UNIVERSITY
OF EASTERN
FINLAND

University of Eastern Finland

School of Computing

Master's Thesis

Location-based web crawler for geotagged images retrieval

Nguyen Quang Khue

28th of May 2018

ABSTRACT

With the development of mobile devices with GPS sensor, there are more and more content embedded with geographic information, or geotagged content, published in the Internet. The use of geotagged content can be seen in many modern location-based applications on both web and mobile platforms. O-Mopsi is a location-based game whose gameplay features require player to reach real-world locations as fast as possible. It uses photos uploaded by community or game administrator to support player in identifying places they travel to and for creating new game content. The main issue of uploading location photos is it is exhaustive and time consuming. Thus, we developed a web crawler system in attempt to automatize the process of retrieving location photos on the Internet. Our system focuses on downloading geotagged images, which already contain GPS location in their metadata. In this thesis we introduce the O-Mopsi game and its content creation process in general. Then, we study the web crawler system, how it is structured, its working mechanism and crawling algorithms. The rest of the thesis focuses on the crawler system implementation, in which we describe in detail all its components, the method and algorithm we choose and how the system works in practice. In addition, we study in brief about geographical information retrieval for determining location information of non-geotagged images. At the end of the thesis, we present our experimental results and data collected from running the system on real production environment. We also compare three crawling algorithms: depth-first search, breadth-first search and best-first search to conclude which algorithm performs best.

Keywords: Location-based application, GPS, web crawler, geotagged images, location photos, web application.

ACKNOWLEDGEMENTS

I am grateful to University of Eastern Finland, Joensuu Campus and to all the teachers who helped me obtain new knowledge and experience in computer science, especially machine learning and clustering research areas. It was a good opportunity for me to be part of the IMPIT program where I had studied with students from different parts of the world and learnt from the diversity.

I would like to express my gratitude and great thank towards my supervisor, Professor Pasi Fränti, for his guidance and encouragements on my study and research. Without his dedications, useful comments, remarks and advices, it would have been impossible for me to finish this thesis on time. Through his teachings, I believe I have improved significantly my problem-solving, research and time management skills that will be useful for my future career.

I would like to give my thanks to every member of the Machine Learning research group, especially Dr. Radu Mariescu-Istodor and Dr. Najlah Gali for their supports in completing this thesis. I also want to give my thank to Mr. Juha Hakkarainen for his patience and the time he spent, whenever I requested his technical supports for deployment of the project associated to this thesis.

Finally, I want to thank my family, my friends and my fellow classmates Loc Cooc Khin and Nguyen Thu Linh for their emotion and mental support when I got stuck or needed reclusion. They are always by my side, supporting me any way they can. I appreciate all the discussion we had and all the good advices they provide that brought me to the completion of this thesis.

LIST OF ABBREVIATIONS

MIC	Mopsi Image Crawler
UEF	University of Eastern Finland
DFS	Depth-first search
BFS	Breadth-first search
BEFS	Best-first search
SQL	Structured Query Language
GPS	Global Positioning System
A-GPS	Assisted GPS
S-GPS	Synthetic GPS
EXIF	Exchangeable Image File Format
LBS	Location-based Service
LBG	Location-based Game
URL	Uniform Resource Locator
HTML	Hyper Text Markup Language
DOM	Document Object Model
SHA256	Secure Hash Algorithm 256
GUI	Graphical User Interface
API	Application Programming Interface
REST	Representational State Transfer
GIR	Geographic Information Retrieval
JSON	Javascript Object Notation

CONTENTS

1 INTRODUCTION

In the last decade, there has been a significant growth in mobile device usage, in which smartphone is the most popular device. According to statistics collected by eMarketer[1], the total number of smartphone users is expected to increase from 2.1 billion in 2016 to more than 2.8 billion in 2020. Of all features of smartphone, tracking geographical data is crucial and considered the indispensable feature in almost every phone. It is estimated that at least 10 systems are in use or being developed for location detection[2]. Within the scope of this thesis, we will describe in brief two most popular systems, which are *GPS* and *Wi-Fi*.

GPS stands for Global Positioning System which was developed by U.S Department of Defense in 1973, first introduced to cellular phones in the late 1990s and best-known for its capability to detect outdoor user's location. It was developed as a global navigation system in the form of a constellation of satellites that sends geolocation and time information to any device equipped with a GPS receiver. The system is independent from any telephonic or internet reception, it provides data to the user whenever there is no blockage to the line of sight of four or more GPS satellites.

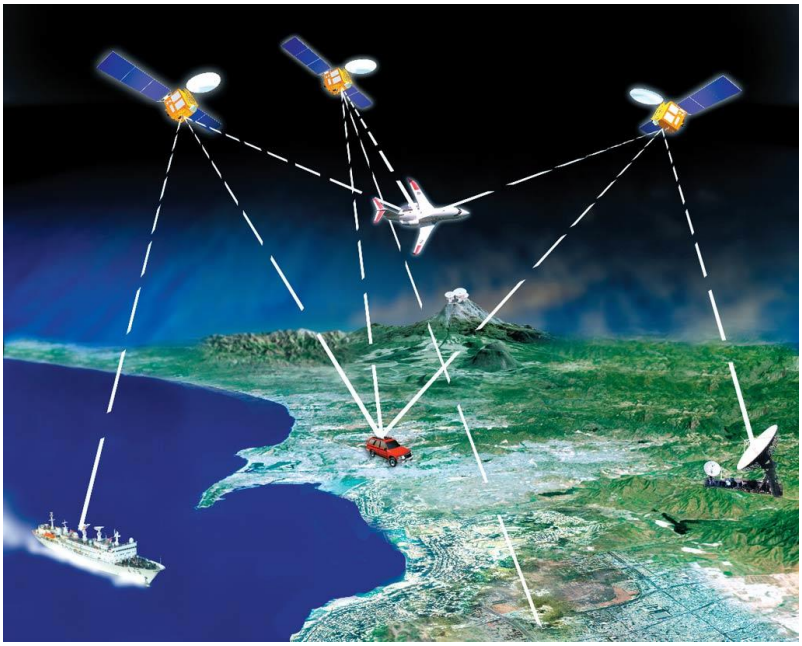

Figure 1: Illustration of the GPS satellite constellation[3]

GPS works well when user's device finds three or four satellites in the satellite constellation as illustrated in Figure 1. However, it is less effective or even not working when the user is indoor or in area surrounded by buildings that reflect satellite signals. There are two technologies that enhance the GPS system accuracy: *Assisted GPS (A-GPS)*[4] and *Synthetic GPS (S-GPS)*. Both A-GPS and S-GPS systems are implemented as application software in the device. A-GPS is a system that downloads orbital data from the internet via Wi-Fi or network connection when the device cannot directly receive signals from satellites. For instance, when the user is indoor and network connection is available, the device can

[1] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/
[2] https://www.pcworld.com/article/253354/ten_ways_your_smartphone_knows_where_you_are.html
[3] http://www.pcworld.com.vn/articles/cong-nghe/song-va-cong-nghe/2014/09/1235975/he-thong-dinh-vi-toan-cau-gps-hoat-dong-the-nao/
[4] http://www.gpsworld.com/wp-content/uploads/2012/09/gpsworld_Innovation_0302.pdf

get GPS information from the internet until the device can receive actual GPS signals. In contrast, S-GPS system calculates the satellites' locations in case the user is indoor and there is no internet connection. A device embedded with this technology can identify its location in under two seconds.

The Wi-Fi system uses internet access points to determine device's location. There are two methods used for determining location using Wi-Fi. *Received signal strength indication (RSSI)* is the most common method. Whenever user's device detects signals from nearby access points, RSSI takes the signals and matches against a database of access point locations. Based on the strength of the signals, RSSI can determine the distance between the device and the nearby access point, hence give user's location in relation to the access point. The second method for determine location using Wi-Fi is called *wireless fingerprinting*. Based on the pattern of Wi-Fi signals the device receives in a place, a "fingerprint" profile of the place is built. The collection of such fingerprints is later used for detecting location. The fingerprint can be created and stored to the device when the user visits a place for the first time, or it can be provided by a service provider. Figure 2 illustrates differences between GPS positioning and Wi-Fi positioning.

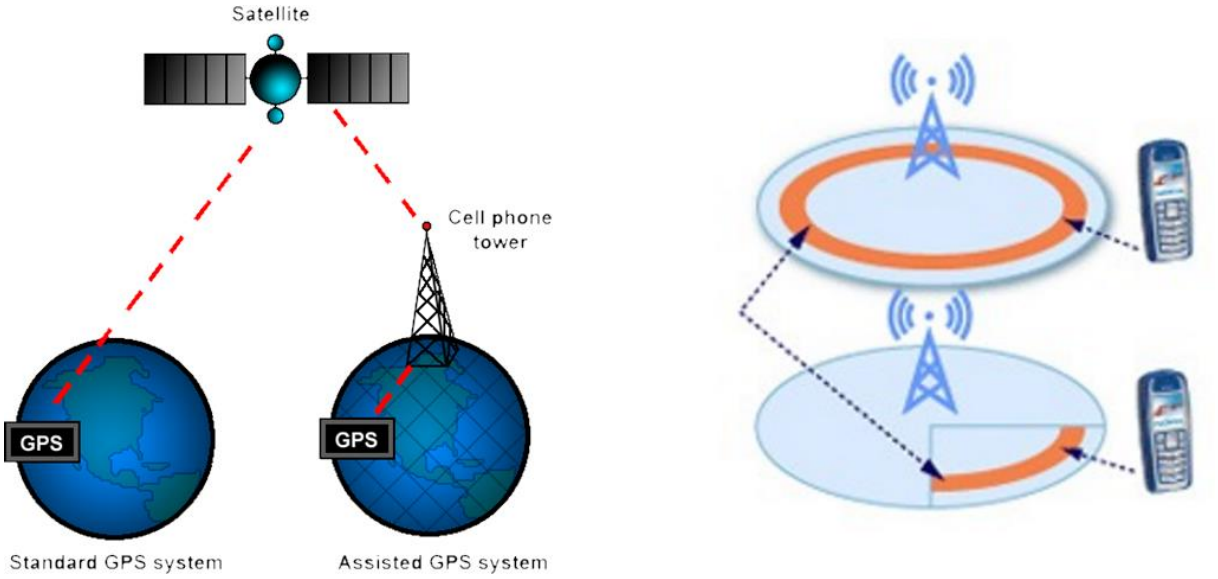

Figure 2: GPS positioning vs. Wi-Fi positioning[5]

The ability to track location of smartphone can be used to create content embedded with location information, of which digital photos are the most popular content. A g*eotagged photo* is a photo associated with geographical location. Whenever an image is captured by the smartphone, it automatically saves location information to the image's *metadata*, which is a set of information embedded inside the image's file to describe characteristics of the image. While there exist several types of image metadata, the most commonly used metadata type that contains geographic information is *Exchangeable Image File Format (EXIF)*[6]. Figure 3 shows an example of EXIF metadata with location information included in a photo image we captured in Joensuu, Finland.

[5] http://radupoenaru.com/gps-vs-agps-vs-wifi-vs-gsm-localization/
[6] https://www.photographymad.com/pages/view/exif-data-explained

EXIF DATA

Camera make :	Xiaomi
Camera model :	Redmi Note3
Date/Time :	2017/07/21 16:25:27
Resolution :	4608 x 3456
Flash used :	No
Focal length :	3.6mm (35mm equivalent 4mm)
Exposure time :	0.0009 s (1/1106)
Aperture :	f/2.0
ISO equiv. :	100
Whitebalance :	Auto
Metering Mode :	center weight
GPS Latitude :	N 62° 36' 45.5276"
GPS Longitude :	E 29° 41' 47.3544"
GPS Altitude :	94.00m

Figure 3: Example EXIF metadata from an image we captured, extracted using an online web application[7]

The above process of adding geographic information to metadata is called *geotagging*. Geographical coordinates, known as *latitude* and *longitude*, are the minimum required location information needed for geotagging. GPS data and geotagged photos are valuable content for location-based software as they help visualizing locations on map, which encourage user to use the application or visit the real locations. There are two types of location-based software which are *Location-based Service* and *Location-based Game*.

Location-based Service (LBS) is a service or an application software that uses geographical position of mobile device for operation of its features [1]. The usage of LBS systems are mostly practical, especially in business and tourism areas[8]. Some examples of LBS in daily life includes: finding shopping locations, providing traffic updates and weather reports, suggesting tourist attractions or even credit card fraud prevention. Nowadays, there are many location-based applications available as web application or mobile application which use GPS data. Figure 4 illustrates a use case of Google Trip[9] mobile application, in which user can use their own location as input to request for suggestions of nearby places to visit. The application uses geotagged photos other users contributed to help provide good visual information about the suggested place.

[7] https://www.verexif.com/en/ver.php
[8] https://www.businessnewsdaily.com/5386-location-based-services.html
[9] https://get.google.com/trips/

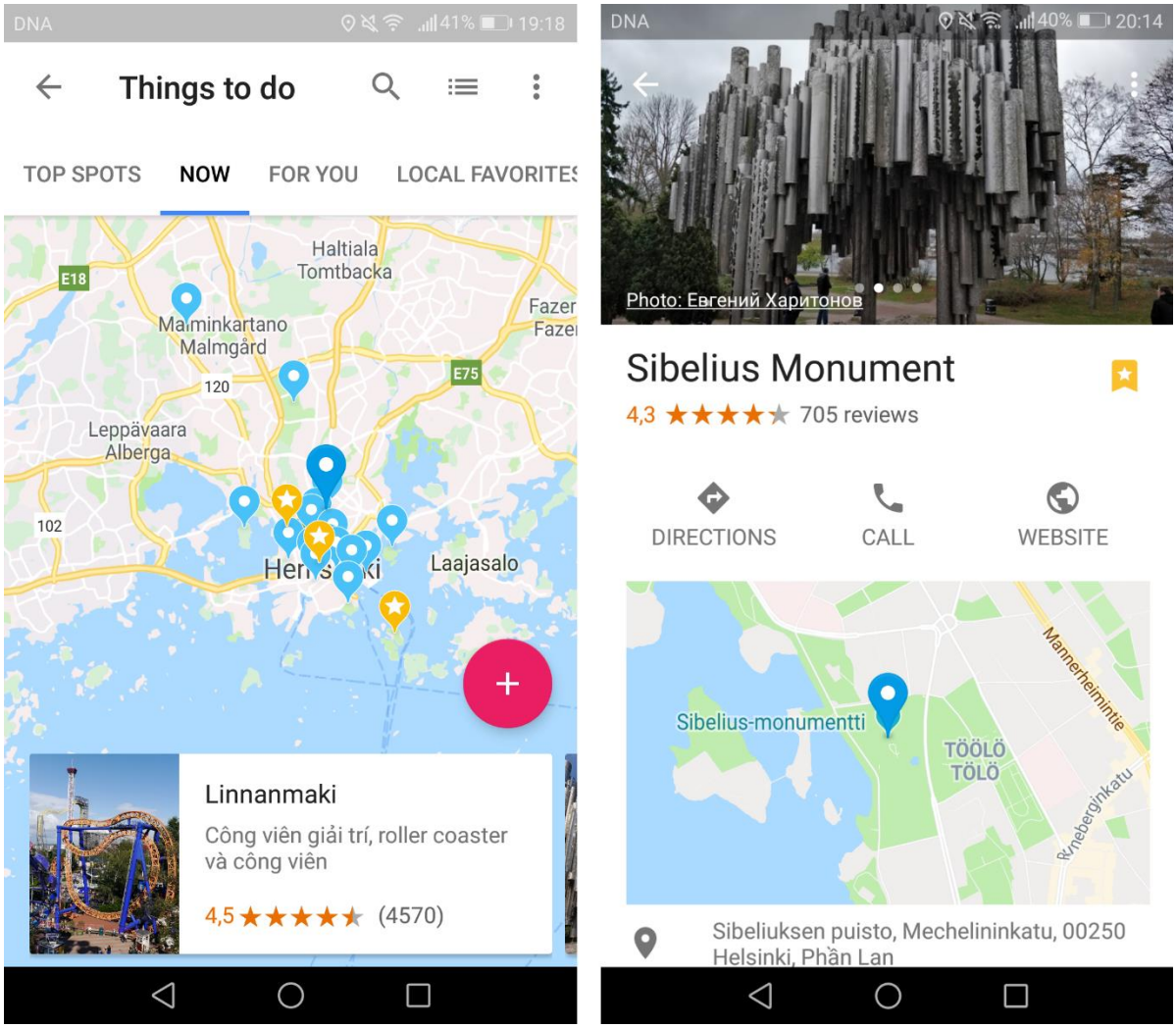

Figure 4: Example of Google Trip mobile application for suggesting things to do near Vantaa, Finland (left) and example of user photo contributed to google trip location information (right)

Location-based game (LBG) is a type of pervasive game whose gameplay evolves and progresses based on the player location. Most modern LBGs are implemented for mobile device with extensive use of GPS sensor to determine the location. Some notable LBGs on the market recently are *Pokemon* GO[10], *CodeRunner*[11], *Ingress*[12] and *Zombies, Run*[13]. Player experience in the LBG depends mostly on how well location data are used to represent the physical world. As players of the game must travel in the real world to make progress, visual information of the locations is essential for many LBG.

O-Mopsi [2] is a location-based game that is based on the classical concept of orienteering and exploits user geotagged photo collection[14]. The game is currently available as mobile client on four different mobile platforms including Nokia Symbian, Windows phone, Android and iOS. In O-Mopsi, a game is created by specifying a set of targets for the user to visit to complete the game [3]. The mobile client can plot targets for travelling, display compass data and give audio clue in different pitch and frequency about the distance between player and the target. Photos are used for aiding the user in identifying the target. Since O-Mopsi uses real world locations as its targets, gathering location photos for the

[10] https://www.pokemongo.com/
[11] https://itunes.apple.com/us/app/coderunner/id463639902?mt=8
[12] https://www.ingress.com/
[13] https://zombiesrungame.com/
[14] http://cs.uef.fi/mopsi/photos/

game content creation can be a challenging task. Currently, the game depends on its user and system administrators for uploading the location photos, which consume considerable time and effort.

To find solution for the photo content issue of O-Mopsi, we have developed a system called *Mopsi Image Crawler (MIC)*[15] for automated retrieval of geotagged photos on the Internet. Our system is developed based on the concept of *web crawler*, a software application that systematically browses the World Wide Web and download content. The photos collected by our system are expected to provide material for the O-Mopsi game's content creation. Currently, the system is running automatically on University of Eastern Finland's Mopsi server with the web GUI accessible through subdomain of Mopsi system website[16]. At the time of writing, the system visited 29788 URLs and downloaded 65525 of images, of which there are 571 geotagged images from places around the world.

We organize the remainder of the thesis as follows. In Section 2, we give an overview of the O-Mopsi game and challenges in its content creation process. In Section 3, we provide background information about the concept of web crawler. We will describe in detail about our MIC system in Section 4, with the information about the system architecture, its components and method we use for crawling. In Section 5, we provide additional description about Geo Information Retrieval, which can be useful for future development of the system for automatic geotagging of photo content. Finally, in Section 6, we show our experiment results and draw final conclusions in Section 7.

