Optimizing Automated Term Extraction for Terminological Saturation Measurement

Victoria Kosa[1 [0000-0002-7300-8818]], David Chaves-Fraga[2 [0000-0003-3236-2789]], Hennadii Dobrovolskyi[1 [0000-0001-5742-104X]], Egor Fedorenko [1, 3 [0000-0001-6157-8111]], and Vadim Ermolayev [1 [0000-0002-5159-254X]]

[1] Department of Computer Science, Zaporizhzhia National University, Zaporizhzhia, Ukraine
victoriya1402.kosa@gmail.com, gen.dobr@gmail.com, vadim@ermolayev.com

[2] Ontology Engineering Group, Universidad Politécnica de Madrid, Madrid, Spain
dchaves@fi.upm.es

[3] BaDM, Dnipro, Ukraine
egorfedorencko@gmail.com

Abstract. Assessing the completeness of a document collection, within a domain of interest, is a complicated task that requires substantial effort. Even if an automated technique is used, for example, terminology saturation measurement based on automated term extraction, run times grow quite quickly with the size of the input text. In this paper, we address this issue and propose an optimized approach based on partitioning the collection of documents in disjoint constituents and computing the required term candidate ranks (using the c-value method) independently with subsequent merge of the partial bags of extracted terms. It is proven in the paper that such an approach is formally correct – the total c-values can be represented as the sums of the partial c-values. The approach is also validated experimentally and yields encouraging results in terms of the decrease of the necessary run time and straightforward parallelization without any loss in quality.

Keywords: Automated term extraction, terminological saturation, partial c-value, merged-partial c-value, optimization

1 Introduction

Ontology learning from texts is a developing research field that aims to extract domain description theories from text corpora. It is increasingly acknowledged as a plausible alternative to ontology development based on the interviews of domain knowledge stakeholders. One shortcoming of learning an ontology from texts is that the input corpus has to be quite big for being representative for the subject domain. Another shortcoming is that learning ontologies from text is expensive, in terms of taken time, as it involves the use of several algorithms, in a pipeline [1], that are computationally hard.

Automated term extraction (ATE) is an essential step at the beginning of the pipeline for ontology learning [1, 2], that is known to be bulky in terms of the increase of the run time with the growth of the input text corpus. Therefore, finding a way to reduce: (i) either the size of the processed text; or (ii) the time spent for term extraction; or (iii) both is of importance.

In our prior work [2, 3, 4, 5], we developed the ATE-based approach (OntoElect) that helps circumscribe the minimal possible representative part of a documents collection, which forms the corpus for further ontology learning. This technique is based on measuring terminological saturation in the collection of documents, which is computationally quite expensive in the terms of the run time.

In this paper, we present the approach, based on the partitioning of a document collection, which allows substantially reducing ATE run time in the OntoElect processing pipeline.

The remainder of the paper is structured as follows. In Sect. 2, we outline our OntoElect approach to detect terminological saturation in document collections describing a subject domain. In Sect. 3, we review the related work in ATE and argue for the choice of the c-value method as the best appropriate for measuring terminological saturation. In Sect. 4, we explain our motives to optimize the c-value method based on partitioning a document collection and present a formal framework for that. Section 5 reports on the setup and results of our experimental evaluation of the proposed optimization approach. Finally, we draw the conclusions and outline our plans for the future work in Sect. 6.

2 Background and Research Problem

OntoElect is the methodology for learning a domain ontology from a statistically representative sub-collection (DSC_{sat}) of the complete collection of documents ($DC = \{d_i\}$)[1] describing this subject domain. The representativeness of a sub-collection is decided using a successive approximation method, based on measuring terminological saturation. In this method, sub-collections are incrementally extended by adding several (inc) documents to the previous sub-collection in the sequence.

Let $DSC_1, DSC_2, \dots, DSC_i, \dots$ be the sequence of incrementally extended document sub-collections, such that $DSC_0 = \emptyset$ and $DSC_i = DSC_{i-1} \cup \{d_j\}_i$, where $d_j, j = 1, \dots, inc$, are chosen from the remainder of the DC using one of the possible ordering criteria [6]: chronological, reversed-chronological, bi-directional, random, or descending citation frequency. Let also $T_1, T_2, \dots, T_i, \dots$ be the bags of retained significant terms extracted from $DSC_1, DSC_2, \dots, DSC_i, \dots$. In OntoElect, the measure of terminological difference (thd) is used for comparing the bags of terms T_i, T_{i+1} retained from the successive DSC_i, DSC_{i+1}. It returns the difference as a real positive value. If, at some i: (i) thd goes below the threshold of the statistical error ε; and (ii) there is a convincing

[1] In OntoElect, we do not require the availability of this complete collection. Instead, we require that a substantial part of it is available, which presumably contains all the significant terms describing the subject domain. If so, it is further revealed that $DSC_{sat} \subset DC$.

evidence that it will never go above this threshold; then the difference (distance) between T_i and hypothetical T_{DC} is not higher than ε. Such a T_i could be used as an ε -approximation of the representative set of significant terms describing the domain. This representative set of terms is denoted as the **terminological basis** (TB) of the subject domain. This T_i, labelled further as T_{sat}, is denoted as the **saturated term set**, and the corresponding DSC_i, labelled further as DSC_{sat}, is the saturated DSC. The difference (thd) between T_{sat} and any successive T_i, including T_{DC}, is within the statistical error: $thd(T_{sat}, T_{CDC}) < \varepsilon$.

