Problem Description

Working title:
1: Top-k relevant spatio-temporal retrieval on knowledge graphs using keyword search
2: Spatio-temporal keyword search on RDF graphs.

Knowledge graphs allows for easy extraction of facts from an entity, and the structure makes it efficient to find multi degree relations between different entities and facts. The task is to study indexing and search techniques on graphs containing spatio-temporal data, and propose effective ways to retrieve the data contained in the graph based on keyword searches.

Summary

Write your summary here...

Preface

Spatio-temporal keyword search on RDF graphs.

Table of Contents

List of Tables

List of Figures

Abbreviations

Graph G Vertex V Edge E N3 RDF Subject Predicate Object

Chapter 1

Introduction

Creating efficient structures for computers to interpret information opens up many new areas for information retrieval. One method is structuring the information as a graph with vertices connected by edges (G(v, e)). In such a structure each vertex (v) can hold a small piece of information, and the edge (e) represents a relation. When information is modeled in this way, it can be called a knowledge graph (KG).

Graph based models allows complex relations to be modeled. Finding relations in a graph is also computationally cheaper than other models. Using standardized models such as the Resource Description Framework (RDF), which is created specifically for graph based data, has made it possible to develop efficient storage and retrieval solutions for graph data. One such solution is the framework Jena **?** for Java.

Using RDF projects like YAGO (Yet Another Great Ontology) **?** have created graphs containing spatial and temporal data (spatiotemporal). This makes it possible to apply reason to the dataset, and derive new information from what is already there. This is a feature that typically separates a KG from an ontology **?**. In a graph, spatiotemporal data will be described using at least 2 vertices, one for a start date, and one for a place. When modeling time intervals, multiple vertices are needed.

There are few applications using RDF as an information storage system. Some of the more well known are DBPedia, YAGO, and Creative Commons. These applications often consists of a large linked dataset, or in the case of Creative Commons, it is used for embedding licenses. The applications built with RDF usually don't have much utility. A common utility built on top of RDF data sets are tools used to describe social relations, though there have been developed applications such as MusicBrainz that use RDF for relations between songs, artists, albums and other tags given to entities.

Creating methods that allows for easy access to information in RDF data makes it possible to create applications with more utility. Keyword and text search in RDF data is still something being developed and improved. Using frameworks such as Jena, it is possible to build solutions for keyword search, and spatiotemporal search. In this thesis (paper?) methods for efficient keyword search on spatial and temporal data will be explored.

Research questions?
RQ1: How can spatio-temporal data be integrated into exiting keyword query methods for RDF data.
RQ2: What methods can be used to create more effective queries on RDF data.
RQ3: How do spatial and temporal RDF query methods differ from from other query methods.
RQ4: What methods can be used to query RDF data based on a users keyword search.

1.1 Previous work

There has been done some work keyword search on RDF graphs. This includes **?** and **?**. None of these takes spatial or temporal data into consideration. The methods for keyword queries outlined in the papers are based on a combination of indexing the graph and traversing subgraphs. The indexing is similar in both papers, but retrieval, scoring and traversal methods varies. One method of retrieval and scoring that is used as a baseline is a breadth first search for traversal and a minimal subgraph for scoring. These methods are easy to implement and understand, but often inefficient, and inaccurate.

In paper **?** some methods for indexing, searching and ranking keyword searches on spatial RDF graphs is proposed. These methods builds upon the methods outlined in **??**. The most substantial difference is the use of R-trees to index the spatial dimension of the graph. This is done so that a subgraph containing the keywords can be retrieved given a location. This method starts by finding the closest spatial vertex and uses that as the root for the subgraph. Subgraphs rooted in a spatial node requires different ranking, and there is proposed a set of rules in the paper.

R-trees are designed around the spatial dimension, but some work on modifications to include a temporal dimension has been done. Some research done on spatio-temporal tree structures include **??**. Most of the spatio-temporal trees are however created to be able to query and index moving or frequently updated objects. To include a temporal dimension some differences in the tree structures are made. Most approaches builds on the R-tree, but modifies the tree in different aspects. For the purpose of this thesis a different tree structure is not needed.