2 O-MOPSI GAME

O-Mopsi is a mobile location-aware gaming system, which is based on the classical concept of orienteering. The game was first introduced in Science Festival (SciFest) in Joensuu, Finland and has been played since 2011. Its main target audience is school kids. The festival helps them get familiar with science. Overall experience and feedback shows the game's potential to become popular in the future. Figure 5 shows the user interface of O-Mopsi game on Android platform before starting the game and while playing.

[15] http://mopsi.uef.fi/crawler/
[16] http://mopsi.uef.fi/

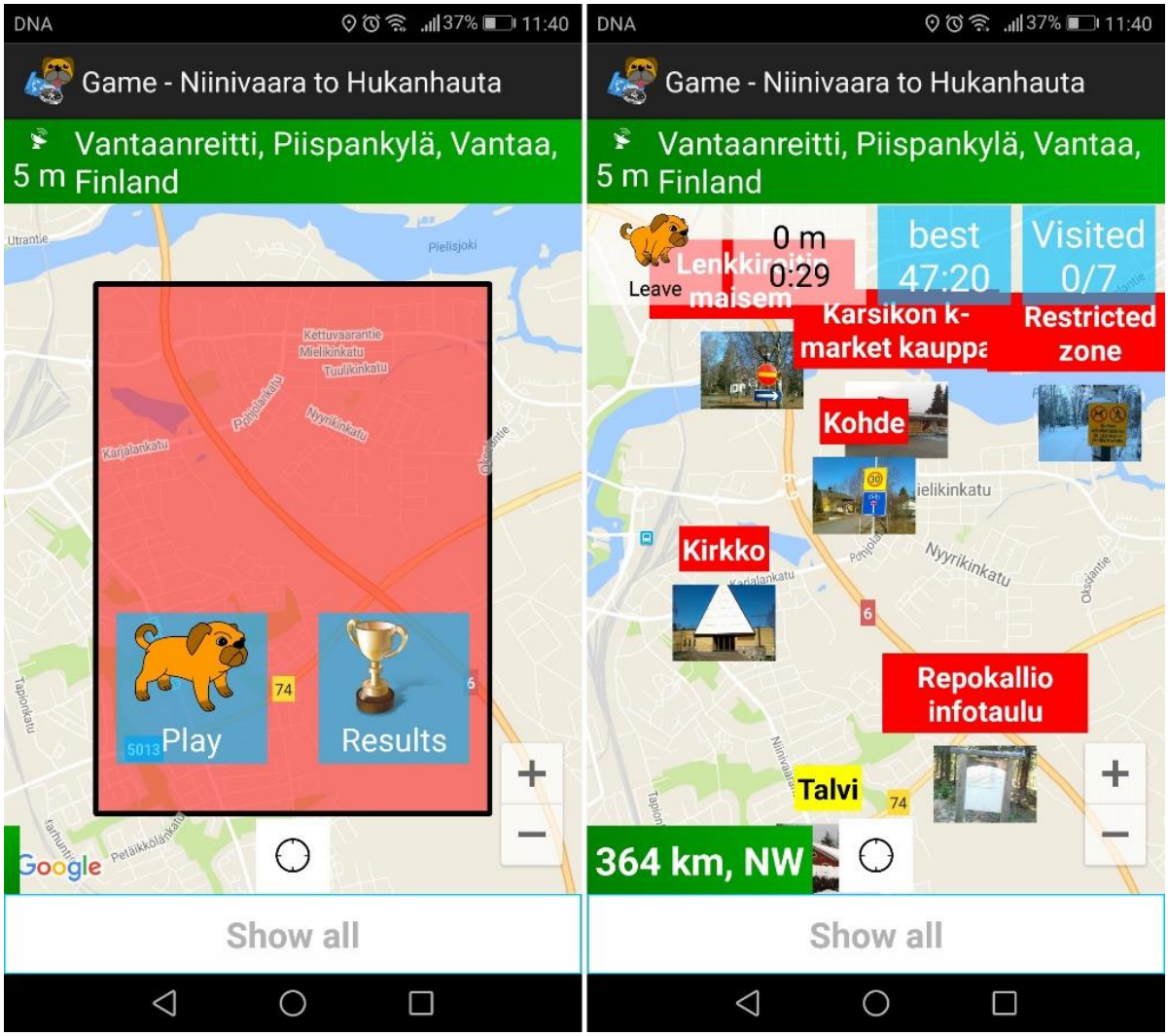

Figure 5: User interface of O-Mopsi on Android platform before starting the game (left) and while playing (right)

To initialize a new game, a player can either choose from one of the game made by other players or create their own. A game is an unordered set of goals, which are real-world locations on a map. There are three playing scenarios:

- Competition
- Educational
- Sight-seeing

Among the three modes, competition mode has been already implemented and the other modes are under consideration. In competition mode, the player completes a game by travelling to the designated location by either walking, using private vehicle, taking public transport or any means necessary. To get top rank the player also needs to travel faster than other.

While being developed as a game, O-Mopsi is not all about competition but also sharing travelling experience. The idea of sight-seeing mode is based on assumption that when people travel, they might not always interest in standard tourist attractions but thematic tours such as discovering historical places, viewing natural sceneries or simply enjoying top restaurants in town. O-Mopsi's players would create tours for others to choose based on their own interest or friend recommendations. A sight-seeing tour can be made by suggesting the order of travel using the same start and end location or

point-to-point trips between locations. Such sharable tours are also known as *game scenarios,* which are crucial content of the O-Mopsi system.

From the description of O-Mopsi, it is clear that unlike other location-based games such as Pokémon Go[17] or Ingress[18], O-Mopsi game's targets are not artificial creatures or computer-generated areas but real-world locations. Multiple targets can be organized into different *game scenarios*, which are the sharable tours discussed previously. Thus, quality of the targets is crucial for creating content that motivate user to play O-Mopsi. A target in O-Mopsi is created from three entities: *name, photo* and *location* in the form of geographical coordinates. According to [2], the quality of a target depends on:

- Quality of its photos
- Accessibility of the location
- Attractiveness of the location

Quality of a photo can be determined by multiple factors such as whether it is the right image of the target's location or the amount of detail in the photo that describe the location. Accessibility of the target's location depends on its geographic coordinates, for example a temple on top of a mountain is harder to reach than a restaurant in urban area. Finally, whether the target's location is attractive enough depends on its public popularity, how frequent it is visited and recommended by people travelling there. All of the three factors are important for creating good content for the O-Mopsi system.

When creating game scenarios manually or automatically, the biggest challenge is to establish a huge database of potential targets on earth as it is the playground of the game. There are three possible approaches to the challenge of collecting material for target creation:

- Collect the material manually
- Utilize existing geotagged database
- Use web mining to search for material
- Use a web crawler to download material from the web

The first approach is time-consuming since it requires human efforts to travel and gather information in different parts of the world. Contribution from community is also possible in this approach but it would introduce another challenge of managing quality of uploaded material. The second approach is to use existing geo-tagged databases of photos from various providers. Currently, geo-tagged data from AViewOnCities[19] have been permitted to be used in O-Mopi, but the database lacks diversity since it only convers a few European cities. The third approach is to search as much targets as possible around a given approximate location where a new game is initialized. Only the name or the address of the place is required for searching for targets. However, the amount of information returned by a web search is huge and determining which information is relevant for constructing potential targets can be challenging.

In this thesis, we focus on the last approach: to build a system that downloads material from web content automatically. Similar system called *Never Ending Image Learner (NEIL)* has been developed in [4] that utilizes semi-supervised learning techniques to find images of certain categories, based on visual knowledge base. The visual knowledge base of NEIL consists labeled examples of three

[17] https://www.pokemongo.com
[18] https://www.ingress.com/
[19] http://www.aviewoncities.com/

categories *Object* (e.g. car, road, ball), *Scene* (e.g. beach, mountain, forest) and *Attribute* (e.g. color or size), and their four types of relationships including:

1. Object-Object (e.g. keyboard is a part of computer)
2. Object-Attribute (e.g. the cup is big)
3. Scene-Object (e.g. a cow is on the rice field)
4. Scene-Attribute (e.g. the alley is narrow)

The NEIL's semi-supervised algorithm has four main steps. The first step is to build classifiers for the mentioned categories. It uses Google Image Search to find and download thousands of images. In the second step, the visual data of the images are clustered into categories using affinity propagation algorithm. Three-quarters of the clustered images are used for training classifier and the remaining quarter is used as validation set. In the third step, the trained classifier automatically discovers the relationships between the image categories and use such information to find new instances of different objects and scene categories. In the fourth step, the new instances are used as updated data for training new classifier. The entire process repeats and continues forever.

While useful for retrieving objects and scene images, the NEIL system is complicated in structure and dependent on initial image data with limited categories and no consideration for location-based data. Our system is simpler, it aims to get as many images as possible, regardless of their topic, from web content with focus on those that are geotagged. The web crawler system has some benefits over other approaches. First of all, it requires less human interaction or manual efforts to collect data like the first approach. Secondly, system architecture and crawling algorithm is customizable, which gives us some control of the method used for collecting data. Third, there is an obvious increment in usage of geo-tagged data for web pages. From 2004, only less than 0.1% of Finnish websites used the meta tag for geo-coordinates according to study in [5], in another report 4 years later this number rose to nearly 1% [6]. In 2017 [2], already 7% of the web pages had geo-coordinates in their online data. The growth in size of web content guarantees that this number will continue to increase in the future. Furthermore, modern websites tend to use more image content, including digital photos taken by camera or mobile devices with GPS functionality. Thus, downloading images on the web may provide better chance to collect good quality and relevant material. Regardless, there are challenges existing in this approach relating to the following factors:

- Performance of the crawler
- Quality of downloaded images
- Potential copyright issues

The goal of this thesis is to apply the concept of web crawler for traversing the web and gathering required material for the O-Mopsi game, which will be discussed in the next section of this thesis.

3 WEB CRAWLER

A *web crawler* has been called by different names such as *Web spider*[20], *ant*, *automatic indexer* [7] or *Web scutter* in *Friend of a Friend (FOAF)*[21] software context. It is defined as *a system for the bulk downloading of web pages* [8] or *a program that retrieves Web pages* [9]. Elyasir et al. [10] defined it as a tool to download web pages or their partial content in an automated manner. Web crawler basically operates based on a simple graph search algorithm, such as depth-first or breadth-first search,

[20] https://web.archive.org/web/20040903174942/archive.ncsa.uiuc.edu/SDG/IT94/Proceedings/Agents/spetka/spetka.html
[21] http://xmlns.com/foaf/spec/

with the assumption that all web pages are linked together and there are no multiple links and nodes [11].

There are three main applications of web crawler, which are:

- Web indexing
- Web archiving
- Web data mining

Web indexing is a collection of methods for collecting and organizing content of a website or the Internet systematically according to a set of predefined criteria[22]. It is the core of modern search engines, where their base components are built on. In search engines, the web crawler component browses web pages systematically, and organizes visited pages by keywords or metadata and store them to a local repository or a database. Instead of real-time search, when receiving input from users, the search engines analyses their indexed information to produce a list of matching web pages [11].

A closely related application to web indexing is *web archiving*, a process in which copies of website content are periodically gathered and preserved for different purposes [12]. Like web indexing, web archiving systems utilize web crawlers for capturing massive amount of web content automatically.

Web data mining systems extract statistical properties of web pages. Data collected from the mining process are analyzed to reveal information of interests for business, e.g. copyright infringements or marketing trends [8]. In web data mining systems, a web crawler does not only handle retrieval of raw web content but also perform additional steps to obtain statistical information, such as counting frequency of a set of terms in the web page, collecting the number of visitors or analyzing the balance between text and multimedia content in a web page based on percentage of each content type over a website or the whole web.

More often, crawling techniques and algorithms are kept as business secrets and not release to the public [10], which means there exists many research possibilities surrounding web crawler. Meanwhile, thanks to the development of mobile phones and digital cameras with ability to keep track of user's location, uploading image content with location information becomes a frequent activity for the Internet users. Thus, research on web crawler for location-based image retrieval helps enhancing current applications in social network, education, tourism and entertainment.

3.1 Architecture of Web Crawler

Standard web crawler consists of four main components [13] as illustrated in Figure 6:

1. The queue
2. The downloader
3. The scheduler
4. The storage

[22] American Society for Indexing, *"Indexing the Web"* at https://www.asindexing.org/reference-shelf/indexing-the-web/ (Accessed 24th August 2017)

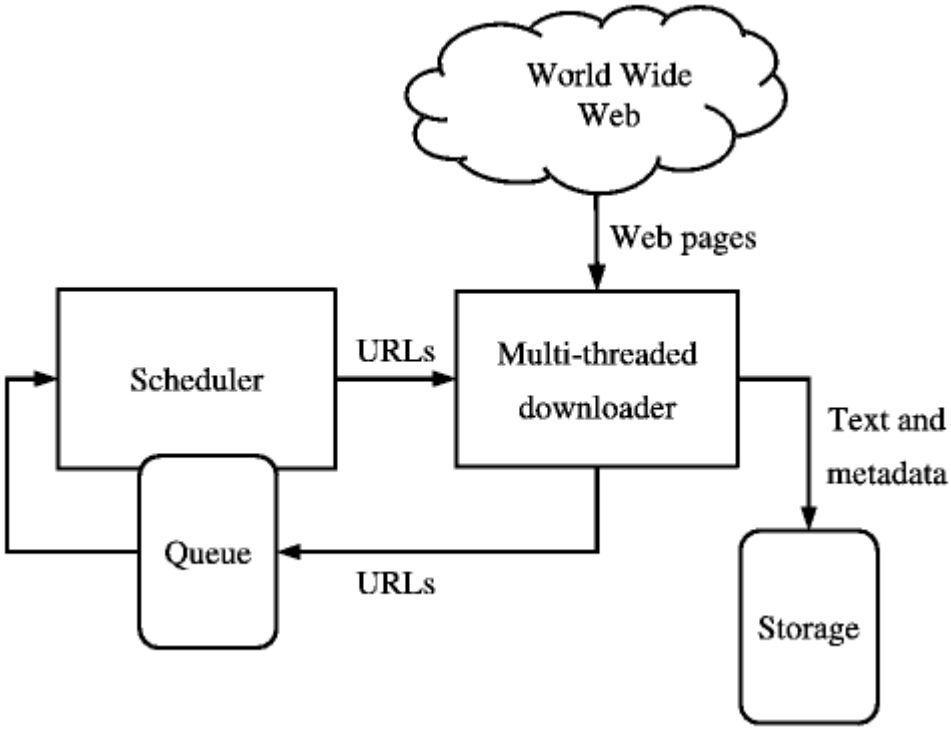

Figure 6: High-level architecture of a standard web crawler [13]

The queue is a data structure that stores a list of URLs that link to different web pages. Within web crawler context, the queue performs two operations which are adding an input URL into the queue and getting the next URL for continue the crawling process. In practice, the queue can be either a normal queue or a priority queue. A normal queue organizes the URLs following the *first in, first out (FIFO)*[23] method, in which URL that is put to the queue first is chosen as the next URL to continue the crawling process. Meanwhile, a priority queue organizes the URLs by a *priority* value, meaning that the next is always the highest-priority URL. Priority of a URL is determined by predefined criteria, it can be either the URL's relevance to the content of crawled web pages or how frequent its associated web page is visited. The choice of normal or priority queue depends on the crawling algorithm used for implementation and the purpose of the web crawler.

The downloader is central component of the crawling system as it performs actual crawling task. It is basically a program that performs crawling algorithms such as depth-first search, breadth-first search or similar alternatives to explore the Internet for web pages and downloads their content recursively. Given a set of URLs as input, the downloader retrieves content of their associated web pages and save them to corresponding physical files as output. Its working principle follows the basic crawling mechanism, which will be described in the next part of this section. The downloader may contain additional sub-components to read *Hyper-Text Markup Language (HTML)*[24] or perform processing steps to produce meaningful output data.

It is technically possible for the crawling process to run indefinitely. However, the environment in which the crawler operates in usually has certain constraints on computing resources, such as processing power, amount of system storage or memory. Thus, in practice the whole crawling process is not continuous but a combination of smaller crawling tasks. *The scheduler* is a program that manage

[23] https://techterms.com/definition/fifo
[24] https://www.w3.org/TR/html/

execution of the crawling tasks. It ensures that there are adequate computing resources for the crawling process to continue and decides when the next task in the crawling process happens. There are three inputs for the scheduler. The first input is status of crawling process from the web crawler, which provides information on whether a crawling task is running or not. The second input are current system resource usages, which is provided by the operating system. The final input is a set of predefined schedules for continuing the current crawling task or executing subsequent crawling tasks. If system resources are inadequate, the scheduler terminates the current crawling task if it is running or ignore execution of subsequent crawling tasks, which in turn stops the whole crawling process completely. It should be noted that the last input is optional as the scheduler can execute the next crawling task as soon as system resources are available or it can be implemented to setup schedules automatically.

The storage is a data structure that can store and manage result data from the crawling process. The result data can be text content, metadata or multimedia resources. In practice, the storage can either be the file system or the database system connect to physical data storage devices such as hard disk, physical memory unit or magnetic tape. Modern web crawler often uses database systems capable of managing an enormous collection of data.

3.2 Working Mechanism of Web Crawler

Before going into details about how the web crawler works, it is good that we go through some basic concepts of the Internet and World Wide Web. The *World Wide Web (WWW)* or *the web* is defined as "an information space in which the items of interest, referred to as resources, are identified by global identifiers called *Uniform Resource Identifiers (URI)*" [14]. The web is commonly visualized as a huge graph or tree structure named the *web graph*. A web graph is a directed graph, in which each vertex or *web node* represents a web resource. Web nodes are connected by *hyperlinks*, usually referred to as *links*, which are pointers to their URL references. Thus, in the web graph context the term hyperlink, link or URL are interchangeable in this thesis. Illustration of web graph can be seen in Figure 7.