In our prior work, it has been demonstrated that thd is the measure, which can be effectively used for comparing terminological sets as vector representations of the semantic similarity/distance between document collections. However, one substantial shortcoming of this approach is that it is computationally expensive. Indeed, given an approximately fixed length of an increment $\{d_j\}_i$ and the increasing size of DSC_i, the method processes more and more the same part of the collection with the growth of i.

Therefore, the computational cost (run time) for measuring thd would have been substantially lowered if there was a way to process only the increments of the successive sub-collections. This processing is, essentially the ATE pipeline. Hence, the research problem is to prove that modifying the ATE processing pipeline for measuring terminological saturation to process:

- Only the disjoint parts $\{d_j\}_i$ of a document collection
- Instead of sub-collections DSC_i

yields the same result and takes substantially less execution time.

3 Related Work in ATE

In the majority of approaches to ATE [7, 8] processing is done in two consecutive phases: linguistic and statistical. Linguistic processors, like POS taggers or phrase chunkers, filter out stop words and restrict candidate terms to *n*-gram sequences: nouns or noun phrases, adjective-noun and noun-preposition-noun combinations. Statistical processing is then applied to measure the ranks of the candidate terms. These measures are [9]: either the measures of unithood, which focus on the collocation strength of units that comprise a single term; or the measures of termhood, which point to the association strength of a term to domain concepts.

For unithood, the measures are used such as mutual information [10], log likelihood [10], t-test [7, 8], modifiability and its variants [11, 8]. The measures for termhood are either term frequency-based (unsupervised approaches) or reference corpora-based (semi-supervised approaches). The most used frequency-based metrics are TF/IDF [12, 13], weirdness [14], and domain pertinence [15]. More recently, hybrid approaches were proposed, that combine unithood and termhood measurements in a single value. A representative measure is c/nc-value [16]. C/Nc-value-based approaches to ATE have received their further evolution in many works: [7, 15, 17] to mention a few.

Linguistic processing is organized and implemented in a very similar fashion in all ATE methods, except some of them that also include filtering out stop words. Stop words could be filtered out also at a cut-off step after statistical processing. Statistical processing is sometimes further split in two consecutive sub-phases of term candidate scoring, and ranking. For term candidates scoring, reflecting its likelihood of being a term, known methods could be distinguished by being based on (c.f. [12]) measuring occurrences frequencies (including word association), assessing occurrences contexts, using reference corpora, e.g. Wikipedia [18], topic modelling [19, 20].

The cut-off procedure, takes the top candidates, based on scores, and thus distinguishes significant terms from insignificant (or non-) terms. Many cut-off methods rely upon the scores, coming from one scoring algorithm, and establish a threshold in one or another way. Some others that collect the scores from several scoring algorithms use (weighted) linear combinations [21], voting [9, 3], or (semi-)supervised learning [22]. In our set-up [3], we do cut-offs after term extraction based on retaining a simple majority vote. Therefore, the ATE solutions, which perform cut-offs together with scoring, are not relevant for our approach.

Based on the evaluations in [9, 12, 23], the most widely used ATE algorithms, for which their performance assessments are published, are listed in Table 1. The table also provides the assessments based on the aspects we use for selection.

Table 1. Comparison of the most widely used ATE measures and algorithms (revision of the corresponding table in [24])

Method [Source]	**Domain-independence (+/-)**	**Supervizion (U/SS)**	**Measure(s)**	**Term Significance**	**Cut-off (+/-)**	**Precision (GENIA; average)**	**Run Time (related to c-value method)**	**Tool**
TTF [25]	+	U	Term (Total) Frequency	+	-			ATR4S
						0.70; 0.35	0.34	JATE
ATF [23]	+	U	Average Term Frequency	+	-	0.71; 0.33	0.37	ATR4S
						0.75; 0.32	0.35	JATE
TTF-IDF [26]	+	U	TTF+Inverse Document Frequency	+	-			ATR4S
						0.82; 0.51	0.35	JATE
RIDF [27]	+	U	Residual IDF	-		0.71; 0.32	0.53	ATR4S
						0.80; 0.49	0.37	JATE
C-value [16]	**+**	**U**	C-value, NC-value	**+**	**-**	0.73; **0.53**	1.00	ATR4S
						0.77; **0.56**	1.00	JATE
Weirdness [14]	+/-	SS	Weirdness	-		0.77; 0.47	0.41	ATR4S
						0.82; 0.48	1.67	JATE
GlossEx [21]	+	SS	Lexical (Term) Cohesion, Domain Specificity	-				ATR4S
						0.70; 0.41	0.42	JATE
TermEx [15]	+	SS	Domain Pertinence, Domain Consensus, Lexical Cohesion, Structural Relevance	-	+			ATR4S
						0.87; 0.46	0.52	JATE
PU-ATR [18]	-	SS	Nc-value, Domain Specificity	-	+	0.78; 0.57	809.21	ATR4S
								JATE

Comments to Table 1:

Domain Independence: "+" stands for a domain-independent method; "-" marks that the method is either claimed to be domain-specific by its authors, or is evaluated only on one particular domain. We look for a domain-independent method.

Supervision: "U" – unsupervised; "SS" – semi-supervised. We look for an unsupervised method.

Term Significance: "+" – the method returns a value for each retained term, which could further be used as a measure of its significance compared to the other terms; "-" marks that such a measure is not returned or the method does the cut-off itself. We look for doing cut-offs later.

Cut-off: "+" – the method does cut-offs itself and returns only significant terms; "-" – the method does not do cut-offs. We look for "-".

