Conceptualization with Incremental Bron-Kerbosch Algorithm in Big Data Architecture

László Kovács[1], Gábor Szabó[2]

[1]University of Miskolc, Institute of Information Technology, H-3515 Miskolc-Egyetemváros, Hungary, e-mail: kovacs@iit.uni-miskolc.hu

[2]University of Miskolc, Hatvany József Doctoral School of Information Sciences, H-3515 Miskolc-Egyetemváros, Hungary, e-mail: szabo84@iit.uni-miskolc.hu

Abstract: The paper introduces a novel conceptualization algorithm optimized for a distributed, Big Data environment. The proposed method uses a concept generation module based on clique detection in the context graph. The presented work proposes a novel incremental version of the Bron-Kerbosch maximal clique detection method. The efficiency of the method is evaluated with random context tests. The presented incremental model is even comparable with the usual batch methods. The analysis of the clique detection algorithm in MapReduce architecture provides efficiency comparison for large scale contexts.

Keywords: ontologization; clique detection; incremental clique generation; mapreduce architecture

1 Introduction

One of the big challenges of current information technology is the efficient information management and knowledge engineering in Big Data environment. With the spread of new technologies like the Internet of Things, the amount of gathered data steadily increases and new data repository techniques are needed to provide an efficient data management. Another trend to be witnessed is the increased demand on intelligent smart applications. The adaption of knowledge engineering methods on Big Data collection is a real challenge for the IT community. The term 'ontology' in information science is defined as "a formal, explicit specification of a shared conceptualization" [10]. The term conceptualization refers to determination of the concept classes and concept relationships at an abstract model level for the phenomenon in the domain world. The ontology models cover not only concepts, but they also include the corresponding constraints on their usage, too. The ontologies emphasize aspects, such as, inter-agent communication and interoperability [25]. From the viewpoint

of engineering, the term 'concept' is used as an identifier or a descriptor for a cluster of objects. In this sense, the concept describes besides the naming also the properties of the cluster. In the history of knowledge engineering, a great variety of data structures was developed to represent the meaning of concepts. Nowadays, these models exist parallel and are used for different purposes.

According to [19], ontology models can be classified into very different model categories, based on the purpose, specificity and expressiveness. Application ontology is used to control some computational applications and has a methodological emphasis on fidelity. Reference ontology has a theoretical focus on representation and it is used primarily to reduce terminology ambiguity among members of a community. Based on the abstraction level, generic (upper level) and core and domain ontologies can be distinguished. According to [22], the main representation levels of ontologies are *f*

- Taxonomy: Objects are hierarchically classified, e.g. A is child of B.
- Thesaurus: Objects are related (e.g. A is a B; A is related with B). *f*
- Logic-mathematical representation: Object relations are presented in formal notations (e.g. synonym(a, b):=synonym(b, a);).

Analogously to a database, wherein structure and data form the whole, an ontology consists of rules and concepts. Languages for the description of ontologies are RDF-S, DAML+OIL, F-Logic, OWL, WSML or XTM. Using rule-based representations in deductive databases ensures further facts can be deduced from stored relations.

Current ontology languages, like OWL, provide efficient tools to perform complex operations on the ontology database, like consistency verification, rule induction or reasoning process. The main application area of ontology frameworks is the area of knowledge engineering [11], where the ontology engine is integrated into internal module of expert systems. The efficiency of the ontology engines is based primarily on the correctness, completeness and integrity of the ontology database. From this point of view, the key element in ontology management is application of efficient and proper ontology database construction methods.

There are many difficulties in ontology construction, for instance, huge amounts of information to be collected, huge diversity of information sources and inconsistency within the different information sources. The ontology database construction usually contains the following steps [23, 42]:

1. Ontology scope
2. Ontology capture
3. Ontology encoding
4. Ontology integration

5. Ontology evaluation
6. Ontology documentation.

The current ontology construction methods are mainly based on automated ontology construction methods. In the automated ontology construction methods, the information is usually extracted from documents in natural language. This step requires a complex knowledge extraction engine including among others a natural language processing (NLP) module and a concept identification module.

The main goal of the paper is to introduce a novel conceptualization algorithm optimized for a distributed, Big Data environment. The next section provides a survey on the development of the text to ontology methods. The main goal of text to ontology module is to assign the best matching concept cluster to the new words found in the source text. The third section introduces some approaches for concept assignment. The most sophisticated and most complex one is based on clique detection mechanism in the context graph. The paper presents a novel incremental version of Bron-Kerbosch maximal clique detection method. The efficiency of the method is evaluated with random context test. The fourth section contains the analysis of the distributed Big Data architecture. This architecture provides an efficient implementation framework to process a large amount of heterogeneous text document sources. The proposed architecture and map-reduce processing models are implemented in a test environment where the efficiency of the prototype system could be evaluated.

