# Extending Yioop! With Geographical Location Local Search

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment of the

Requirements for the

Degree Master of Computer Science

By

Vijaya Sinha

Spring 2012

SAN JOSÉ STATE UNIVERSITY

The Undersigned Writing Project Committee Approves the Writing Project Titled

Extending Yioop! With Geographical Location local search

By

Vijaya Sinha

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

_____

Dr. Chris Pollett, Department of Computer Science 01/20/2012

_____

Dr. Soon Tee Teoh, Department of Computer Science 01/20/2012

_____

Dr. Mark Stamp, Department of Computer Science 01/20/2012

**ACKNOWLEDGEMENTS**

I would like to thank Dr. Chris Pollett, for his encouragement, and guidance throughout this project. I also want to thank my committee members Prof.Soon Tee Teoh and Dr. Mark Stamp for their suggestions, time and support. I also would like to thank my family and friends for their support.

## Abstract

It is often useful when doing an internet search to get results based on our current location. For example, we might want such results when we search on restaurants, car service center, or hospitals. Current open source search engines like those based on Nutch do not provide this facility. Commercial engines like Google and Yahoo! provide this facility so it would be useful to incorporate it in an open source alternative.

The goal of this project is to include location aware search in Yioop!(Pollett, 2012) by using geographical data from OpenStreetMap(“Open Street map wiki”, 2012) and hostip.info (“DMOZ”, n.d.) database to geolocate IP addresses.

# Table of Contents

**List of figures**



**List of tables**

# 1 Introduction

Whether looking for the nearest car service center to repair your car or the nearest school for your child, all one needs to do is get connected to the internet and do a search. Location-based search services makes it easy for users to find information effectively. For example, if a user does a search on hospitals, the search engine should return results near the user's home or current location. More and more search engines are adding location based search capabilities. Major search engines like Google and Yahoo! already provide this facility but open source search engines like those based on Nutch do not provide this facility.

In order to build a location specific search engine the first and foremost important part is identifying the geographic data to be included, such form of data must include geo-coded information like latitude, longitude , elevation or zip code or some street address. The second part comes from ranking the search results according to some distance function so the search results are local to user.

The goal of this project is to enhance Yioop!(Pollett, 2012) with location aware search by indexing the geographic data from OpenStreetMap("Open Street map wiki", 2012) to provide search results based on the current location of the user using this indexed data.

Yioop!(Pollett, 2012) is an open source search engine developed by Dr. Chris Pollett. It is a search engine written completely in PHP and allows crawls of different file formats like Open Directory Project RDF data, Wikipedia xml dumps, etc. The ability of Yioop! to crawl different file formats allows one to have control over the kinds of results that will be returned while doing a search. This feature of Yioop! is used to index the open source data from Open Street Map to have it return geographic information.

In Section 2 of this report, two main sources of Geographic data namely planet.osm from OpenStreetMap project, hostip.info database, a community based project to geo-locate IP-addresses and the cloudmade web maps API("Cloudmade", 2012) will be discussed. In Section 3, Yioop's architecture and its features will be discussed. The design and implementation will be discussed in Section 4. Final section deals with test results and conclusion followed by references.

# 2 Geographical data

In order to have Yioop! return geographic results it was important to crawl and index geographic data into Yioop!. The source of this data was chosen as Open Street Map. To perform the geocoding, the database from hostip.info was used. Below section describes the same.

## 2.1 OpenStreetMap

OpenStreetMap("Open Street map wiki", 2012) is a collaborative project founded in 2004 which allows the distribution of free geographic data of the whole planet. The sources of OpenstreetMap include data from portable GPS devices, aerial photography, or people updating the data using their own local knowledge. The OpenStreetMap data is published under on Open Content License.

The OpenStreetMap data is available in different file formats namely PBF (efficient binary) or compressed OSM XML and o5m. For our implementation we used the OSM XML compressed file.

Planet.osm is data from OpenStreetMap map project. It is around 250 GB of uncompressed data (16 GB compressed). The OpenStreetMap project allows to extract downloads of individual countries as well. For testing purposes the extract from the state of Delaware was used.

### 2.1.1 OpenStreetMap structure

To index data using the OpenStreetMap dump into Yioop!, it was necessary to understand its structure so it could use its existing features to index the OSM data. The OpenStreetMap data is one Big XML file with data objects like nodes, ways and relations. Different logical features like street, bus stop, tram lines are defined using these data objects.

Nodes are the most important data primitive in the OpenStreetMap schema which consist of geographic information like latitudes and longitudes. They represent single geospatial point on the earth. They can be used to describe specific points on earth having tags like amenity=telephone or could be part of objects like ways and relations.

A node point in the OpenStreetMap has the following features:

**Table 1-Node attributes**

| name | Value | description |
|---|---|---|
| id | integer>=1 | Every node in Open Street map has an id which is unique among the nodes; this node id is used to identify that node. |
| lat | float >-90.0 and <90 up to 7 decimal places | Latitude coordinate .Every node in Open street map has this attribute. |
| lon | float >−180 and <180 | Longitude coordinate. Every node in Open street map has this attribute. |
| tag | a set of key /value pairs | It can take different kinds of key value pairs, an e.g. is name which is used to denote the heading of search result which is a |

| | | |
|---|---|---|
| | | node page. |

An example of node:

```
<node id="371190303" lat="37.4574202" lon="-121.9115501" user="Roozbeh" uid="6069"
visible="true" version="11" changeset="1130471" timestamp="2009-05-09T18:56:46Z">
  <tag k="highway" v="traffic_signals"/>
 </node>
```

Ways on the other hand describe things like railway lines, footpath, river, fence etc. They comprise of an ordered collection of nodes.

A way point in OpenStreetMap has the following features:

**Table 2-Way attributes**

| Name | Value | description |
|---|---|---|
| id | integer>=1 | Like a node tag in OSM every way point is denoted by a unique id which is unique among the ways. |
| nodes | List | A list of all the node points with their ids which are part of the way. |
| tag | Set | Same as node, the way tags can also be tagged with some key/value pairs. |

Example of a way point in OSM:

```
<way id="33289926" user="Roozbeh" uid="6069" visible="true" version="3" changeset="597814"
timestamp="2009-04-16T21:52:32Z">
        <nd ref="378341727"/>
        <nd ref="330146871"/>
        <nd ref="330146872"/>
        <nd ref="330146873"/>
        <nd ref="330146874"/>
        <nd ref="378341727"/>
        <tag k="created_by" v="Potlatch 0.10f"/>
```

```
        <tag k="landuse" v="residential"/>
        <tag k="name" v="Mobilodge of Milpitas"/>
 </way>
```

Relations are used to define relationships between objects by assigning roles to a given data primitive like node or way.

A relation in OpenStreetMap has the following features:

**Table 3-Relation Attributes**

| Name | Value | description |
|---|---|---|
| id | integer>=1 | Relations in Osm data are also denoted by a unique id. |
| tag | List | Key/value pair describing the relation. |
| members | list(ordered) | A relation may contain a set of nodes, way points with their associated role to denote an area or building. |

An example of a relation is:

```
<relation id="112134" user="Chris Lawrence" uid="36121" visible="true" version="1"
changeset="462766" timestamp="2009-04-13T03:41:12Z">
  <member type="way" ref="33106359" role="outer"/>
  <member type="way" ref="33106366" role="inner"/>
  <member type="way" ref="33106367" role="inner"/>
  <tag k="type" v="multipolygon"/>
</relation>
```

For the search results to be meaningful, it was important to recognize only certain points of interest as the dataset is very large. I decided to index only nodes and named ways with names forming the title of the page, description as tags value pairs and node references in case of a way page.