Precision and Run Time: The values are based on the comparison of the two cross-evaluation experiments reported in [12] and [23]. Empty cells in the table mean that there was no data for this method in this experiment using this tool. Survey [12] used ATR4S [12] – an open-source software tool for automated term recognition (ATR) written in Scala (4S). It evaluated 13 different methods, implemented in ATR4S, on five different datasets, including the GENIA dataset [28]. Survey [23] used JATE 2.0 [23], free software for automated term extraction (ATE) written in Java (J). It evaluated nine different methods, implemented in JATE, on two different datasets, including GENIA. Hence, the results on GENIA are the baseline for comparing the precision. Two values are given for each reference experiment: precision on GENIA; average precision. Both [12] and [23] experimented with c-value method, which was the slowest on average for [23]. So, the execution times for c-value were used as a baseline to normalize the rest in the **Run Time** column.
Tool: The last column in the table names the tools used in the corresponding experiments.

The information in Table 1 supports the conclusion of [23] stating that c-value is the most reliable method. The c-value method obtains consistently good results, in terms of precision, on the two different mixes of datasets [23, 12]. It could also be noted that c-value is one of the slowest in the group of unsupervised and domain-independent methods, though its performance is comparable with the fastest ones. Still, c-value outperforms the domain-specific methods, sometimes significantly – as it is in the case of PU-ATR. Therefore, we have chosen c-value as the method for our experimental framework.

4 Motivation and Formal Framework

ATE is known to be computationally expensive in the terms of run time versus the volume of input text. The c-value method that we have chosen for our terminological saturation measurement pipeline (Table 1) is more expensive than the other unsupervised and domain neutral methods. Furthermore, ATE implementations are often constrained[2] in the volume of input text. Hence, reducing the volume of text to be processed by the method, and partitioning it in relatively small chunks, may substantially lower this expense and contribute to the better scalability of the solution.

4.1 Motivation

The c-value method [16], as mentioned in Sect. 3, is hybrid and combines linguistic and statistical steps applied to the entire document collection (text corpus). The method starts with the linguistic pipeline, which outputs the list of term candidate strings. It then continues with the statistical part, which computes significance scores for these term candidates as c-values. The diagram of the measured run time versus the volume

[2] For example, the UPM Term Extractor software [29], which is based on the c-value method and used in our experiments, does not take in texts of more than 15 Mb in volume.

of input text, provided in Sect. 5 (Fig. 3), in the case of the conventional pipeline illustrates, by run time values, the computational complexity of the c-value method.

Let us now consider a document collection D as a composition of its disjoint parts.

Definition 1 (**A partial collection and a partition of a document collection**). *$D_i, i = 1, \ldots, n$ are the partial document collections of D* and $\{D_i\} = \{D_i\}_{i=1}^{n}$ is the partition of D *if the following conditions hold*:

$$\begin{aligned} &\textit{Condition } 1\text{: } D = \bigcup_{i=1}^{n} D_i, \\ &\textit{Condition } 2\text{: } \bigcap_{i=1}^{n} D_i = \emptyset. \end{aligned} \tag{1}$$

The linguistic part processes separate sentences. Therefore: (i) its computational complexity is the function of the number of sentences in a document collection; and (ii) the partial collections $D_i, i = 1, \ldots, n$ (Definition 1) could be processed independently and the outputs further merged. Hence, applying the linguistic step to the partition of D could at least be parallelized, which results in the runtime gain of n times.

In the case of OntoElect pipeline, the linguistic step is iteratively applied to the incrementally enlarged datasets (see Sect. 2). Therefore, the same chunks of text are processed many times. Let us suppose that D contains k documents and $inc = k/n$ is the increment to enlarge datasets. Then, the number of documents to be processed is:

- In the case of the incrementally enlarged datasets: $inc + 2 \cdot inc + \cdots + n \cdot inc = k \cdot \frac{1+2+\cdots+n}{n} \approx (n+1)/2 \cdot k$, which is substantially more than k if $n > 1$
- In the case of partial collections: k

Hence, processing partial collections instead of incrementally enlarged datasets gives a substantial additional gain in runtime, which is $(\frac{n+1}{2} - 1) \cdot k$ times.

Similarly, it might be reasonable to apply the statistical step of the pipeline not to the incrementally enlarged datasets, but to partial collections. However, it is not straightforward that:

- Computing c-values for the terms extracted from the partial collections; and
- Merging further these bags of terms with their significance scores

will give the same result as applying the statistical step to incrementally enlarged datasets. In the remainder of this section, we prove that partitioning c-value computation with later results merging gives correct results.

C-value [16], further labelled as cv in formulae and equations, is built using several statistical characteristics of the corresponding term candidate string. These characteristics are:

- The total frequency (number) of occurrence(s) of the candidate string in the document corpus
- The frequency (number) of occurrence(s) of the candidate string as a part of other longer candidate terms
- The number of these longer candidate terms
- The length of the candidate string (in the number of words)

Let: s be a term candidate string; $|s|$ – the length of s in words; ls – a longer term candidate string in which s is nested as a sub-string; $f(.)$ – the frequency (number) of occurrence(s) of a term candidate string in a collection of textual documents D; T^s – the set of extracted term candidate strings ls that nest s; and $P(T^s)$ – the number of these ls. Then a (complete) cv of s is denoted [16] as follows:

$$cv(s) = \begin{cases} log_2(|s|) \cdot f(s) & if\ s\ is\ not\ nested\ in\ any\ ls\ extracted\ from\ D \\ log_2(|s|) \cdot \left(f(s) - \frac{1}{P(T^s)} \sum_{ls \in T^s} f(ls)\right) & otherwise \end{cases} . \quad (2)$$

4.2 Merged Partial C-values

Definition 2 (**Partial c-value**). *The partial c-value of the term candidate string s extracted from the partial document collection D_i is computed as*:

$$pcv_i(s) = \begin{cases} log_2(|s|) \cdot f_i(s) & if\ s\ is\ not\ nested\ in\ any\ ls\ extracted\ from\ D_i \\ log_2(|s|) \cdot \left(f_i(s) - \frac{1}{P(T_i^s)} \sum_{ls \in T_i^s} f_i(ls)\right) & otherwise \end{cases} , \quad (3)$$

where: T_i^s is the set of term candidate strings ls, that nest s, extracted from D_i; $f_i(.)$ is the number of occurrences of s or ls in D_i.

