**Introduction**

**System description**

Our system to parse geographically oriented queries consists of 5 stages:

1. Geo tag search
2. Location search
3. Geo tag and Location merge
4. Geo tag type resolution
5. What type resolution

We have decided not to utilize Freeling or any other Part Of Speech (POS) taggers or Named Entity Recognizers (NER). The reason for this is because we found they give unreliable results if queries are all lowercase/uppercase or each word is capitalized. We have tested Freeling as well as Standford NER and we found they both give wrong results in such cases, therefore we decided to build our system without these methods.

We do not use any punctuations or letter capitalization as a clue for locations or named entities. For this reason each query before processing is made lowercase and any punctuation like commas, dots, semi-colons, colons are removed with the exception of dash ('-') as it could be part of a named entity.

**Geo tag search**

In this part of the system, the program finds possible candidates for geo tags ('in the south of', 'in', etc.). We have used regular expressions in several stages to find appropriate tags. Firstly, the system looks for longest geo tags, such as 'in and around' , 'in the south west of', etc. The short geo tags such as 'in' or 'of' are only matched in the absence of longs geo tags.

All the matched geo tags are then removed from the query and the sentence is divided into several smaller parts. For example, performing geo tags search for a query *'accommodation near fort william'* returns geoTag = 'near' and remaining sentence divided into ['accommodation','fort william']. If no geo tags are found, sentence is returned as is.

If no locations are found in the sentence in further steps, the geo tag search is repeated, omitting previously found tags. For example *'imaging in west covina'* firstly matches 'in west' as a geo tag. Since neither 'imaging' nor 'covina' are locations, the geo tag search is repeated. The second time, the matched geo tag is *'in'*, and consequently 'west covina' is found as a location.

**Location search**

The returned sentence from geo tag search is then queried in Dbpedia to determine if is contains a location. We have used are own script from part 1 of this project, and provided additional functionality to the code to return the probability of a string being a location. Dbpedia accessor functionality is detailed in Appdendices. A minimum threshold of 50% has been set. Only strings with more than 50% certainty are treated as locations.

The system uses the sentence returned from Geo Tag matching to build all possible combinations of up to 4 consecutive words. If sentence was split, then each split is treated individually. All these combinations are passed to Dbpedia one-by-one, starting with the ones having most words. If a location is found, it is removed from the original sentence and the whole procedure is repeated until no furhter removals are made. For example, ['accommodation','fort william'] is built to 'accommodation', 'fort william', 'fort' and 'william'. Using Dbpedia accessor, 'fort william' is used as a first choice and matched as a location. It is then removed from sentence and remaining ['accommodation'] is then queried. In total - Dbpedia accessor is used twice.

**Geo tag and Location merge**

In most sentences, there are more geo tags then there are locations; or geo tags do not correspond to a location. Therefore in this stage, the system matches locations with their appropriate geo tags. Only tags, which correspond to a location, are then treated as actual geo tags in further steps.

**Geo tag type resolution**

Once the actual geo tags have been identified, they are classed into categories given in the task specification. Since there is always only one location per query, it results into a single geo tag class. The logic between categorization is the following:

1. If the geo tag is a single word, this word is the name of the category in capitals. Ex. 'IN'
2. If there is a number followed by distance measure, the category is 'DISTANCE'
3. For directional geo tags ('south of', 'to the north west of'), the category is the direction ('south', 'north west') followed by leading preposition (or trailing, if leading is absent). Ex. 'SOUTH\_OF', 'NORTH\_WEST\_TO'.
4. If words 'in' and 'around' are present, the category is 'IN\_NEAR'
5. If combinations like 'next to' or 'near to' are present, the category is 'NEAR'
6. Otherwise, it is 'UNDEFINED'

**What type resolution**

Once the location and its corresponding tag have been removed from the sentence, the remaining string is treated as a 'WHAT' tag. In order to correctly identify the type of the this tag ('Map', 'Yellow page','Information') the system uses two stage approach:

1. Check if the string is a possible name of the street. We have used a list of all possible english street types and abbreviations[[1]](#footnote-2), such as STREET; STRT; ST; STR; STREETS; WY; WAY; WAYS and many others. Check *streetAbbrs.txt* text file for the full list. If any of these abbreviations are matched at the end of the 'what' tag, the whole 'what' tag is treated as an address and 'Location' is the whole original query.
2. If it is not an address, the system checks if the 'what' tag is a Yellow Pages query. To check this, we have used a rudimentary list of categories, for which one would consult yellow pages. This includes 'Housing' (flats, houses,apartments), 'Service' (rent, buy, sell), 'Professions'[[2]](#footnote-3) (plumbers, repairers,laywers) and 'Businesses and Institutions' (banks, police, schools). The full list can be found in *categoriesYP.txt*. If any of these categories are matched in the 'what' tag, then we treat this as a Yellow Page querying

If the category is not 'Map' and not 'Yellow page', it is considered as 'Information'.

**Acknowledgements**

For writing XML files we have used XML writer provided by Group 17, Jordi Ganzer and Nil Sanz.

For XML evaluation we have used the scorer provided by Daniel Duato Catalan, Luis Fabregues de los Santos and Javier Selva Castello.

**Results**

**Appendices**

**DBpedia accessor**

Just to recall our Dbpedia accessor - we are using SPARQL querying language to access this service and the querying is done in several stages:

1. Find resource with direct URL query with '<http://dbpedia.org/page/>' + name. For example for Berlin, the accessor would try <http://dbpedia.org/page/Berlin>. If resource is found, the system checks if it is a location by checking for appropriate tags in rdf:type ontology.



1. If there are no direct resources, the system performs a supervised keyword search in *rdfs:label* property. Supervised in the sense, that it does not return all found resources with given keywords, but instead only very similar in content and length. It mainly accounts for additional punctuation and capitalization. For example, searching 'brooksville florida' would only match 'Brooksville, Florida'. As per step 1, the resource is checked if location
2. If nothing is found, it looks for page redirections in *dbo:wikiPageRedirects*

**

This could sometimes account for frequent typos for popular locations or postal codes, for example querying 'Berlib' would return no results in previous steps, but would return a redirection. As per step 1, the check for location is performed.

1. Lastly if there are no direct resources or redirections, the accessor checks if the page contains 'dbo:Disambiguates', meaning there are many possible Dbpedia pages with such name. For example 'Bellvue' has 58 disambiguates



The system then accesses each of these disambiguates and checks if they correspond to a location. It then returns a score, how likely that the query is a location.

Score = (#disamb. which are locations)/(Total # of disamb.)

Dealing with geographical queries - if there are more then 1 disambiguate in the query, the one with the highest score is the only one treated as location.

1. http://pe.usps.gov/text/pub28/28apc\_002.htm [↑](#footnote-ref-2)
2. http://www.occupationsguide.cz/en/abecedni/abecedni.htm [↑](#footnote-ref-3)