents en les diferents conferències.

6.3.1. PC Participation

En primer lloc, trobem el que anomenem PC Participation. Aquesta variable ens permet determinar la importància que té el fet de pertànyer a la mateixa conferència a l'hora de publicar un article en aquesta. El que es mesura amb aquesta variable és el nombre de membres pertanyents al Program Committee que han publicat articles al mateix temps que han estat membres del comitè d'acceptació de la conferència. Si aquesta té un valor elevat, ens pot indicar que un autor membre del comitè podria tenir avantatge per damunt d'un altre que no hi pertanyés. Aquesta variable es calcula de la següent manera:

$$PC_p = \frac{M_{PP}}{M_{total}}$$

- PC_p: Program Committee Participation
- M_{pp}: Nombre de membres del Program Committee que han publicat un article
- M_{total}: Nombre total de membres del Program Committee

A continuació es troben els gràfics realitzats, en els quals es poden observar els diferents valors del PC_p obtinguts per cadascuna de les deu conferències en rang d'anys estudiat.

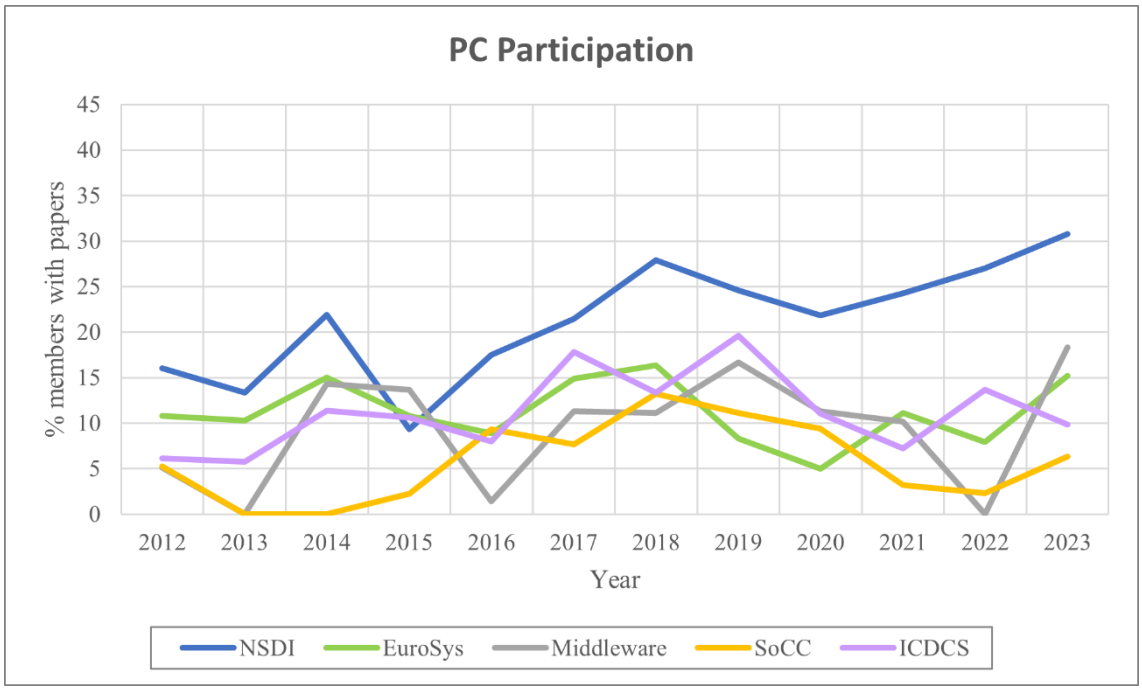


Figura 16. Resultats PC Participation 1

En aquesta primera gràfica es pot observar com hi ha una conferència que sembla tenir un valor bastant més elevat de PC_p respecte a la resta de conferències mostrades. Es tracta de NSDI, la qual sembla arribar a valors de fins a 30%.

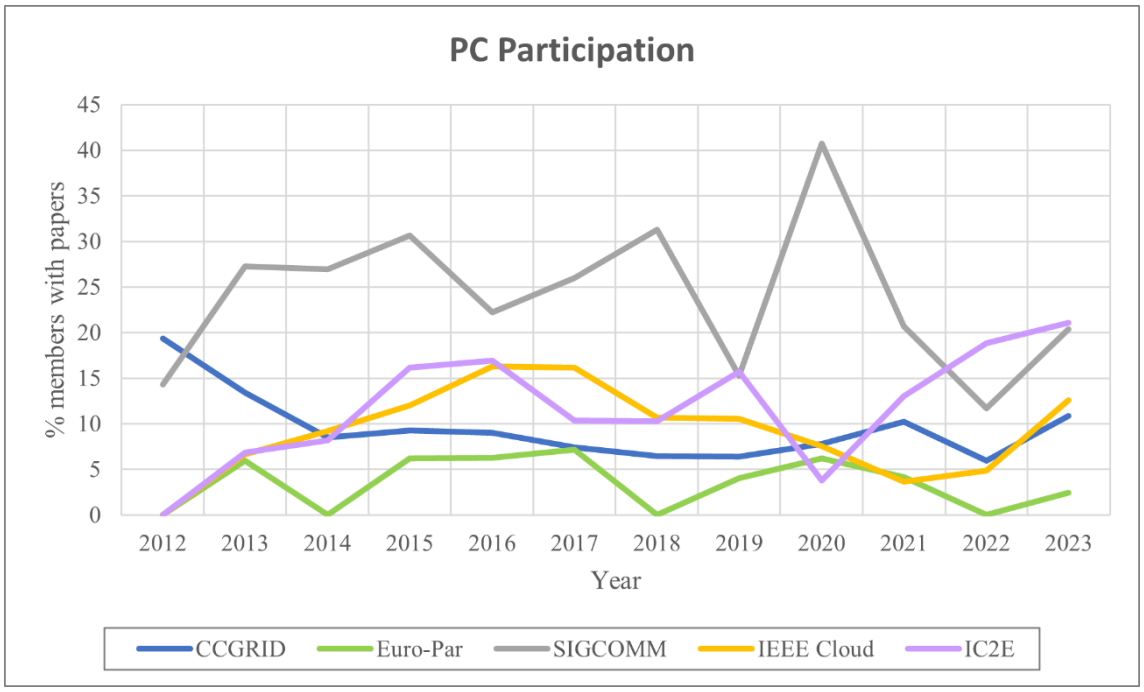


Figura 17. Resultats PC Participation 2

En la gràfica anterior podem observar els valors de PC_p per les conferències restants. En aquesta veiem que també hi sembla haver una conferència amb un percentatge major respecte de les altres, és tracta de SIGCOMM. Encara que en aquest cas sembla que en els últims tres anys analitzats aquesta tendència sembla reduir-se.

6.3.2. Country Diversity

La segona variable que s'usarà per determinar la diversitat de països d'una conferència és Country Diversity. Aquesta permetrà identificar com és d'internacional una conferència. Si aquesta diversitat té un valor reduït, pot indicar que la conferència pot estar dominada per un grup reduït de països, o fins i tot, per un sol país. Per tant, podria estar indicant que per als països aliens a aquests grups és més complicat publicar-hi un article.

Pel que fa als càlculs realitzats, s'ha de comentar un punt essencial per entendre de forma correcta els resultats que s'han obtingut. Com amb aquesta variable es vol determinar a quin país pertany un article, per posteriorment realitzar els càlculs pertinents, es van haver de distribuir els diferents articles en països. Per fer-ho es van tenir en compte els països de les institucions dels autors d'aquest. Finalment, es va determinar que un article seria d'un país determinat si més de la meitat dels seus autors pertanyien a institucions d'un cert país. En cas de tenir més d'un país majoritari, l'article constarà de més d'un país predominant, per la qual cosa aquest podrà ser comptabilitzat com un article

pertanyent a qualsevol dels seus països predominants. Així doncs, el que s'observarà en els resultats obtinguts en aquest apartat seran respecte al país o països predominants dels articles d'una certa conferència.

Així doncs, un cop explicat com es classifiquen els articles per països, a continuació s'explica de forma detallada els càlculs i variables que es tenen en compte per poder determinar el Country Diversity.

- Nombre de països predominants diferents que han aparegut durant els diversos anys de la conferència: Encara que amb aquest valor no es pot determinar la diversitat de forma concreta, si ens dona una petita mostra del que podria ser una conferència més oberta, o una de menys. Estudiant el nombre de països que han aparegut en les conferències, juntament amb el nombre d'articles publicats en cadascuna d'aquestes podem observar la relació que hi ha entre aquestes dues variables.

- Percentatge d'articles publicats de cada país (tenint en compte el país predominant): Aquest estudi ens permet veure de forma clara la participació dels diferents països dins de la conferència, ja que permet determinar si hi apareixen grups de països predominants. El fet de tenir un petit grup de països predominant indica que és molt més difícil per un article d'un altre país aconseguir publicar un article dins d'aquesta. En cas contrari, si una conferència sembla tenir un percentatge més equitatiu entre diversos països, podria estar indicant que aquesta és més diversa i oberta a publicar articles pertanyents a diferents països.

- Percentatge d'articles distribuïts per continents: De forma similar que en el punt anterior, els articles s'han classificat en continents segons el país o països predominants. Aquesta mesura ens mostra quins continents predominen en les diverses conferències. D'igual manera que en el punt anterior, un percentatge repartit de forma equitativa ens indica una major diversitat, en aquest cas de continents. És important tenir en compte que hi ha continents amb un nombre de països més elevat respecte a la resta, la qual cosa s'ha de tenir en compte a l'hora d'estudiar els resultats obtinguts.

En definitiva, la diversitat de països o Country Diversity pot indicar una major diversitat en la conferència, mentre que una menor diversitat pot indicar una major dificultat de publicació d’articles que no pertanyin a un cert grup de països.

A continuació es troben els resultats obtinguts al realitzar els diferents anàlisis presentats en els punts anteriors.

Nombre de països

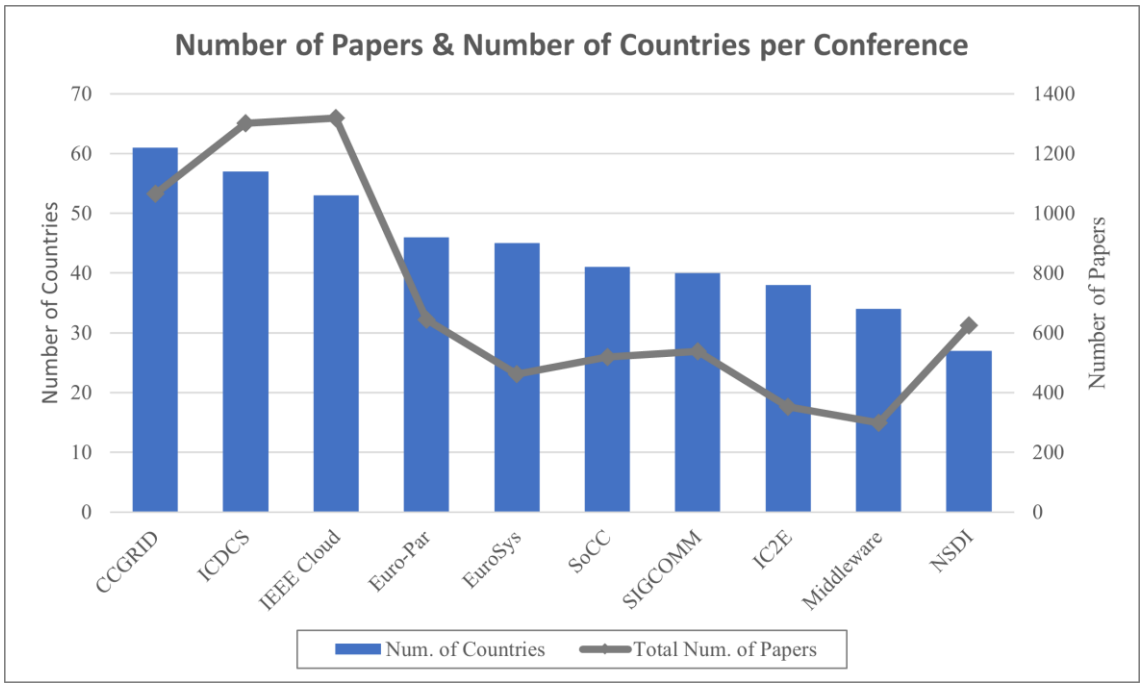


Figura 18. Nombre de països i articles publicats en les conferències

Com ha comentat amb anterioritat, aquest valor del nombre de països diferents en si mateix no ens aporta informació amb la qual es pugui determinar si cap de les premisses que es volen comprovar són certes, però duent a terme l'anàlisi juntament amb el nombre d'articles publicats si podem observar una relació interessant.