Chapter 2

Background

2.1 Definitions

- definitions (Find names, and some definitions)
- Result tree: Result r for a given query q with tokens Qt on an RDF graph G$\langle$V, E$\rangle$, where the result forms a minimum spanning tree Tm$\langle$V', E'$\rangle$, V' contains a set of tokens Tt, Tm is rooted in a place vertex p, so that V' $\subseteq$V, E'$\subseteq$E, and Tt$\subseteq$Qt

- Possible results: Collection of all result trees rt for a given query q

- Root nodes: Node with a yagoGeoEntity class, used as the starting node when traversing the graph.

- Accuracy: Score given to a result tree, where score is based on nodes n in the result tree, node distance nd from root, query q, and node tokens nt, and number of query words hit h so that sum(nt/len(q))/sum(n*(nd+1)) = score and nt (= q

- Queries: Any set of words from in English set of words.

- Pruning: Removal of unnecessary nodes n, where the removed nodes tokens nt is not part of the query tokens qt, nt$\nsubseteq$Qt.

Knowledge base, ontology and knowledge graph are terms with multiple definitions, and are often used interchangeably. The definitions here are used to clarify what they mean in this paper, and is not a definitive definition.

2.1.1 Knowledge Base

The term knowledge base can have many definitions, often because the terms knowledge base, ontology, and knowledge graph is used interchangeably **?**. In this paper the term knowledge graph will be defined as follows: "A knowledge base is a dataset with some formal semantics. This could include multiple axions, definitions, rules, facts, statements, and other primitives." **?**

2.1.2 Ontology

Like knowledge base, ontology does not have one clear definition. This is because the term have many different interpretations, and is often confused with other terms **?**. Most definitions of ontologies say that ontologies represent a schema, basic theory, or conceptualization of a domain, which knowledge bases do not. **?** To differentiate ontologies and knowledge bases in this paper the definition of an ontology will be "An ontology is an extended knowledge base that allows for semantic modeling of knowledge." With this definition, ontologies can be considered a specialized form of knowledge bases.

2.1.3 Knowledge Graphs

A broad definition of knowledge graph is "A knowledge graph acquires and integrates information into an ontology and applies a reasoner to derive new knowledge." **?** This definition is encompasses multiple technologies. In this paper we will use the more specific definition that fits the data used "We define a Knowledge Graph as an RDF graph. An RDF graph consists of a set of RDF triples where each RDF triple (s, p, o) is an ordered set of the following RDF terms: a subject s $\in$ U $\cup$ B, a predicate p $\in$ U, and an object U $\cup$ B L. An RDF term is either a URI u $\in$ U, a blank node b $\in$ B, or a literal l $\in$ L" **?**

Subject

In a RDF triple the subject is a URI or a blank node. The URI when used in the subject identifies an entity, or is an alias, different language or other variation of an entity. This subject can have relations to objects describing the same entity.

Predicate

Predicates are always a URI. URIs used for predicates differ from the ones used for subjects and objects in that predicates are of a type. A type is an identifier used to describe the relation between the subject and object.

Object

Objects have the widest range of possible entries. Like subjects and predicates, objects can be URIs. Object URIs can be entity identifiers like subjects, they can be class identifiers, or they can contain some data and a data type. Blank nodes are just that, blank, and literals are an atomic value.

2.1.4 Facts and entities

A fact is a term often used when describing knowledge bases, ontologies and knowledge graphs. Usually a fact is the smallest piece of information in such a system. In a knowledge graph this is a single RFD triple. Another term often used is entity. An entity is a collection of facts, usually from the same article. Entities can be linked together through facts. In such a fact, one entity is used as the subject, the predicate describes the relation, and the object is the other entity. In such a relation both entities will be URIs.

2.2 Existing knowledge graphs and ontologies

Currently two of the largest open technologies for knowledge graphs are Yago and DBPedia. Both these projects use automatic extraction from Wikipedia to create the graph, but differ in the ontology used to build the graphs. Yago also includes data from WordNet and GeoNames to accurately assign entities to classes. Both projects use RDF triples to create a knowledge graph.

2.2.1 Uses for Knowledge graphs

One of the uses of knowledge graphs today is to find and display a info box in search engines. This information is a compact set of facts that tries to fit the search query. Because of the graph structure of knowledge graphs the information in the info box can be adapted to the query by choosing the predicates and related facts closest related to the query. This makes it possible to create a set of information that can give the user a quick overview of the information retrieved by the query.