## 2.2 Geocoding

The process of assigning geographic information to web pages that provide information related to these locations is called geo coding.

Geocoding is an important component in getting local search results. In this project it was important to get the user's location, so we could use the geographic information from Open Street map to retrieve relevant results local to the user. PHP comes with a super global variable called $_SERVER['REMOTE_ADDR'] which can give the ip-address of the user's machine used to query the search engine, but there was a need to translate this ip-address to geo-location like the state, city or latitude ,longitude so we could get user's location. The hostip.info provides this ability to convert between an ip-address and geo-location. Below is a description of the same:

### 2.2.1 Hostip.info

The Hostip.info("DMOZ",n.d.) is a community-based project that provides complete raw data dumps to geolocate ip addresses. In order to use the hostip.info database the sql dump was imported into a sqlite database. The Hosip.info has the following schema:

countries table which consists of the countries name and country code identified by primay key id.

cityByCountry table which consists of the cities, states, longitude and latitude identified by primary key city. The cityByCountry table references the countries table by id.

The ip4 tables (255) containing the second and third quad of the ip4_address and the Corresponding ip4 table number denotes the first quad. The ip4 table references the citybycountry , table by city and countries table by id.

The following code snippet shows the conversion of an IP address to geolocation like country, state, country code, latitude and longitude:

```
$input= $ipaddress;

$table_a = "ip4_".$input[0];
$net_b = $input[1];
$net_c = $input[2];

$query =$db->query("SELECT cityByCountry.name AS city, cityByCountry.state AS state, countries.name AS country, countries.code AS country_code, cityByCountry.lat AS lat, cityByCountry.lng AS lon FROM $table_a, countries, cityByCountry WHERE $table_a.city = cityByCountry.city AND $table_a.country = cityByCountry.country AND $table_a.country = countries.id AND b =$net_b AND c =$net_c");
```

### 2.3 Cloudmade

Cloudmade("Cloudmade", 2012) was founded in 2007 and provides a number of APIs, and geographic based services to develop location based applications. It was a good choice for rendering our geographic data as it uses the Open Street Map data so the search results plotted on the map are in sync with the search results obtained. To plot the geographic data from Yioop! the Web Maps API from Cloudmade was used.

## 3 Yioop! Search Engine

Yioop! search engine (Pollett, 2012) is a GPLv3, open-source, PHP search engine developed by Dr. Chris Pollett. The aim of this project is to add location based search capabilities to Yioop!. It is based on Open Source Licensing and the source code is freely available. The version used for this project is version 0.80.

### 3.1 Introduction

The Yioop! Search engine can produce indexes of web pages or a set of web pages whose total number of pages are in the tens of millions. It allows users to control the sites that should be indexed and hence gives control over the search results that will be returned.

In order to maintain the indexes of pages crawled using Yioop! it maintains its own web archive format. The web archives are used for storing web pages, urls, and summary data and other kind of information like the mime type, encoding etc..

Apart from having a normal web crawl, Yioop! can also be used to crawl and index special formats like ARC files, Media Wiki XML files, ODP RDF files, it uses special iterators for each kind of file format. Hence having an iterator that can process pages in a way that Yioop! understand's is important so that we can get any kind of data to be indexed by Yioop! and have it return the kind of data we want.

## 3.2 Yioop's architecture in brief

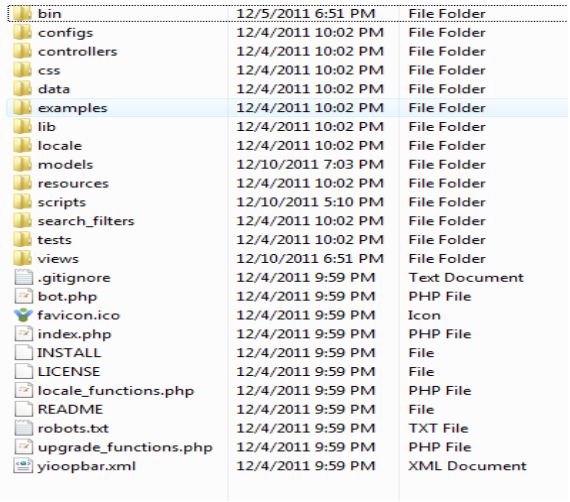
Yioop! is designed using the Model View Controller architecture and programmed in PHP. The minimum requirements to run Yioop! on one's machine are("Yioop Documentation", n.d.):

1. A web server like Apache or IIS.
2. PHP 5.3 or later

The PHP build should have curl libraries for downloading web pages, mb_prefixed functions , sqlite and GD graphics library.

It was necessary to understand the directory structure and source code of Yioop! as it has to be extended with geographical data search to look up geographic data.

The directory structure of Yioop! looks like the below figure 1:



| | | |
|---|---|---|
| bin | 12/5/2011 6:51 PM | File Folder |
| configs | 12/4/2011 10:02 PM | File Folder |
| controllers | 12/4/2011 10:02 PM | File Folder |
| css | 12/4/2011 10:02 PM | File Folder |
| data | 12/4/2011 10:02 PM | File Folder |
| examples | 12/4/2011 10:02 PM | File Folder |
| lib | 12/4/2011 10:02 PM | File Folder |
| locale | 12/4/2011 10:02 PM | File Folder |
| models | 12/10/2011 7:03 PM | File Folder |
| resources | 12/4/2011 10:02 PM | File Folder |
| scripts | 12/10/2011 5:10 PM | File Folder |
| search_filters | 12/4/2011 10:02 PM | File Folder |
| tests | 12/4/2011 10:02 PM | File Folder |
| views | 12/10/2011 6:51 PM | File Folder |
| .gitignore | 12/4/2011 9:59 PM | Text Document |
| bot.php | 12/4/2011 9:59 PM | PHP File |
| favicon.ico | 12/4/2011 9:59 PM | Icon |
| index.php | 12/4/2011 9:59 PM | PHP File |
| INSTALL | 12/4/2011 9:59 PM | File |
| LICENSE | 12/4/2011 9:59 PM | File |
| locale_functions.php | 12/4/2011 9:59 PM | PHP File |
| README | 12/4/2011 9:59 PM | File |
| robots.txt | 12/4/2011 9:59 PM | TXT File |
| upgrade_functions.php | 12/4/2011 9:59 PM | PHP File |
| yioopbar.xml | 12/4/2011 9:59 PM | XML Document |

**Figure 1: Yioop! directory structure**

The first folder inside a Yioop! directory is the bin folder which has the fetcher.php and queue_server.php which are used to start a crawl. The queue_server.php is responsible for maintaining the list of urls to be crawled and also information about which sites have been crawled where as the fetcher is responsible for actually downloading the web pages provided by the queue_server. The index archives that are used while doing a search are also created by the queue_server.php.

The controller's folder contains the controller classes which are responsible for calling the required model or view.

The css folder holds the style information which controls how any of the view is rendered. Some style information was added to this style.css to display the maps in the search view.