En la representació anterior les conferències es troben ordenades de major a menor respecte al nombre de països diferents en aquesta. Aquest nombre de països es troba representat amb les barres blaves. Per altra banda, el nombre d'articles es troba representat amb la línia grisa. A ull nu es pot observar una relació entre aquestes dues variables, la qual sembla indicar que a mesura que el nombre d'articles publicats disminueix, el nombre de països també tendeix a reduir-se.

Per observar aquesta relació matemàticament, s'ha aplicat la correlació de Pearson entre les dues variables. Aquesta correlació es calcula seguint la següent fórmula:

$$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\,\sqrt{n\sum y_i^2 - (\sum y_i)^2}}$$

- r_{xy}: correlació de Pearson
- n: mida de la mostra
- x_i, y_i: punts mostrals individuals indexats per i

En calcular la correlació per les dades mostrades en la gràfica anterior obtenim el valor de 0,778. Aquest ens està indicant que existeix una correlació positiva entre les dues variables, aquesta indica que quan una de les variables augmenta el seu valor, l'altra tendeix a fer el mateix. Malgrat això, podem veure com aquesta relació no és positiva perfecta, ja que hi ha casos on una de les variables augmenta, però l'altra disminueix el seu valor. El cas més visible d'aquest fet és el de la conferència NSDI, on el nombre d'articles publicats és pràcticament el doble comparat amb les conferències anteriors a ella, però el nombre de països diferents d'aquesta és menor respecte a la resta de conferències.

Percentatges dels diferents països

A continuació es mostraran uns gràfics amb la finalitat de mostrar els països més predominants a la conferència (tenint en compte el país que hi predomina en cada article). Per tal de facilitar la comprensió de les gràfiques, aquestes estan ordenades de major a menor respecte al percentatge del país més predominant.

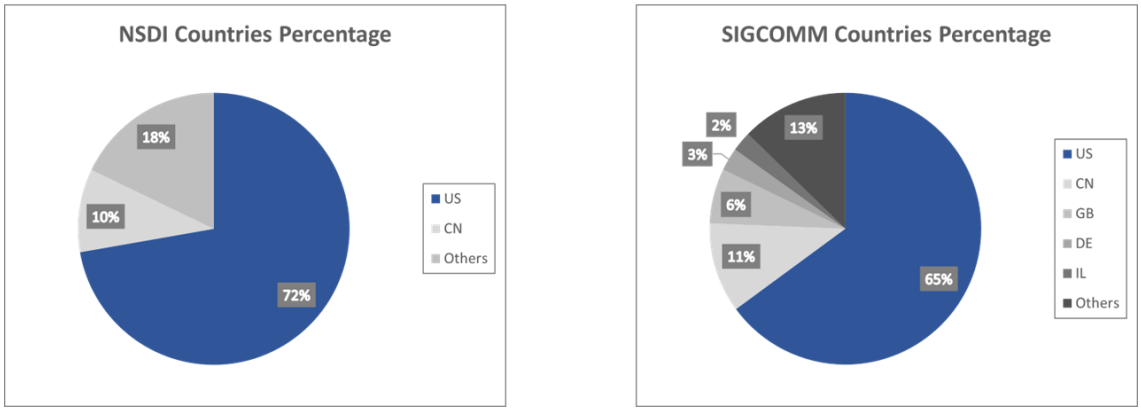


Figura 19. Percentatge de països de NSDI i SIGCOMM

Com es pot observar, la conferència amb el percentatge més gran és NSDI, amb un 72% dels articles tenint els Estats Units com a país predominant, seguit per la Xina representat per un 10% dels articles. Tots els altres països que han participat en la conferència representen el 18% dels articles publicats al llarg dels anys estudiants. NSDI és seguida per SIGCOMM, que, de forma similar té als Estats Units com a país predominant amb un 65% dels articles. D'igual manera, aquest precedeix a la

Xina, amb l'11% dels articles publicats. En aquest cas, trobem dos països més, en aquest cas Europeus, que sumen el 9% dels articles.

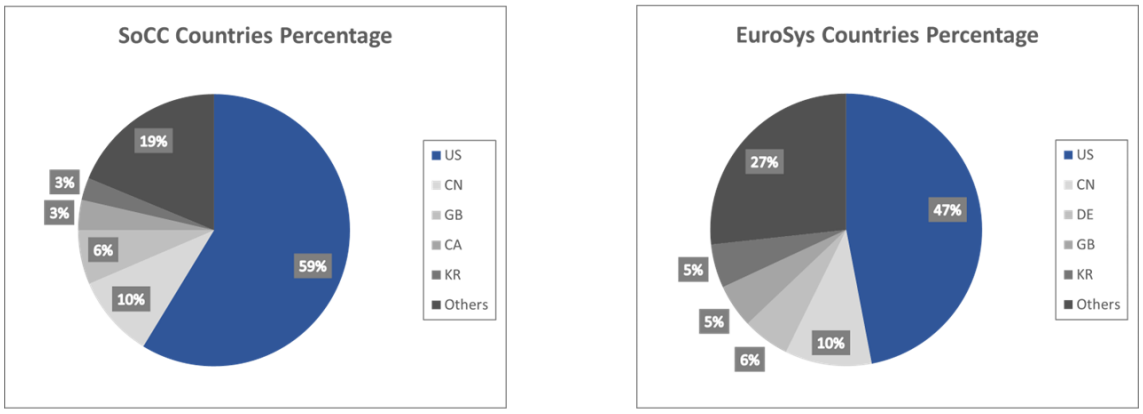


Figura 20. Percentatge de països de SoCC i EuroSys

Seguint a NSDI i SIGCOMM trobem SoCC i EuroSys, que també tenen als Estats Units com a país predominant en els articles de la conferència amb un 59% i 47% respectivament. D'igual forma que en les dues conferències anteriors, Xina és el segon país més predominant en els articles, amb un 10% en les dues conferències.

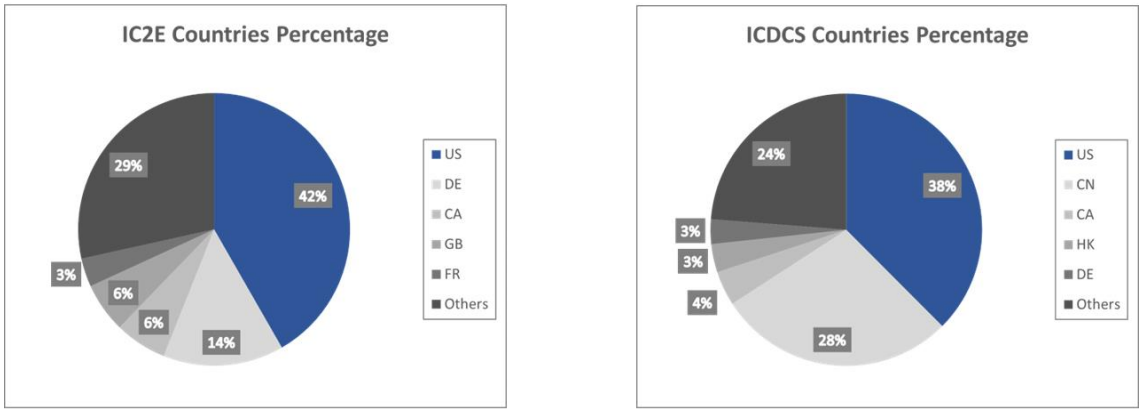


Figura 21. Percentatge de països de IC2E i ICDCS

Tot seguit, trobem a IC2E i ICDCS, que continuen mantenint als Estats Units com el país més predominant amb el 42% i 38% dels articles respectivament. En el cas de IC2E el segon país més predominant canvia respecte a les conferències anteriors, i passa a ser Alemanya amb un 14% dels articles. Per ICDCS el segon país més predominant continua sent la Xina, però en aquest cas veiem un augment del percentatge respecte a les altres conferències, un 28%. Es pot observar com ICDCS és la primera

conferència de les que ha aparegut fins ara en aquests gràfics on la diferència entre el percentatge del país més predominant respecte al segon és 10% o menor.

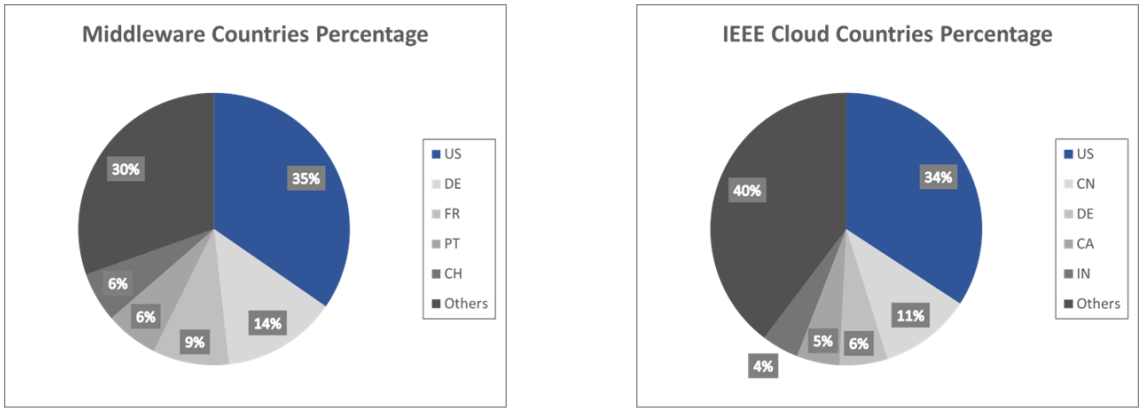


Figura 22. Percentatge de països de Middleware i IEEE Cloud

Observem que tant per Middleware com per IEEE Cloud, Estats Units continua sent el país més predominant amb un 35% i un 34% respectivament. Igual que en les últimes conferències, el segon país més predominant és Alemanya per Middleware i la Xina per IEEE Cloud. A més a més, aquestes són les dues primeres conferències on podem veure que el percentatge d'altres països (Others) s'apropa molt al percentatge del país més predominant, arribant a sobrepassar-lo en el cas de IEEE Cloud.