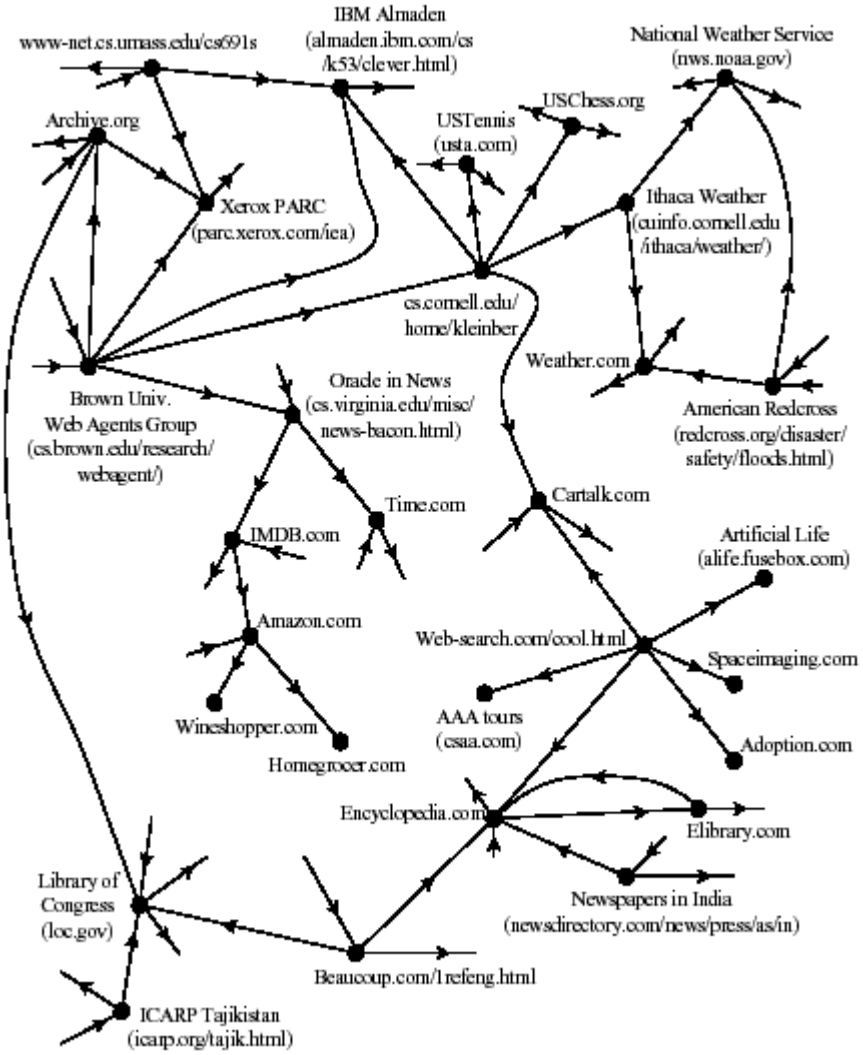

Figure 7: Illustration of a web graph[25]

Being an information space, the web contains a huge amount of *resources*. When a resource is given an identifier, it is called a *web resource* [15]. Examples of resources are text documents, digital images, e-mail messages, programming scripts or audio files. Within the scope of this thesis, only two types of web resources are considered which are web pages and images.

The most important and commonly accessed web resources are *hypertext* documents [16]. Hypertext documents on the web are called *web pages*, which are typically written in a structural language named *Hypertext Markup Language* (HTML). A set of related web pages forms a *website*. Like any other web resources, each web page is identified by a *Uniform Resource Locator (URL)*[26], also known as *web address* which is a unique text used for accessing the web page through web browser software.

Based on the high-level architecture described in the previous part of this section, the general working mechanism of a web crawler can be described as follows: the crawler receives a list of URLs as input, also known as the *seeds*, and add them into a priority queue. For each of the seeds, the crawler scans through the corresponding web page to collect content of interest. Scanned pages are then indexed by a *client* which is also responsible for saving, summarizing or analysing crawled content [9]. A

[25] http://www-inst.eecs.berkeley.edu/~cs61bl/r//cur/graphs/world-wide-web.html?topic=lab24.topic&step=6&course
[26] http://searchnetworking.techtarget.com/definition/URL

crawling task generally does not terminate as long as the crawler is designed to follow all links in every web page it crawls. Such design may introduce resource constraint issue as operating environment of the crawler often has limited storage, memory and processing power. Hence, web crawlers are usually designed to stop current crawling task when reaching a certain threshold or a resource usage limit manually defined by system administrator. In practice, the mentioned threshold can be a limit on number of web pages for each crawling task or how deep the crawler should navigate the web structure.

Once the crawler completes a crawling task, the scheduler component queries for operating environment's resources to decide whether the system can continue the crawling process. Should the crawling process continue, the scheduler plans the next task in advance. At a specific timestamp after the termination of previous crawling task, the scheduler executes a new task with a new input seed obtained from the seed queue, which is the URL with highest priority, and repeat the whole crawling process until the seed queue is empty.

Since web pages are connected interchangeably, it is possible for the crawler to retrieve the same web page more than once. Additionally, there exists the possibility that multiple URLs link to a same web page. To avoid retrieving the same web page twice, the crawler can be implemented such that it maintains a cache of URLs and possibly page content for checking content similarity or use different hashing techniques [17].

3.3 Types of Web Crawler

Based on functionality, we can classify web crawlers into two main types [10]:

- Generic web crawler
- Focused web crawler

Generic web crawler tends to traverse the web graph in all directions and retrieves as much web resources as possible. Due to the termination point is unknown, the generic web crawler stops only when it reaches all pages of the web and criteria to halt the crawling process must be defined manually. Given a web page URL as *seed*, the crawler analyses its content and extracts every URL on the page and puts them into its queue without concern about how relevant the content of the URL's associated page is to the previously crawled pages. The next URL chosen to continue the crawling process is the first URL found on the page. There is no limit to the crawling scope, which means the crawler may get pages from websites outside of the seed page's domain.

The generic crawler is simple to implement with commonly used graph traversing algorithms such as breadth-first or depth-first. Since the crawler does not need to evaluate the relevance of each URL it collects, it requires less computational resources with high performance of execution. In addition, generic crawler is also highly customizable by changing the algorithm or crawling options. Despite the large quantity of output results, pages retrieved by the generic crawler do not necessarily have any relationship with each other. For instance, an online news may have links in different topics and to different websites on the same page, an article about sport thus may have links pointing to another article about movies. In practice, this means generic crawler can download data from any web pages, even ones with advertisements or adult content, which lead to lower quality of the crawling results.

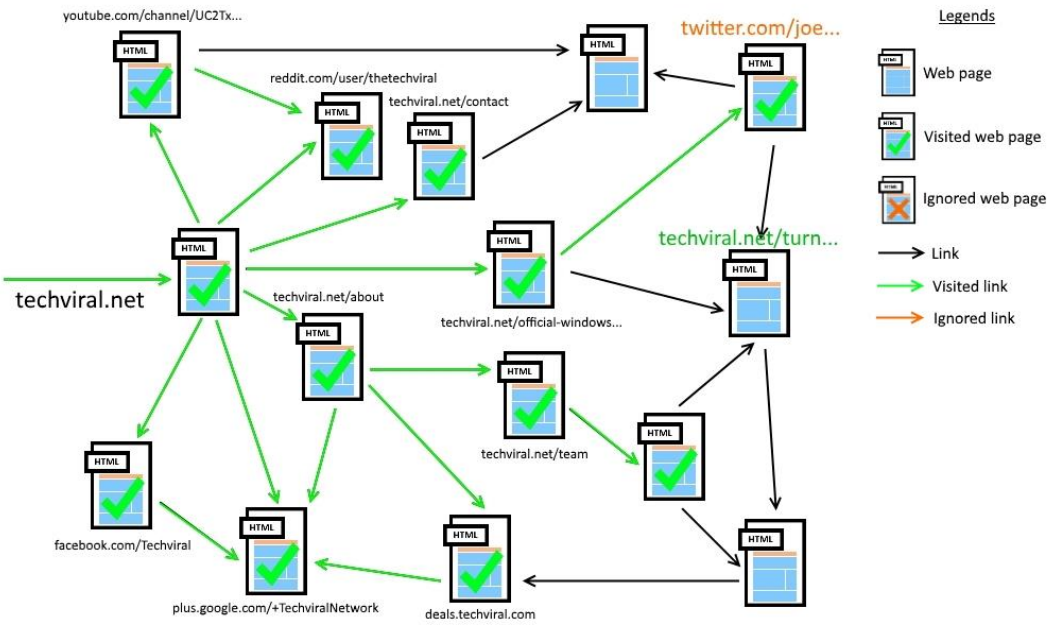

Figure 8: Illustration of generic web crawler

Unlike the generic web crawler, *focused web crawler* [18] explores the web based on a specific set of predefined rules. In contrast to the random crawling process of the traditional crawler, the focused crawler only retrieves content from pages relate to one another. The crawler evaluates each web page based on certain criteria and produce a weighted value or score that helps determining whether the page is relevant or important enough to be visited next. The crawler then simply chooses the page with highest relevance or importance score from the queue to continue the crawling task.

Focused crawler also provides more control over stopping criteria of a crawling task. For instance, a certain web page or a specific level of the web graph can be set as a goal to terminate a crawling task. In comparison to the generic crawler, the focused crawler has a more complex and less flexible system design that utilizes classification techniques for determining link's relevance [10]. The focused crawler thus requires higher computational resources. However, since focused crawler considers relationship between web pages, its final crawling results are often considered as higher in quality.

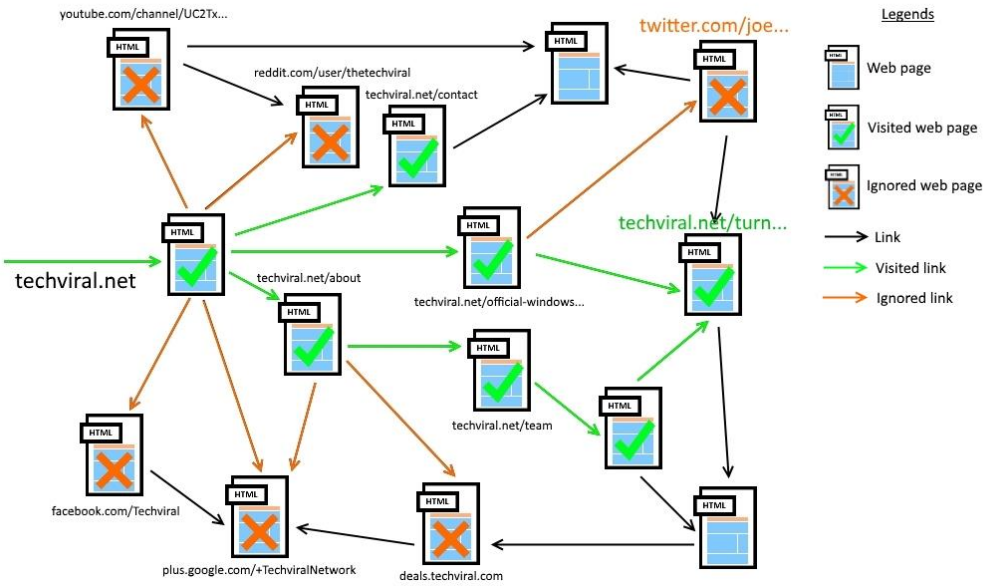

Figure 9: Illustration of focused web crawler

Figures 8 and 9 illustrate how generic and focused web crawler operate, given the home URL of Tech Viral[27] website as input seed. The arrowed lines represent the links between web pages, each link has a unique URL associated to it that leads to a specific web page. Unvisited links are displayed in black, green links are the chosen links for continuing the crawling process and orange links are irrelevant links ignored by the crawler. Crawled web pages are displayed with green check mark while pages that are not meant to be retrieved are marked with a red cross.

As can be seen from the illustrations, the generic crawler exhaustively traverses all web pages regardless of their host name and retrieves content outside the website's domain. As a result, links chosen for subsequent crawling tasks are scattered as all links are considered as relevant. Meanwhile, the focused crawler is configured to forward only links that lead to web pages with the same host name as the seed in this example. Thus, the links chosen for crawling are not scattered and the crawler gather more pages related to the seed. In case the crawling process is interfered and terminated in the middle, the last page crawled by generic crawler is likely to not have any relationship to the seed or previous pages, while it is not the case with focused crawler.

3.4 Crawling Algorithms

In the previous part of this section, we have discussed about two web crawler types which are generic and focused web crawlers. Even though having the same working mechanism, generic and focused crawlers are fundamentally different in the order of web pages they choose to visit when traversing the web graph. Such order is actually determined by the algorithm used for implementing the crawler. Two most commonly used algorithms for web crawlers are *breadth-first search* (BFS) and *depth-first search* (DFS).

3.4.1 Breadth-first search

BFS follows *first in, first out (FIFO)*[28] method for navigating the web graph. Given a link, a crawler implements BFS algorithm retrieves content of its associated web page and extracts links to other web pages. Each extracted link is appended to the end of the queue if not there already. Thus, the next link

[27] https://techviral.net/
[28] https://techterms.com/definition/fifo

chosen for continuing the crawling process is the neighbor link of the current link that points to a web page of the same level as current link's web page. While it is slow in terms of performance and memory-intensive[29], BFS provides efficient web graph traversal strategy. It ensures coverage of all possible web pages in each level of the graph, thus avoid getting trapped at one web branch forever. The worst time complexity of the BFS algorithm is $O(N)$, where N is the number of web pages the crawler visits. Figure 10 provides illustrations of breadth-first search.

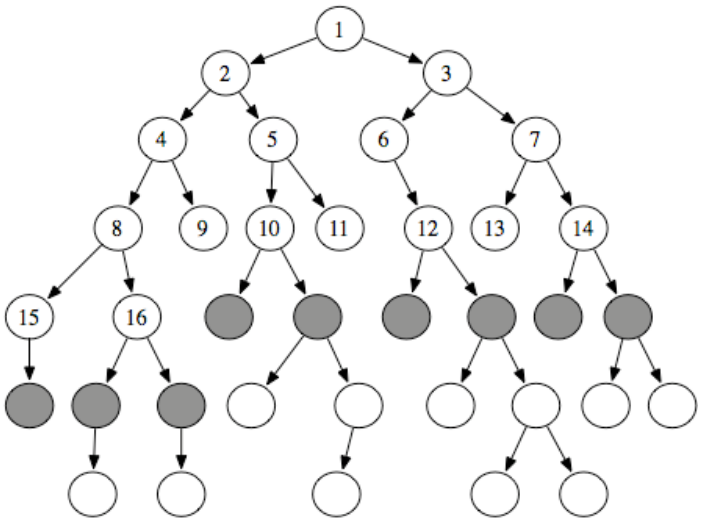

Figure 10: Illustration of breadth-first search[30]

3.4.2 Depth-first search

A crawler implementing DFS algorithm tends to travel deeper into the web hierarchy whenever possible. This method of traversing the web graph is known as *first in, last out (FILO)* method[31]. Instead of appending extracted links to the end of the queue, the crawler prepends them to the beginning of the queue, consequently prioritizes links to web pages of the same web branch. When the crawler is unable to get any unvisited link from the current web page or reaches the end of a web branch, it continues the crawling process with the next unvisited link in the queue, which points to web page of different web branch. In comparison to BFS, DFS is less memory-intensive as it needs to keep track of fewer visited links when traversing the web graph. However, if we set a web page as a goal and configure the crawler to stop when reaching the page, DFS may take longer time to finish a crawling task when the goal page is on some arbitrary branch in the middle of the web graph. The worst time complexity of DFS algorithm is $O(N + M)$ where N is the number of web pages the crawler visits and M is the number of links in the web graph. Figure 11 illustrates the depth-first search algorithm.

[29] http://intelligence.worldofcomputing.net/ai-search/breadth-first-search.html#.WpG-MKhubtW
[30] https://80legs.groovehq.com/knowledge_base/topics/how-80legs-crawls-urls-depth-first-vs-breadth-first-vs-greedy
[31] https://techterms.com/definition/filo

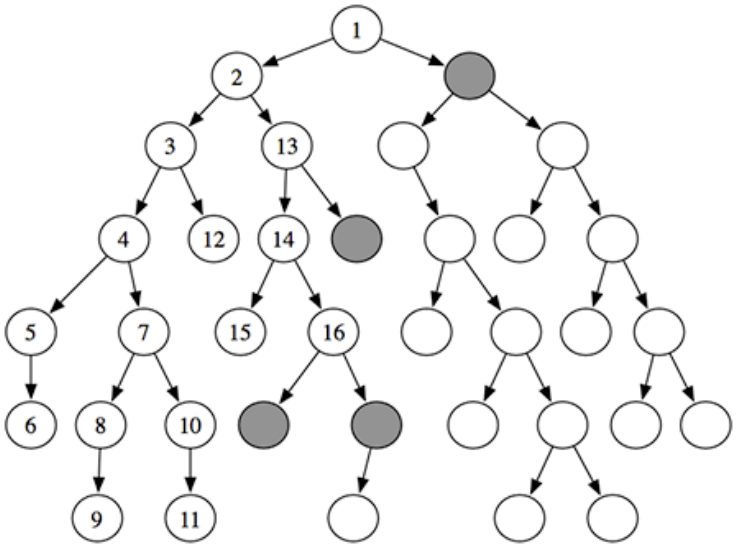

Figure 11: Illustration of depth-first search[32]

Thus far we have explored the two most common algorithms that are suitable for generic web crawlers. Both BFS and DFS are simple to implement and either algorithm may satisfy our system basic requirements for navigating the WWW. However, implementing a web crawler with either DFS or BFS algorithm at its core does not guarantee high-quality crawling results. We want our system to retrieve only web pages that have high possibility of containing images with location information, while neither algorithm considers the relation between web pages. Hence, the crawler is more likely to visit random web pages with unexpected content such as advertisements, pornographies or simply non-location content. We considered results from the crawler in such case as unimportant or low-quality.

3.4.3 Best-first search

Neither DFS nor BFS algorithm is suitable for our system due to the issue of low-quality crawling results mentioned above. We found many variants of either BFS or DFS algorithm such as *Fish Schools Search* [19] or BFS with Google's *Page Rank* [20] measurement can solve the above issue. However, Fish Schools Search is highly influenced by manually defined parameters [21], while applying PageRank for a large set of web data can be overly complicated[33].

In fact, the above issue can be solved by creating a function, known as a *heuristic*, to evaluate relevance between web pages. Then, instead of following FIFO or FILO ordering, the algorithm chooses the highest-priority in the queue to visit next. This method is known as *best-first search (BEFS)* [34], which is the algorithm we choose as the main algorithm of our crawler. DFS and BFS are also implemented but solely used for experimental purpose. The main advantage of BEFS is its simplicity in implementation and is not restricted to only exploring a small subset of the web graph like BFS or DFS. There is also no dead-end situation for BEFS as it makes it possible to continue with other links, as long as they are unvisited [22].

[32] https://80legs.groovehq.com/knowledge_base/topics/how-80legs-crawls-urls-depth-first-vs-breadth-first-vs-greedy
[33] http://www.cs.princeton.edu/~chazelle/courses/BIB/pagerank.htm
[34] https://courses.cs.washington.edu/courses/cse326/03su/homework/hw3/bestfirstsearch.html

Pseudocodes for our best-first crawling algorithm is shown in Figure 12. We will discuss in more details regarding how we apply the algorithm in Section 4 of this thesis.

```
function befsCrawl(seed, limit)
{
      // Initialize the queue
      createPriorityQueue();

      // Add the seed link as first link in the queue
      setRelevance(seed, 1.0);
      addToQueue(seed);

      // Keep track the number of downloaded pages
      noOfPages = 0;

      while (queueIsNotEmpty() and noOfPages < limit)
      {
            // Get the most relevant link out of the queue
            current = getMostRelevantLink(queue)

            // Download web page's content from the link
            page = requestWebPage(current);

            // Extract all links contain in the web page
            links = extractLinks(page);

            // For each extracted link
            for (i = 0; i < size(links); i++)
            {
                  // If the link has not been visited and not in queue
                  if (notVisited(links[i]) and notInQueue(links[i]))
                  {
                        // Calculate relevance of the link
                        relevance = calculateRelevance(page, links[i]);
                        setRelevance(links[i], relevance);

                        // Add the new link to the queue
                        addToQueue(links[i]); // Add the link to queue
                  }
            }

            // Mark current link as visited and repeat the procedure
            markAsVisited(current);
            noOfPages = noOfPages + 1;
      }
}
```

Figure 12: Pseudocodes of the best-first search algorithm with link relevance calculation

The worst time complexity for the BEFS algorithm is $O(N * logN)$, where N is the number of web pages the crawler visit. Note that the priority queue used for this algorithm is in a min or max heap structure so insert and remove operations take $O(logN)$ time. The overall performance of the algorithm depends largely on the performance of the relevance calculation function, which can make the algorithm slower than DFS and BFS. We will discuss about the relevance calculation function in

Section 4. In Section 6, we will compare the algorithms performance and quality of output results in one of the experiment.