Lemma 1 (**The total frequency of nested occurrences**). *The total value of the frequency of nested occurrences, in D, of a term candidate string s in longer term candidate strings ls is the sum of the total frequency values of nested occurrences in all partial collections D_i of D*:

$$tnest(s) = \sum_{ls \in T^s} f(ls) = \sum_{i=1}^{n} \left(\sum_{ls \in T_i^s} f_i(ls)\right) = \sum_{i=1}^{n} (tnest_i(s)). \quad (4)$$

Proof. It implies from Definition 2 (of partial c-value), that $tnest_i(s)$ is the total number of occurrences of the term candidate string s in all longer term candidate strings ls extracted from the partial collection D_i. The number of these longer term candidate strings equals to $P(T_i^s)$. Due to the disjointness of the partial collections D_i (Condition 2 of Definition 1), $f(ls) = \sum_{i=1}^{n} f_i(ls)$. Therefore, and due to the Condition 1 of Definition 1, the total number of occurrences of s in all ls extracted from D is:

$$tnest(s) = \sum_{ls \in T^s} f(ls) =$$
$$= \sum_{ls \in T^s} \sum_{i=1}^{n} f_i(ls) = \sum_{ls \in \cup_{i=1}^{n}(T_i^s)} (\sum_{i=1}^{n} f_i(ls)) =$$
$$= \sum_{i=1}^{n} \left(\sum_{ls \in T_i^s} f_i(ls)\right) = \sum_{i=1}^{n} (tnest_i(s)).$$

□

Definition 3 (**Merged partial c-value**). *The merged partial c-value of the term candidate string s is computed as*:

$$mpcv(s) = \sum_{i=1}^{n} pcv_i(s). \quad (5)$$

The following Theorem 1 allows computing $cv(s)$ for the whole collection D based on the merging of the known partial c-values $pcv_i(s), i = 1, \dots, n$ for the partial collections $D_i, i = 1. \dots, n$ of D.

Theorem 1 (**Equality of $\boldsymbol{cv}$ and $\boldsymbol{mpcv}$**). *If a document collection D is partitioned as $\{D_i\} = \{D_i\}_{i=1}^n$, which means that Conditions 1 and 2* (1) *hold, then*

$$cv(s) = mpcv(s) \tag{6}$$

Proof. The proof is structured in three cases: (1) s is never nested in ls; (2) $\forall D_i$, s is nested at least once and at least in one ls; and (3) s is nested in ls for some D_i.

Case 1: not nested. If, $\forall i = 1, \dots, n$, s extracted from D_i is not nested in any ls extracted from D_i, then s is not nested in any ls extracted from D. Therefore, for such an s:

$$\begin{aligned} mpcv(s) &= \textstyle\sum_{i=1}^n pcv_i(s) = \sum_{i=1}^n log_2(|s|) \cdot f_i(s) = \\ &= log_2(|s|) \cdot \textstyle\sum_{i=1}^n f_i(s) = log_2(|s|) \cdot f(s) = cv(s), \end{aligned} \tag{7}$$

due to Conditions 1 and 2 (2) and the definition of $f(.)$.

Case 2: all nested. If, $\forall i = 1, \dots, n$, s extracted from D_j is nested in an ls extracted from D_j, then: (i) this s (extracted from D) is nested in this ls (extracted from D); and (ii) $ls \in T_j^s \subset T^s$ – because $D_i \subset D$ due to condition 1. Therefore:

$$\begin{aligned} mpcv(s) &= \textstyle\sum_{i=1}^n pcv_i(s) = \\ &= \textstyle\sum_{i=1}^n \left(log_2(|s|) \cdot \left(f_i(s) - \frac{1}{P(T_i^s)} \sum_{ls \in T_i^s} f_i(ls) \right) \right) = \\ &= log_2(|s|) \cdot \textstyle\sum_{i=1}^n \left(f_i(s) - \frac{1}{P(T_i^s)} \sum_{ls \in T_i^s} f_i(ls) \right) = \\ &= log_2(|s|) \cdot \left(\textstyle\sum_{i=1}^n f_i(s) - \sum_{i=1}^n \left(\frac{1}{P(T_i^s)} \sum_{ls \in T_i^s} f_i(ls) \right) \right) = \\ &= log_2(|s|) \cdot \left(f(s) - \textstyle\sum_{i=1}^n \left(\frac{1}{P(T_i^s)} \sum_{ls \in T_i^s} f_i(ls) \right) \right) \approx_{|h1} \\ &\approx_{|h1} log_2(|s|) \cdot \left(f(s) - \frac{1}{P(T^s)} \textstyle\sum_{i=1}^n \left(\sum_{ls \in T_i^s} f_i(ls) \right) \right) = \\ &= log_2(|s|) \cdot \left(f(s) - \frac{1}{P(T^s)} \textstyle\sum_{ls \in T^s} \left(f(ls) \right) \right) = cv(s) \end{aligned} \tag{8}$$