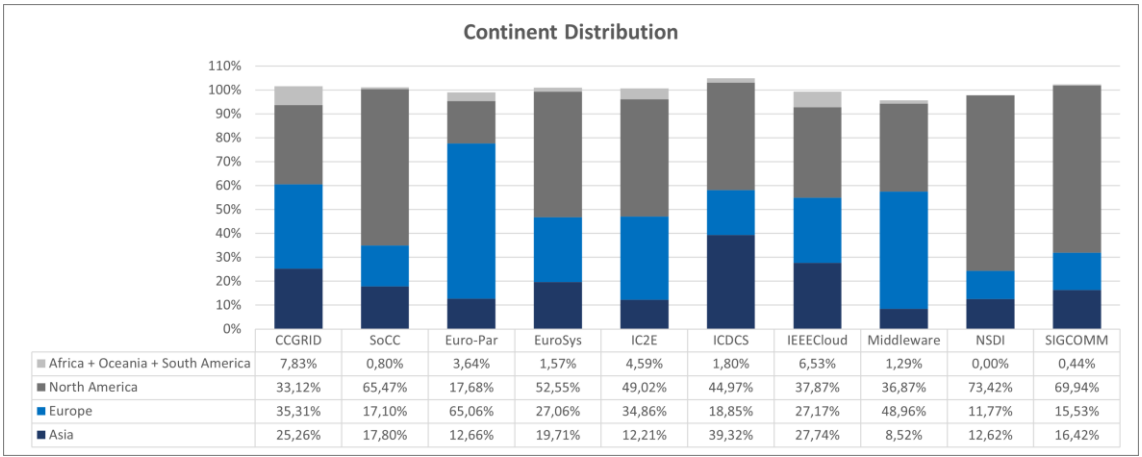
Figura 23. Percentatge de països de CCGRID i Euro-Par

Finalment, trobem a CCGRID i Euro-Par. Per CCGRID veiem que segueix la tendència d'algunes de les conferències anteriors, sent els Estats Units el país més predominant amb un 30% i la Xina el segon amb un 11%. A més, també s'observa com el percentatge de la resta de països supera el percentatge del país més predominant. Pel cas d'Euro-Par podem veure com és l'única conferència on el país més predominant no és els Estats Units, sinó que en aquest cas és França amb un 19% dels papers. El segon país

més predominant que trobem sí que es tracta dels Estats Units, amb un 17%. Podem veure com Euro-Par és la conferència on els tres països més predominants tenen un percentatge més similar.

En conclusió, podem veure com els Estats Units és un país molt actiu en totes les conferències que s'han estudiat, seguit de la Xina i Alemanya. També es pot observar com hi ha certes conferències que tendeixen a tenir un sol país predominant, tenint un percentatge major al 50% dels articles publicats en la conferència.

Distribució d'articles per continents



	CCGRID	SoCC	Euro-Par	EuroSys	IC2E	ICDCS	IEEECloud	Middleware	NSDI	SIGCOMM
Africa + Oceania + South America	7,83%	0,80%	3,64%	1,57%	4,59%	1,80%	6,53%	1,29%	0,00%	0,44%
North America	33,12%	65,47%	17,68%	52,55%	49,02%	44,97%	37,87%	36,87%	73,42%	69,94%
Europe	35,31%	17,10%	65,06%	27,06%	34,86%	18,85%	27,17%	48,96%	11,77%	15,53%
Asia	25,26%	17,80%	12,66%	19,71%	12,21%	39,32%	27,74%	8,52%	12,62%	16,42%

Figura 24. Distribució d'articles per continents

La figura anterior mostra la distribució dels articles de cada conferència dividits en continents. Com es pot observar, en la majoria de les conferències, el continent Nord Americà és el més predominant, fet que té sentit si s'han observat els gràfics de l'apartat anterior.

Si parlem d'una distribució equitativa, es pot veure com CCGRID i IEEE Cloud semblen ser les que més ho són. D'altra banda, NSDI, SIGCOMM, SoCC i Euro-Par semblen ser les menys equitatives. Les tres primeres són dominades pel continent Nord Americà, mentre que l'última és dominada pel continent Europeu. Malgrat això, la diversitat de països d'Euro-Par continua sent major respecte a les altres tres conferències, perquè hi ha una distribució equitativa dels articles entre els diferents països europeus, en canvi, en les altres tres conferències, la majoria dels articles Nord Americans van destinats als Estats Units.

6.3.3. Citations Proportion

Amb aquesta mesura es vol determinar a quins països i continents se citen en cada conferència, per tal de poder identificar patrons de citacions entre els diferents països i continents, i poder determinar la diversitat de les citacions dels diferents països d'una conferència, així com la diversitat global de la conferència.

Per poder realitzar aquest estudi es realitza un recompte de quins països cita un article amb un determinat país predominant, i a partir d'aquest recompte, s'extreuen els resultats.

Pel que fa a la diversitat de les cites de cada país, aquesta es calcula tenint en compte la proporció d'autocites del país que s'està estudiant, el nombre de països diferents dels quals cita, el nombre total d'articles de cada país que han estat citats en la conferència i el nombre d'articles del mateix país que han estat publicats en la conferència i citen. A continuació es mostren les fórmules que fan possible aquests càlculs.

Per cada país de la conferència es tenen les següents dades:

- C: país del qual es vol analitzar la diversitat
- Pi: Nombre d'articles del país C que citen articles d'un determinat país i
- Ti: Nombre total d'articles citats del país i que han estat citats pels articles del país C
- N: Nombre total de països diferents citats per articles del país C

1. Càlcul de la proporció de cites del país C a cada país i

$$p_i = \frac{T_i}{\sum_j T_j}$$

2. Càlcul de l'índex de Shannon

$$H = -\sum_{i=1}^{N} p_i \cdot \log(p_i)$$

3. Proporció d'autocites del país C

$$PA = \frac{T_C}{\sum_i T_i}$$

4. Proporció d'articles del país C que citen al país i

$$q_i = \frac{P_i}{\sum_j P_j}$$

5. Ajustar l'índex de Shannon

$$H_{adjusted} = H \times (1 - PA) \times \sum_{i=1}^{N} q_i$$

Per realitzar el càlcul de la diversitat global de la conferència es necessiten les següents dades:

- Hi : diversitat de cites del país i
- $\sum_j T_j$: total de cites realitzades pel país i
- $\sum_i \sum_j T_{ij}$: total de cites realitzades per tots els països en la conferència

$$H_{global} = \sum_i \left(\frac{\sum_j T_j}{\sum_i \sum_j T_{ij}} \right)$$

A continuació es mostren les taules i gràfiques realitzades, les quals contenen la informació més rellevant de les cites realitzades pels països més predominants a cadascuna de les conferències.

La informació que es mostrarà en les taules és la mateixa per totes les conferències, primer trobem els cinc països més predominants en la conferència, i per cadascun d'aquests països trobem els cinc països als quals sol citar més sovint. El percentatge que es troba amb aquests països representa el percentatge d'articles que ha citat el país de l'altre país respecte al total d'articles citats del país. En la taula també es troba el país més citat de la conferència, juntament amb el percentatge d'articles d'aquell país respecte al total dels articles citats de tots els països. Finalment, trobem la diversitat de les cites de cadascun dels països més predominants de la conferència, juntament amb la diversitat global de la conferència.

NSDI

NSDI		Most Cited Country			US: 64,48%		*Diversity of the citing country*
		Most Cited Countries in Order (by the top countries)					
Conference Top Countries	US	US: 75,58%	GB: 73,80%	CN: 63,30	DE: 67,57%	CA: 73,10%	0,0078
	CN	US: 7,31%	CN: 21,94%	GB: 8,03%	CH: 9,06%	CA: 8,54%	0,0323
	CH	US: 3,54%	CH: 12,57%	GB: 4,54%	DE: 5,69%	CN: 1,76%	0,0515
	GB	US: 3,03%	GB: 4,65%	CN: 2,39%	CH: 5,26%	CA: 4,11%	0,0471
	DE	US: 0,90%	DE: 2,22%	CN: 1%	CA: 2,21%	IT: 3,52%	0,0737

Taula 2. Taula de citacions de NSDI

Diversitat Global = 0,0216

Com es pot veure, el país més citat és els Estats Units, que a més a més, és el país que més se cita a si mateix, i el que té una diversitat de citacions menor respecte a la resta de països.

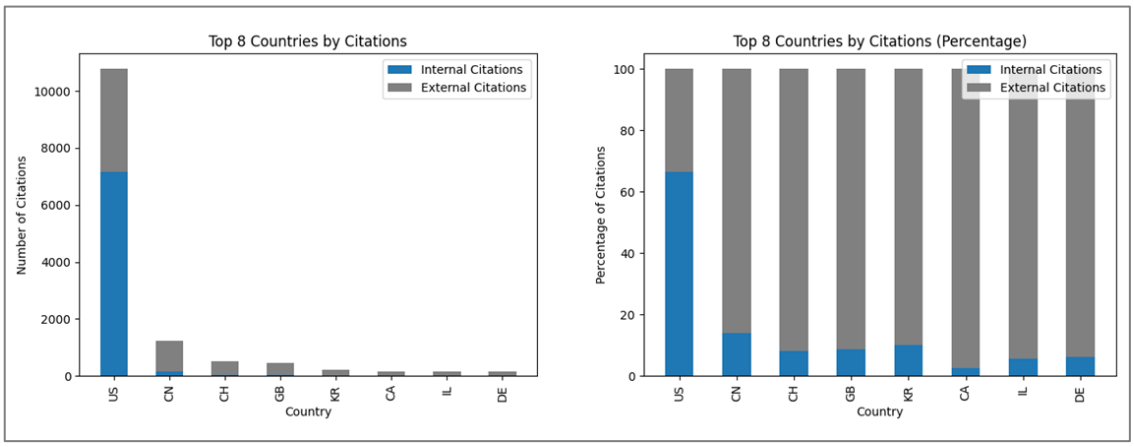


Figura 25. Resultats cites NSDI

En els gràfics anteriors es pot observar com els Estats Units és el país que més cita, sent més del 60% de les citacions que realitzen cap a ells mateixos. Per la resta de països, menys del 20% de les citacions que realitzen són per a ells mateixos.

SoCC

SoCC		Most Cited Country			US: 63,92%		*Diversity of the citing country*
		Most Cited Countries in Order (by the top countries)					
Conference Top Countries	US	US: 63,28%	GB: 51,87%	CN: 38,46%	DE: 54,86%	CA: 55,31%	0,0079
	CN	US: 8,77%	CN: 26,37%	GB: 10,23%	DE: 9,45%	CA: 11,70%	0,0395
	GB	US: 6,89%	GB: 12,28%	CN: 7,53%	DE: 6,21%	CA: 5,67%	0,0416
	CA	US: 3,29%	CN: 4,08%	GB: 4,43%	CA: 7,8%	DE: 3,78%	0,0533
	KR	US: 1,11%	CN: 3,61%	GB: 2,21%	DE: 3,51%	KR: 8,82%	0,0756

Taula 3. Taula de citacions de SoCC

Diversitat Global = 0,0332

En la taula podem observar que per SoCC les citacions són bastant similars a les de NSDI, on els Estats Units és el país més citat amb més del 60% de les citacions anant cap a ells. A més a més, també podem veure com els Estatunidencs citen als mateixos països que en NSDI. D'igual manera, Estats Units és el país amb menor diversitat de citacions.