2 Process of Word to Concept Mapping

The Word Sense Disambiguation (WSD) [18] is a method to select the appropriate meaning for a given context. The Word Category Map method [13] clusters the words based on their semantic similarity. The words having similar contexts belong to the same cluster or they appear close to each other on the map. The context of a word can be constructed in many different ways. In the simplest case, the context is equal to the set of neighboring words within the sentences [21]. In this model, the context is converted into a vector representation calculating the average neighborhood vector. The main benefit of this method is the cost efficiency and the simplicity. On the other hand, this model cannot manage the ambiguity of the words, i.e. a word can carry many different meanings. In this model, the semantic similarity is measured using the standard vector distance methods. Beside the simple vector similarity methods, there are approaches to use more sophisticated clustering methods, like the SOM method [21]. In the other group of approaches, the context is given with a graph instead of a simple vector. In ontology management, the thesaurus graphs are used to denote the higher level semantics.

The WSD ontologization process can work in one the following modes [4]: dictionary based, supervised, semi-supervised and unsupervised. In the first mode, a dictionary containing the definitions of the different concepts is used as background knowledge. The similarity of two words is measured with the overlap of their dictionary definitions [16]. In the case of supervised mode, a background ontology is referenced to get information on existing semantic clusters. In the ontology database, a word can be assigned to several meanings. In order to distinguish the different concepts related to a word, a specific component is introduced which represents the meaning. The most widely used implementation of this component is the synset component defined in a wordnet ontology database.

Wordnets are usually based on architecture presented first in [8] and they organize semantic-lexical knowledge into a graph knowledge base. Nowadays, Wordnet knowledge bases are available for many different languages.

The synset can be considered as the set of words carrying a common meaning. The elements of a synset are synonyms and a word can be an element of different synsets. Using the wordnet knowledge base, the ontologization process performs a pattern matching task, where the matching method locates the synset most similar to the given word. Within this method, the key parameters refer to the similarity measure applied to compare a word and a synset. In most approaches, the similarity measure between the word and the synset is aggregated from the similarity values between the target word and the words in the synset.

In the literature there are different approaches to measure similarity between a word and a synset. The extension of the related proportion approach [20] yields the neighborhood similarity measured with:

$$d(w,C) = \sum_r \alpha_r \sum_{v \in N_{wr}} \frac{n_{Crw}}{1+\log(|C|)}.$$

Where

w: the target word

C: the synset

v: word in the knowledge base

r: relation in the semantic graph

n_{Crv} : the number of edges from elements of C to v along with an edge type r

N_{wr} : the set of words connected to w along with an edge of type r

α_r: the weight factor of the relationship r.

The average cosine method (see for example [3]) calculates distance as the average cosine value between the description vectors:

$$d(w,C) = \frac{\sum_{c \in C} \cos(\overline{N}_c, \overline{N}_w)}{|C|}.$$

Where

$\overline{N}_v$: the adjacency vector for word v (describing the words connected to v).

Having an ontology graph, an important step in text to semantic conversion is to select the proper meaning of the word.

The supervised approach requires the development of a background knowledge base in the form of a dictionary or of an annotated text corpus. The quality of the WSD process depends on the quality of the background knowledge base. The main problem of this approach is the high cost in the generation of a comprehensive and valid knowledge base. The unsupervised methods provide automatic approaches for construction of the knowledge base.

The first important result on the field of unsupervised WSD was the semi-supervised proposal in [27]. The proposed method is based on two main properties of human languages:

- Nearby words provide strong and consistent clues as to the sense of a target word
- The sense of a target word is highly consistent within any given document.

The algorithm first generates a small set of seed representative of the different senses of a word. This step is a supervised phase to assign the semantic label to some of the word occurrences. In the next step, a supervised classification algorithm is used to learn the differences between the contexts of the different word senses. In the last step, all word occurrences are classified with the generated classifier to one of the sense labels.

The unsupervised method of [4] requires only the WordNet sense inventory and unannotated text to determine the meaning category of a target word. The algorithm includes the following processing steps. First, a pool of application context is collected from the web. In the next step, the sentences containing the target word are parsed with the help of a dependency parser in a parser tree. The third phase is used to merge these trees into a dependency graph. In the last step a graph matching algorithm is applied to find the appropriate meaning. This phase is based on the idea that if a word is semantically coherent with its context, then at least one sense of this word is semantically coherent with its context.

In the case of fully unsupervised methods, no background knowledge base is available, only an unannotated source text can be used. In this case, only the unsupervised clustering method can be used to create synsets for the words in the document pools. Clustering methods are used to partition objects into groups where the objects within the same group are similar to each other while objects from different groups are dissimilar. In general, the mentioned clustering methods do not usually meet this basic constraint of clustering, as they allow building of large clusters containing object pairs with low similarity. One way to provide

better clustering is the application of Quality Threshold Clustering [12]. The QTC method ensures that the distance between any two elements within a cluster should be below a given threshold.

All of the mentioned clustering algorithms generate non-overlapping clusters. There are some application areas where the constraint that every element must belong to only one cluster that is not met. In the semantic clustering, for example, a word may belong to different clusters, i.e. to different meanings at the same time. Clustering in social networks or in distributed networking are other application areas where non-overlapping clustering yields in significant loss of information. It is shown in [15] that overlapping improves the approximation algorithms significantly for minimizing graph conductance.

Overlapping clustering can be considered as a generalization of the standard clustering methods. The first important approach for overlapping clustering is given in [14]. In this approach, the input structure is a graph where the edges denote object pairs having a similarity value above a given threshold. A cluster is considered as a maximal complete subgraph, i.e. all of the members are connected to each other. The proposed cluster detection method is based on the k-ultarmetrics.