The lib folder consists of subfolders like archive_bundle_iterators, compressors, index_bundle_iterators, processors and stemmers as well as other classes that handle things like indexing, storing data to files and parsing. The archive_bundle_iterator contains different kinds of iterators for iterating over different types of arc files. An iterator was created for OpenStreetMap to iterate over the Planet.osm file and return html pages to the fetcher.php.

The models folder contains the subclasses of the models used to access the database and archive folders for storing the web pages.

The scripts folder consists of the JavaScript files. The JavaScript functions used to render the map for the geographical data are also included in the scripts folder.

The views folder contains the various view classes which contain the required HTML for rendering the various views. It also contains folders like elements, layouts and helpers.

In addition to Yioop! application directory, Yioop! also consists of a WORK_DIRECTORY to store the crawled pages and serve the search results.

## 3.3 Manage Archive crawls

The project required getting a crawl of the Open street map data in the form of an archive crawl. Archive crawl in Yioop! is different from web crawl and requires a different setup. Using archive crawls one can obtain recrawls of previous crawls as well as crawls of different archives like arc, MediaWiki XML and ODP RDF.

In order to crawl arc files an IndexData(timestamp) folder needs to be created under WORK_DIRECTORY/cache on the queue server. Inside this folder a single file like arc_description.txt needs to be created. The text inside this file appears in the drop down under Crawl or Arc Folder to re-index. One needs to select that option in order to crawl that particular arc file. In addition, we should create a folder like Archive (same timestamp) under WORK_DIRECTORY/cache on each of the fetchers participating in the crawl. Again this folder needs to have a file like arc_type.txt saying what kind of archive bundle it is. In our case the text was:

OsmArchiveBundle

With proper setup the Yioop archive crawl interface should display your archive as:

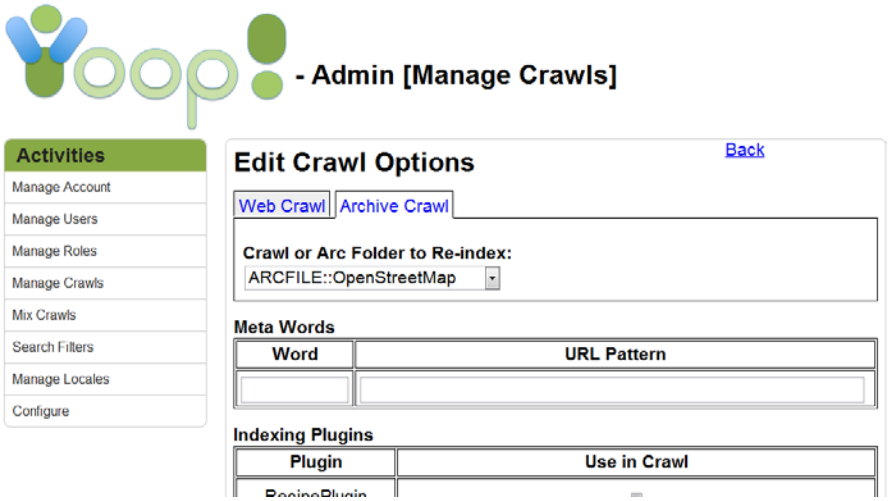



**Figure 2: Manage Archive crawl**

# 4 Design and Implementation

## 4.1 Design

For now Yioop! does not have any capability to return geographic search results, so let's say a user comes to the Yioop! search engine and wants to know all the search results for the parks in a particular area. In present scenario he will only get search results from web pages containing the keyword park contrary to what the user desires. The reason for this is that Yioop! currently do not have any geographical data in its index. It can only provide information from the web pages that it has crawled. A lot of search engines are equipped with this facility to return geographical search results. By providing Yioop! with this capability we can have Yioop! also return geographic results is response to queries related to locations.

This project needs to be incorporated with Yioop! search engine and it should use the same framework as Yioop! uses now.

In order to have Yioop! return geographic search results our design should basically focus on below four components:

**Process**: The process of obtaining location information is pretty straightforward. The user sends query term to the Yioop! search engine. The Yioop! search engine send the query to the given index containing the geographic data which returns the pages formed from the Open street Map data containing those keywords. These results are then ranked in ascending order of distance.

**Geocoding:** Geocoding refers to the process of assigning geospatial information like latitude and longitude to the user performing the search. A user coming to the Yioop! search engine has a IP information which can be mapped to its corresponding geo location, which in turn can be used to rank the search results using a distance factor.

**Distance calculation:** The ranking mechanism includes distance calculation for each of the search result. The distance is calculated on the assumption that earth is flat model which gives faster results with the loss of little bit of precision.

**Ranking:** A distance is calculated based on the location of user and the location of search results which is then added to the overall distance of the total score.

The overall architecture of the location based search facility can be represented as:

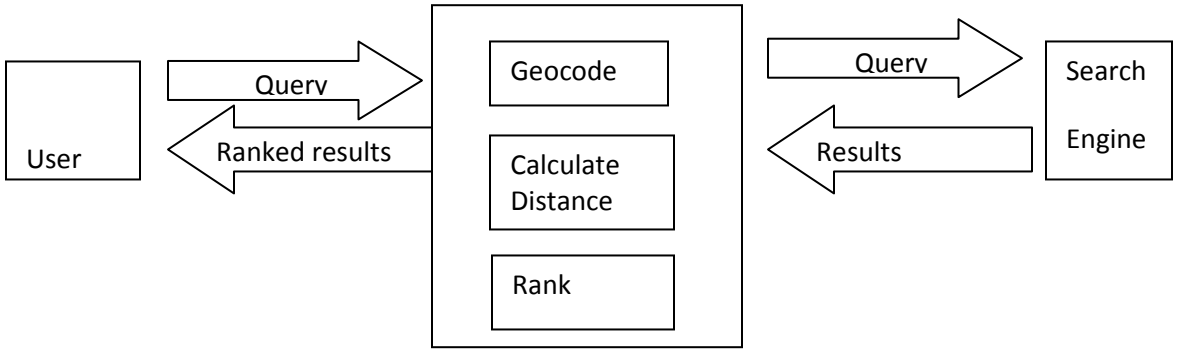


**Figure 3: Overall architecture**

The goal of this project was to enhance Yioop! to return geographic local search results instead of global search results. The second part of the implementation required the search results to be plotted on a map. In order to have Yioop! return geographic results it was important to index geographic data in Yioop!.

In order to implement this project our strategy was to first to identify the source of geographic data and see how it could to be incorporated into Yioop!. Yioop! comes with a number of archive_bundle_iterators which can iterate over different kinds of data. Using this feature of Yioop! we could make our own iterator which could iterate over our plant.osm file, as this data is iterated and returned to the fetcher it could be processed in a similar fashion like a web page based on the mime type.