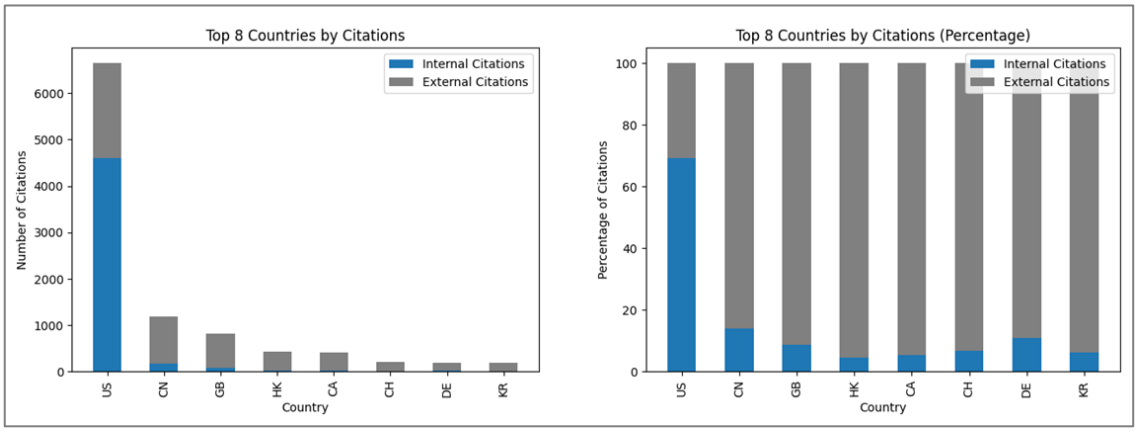


Figura 26. Resultats cites SoCC

D'igual forma que en NSDI, Estats Units és el país que més cita amb diferència, i més del 60% de les cites que realitzen són cap a ells mateixos.

Middleware

Middleware		Most Cited Country			US: 50,48%		*Diversity of the citing country*
		Most Cited Countries in Order (by the top countries)					
Conference Top Countries	US	US: 36,11%	CN: 33,33%	GB: 25,29%	DE: 18,33%	CA: 29,10%	0,0127
	DE	US: 10,62%	DE: 47,85%	GB: 15,54%	CH: 14,68%	CN: 14,11%	0,0434
	FR	US: 8,78%	FR: 29,45%	CH: 15,78%	DE: 9,76%	GB: 9,28%	0,0485
	PT	US: 6,77%	PT: 34,56%	FR: 10,90%	GB: 6,03%	CH: 6,09%	0,0547
	CH	US: 6,91%	CH: 23,54%	FR: 9,45%	GB: 5,56%	DE: 4,52%	0,0496

Taula 4. Taula de citacions de Middleware

Diversitat Global = 0,0486

Per aquesta conferència podem observar com els Estatunidencs continuen sent els més citats, aquest cop amb un percentatge menor, representant el 50% de les cites totals realitzades en la conferència. Si observem les citacions dels Estats Units, es pot veure com encara citen als mateixos països, encara que en un ordre una mica diferent. D'igual forma que en les altres conferències, Estats Units encara és el país amb menor diversitat de cites.

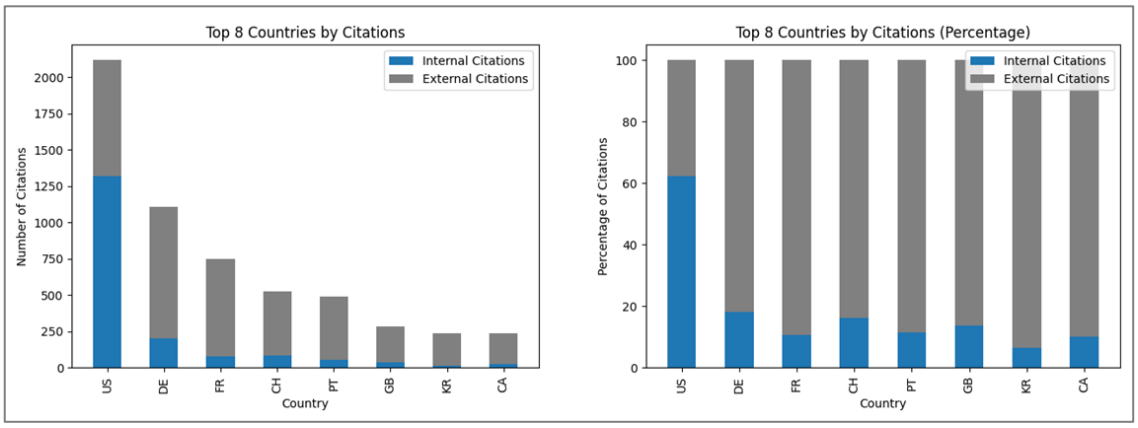


Figura 27. Resultats cites Middleware

D'igual manera que en les conferències anteriors, Estats Units és el país que més cita, sent el 60% de les cites cap a ells mateixos.

EuroSys

EuroSys		Most Cited Country			US: 60,56%		*Diversity of the citing country*
		Most Cited Countries in Order (by the top countries)					
Conference Top Countries	US	US: 50,94%	GB: 45,02%	CN: 37,67%	DE: 42,6%	CH: 46,72%	0,0102
	CN	US: 10,64%	CN: 28,08%	GB: 8,97%	CA: 10,51%	DE: 7,4%	0,0367
	DE	US: 5,46%	DE: 18,6%	GB: 9,80%	CN: 4,79%	PT: 29,21%	0,0589
	GB	US: 4,71%	GB: 6,62%	CN: 3,42%	DE: 4,4%	CH: 3,73%	0,0574
	KR	US: 4,65%	GB: 6,35%	KR: 20,97%	CN: 5,34%	DE: 5,2%	0,0536

Taula 5. Taula citacions EuroSys

Diversitat Global = 0,0392

En EuroSys tornem a trobar als Estats Units com el país més citat, corresponent al 60% de les cites totals realitzades. D'igual manera, podem veure com els Estats Units continua citant al mateix grup de països.

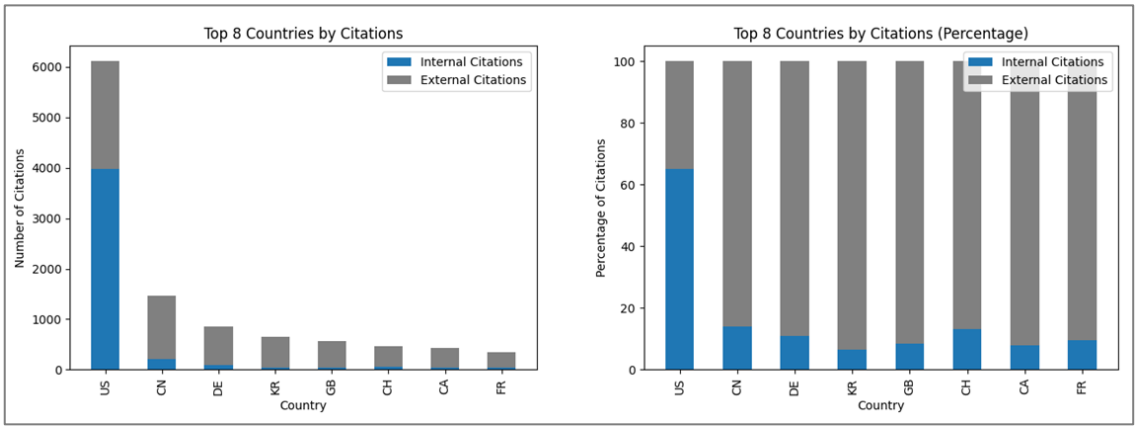


Figura 28. Resultats cites EuroSys

En aquesta conferència tornem a veure com els Estats Units és el país que més cita, sent més d'un 60% de les citacions que realitzen cap a ells mateixos.

ICDCS

ICDCS		Most Cited Country			US: 48,78%		*Diversity of the citing country*
		Most Cited Countries in Order (by the top countries)					
Conference Top Countries	US	US: 49,36%	CN: 28,47%	GB: 38,40%	CA: 33,71%	DE: 30,35%	0,0107
	CN	US: 26,06%	CN: 54,91%	GB: 27,37%	DE: 25,41%	CA: 25,60%	0,0238
	CA	US: 3,95%	CA: 13,53%	CN: 2,74%	FR: 7,11%	GB: 4,01%	0,0477
	HK	US: 3,22%	CN: 5,18%	HK: 17,57%	GB: 4,55%	DE: 2,98%	0,0506
	DE	US: 2,25%	DE: 11,93%	CN: 1,43%	GB: 3,47%	CA: 3,33%	0,0508

Taula 6. Taula citacions ICDCS

Diversitat Global = 0,0325

D'igual manera que en la resta de conferències, Estats Units és el país que més citacions rep, encara que per primera vegada veiem que el percentatge no supera el 50%.

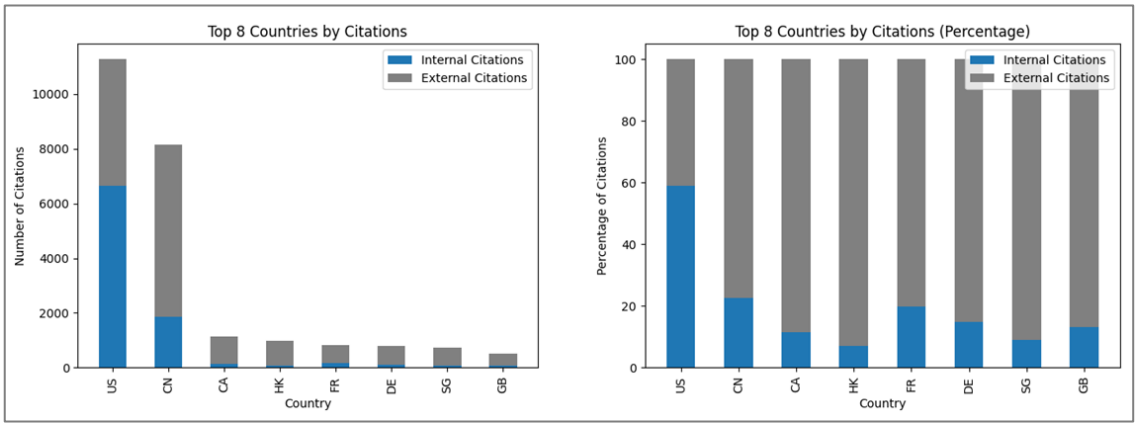


Figura 29. Resultats cites ICDCS

D'igual forma que en la resta de les conferències vistes fins ara, veiem que el país que més se cita a ell mateix és els Estats Units.

CCGRID

CCGRID		Most Cited Country			US: 50,11%		*Diversity of the citing country*
		Most Cited Countries in Order (by the top countries)					
Conference Top Countries	US	US: 45,10%	CN: 20,09%	GB: 23,39%	DE: 19,87%	FR: 14,98%	0,007
	CN	US: 9,39%	CN: 31,01%	GB: 9,18%	CA: 11,73%	FR: 5,69%	0,0336
	FR	US: 7,32%	FR: 44,94%	GB: 7,60%	DE: 8,27%	CN: 4,58%	0,0357
	AU	US: 2,92%	AU: 20,29%	CN: 4,05%	GB: 4,88%	DE: 5,41%	0,0543
	IN	US: 2,36%	CN: 8,02%	IN: 22,42%	GB: 5,31%	DE: 4,29%	0,0469

Taula 7. Taula citacions CCGRID

Diversitat Global = 0,0418

Per CCGRID tornem a veure als Estats Units dominant les cites amb un percentatge del 50%. D'igual manera veiem com aquest continua sent el país amb menys diversitat de cites, i s'observa com continua citant majoritàriament al mateix grup de països als quals ha anat citant en les conferències anteriors.