4 MOPSI IMAGE CRAWLER

Mopsi Image Crawler (MIC) is an application implemented for automatic retrieval of location-based material for the O-Mopsi game's content creation. It aims to build a focused web crawler, whose target web resources are digital images with geographical information. The downloaded images and their location information are used for generating goals for the O-Mopsi game, thus reducing the manual works for the game's administrators. Integration to O-Mopsi game platform can be done via client-server HTTP request for data from our system's *Application Programming Interface (API)*, created based on *REpresentational State Transfer (REST)* [35] architecture standard. The crawling process operates in the background automatically, while user can access and manage crawled photos using web-based graphical user interface that is accessible from web browser application of desktop computer and mobile devices.

4.1 SYSTEM ARCHITECTURE

We designed the MIC system loosely based on standard web crawler system architecture described previously in Section 2. Its high-level architecture consists of five main components, which are the downloader, the queue, the scheduler, the storage and the web-based graphical user interface.

Figure 13 illustrates the architecture of MIC and interactions between the components. In general, the system starts when the scheduler initializes a *crawling task* that is scheduled to run at a certain time. A predefined seed URL is sent to the top of the queue, which is then selected as initial input for the downloader. The downloader requests for web page associated with the URL from the web and receives response data in the form of HTML document. Then, it extracts all URLs in the document and downloads all potential geotagged images. The extracted URLs are sent to the queue where they are organized by priority, while the images and their metadata are stored in the storage. The above process repeats with the next URL from the queue, which is the highest priority URL. Once the queue is empty or the number of visited URLs reaches a certain limit, the current running crawling task is terminated. The crawled data saved to the storage can be requested by the web-based graphical user interface component and shown to the user on demand.

[35] https://www.codecademy.com/articles/what-is-rest

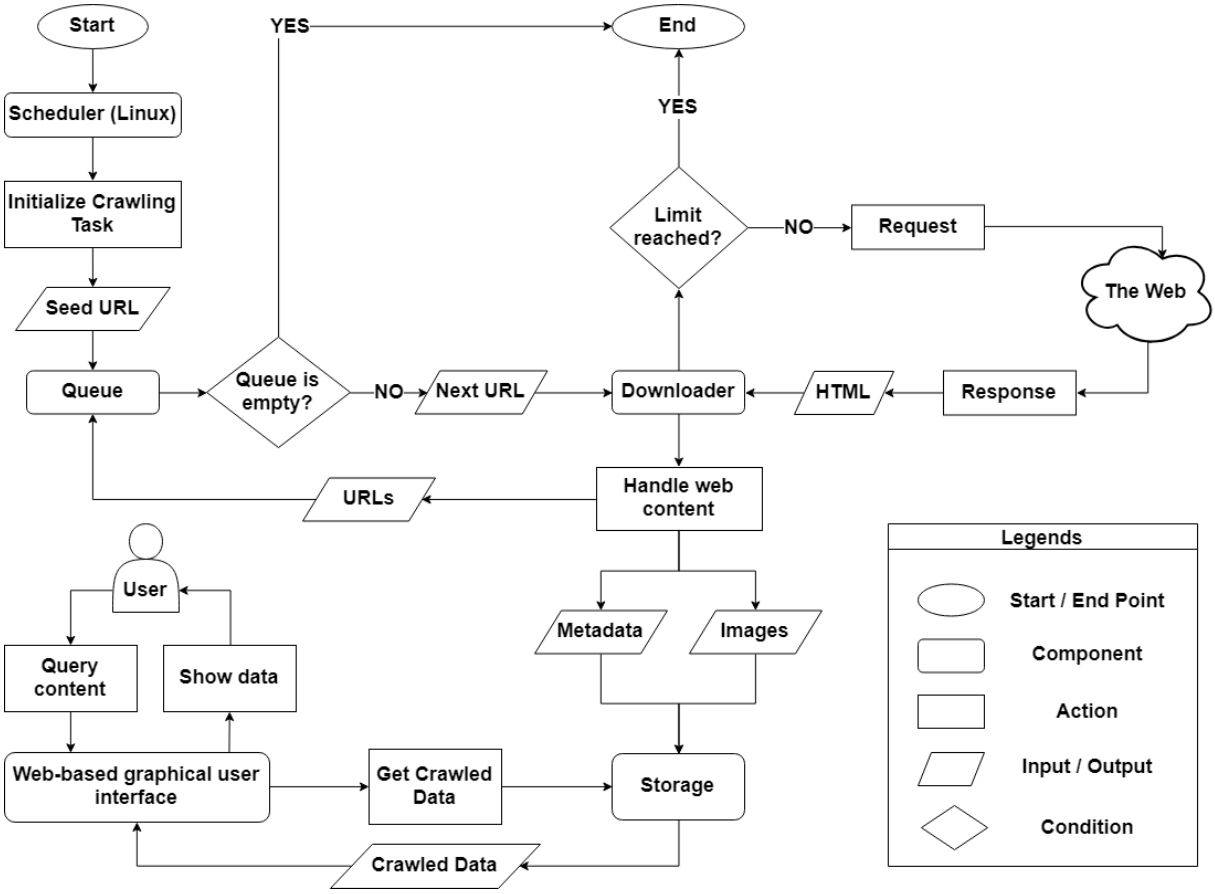

Figure 13: System architecture of Mopsi Image Crawler and interactions between components

4.2 The Web-based Graphical User Interface

We implemented the web-based graphical user interface (web GUI) as a standalone web application that consumes the data obtained from our system's crawling outcomes. Through the web GUI, user can view the digital photos and their location on a world map. They can perform basic data manipulation operations such as viewing a downloaded photo and its metadata, updating metadata of a downloaded digital photo and removing an existing photo from the system. User can also view the summary of the crawling results through statistics collected by the system organized in charts and tables. The web GUI is designed such that it is accessible via different web browsers on desktop computers or mobile devices as shown in Figure 14 and 15.

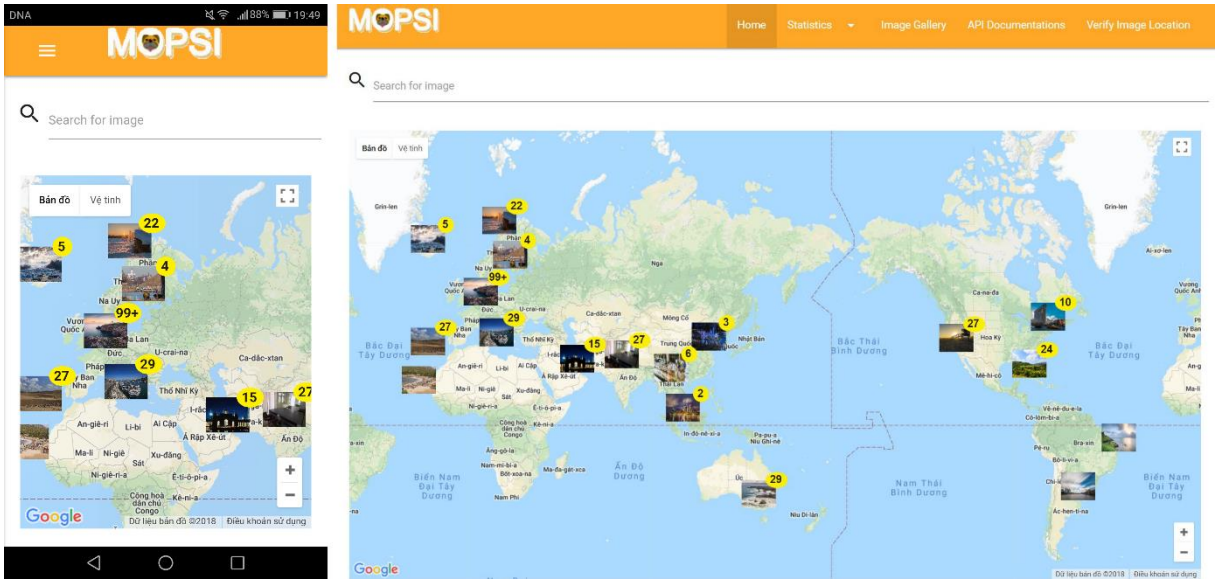

Figure 14: Graphical user interface of the Mopsi Image Crawler system on mobile (left) and on web (right)

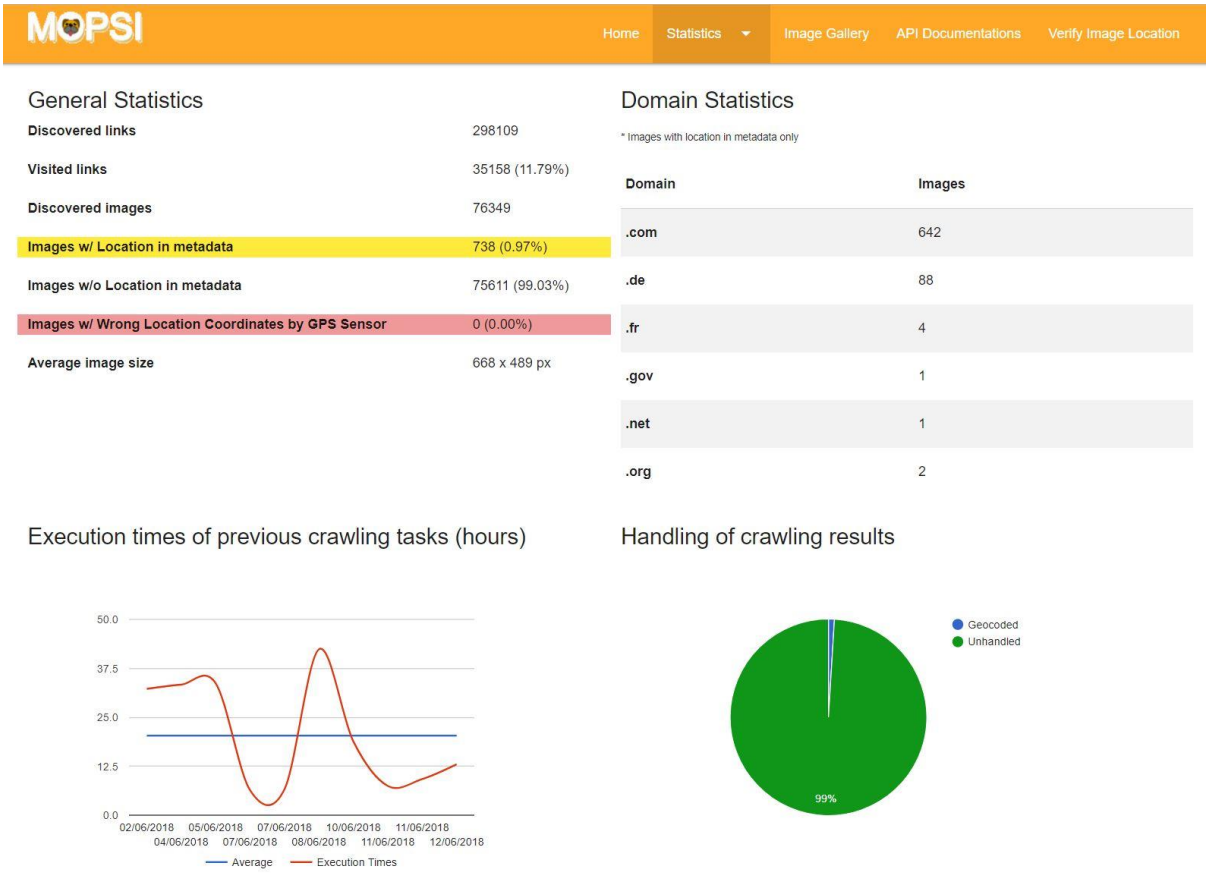

Figure 15: Crawling results as statistical data collected by the system

4.3 THE DOWNLOADER

The downloader component handles the actual crawling process. It receives a URL as input and navigates the web to download digital photos and extract their metadata to save to storage component. Before actually processing a web page, the downloader converts the HTML content to a

tree-like structure called the *Document Object Model (DOM)* tree[36]. The conversion process is handled by calling the built-in library of the software development framework we used for developing the project[37]. As previously discussed, the downloader utilizes a version of best-first search with a heuristic method for determining relevance of links. An example of the DOM tree can be seen in Figure 16.

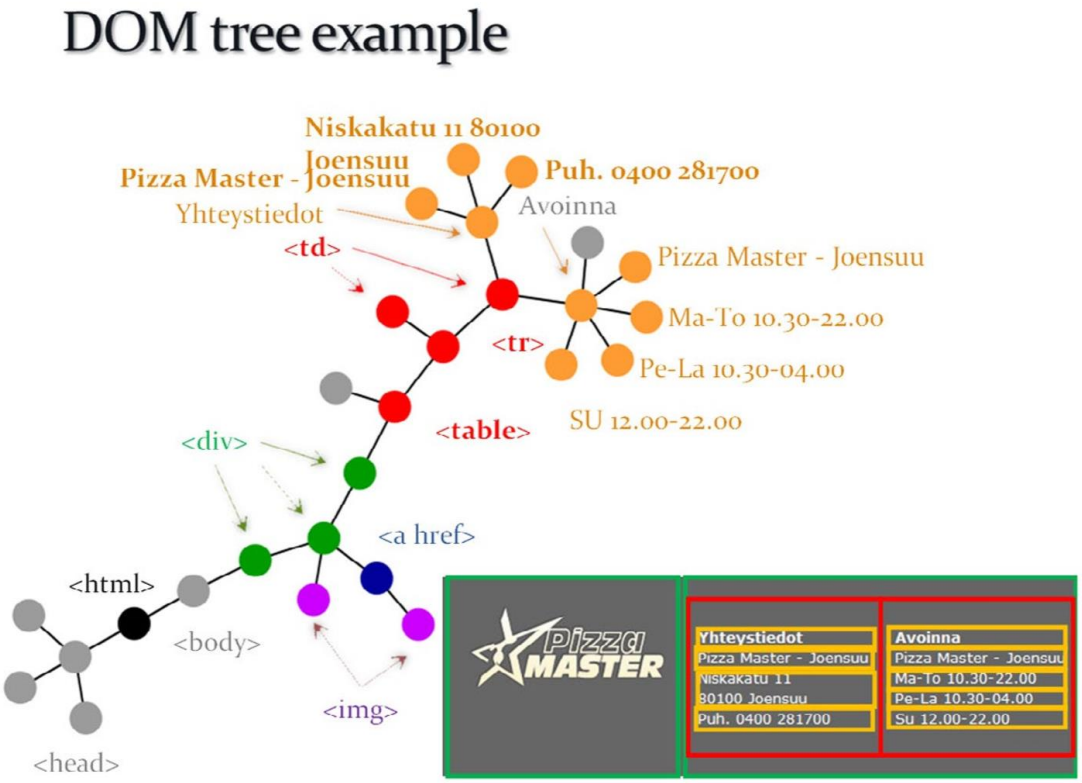

Figure 16: An example of the DOM tree, in which each HTML element is represented as a tree node [23]

4.3.1 Heuristic Method for Determining Relevance of Links

In our work, we aim to design a system that can download as many web images from as many websites as possible. Therefore, we do not strictly require that all web pages to be highly related to the initial seed page or to the whole crawling result set. We instead expect that each pair of web pages in the crawling results are related because of two reasons. First of all, if two web pages are of the same website, they are already related via the same host domain and their content are more likely relate to the same topic. Secondly, even when the two web pages are not of the same website, they should still be related in order to satisfy *Search Engine Optimization (SEO)* metrics for external link[38], which determine ranking of modern websites in search engine.

From the above requirements, we designed our own heuristic method for the BEFS algorithm based on relevance of links to the web page contains it. We expect our system to be able to determine relationship between a web page and another connected to it without downloading its content. We found that per W3C standards[39], links on modern websites need to be created with descriptive text title. We can take advantage of such title for calculating relevance score to increase the crawling speed.

[36] https://developer.mozilla.org/vi/docs/Web/API/Document_Object_Model
[37] https://symfony.com/doc/2.8/components/dom_crawler.html
[38] https://moz.com/learn/seo/external-link
[39] https://www.w3.org/TR/html4/struct/links.html

Thus, the relevance between links and the web page is important. A link is considered relevant to the web page if it satisfies one of the following criteria:

1. External relevance
2. Internal relevance

The first criterion means that both the web page associated to the link and the web page containing the link should be of the same website through similarity in their URL's hostname. In our system, links to web pages of the same website are preferable over links to external websites since such web pages are more likely related as discussed previously. We define this criterion based on assumption that pages of the same website are more likely to be related and thus should be given higher priority. The second criterion influences the degree of relevance between the link and the page. We look for some words in the text that describes the link, or *link title*, that occur more than once in the content of the web page. The link title is obtained from the text surrounded by the HTML anchor tag <a/> through the use of DOM tree. Then, some significant words of the link title, called *keywords*, are extracted by a method based on the candidate keyword extraction framework described in [24].

For each extracted keyword, we calculate the number of times it occurs in web page content, this is called *term frequency (TF)*[40]. The higher the number of high-frequency keywords, the more relevant the link is to the web page. As keywords in the link's title obviously occur at least one time in the web page's content, only keywords that occur more than once ($TF > 1$) contribute to the degree of relevance of the link. Details of the method to extract link title's keywords and calculate TF value of each keyword will be discussed in the next section. Our process for calculating link relevance is described in Figure 17's pseudo codes.

```
function calculateRelevance(link, page)
{
	score = 0.0;

	if (sameHostname(link, page))
	{
		score = score + 1;
	}

	words = getKeywords(getTitle(link));

	for (i = 0; i < size(words); i++)
	{
		tf = countWordOccurrence(words[i], getContent(page));

		if (tf > 1)
		{
			score = score + 1;
		}
	}

	return score / (1 + size(words));
}
```

Figure 17: Pseudocodes of link relevance calculation algorithm

[40] http://www.tfidf.com/

As discussed in Section 3, the performance of BEFS algorithm we use depends on the performance of the relevance calculation. Our relevance calculation time complexity is $O(N * M)$, where N is the number of keywords extracted from the title of a link appears in a web page, and M is the number of words in the extracted text content of the web page. This means the length of the text content in a web page has an impact on the time needed for processing the page. Our algorithm thus runs best when the only text in the web page is the title of the link.

4.3.2 Extract keywords from title of a link

A link title is the text surrounded by HTML tag <a/> or the value of the tag's title attribute. This text provides descriptive information about the topic of the link. Figure 18 shows an example of a link and its corresponding HTML element, which has a text title associated.