2.2.2 Yago

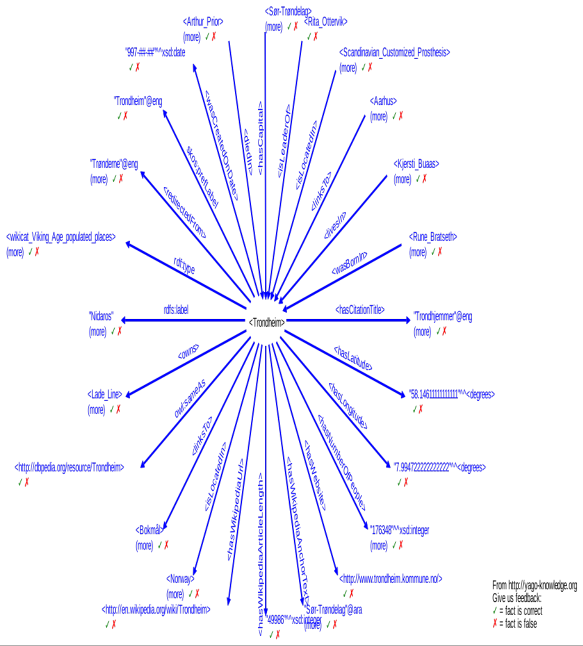
Yago is an acronym for Yet Another Great Ontology, and is main data source in this paper. The project describes it self as a knowledge base**?** and an ontology **?**, but is often described as a knowledge graph by others. In this paper Yago is described as a knowledge graph.

Temporal data

Yago have many entities with facts describing date and spatial data.**?** Date facts follows the ISO 8601 format, YYYY-MM-DD, and introduces # as a wildcard symbol. A fact can only hold information on a single point in time, and uses yagoDate as a data type in addition to information on the date.**?** In a date fact, the object holds the date information, and the predicate describes a connection between subject and date. “Nidaros_Cathedral wasCreatedOnDate 1300-##-## .” In this example the predicate wasCreatedOn is used to describe a relation between the subject and a date.

To describe a timespan two facts are required. One of the facts describes a start date, and the second en end date. Since an entity can have multiple date facts connected, all date

Figure 2.1 Trondheim as a subject in YAGO


<Trondheim>
<Sør-Trøndelag> (more) ✓ ✗
<Arthur_Prior> (more) ✓ ✗
<Rita_Ottervik> ✓ ✗
"997-##-##"^^xsd:date ✓ ✗
<Scandinavian_Customized_Prosthesis> (more) ✓ ✗
"Trondheim"@eng ✓ ✗
<Aarhus> (more) ✓ ✗
"Trønderne"@eng (more) ✓ ✗
<Kjersti_Buaas> (more) ✓ ✗
<wikicat_Viking_Age_populated_places> (more) ✓ ✗
<Rune_Bratseth> (more) ✓ ✗
"Nidaros" (more) ✓ ✗
"Trondhjemmer"@eng (more) ✓ ✗
<Lade_Line> (more) ✓ ✗
"58.14611111111111"^^<degrees> ✓ ✗
<http://dbpedia.org/resource/Trondheim> ✓ ✗
"7.99472222222222"^^<degrees> ✓ ✗
<Bokmål> (more) ✓ ✗
"176348"^^xsd:integer (more) ✓ ✗
<Norway> (more) ✓ ✗
<http://www.trondheim.kommune.no/> ✓ ✗
<http://en.wikipedia.org/wiki/Trondheim> ✓ ✗
"49986"^^xsd:integer ✓ ✗
"Sør-Trøndelag"@ara (more) ✓ ✗
<wasCreatedOnDate>
<diedIn>
<hasCapital>
<isLeaderOf>
<isLocatedIn>
skos:prefLabel
<linksTo>
<redirectedFrom>
<livesIn>
rdf:type
<wasBornIn>
rdfs:label
<hasCitationTitle>
<owns>
<hasLatitude>
owl:sameAs
<hasLongitude>
<linksTo>
<hasNumberOfPeople>
<isLocatedIn>
<hasWebsite>
<hasWikipediaUrl>
<hasWikipediaAnchorText>
<hasWikipediaArticleLength>
From http://yago-knowledge.org
Give us feedback:
✓ = fact is correct
✗ = fact is false

predicates are also assigned to a class. Start dates are assigned to a predicate with a type that has a "creation" class, such as "StartedOnDate". End dates are assigned to "destruction" type predicates. This makes it possible to deduce a timespan for a given subject and predicate combination.**?**