Another group of approaches is based on the extension of the classical clustering methods. In [5], the k-means method is adjusted for overlapping clustering. Initially, random cluster centers are specified. In the next phase, the elements are assigned to a subset of clusters. For a given element *x*, the set of container clusters is generated in the following way. First, sort the cluster centers based on the increasing distances from the given element. Calculate the set *A* of the first *k container c*lusters for which:

$$\sum_x \| x - \varphi(x) \|^2 \to \min$$

is met, where

$$\varphi(x) = \frac{\sum_{c \in A} m_c}{|A|},$$

m_c : the cluster prototype for cluster c.

In the third step, the new positions of the cluster centers are calculated. The performed tests show that this method is a good alternative of the more sophisticated complex techniques.

The third important category of overlapped clustering methods is based on the mixture model. In general, the key input source for a mixture model is the observation matrix *X*. The unknown parameter set is denoted with Θ. The basic assumption is that each data X_i point belongs to the following probability density [1]:

$$p(X_i \mid \Theta) = \sum_{h=1}^{k} \beta_h p_h(X_i \mid \Theta_h)$$

where

k: the number of mixture components

Θ_h: the parameters of the h-th mixture component

β_h: the probability of the h-th mixture component.

The parameter values are usually calculated with the EM method where the goal is to maximize likelihood of the given observation set:

$$\prod_{i=1}^{n} p(X_i \mid \Theta).$$

3 Incremental Bron-Kerbosch Algorithm

Our investigation focuses on the clustering with the clique detection approach originated from the work of [14]. The observation graph can be considered as a similarity graph. The nodes are objects of the problem domain and there is an undirected edge between two vertices (v_i,v_j) if and only if the $d(v_i,v_j) < \varepsilon$ for a given threshold ε. A clique, i.e. maximal complete subgraph corresponds to a cluster. A cluster symbolizes a concept in the target ontology knowledge base. The goal of the investigated algorithm is to detect all maximal cliques in the observation graph.

The Bron-Kerbosch algorithm is one of the most widely known and most efficient algorithms for maximal clique detection. The algorithm was presented first in [2]. The Bron–Kerbosch algorithm uses a recursive backtracking method to search for all maximal cliques in a given graph $G(V,E)$. The pseudocode of the algorithm:

BronKerbosch(R, P, X):

if $|P| = |X| = 0$

  report R as a maximal clique

$\forall v \in P$:

  BronKerbosch($R \cup \{v\}$, $P \cap N(v)$, $X \cap N(v)$*)*

  $P = P \setminus \{v\}$

  $X = X \cup \{v\}$

In the pseudocode, the following notations are used:

P: the subset of V that can have some common elements with the investigated clique

R: the subset of V that share all of its elements with the investigated clique

X: the subset of V that is disjoint with the investigated clique

N(v): the set of vertices connected to v.

Initially, the set P contains all of vertices of G and both R and X are empty sets. In every iteration call, an element of P is processed and the sets P, R, X are refined based on the current neighborhood set. For the recursive call, the set of possible clique members are restricted to the elements in the neighborhood set. Similarly, the set of possible excluded elements are reduced to a subset of the neighborhood set. In the main loop, the elements of P are tested in a predefined order. After testing an element v from P, it will be removed from the set of candidate vertices.

The basic version of the Bron-Kerbosch algorithm uses a large number of recursive calls, resulting in an execution complexity of worst-case running time $O(3^{n/3})$ [7] [24]. The updated method uses a specific pivoting strategy to cut computational branches. The pivot element is selected as the element with highest number of neighbors.

TomitaBronKerbosch(R, P, X):

if $|P| = |X| = 0$

report R as a maximal clique

let $u \in P \cup X: |N(u) \cap P| \rightarrow max$

$\forall v \in P \backslash N(u)$

TomitaBronKerbosch($R \cup \{v\}, P \cap N(v), X \cap N(v)$)

$P = P \setminus \{v\}$

$X = X \cup \{v\}$

The presented methods generate the output structure for a fixed given input context. This approach is used for static investigation when there is no change in the input context. The incremental construction method is used for such problems where the initial context is extended incrementally. The incremental methods are used for applications where context changes from time to time. The modeling of cognitive learning processes is a good example of such dynamic problem domains.

Our investigation focuses on the incremental clique generation method. For the analysis of the proposed method, the following initial notations are used:

$G(V,E)$: context graph

V : set of vertices in G

E : set of edges in G

N : number of vertices in V.

It is assumed that there exists a total ordering of the vertices, .i.e.

$$V = \{v_i \mid i \in [1..N]\}.$$

Taking only the first n elements of V, we get a reduced context graph $G_n(V_n,E_n)$, where

$$V_n = \{v_i \mid v_i \in V, i \le n\}$$

$$E_n = \{(v_i, v_j) \mid (v_i, v_j) \in E, v_i, v_j \in V_n\}.$$

Let $C(G_n)$ denote the set of maximal cliques related to G_n. The cliques in $C(G_n)$ can be separated into two disjoint groups, depending on the property whether they are new cliques or old cliques related to $C(G_{n-1})$.

$$C(G_n) = C^u(G_n) \cup C^o(G_n)$$

$$C^o(G_n) = \{c \mid c \in C(G_n) \cap C(G_{n-1})\}$$

$$C^u(G_n) = C(G_n) \setminus C^o(G_n)$$

It can be easily verified that for every $c \in C^u(G_n)$,

$$v_n \in c.$$

Regarding $C^u(G_n)$, the following proposition can be used in the incremental clique generation method.