Secondly we should be able to rank the search results returned by Yioop! locally to the user's location. For this we need to identify user's location which could be done by identifying the user's ip address, next we need a geocoder which could geocode the ipaddress to some kind of location identifier like the country, city or latitude and longitude. To do this geo-coding the hostip.info dump could be used. Once the geocoding was done we needed to rank the search results using a distance factor. We can see the flow of our design of osm_bunde_iterator using the following sequence

diagram:

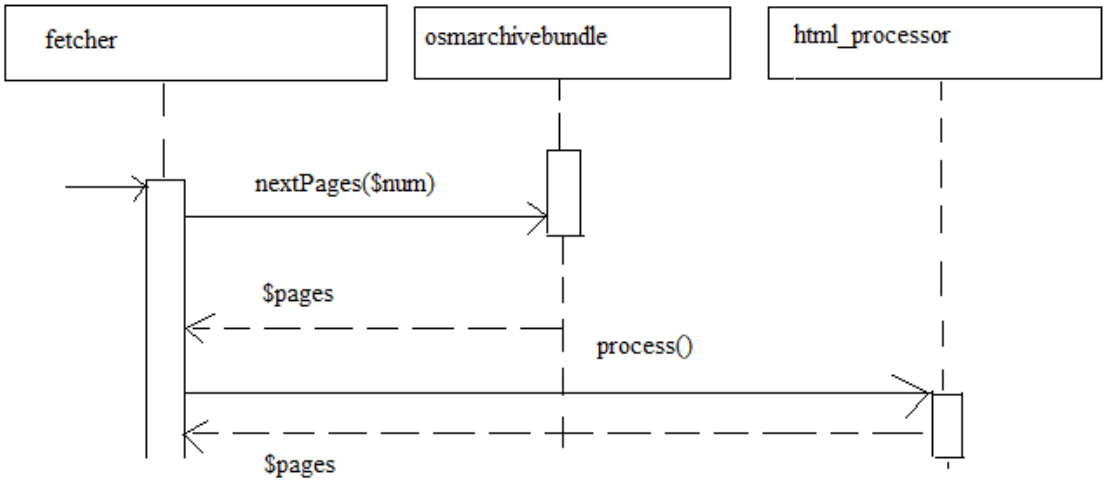



**Figure 4:Sequence digram**

## 4.2 Implementation

In order to implement this project, the Yioop! Search engine was thoroughly analyzed in terms of how it crawl's web pages and indexes them for later retrieval. The following observations were made:

The queue_server and fetcher are the two most important components related to crawl. The queue server is the main coordinator of the crawl. It maintains a priority queue of the list of url's to be crawled and gives them to the fetcher, the information is provided to fetcher in the form of schedules. The fetcher receiving the batch of url's, processes it using the various processors's, builds a mini inverted index of words which appear in a particular document and gives this result back to the queue_server which merges this result into current index.

In order to crawl an archive, Yioop! uses various kinds of archive_bundle_iterators placed under the lib folder which iterates over the archive placed in the Archive folder in WORK_DIRECTORY/cache.

In order to iterate over the planet.osm data, the various archive_bundle_iterators were studied in order to understand how various archive_iterators are implemented to return data which are not regular web pages as processed by Yioop!.

The archive_bundle_iterator consists of a base class called archive_bundle_iterator that every archive_bundle _iterator must extend. The main components of this archive_bundle_iterator are:

$iterate_timestamp - The timestamp of the archive being iterated over.

$result_timestamp - The timestamp of the archive used to store results later on used to look up search results.

$end_of_iterator - a Boolean variable which denotes the end of file you are trying to iterate over. The iterator will continue to return pages until it reaches the end of file denoted by a true value of this variable.

In addition to these variables the archive_bundle_iterator consists of two functions called:

nextpages() –this function is called by the fetcher until the $end_of _iterator is not true. The function takes a parameter $num of the number of pages to return from the iterator.

reset()- this function reset the iterator to the start of the iterator. This function is called at the start of crawl.

All archive_bundle_iterators must override the functions nextpages() and reset() with their desired behaviour.

On careful study of other archive_bundle_iterators and the way the fetcher calls the archive_bundle_iterarators to process the pages returned by these itertators, an itertator to iterate over the osm data was written and added to the archive_bundle_iterators folder.

### 4.2.1 Indexing geographical data in Yioop!

Once we have the osm_bundle _iterator, selecting Archive: OpenStreetMap from the drop down menu of the archive crawl from the Yioop! Interface, calls the osm_bundle_iterator to return the pages that contain geographic information.

Since Yioop!(Pollett, 2012) can process data using any of its processors namely html, image, pdf, png, ppt, xml etc it was necessary to return the data from the iterator in any of the given mime types so the appropriate processor can be called based on the mime-type. As a result of this analysis, the osm_bundle_iterator was designed to return results as html pages.

The planet.osm(“Open Street Map”, 2012) is one big compressed file Xml file, below is a snapshot of the planet.osm file :

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="OpenStreetMap planet.c" timestamp="2011-02-16T01:11:04Z">
<!-- copyright="OpenStreetMap and contributors" attribution="http://www.openstreetmap.org/copyright/"
license="http://creativecommons.org/licenses/by/2.0/" -->

  <bound box="-90,-180,90,180" origin="http://www.openstreetmap.org/api/0.6" />
<changeset id="878132" created_at="2009-04-21T11:33:10Z" num_changes="5" closed_at="2009-04-
21T11:34:22Z" open="false" min_lon="141.4289062" min_lat="43.0394291" max_lon="141.4390275"
max_lat="43.0435509"  user="masaminh" uid="93901">
<tag k="comment" v="Add a tertiary road" />
<tag k="created_by" v="JOSM" />
</changeset>
<node id="270387" lat="50.8777604" lon="-1.5338646" timestamp="2006-08-31T14:39:25Z" version="1"
changeset="99256" user="nickw" uid="94">
<tag k="created_by" v="osmeditor2" />
<tag k="name" v="Jacklin &amp;amp; Escuela"/>
  <tag k="operator" v="VTA"/>
  <tag k="route_ref" v="46;47;66"/>
 </node>
<way id="33289926" user="Roozbeh" uid="6069" visible="true" version="3" changeset="597814"
timestamp="2009-04-16T21:52:32Z">
  <nd ref="378341727"/>
  <nd ref="330146871"/>
  <nd ref="330146872"/>
  <nd ref="330146873"/>
  <nd ref="330146874"/>
  <nd ref="378341727"/>
  <tag k="created_by" v="Potlatch 0.10f"/>
  <tag k="landuse" v="residential"/>
  <tag k="name" v="Mobilodge of Milpitas"/>
 </way>
   <relation id="110112" user="Roozbeh" uid="6069" visible="true" version="2" changeset="380828"
timestamp="2009-04-10T03:27:48Z">
  <member type="node" ref="371266568" role="via"/>
  <member type="node" ref="371267241" role="location_hint"/>
  <member type="way" ref="32969825" role="to"/>
  <member type="way" ref="32969906" role="from"/>
  <tag k="restriction" v="no_left_turn"/>
  <tag k="type" v="restriction"/>
 </relation>
</osm>
```

From the snapshot it is clear that the nodes appear first followed by the ways and the relations. So in the iterator, the file was read using an xml parser called XMLReader to read a node at a time, followed by the ways and relations, since the data set is very large so only named nodes

and ways were chosen for indexing so the search results that are returned look meaningful. The Xml reader reads the nodes at a time and creates an html page. Each of the search results displayed by the Yioop! has a title followed by a brief summary of that page, so having an html page for the geographic data would have the advantage of using the html processor to create the search results.

The html page for a node point has the form:

```
<html>
<head>
<title> Jacklin &amp;amp; Escuela </title>
</head>
<body>
<h1> nodeid -270387 lat 50.8777604  lon -1.5338646 operator vta </h1>
</body>
</html>
```

Similary for a way the html page would be of the form:

```
<html>
<head>
<body>
<title> Mobilodge of Milpitas </title>
<body>
<h1> wayid 33289926  nodeid 378341727 nodeid 330146871 nodeid 330146872
nodeid378341727  landuse  residential created_by Potlatch 0.10f </h1>
</body>
</html>
```

Once the html pages are created, some other key values must be set by the iterartor so it can be recognized by the fetcher and the fetcher can create a mini inverted index of the pages and pass it to the appropriate processor. After some experimentation, the parameters that must be set by the iterator are as shown below:

```
$site[self::PAGE] ="<html><head><body><title> Jacklin &amp;amp; Escuela </title><h1> nodeid -
270387 lat 50.8777604  lon -1.5338646 operator vta </h1></body></head></html>";
 $site[self::TYPE] ="text/html";
 $site[self::HTTP_CODE] = 200;
 $site[self::URL] = "http://www.Openstreetmap.org/nodeid/".'nodeid'.$id;
 $site[self::ENCODING] = "UTF-8";
 $site[self::SERVER] = "unknown";
 $site[self::SERVER_VERSION] = "unknown";
 $site[self::OPERATING_SYSTEM] = "unknown";
 $site[self::HASH] = FetchUrl::computePageHash($site[self::PAGE]);
```

Using the osm_bundle_iterator the geographic data is crawled and indexed in Yioop!.

The following figure shows the output once the geographic data is indexed in Yioop! and a search is performed:

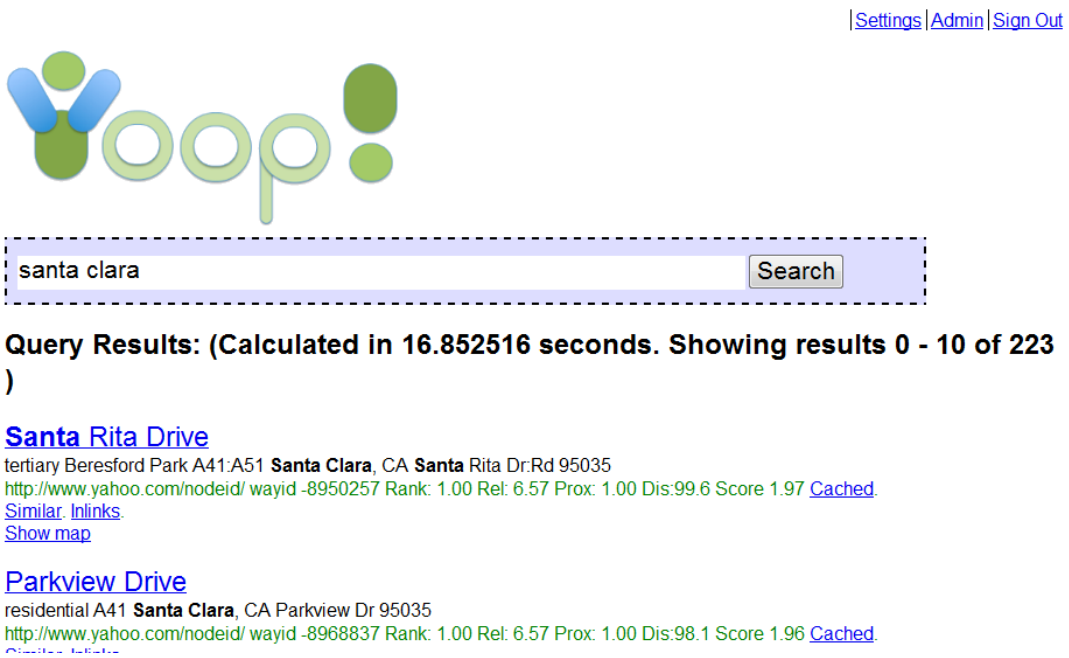



**Figure 5:Search results**

### 4.2.2 Plotting results on the map

To plot the results on the map the Web maps API from cloudmade was used.

Below is an example of a simple map using the API:

```
var cloudmade = new CM.Tiles.CloudMade.Web({key:
'8ee2a50541944fb9bcedded5165f09d9'});
var map = new CM.Map('cm-example', cloudmade);
map.setCenter(new CM.LatLng(51.516, -0.138), 15);
```

The above code snippet creates a new Cloudmade tile object with the API key, then creates a new Map object with the CM.Map class and uses the setCenter function which decides where to center the Map with zoom level 15.

Thus we can see that the API needs a set of latitude, longitude information to plot the map. So in order to plot the search results on the map, each of the search results must have latitude, longitude information.

### 4.2.3 Modifications to Yioop! Architecture

From the above discussion we know that the nodes pages have that spatial information like latitude and longitude but the way pages do not have such information so it was important to include the latitude and longitude information in the description of way pages. To achieve this part, some modifications were made to the Yioop! architecture as:

1. **Modifications to phrase_model**: The phrase_model was modified to detect way pages and if it was a way page, identify the unique nodeid's that are part of the way page, create a word _iterator for that nodeid and call nextDocsWithWord() on that word to get the node pages corresponding to that nodeid and ultimately get the latitude, longitude

information from its description, and add that latitude longitude to the description of the way pages.

2. **Modifications to displayresults_helper:** The displayresults_helper was modified so that garbage data like nodeid's and wayid's could be removed from the summary of the search results. Using regular expressions the nodeid's, wayid's, lat, lon string were identified and removed.

3. **Modifications to models.php:** The models.php was modified so that it the search on geographic data was made, the getSnippets function which shortens the summary data of a page was not called .This was done because the getSnippets function trimmed the summary data and so the summary did not contain the latitude, longitude information which was necessary to plot the data on the map.

4. **Addition of map.js to scripts:** A map.js was added to the scripts folder to include the web maps API to plot the search results on a map.

5. **Modification to search view:** The search view was modified to include a div element where the map could be plotted in the search front end and a show map link was provided along with all search results, clicking the link would bring the map on the right.

6. **Modification to styles.css:** The added div had to be styled so it occupied the area in the right side so in styles.css the div added to the search view was floated right.

Below is a snapshot of the output after making the above changes:


**Figure 6:Map with search results**

### 4.2.4 Provide local search capabilities for geographic search

The last part of the implementation required that the search results were local to the user performing the search, for this one required to track the location of the user, PHP provides for a super global variable called $_SERVER [‘REMOTE_ADDR’] using which one can know the ipaddress of the user performing the search. In this way, using the ipaddress we can track the geolocation of the user.

Although a lot of geocoder’s are available like the ones from Google and Yahoo which use the reference data from TeleAtlas and Navtec, none of them are free. So to encode the data the Hostip.info database which is freely available was used.

In order to get the database to work with Yioop! whose default database is sqlite, the hostip.info data dump was imported into a sqlite database. The hostip.info database allows geocoding of ipaddress’s into geo-locations like:


**Figure 7: IP address to geolocation**

In this way we can obtain the geo-location of the user. At this point we have the geo-location of the user and geo-location of the search result. So using this information we could use a distance algorithm to rank results according to user's location and display the results closest to the user at the top followed by other results.

### 4.2.5 Distance calculation

The last part of the implementation required ranking the documents according to its proximity from the user so as to provide for local search capabilities.

Various algorithms that can be used for local search were studied .The most important of these were;

1. **Distance Calculation:** Distance calculations for use in GIS applications vary according to their approach, each of these have their pros and cons. They can mainly be divided into three groups, which are decided by how they choose to represent the earth. Assuming

earth as a flat model leads to a faster results in exchange of some precision. Applications built on top of a flat model calculate distance that is variation of Pythagorean Theorem.

There is another class of distance calculation algorithms that assume that earth is a spherical in nature and the distance calculation is based on great circle distance, which calculates the shortest distance between any two points on the sphere's surface. This sort of calculation is suitable when the requirement is for places that are far from each other and a greater precision is needed. The last category assumes that earth is an ellipsoid and makes distance calculations on the basis of Vincenty formula.

2. **Bounding Box filter :** Bounding box filtering helps when the dataset to be searched is very large so it is better to narrow down the search space and perform a search on the given search space.

3. **Query parsing:** Query parsing relies on the information passed in the query to perform the look up. There are two parts to this method, first of which requires determines the keyword to look for in the search query and the second part is concerned with finding the location specific details. This form of search is rather complex as it requires some preprocessing like normalizing the query and converting addresses to locations.

In order to rank documents in Yioop! the first approach called distance calculation was used as Yioop! already has a ranking mechanism which it uses to rank web pages.

At present Yioop! rank's the documents according to a score which is a summation of Document's rank and its relevancy, thus modifying this score for geographic searches such that the document's nearer to user where placed at the top would serve the purpose.

Thus the distance calculation takes this into factor and before the score for a document is calculated, an additional distance factor is added to the whole score of the document:

The function that calculates this distance is given as:

```
function calcDist($lat_A, $long_A, $lat_B, $long_B)
    {

    $distance = sin(deg2rad($lat_A))
            * sin(deg2rad($lat_B))
            + cos(deg2rad($lat_A))
            * cos(deg2rad($lat_B))
            * cos(deg2rad($long_A - $long_B));

    $distance = (rad2deg(acos($distance))) * 69.09;
    $actualscoredis = log(1/$distance+1);
    return $actualscoredis;
    }
```

The above function takes the user latitude ,longitude information obtained from the hostip.info database and the latitude, longitude of the search result ,calculates the distance as log(1/$distance+1), which is then added to the total score, hence the documents that are nearer to user's location are ranked higher than the rest of the search results.

# 5 Testing and results

The success of our location based search clearly depends on the ability to produce search results as relevant as possible to the search query and the ability to map these results to geographical coordinates. This test results of Yioop! were compared with search results obtained from Nominatim(“Nominatim”,n.d.), a search facility that uses Open Street map and then with other location based search engines like Yahoo local and Askcity.

**Testing relevance of search results**

When considering a location based search engine, the relevancy of search results is measured by relevancy of search results related to the subject of the query term and its geographical location related to the location of the query.

A location based query is a query that is directed at a search for a resource or business in a particular area and the quality of results can only be determined if the search results answers the following questions.

How well the search results are related to the query term?

How well they map to the location of the query term?

Two metrics used to determine the quality of search engine are

**Precision:** precision is a measure of how many of the documents returned to a given query are relevant(“Precision Wikipedia”, n.d.)

Mathematically we can define precision as :

$$Precision=|Rel \cap Res||Res|$$

**Recall:** recall is a measure of how many of all known documents are retrieved by the system(“Recall Wikipedia”, n.d.).

Similarly we can represent recall as:

Recall=|Rel∩Res||Rel|

Any good search engine should have a high recall and low precision. Both of the above measures are often determined by the retrieval score assigned to documents which ranks the documents based on this score.

If we say the total score assigned to a document is Ts, then it can be calculated as the sum of all the weights used by the search engine Wt. Most of the times this weight consists of two sub weights Wt(rel) and Wt(imp).So we can say that Ts can be determined as:

$$Ts = Wt(rel) + Wt(imp)$$

In case of a location based search engine another parameter called Wt(loc) must also be considered, for eg : for a user searching on “parks santa clara” should not only get results related to parks but parks that are located in Santa Clara county. Thus we can say for any location based tool, the following three metrics are of particular importance:

1. Subject relevance
2. Document importance
3. Location of the search result

In other words now we can add another weight called Wt(loc) to rank results based on the location of the document

Or,

$$Ts= Wt(rel) + Wt(imp) + Wt(loc).$$

To test the relevancy of the location based results from Yioop! the same metrics of recall and precision were used . The test was conducted for the state of Delaware, with a collection of total 455145 documents and recall and precision were observed for six queries:

| | |
|---|---|
| Query 1 | “hospital” |
| Query 2 | “park” |
| Query 3 | “high schools” |
| Query 4 | “hotels” |
| Query 5 | “bus stop” |
| Query 6 | “starbucks” |

**Test relevance using Nominatim**

The target to compare results was chosen as Nominatim, a search facility that facilities searching data using Open street Map data and powers the search page of OpenStreetMap.org. To obtain results for what would be considered relevant, three users were asked to look at the search results and give the numbers of what they would consider as relevant and then relevancy metrics like recall and precision were calculated for both of them by taking an average of relevancy score from all three users.

To calculate the recall for the search engine that returned smaller set of results we considered the relevance value of search facility that returned more results and added that to its relevancy score. On querying Yioop! and Nominatim with above queries, the findings were as given in the following table:

| **Q** | **$T_y$** | **$T_N$** | **$R_Y$** | **$R_N$** | **$Re_Y$** | **$Re_N$** |
|---|---|---|---|---|---|---|
| Query1 | 10 | 25 | 8 | 12 | 0.57 | 0.48 |
| Query2 | 5 | 8 | 4 | 7 | 0.50 | 0.87 |
| Query3 | 1 | 9 | 1 | 6 | 0.16 | 0.66 |
| Query4 | 6 | 17 | 5 | 15 | 0.31 | 0.88 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Query5 | 7 | 10 | 5 | 8 | 0.20 | 0.80 |
| Query6 | 16 | 11 | 10 | 9 | 0.62 | 0.81 |