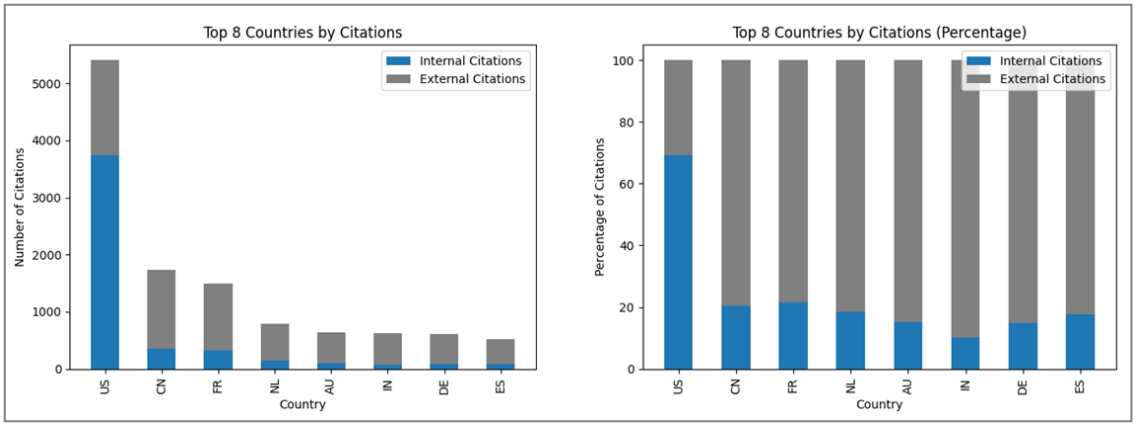


Figura 30. Resultats cites CCGRID

D'igual forma, Estats Units és el país que més se cita a si mateix, encara que sembla que aquesta és la conferència on major són els percentatges d'autocitacions.

Euro-Par

Euro-Par		Most Cited Country			US: 47,78%		*Diversity of the citing country*
		Most Cited Countries in Order (by the top countries)					
Conference Top Countries	FR	US: 18,03%	FR: 62,46%	DE: 11,92%	GB: 14,20%	CN: 18,00%	0,0346
	US	US: 25,83%	GB: 20,85%	DE: 11,92%	FR: 7,68%	CA: 18,78%	0,0142
	DE	US: 12,62%	DE: 45,02%	FR: 7,10%	CN: 12,80%	GB: 9,06%	0,039
	ES	US: 8,15%	ES: 51,96%	GB: 8,76%	CN: 9,60%	DE: 3,38%	0,04
	CN	US: 1,93%	DE: 1,96%	GB: 2,11%	CN: 2,40%	JP: 3,48%	0,0884

Taula 8. Taula citacions Euro-Par

Diversitat Global = 0,0537

Recordem que Euro-Par és l'única conferència on els Estats Units no és el país predominant, encara així continuem veient com el país més citat encara són els Estats Units amb un 47%. Un fet interessant que es pot veure en aquesta taula és com els Estats Units continua citant al mateix grup de països que cita en totes les conferències, però en aquest cas, en tractar-se d'una conferència majoritàriament europea, veiem com primer cita a països europeus.

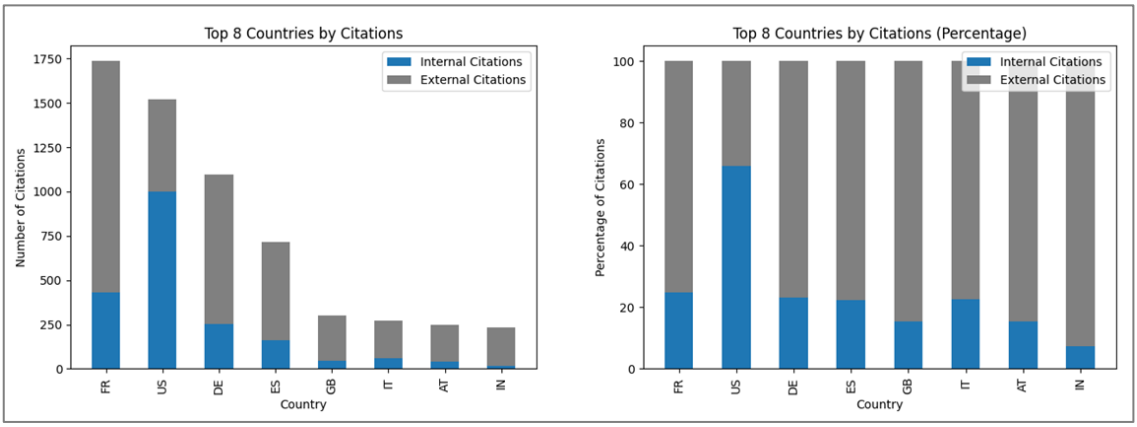


Figura 31. Resultats cites Euro-Par

En les gràfiques anteriors podem observar, que malgrat que els Estats Units no és el país que més citacions té en la conferència (en aquest cas correspon a França), encara veiem com el país que més se cita a si mateix és els Estats Units.

SIGCOMM

SIGCOMM		Most Cited Country			US: 64,20%		*Diversity of the citing country*
		Most Cited Countries in Order (by the top countries)					
Conference Top Countries	US	US: 66,79%	GB: 61,25%	CN: 54,33%	DE: 52,55%	CA: 63,32%	0,009
	CN	US: 12,88%	CN: 30,89%	GB: 12,03%	DE: 10,71%	CH: 18,98%	0,032
	GB	US: 6,85%	GB: 14,80%	CN: 3,49%	DE: 7,14%	CH: 9,26%	0,0377
	DE	US: 1,96%	DE: 11,22%	CN: 2,88%	GB: 2,05%	IL: 3,16%	0,0536
	IL	US: 2,16%	IL: 9,88%	GB: 2,29%	DE: 3,06%	CN: 1,44%	0,0647

Taula 9. Taula citacions SIGCOMM

Diversitat Global = 0,0285

Continuant amb la tendència de les conferències anteriors, Estats Units és el país més citat amb un 64% dels articles totals citats. D'igual forma que en les conferències anteriors, Estats Units continus citant al mateix grup de països.

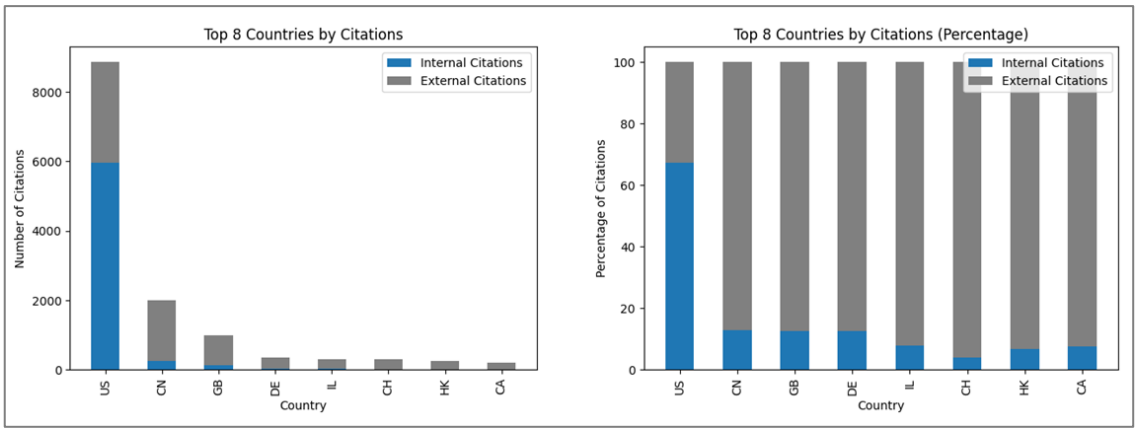


Figura 32. Resultats cites SIGCOMM

D'igual manera, veiem com els Estats Units és el país amb més citacions internes, és a dir, cap a si mateix.

IEEE Cloud

IEEE Cloud		Most Cited Country			US: 43,72%		*Diversity of the citing country*
		Most Cited Countries in Order (by the top countries)					
Conference Top Countries	US	US: 47,40%	CN: 24,42%	GB: 31,43%	DE: 22,51%	CA: 28,70%	0,0121
	CN	US: 9,15%	CN: 31,05%	AU: 11,79%	GB: 8,76%	DE: 7,26%	0,0288
	DE	US: 4,16%	DE: 22,20%	GB: 6,54%	AU: 5,77%	CN: 2,36%	0,0386
	CA	US: 4,10%	CA: 18,74%	CN: 5,01%	AU: 4,49%	DE: 3,53%	0,0415
	IN	US: 3,67%	CN: 3,40%	IN: 13,04%	AU: 3,59%	FR: 4,63%	0,046

Taula 10. Taula citacions IEEE Cloud

Diversitat Global = 0,0394

Per IEEE Cloud tornem a trobar als Estats Units com el país més citat, encara que en aquest cas amb el percentatge més petit que trobarem en qualsevol de les conferències mostrades.

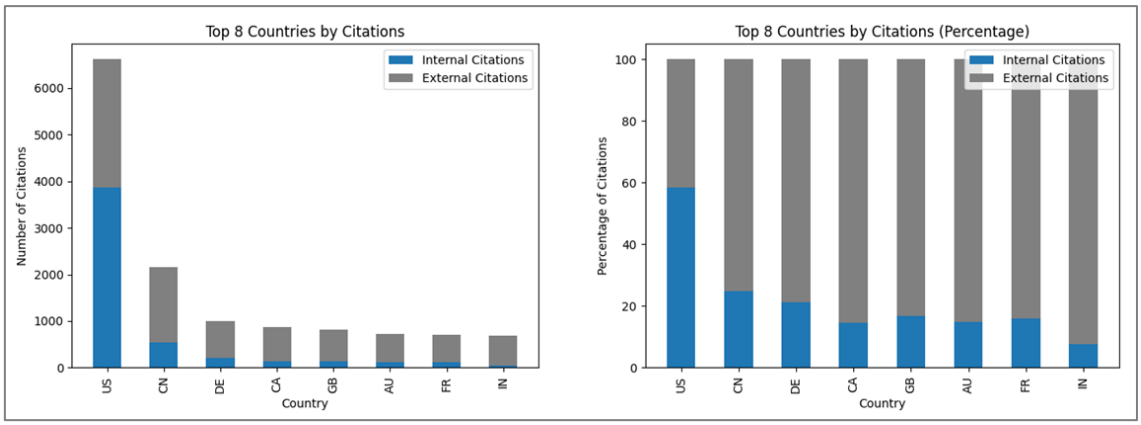


Figura 33. Resultats cites IEEE Cloud

D'igual forma, Estats Units continua sent el país amb major percentatge d'autocitacions, encara que en aquesta conferència també trobem països amb percentatges més elevats respecte a la resta de conferències.

IC2E

IC2E		Most Cited Country			US: 46,38%		*Diversity of the citing country*
		Most Cited Countries in Order (by the top countries)					
Conference Top Countries	US	US: 54,37%	GB: 41,46%	CN: 32,09%	DE: 20,41%	CA: 36,00%	0,0133
	DE	US: 10,70%	DE: 53,88%	CN: 12,89%	AU: 21,05%	GB: 12,54%	0,0343
	CA	US: 4,12%	CA: 18,29%	CN: 8,60%	IN: 10,81%	DE: 2,24%	0,0615
	GB	US: 5,46%	GB: 18,12%	DE: 4,08%	AU: 7,60%	CA: 6,86%	0,056
	FR	US: 2,68%	FR: 22,73%	DE: 3,88%	CN: 3,44%	GB: 4,18%	0,0678

Taula 11. Taula citacions IC2E

Diversitat Global = 0,0499

L'última conferència que trobem és IC2E, on podem veure que, un cop més, els estatunidencs són els més citats, tot i que el percentatge encara és menor al 50%.