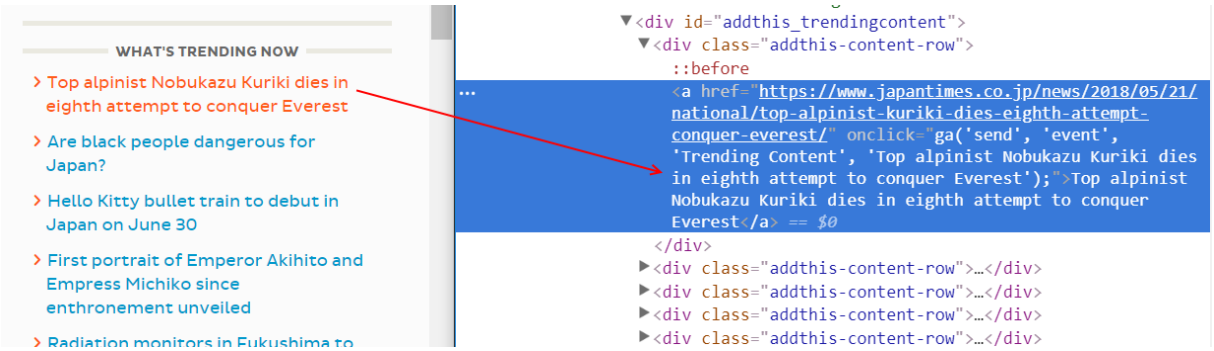

Figure 18: Example of a link and its HTML element, the title of the link is placed inside HTML tag <a/>[41]

Since link title may contain stop words which are high-frequency and less descriptive words such as *a*, *an*, *the*… or special characters, we apply a generic keyword extraction method to extract only meaningful keywords that can be used for the calculation in the heuristic function. We implemented our own version of the method described in [24] for extracting link title's keywords. Since title of a link is short, we do not include the ranking step for keyword candidate in our implementation. The keyword extraction procedure has three steps:

1. Normalize
2. Purify
3. Tokenize

In normalize step, we simply convert all alphabet characters in the link title text to lower-case characters and trim all white spaces before and after the link title's text.

In purify step we replace all special characters and stop words to white spaces. This step is important since it filters out all special characters and high-frequency words. While special characters are not necessary for relevance calculation, stop words do not have any explicit meaning in the content of the text, but happen frequently in all texts. Without removing stop words, our relevance calculation may potentially give higher priority to irrelevant links whose title texts contain more stop words than other relevant links. To be able to detect as many stop words as possible, we import the pre-defined list[42] of 7439 stop words from 28 different languages. Replacing these special characters and stop words to white spaces allow us to apply *regular expressions*[43], which is a method to find matching patterns in text, to break the link title into a set of individual words for relevance score calculation.

[41] https://www.japantimes.co.jp/
[42] https://sites.google.com/site/kevinbouge/stopwords-lists
[43] http://www.regular-expressions.info/tutorial.html

After the purify step, the remaining words in link title are considered as keywords and ready to be separated into individual words, called *tokens*. We use regular expression to detect all white spaces and use them as delimiters to separate our keywords to individual words called *tokens* and put them into a list used for previously mentioned relevance score calculation of links. Figure 19 demonstrates the process of extracting keywords from a real link title of The Japan Times online newspaper[44].

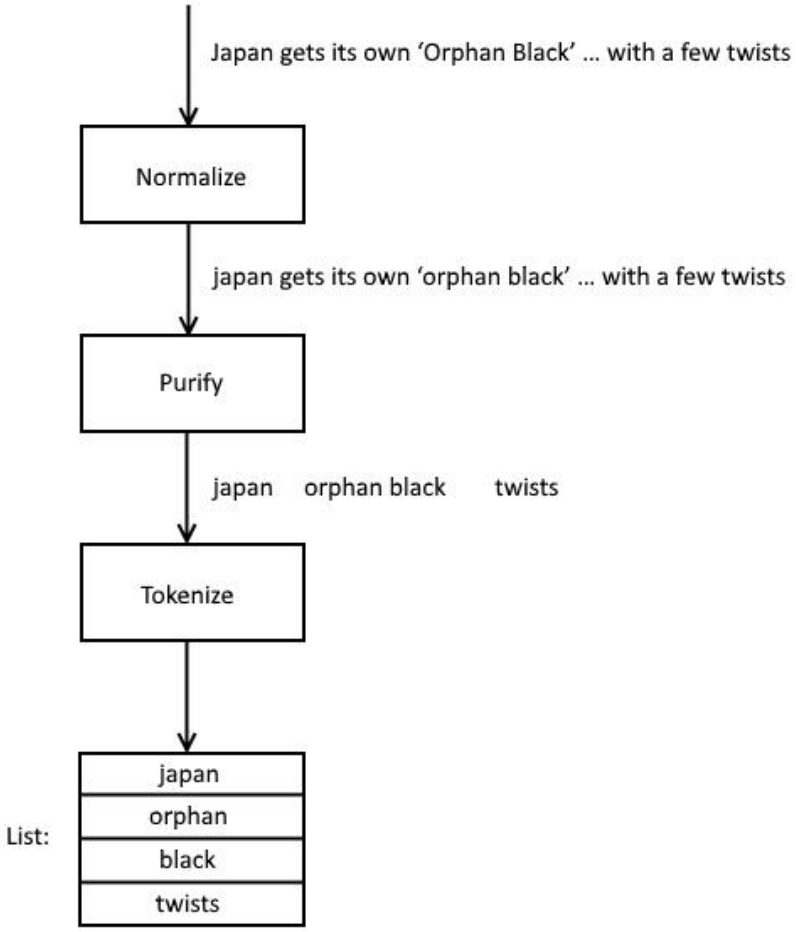

Figure 19: Demonstration of keyword extraction method for link title

4.3.3 Calculate keyword relevance score

The general equation for calculating relevance score of a link is as follow:

$$R = \frac{h + \sum_{TF(k)>1} k}{1 + \sum_{\forall TF(k)} k}$$

Where h takes either value 0 or 1 depends on whether the link has the same host name as the web page or not. $\sum_{TF(k)>1} k$ is the total number of keywords with TF > 1 and $\sum_{\forall TF(k)} k$ is the total number of all keywords.

[44] https://www.japantimes.co.jp/

As an example, we consider a web page entitled "The birth of the web"[45] from The European Organization for Nuclear Research (CERN) website, with links as shown in Table 1. Main content of the web page is assumed to be extracted as shown in Figure 20.

Table 1: Example links for relevance calculation

Link	URL	Title	Title's keywords
L1	http://first-website.web.cern.ch/	project to restore the first website	project, restore, first, website
L2	http://line-mode.cern.ch/	basic browser	basic, browser
L3	http://home.cern/students-educators	Students & Educators	students, educators

CERN Accelerating science
Sign in
Directory
CERN
Main menu
About CERN
Students & Educators
Scientists
CERN community
English
Français
Topic The birth of the web

This content is archived on the CERN Document Server

Tim Berners-Lee, a British scientist at CERN, invented the World Wide Web (WWW) in 1989. The web was originally conceived and developed to meet the demand for automatic information-sharing between scientists in universities and institutes around the world.

The first website at CERN - and in the world - was dedicated to the World Wide Web project itself and was hosted on Berners-Lee's NeXT computer. The website described the basic features of the web; how to access other people's documents and how to set up your own server. The NeXT machine - the original web server - is still at CERN. As part of the project to restore the first website, in 2013 CERN reinstated the world's first website to its original address.

On 30 April 1993 CERN put the World Wide Web software in the public domain. CERN made the next release available with an open licence, as a more sure way to maximise its dissemination. Through these actions, making the software required to run a web server freely available, along with a basic browser and a library of code, the web was allowed to flourish.

Figure 20: Sample web page's text content for link relevance calculation

Among the links in Table 1, only link L3 points to a web page of the same website with the web page in consideration. Therefore, hostname relevance score for links L1, L2 and L3 are 0, 0 and 1

[45] http://home.cern/topics/birth-web

respectively. Calculating the TF values of each link title's keywords, we count the number of keywords that has TF value higher than 1 to produce the keyword relevance scores shown in Table 2.

Table 2: Keyword relevance scoring for the sample links

Link	**L1**	**L2**	**L3**
Keyword & Frequency	project: 2 * restore: 1 first: 3 * website: 4 *	basic: 2 * browser: 1	student: 1 educators: 1
Score	3	1	0

** Keywords with TF larger than 1*

Finally, we divide the sum of host name relevance score and keyword relevance score by the total number of keywords plus one and get the final relevance score R_{L_1}, R_{L_2} and R_{L_3} for links L1, L2 and L3 respectively as follow:

$$R_{L_1} = \frac{0+3}{1+4} = 0.6$$

$$R_{L_2} = \frac{0+1}{1+2} \approx 0.3$$

$$R_{L_3} = \frac{1+0}{1+2} \approx 0.3$$

From the above example, the most relevant link is L1 although it points to external website of different hostname, while L2 is less relevant and L3 is somewhat relevant thanks to its matching hostname.

4.3.4 Rules for downloading image

A web page may contain various types of images that are classified efficiently into five different categories based on their functionality in [25]: *representative*, *logo*, *banners*, *advertisement* and *formatting and icons*. However, as the five categories are overlapping, and we aim for simplicity, our system organizes web images into only two categories *representative* and *non-representative* that are described below with example shown in Figure 21.

- **Representative:** Images whose size meets standard aspect ratio[46], known as the ratio between image's width and height, of a photograph, or images that are directly related to the content of the website;
- **Non-representative:** Small images, images whose size does not meet the photographic aspect ratio or those are not directly related to the website's content, such as logo, banner or advertisements.

Table 3 provides the rules we use for categorizing website images and Table 4 lists the standard aspect ratios our system supports. The idea of using aspect ratio for downloading specific type of image is already introduced in [25], in which it is used for detecting logo and banner images and is highly experimental. In our system, we aim to use aspect ratio as a base for detecting representative images that are potentially photographs. Our method is based on the following assumption: digital cameras or smartphone's built-in cameras have output image sizes subjected to international standards described in [26]. Meanwhile, web images created manually by human, especially icons or banners, follow different standards or no standard, thus making them unlikely photographic images.

[46] https://en.wikipedia.org/wiki/Aspect_ratio_(image)

Furthermore, since most modern digital cameras can capture images with resolution higher than 2 Megapixels, or 1600 pixels in width and 1200 pixels in height, the possibility that a large-size image is a photograph is higher than other small images. The only issue we have with analyzing aspect ratio is we need to download the image before knowing its size, which impacts the crawling speed. Currently, there is no known method for analyzing an image's size without downloading the image.

For detecting keywords of the image, we use the same keyword extraction method described in the previous part of this section. This time, the input for keyword extraction is the text in the web page that describes the image, or image description. We will describe in detail the method for extracting such description text in Section 5.

Our crawler system only seeks for images of representative type because of two reasons. Firstly, if an image is an actual photograph, it more likely contains location information in its metadata. Secondly, since the image relates to the content of the website, even if the image contains no location information in its metadata, we can still determine its relative location by analyzing the text content of the web page contains it. If the system finds an image in <img/> HTML tag that falls into representative image category, it is downloaded to the web server's storage and the system proceeds necessary action to determine its location information either from metadata or web page's text content. The process for determining image location information will be described in Section 5.

Table 3: Rules for categorizing web images

Category	**Features**	**Keywords**
Non-representative	Width < 400px or Height < 400px Non-standard aspect ratio	Logo, banner, header, footer, button, free, adserver, advertisement, ads, now, buy, join, click, affiliate, adv, hits, counter, sprite
Representative	Not in the non-representative category	

Table 4: Supported standard aspect ratios of digital image, grouped by image orientation

Orientation	**Aspect Ratio**	**Decimal**	**Example image resolutions in pixels (width x height)**
Rectangle	1:1	1.00	480 x 480, 512 x 512, 1024 x 1024
Landscape	4:3	1.33	640 x 480, 800 x 600, 832 x 624
	5:4	1.25	600 x 480, 1280 x 1024, 1600 x 1280
	3:2	1.50	960 x 640, 1152 x 768, 1440 x 960
	5:3	1.67	800 x 480, 1280 x 768
	16:9	1.78	960 x 540, 1024 x 576, 1280 x 720
	3:1	3.00	1200 x 400, 1500 x 500, 1800 x 600
Portrait	1:3	0.33	700 x 2100, 800 x 2400, 900 x 2700
	3:4	0.75	720 x 960, 768 x 1024, 864 x 1152
	3:5	0.60	480 x 800, 768 x 1280
	4:5	0.80	1280 x 1600, 1440 x 1800, 2048 x 2560
	9:16	0.56	900 x 1600, 1080 x 1920, 1440 x 2560
	2:3	0.67	1280 x 1920, 1440 x 2160, 1824 x 2736

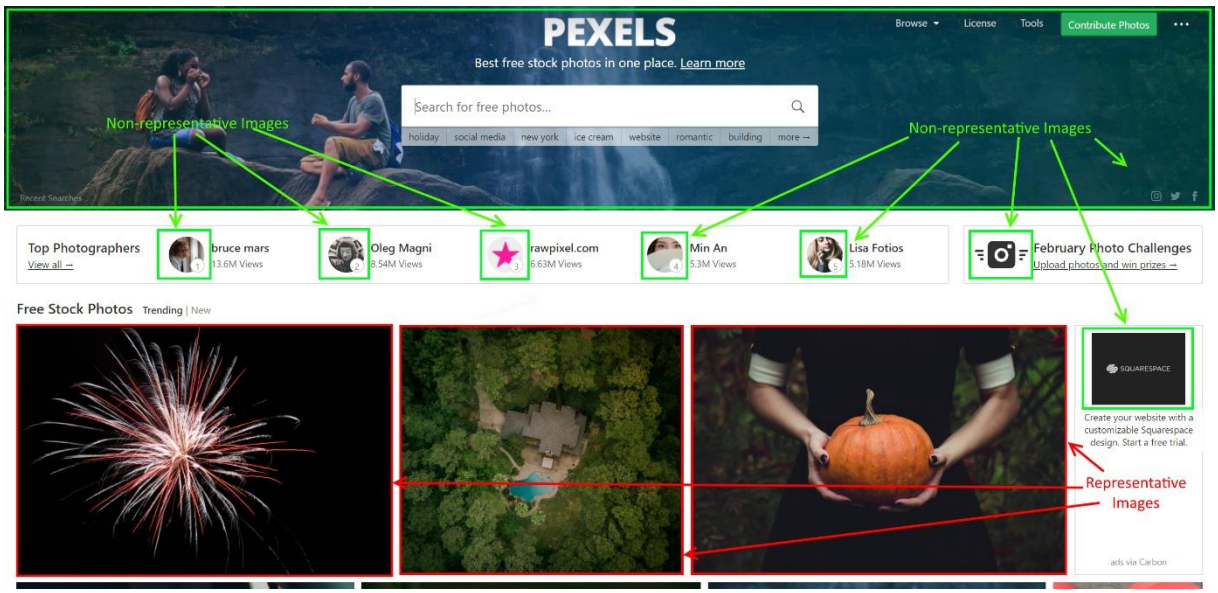

Figure 21: Example of image categorization for representative and non-representative images

4.4 THE STORAGE

MIC storage component is composed of two sub-components namely the file storage and the metadata database. The file storage is a partition of the web server's hard disk used for storing downloaded images. Data management functionalities are provided by the Linux operating system. The image files are organized into different directories whose name are domain names of the websites contain the photos. For example, in MIC project, all images belong to the web pages of website www.locationscouts.net are saved into *images* directory in the project's source codes directory. The directory is located under the path */var/www/html/crawler/imc/web/downloaded/*, where we have full access permission provided by server administrator. The directory structure is illustrated in Figure 22.

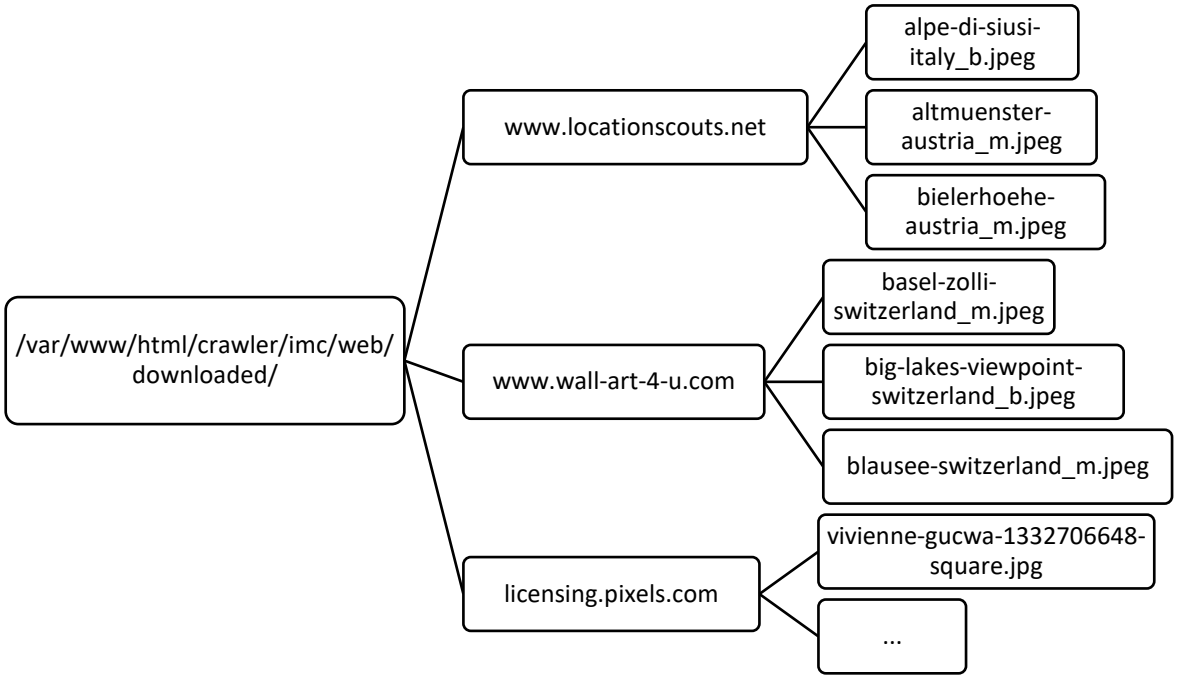

Figure 22: Illustration of directory structure for storing physical image file on server's storage

The metadata database aims to keep cache records of information about the downloaded web resources. It is because we do not want to perform extraction of web resource metadata twice. When we want to display web resources information on the GUI, reading data available from a database is faster. We use *Structured Query Language (SQL)*[47] for querying the data with support of MySQL[48] system as our database management system. The SQL database stores data in table structure, in which data are stored in different rows with each row has its own index as identifier. Figure 23 shows example of table Image, in which store metadata extracted from the web images.

#	id	filename	type	is_exif_location	latitude	longitude
1	198	shangnight.jpg	JPEG	1	31.240664	121.485283
2	202	byd.jpg	JPEG	1	30.993492	121.283011
3	1062	sylvesterpihapojat.jpg	JPEG	1	60.196872	24.936536
4	1097	flowrox.jpg	JPEG	1	61.058922	28.193992
5	1099	flowrox_digikoulu.jpg	JPEG	1	61.059128	28.194797
6	1101	20180418_154026.jpg	JPEG	1	62.598056	29.743889
*	NULL	NULL	NULL	NULL	NULL	NULL

Figure 23: The Image table in MIC database, where extracted image metadata are stored

There are two types of metadata saved to the database storage: The first type is link data, which are the links of the visited URLs, their title and description. Keeping track of visited URLs is particularly useful to avoid crawling the same URL more than once as the system continuously check for existence of visited URL during a crawling task. The second type of metadata is image's EXIF metadata obtained from downloaded images. We specifically extract the geographical coordinates in the EXIF metadata, then use Google's geocode API[49] to determine address of the location to save to the database. We

[47] https://docs.microsoft.com/en-us/sql/odbc/reference/structured-query-language-sql?view=sql-server-2017
[48] https://www.mysql.com/
[49] https://developers.google.com/maps/documentation/geocoding/intro

consider location address necessary because when integrating the image content to O-Mopsi, address information can be of more interest to users rather than the coordinates.