Spatial data

Yago only contains permanent spatial data for entities on earth. This means that entities like cities, buildings, rivers and mountains are given a spatial dimension. In addition, events, people, groups and artifacts can be given a spatial dimension by relating the entity to a specific place. All spatial facts must have a predicate that fall under the yagoGeoEntity class, and all objects used in a fact with a yagoGeoEntity must have a relation containing both "hasLatitude" and "hasLongitude".

2.2.3 DBPedia

2.2.4 Common tools and technology

2.2.5 Jena

Apache Jena is a Java framework created for linked data stores. This is used to create a store that can be queried, and traversed using SPARQL as a query language.

Chapter 3

Basic retrieval methods

3.1 Indexing

- General indexing, need for index.
- Keyword index

3.1.1 Spatial indexing

- Different types of indexes.
- Uses for index in general and thesis.
- Querying(?).

Most spatial indexing is done using R-trees. R-trees are based on a B-tree, but is optimized for indexing spatial dimensions. R-trees are able to effectively index spatial data by dividing space into smaller and smaller sets, or bounding boxes, and having the leaf nodes of the trees representing the smallest unit of space used in the data set, usually a small area or a point.

3.1.2 Temporal indexing

In the dataset used in this thesis objects do not move, or move rarely. This makes indexing time unnecessary and it can be treated as a fact in the object with a start, and possible end time. This allows time to be retrieved as a fact when traversing the graph the same way as other facts.

3.2 Search

3.2.1 BFS search

The most basic method of finding a match for keywords is using a breadth first search (BFS) **?**. For each keyword in the query the algorithm will find all vertices that contain the keyword. From that set of vertices the BFS search finds the first vertex that can connect all the vertices in the set. The vertex that connects the the rest of the set is the one that best fits the keywords in the search. There is however no guarantee that this set of vertices contains any spatial or temporal data. If this is the case, the search will continue until the a vertex containing spatial and temporal data is found. The first set of vertices containing all the keywords, a place, and time will be the best result using this search. The BFS search can however contain multiple times, or multiple places. To determine what time or what place best fits the query, a separate ranking algorithm can be used. BFS is also a time consuming algorithm and is used as a proof of concept and baseline in this thesis.

Algorithm 1 BFS

```
1: procedure BFS nodes = getRootNodesFromPlace()
2: maxDepth = 3
3: while nodes is not empty:
4: currentNode = node[0]
5: if node.depth ¿ maxDepth: break
6: if node contains query word:
7: add node to list of hits
8: if all query words are found:
9: set maxDepth to current node depth
10: add neighbors of node to nodes list
11:
```

- Traversing the main graph

The first part of the search is to find a set of root nodes for sub-graphs. This is done by collecting all neighbors from the place queried. The neighbors can be connected by any predicate in the first developed algorithm, but as explain in **??** this selecting specific predicates can greatly increase speed by limiting the amount of nodes traversed. The BFS algorithm described above is used for each of the possible roots of subgraphs, stopping when all keywords are matched, or the depth of the graph exceeds the three or exceeds the currently shallowest subgraph that has matched all keywords. The reason for limiting the depth of subgraphs is to ensure at least a partial hit within a reasonable time. Limiting the depth of subgraphs to the current shallowest subgraph is done because no a deeper graph cannot be a more accurate hit.

- Tokens

- Objects

When traversing the graph and finding a match, a node object is created. This object is used to keep track of the pseudo hierarchy in the graph. All objects contain information on the depth of the node, matched query terms, and relation to parent and children, if any. In addition root objects contains a list of all query terms hit in the sub graph. If a child node is found within multiple subgraphs, a new object will be created representing the node for each subgraph. This creates some objects that are nearly identical, but with different relations and possible different depth.