Proposition 1:

The set of new cliques of G_n is equal to the clique set generated for the inclusive neighborhood of the new vertex.

$$C^u(G_n) = C(S_n)$$

where

$S_n(x) = (V'_n, E'_n)$

$V'_n = \{v_i \mid v_i \in V, i < n, (v_i, v_n) \in E\} \cup \{v_n\}$

$E'_n = \{(x,y) \mid (x,y) \in E, x \in V'_n, y \in V'_n\}.$

The inclusive neighborhood set S_n is defined as the neighborhood of v_n extended with the element v_n.

Proof.

Let us take a new clique in G_n:

$$c \in C^u(G_n).$$

It can be seen that

$$v_n \in c\,,$$

i.e. v_n is connected to all other elements of c. Thus, all elements of c are in the inclusive neighborhood of v_n and all the edges in G_n are also edges in S_n. This means that c is a complete subgraph of S_n, too. As the assumption that c is maximal in G_n and not maximal in S_n, imply a contradiction, the clique c is a maximal complete subgraph in S_n. Thus

$$C^u(G_n) \subseteq C(S_n)\,.$$

On the other hand, taking a c clique from $C(S_n)$, c will be complete in G_n, because all the edges in S_n are also edges in G_n. If c is not maximal in G_n then there exists a vertex v_i such that

$$v_i \notin c, v_i \in V_n, (v_i, v_n) \in E_n\,.$$

Thus, $i < n$ and

$$v_n \in V^n\,.$$

This means that c is not maximal in S^n, and this is a contradiction, i.e.

$$C^u(G_n) \supseteq C(S_n)\,.$$

Based on the considerations shown in the proof, we get:

$$C^u(G_n) = C(S_n)\,.$$ □

Proposition 1 can be used to generate the new cliques in an effective way, but not only the insertion of the new cliques is the required update operation on $C(G_{n-1})$. Namely, some of the cliques in $C(G_{n-1})$ may become invalid as they are not maximal anymore. For example, in Figure 1, the clique set of G_4 is {(1,3),(2,3),(3,4)}. After adding v_5, S_5 is equal to ({2,3,4,5}, {(2,3), (2,5), (3,4), (3,5), (4,5)}). The clique set for S_5 is {(2,3,5),(3,4,5)}. The resulting clique set for G_5 is {(1,3), (2,3,5), (3,4,5)}. Thus, the clique (2,3) is covered by (2,3,5) and (3,4) is covered by (3,4,5).

Thus, the update of $C(G_{n-1})$ after generation of $C(S_n)$ includes the removal of some existing cliques and the insertion of some new cliques. As the covered clique is the same as the new clique except the v_n vertex, it is worth to reduce S^n to the elements of the neighborhood and to exclude v_n. This new graph is denoted with S_{n-}.The incremental clique generation algorithm can be summarized in the following listing:

IncrementalBronKerbosch(G_{n-1}, $C(G_{n-1})$, S_n):

$C(S_{n-})$ = *BronKerbosch(S_{n-})*

$C(G_n) = C(G_{n-1})$

$\forall c \in C(S_{n-})$:

$C(G_n) = C(G_n) \setminus \{c\}$

$c' = c \cup \{v_n\}$

$C(G_n) = C(G_n) \cup \{c'\}$

return $C(G_n)$

The lookup operation in the set $C(G_n)$ has a central role in optimization of the algorithm. Instead of a naive sequential lookup operation having a cost $O(D)$ where D denotes the number of elements in the set, a prefix tree structure is used. In the prefix tree structure, the clique sets are represented with an ordered list of vertex identifiers. The tree stores these ordered lists where the lists having the same prefix part share the same prefix segments in the tree.

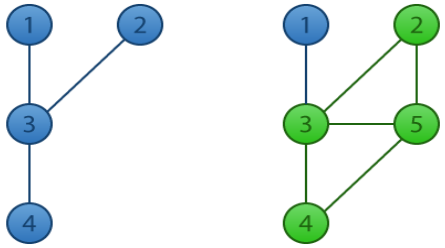

Figure 1
Extension of the similarity graph

The results of the performed direct tests show that the proposed method provides an efficient solution for incremental clique detection tasks. In Figure 2, the comparison of the naive incremental method and the proposed incremental method is presented. The time shows the execution time in logarithmic scale. As the result shows, the proposed algorithm is about 100 times faster than the naive method. In the naive method, for every new incoming object, the whole clique set is recalculated from scratch. In Figure 2, the data set NI denotes the naive method and symbol I is for the proposed incremental method.