Here "$\approx_{|h1}$" stands for hypothetically approximately equal. The hypothesis $h1$ about the approximate equality in $\sum_{i=1}^n \left(\frac{1}{P(T_i^s)} \right) \approx \frac{1}{P(T^s)}$. Formally, $\sum_{i=1}^n \left(\frac{1}{P(T_i^s)} \right) > \frac{1}{P(T^s)}$. However, asymptotically, $P(T_i^s) = o(tnest_i(s))$ and $P(T^s) = o\big(tnest(s)\big)$ due to: (i) the overlaps in T_i^s; and (ii) possible nestings in several instances of ls. Therefore, the influence of those denominators in (8) becomes lower with the growth of the volume of D and its partial collections D_i. This is a promise that $h1$ might be true.

Case 3: s is sometimes nested in ls. There exist several partial collections D_j, for which Case 2 is applied. For the rest of partial collections D_k Case 1 is applied. In this situation two partial sums – $mpcv_1(s)$ and $mpcv_2(s)$ – are computed for these disjoint subsets of the partition of D. Similarly to Case 2, $cv(s) \approx_{|h1} mpcv_1(s) + mpcv_2(s)$.

Hence, if the hypotheses $h1$ holds true, Cases 1-3 prove Theorem 1.

□

For checking $h1$, complete (cv) and merged partial ($mpcv$) c-values are experimentally computed and compared, as presented in Sect. 5.

A straightforward corollary from Theorem 1 is that c-values do not depend on the partitioning of a document collection.

Corollary 1 (**Size of a partial collection**). Let $\{D_i\}_{i=1}^{n}$; $\{D_j\}_{j=1}^{m}$; $n \neq m$ be two different partitions of a document collection D. Then:

$$\forall s, mpcv(s)|_{\{D_i\}} = mpcv(s)|_{\{D_j\}} \approx_{|h1} cv(s), \tag{9}$$

where: s is a term candidate string extracted from the document collection D; $mpcv(s)|_{\{D_i\}}$ is the merged partial c-value of the term candidate string s computed for the partition $\{D_i\}$ of D; $mpcv(s)|_{\{D_j\}}$ is the merged partial c-value of the term candidate string s computed for the partition $\{D_j\}$ of D.

Based on Corollary 1, the size of a partial collection $D_i \in \{D_i\}$ may be reasonably chosen based on the specifics of the problem and available hardware resources – RAM in particular. One possible scenario and problem might be extracting terms from a stream of textual documents, like blog posts or tweets. In this setting, the size of a partial collection has to be smaller than the size of the stream window.

```
Algorithm MPCV. Merge partial c-values from two Bags of Terms
Input:
 Ti, Ti+1 – the bags of retained significant terms.
      Each term Ti.term is accompanied with its Ti.pcv.
      Ti, Ti+1 are sorted in the descending order of Ti.pcv, Ti+1.pcv.
Output: the bag of terms Ti+1 with merged Ti.pcv into Ti+1.pcv for every term
resort := .FALSE.
for k := 1 to |Ti|
  match := .FALSE.
  for m := 1 to |Ti+1|
      if (Ti.term[k] = Ti+1.term[m])
          then begin Ti+1.pcv[m] += Ti.pcv[k]; match := .TRUE.; end
  if (.NOT. match)
     then begin append(Ti.term[k]+Ti.pcv[k], Ti+1); resort := .TRUE.; end
   end for
end for
if (resort) then sort(Ti+1, Ti+1.pcv, desc)
```

Fig. 1: The **MPCV** algorithm for merging partial c-values in two bags of retained significant terms

The **MPCV** algorithm (Fig. 1) is used for merging partial c-values in the bags of significant terms retained from the textual datasets representing the partial document collections of the complete document collection.

5 Experimental Evaluation

The idea of experimental evaluation is to compare the conventional and optimized processing pipelines based on checking:

- **The correctness**. Are the merged partial c-values computed using the optimized pipeline practically the same as the c-values computed by the conventional pipeline?
- **Execution time**. What is the difference in the duration of the extraction of the same bags of retained significant terms between the conventional and optimized pipelines?

Checking correctness validates the hypothesis $h1$ (Sect. 4) to fully prove Theorem 1. If $h1$ holds true, the optimized processing pipeline could be used for extracting terms in the process of measuring terminological saturation in document collections. Comparing the execution times of the conventional and optimized processing pipelines allows assessing the efficiency of the optimized pipeline.

5.1 Experimental Data

The document collection used in our experiments is the DMKD-300 collection, which contains the subset of (300) full text articles from the Springer journal on Data Mining and Knowledge Discovery[3] published between 1997 and 2010. These papers have been automatically pre-processed to plain texts [24] and have not been cleaned. Therefore, the resulting datasets, representing partial collections, were moderately noisy. We have chosen the increment (inc) for generating the datasets to be 20 papers. Hence, based on the available texts, we have generated, using our Dataset Generator software (Sect. 5.3):

- 15 incrementally extended datasets $D_1 = \{d_j\}_{j=1}^{20}$ (20 papers), $D_2 = D_1 \cup \{d_j\}_{j=1}^{20}$ (40 papers), …, $D_{15} = D_{14} \cup \{d_j\}_{j=1}^{20}$ (300 papers)[4] for the conventional pipeline
- 15 datasets of inc size forming the partition $\left\{D_i = \{d_j\}_{j=1}^{20}\right\}_{i=1}^{15}$[5] of the DMKD-300 collection for the optimized pipeline

The **descending citation frequency** (**DCF**) order [6] of adding documents to partial collections has been used in both cases.