Algorithm 2 minimum spanning graph

```
1: procedure MINIMUM Algo go here
```

- Start by finding all trees with the same minimum depth for each node (Roots for different trees) related to the place in the query.
After traversing the main graph we have found all subgraphs containing at least one query term. These subgraphs contain many nodes which hit the same terms. Before ranking the subgraphs the minimum spanning graph needs to be found. The minimum spanning tree is found using a greedy algorithm that iterates through all nodes in the subgraphs, then keeping the nodes containing the most terms, and lowest depth. The minimum tree will contain as many terms as possible, a term will only be found in one node, and the graph will be as shallow as possible.

- Find the minimum spanning tree for each of the root nodes containing the maximum amount of the query words.
- Rank based on 1. Query words hit 2. Nodes in the tree 3. depth of the tree
The final piece is to rank the minimum subgraphs. This is done using the nr. 2 formula found in **?**.

Chapter 4

Optimized methods

- Alternative search methods/ algorithms.

4.0.1 Pruning

When using the BFS search method, all possible spatial vertices close to the queried place will be explored. This is expensive and many of the vertices will be irrelevant. Pruning the potential place vertices will reduce the amount of subgraphs traversed, and will in turn reduce the overall time used to find results for the query.

- predicate pruning.

When traversing the graph a lot of unnecessary predicates are followed, resulting in many extra nodes added to the search. Specifying a set of predicates that contain the relevant information can greatly increase the speed of the algorithm, and also keep the memory requirements a lot lower. When selecting predicates for traversal the information expected from the search should be the top priority. Because of this, all predicates that may contain spatial or temporal data should be kept.

When using the entirety of the Yago data set, most nodes are highly connected. Many of the links in the graph are from predicates such as “linksTo” or “redirectedFrom”. These predicates creates a highly connected graph, and ensures a hit within a few nodes of the start. The same predicates will also often add the same nodes multiple times, create circular graphs, and take up unnecessary CPU power and memory.

When pruning predicates there are two methods that are possible to implement. The first will remove the predicates that contain little or no new information, such as “linksTo” or “redirectedFrom” mentioned above. This will still keep the graph connected, and keeps the predicates containing more useful information.

An other method of pruning is to create a list of predicates to be followed. This can drastically reduce the connections in the graph, but the results will only contain information relevant to the query. When preselecting predicates there is a much greater chance of not finding a match for a query. In addition a lot of metadata could be lost, if the metadata predicates are not added to the list of predicate to be explored.

- unqualified place pruning.
For any spatial search, a lot of places will be found. The graph also contains information on the relationship between places. To find the best possible matches for a spatial query, only places of a higher resolution should be chosen. This means that places of similar or smaller expanse should be allows to be queried, e.g. a query of Boston can return the entirety of Boston, close to Boston, or within Boston. Massachusetts has multiple predicates linking it to Boston, but should not be queried because the information returned would be less detailed than what is queried.

- Bounding.
When selecting places or times for a query, the boundaries should be set so that they encompass the entirety of the queried time and or place.

Chapter 5

Methodology

5.1 Dataset

When selecting a data set an important feature was accurate spatial and temporal data. For this Yago was selected. Yago allows for download of subsets of data, and contains all the necessary data. Yagos high accuracy was also a benefit of this data set. From Yago, the data selected was the entire Yago taxonomy, the entire Yago core, and from Geonames the sets GeonamesOnlyData, GeonamesClassIds, GeonamesGlosses, GeonamesTypes, and GeonamesClasses were selected. These sets makes it possible to build a complete graph of the entities in Yago, but keeps the disk usage as low as possible.

Yago taxonomy is used to create a structure ...
- Using parts of Yago
- Short description of each of the parts
- preprocessing: replace non-unicode chars, replacing space with underscore in URIs, terminate triplet where missing termination, remove double quotes in URIs, remove backslash unless legal escape sequence
- tdbloader for persistent queriable storage
- Structure of graph? Here or in Yago description?

5.2 Queries

When selecting the keywords used in queries, a semi random selection method was used at first. This method created three pools of words, rare words (less than 1000 occurrences) uncommon words (less than 100 000 occurrences) and common words (more than 100 000 occurrences). The three pools where created by stemming all words, then counting occurrences. From each pool of words, a set of 100 where randomly selected, and from those 100, 10 where selected manually to ensure a range of occurrences, and to ensure no stop words, misspelled words or other errors were in the final set of keywords.