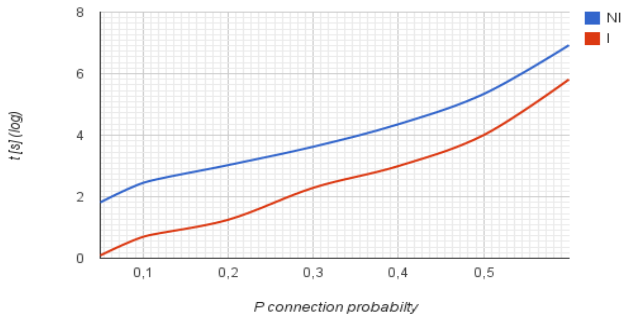

Figure 2
The comparison of the naive and the proposed methods

The method outperforms not only the other incremental approaches but it has better cost characteristics than the standard batch method. For larger input graphs, the proposed method is significantly faster than the basic Bron-Kerbosch method (see Table 1, Table 2). The runtime is displayed in seconds.

Table 1
Comparison of the basic Bron-Kerbosch and the proposed methods for edge probability 0.5

Vertices	Cliques	BK Runtime	Incremental Runtime
100	5407	0.41	0.32
150	23440	2.47	1.91
200	76838	10.42	8.36
250	214457	38.1	28.6
300	496754	103.5	82.4

Table 2
Comparison of the basic Bron-Kerbosch and the proposed methods for edge probability 0.3

Vertices	Cliques	BK Runtime	Incremental Runtime
100	749	0.04	0.04
150	2277	0.12	0.08
200	5126	0.18	0.12
250	9566	0.41	0.24
300	16858	0.80	0.51

The efficiency of the proposed incremental algorithm is based on the divide and conquer paradigm. The algorithm generates the clique set only for the neighborhood graph during each iteration. The size of the neighborhood graph depends on the edge probability in the input graph. If the cost function of the basic Bron-Kerbosch algorithm is denoted with

$$O(C_b(n))$$

then the complexity of the proposed incremental method belongs to

$$O(n(C_b(n') + D_b(n')))$$

where

n' : the average size of the neighborhood graph,

$D_b(n)$: the number of cliques in a graph containing n vertices.

The first term in the formula corresponds to the generation of the cliques for the neighborhood graph while the second term relates to the elimination of the covered cliques from the result set. The presented formula provides an acceptable approximation of the measured cost values but further investigation is needed to work out a more accurate cost function involving the additional parameters, too.

4 Implementation in the MapReduce Architecture

Although the speed of computers is increasing rapidly, the fact that using several machines in an interconnected system can be more effective in certain cases was recognized many years ago. There are many solutions in the literature that can be used in Big Data analysis. MPP (Massively Parallel Processing) systems [29] store data after splitting it according to its features, for example we could store regional data split by the country or state. In-memory database systems [30] are very similar, except that they store data in memory, thus speeding up the retrieval of the records. This area is the topic of active research by Big Database vendors such as Oracle. BSP (Bulk Synchronous Parallel) systems [31] use multiple transformation processes that run in parallel on different nodes. Each process gets data from a master node and then sends the result back. After this barrier-like synchronization the next iteration can be executed.

The big breakthrough came when Google published their scientific article involving a new architecture for processing huge amounts of data. This architecture is called MapReduce. The article called *MapReduce: Simplified Data Processing on Large Clusters* [28] created a whole new concept that became the basis for the most popular framework of Big Data analysis to date. The concept itself is very simple. We all know that parallel tasks are most effective when we can run in parallel for a long time without any synchronization barriers. This means that if we can divide our algorithms in such a manner, the result can outperform the single-threaded version. Conversely, if we need to communicate between the threads, these synchronization points can slow down the algorithm.

MapReduce got its name from its two most important phases: map and reduce. The map phase produces key-value pairs which become the input of the reducer phase that yields the final results. Although Google's implementation is not open source, there are multiple open source implementations among which the most popular one is Apache Hadoop. This framework is actively developed, constantly improving and has many technologies built on top of it like Apache Pig (SQL-like interface), Apache Hive (data warehouse infrastructure), Apache HBase (distributed NoSQL database), etc. that specializes the Hadoop platform to more specific problems.

The base concept of MapReduce and Apache Hadoop [32] is to split the incoming data to multiple nodes and process them locally. This way – after the initial data copies – the mapper tasks can work on their local data instead of retrieving them during processing. To make sure that the data is not lost on node failures, every data chunk is stored on multiple nodes, but in case of normal work without failures every node works on its local hard disk. These low-level interactions are abstracted by the Hadoop Distributed File System (HDFS), inspired by the Google File System (GFS) [9]. This is the most important component of Hadoop as it provides a distributed file system for the MapReduce tasks. The input data must be copied to this file system – which makes sure that every data chunk is persisted on multiple nodes –, then the mappers get data from this file system and the reducers write the results back to HDFS. Figure 3 displays an overall view of a very simplified MapReduce task. On the left side of the image we can see the input records that come from HDFS. Next to them there are some mapper tasks that receive one input record at a time and produce key-value pairs from them. For simplicity the figure only has two types of keys (green and blue). The reducers receive objects with common keys and create the final output of the application that is persisted back to HDFS.