Where Q = queries

$T_y$= Total results from Yioop

$T_N$=Total results from Nominatim

$R_Y$= Average relevant documents from three users Yioop!

$R_N$= Average relevant documents from three users Nominatim

$Re_{Y=}$ Recall for Yioop!

$Re_{N=}$ Recall for Nominatim

The following graph gives the graphical representation of the test results as depicted in the table:

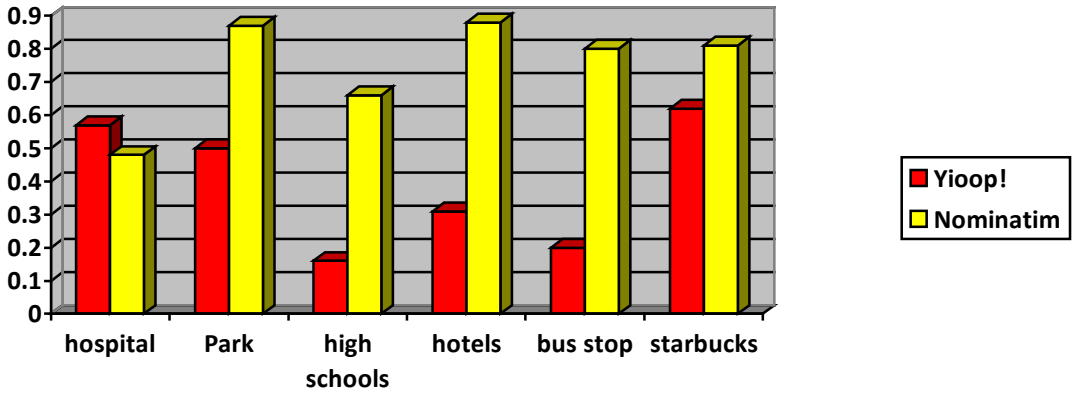



**Figure 8: Recall comparison of Yioop! and Nominatim**

Now we can calculate the precision for both the search facilities from the same table as:

| **Q** | **$T_y$** | $T_N$ | $R_Y$ | $R_N$ | $P_Y$ | $P_N$ |
|---|---|---|---|---|---|---|
| Query1 | 10 | 25 | 8 | 12 | 0.80 | 0.48 |
| Query2 | 5 | 8 | 4 | 7 | 0.90 | 0.87 |
| Query3 | 1 | 9 | 1 | 6 | 1.00 | 0.66 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Query4 | 6 | 17 | 5 | 15 | 0.90 | 0.88 |
| Query5 | 7 | 10 | 5 | 8 | 0.71 | 0.80 |
| Query6 | 16 | 11 | 10 | 9 | 0.62 | 0.81 |

Where Q = queries

$T_y$= Total results from Yioop

$T_N$=Total results from Nominatim

$R_Y$= Average relevant documents from three users Yioop!

$R_N$= Average relevant documents from three users Nominatim

$P_{Y=}$ Precision for Yioop!

$P_{N=}$ Precision for Nominatim

The following graph gives the graphical representation of the test results as depicted in the table:

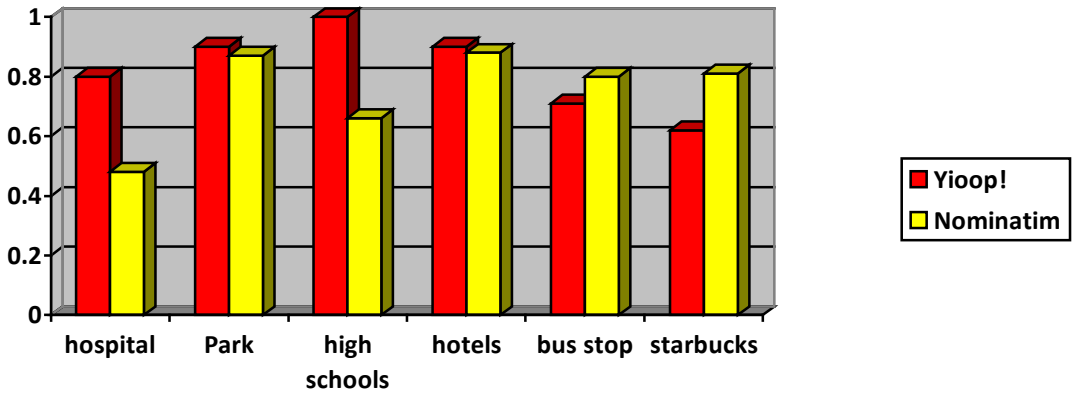



**Figure 9: Precision comparison of Yioop! and Nominatim**

The above results show that the search results as obtained by Yioop! gave good values of precision but have varying values of recall as compared to Nominatim The difference in the recall values can be attributed to the fact that Nominatim might be using a different version of Open street Map, it could also be of the difference in the search algorithm used by Nominatim and our search algorithm producing different results.

**Test Relevance using search results from other search engines**

Another test was conducted by comparing the search results as returned by Yioop! and other location based search engines using different sources of geographical data. The test was conducted using the same set of queries.

The relevancy score was calculated based on the search results from two different location based search services called askcity and yahoo local for the state of Delware compared with the ones returned by the Yioop! search engine using OpenStreetMap data.

The findings for Precision for the three search results were recorded in the same manner by taking the average of relevancy score from all the three users. The findings were as follows:

| **Q** | **Ty** | **$T_{YH}$** | **$T_{AK}$** | **$R_Y$** | **$R_{YH}$** | **$R_{AK}$** | **$Re_Y$** | **$Re_{YH}$** | **$Re_{AK}$** |
|---|---|---|---|---|---|---|---|---|---|
| Query 1 | 10 | 30 | 33 | 8 | 25 | 28 | 0.48 | 0.75 | 0.84 |
| Query 2 | 5 | 44 | 40 | 4 | 40 | 38 | 0.87 | 0.90 | 0.90 |
| Query 3 | 1 | 34 | 32 | 1 | 28 | 30 | 0.66 | 0.77 | 0.93 |
| Query 4 | 6 | 23 | 20 | 5 | 17 | 18 | 0.88 | 0.70 | 0.90 |
| Query 5 | 7 | 45 | 44 | 5 | 38 | 40 | 0.80 | 0.80 | 0.90 |
| Query 6 | 16 | 13 | 17 | 10 | 10 | 12 | 0.81 | 0.66 | 0.70 |

Where Q = queries

$T_y$= Total results from Yioop

$T_{YH}$=Total results from yahoo

$T_{AK}$ = Total results from askcity

$R_Y$= Average relevance score for Yioop!

$R_{YH}$= Average relevance score for yahoo

$R_{AK}$= Average relevance score for askcity

$Re_Y$= Recall for Yioop!

$Re_{YH}$= Recall value for Yahoo!

$Re_{AK}$= Recall value for Askcity

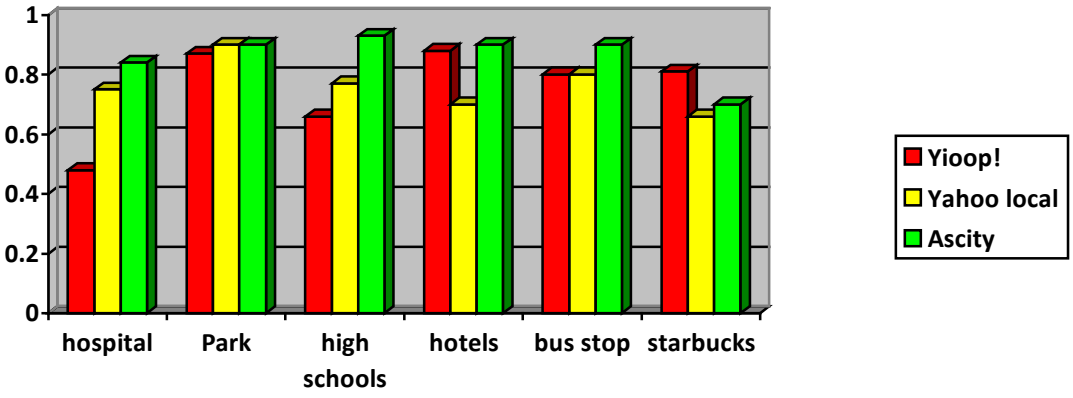



**Figure 10: Recall comparion of Yioop! with other search engines**

Next all the three search engines were compared for their precision values. We queried all the three search engines with the above three queries and calculated their precision value. The results were as follows:

| **Q** | **$T_y$** | **$T_{YH}$** | **$T_{AK}$** | **$R_Y$** | **$R_{YH}$** | **$R_{AK}$** | **$P_Y$** | **$P_{YH}$** | **$P_{AK}$** |
|---|---|---|---|---|---|---|---|---|---|
| Query 1 | 10 | 30 | 33 | 8 | 25 | 28 | 0.80 | 0.83 | 0.84 |
| Query 2 | 5 | 44 | 40 | 4 | 40 | 38 | 0.90 | 0.90 | 0.95 |
| Query 3 | 1 | 34 | 32 | 1 | 28 | 30 | 1.00 | 0.82 | 0.93 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Query 4 | 6 | 23 | 20 | 5 | 17 | 18 | 0.90 | 0.73 | 0.90 |
| Query 5 | 7 | 45 | 44 | 5 | 38 | 40 | 0.84 | 0.80 | 0.90 |
| Query 6 | 16 | 13 | 17 | 10 | 10 | 12 | 0.62 | 0.76 | 0.70 |

Where Q = queries

$T_y$= Total results from Yioop

$T_{YH}$=Total results from yahoo

$T_{AK}$ = Total results from askcity

$R_Y$= Average relevance score for Yioop!

$R_{YH}$= Average relevance score for yahoo

$R_{AK}$= Average relevance score for askcity

$P_Y$= Precision for Yioop!

$P_{YH}$= Precision value for Yahoo!

$P_{AK}$= Precision value for Askcity

The above results can be seen graphically by the following graph:

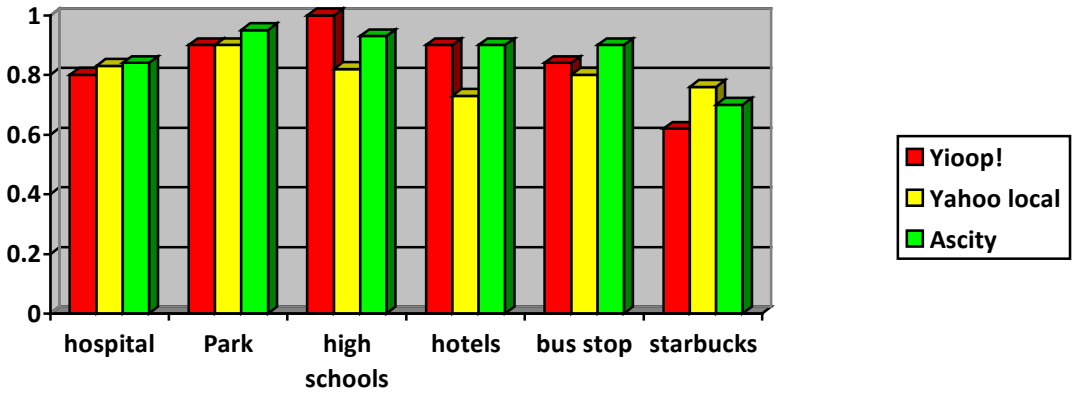