A second set of words were also chosen to ensure a larger hit rate. This set was a random selection of 20 words from the top 150 words. From the 20 random words 10 where manually selected, the 150 word number was selected based on Zipf's law **?**.

When selecting places for spatial queries, a set of 20 random places were selected from the YagoGeonamesOnlyData set, and from that set, 5 were chosen manually for the final set. In the final set, three places where added manually, Oslo, London, and New York City were added to ensure variation in placement and node connections.

Generating random queries can create results that will not have any hits.
Find a good method of creating queries.

5.3 Pruning

5.3.1 Predicate pruning

5.4 Evaluation

There are multiple possible methods for evaluating the indexing and search methods. A good method for evaluating the retrieval methods is time. The faster information can be retrieved the better the retrieval method should be, assuming the information retrieved is correct. For the purpose of this theses time complexity will be the main evaluation method, along with evaluation of how well the retrieved information fits the query.

5.4.1 Time complexity

When evaluating the time complexity of a solution, a simple timer is sufficient to compare. In addition the big O notation of a solution should be described and explained.

BFS

A BFS search has the complexity of O(V+E) meaning that we simply add the nodes and edges. The best case is O(1), meaning the query terms are found on the first node. Worst case is still O(V+E), but a worst case will not find the terms, returning a empty result.

BFS with pruning

Time complexity with pruning is the same as without pruning, but because there will be less nodes the real time taken will be lower.

5.4.2 Space complexity

BFS

BFS with pruning

Space complexity is in theory the same as without, the real world values will however be smaller because of the reduced number of nodes, and reduction in number of duplicates.

5.4.3 Information match

All methods for retrieving information should find the same results. When ranking the results the ranking should also be the same for all methods implemented.

BFS

BFS with pruning

Chapter 6

Results

6.1 BFS search

BFS search without any pruning and search depth of 3 were unable to finish due to memory constraints and taking too long. Search with a depth of 2 may finish, depending on on the connectedness of the nodes discovered, and the amount of root nodes found in the search.

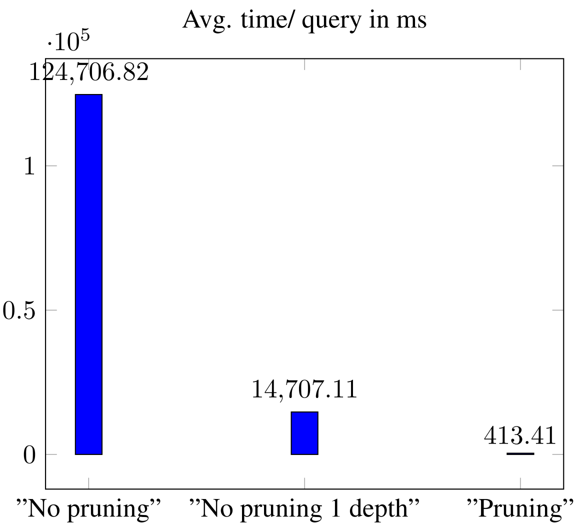
6.1.1 Time

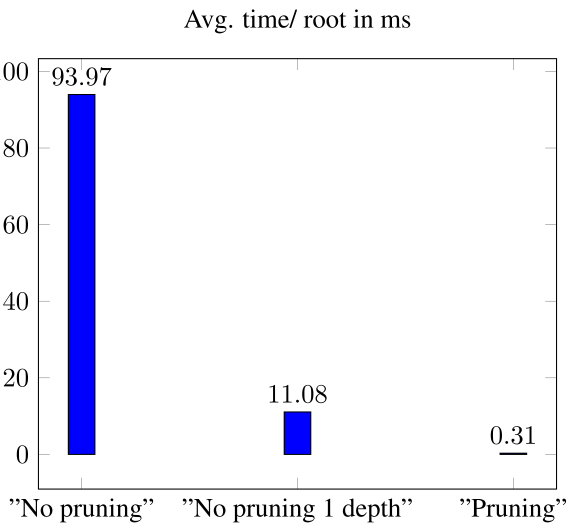
When running the query, two sets are displayed, one with mostly random input data, and one with more closely selected data.

Because most of the randomly selected places had few possible root nodes, the table contains only the manually selected.

