Title: Similar Terms Grouping Yields Faster Terminological Saturation

Author(s), **ORCID**(s): Victoria Kosa[0000-0002-7300-8818], David Chaves-Fraga[0000-0003-3236-2789], Natalya Keberle[0000-0001-7398-3464] and Aliaksandr Birukou[0000-0002-4925-9131]
Group: Intelligent Systems
Project: OntoElect

Published as: Kosa V., Chaves-Fraga D., Keberle N., Birukou A. (2019) Similar Terms Grouping Yields Faster Terminological Saturation. In: Ermolayev V., Suárez-Figueroa M., Yakovyna V., Mayr H., Nikitchenko M., Spivakovsky A. (eds) Information and Communication Technologies in Education, Research, and Industrial Applications. ICTERI 2018. Communications in Computer and Information Science, vol 1007. Springer, Cham

Abstract: This paper reports on the refinement of the algorithm for measuring terminological difference between text datasets (THD). This baseline THD algorithm, developed in the OntoElect project, used exact string matches for term comparison. In this work, it has been refined by the use of appropriately selected string similarity measures (SSM) for grouping the terms, which look similar as text strings and presumably have similar meanings. To determine rational term similarity thresholds for several chosen SSMs, the measures have been implemented as software functions and evaluated on the developed test set of term pairs in English. Further, the refined algorithm implementation has been evaluated against the baseline THD algorithm. For this evaluation, the bags of terms have been used that had been extracted from the three different document collections of scientific papers, belonging to different subject domains. The experiment revealed that the use of the refined THD algorithm, compared to the baseline, resulted in quicker terminological saturation on more compact sets of source documents, though at an expense of a noticeably higher computation time.
Keywords: Automated Term Extraction, OntoElect, Terminological Difference, String Similarity Measure, Bag of Terms, Terminological Saturation.

DOI: https://doi.org/10.1007/978-3-030-13929-2_3

Similar Terms Grouping Yields Faster Terminological Saturation

Victoria Kosa[1] [0000-0002-7300-8818], David Chaves-Fraga[2] [0000-0003-3236-2789],
Natalya Keberle[1] [0000-0001-7398-3464] and Aliaksandr Birukou[3, 4] [0000-0002-4925-9131]

[1] Department of Computer Science, Zaporizhzhia National University,
Zaporizhzhia, Ukraine
{victoriya1402.kosa, nkeberle}@gmail.com,
[2] Ontology Engineering Group, Universidad Politécnica de Madrid, Madrid, Spain
dchaves@fi.upm.es
[3] Springer-Verlag GmbH, Heidelberg, Germany
aliaksandr.birukou@springer.com
[4] Peoples' Friendship University of Russia (RUDN University), Moscow, Russia

Abstract. This paper reports on the refinement of the algorithm for measuring terminological difference between text datasets (THD). This baseline THD algorithm, developed in the OntoElect project, used exact string matches for term comparison. In this work, it has been refined by the use of appropriately selected string similarity measures (SSM) for grouping the terms, which look similar as text strings and presumably have similar meanings. To determine rational term similarity thresholds for several chosen SSMs, the measures have been implemented as software functions and evaluated on the developed test set of term pairs in English. Further, the refined algorithm implementation has been evaluated against the baseline THD algorithm. For this evaluation, the bags of terms have been used that had been extracted from the three different document collections of scientific papers, belonging to different subject domains. The experiment revealed that the use of the refined THD algorithm, compared to the baseline, resulted in quicker terminological saturation on more compact sets of source documents, though at an expense of a noticeably higher computation time.

Keywords: Automated Term Extraction, OntoElect, Terminological Difference, String Similarity Measure, Bag of Terms, Terminological Saturation.