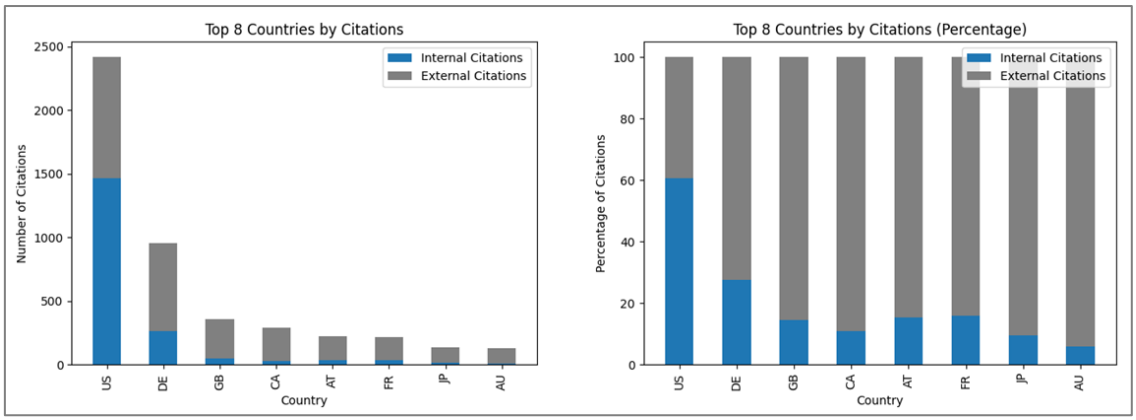


Figura 34. Resultats cites IC2E

D'igual manera, Estats Units continua sent el país amb un major percentatge d'autocitacions.

6.3.4. Conclusions dels Resultats

A partir de les anàlisis elaborades, els resultats obtinguts i tenint en compte les tres variables diferents en les quals ens basem, podem determinar la diversitat i representació de diferents països de les diferents conferències mostrades.

En primer lloc, podem determinar que NSDI i SIGCOMM semblen ser les conferències menys diverses. Aquests fets els veiem gràcies a les variables de PC Participation i Country Diversity., on podem veure per la primera que aquestes dues

conferències són les que tenen els valors més elevats al llarg dels anys. Pel que fa al Country Diversity, podem veure com NSDI és la conferència amb un menor nombre de països, mentre que SIGCOMM es troba en la meitat baixa de la taula. També s'observa com hi ha un país clarament predominant en les dues conferències, Estats Units, el qual engloba més d'un 70% dels articles publicats per NSDI, i un 65% per SIGCOMM.

Seguint a NSDI, i SIGCOMM trobem a SoCC, la qual també presenta als Estats Units com a país predominant, i igual que SIGCOMM, es troba en la meitat baixa de la taula respecte al nombre de països que hi participen. Per aquest motiu, si observem la distribució d'articles per continent, podem veure com aquestes tres conferències tenen al continent Nord Americà com el més predominant. D'aquesta manera, podem indicar de forma objectiva que la major part dels articles que s'hi publiquen en aquestes tres conferències provenen d'Estats Units, per la qual cosa, es pot afirmar que aquest mateix país és el predominant en aquestes conferències.

En segon lloc, podem veure com hi ha una altra conferència amb un continent més predominant respecte a la resta de conferències, aquesta es tracta d'Euro-Par, on veiem que el continent Europeu és el més predominant. Malgrat això, podem veure com la distribució d'articles per país és una de les més equitatives respecte de la resta de les conferències.

En tercer lloc, si observem la variable de Country Diversity aplicada a la distribució d'articles per continents, podem veure com les conferències amb una repartició més equitativa són CCGRID i IEEECloud. D'igual forma, són les dues conferències amb major percentatge de representació dels continents Sud Americà, Oceànic i Africà. Tanmateix, s'ha de tenir en compte que aquestes són dues de les conferències amb un major nombre d'articles publicats, fet que permet una major diversitat de països.

Finalment, referint-nos a la variable de Citations Proportion podem dir que els estatunidencs són els més citats, independentment de si la conferència es considera Americana, Europea o Asiàtica. D'igual forma, en totes les conferències es pot observar com els estatunidencs són els que se citen més a ells mateixos, on aproximadament un 60% de les citacions que realitzen són cap a articles dels Estats Units. Un altre fet que es pot observar amb els resultats que s'han extret, és que els Estats Units sembla dirigir la major part de les seves cites a un grup reduït de països. Segons els resultats obtinguts, aquest grup sembla estar conformat per la Xina, Gran Bretanya, Alemanya, Canadà, Suïssa i França.

7. Futur del Projecte

En aquest apartat s'explorarà el futur del projecte, explicant les possibles millores que es poden implementar i diferents anàlisis que es podrien realitzar a les dades.

D'entrada, cal indicar que pel que fa al crawler, no s'està plantejant realitzar cap millora, ja que actualment és possible obtenir totes les dades que són necessàries per als estudis actuals, i els que es plantegen en un futur.

Respecte a les dades, s'espera en un futur poder implementar una base de dades vectorials per indexar els abstracts i TLDRs obtinguts, per posteriorment, realitzar una cerca semàntica per similitud usant el que es coneix com embeddings. L'objectiu d'aquest nou estudi seria poder identificar articles que parlen d'un mateix tema, i fins i tot poder veure si hi ha articles que s'haurien de citar pel fet de tractar del mateix tema, però no ho fan. Tenint en compte això, podríem identificar comunitats científiques tancades en el sentit de què no citen articles pertanyents a altres comunitats.

Sobre les anàlisis que es realitzen sobre aquestes dades, en un futur s'espera determinar noves mètriques que ens permetin estudiar la diversitat i representació en les diverses conferències. Ens agradaria poder tenir accés a les dades referents als articles enviats a les conferències, per tal de veure si els articles presentats són diversos pel que als països o no. Això ens ajudaria a poder determinar si el fet que una conferència sigui més diversa que una altra es deu al fet que només s'accepten articles d'uns certs països, o bé que només es presenten articles d'uns certs països.

En resum, es pretén realitzar un estudi i anàlisi el més exhaustiu possible, per tal de poder estudiar i entendre en profunditat aquest camp de recerca científica.

8. Conclusions

Aquest projecte ha demostrat la viabilitat i efectivitat del desenvolupament d'un crawler web per l'extracció de dades relacionades amb articles de conferències en l'àmbit de la informàtica i les ciències de la computació. A través de la implementació del crawler en Python i mitjançant l'ús de les API d'OpenAlex i Semantic Scholar, s'ha pogut recopilar una quantitat significativa de dades que permeten realitzar un anàlisi detallat sobre aquestes.

Malgrat començar el projecte sense coneixement en l'àmbit del crawling de dades, i el processament i anàlisi de dades, s'han pogut comprendre i aplicar els conceptes més rellevants d'aquests camps.

En relació amb els resultats assolits, d'una banda, s'ha pogut implementar un crawler funcional i flexible, el qual és capaç d'extreure una gran varietat de dades relacionades amb articles acadèmics i de recerca. A més a més, aquest es pot modificar de forma senzilla, per tal de poder assolir els requisits de l'usuari final. D'altra banda, s'han pogut analitzar les dades obtingudes mitjançant el crawler implementat, i s'han pogut resoldre les preguntes plantejades inicialment. A més a més, aquestes dades s'han emmagatzemat de forma estructurada usant diferents bases de dades, les quals serviran per a que la comunitat científica en pugui fer ús.

En resum, aquest treball ha suposat un assoliment significatiu, demostrant la capacitat d'abordar desafius i permetent l'adquisició de nous coneixements en àrees tan rellevants com l'anàlisi de dades o el crawling web. El crawler desenvolupat compleix amb els objectius plantejats, d'igual manera que ho fa l'estudi realitzat sobre les dades. Aquest projecte posa les bases per fer una anàlisi profunda i detallada en el camp de la recerca i destaca la necessitat de continuar treballant cap a un accés més equitatiu a la recerca científica global.

Referències i Bibliografia

[1] *DBLP*: https://dblp.org

[2] *OpenAlex:* https://openalex.org

[3] *Semantic Scholar*: https://www.semanticscholar.org

[4] *Security Circus*: https://s3.eurecom.fr/~balzarot/security-circus/

[5] Payer, M. (s. f.). *Top Authors, the Systems Circus*. https://nebelwelt.net/pubstats/

[6] Jpwahle. (s. f.). *GitHub - jpwahle/cs-insights: The main controller for services in the cs-insights project through docker-compose.* https://github.com/jpwahle/cs-insights