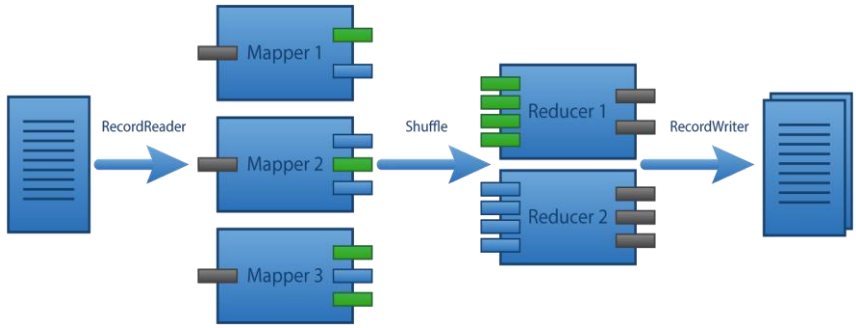

Figure 3
The MapReduce architecture

There are many scientific research areas that are directly or indirectly related to MapReduce and Apache Hadoop. During literature research we can find scientific articles from the area of agriculture [33] through telecommunication [34] to AI-based recommendation systems [35] that use Hadoop as the application platform. It is not surprising that graph algorithms can be ported to this parallel architecture and thus speeds up different search methods.

The main research areas of Hadoop based graph processing are the semantic web related problems and processing of social network data. This work [36] tries to solve the problem of maintaining and querying huge RDF graphs by using Hadoop, and provides an interface on top of it to answer SPARQL queries. (SPARQL [37] is the W3C standard language for querying ontologies based on

RDF triplets.) This is a classic ontology related problem that can be integrated with MapReduce to make the processing distributed.

There are also proposals [38] on a Hadoop based solution to the problem of processing huge graphs in parallel. Unlike most solutions that try to separate the nodes in the form of classic MapReduce jobs, this article presents a system that has a highly flexible self-defined message passing interface that makes it easier to port graph algorithms on top of the MapReduce platform. GPS (Graph Processing System) [39] is a complete open-source system similar to Google's Pregel [40], extending it to provide an interface for processing huge graphs in parallel with global communication among the nodes. As we can see, these two research results combine Big Data analysis with graph processing, thus connecting to our research area and results presented in this article.

The Spider system [41], tries to solve the problem of processing data on the semantic web. The system has two modules: one that loads the graph and one that can query the previously loaded graph leveraging the Hadoop framework.

Considering the implementation of the clique detection algorithm, two different approaches were tested. In the first approach, the task unit is the processing of the neighborhood graph of a node. Here, for every node a separate task is generated. The mapper calculates the clique set in the neighborhood. The drawback of this approach is that it generates the same clique several times. The main benefit of the method is that every node can work separately with a smaller amount of local data.

In the second approach, all of the nodes work with the global data set and, share the global graph. The work is separated in such way that every clique is generated only once. Here, the merging process can be executed with low cost. Alternately, every node requires the whole dataset.

Regarding the first approach (A), the full graph is divided by its nodes. After the GraphRecordReader reads the input file from HDFS and reconstructs the graph object, it takes each node from the graph and all of its neighbors and yields a new subgraph from these nodes. This way if we find a maximal clique in the subgraph, it will be a maximal clique of the full graph as well. The Hadoop framework distributes these subgraphs based on the system configuration and it makes sure that every mapper node gets approximately the same number of subgraphs. Each mapper process gets one subgraph at a time with a null key, and executes one of the Bron-Kerbosch variations on it. After getting the maximal cliques, it calculates the hash code of the resulting cliques and yields key-value pairs consisting of the hash code as the key and the clique as the value. Since the mapper node might produce the same clique multiple times, a combiner is used on each node that drops every repeated result to optimize speed so that these don’t have to be sent over the network to the reducer nodes.

The reducer then gets one hash code at a time and the list of cliques that have that hash code. Ideally this list only contains one clique, but if multiple mappers produced the same clique, the size of the list can be more than one. Therefore the reducer yields only the first element of the list. The Hadoop framework then writes the resulting cliques from the reducer's output to HDFS.

In the second approach (B), the set of cliques is divided by the smallest node index value within the clique. This method yields in a disjoint partitioning, thus the merge phase can be implemented with a minimal cost. In the implementation of the method a shared HDFS storage is used to store the input similarity graph. The map component performs a Bron-Kerbosch clique detection algorithm where the main loop is restricted to the nodes assigned to this mapper process. The key value in the MapReduce framework is equal to the set of minimum index values assigned to a node for processing. Every mapper node works with the common shared input graph to generate the corresponding clique set.

We tested the two approaches on randomly generated graphs with fixed node counts and edge probabilities to verify their correctness in practice. Although we didn't have a full Hadoop cluster up and running, we simulated such environments with the help of Docker, a lightweight virtualization software. We set up two docker images and initiated one master and related slave containers, both executing map and reduce tasks. In the future we would like to set up a physical Hadoop cluster and verify our assumptions in a real distributed environment.

The cost models of the proposed methods can be approximated with the following formulas:

$$\cos t_A = \frac{N}{L} C(N_n) + N \cdot C(N) + N_n \cdot N$$

$$\cos t_B = \frac{C(N)}{L} + L \cdot N$$

where

N: the number of nodes in the input graph

N_n : the number of nodes in a neighborhood graph

L: the number of mapper nodes

$C(n)$: the number of cliques for a graph having n vertices.