1 Introduction

The research presented in this paper[1] is the part of the development of the methodological and instrumental components for extracting representative (complete) sets of significant terms from the representative sub-collections of textual documents having minimal possible size. These terms are further interpreted as the required features for engineering an ontology in a particular domain of interest. Therefore, it is assumed that the

[1] This paper is a refined and extended version of [1].

documents in a collection cover a single and well-circumscribed domain. The main hypothesis, put forward in this work, is that a sub-collection can be regarded as representative to describe the domain, if it is the terminological core. It means that any additions of extra documents from the entire collection to this sub-collection do not noticeably change the terminological footprint on the domain. Such a sub-collection is further considered as complete. Therefore, a representative bag of significant terms describing its domain can be extracted from it. The approach to assess the representativeness does so by evaluating terminological saturation in a document (sub-)collection [2], [3]. Practically, the approach allows extracting statistically the same set of significant terms from a part of a collection, instead of processing the whole collection. Automated term extraction is known to be a computationally bulky process. Therefore, lowering the number and the overall volume of the input documents might substantially decrease processing times and improve scalability. For example, the terminological core of the TIME collection of 437 conference papers (Section 5.2), detected using the baseline THD algorithm, contains 220 papers (50.34 percent of the total number). We demonstrate in the paper that using the proposed THD refinement, the size of a terminological core is additionally lowered by 22 to 46 percent without any loss in quality.

Detecting saturation is done by measuring terminological difference (*thd*) among the pairs of the consecutive incrementally enlarged datasets, as described in Section 5. This measure is based on evaluating differences between individual terms. A (baseline) THD algorithm for computing *thd* [2] has been developed and implemented in the OntoElect project[2]. OntoElect develops a methodology and instrumental tool suite for refining domain ontologies. It exploits the allusion of public elections to find out what is the prevailing sentiment of the domain knowledge stakeholders. The sentiments are elicited indirectly, through terms extraction from a saturated document collection, describing the domain. Term significance scores are interpreted as the votes in favour of the corresponding ontology features. The features satisfying a simple majority of voters are represented by the terms, ordered by descending scores, with the minimal sum of the scores being higher than 1/2 of the total sum.

The baseline THD algorithm uses a simple string equivalence check for detecting similar (the same) individual terms. The objective of the research presented in this paper is to find out if it is possible to achieve better performance in measuring terminological difference by using a proper string similarity measure to compare individual terms.

The remainder of the paper is structured as follows. Section 2 reviews the related work and outlines our contributions. Section 3 presents the chosen string similarity measures and reports about the choice of the proper terms similarity thresholds for terms grouping. Section 4 sketches out the approach of OntoElect for measuring *thd* and presents our refinement of the baseline THD algorithm. Section 5 reports the setup and results of our evaluation experiments. Our conclusions are given and plans for the future work outlined in Section 6.

[2] https://www.researchgate.net/project/OntoElect-a-Methodology-for-Domain-Ontology-Refinement

2 Related Work

The work related to the presented research has been sought in: (i) automated term extraction (ATE) from English texts; (ii) string similarity (distance) measurement in the pairs of text strings.

2.1 Automated Term Extraction

In the majority of approaches to ATE, e.g. [4] or [5], processing is done in two consecutive phases: linguistic processing and statistical processing. 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 [6]: 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 [7], log likelihood [7], t-test [4, 5], modifiability and its variants [8, 5]. The measures for termhood are either term frequency-based (unsupervised approaches) or reference corpora-based (semi-supervised approaches). The most used frequency-based measures are TF/IDF (e.g. [9, 10]), weirdness [11], and domain pertinence [12]. More recently, hybrid approaches were proposed, that combine unithood and termhood measurements in a single value. A representative measure is c/nc-value [13]. C/nc-value-based approaches to ATE have received their further evolution in many works, e.g. [4, 12, 14] to mention a few.

Linguistic processing is organized and implemented in a very similar way 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 measuring occurrences frequencies, including word association (c.f. [9]) or assessing occurrences contexts, using reference corpora, e.g. Wikipedia [15], or topic modelling [16, 17].

2.2 Text Similarity Measurement

In the recent surveys on text similarity measurement approaches, e.g. [18, 19], methods (or measures)[3] are grouped based on analysing: (i) characters and their sequences; (ii) tokens; (iii) terms; (iv) text corpora; or (v) synsets. In [19] hybrid measures that allow fuzzy matching between tokens are also mentioned. Brief characteristics of the groups are given immediately below. The individual methods belonging to the groups are detailed in Table 1.