4.5 THE QUEUE

We implemented the queue as a priority queue, which uses the web server's memory unit as storage for URLs. It is based on the concept of *binary heap* [27], an abstract data structure in the form of a *complete binary tree*[50], in which each node should always have two child nodes and all tree levels are fully filled. The only exception is for the last level of the tree, if it is incomplete it can be filled with nodes from left to right. Each node in the binary heap has its own data or value that is comparable. The binary heap must be either a *max-heap* or a *min-heap*. In case of max-heap, the value of each element is greater than or equal to the value of its children. In contrast, nodes in min-heap have their value less than or equal to their children's values.

MIC's queue is a max-heap in which URLs resemble binary tree's nodes and their relevance score are node values. URLs in higher level of the heap are more relevant to the crawler and the next URL chosen for a crawling task is always the top most URL in the heap. We implement the queue to save link data to server's memory unit during execution of a crawling task. When the crawler is halted, link data in the queue are saved to the database since they will be freed from the server's memory and lost. The next crawling task starts with the URL of the best link from the database as seed.

The queue supports two main operations:

- Add URL
- Get most relevance URL

To add a new URL to the queue, we perform an operation called *up-heap*, which repeatedly compares relevance score of the newly inserted URL to that of other URLs in the queue to determine what URL is more relevant and swap their positions until all URLs are in correct order. The worst time complexity of up-heap algorithm is $O(\log n)$ where n is the number of nodes in the queue[51]. Up-heap algorithm is described as pseudocodes in Figure 24, assuming the new URL does not exist in the queue's heap.

[50] https://xlinux.nist.gov/dads/HTML/completeBinaryTree.html
[51] http://wcipeg.com/wiki/Binary_heap

```
function upHeap(queue, newNode)
{
	// If a node of the url already exist in queue, we do not need to
	// do anything
	if (hasNode(queue, newNode))
	{
		return;
	}

	// Add the URL as a new node to the end of the heap
	queue[size(queue)] = newNode;
	currentIndex = size(queue) - 1;

	// Compare the new URL node's relevance score with that of its
	// parent node
	parentIndex = floor((currentIndex - 1) / 2);

	while (parentIndex > 0)
	{
		current = queue[currentIndex];
		parent = queue[parentIndex];

		// If relevance score of the new URL node is more than its
		// parent node's
		if (getRelevanceScore(parent) < getRelevanceScore(current))
		{
			// Swap the new URL node with its parent
			currentIndex = parentIndex;
			parentIndex = floor((currentIndex - 1) / 2);
		}
	}
}
```

Figure 24: Pseudocodes of up-heap operation

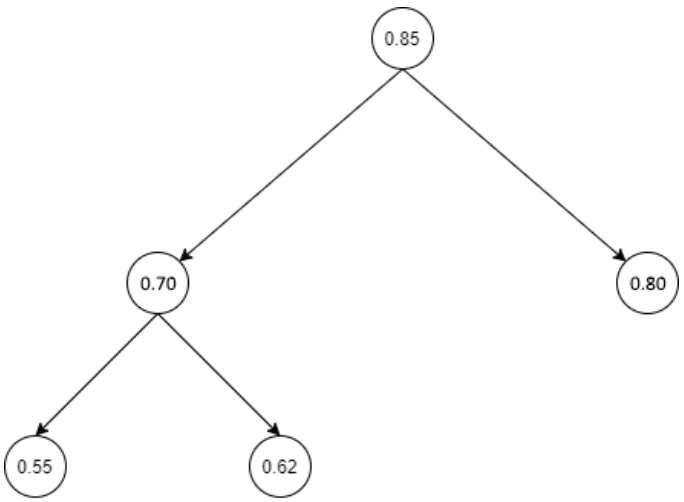

Figure 25: Heap structure of the queue, where each node represents a link with its relevance scores

As an example, let the queue's heap structure be as illustrated in Figure 25. Suppose we want to add a URL with relevance score 0.83 to the heap and we assume that the new URL is not already in there. The main procedure for up-heap is to add the new URL as new node to the last level of the heap following the filling rule from left to right described previously. Thus, the new node is initially the left child of the node with relevance score 0.83 on the second level of the heap. Since the max-heap property is violated as the new URL node is more relevant than its parent node, we need to swap them. After the swap, the algorithm again compares the new node to its immediate parent node, which is now having greater relevance score. As the heap structure is a valid max-heap, the algorithm stops performing additional swap. Figure 26 demonstrates the up-heap operation of the queue.

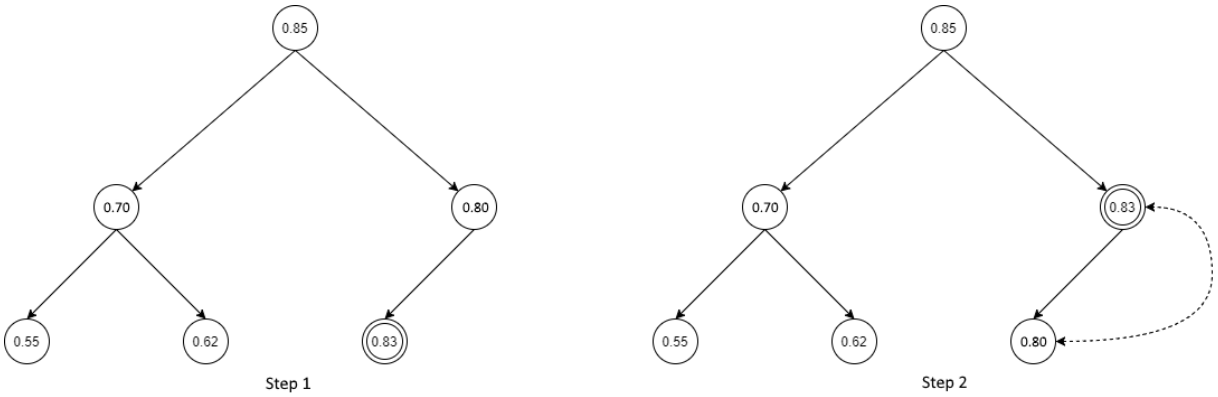

Figure 26: From left to right, the two steps of the up-heap operation. In the first step, the new URL node is placed on the last level of the queue. In the second step, the dotted line indicates the swap between new node and its parent

Getting the most relevant URL from the queue performs the *down-heap* operation, which extracts URL from the top most node, or root node, and removes it from the queue's heap, then reorganize the heap to preserve its max-heap properties before return the copy of root node as output. Same as up-

heap operation, down-heap has worst time complexity $O(\log n)$ where n is the number of nodes in the queue. The down-heap algorithm pseudocodes are shown in Figure 27:

```
function downHeap(queue)
{
	nextNode = queue[0];

	if (notEmpty(queue))
	{
		if (size(queue) > 1)
		{
			queue[0] = queue[size(queue) - 1];
			maxHeapify(queue, 0);
		}
	}

	return nextNode;
}

function maxHeapify(queue, i)
{
	// Get the max node at index i
	// and its left and right children
	leftIdx = 2 * i + 1;
	rightIdx = 2 * i + 2;
	maxIdx = i;

	leftNode = queue[leftIndex];
	rightNode = queue[rightIndex];
	maxNode = queue[maxIdx];

	if (leftIdx < size(queue) && getRelevanceScore(leftNode) >
getRelevanceScore(maxNode))
	{
		maxNode = leftNode;
		maxIdx = leftIdx;
	}

	if (rightIdx < size(queue) && getRelevanceScore(rightNode) >
getRelevanceScore(maxNode))
	{
		maxNode = right;
		maxIdx = rightIdx;
	}

	if (maxIdx != i)
	{
		queue[maxIdx] = queue[i];
		queue[i] = maxNode;
		maxHeapify(queue, maxIdx);
	}
}
```

Figure 27: Pseudocodes of the down-heap algorithm

Continue from the result of the up-heap algorithm, we now perform down-heap operation on the queue's heap to get the most relevant URL. Assume we keep the URL of the root node in the memory,

we remove it from the heap and replace it with the last node, which is the node whose relevance score is 0.8. The max-heap is not valid, so the algorithm compares the new root node's relevance score with its children. Of the two child nodes, the child node with relevance score 0.83 is more relevant so a swap is performed to switch the root node with its child node on the right. The queue's heap is now valid, the algorithm stops and return the old root node's URL that is later used for the downloader component. Figure 28 demonstrates the down-heap operation on our sample heap.

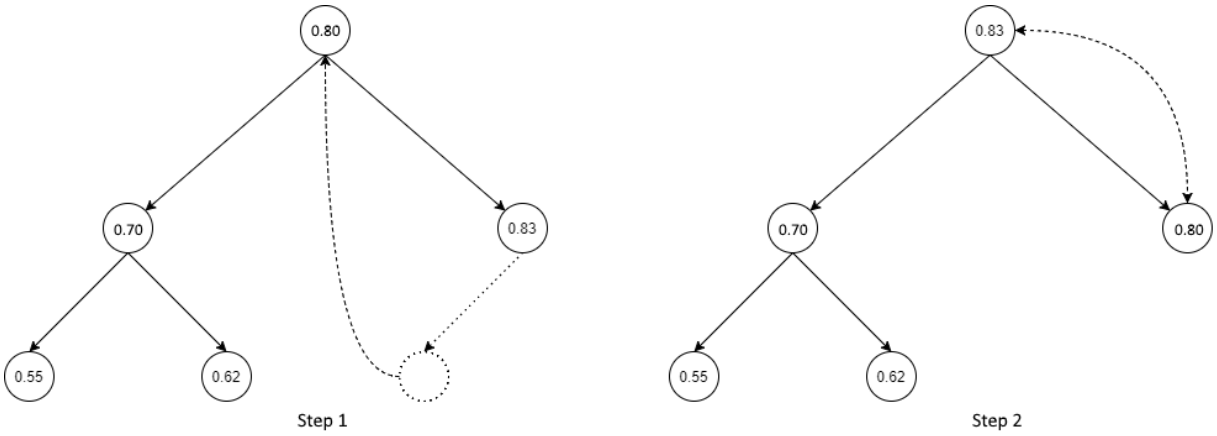

Figure 28: From left to right, the illustration of down-heap operation. First, the root node is removed, and the last node is changed to root node. In the second step, the new root node is swapped with its right child node, which is the more relevant node

We observe that on a website, shared links such as link to homepage, menu links or social network links are more likely to be duplicated when adding to the queue. Table 5 shows results of some test runs on 5 websites as seeds. In the tests, we gather information about the number of times the link to homepage, which is also the seed, appears in the queue over the total number of links regardless of duplications. For each of the seed, we let the crawler visit and extract links of 10 pages.

Table 5: Occurrences of homepage link in the queue which lead to URL duplication issue

Website	Links in queue	Occurrences of homepage link
http://www.siliconera.com/	1034	14
https://imagelocations.com/	733	8
https://techviral.net/	589	14
http://sea.ign.com/	1904	21
https://www.japantimes.co.jp/	2479	25

While the metadata database provides a layer to prevent duplication of URL for the entire system, we still need to ensure that no URL is added to the queue more than once. The heap structure of our queue component is represented as a list of URLs with their respective relevance scores. When we want to check whether a URL already exists in the queue, it is inefficient to go through every entry of the list and do string comparison. Therefore, we encrypt each URL using the Secure Hash Algorithm 256 (SHA-256)[52] to receive a unique series of 64 characters called the *hash* string. We then use the hash string as unique identifier and index of the link entry in the list. Our queue thus becomes a list of key-value entries, called a *hash table*, in which the key is the hash string and the value composed of a URL and its relevance score.

[52] https://www.thesslstore.com/blog/difference-sha-1-sha-2-sha-256-hash-algorithms/

The benefit of this encryption method is we can use the hash as index to check for existence of URL. The process of searching URL by hash index is fast since accessing an element of the hash table is a constant-time operation on average[53]. In addition, regardless of the URL's length, the hash produced by SHA256 algorithm always has the length of 64 alphanumeric characters and is unique with low collision probability[54]. Table 6 shows a few examples of encrypted URLs using SHA256. A sample of our queue as a hash table is illustrated as in Figure 29.

Table 6: Example of hashes produced from URL encryption using SHA-256

Original URL	**SHA-256 encrypted string identifier**
https://www.locationscout.net/	901347838dd47f1266b708382b4e1c8012b81 5a9a0f9b7b7aa5700259a1508a2
https://www.locationscout.net/germany	b47842630880fc38536ff12f4c84c975f05050d dc4551e739db19b5c45100179
https://www.locationscout.net/locations/4-paris	3c008b8fd5ab9c4f6a7cd721ecaa38c9c8e1996 607542118b4122a00276dd9ea
https://www.locationscout.net/sign-up	bd7b5094bed1b05aaa0ec74498276effc5e18f 4c4c7d83374d6352c6d671199e

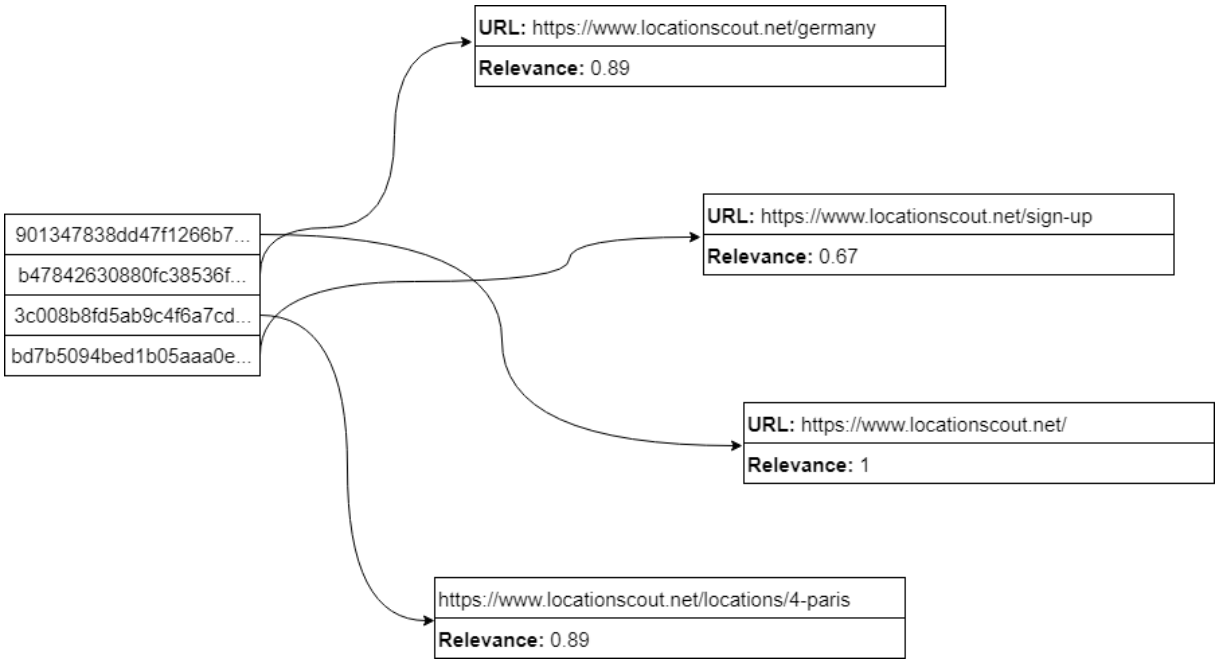

Figure 29: Illustration of the hash table stored in server's memory used for heap structure of the queue

Whenever the downloader put a URL to the queue, it must go through two layers of duplication detection. The first layer checks whether the URL is already in the metadata database, e.g. it is already visited, and the second layer checks whether the URL's hash already exists in the hash table previously described.

Since the queue component is an in-memory data structure, the number of links saved into the queue grows quickly after each of crawling algorithm iterations, while one iteration only gets the most relevant link out of the queue. To avoid exceeding the web server's memory, we limit the number of

[53] http://www.programming-algorithms.net/article/50101/Hash-table

[54] https://crypto.stackexchange.com/questions/24732/probability-of-sha256-collisions-for-certain-amount-of-hashed-values

visited links for a crawling task to 1000. After visiting 1000 links, the algorithm stops, which in turn stopping a crawling task. Figure 30 shows the size of the queue growing after each algorithm iteration.

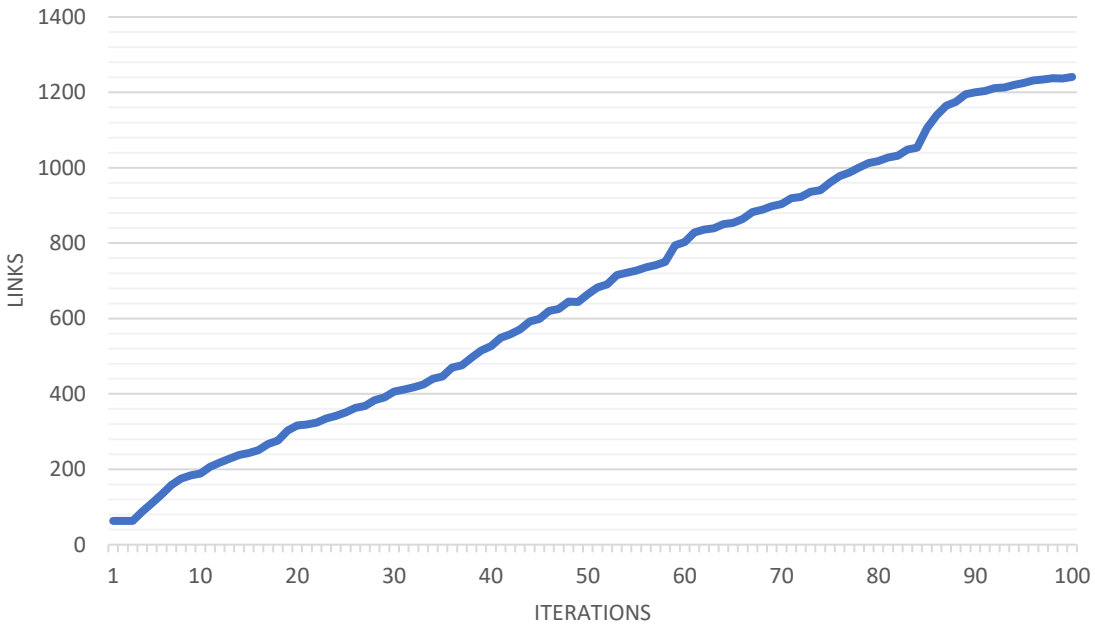

Figure 30: Size of the queue after each algorithm iteration

4.6 THE SCHEDULER

Unlike the downloader, the storage, the queue and the GUI, the scheduler is not an internal component of the MIC system but rather a part of the server's operating system. As we implement the MIC system on Linux-based operating system, we take advantage of the Linux's *Cron Scheduler*[55], which is an operating system's background program, also known as a *daemon*[56], for scheduling and executing other programs in the system.