Annex A: Codi Font Base Crawler

```
import time
import threading
from auxiliar import file, thread
from bs4 import BeautifulSoup
import requests
import re
import logging

data_per_year = {}

class BaseCrawler:
    def __init__(self, conferences, years, num_threads, output_dir):
        self.conferences = conferences
        self.years = years
        self.num_threads = num_threads
        self.output_dir = output_dir
        self.semaphore = threading.Semaphore(1)

    def crawl(self):
        initial_time = time.time()
        first_year, last_year = self.years
        for conf in self.conferences:
            global data_per_year
            data_per_year = {}

            threads = thread.Thread(self.num_threads)
            threads.run(self._search, (conf, first_year, last_year))

            file.save_json(f"{self.output_dir}/{conf}_basic_data",
data_per_year)

        final_time = time.time()
        minutes = (final_time - initial_time) / 60
        print(f"(BASE) - Done in {minutes:.3f} minutes")

    def _search(self, conf, first_year, last_year):
        """Retrieve all initial information of a conference and save
it into a JSON file. """
        links = self._get_links(conf)
```

```
        valid_links = []
        for link in links:
            if self._filter_dblp_links(conf, link) and any(str(year)
in link for year in range(first_year, last_year + 1)):
                valid_links.append(link)

        for link in valid_links:
            pub_list_raw = self._get_pub_list(link)
            for pub in pub_list_raw:
                article_items = pub.find_all('li', {'itemtype':
'http://schema.org/ScholarlyArticle'})
                header = pub.find_previous('h2')
                if self._filter_section(header.text):
                    continue
                for child in article_items:
                    pub_data = self._get_dblp_paper_data(child)
                    if pub_data is None:
                        continue
                    if pub_data['Year'] not in data_per_year:
                        self.semaphore.acquire()
                        data_per_year[pub_data['Year']] = []
                        self.semaphore.release()

                    self.semaphore.acquire()
                    data_per_year[pub_data['Year']].append(pub_data)
                    self.semaphore.release()


    def _get_links(self, conference):
        """Search for the links for each year of a specific conference
        # obtain the links for every year
        url = "https://dblp.org/db/conf/" + conference + "/"
        html_page = requests.get(url, timeout=10)
        soup = BeautifulSoup(html_page.text, 'html.parser')
        link_list = set()
        for link_elem in soup.findAll('a'):
            link = link_elem.get('href')
            if link and url in link:  # to avoid repeated links
                link_list.add(link)
        return link_list  # list with all the papers for each year
```

```
    def _get_pub_list(self, link):
        """Search all publications available on DBLP by parsing the
HTML file and searching for the class named publ-list. """

        resp = requests.get(link, timeout=10)
        soup = BeautifulSoup(resp.content, features="lxml")
        return soup.findAll("ul", attrs={"class": "publ-list"})


    def _get_dblp_paper_data(self, publication):
        """This function extracts all the data to then pass it to the
search() function. """

        publication_year = 'nothing'
        paper_title = 'nothing'
        authors_names =[]
        authors_institutions = None

        for content_item in publication.contents:
            class_of_content_item = content_item.attrs.get('class',
[0])
            if 'data' in class_of_content_item:
                # get the paper title from dblp
                paper_title = content_item.find('span',
attrs={"class": "title", "itemprop": "name"}).text
                if self._filter_paper_title(paper_title):
                    return None
                # get the publication year from dblp
                for datePublished in content_item.findAll('span',
attrs={"itemprop": "datePublished"}):
                    publication_year = datePublished.text
                if publication_year == 'nothing':
                    publication_year = content_item.find('meta',
attrs={"itemprop": "datePublished"}).get("content")

                # get the author's names from dblp paper
                for author in content_item.findAll('span',
attrs={"itemprop": "author"}):
                    authors_names.append(author.text)
```

```
            if 'publ' in class_of_content_item:
                links = content_item.contents[0].findAll("a")
                openalex_link = [l.get("href") for l in links if
"openalex" in l.get("href")]
                doi_number, authors_institutions, referenced_works =
self._get_openalex_data(openalex_link[0]) if openalex_link != [] else
(None, None, None)

            if authors_institutions is None:
                auth_list = []
                for author in authors_names:
                    auth_list.append({'Author': author,
'Institutions': None})

        return {'Title': paper_title,
                'Year': publication_year,
                'DOI Number': doi_number,
                'OpenAlex Link': openalex_link[0] if openalex_link !=
[] else None,
                'Authors and Institutions': authors_institutions if
authors_institutions is not None else auth_list,
                'OpenAlex Referenced Works': referenced_works}



    def _get_openalex_data(self, link):
        """Function that extracts the authors and institutions data
and the referenced works from the OpenAlex API. """
obtained from the initial data (dblp).

        response = requests.get(link)
        if response.status_code == 200:
            response_data = response.json()
            doi_link = response_data['doi']
            doi_number = doi_link.replace("https://doi.org/", "")
            authors_institutions =
self._get_authors_and_institutions(response_data)
            referenced_works =
self._get_referenced_works_openalex(response_data)
            return doi_number, authors_institutions, referenced_works
if referenced_works != [] else None
        else:
```

```
            logging.error(f"(BASE) - {response.status_code} in request
for link {link}")
            return None


    def _get_authors_and_institutions(self, response_data):
        """This function extracts the authors and institutions data
from the OpenAlex API response. Used in the _get_openalex_data
function. """

        auth_data = []
        authors = response_data['authorships']
        for a in authors:
            institutions = []
            author_name = a['author']['display_name']
            author_institutions = a['institutions']
            for inst in author_institutions:
                institution_data = self._get_institution_data(inst)
                institutions.append(institution_data)

            auth_data.append({'Author': author_name, 'Institutions':
institutions})

        return auth_data


    def _get_institution_data(self, institution):
        """Function that extracts the institution data from the
OpenAlex API response. Used in the _get_authors_and_institutions
function. """
        id_inst = institution['id'].replace("https://openalex.org/",
"")
        url = f"https://api.openalex.org/institutions/{id_inst}"
        response = requests.get(url)
        if response.status_code == 200:
            response_data = response.json()
            institution_name = response_data.get('display_name', None)
            institution_country = response_data.get('country_code',
None)
            return {'Institution Name': institution_name, 'Country':
institution_country}
```

```
        else:
            logging.error(f"(BASE) - {response.status_code} in request
for link {url}")
            return None


    def _get_referenced_works_openalex(self, response_data):
        """Function that extracts the referenced works from the
OpenAlex API response. Used in the _get_openalex_data function. """

        referenced_works_list = []
        referenced_works = response_data.get('referenced_works', None)
        if referenced_works is not None:
            for c in referenced_works:
                work_id = c.replace("https://openalex.org/", "")
                referenced_works_list.append(work_id)
        return referenced_works_list


    def _filter_section(self, header):
        """Filter the articles that are not relevant to the search."""

        lower_header = header.lower().replace('\n', '')
        non_relevant_sections = ["workshop", "tutorial", "keynote",
"panel", "poster", "demo", "doctoral", "posters", "short papers",
"demos"]
        if any(section in lower_header for section in
non_relevant_sections):
            return True
        return False
    def _filter_paper_title(self, title):
        """Filter the title of the paper"""

        pattern = r'^(Demo:|Poster:|Welcome Message)'
        coincidence = re.match(pattern, title)

        return bool(coincidence)


    def _filter_dblp_links(self, conf, link):
        """ Filters the dblp obtained links to match only the base
```

```
conference"""
        # socc is the only conference that has a different name in the
link
        # remebmer to add more conferences if needed
        if conf == "cloud": conf2 = "socc"
        else: conf2 = conf

        pattern =
rf"https://dblp.org/db/conf/{conf}/{conf2}\d{{4}}\.html"
        return bool(re.match(pattern, link))
```

Codi 20. BaseCrawler class()

Annex B: Codi Font Extended Crawler

```
from auxiliar import file
from auxiliar import thread
from crawler.base_crawler import BaseCrawler
import time
import logging
import requests
import sys
import threading

class ExtendedCrawler(BaseCrawler):
    def __init__(self, conferences, years, num_threads, output_dir):
        super().__init__(conferences, years, num_threads, output_dir)
        self.api_key = file.api_key_in_env()
        self.semaphore = threading.Semaphore(1)

    def crawl(self):
        initial_time = time.time()
        first_year, last_year = self.years
        for conf in self.conferences:
            global data_per_year
            data_per_year = {}

            data_dir = f"./data/base_crawler_data/{conf}_basic_data"
            if file.exists_file(data_dir):
                basic_data = file.load_json(data_dir)
            else:
                sys.exit(f"Error: The basic data for the conference
{conf} does not exist. Please run the base crawler first.")

            for year in range(first_year, last_year + 1):
                if not file.year_exists_in_file(year, data_per_year):
                    logging.info(f"(EXTENDED) - {year} data not
found.")

            threads = thread.Thread(self.num_threads)
            threads.run(self._get_paper_data, (basic_data, first_year,
last_year))

            file.save_json(f"{self.output_dir}/{conf}_extended_data",
```

```
data_per_year)

        final_time = time.time()
        minutes = (final_time - initial_time) / 60
        print(f"(EXTENDED) - Done in {minutes:.3f} minutes")

    def _get_paper_data(self, data, start_year, end_year):
        """Function that gets the paper data for a range of years. It
uses the _get_paper_s2_data_request and _get_openalex_data functions
to get the data. """

        for year in range(start_year, end_year + 1):
            paper_data = []
            if str(year) not in data:
                continue
            for elem in data[str(year)]:
                paper_title = elem['Title']
                paper_doi_num = elem['DOI Number']
                paper_pub_year = elem['Year']
                paper_openalex_link = elem['OpenAlex Link']
                authors_institutions = elem['Authors and
Institutions']
                referenced_works = elem['OpenAlex Referenced Works']

                s2_data = self._get_s2_paper_data(paper_title,
paper_doi_num)
                paper_id, abstract, tldr, embedding, citations_s2,
doi_s2 = s2_data if s2_data is not None else (None, None, None, None,
None, None)

                if doi_s2 is not None and paper_openalex_link is None:
                    openalex_data =
super()._get_openalex_data(f"https://api.openalex.org/works/https://do
i.org/{doi_s2}")
                    paper_doi_num, authors_institutions,
referenced_works = openalex_data if openalex_data is not None else
(None, None, None)

                paper_data.append ({
                    'Title': paper_title,
                    'Year': paper_pub_year,
```

```
                    'DOI Number': paper_doi_num,
                    'OpenAlex Link': paper_openalex_link,
                    'S2 Paper ID': paper_id,
                    'Authors and Institutions': authors_institutions,
                    'OpenAlex Referenced Works': referenced_works,
                    'Citations S2': citations_s2,
                    'Abstract': abstract,
                    'TLDR': tldr
                    #'Embedding': embedding
                })

            self.semaphore.acquire()
            data_per_year[year] = paper_data
            self.semaphore.release()


    def _get_s2_paper_data(self, title, doi):
        """This functions uses the Semantic Scholar API to get the
paper data. If the DOI is not provided, it will search for the paper
using the title. """

        data_results = None
        paper_data_query_params = {'fields':
'abstract,tldr,embedding,references,externalIds'}

        # if the DOI is provided, search for the paper using the DOI
        if doi is not None:
            url =
f'https://api.semanticscholar.org/graph/v1/paper/{doi}'
            response = self._make_request_func(url,
paper_data_query_params, self.api_key)

            time.sleep(1)
            if response.status_code == 429 or response.status_code ==
504:
                time.sleep(5)
                response = self._make_request_func(url,
paper_data_query_params, self.api_key)

            if response.status_code == 200:
                response_data = response.json()
```

```
                data_results = self._extract_s2_data(response_data)
            else:
                logging.error(f"(EXTENDED) - {response.status_code} in
request for paper {title}")

        # if the DOI is not provided, search for the paper using the
title
        if doi is None:
            url =
f"https://api.semanticscholar.org/graph/v1/paper/search"
            query_params = {
                'query': title,
                'limit': 1
            }
            response = self._make_request_func(url, query_params,
self.api_key)

            time.sleep(1)
            if response.status_code == 429 or response.status_code ==
504:
                time.sleep(5)
                response = self._make_request_func(url, query_params,
self.api_key)

            if response.status_code == 200:
                response_data = response.json().get('data', None)
                if response_data is None:
                    return None
                paper_id = response_data[0]['paperId']
                base_url =
f"https://api.semanticscholar.org/graph/v1/paper/{paper_id}"

                response = self._make_request_func(base_url,
paper_data_query_params, self.api_key)

                if response.status_code == 429 or response.status_code
== 504:
                    time.sleep(5)
                    response = self._make_request_func(base_url,
paper_data_query_params, self.api_key)
```

```
                if response.status_code == 200:
                    response_data = response.json()
                    data_results =
self._extract_s2_data(response_data)
                else:
                    logging.error(f"(EXTENDED) -
{response.status_code} in request for paper {title}")
            else:
                logging.error(f"(EXTENDED) - {response.status_code} in
request for paper {title}")

        if data_results:
            return data_results
        else:
            return None


    def _extract_s2_data(self, response_data):
        """Function that extracts the paper data from the Semantic
Scholar API response. Used in the _get_paper_s2_data_request function.
"""
        tldr = None
        embedding = None
        citations = None
        paper_id = response_data.get('paperId', None)
        abstract = response_data.get('abstract', None)
        tldr = response_data.get('tldr', None)
        ids = response_data.get("externalIds", None).get("DOI", None)
        if tldr is not None:
            tldr = tldr.get('text', None)
        embedding = response_data.get('embedding', None)
        if embedding is not None:
            embedding = embedding.get('vector', None)
        citations = response_data.get('references', None)

        return paper_id, abstract, tldr, embedding, citations, ids


    def _make_request_func(self, url, params, api_key):
        """Function that makes a request to an API and returns the
response data. Used in the _get_paper_s2_data_request function. """
```

```
        if api_key is not None:
            response = requests.get(url, params=params, headers={'x-
api-key': api_key})
        else:
            response = requests.get(url, params=params)

        return response
```

Codi 21. ExtendedCrawler class()