5.2 Instrumental Software and Computational Environment

Our experimental workflow is appropriately supported by the developed and used instrumental software. The toolset is concisely presented in Table 2.

[3] https://link.springer.com/journal/10618

[4] **DMKD-300** collection in plain texts: http://dx.doi.org/10.17632/knb8fgyr8n.1#folder-637dc34c-fa29-4587-9f63-df0e602d6e86; incrementally enlarged datasets generated of these texts: http://dx.doi.org/10.17632/knb8fgyr8n.1#folder-b307088c-9479-43fb-8197-a12a66ff685b

[5] The partition of the **DMKD-300** collection: https://github.com/OntoElect/Data/blob/master/DMKD-300-DCF-Part.zip

Table 2: The modules of the instrumental software toolset used in experiments

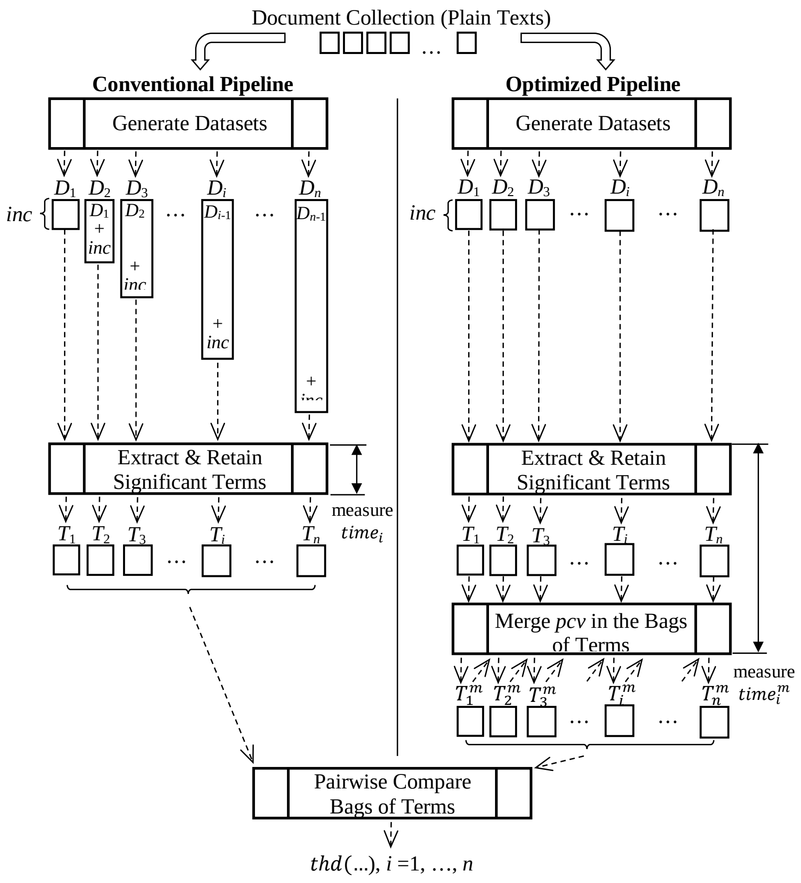
Phase / Task	Tool	Input	Output	Implementation	Constraints
Pre-Processing Phase					
Generate Datasets	Dataset Generator	the folder with plain text documents; the XLS file with citation frequencies and document file names	the folder with Plain Text datasets; the table with run time per dataset	C#, https://github.com/OntoElect/Code/tree/master/DataSetGen-cs	
Terms Extraction Phase					
Extract Terms	UPM Term Extractor [29]	the folder with plain text datasets	the folder with the bags of terms; the table with run-time per bag of terms	Java, https://github.com/ontologylearning-oeg/epnoi-legacy	English texts only, *c-value* method [16], input data of at most 15 Mb
Merge partial c-values	MPCV	the folder with the bags of terms; the list of files to be processed	the folder with the bags of terms with merged c-values; the table with run-time per bag of terms	Python, https://github.com/OntoElect/Code/tree/master/MPCV	
Post-processing Phase					
Compute Terminological Differen-ces	Baseline THD	the folder with the bags of terms; the list of files to be processed	The table containing *eps*, *thd*, values for the consecutive pairs of the bags of terms	Python, https://github.com/OntoElect/Code/tree/master/THD	uses the baseline THD algorithm [4]

5.3 Experimental Flow

The set-up of our experiments includes the configuration of the execution flow in two parallel processing pipelines – conventional and optimized, as pictured in Fig. 2.

The conventional pipeline implements the processing of incrementally extended document sub-collections, as explained in Sect. 2. It takes in the files of the document collection in the specified (**DCF**) order and generates the incrementally extended datasets (Sect. 5.1) using the dataset generator (Table 2). At the next step, the datasets are fed into the term extractor software (Table 2) which outputs the bags of extracted terms T_i and measures run times $time_i$.

The optimized pipeline implements the processing of the partitioned document sub-collections as explained in Sect. 4. It takes the files of the document collection in the same order (**DCF**) and generates partition datasets (Sect. 5.1) using the dataset generator (Table 2). At the next step, the datasets are fed into the term extractor software (Table 2) which outputs the bags of extracted terms T_i and measures run times. At the subsequent step, the extracted sets o terms are fed into the merger module (Table 2) which applies the **MPCV** algorithm (Sect. 4.2) consequently to the pairs $\{T_i, T_{i+1}\}$ as pictured in Fig. 2. As a result, the merged bags of terms $T_i^m = \bigcup_{k=1}^{i} T_i$ are generated. The run times of the merge operation are also measured. The required total run times ($time_i^m$) are computed as the sums of the respective term extraction and merge run times.