[3] In this context, we do not distinguish a method and a measure. A method is understood as a way to implement the corresponding measure function.

Character- and character sequence-based measures compare characters and their sequences in strings, taking into account also the order of characters. These include the measures of common character sequences, e.g. substrings; edit distance; the number and order of the common characters between two strings.

Token-based methods model a string as a set of tokens. Individual characters, character *n*-grams, or separate words could be regarded as tokens. Quantification is done by computing the size of the overlap normalized by a measure of string length.

Term-based measures are similar to token-based measures but the tokens are different. Those are not character *n*-grams but terms, which are word *n*-grams with possibly varying *n*. Furthermore, the weights of the terms, e.g. their frequencies of occurrence, are taken into account. These measures apply more on long character strings, or documents, hence are better suited to measure document or text dataset similarity.

Corpus-based and **synset-based** (or knowledge-based) **methods** are very marginally relevant to our purposes in this paper. Corpus-based approaches determine the similarity between words based on (statistical) information gained from large text corpora. Synset-based approaches rely on semantic networks, like WordNet [20], to derive semantic similarity between words. Both approaches are therefore too bulky computationally, though may be applied to ATE – e.g. for deciding about cut-offs. Term grouping, the technique we report in this paper, is however performed after the terms have already been extracted. Hence, we omit looking at corpus- and synset-based measures.

The overview of the most popular text / string similarity measures, grouped by method types, is provided in Table 1. This overview is by far not complete as many other variants of SSM are available in the literature. Those we omit are however based on the same principles compared to the listed in Table 1, to the best of our knowledge.

Table 1: The overview of text similarity / distance measures

Name, source	**Description**	**Specifics**	**Relevance**	
			Term Similarity	***thd*** [4]
Character- and character sequence-based measures				
Longest Common Substring [21]	common character sequence based measure	returns the integer length of the longest common substring; could be normalized by the total length	moderate	irrelevant
Levenshtein distance [22]	edit distance based measure	returns an integer number of required edits	marginal	irrelevant
Hamming distance [23]	edit distance based measure	strings have to be of equal length	marginal	irrelevant
Monger-Elkan distance [24]	edit distance based measure	returns an integer number of required edits	marginal	irrelevant

[4] *thd* is the measure for terminological difference developed in OntoElect [2] and used in our approach – see also Section 4. Hence, “relevant” in this column means being appropriate for measuring terminological difference between documents of text datasets.

Name, source	**Description**	**Specifics**	**Relevance**	
			Term Similarity	***thd*** [4]
Jaro distance [25]	counts the minimal number of one character transforms in one string for arriving at the other string	returns a normalized real value from [0, 1]	good	irrelevant
Jaro-Winkler distance [26]	refines Jaro measure by using a prefix scale value – prioritizes the stings that match at the beginning	returns a normalized real value from [0, 1]	good	irrelevant
Token-based measures				
Sørensen-Dice coefficient [27, 28]	Counts the ratio of identical character *bi*-grams to the overall number of bi-grams in both strings	returns a normalized real value from [0, 1]	good	irrelevant
Jaccard similarity [29]	counts the ratio between the cardinalities of the intersection and union of the character sets (*uni*-grams) in the strings	returns a normalized real value from [0, 1]	good	irrelevant
Cosine similarity [19]	Size of overlap in character *uni*-grams divided by the square root of the sum of the squared total numbers of uni-grams in both strings	returns a normalized positive real value	marginal (computationally hard)	irrelevant
Term-based measures				
Euclidian distance [30]	Measures traditional Euclidian distance in an *n*-dimensional metric space (of positive reals)	works for documents; returns a real positive value	irrelevant	relevant
Cosine similarity [31]	Computes a cosine between two vectors in the term space; vectors are specified by term weights (e.g. TF of C-value)	works for documents; returns a normalized positive real value	irrelevant	marginal
Pearson correlation [30]	Computes Pearson correlation for a pair of vectors in the term vector space	works for documents; returns a normalized real value that ranges from +1 to −1; it is 1 when vectors are fully identical	irrelevant	marginal
Manhattan (block) distance [18]	the distance to be traveled to get from one data point to the other if a grid-like path is followed	works for documents; resembles the *thd* measure [2]	irrelevant	relevant