Annex C: Codi Font Citations Crawler

```
from crawler.base_crawler import BaseCrawler
import threading
import time
import sys
from auxiliar import file
from auxiliar import thread
import requests
import logging

all_citation_data = {}
papers_data = []
all_papers_id = {}

class CitationsCrawler(BaseCrawler):
    def __init__(self, conferences, years, num_threads, output_dir):
        super().__init__(conferences, years, num_threads, output_dir)
        self.semaphore_s2 = threading.Semaphore(1)
        self.semaphore_oa = threading.Semaphore(1)

    def crawl(self):
        initial_time = time.time()
        first_year, last_year = self.years
        for conf in self.conferences:
            global papers_data
            papers_data = []
            global all_citation_data
            all_citation_data = {}

            data_dir =
f"./data/extended_crawler_data/{conf}_extended_data"
            if file.exists_file(data_dir):
                extended_data = file.load_json(data_dir)
            else:
                sys.exit(f"Error: The extended data for the conference
{conf} does not exist. Please run the extended crawler first.")

            self._get_all_paper_data(self.conferences)

            threads = thread.Thread(self.num_threads)
```

```
            threads.run(self._search_citations_data, (extended_data,
first_year, last_year))

            threads.run(self._get_citation_data, (papers_data, 0,
len(papers_data)))

            file.save_json(f"{self.output_dir}/{conf}_extended_data",
all_citation_data)

        final_time = time.time()
        minutes = (final_time - initial_time) / 60
        print(f"(CITATIONS) - Done in {minutes:.3f} minutes")



    def _search_citations_data(self, data, start_year, end_year):
        papers = {}
        for year in range(start_year, end_year + 1):

            # obtain all the papers ids from the citations
            try:
                for paper in data.get(str(year), []):
                    citations = paper.get("Citations S2", [])

                    if citations:
                        paper_ids = [citation["paperId"] for citation
in citations if citation.get("paperId")]
                        papers[paper["Title"]] = paper_ids
            except KeyError:
                pass

        self._batch_request_s2(papers)

    def _batch_request_s2(self, papers):
        url_s2 =
"https://api.semanticscholar.org/graph/v1/paper/batch"
        #print("Making requests to Semantic Scholar API...")
        for title, citations in papers.items():
            responses = []
            citations_len = len(citations)
            num_iterations = citations_len // 500
            rest = num_iterations + 1 if citations_len % 500 > 0 else
```

```
num_iterations
            for j in range(rest):
                ini = j * 500
                fin = min((j + 1) * 500, citations_len)
                c = citations[ini:fin]
                r = requests.post(url_s2,
                    params={'fields':
'title,year,venue,externalIds,authors.name'},
                    json={"ids": c})
                time.sleep(1.25)
                if r.status_code == 200:
                    responses.append(r.json())
                else:
                    logging.error(f"(CITATIONS) - {r.status_code} in
request for paper {title}")

            self.semaphore_s2.acquire()
            papers_data.append({"Title": title, "Response":
responses})
            self.semaphore_s2.release()

    def _get_citation_data(self, data, start, end):
        #i = 0
        for elem in data[start:end]:
            cited_data = []
            main_paper_title = elem.get("Title", None)
            response = elem.get("Response", None)
            if response == []: continue

            for cited_paper in response:
                for c in cited_paper:
                    data = self._get_cited_paper_data(c)
                    cited_data.append(data)

            self.semaphore_oa.acquire()
            all_citation_data[main_paper_title] = cited_data
            self.semaphore_oa.release()


    def _get_cited_paper_data(self, cited_paper):
        url_openalex = "https://api.openalex.org/works/"
```

```
        # if there is no data, continue with the next paper
        if cited_paper is None:
            return

        authors = []
        title = cited_paper.get("title", None)
        paper_id = cited_paper.get("paperId", None)
        venue = cited_paper.get("venue", None)
        link = cited_paper.get("externalIds", None).get("DOI", None)
        year = cited_paper.get("year", None)

        # check if the paper is in the already existing data
        if paper_id in all_papers_id:
            authors = all_papers_id[paper_id]["Paper"]['Authors and
Institutions']
            venue = all_papers_id[paper_id]['Conference']
            year = all_papers_id[paper_id]["Paper"]['Year']
        elif link is not None:
            title_aux, authors =
self._get_openalex_data(f"{url_openalex}https://doi.org/{link}")
            if title_aux == "not found":
                title = title_aux
        else:
            auth = cited_paper["authors"]
            for a in auth:
                authors.append({"Author": a["name"], "Institutions":
None})

        return {"Title": title, "Authors": authors, "Venue": venue,
"Year": year}


    def _get_openalex_data(self, openalex_link):
        """Function that extracts the authors and institutions data
and the referenced works from the OpenAlex API."""

        response = requests.get(openalex_link)
        if response.status_code == 200:
            response_data = response.json()
            title = response_data.get('title', None)
            authors_institutions =
```

```
self._get_authors_and_institutions(response_data)
            return title, authors_institutions
        else:
            logging.error(f"(CITATIONS) - {response.status_code} in
request for link {openalex_link}")
            return  None, None


    def _get_authors_and_institutions(self, response_data):
        """This function extracts the authors and institutions data
from the OpenAlex API response. Used in the _get_openalex_data
function. """

        auth_data = []
        authors = response_data['authorships']
        if authors == []: return None
        for a in authors:
            institutions = []
            author_name = a['author']['display_name']
            author_institutions = a['institutions']
            for inst in author_institutions:
                institution_name = inst.get('display_name', None)
                institution_country = inst.get('country_code', None)
                institutions.append({'Institution Name':
institution_name, 'Country': institution_country})
            auth_data.append({'Author': author_name, 'Institutions':
institutions if institutions != [] else None})

        return auth_data


    def _get_all_paper_data(self, conferences):
        for conf in conferences:
            data =
file.load_json(f"./data/extended_crawler_data/{conf}_extended_data")
            for year, paper in data.items():
                for p in paper:
                    all_papers_id[p["S2 Paper ID"]] = {"Paper": p,
"Conference": conf}
```

Codi 22. CitationsCrawler class()

Annex D: Evolució dels Continents en les Conferències

El que es pot observar en les següents gràfiques és l'evolució de la distribució dels articles en continents al llarg dels anys analitzats. Les conclusions que es poden extreure en observar els resultats són que els articles Nord Americans semblen anar reduint-se en els últims anys per deixar pas a articles majoritàriament Asiàtics.

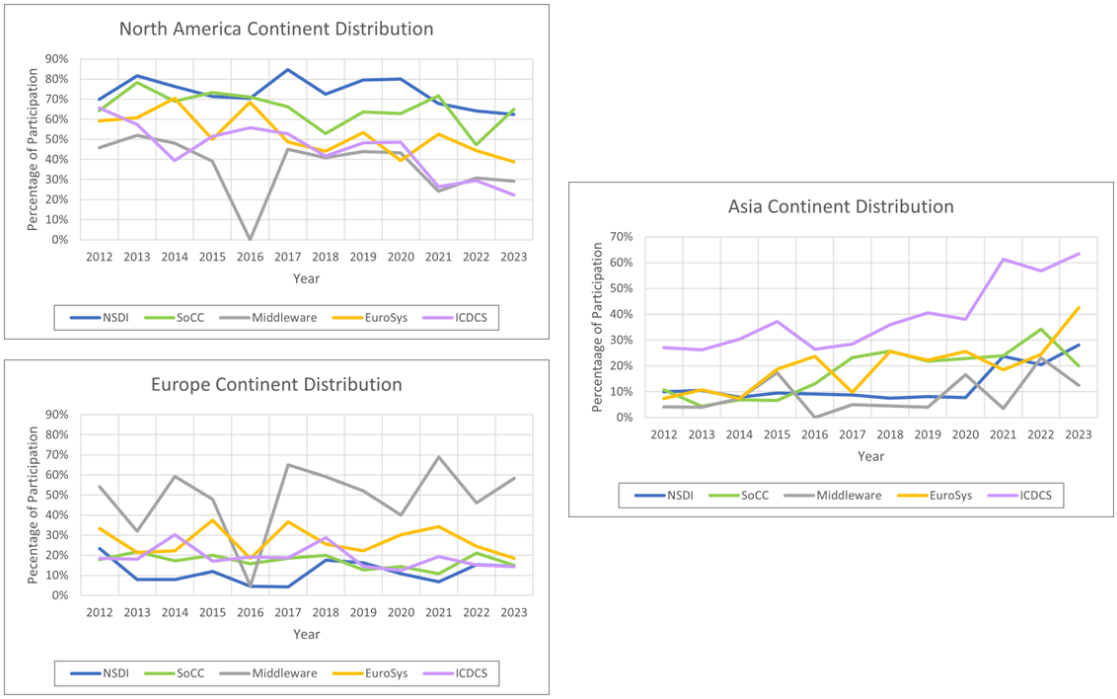


Figura 35. Distribució d'articles per continents al llarg dels anys 1

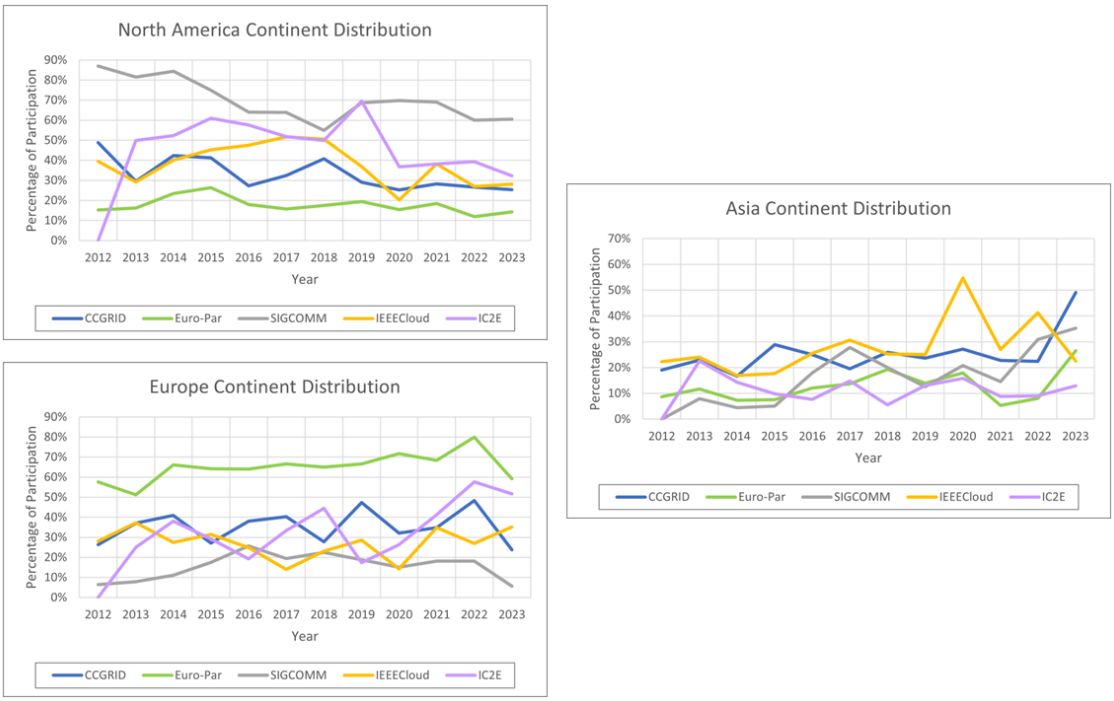


Figura 36. Distribució d'articles per continents al llarg dels anys 2