Document Collection (Plain Texts)
Conventional Pipeline
Optimized Pipeline
Generate Datasets
D_1 D_2 D_3 D_i D_n
inc
D_1 + inc
D_2 + inc
D_{i-1} + inc
D_{n-1} + inc
Extract & Retain Significant Terms
measure $time_i$
T_1 T_2 T_3 T_i T_n
Merge pcv in the Bags of Terms
measure $time_i^m$
T_1^m T_2^m T_3^m T_i^m T_n^m
Pairwise Compare Bags of Terms
$thd(...)$, i =1, …, n

Fig. 2: The execution flow of evaluation experiments

After executing these two parallel branches, if $h1$ (Sect.4) holds true, T_i coming from the conventional pipeline and T_i^m coming from the optimized pipeline have to contain very similar sets of terms with approximately the same c-values. This is checked by applying the THD algorithm [4] implemented in the Baseline THD module (Table 2). THD is applied: (i) to the pairs $\{T_i, T_{i+1}\}$ and $\{T_i^m, T_{i+1}^m\}$ for comparing saturation curves for conventional and optimized cases; and (ii) to the pairs $\{T_i, T_i^m\}$ for computing terminological difference between hypothetically the same sets of terms.

All the computations, except term extraction, have been run on a Windows 7 64-bit HP ZBook 17 G3 PC with: Intel® Core™ i7-6700HQ CPU, E7400 @ 2.60 GHz; 8.0 Gb on-board memory; NVIDIA Qadro M3000M GPU. Term extraction has been run on an Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz, 64 cores, 256GB server.

5.4 The Results of Experiments and Discussion

The results of our experiment are presented in a tabular form in Table 3 and graphically pictured in Fig. 3 and Fig. 4.

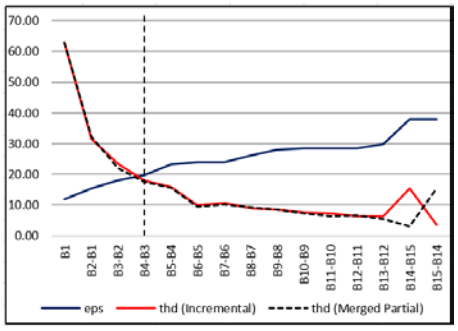
Table 3: *thd* measurements and run times for the conventional and optimized pipelines

Bag of Terms No (*i*)	***eps***	***thd***			**Run Times (sec)**	
		$\boldsymbol{T_{i-1}, T_i}$	$\boldsymbol{T^m_{i-1}, T^m_i}$	$\boldsymbol{T_i, T^m_i}$	$\boldsymbol{T_i}$	$\boldsymbol{T^m_i}$
1	12.00	62.89	62.89	0.00	30.41	31.94
2	15.50	31.51	32.22	3.64	60.47	32.79
3	18.00	23.38	22.17	2.85	85.34	31.64
4	19.65	18.04	17.63	3.08	111.72	31.85
5	23.22	15.80	15.57	3.31	147.30	37.11
6	24.00	10.07	9.50	3.86	153.78	32.25
7	24.00	10.67	10.14	3.40	195.27	41.49
8	26.00	9.14	9.19	5.28	217.62	43.28
9	28.00	8.68	8.54	6.95	268.59	47.16
10	28.53	7.56	7.42	5.21	296.49	41.32
11	28.53	7.30	6.44	6.13	324.39	41.58
12	28.53	6.53	6.56	5.88	363.05	45.20
13	30.00	6.43	5.42	10.07	401.94	40.55
14	38.00	15.47	3.13	9.09	401.19	39.81
15	38.00	3.53	15.37	6.14	459.75	50.12

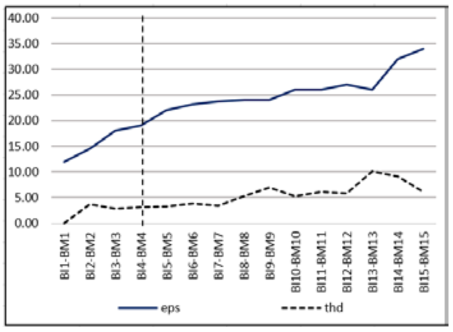
Fig. 3(a) clearly shows that the results of saturation measurements using the bags of terms resulting from the optimized pipeline ($\boldsymbol{T^m_{i-1}, T^m_i}$ column in Table 3 and Merged Partial curve in Fig. 3(a)) and conventional pipeline ($\boldsymbol{T_{i-1}, T_i}$ column in Table 3 and Incremental curve in Fig. 3(a)) are practically the same, except the last two measurements. The deviation at the tail could be explained that regular noise is accumulated differently in these two cases. A nice side result in this context is that the optimized pipeline using merged partial c-values accumulates less regular noise. Fig. 3(b) clearly pictures the fact that the difference between the bags of terms T_i and T^m_i does not exceed approximately 1/3 of the individual term significance threshold eps that is used to cut-off insignificant terms. In the combination, these two observations reliably prove[6] our hypothesis $h1$ (Sect. 4).

The comparison of the run times presented in Table 3 and Fig. 4 clearly demonstrates that the optimized pipeline, with near to constant values, outperforms the conventional pipeline significantly.