**Figure 11: Precision comparison of Yioop! and other search engines**

From our observation of the relevance results we can see that the results obtained by the search engine have a high precision and is accurate in terms of retrieving relevant documents. Hence justifying our design and extracting relevant information from the dataset but the recall value seems to differ in most of the search results because of size of dataset. This will produce good results in terms of recall as the dataset is updated and is more accurate.

# 6 Conclusion

Location based search facilities are finding it's place in all kinds of applications and services like GPS enabled phones, social networking sites or business's looking to enhance their services.
In this project we have tried to implement a similar application by combining Yioop! with spatial information and map based service.

Adding spatial information to the Yioop! search engine was made possible by using the open source data from OpenStreetMap project which is being used by a large number of developers to build GIS applications. By using an archive_bundle_iterator, addition of this geographic data was made possible. Once the geographic data was crawled and made part of the index, the search results were cleaned so the search results could look meaningful. Also spatial information like latitude, longitude had to be included in all the search results to facilitate plotting that information on a map. The Web-Map API's from cloudmade which comes bundled with a set of classes, functions and events were used to plot the search results on a map. Last part of the implementation included sorting results based on the distance and various algorithms were studied and ultimately one algorithm which sorts the results based on the log of distance was used.

The Yioop! Framework with its capability to crawl data other than HTML, pdf, ppt or XML like Media Wiki files and ODP RDF data was especially helpful in crawling the OpenStreetMap data and indexing it so that searches on the same could be performed. Improvements to this project could include improving the ranking of search results by getting a bounding box and retrieving search results from the bounding box. This way the precision for local search results can be improved.

# 7 References


1. S.Buttcher, C. L.A. Clarke, G.V Cormack. Information Retrieval:Implementing and Evaluating Search Engines. MIT Press. 2010.

2. Planet osm-Wikipedia.(n.d). Retreived April 6,2011 from Wikipedia web page: http://wiki.openstreetmap.org/wiki/Planet.osm

3. Stefan Dlugolinsky, Michal Laclavik, Ladislav Hluchy (2010).Towards a search system for the web exploiting spatial data of a web document.

4. Yates, J.D, Xiaofang Zhou.Searching the web using the map. Retrieved from Proceedings of first International Conferance on Web Information Systems Engineering.19-June 2000.

5. J. Marquez, Jose E. Corcoles, Antonio Quintanilla. Scoring Results in a Geospatial Services Search Engine according to Geographic and Semantic Awareness. Retrieved from Proceedings of 7th International Conference on Next Generation Web services Practices(2011).

6. Yioop! Documentation. (n.d.) Retrieved Jan 10, 2011 from SeekQuarry web page:http://seekquarry.com/?c=main&p=documentation

7. Yecheng Yuan. Extracting Spatial Relations from Document for Geographic Information Retrieval. 19th International Conference on Geoinformatics. 2011

8. Hostip.info –DMOZ.(n.d).Retrieved Jan 10 2011 from Open Directory RDF Dump: http://www.dmoz.org/rdf.html

9. C. D. Manning, P. Raghavan and H. Schütze. Introduction to Information Retrieval. Cambridge University Press. 2008.

10. Philip O'Brien, Xiao Luo, Tony Abou-Assaleh, Weizheng Gao, Shujie Li(2009). Personalization of Content Ranking in the Context of Local Search

11. Cloudmade-Wikipedia(n.d). Retrieved 21 January 2012. http://blog.cloudmade.com/2011/01/17/958/.

12. Precision and recall – Wikipedia. (n.d). Retrieved Dec 1, 2011 from Wikipedia web page: http://en.wikipedia.org/wiki/Precision_and_recall

13. Makhoul, John; Kubala, Francis; Schwartz, Richard; and Weischedel, Ralph (1999); Performance measures for information extraction, in Proceedings of DARPA Broadcast News Workshop, Herndon, VA, February 1999

14. Location aware search with Lucene and Apache.(n.d). Retrieved Jan 12, 2012 from IBM developer's home page: http://www.ibm.com/developerworks/opensource/library/j-spatial/

15. Geographical distance.(n.d.). Retrieved April 08, 2012 from Wikipedia web page: http://en.wikipedia.org/wiki/Geographical_distance

16. Geographic Information system(n.d.) Retrieved April 08, 2012 from Wikipedia web page: http://en.wikipedia.org/wiki/GIS

17. Map projection(n.d.). Retrieved March 29, 2012 from Wikipedia web page: http://en.wikipedia.org/wiki/Spatial_reference

18. Osm data primitives(n.d.). Retrieved March 07, 2012 from Wikipedia web page: http://wiki.openstreetmap.org/wiki/Data_Primitives

19. OSM XML(n.d). Retrieved December 23, 2011 from Wikipedia web page: http://wiki.openstreetmap.org/wiki/.osm

20. Web maps API Cloudmade(n.d). Retrieved March 10, 2012 from cloudmade developer's page: http://developers.cloudmade.com/projects/show/web-maps-api