The formula is based on the fact that the cost of clique detection algorithm is a linear function of the number of generated cliques. The experimental function of $C(N)$ is shown in Figure 4, while Figure 5 presents the corresponding execution costs. These formulas and the test results show that method A is suitable for those architectures where the neighborhood graph is relatively small (sparse input graph) and the L value is large. The method B is optimal for the cases where L is small and the graph is relatively dense.

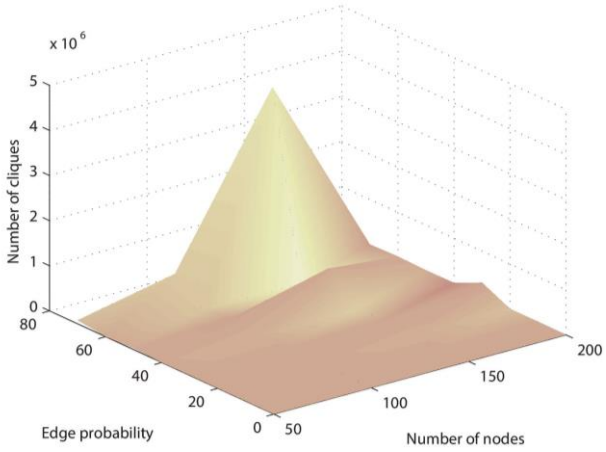

Figure 4
The number of generated cliques

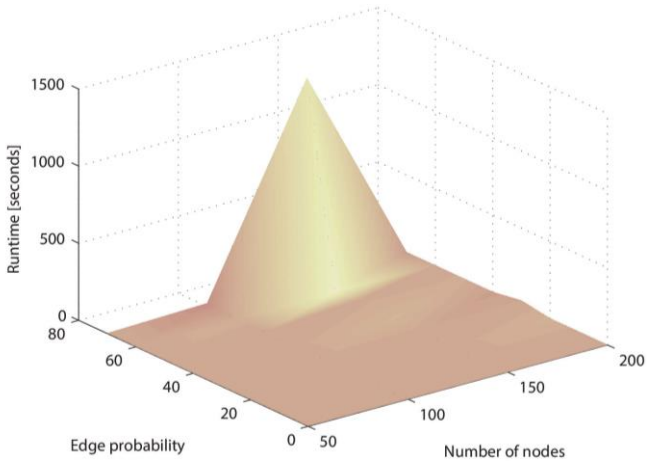

Figure 5
The cost function of the proposed iterative method

Conclusions

One option to generate concepts in an ontology framework, is to build clusters of similar objects. The cliques in a similarity graph can be considered as overlapping quality threshold clusters. The proposed incremental clique detection algorithm provides an efficient clustering method for dynamic contexts changing from time to time. The presented incremental model is comparable even with the usual batch methods. The cost analysis for large contexts shows that the optimal implementation in the MapReduce architecture depends on the basic parameters of the input similarity graph.

References

[1] Banerjee A., Krumpelman C., Basu S., Mooney R.: Model-based Overlapping Clustering, Proc. of Eleventh ACM SIGKDD, pp. 532-537

[2] Bron C., Kerbosch J., "Algorithm 457: Finding All Cliques of an Undirected Graph", Communications of ACM 16 (9): 1973, pp. 575-577

[3] Caraballo S.: Automatic Construction of Hypernym-Label Noun Hierarchy from Text, Proc. of Annual Meeting ACL, 1999, pp. 120-126

[4] Chen P, Ding W, Bowes C, Brown D: A Fully Unsupervised Word Sense Disambiguation Method Using Dependency Knowledge, The 2009 Annual Conference of the North American Chapter of the ACL, pp. 28-36

[5] Cleuziou G.: An Extended Version of the k-means Method for Overlapping Clustering, Pattern Recognition, ICPR 2008, pp. 1-4

[6] Dean J., Ghemawat S.: MapReduce: Simplified Data Processing on Large Clusters, Proc. of OSDI 2004, pp. 10-20

[7] Eppstein D., Löffler M., Strash D.: Listing all Maximal Cliques in Sparse Graphs in Near-Optimal Time, Proc. of ISAAC 2010, pp. 403-414

[8] Fellbaum C.: WordNet: An Electronic Lexical Database (Language, Speech and Communication), MIT Press, 1998

[9] Ghemawat S., Gobioff H., Leung S.: The Google File System. Proc. of the 19th SOSP 2003, pp. 29-43

[10] Gruber T.: A Translation Approach to Portable Ontology Specifications, Knowledge Acquisition, 5(2): 1993, pp. 199-220

[11] Happel H., Seedorf S.: Applications of Ontologies in Software Engineering. Proc. of SWESE 2006, Athens, GA, USA

[12] Heyer L., Ramakrishanan R., Livny M.: BIRCH: An Efficient Data Clustering Method for Very Large Databases, Genome Research, Vol. 9, 1999, pp. 1106-1115

[13] Honkela T: Comparisons of Self-Organized Word Category Maps, In Proceedings of WSOM'97, pp. 298-303

[14] Jardine N., Sibson R.: Mathematical Taxonomy, John Wiley and Sons Publ.