[6] One may argue that the reported experiment is just an experiment with one document collection. Hence for a different document collection the results might be different regarding the validity of $h1$. Our counter-argument is that the computation of c-values is collection and domain-independent. Furthermore, the terms with the same c-value are randomly distributed in the documents of the collection.



(a) Saturation measurements



(b) Terminological differences

Fig. 3: Merged partial c-values computed using the optimized pipeline are practically the same as the c-values computed using the conventional pipeline

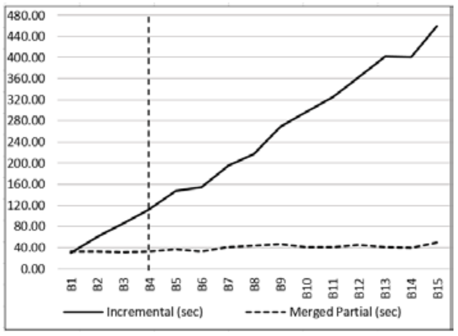


Fig. 4: Run times of the conventional (Incremental) and optimized (Merged Partial) pipelines

6 Conclusions and Future Work

The contribution of this paper is the proposal of computing significance scores (c-values) for term candidates extracted from a document collection using not the incrementally extended datasets, representing sub-collections, but the partitions of the collection. It has been proven formally, up to the validity of the $h1$ hypothesis, that this optimized approach is correct – i.e. gives practically the same results. The hypothesis has been validated experimentally, by comparing the outputs coming from the conventional and optimized processing pipelines.

The experiment also clearly showed that the proposed way of text processing very substantially outperforms the conventional approach. The run times measured while processing partitions of the document collection remained almost constant in consecutive steps. A tiny increase was observed due to the very small overhead for merging the bags of terms extracted from the partition datasets. Yet one more advantage of the proposed approach is that partition datasets could be processed independently as these do not overlap in data. Hence, the optimized pipeline is straightforwardly parallelizable. This fact opens the way to process real world document collections at industrial scales for finding terminological cores within these collections. Choosing a proper partition size also removes the limitation of many software term extractors on the volume of input data.

Our plan for the future work is to apply the optimized processing pipeline to detect terminological saturation in the industrial size paper collection in the domain of Knowledge Management.

References

1. Wong, W., Liu, W., Bennamoun, M.: Ontology learning from text: a look back and into the future. ACM Comput. Surv., 44(4), Article 20, 36 pages (2012). http://doi.acm.org/10.1145/2333112.2333115
2. Ermolayev, V.: OntoElecting requirements for domain ontologies. The case of time domain. EMISA Int J of Conceptual Modeling 13(Sp. Issue), 86--109 (2018)
3. Tatarintseva, O., Ermolayev, V., Keller, B., Matzke, W.-E.: Quantifying ontology fitness in OntoElect using saturation- and vote-based metrics. In: Ermolayev, V., et al. (eds.) Revised Selected Papers of ICTERI 2013, CCIS, vol. 412, pp. 136--162 (2013)
4. Chugunenko, A., Kosa, V., Popov, R., Chaves-Fraga, D., Ermolayev, V.: Refining terminological saturation using string similarity measures. In: Ermolayev, V, et al. (eds.): Proc. ICTERI 2018. Volume I: Main Conference, Kyiv, Ukraine, May 14-17, 2018, CEUR-WS vol. 2105, pp. 3--18 (2018, online) http://ceur-ws.org/Vol-2105/10000003.pdf
5. Ermolayev, V., Batsakis, S., Keberle, N., Tatarintseva, O., Antoniou, G.: Ontologies of time: review and trends. International Journal of Computer Science and Applications 11(3), 57--115 (2014)
6. Kosa, V., Chaves-Fraga, D., Naumenko, D., Yuschenko, E., Moiseenko, S., Dobrovolskyi, H., Vasileyko, A., Badenes-Olmedo, C., Ermolayev, V., Corcho, O., and Birukou, A.: The influence of the order of adding documents to datasets on terminological saturation. Technical Report TS-RTDC-TR-2018-2-v2, 21.11.2018, Dept. of Computer Science, Zaporizhzhia National University, Ukraine, 72 p. (2018)
7. Fahmi, I., Bouma, G., van der Plas, L.: Improving statistical method using known terms for automatic term extraction. In: Computational Linguistics in the Netherlands, CLIN 17 (2007)
8. Wermter, J., Hahn, U.: Finding new terminology in very large corpora. In: Clark, P., Schreiber, G. (eds.) Proc.3rd Int Conf on Knowledge Capture, K-CAP 2005, pp. 137--144, Banff, Alberta, Canada, ACM (2005) http://doi.org/10.1145/1088622.1088648
9. Zhang, Z., Iria, J., Brewster, C., Ciravegna, F.: A comparative evaluation of term recognition algorithms. In: Proc. 6th Int Conf on Language Resources and Evaluation, LREC 2008, Marrakech, Morocco (2008)
10. Daille, B.: Study and implementation of combined techniques for automatic extraction of terminology. In: Klavans, J., Resnik, P. (eds.) The balancing act: combining symbolic and statistical approaches to language, pp. 49--66. The MIT Press. Cambridge, Massachusetts (1996)
11. Caraballo, S. A., Charniak, E.: Determining the specificity of nouns from text. In: Proc. 1999 Joint SIGDAT Conf on Empirical Methods in Natural Language Processing and Very Large Corpora, pp. 63--70 (1999)
12. Astrakhantsev, N.: ATR4S: toolkit with state-of-the-art automatic terms recognition methods in scala. arXiv preprint arXiv:1611.07804 (2016)