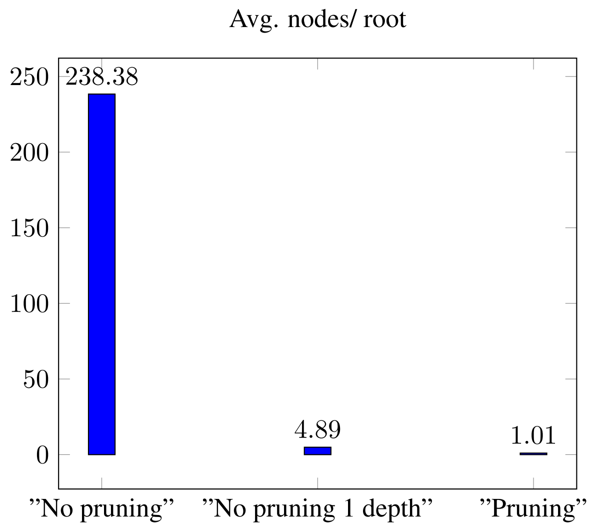
- time/query
- time/root
- roots found/ query

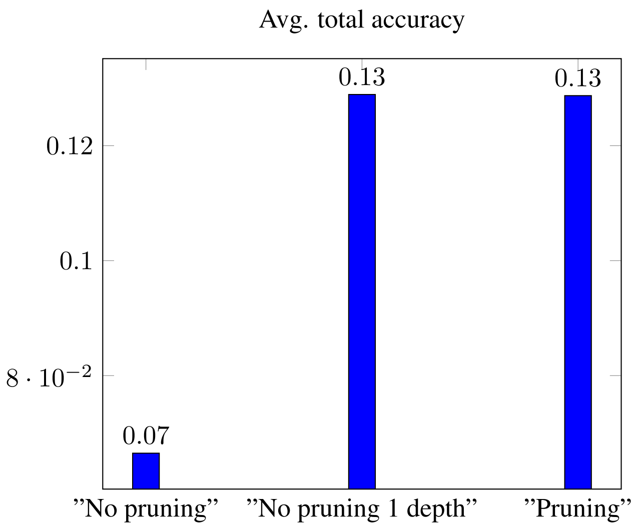
- accuracy/ result
- avg highest accuracy/ query

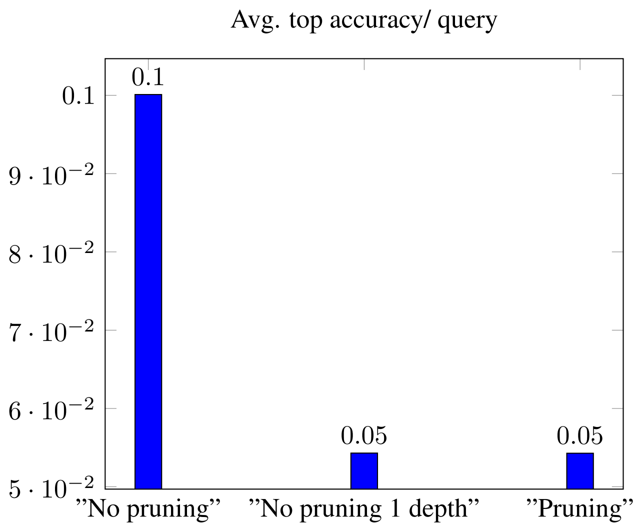

Avg. time/ query in ms
$\cdot 10^5$
124,706.82
1
0.5
0
14,707.11
413.41
"No pruning"
"No pruning 1 depth"
"Pruning"


Avg. time/ root in ms
100
80
60
40
20
0
93.97
11.08
0.31
"No pruning"
"No pruning 1 depth"
"Pruning"

6.1.2 Accuracy






Avg. top accuracy/ query
0.1
0.05
0.05
0.1
$9 \cdot 10^{-2}$
$8 \cdot 10^{-2}$
$7 \cdot 10^{-2}$
$6 \cdot 10^{-2}$
$5 \cdot 10^{-2}$
”No pruning”
”No pruning 1 depth”
”Pruning”

Chapter 7

Discussion

- limitations
- Effectivness of pruning
- choice of query terms
- Root nodes

Chapter 8

Conclusion

Appendix

Write your appendix here...