The Cron Scheduler runs at the time of the operating system boot. In every minute, it examines the *Cron Table (crontab)* file, in which contains a list of the schedule of *cron jobs*. Each cron job is a set of execution instructions for executing program in specific day and time. If the cron job schedule match the moment of time when it is checked, the Cron Scheduler automatically executes it and in turns triggers the command that runs the program associated with the job.

The instructions for scheduling executions of program must follow a strict syntax contains five fields: minutes, hour, day of month, month and day of week and followed by the program command to be run at the interval. For the MIC system, we schedule crawling tasks to run four times a day at 00:00, 06:00, 12:00 and 18:00 every day. Figure 31 illustrates the syntax of the cron job instruction used for scheduling the crawling task in practice.

[55] https://www.pantz.org/software/cron/croninfo.html
[56] http://www.linfo.org/daemon.html

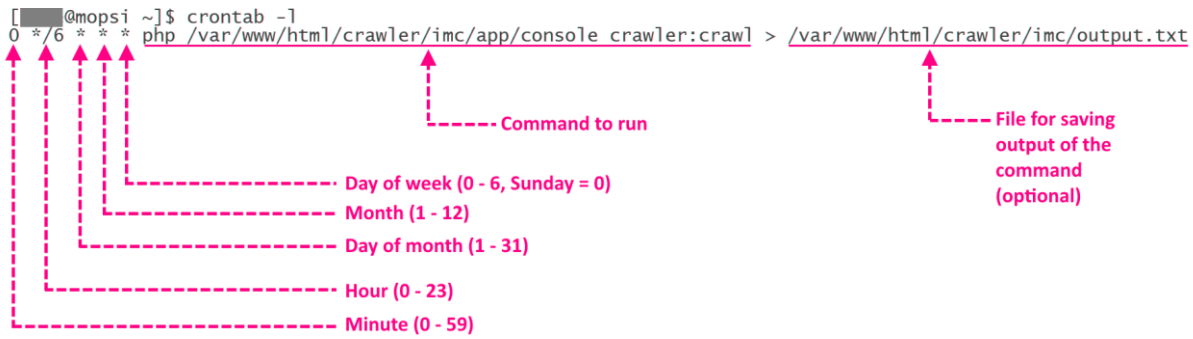

Figure 31: Illustration of cron job instruction syntax for scheduling execution of the MIC system's crawling task

The instruction syntax in Figure 29 can be literally translated to: "Run a crawling task every day for every 6 hours at minute 0 and save the output to output.txt file". Note that the instruction syntax has two fields for specification of days which are month day and weekday as seen above. If both are defined, then they are cumulative and both entries will be executed. For example, if we define day of month as 27 and weekday as 4 in the instruction schedule above, when the current date is day 27th of the month or it is Thursday then the crawling task will be executed. The asterisk sign * in the instructions means the field should match all the valid values specified in the braces of the field's description. The slash sign means repeat pattern such as /6 for execution of the crawling task every 6 hours. Ranges of numbers are allowed in the instructions, separated with a hyphen, i.e. 0-6 or 8-11. A list of number is also allowed, separated with a comma, for example 5,10,15 or 0,15,30. Table 7 shows more examples of cron job instruction syntax and their meanings.

Table 7: Examples of cron job instructions and their meanings

Instruction syntax	Meaning
30 0 1 1,2,3 *	Execute at 00:30 on the 1st of January, February and March
*/5 3 * * *	Execute every day and every 5 minutes between 03:00 and 04:00
0 0 1-7 * *	Execute at midnight on the first week of every month
15,30 12 20 * 1	Execute at 12:15 and 12:30 every Monday and on the 10th of every month

While the built-in Cron Scheduler provides a convenient way for scheduling the crawling tasks, it has a drawback in managing crawling task executions. Since we allow only one crawling task to run at a time, there can be overlap between crawling tasks as their execution times are varied. It is because the BEFS algorithm performance depends on relevance calculation function, which in turns depends on the size of web page content we crawl. As an example, consider a crawling task A starts at 06:00 and crawling task B scheduled to start at 12:00 based on the above cron job definition. If the crawling task A runs for approximately 8 hours, there will be overlap between tasks A and B.

To solve this issue, we save information of a crawling task in the metadata database with three information: its start time, its end time and whether it is running or not. When a crawling task starts, we create a record of its execution in the database as unfinished task. The database record is updated before the crawling task is terminated to inform the system that it is done. When Cron Scheduler starts a new crawling task, the new task must check the metadata database to see if there exists a running task. If that is the case, the new crawling task exit, otherwise it runs according to cron job schedule.

5 GEO INFORMATION RETRIEVAL

Even though image metadata provides the most convenient way for extracting geographical information from images, we found that most images on the web do not have location information in their metadata [23]. In fact, not all imaging devices used for capturing photos to digital image have sensor to record geographical information. The owner of the images may also intentionally turn off the GPS functionality of the imaging device or remove geographical information from the image's metadata for privacy protection. Furthermore, authors of many modern websites may also use image editing application to remove image's metadata in attempt to reduce image's size. The main reason for removing image's metadata relates to website's performance optimization. When there are large number of metadata stored in an image, its file size becomes bigger which in turns increase browser loading time[57]. An independent experiment online recently reported that removing image's metadata leads to about 8.5% smaller image size[58]. In a small experiment, we collected 20 geotagged images from Locationscouts[59] website and use a software called ExifPurge[60] to remove EXIF metadata from the images. The images occupy 25.4 Megabytes of storage in total and after removing EXIF metadata they only occupy 13.1 Megabytes, which is 51.6% reduction of size. For individual image, the reduction percentage ranges from 2% to 73% and on average image without EXIF has 54% less in size.

We performed some test crawling tasks with 8 input seeds from different websites. For each of the websites, our crawler visited 100 web pages. The final crawling results showed that out of 6845 images downloaded, only 14 images have geographical information in their metadata, or 0.2% of total number of images. Later in Section 6, our statistics also show that no more than 2% of downloaded images in our experiments are geotagged. Our experiment results are displayed in Table 8.

Table 8: Number of images with geographical information in their metadata versus number of downloaded images

Website	Images	Geotagged
https://www.locationscout.net/	922	5
http://www.foxnews.com/travel.html	150	1
http://www.lonelyplanet.com/	303	5
http://businessinsider.com/travel/	1714	2
http://www.vogue.com/living/travel/	4	0
http://www.dailymail.co.uk/travel/	3449	1
http://www.bbc.com/travel/	217	0
http://www.visitfinland.com/	86	0
Total	**6845**	**14**

To increase the number of geotagged images in the crawling results, we need a method to determine actual location of the downloaded images based on information associated to them. Such information can be found by visually inspecting the image or use the text descriptions of the images found on web pages they belong to using *Geographic Information Retrieval* techniques.

Geographic Information Retrieval (GIR) is an activity of acquiring geographic information from a resource collection, particularly a collection of texts [28]. The process in which geolocation information is determined and extracted from a text resource is called *geoparsing*.

[57] https://www.keycdn.com/blog/image-metadata/
[58] https://blog.shortpixel.com/how-much-smaller-can-be-images-without-exif-icc/
[59] https://www.locationscout.net/
[60] http://www.exifpurge.com/

The *geoparsing* process takes an unstructured text description of places as input and produce geographic coordinates in the form of latitude and longitude values as output [29]. For instance, the following string of text: "Cherry Blossom Heerstrasse, Bonn. Photo by Anirban Chakraborty" produces latitude value of 50.723920 and longitude value of 7.103690, that correspond to Heerstrasse Avenue in Bonn, Germany.

The idea of geoparsing text for determining address has been discussed by Tabarcea et al. in [23], in which a framework for location-aware search engine, called LBS, was introduced. One of the components of the LBS framework is the address detector, where downloaded web content is searched for postal addresses. The idea of the address detector is to identify individual address elements such as street, city and post code and then aggregate to build an address candidate. Then, gazetteer data from OpenStreetMap data is used to validate the address candidate.

While using geoparsing for extracting location address is an interesting approach, it is not the main focus of this thesis. Furthermore, we do not have access to the source codes of the LBS framework available for testing. Hence, we decide not to replicate the mentioned work but to implement a geoparser component that take advantage of 3rd party geoparsing API for experimental purpose only. The component can be executed independently from the MIC system through command line interface on the web server.

5.1 Determine Geographic Information of Image from Text Content

Geoparsing text description of image is a non-trivial task that requires a huge database of locations and complex process that increase resource usage of the image crawling system if implemented. Thus, a 3rd party geoparsing web service is used to handle the task, while the image crawler is responsible for extracting relevant text description of image. We utilized the free geoparsing service[61], serves as the *geoparser*, through its Application Programming Interface (API). Whenever the crawler successfully downloads an image whose metadata contains no geolocation information, our system triggers the following actions:

1. Extract image description
2. Request geolocation output from geoparser service
3. Save extracted geolocation to the metadata database

Our first action is attempting to describe the image using textual content found in the DOM structure. According to standard[62] defined by the World Wide Web Consortium (W3C), an image element should have its "*alt*" attribute defined with meaningful text content. However, many websites do not define value of such attribute or provide image description in surrounding HTML elements. In addition, some modern websites take advantage of automatic captioning method such as Cloud Vision[63] API by Google. This method produces only description of how the image looks like rather than where the image is taken. Consequently, image description found only in the "alt" attribute are not reliable and informative for the geoparser. From our own observation, textual information used for describing an image likely comes from three sources: the previously mentioned alt attribute, its file name and the nearest DOM element to the image element, called *caption elements*, matching either one of the following HTML tags:

1. The heading tag <h1/> to <h5/>

[61] https://geocode.xyz/api
[62] https://dev.w3.org/html5/spec-preview/the-img-element.html
[63] https://cloud.google.com/vision/

2. The paragraph tag <p/>
3. The anchor tag <a/>

Figure 32 illustrates how we produce an image description text. Our method is simple: using white space as delimiter, we concatenate each of the texts found in the DOM element attribute, the image's file name (without file extension) and the caption element (with inner HTML tags stripped out) to produce a new text string. The benefit of our method is it helps providing image description information for even non-standard web image without using all the texts found on the web page. Furthermore, the method ensures richness of information as we collect texts from different elements of the web page's DOM structure. However, our method fails to work on websites with low amount of text content or dynamic websites, in which text content is likely generated by interactive web script such as JavaScript.

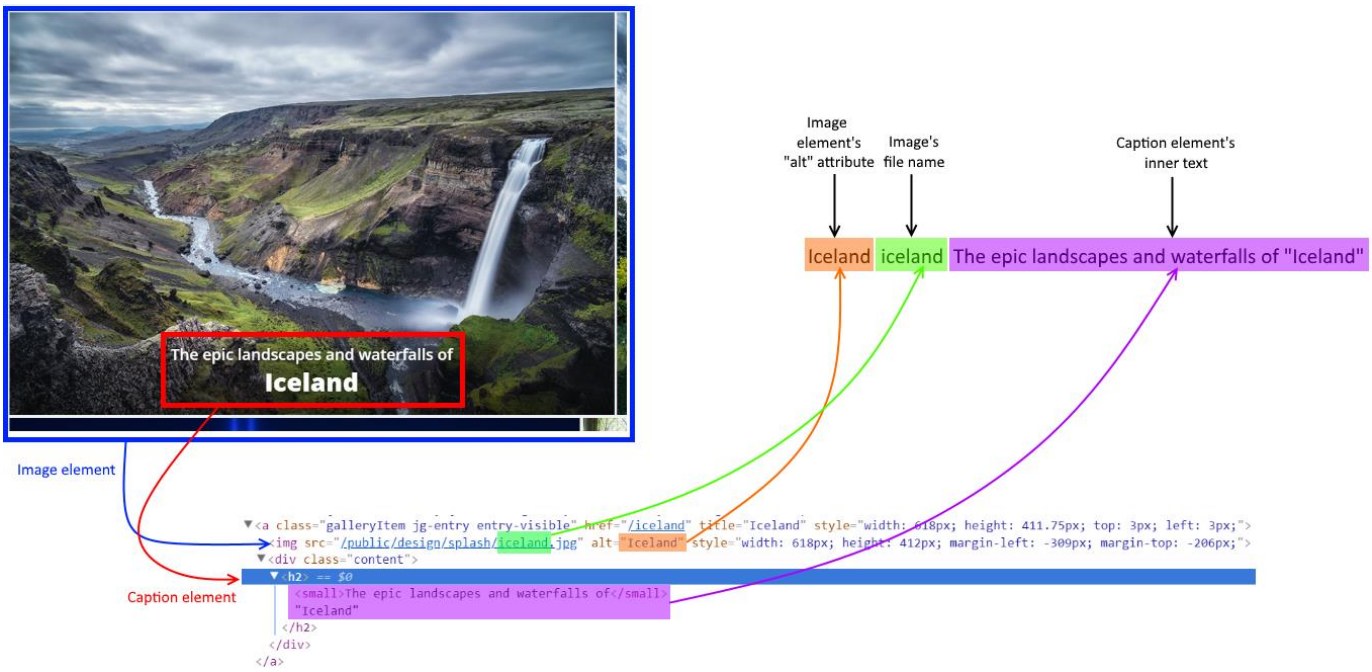

Figure 32: Illustration of the method for producing image description

In the second step, we create a HTTP request to the geoparser and receive some JSON responses that give determined location coordinates and address. A typical request URL string is composed of two parts: the service's address and input HTTP query parameters which are a free-form text and a flag for receiving JSON data as output. Figure 33 provides a sample URL for our system HTTP request to the geoparser.

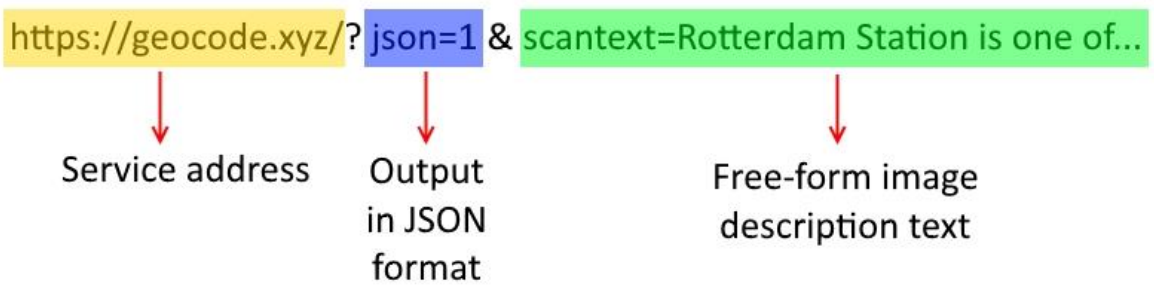

Figure 33: Geoparser request URL parts explained

Where parameter *json* indicates that the service should produce output data in JavaScript Object Notation (JSON)[64] format and parameter *scantext* is the description of the image. Once the HTTP request to the service is successful, it returns output data in which we may find a list of matching. To simplify the geoparsing process, we configure the system to select only the first matching item from the output as it is the most relevant geolocation. Sample output from the service is shown in Figure 34.

Figure 34: Result output of the geoparser service, in which we select the first item from all matching geolocation items of "match" response data object

Of all the output information, our system uses only the geolocation coordinates and its address. Once it receives response from the geoparser, the system updates the metadata of the downloaded image. In case there are errors, or the service cannot determine geolocation information of the image, our system allows user to manually update the geolocation information through GUI frontend.

6 EXPERIMENTAL RESULTS

For experiment purpose, we have setup the MIC system on a personal computer running Ubuntu Linux distribution with the technical specifications shown in Table 9. There are three aspects of the crawling results that we aim to evaluate from the experimental results:

1. Quality of the crawling results
2. Performance of the crawler
3. Impact of seed selection to crawling results

The quality of the crawling results is measured by the number of geotagged images over the total number of images our system retrieves. We do not take accuracy of GPS coordinates information in the images into consideration when evaluating quality, since we assume GPS information are always accurate in our experiments. Moreover, according to the official U.S. government information about

[64] https://www.json.org/

the GPS, its accuracy is very high and reliable[65]. The performance of the crawler is evaluated by collecting data about average execution time of a crawling task, average time for processing a web page and average time of each main step in the crawling process. In addition to the average execution time, average maximum memory usage of the crawler is also taken into consideration for performance evaluation. The impact of seed selection to crawling results is evaluated by counting the number of geotagged images the system retrieves for different seeds.

Table 9: Technical specifications of the computer used for experiment purpose

Processor	7th Generation Intel Core i7-7200U
Memory	16Gb DDR4-2400
Storage	256Gb SSD

We setup experiments to compare the performance between three crawling algorithms implemented in the system: DFS, BFS and BEFS algorithms. In the experiments, we set the limit of 1000 links in total for the crawler to visit by executing 10 consecutive crawling tasks, each task stops after visiting 100 links. The performance experiments are performed on all the three algorithms with Locationscouts[66] website as seed. We gather four pieces of information from each crawling task results:

1. Execution time
2. Memory usage
3. Total number of images
4. Number of geotagged images

Figure 35 shows the comparison chart of execution time between the three crawling algorithms. On average, DFS takes around 14 minutes and BFS takes around 10 minutes to finish a crawling task of 100 pages. Meanwhile, BEFS is slower as it takes around 26 minutes to complete the same crawling task. Based on the shape of the chart, we found that execution times of crawling tasks use BEFS algorithm are not as stable as those use DFS or BFS as its performance depends on the size of content used for calculating relevance between a web page and its links.

[65] https://www.gps.gov/systems/gps/performance/accuracy/#how-accurate
[66] https://www.locationscout.net/

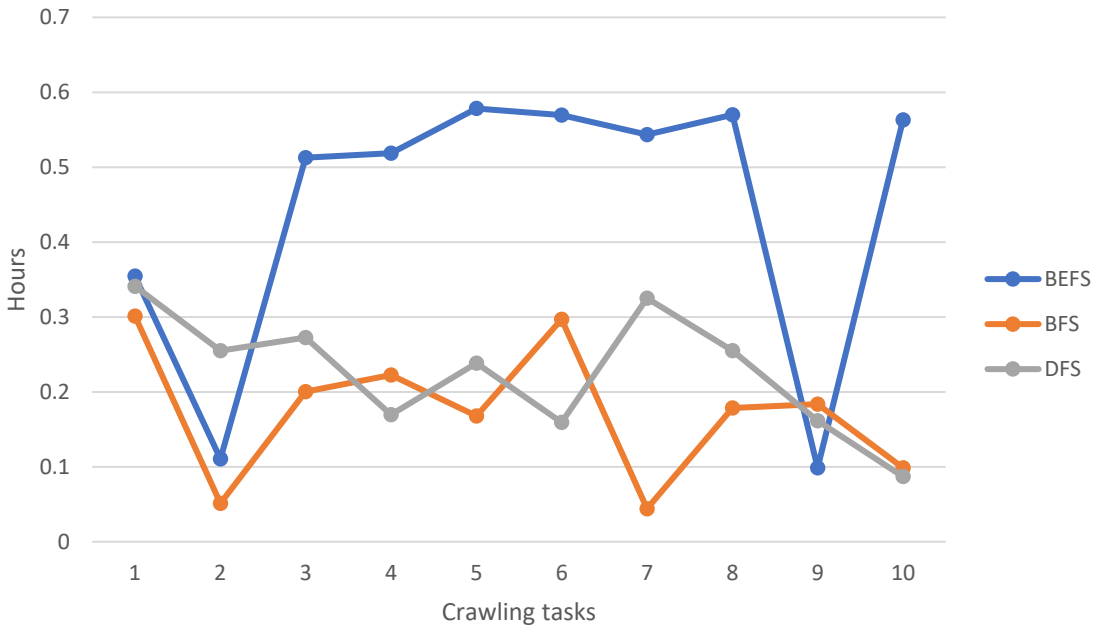

Figure 35: Comparison of execution time between crawling algorithms

Figure 36 shows the comparison of maximum memory usage between the three crawling algorithms. BEFS consumes around 128Mb of memory on average, while BFS and DFS consumes only around 63Mb and 81Mb respectively. Based on the shape of the chart, the amount of memory usage and execution time have a close relation to each other.