[15] Khandekar R, Kortsarz G., Mirrokni V.: On the Advantage of Overlapping Clusters for Minimizing Conductance. Algorithmica, 69(4), 2014, pp. 844-863

[16] Lesk M.: Automatic Sense Disambiguation using Machine Readable Dictionaries: How to Tell a Pine Cone from an Ice Cream Cone. In Proc. of SIGDOC ’86

[17] Moon, J. W., Moser, L., "On Cliques in Graphs", Israel J. Math. 3:, 1965, pp. 23-28

[18] Navigli R.: Word Sense Disambiguation: A Survey, ACM Computing Surveys, 41(2), pp. 1-69

[19] Oberle D.: Semantic Management of Middleware, Volume I of The Semantic Web and Beyond Springer, New York (2006)

[20] Pennacchiotti M., Pantel P.: Ontologizing Semantic Relations, Proc. of 21st COLING ACL, 2006, pp. 793-800

[21] Ritter H., Kohonen T.: Learning 'Semantotopic Maps' from Context. In Proc. IJCNN- 90-WASH-DC, Int. Conf. on Neural Networks, Vol. I, pp. 23-26

[22] Sieber T., Kovács L.: Mapping XML-Documents into Object-Model, Proc. of CINTI 2005, pp. 343-354

[23] Subhashini R., Akilandeswari J.: A Survey on Ontology Construction Methodologies, International Journal of Enterprise Computing and Business Systems, Vol. 1, Issue 1, 2011

[24] Tomita E., Tanaka A., Takahashi H., The Worst-Case Time Complexity for Generating all Maximal Cliques and Computational Experiments, Theoretical Computer Science 363 (1): 2006, pp. 28-42

[25]. Uschold M., Gruninger M.: Ontologies: Principles, Methods, and Applications. Knowledge Engineering Review 11 (1996) pp. 93-155

[26] Wood D.: On the Maximum Number of Cliques in a Graph, Graphs Combin. 23 (2007) pp. 337-352

[27] Yarowsky D.: Unsupervised Word Sense Disambiguation Rivaling Supervised Methods, Proc. of ACL'95, 1995, pp. 189-196

[28] Dean J., Ghemawat S.: Mapreduce: Simplified Data Processing on Large Clusters. Commun. ACM 2008, 51(1), pp. 107-113

[29] Simon, H. D.: Partitioning of Unstructured Problems for Parallel Processing. Computing Systems in Engineering, 2(2), 1991, pp. 135-148

[30] Berkowitz, B. T., Simhadri, S., Christofferson, P. A., and Mein, G. In-Memory Database System. US Patent 6, 2002, 457,021

[31] Gerbessiotis, A. V. and Valiant, L. G.: Direct Bulk-Synchronous Parallel Algorithms. Journal of Parallel and Distributed Computing, 22(2):1994, pp. 251-267

[32] Wadkar, S., Siddalingaiah, M., and Venner, J.: Pro Apache Hadoop. Apress, 2014

[33] Yang, F., Wu, H.-R., Zhu, H.-J., Zhang, H.-H., and Sun, X.: Massive Agricultural Data Resource Management Platform Based on Hadoop [J]. Computer Engineering, 2011, 12:083

[34] Yang, G., Shu, M., Wang, X., and Xiong, A.: Application Practice of Hadoop Platform for Telecommunication Enterprise. Digital Communication, 4:019, 2014

[35] Wang, C., Zheng, Z., and Yang, Z.: The Research of Recommendation System Based on Hadoop Cloud Platform. In Computer Science & Education (ICCSE), 2014 9th Int. Conference on, pages 193-196. IEEE

[36] Husain, M. F., Doshi, P., Khan, L., and Thuraisingham, B.: Storage and Retrieval of Large RDF Graph Using Hadoop and MapReduce. 2009, pp. 680-686

[37] Quilitz, B. and Leser, U.: Querying Distributed RDF Data Sources with SPARQL. In The Semantic Web: Research and Applications, Vol. 5021 of Lecture Notes in Computer Science, 2008, pp. 524-538

[38] Pan, W.; Li, Z.-H.; Wu, S.; Chen, Q.:Evaluating Large Graph Processing in MapReduce Based on Message Passing. Chinese Journal of Computers., 2011, pp. 1768-1784

[39] Salihoglu, S. and Widom, J.: GPS: A Graph Processing System. In Proceedings of the 25th International Conference on Scientific and Statistical Database Management, SSDBM, 2013, pp. 22:1-22:12

[40] Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010) Pregel: A System for Large-Scale Graph Processing. In Proc. of ACM SIGMOD 2010, pp. 135-146

[41] Choi, H., Son, J., Cho, Y., Sung, M. K., and Chung, Y. D.: Spider: A System for Scalable, Parallel/Distributed Evaluation of Large-Scale RDF Data. In Proceedings of ACM Conference on Information and Knowledge Management, 2009, pp. 2087-2088

[42] Furdik K., Tomasek M., Hreno J.: A WSMO-based Framework Enabling Semantic Interoperability in e-Government Solutions, Acta Polytechnica Hungarica, Vol. 8, No. 2, 2011, pp. 61-79