The authors of [32] present an expansion-based framework to measure string similarities efficiently while considering synonyms. This result is also relevant to our work as a synonym is one of the categories of term candidates that may need to be considered for grouping in our settings. In [32], it is also acknowledged that there is a rich set of string similarity measures available in the literature, including character *n*-gram similarity [33], Levenshtein distance [22], Jaro-Winkler measure [26], Jaccard similarity [29], TF/IDF based cosine similarity [34], and Hidden Markov Model-based measure [35].

2.3 Contributions

In this work, we do not contribute any novel method for ATE. The c-value method [13] implemented in the UPM Term Extractor [36] is used as this combination of the method and implementation has been experimentally proven to be the best appropriate for detecting terminological saturation [37].

In difference and complementary to the abovementioned relevant work, we contribute several novel things. Firstly, we propose a way to rationally choose the thresholds that are used to regard string similarity as term similarity (Section 3). Secondly, we develop an algorithm for similar terms grouping that uses string similarity measures and term similarity thresholds (Section 4). Based on its use, we propose the refinement of the baseline THD algorithm [2] for measuring terminological difference between two subsequent text datasets (Section 4).

3 The Choice of SSMs and Terms Similarity Thresholds

From the variety of SSMs, mentioned above, due to the specifics of our task of the approximate comparison of short strings containing a few words, we filter out those: (i) that require long strings or sets of strings of a considerably big size; (ii) that are computationally hard. We also keep the representatives of all kinds of string metrics in our short list as much as possible. As a result, we form the following list of measures to be considered for further use:

- Character-based measures: Levenshtein distance [22], Hamming distance [23], Jaro similarity [25], and Jaro-Winkler similarity [26]
- Token-based measures: Jaccard similarity (*uni*-gram comparison) [29], cosine similarity (*uni*-gram comparison) [19], and Sørensen-Dice coefficient (*bi*-gram comparison) [27, 28]

Among those, Levenshtein and Hamming distances appear to be the least appropriate in our context due to their specifics. Levenshtein returns an integer number of required edits, while the rest of the measures return normalized reals. Hence, it is not clear if normalizing Levenshtein would make the result comparable to the other measures in a way to use the same term similarity threshold. Hamming measure is applicable only to the strings of equal lengths. Adding spaces to the shorter string, however, may lower the precision of measurement. Cosine similarity is based on the same principle as Jaccard, but is more computationally complex due to the presence of the square root in the denominator. Therefore, we finally choose to use Jaro, Jaro-Winkler, Jaccard[5], and Sørensen-Dice for implementation and evaluation in our work. Further, it is briefly explained how the selected measures are computed.

Jaro similarity sim_j between two strings S_1 and S_2 is computed (1) as the minimal number of one character transforms to be done to the first term (string) for getting the second string in the compared pair.

[5] It is expected that Cosine measure, being based on the same principle as Jaccard, is not better than Jaccard in terms of performance, though takes more time to be computed.

$$sim_j = \begin{cases} 0, & if \quad m = 0 \\ 1/3 * (\frac{m}{|S_1|} + \frac{m}{|S_2|} + \frac{m-t}{m}) & otherwise \end{cases}, \qquad (1)$$

where: $|S_1|$, $|S_2|$ are the lengths of the compared strings; *m* is the number of the matching characters; and *t* is the half of the number of transposed characters. The characters are matching if they are the same and their distance from the beginning of the string differs by no more than $\lfloor max(|S_1|, |S_2|)/2 \rfloor - 1$. The number of transposed characters is the number of matching but having different sequence order symbols.