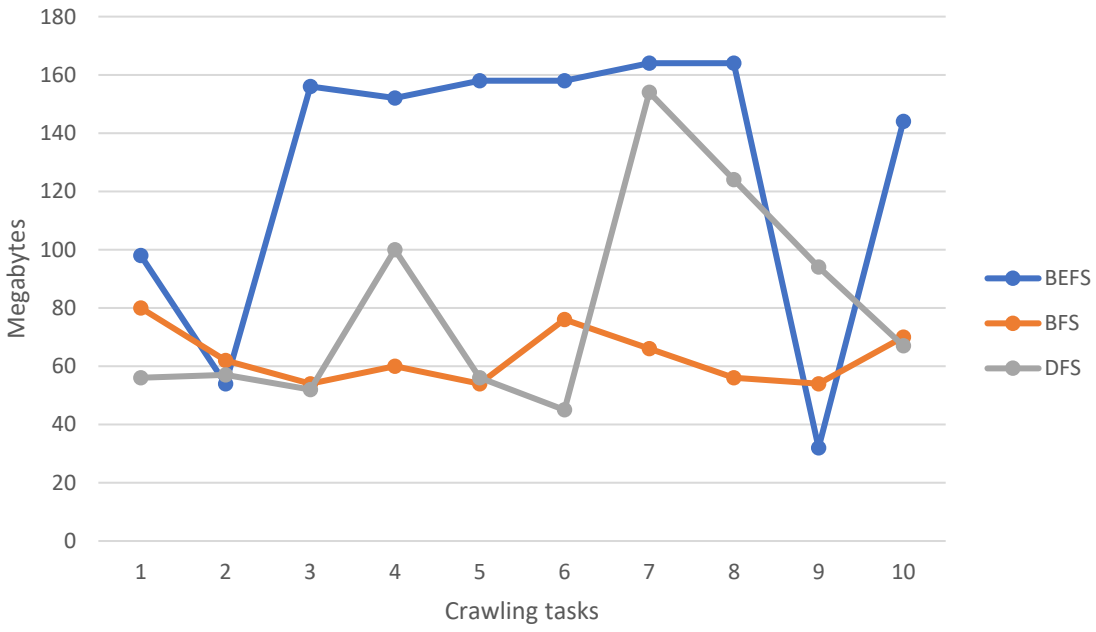

Figure 36: Comparison of maximum memory usage between three crawling algorithms

Our report statistics also shows an average time to process a single web page takes 9.41 seconds for BEFS, 9.23 seconds for BFS and 6.52 seconds for DFS. The process that takes most of the time is downloading images and handling their metadata. This process takes around 86% of the processing time with average time ranges from 5 to 8 seconds for a page contains around 10 images. The other

processes include processing of page's text content, link extraction, queue operations and database reading and writing takes about one second in average for all algorithms. Figure 37 illustrates the proportion of execution times of the steps for processing a single web page in of each crawling algorithm.

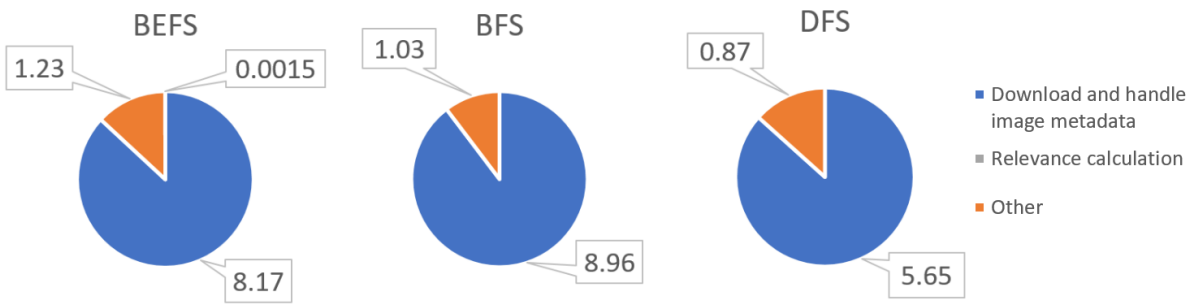

Figure 37: Proportion of execution times of steps for processing a single web page (in seconds)

We can conclude from the above charts and statistics that overall performance of BEFS is lower than BFS and DFS. While BFS and DFS do not have relevance calculation steps, BEFS takes about 1.5 milliseconds to calculate the relevance score of a link. Thus, the number of links and the length of text content in the web page have significant impact on BEFS performance. Figure 38 shows execution times of the crawler running BEFS on 10,000 links through 100 consecutive crawling tasks.

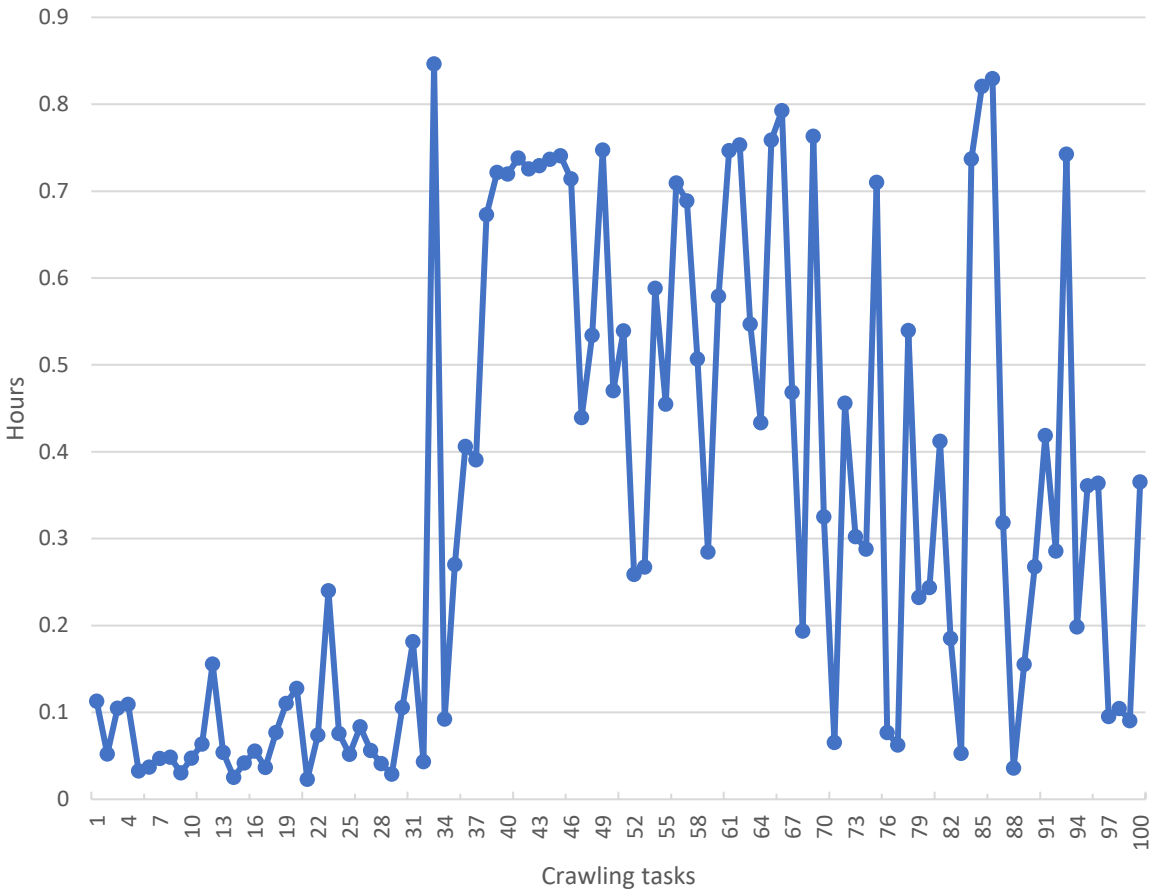

Figure 38: Fluctuation of BEFS algorithm execution times

While BEFS performance is worse than BFS and DFS, it helps retrieving more geotagged images according to the statistics we collected as shown in Table 10. Specifically, BEFS is more likely to discover geotagged images with around 2% of the total images it discovers are geotagged. This percentage is only 1.33% for BFS and DFS has the lowest percentage of geotagged image with about 0.36%.

Table 10: Crawling result statistics of the three crawling algorithms

	BEFS		**BFS**		**DFS**	
#	**Images**	**Geotagged**	**Images**	**Geotagged**	**Images**	**Geotagged**
1	645	23	811	21	972	3
2	222	0	45	0	589	0
3	1272	23	407	0	470	3
4	1170	15	440	5	261	0
5	1295	27	383	10	817	4
6	1280	57	816	5	388	3
7	1360	18	12	0	1	0
8	1390	30	420	9	6	0
9	18	0	306	1	0	0
10	1180	8	258	1	72	0
Total	9832	201	3898	52	3576	13

Reason for the better crawling results of BEFS come from the fact that it considers the relation between individual web page and its associated links as discussed in Section 3, while this is not the case with DFS or BFS. As the seed we chose for the experiment is a website dedicated to images of locations around the world, BEFS has a clear advantage as it likely visits more web pages with the same topic of content.

In the experiment for evaluating seed quality impact on BEFS algorithm, we manually select 20 websites of different topic domains as seeds to our crawler. For each of the seeds, we run only one crawling task that is limited to visit 100 links. Results of the experiment are shown on Table 11.

Table 11: Crawling results of different seeds

#	Seed	Description	Images	Geotagged
1	https://www.locationscout.net/	Geo photo sharing service	945	52
2	https://www.pexels.com/	Photo search & sharing service	781	14
3	https://imagelocations.com/	Photo search service	336	21
4	http://www.foxnews.com/travel.html	Travel news	129	5
5	http://businessinsider.com/travel/	Travel news	1442	2
6	https://www.theguardian.com/uk/travel/	Travel news	59	1
7	http://www.vogue.com/living/travel/	Travel news	5	0
8	http://www.dailymail.co.uk/travel/	Travel news	3296	2
9	http://www.bbc.com/travel/	Travel news	185	5
10	http://www.visitfinland.com/	Finland travel guide	88	4
11	http://www.kausalanautotarvike.fi/	Car accessories	368	1
12	http://www.utranuittotupa.fi/	Restaurant service	42	0
13	https://pkamknkirjasto.wordpress.com/tag/wartsila-talo/	Blog	29	0
14	http://pippurimylly.fi/	Restaurant service	110	0
15	http://vaarakirjastot.fi/paakirjasto	Location information page	9	0
16	http://javerstok.fi/	Catering service	42	0
17	http://www.lahdenmuseot.fi/museot/fi/hiihtomuseo/	Lahti City Museum website	397	17
18	https://news.zing.vn/du-lich.html	Travel news	433	1
19	http://www.huanqiu.com/	World news	749	3
20	https://visitkouvola.fi/ru	Kouvola travel guide	16	0

As can be seen from above data, websites about photo search and sharing service, as well as travel news and information have higher chance to have geotagged images in their content. In contrast, websites about blogs and businesses usually contain no geotagged images. We conclude that the seed plays a vital role in our system operation.

7 CONCLUSIONS

In this thesis we presented the Mopsi Image Crawler system for retrieval of photo images on the web, based on the concept of web crawler. Our system targets geotagged images, which have geographical coordinates information embedded in their EXIF metadata. The geotagged images retrieved by the system can be use as material in O-Mopsi, which is a mobile location-based orienteering game. The system is now running fully automatic on UEF's Mopsi server and its web GUI is available on both desktop computer and mobile platforms.

To study the working mechanism of web crawler, we discussed in detail its concepts and architecture, as well as available crawling algorithms. Then we describe our system architecture components and methods we used. We also developed a set of rules for extracting representative images. Based on the knowledge we learnt from other crawling algorithms, we designed our own system architecture and our own version of best-first search algorithm.

Our best-first search algorithm relies on content of web page to calculate relevance score between two connected pages, of which make our system a focused web crawler. The relevance calculation algorithm we design aims to get a score between a web page and its associated links. Our algorithm compares hostname of the link against the hostname of the page and count frequency of keywords in the link title to contribute to the relevance score. The experimental results show that while our crawling algorithm is slower than depth-first search and breadth-first search, it retrieves more geotagged images. As our system can schedule multiple executions of crawling tasks, the disadvantage in crawling speed of the algorithm can be compensated and the system can run in a long period of time. Thus, we choose our best-first search as the main algorithm used for crawling, while leaving depth-first search and breadth-first search as option for research purpose.

At the end of the thesis, we introduce briefly a method for determining location information from image text description based on geographic information retrieval. Since there are only 1% of web images are geotagged, the method can be useful for future development of the system to produce more location-based image content for the O-Mopsi game or other location-based services.

8 REFERENCES

[1] D. Quercia, N. Lathia, F. Calabrese, G. Di Lorenzo and J. Crowcroft, *Recommending Social Events from Mobile Phone Location Data, in 2010 IEEE International Conference on Data Mining,* 2010.

[2] P. Fränti, R. Mariescu-Istodor and L. Sengupta, *O-Mopsi: Mobile Orienteering Game for Sightseeing, Exercising, and Education, ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM),* vol. 13, no. 4, 2017.

[3] A. Tabarcea, Z. T. Wan, W. K. and P. Fränti, *O-Mopsi: Mobile Orienteering Game using Geotagged Photos, in International Conference on Web Inoformation Systems & Technologies (WEBIST' 13),* Aachen, 2013.

[4] X. Chen, S. Abhinav and G. Abhinav, *NEIL: Extracting Visual Knowledge from Web Data, in IEEE International Conference on Computer Vision (ICCV),* Sydney, NSW, Australia, 2013.

[5] I. Vänskä, *Using Location Information in Web Documents (MSc. Thesis),* University of Eastern Finland, 2004.

[6] D. Ahlers and S. Boll, *Retrieving Address-based Locations from the Web, in Proceedings of the 5th International Workshop on Geographic Information Retrieval,* New York, 2008.

[7] M. Kobayashi and K. Takeda, *Information retrieval on the web, ACM Computing Surveys (CSUR),* vol. 32, no. 2, pp. 144-173, 2000.

[8] C. Olston and M. Najork, *Web Crawling, Foundations and Trends in Information Retrieval,* vol. 4, no. 3, pp. 175-246, 2010.

[9] J. Cho, H. Garcia-Molina and L. Page, *Efficient Crawling Through URL Ordering, Computer Networks and ISDN Systems,* vol. 30, no. 1-7, pp. 161-172, 1998.

[10] A. M. H. Elyasir and K. S. M. Anbananthen, *Focused Web Crawler, in 2012 International Conference on Information and Knowledge Management (ICIKM 2012),* Singapore, 2012.

[11] Z. Markov and D. T. Larose, Crawling the Web, *in Data Mining the Web: Uncovering Patterns in Web Content, Structure and Usage,* New Britain, Wiley, 2007, pp. 6-12.

[12] P. Habibzadeh and S. G. Sciences, *Decay of References to Web sites in Articles Published in General Medical Journals: Mainstream vs Small Journals, Applied Clinical Informatics,* vol. 4, no. 4, 2013.

[13] C. Castillo, *Effective Web Crawling (Ph.D thesis),* University of Chile, 2004.

[14] The W3C Technical Architecture Group, *Architecture of the World Wide Web, Volume One,* The World Wide Web Consortium (W3C), 2004.

[15] The World Wide Web Consortium (W3C), *Web Characterization Terminology & Definitions Sheet,* 1999.

[16] B. Shneiderman and G. Kearsley, *Hypertext Hands-On! An introduction to a New Way of Organizing and Accessing Information,* Addison-Wesley, 1989.

[17] Z. Markov and D. T. Larose, *Data Mining the Web: Uncovering Patterns in Web Content, Structure, and Usage,* John Wiley & Sons, 2007.

[18] Chakrabarti, Soumen, M. v. d. Berg and B. Dom, *Focused crawling: a new approach to topic-specific Web resource discovery, Computer Networks,* vol. 31, no. 11-16, pp. 1623-1640, 1999.

[19] P. De Bra, G.-J. Houben, Y. Kornatzky and R. Post, *Information Retrieval in Distributed Hypertexts, in Proceedings of RIAO'94, Intelligent Multimedia, Information Retrieval Systems and Management,* 1994.

[20] S. Brin and L. Page, *The Anatomy of a Large-Scale Hypertextual Web Search Engine, in Proceedings of the seventh international conference on World Wide Web 7,* Brisbane, Australia, 1998.

[21] M. Hersovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalhaim and S. Ur, *The shark-search algorithm. An application: Tailored web site mapping, Computer Networks and ISDN Systems,* vol. 7, no. 1, pp. 317-326, 1998.

[22] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving,* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1984.

[23] A. Tabarcea, N. Gali and P. Fränti, *Framework for location-aware search engine, Journal of Location Based Services,* vol. 11, no. 1, pp. 50-74, 2017.

[24] N. Gali and P. Fränti, *Content-based Title Extraction from Web Page, in 12th International Conference on Web Information Systems and Technologies,* 2016.

[25] N. Gali, A. Tabarcea and P. Fränti, *Extracting Representative Image from Web Page, in International Conference on Web Information Systems and Technologies (WEBIST 2015),* 2015.

[26] International Organization for Standardization, *ISO 18383:2015 - Photography -- Digital cameras -- Specification guideline,* 2015. [Online]. Available: https://www.iso.org/standard/62322.html. [Accessed 14 April 2018].

[27] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms,* 3rd ed., MIT Press and McGraw-Hill, 2009.

[28] C. D. Manning, P. Raghavan and H. Schütze, *Introduction to Information Retrieval,* Cambridge University Press, 2008.

[29] J. Gelernter and W. Zhang, *Cross-lingual geo-parsing for non-structured data, in GIR '13 Proceedings of the 7th Workshop on Geographic Information Retrieval,* Orlando, Florida, 2013.

[30] T. Berners-Lee, *Uniform Resource Locators (URL): A Syntax for the Expression of Access Information of Objects on the Network,* World Wide Web Consortium (W3C), 1994.

[31] I. R. Brilhante, J. Macedo, F. M. Nardini and R. Perego, *TripBuilder: A Tool for Recommending Sightseeing Tours, in Advances in Information Retrieval - 36th European Conference on IR Research 2014,* Amsterdam, 2014.

[32] Y.-H. Hu and L. Ge, Chapter 11: GeoTagMapper: An Online Map-based Geographic Information Retrieval System for Geo-Tagged Web Content, *in M. P. Peterson, ed. International Perspectives on Maps and the Internet Lecture Notes in Geoinformation and Cartography,* Berlin, Germany, Springer Berlin Heidelberg, 2008, pp. 153-164.