Jaro-Winkler similarity measure sim_{j-w} refines Jaro similarity measure by using a prefix scale value *p*, which assigns better ratings to the strings that match from their beginnings for a prefix length *l*. Hence, for the two strings S_1 and S_2 it is computed as shown in (2).

$$sim_{j-w} = sim_j + l * p * (1 - sim_j), \qquad (2)$$

where: l is the length of a common prefix (up to a maximum of 4 characters); p is a constant scaling factor meaning how much the similarity value is adjusted upwards for having common prefixes (up to 0.25, otherwise the measure can become larger than 1; [26] suggests that *p* = 0.1).

Sometimes Winkler's prefix bonus $l * p * (1 - sim_j)$ is given only to the pairs having Jaro similarity value higher that a particular threshold. This threshold is suggested [26] to be equal to 0.7.

Jaccard similarity index sim_{ja} is a similarity measure for finite sets, characters in our case. It is computed, for the two strings S_1 and S_2, as the ratio between the cardinalities of the intersection and union of the character sets in S_1 and S_2 as shown in (3).

$$sim_{ja} = (|S_1| \cap |S_2|)/(|S_1| \cup |S_2|) \qquad (3)$$

Finally, the Sørensen-Dice coefficient is computed by counting identical character *bi*-grams in S_1 and S_2 and relating these to the overall number of *bi*-grams (4).

$$sim_{sd} = 2n_{\equiv} / (n_{S_1} + n_{S_2}), \qquad (4)$$

where: $n_{\equiv}$ is the number of *bi*-grams found in S_1 and also in S_2; n_{S_1}, n_{S_2} are the numbers of all *bi*-grams in S_1 and S_2 respectively.

For the proper use of the implemented SSM functions in the context of terms comparison and grouping, it is necessary to determine what would be a reasonable threshold to distinguish between (semantically) similar and different terms. For finding that out, the following cases in string comparison need to be taken into account.

Full Positives (FP). In this case, evaluated character strings are fully the same, which clearly gives similar (the same) terms.

Full Negatives (FN). In this case, evaluated character strings are very different and the terms in these strings carry different semantics. This is also a clear situation and is characterized by low values of similarity measures.

Partial Positives (PP). In this case, evaluated character strings are partially the same and the terms in these strings carry the same or similar semantics. The terms in such strings are similar, though it may not be fully clear. The following are different categories of terms that fall into this case: the words in the terms have different endings (e.g. plural/singular forms); different delimiters are used (e.g. "-", or "–", or " - "); a symbol is missing, erroneously added, or misspelled (a typo); one term is the sub-string of the other (e.g. subsuming the second); one of the strings contains unnecessary extra characters (e.g. two or three spaces instead of one, or noise).

Partial Negatives (PN). In this case, evaluated character strings are partially the same but the terms in these strings carry different semantics. The terms in such strings are different, though it may not be fully clear. The following are the categories that fall into this case: the terms in the compared strings differ by a very few characters, but have substantially different meanings (e.g. "deprecate" versus "depreciate"); the compared multi-word terms have common word(s) but fully differ in their meanings (e.g. "affect them" versus "effect them"). These PN are the hardest case to be detected.

The test set of term pairs falling into the cases and categories described above has been manually developed[6]. For each pair of terms in this test set, all four selected string similarity measures have been computed. The extract is presented in Table 2.

Table 2: Similarity measures for different test cases

Ca-se	**Cate-gory**	**Terms Pair**	**Sørensen-Dice**	**Jaccard**	**Jaro**	**Jaro-Winkler**
Different (**FN**)		whirled | world	0.0	0.5	0.790	0.811
		traces | creta	0.0	0.833	0.588	0.588
		time domain | ontology lifecycle	0.0	0.428	0.445	0.445
Same (**FP**)		identical strings | identical strings	1.0	1.0	1.0	1.0
Similar Semantics (**PP**)	Extra charac-ters	*system?problems | system problems	0.814	0.769	0.936	0.936
		sad data mining | sqr data mining	0.769	0.818	0.859	0.873
	Common parts (words)	marcov chain monte carlo methods | monte carlo methods	0.782	0.766	0.629	0.666
		data mining algorithm | data mining	0.642	0.666	0.842	0.904
		cation error | error	0.533	0.333	0.427	0.427
	Typos	fraud detection | froud ditection	0.714	0.916	0.859	0.887
		monte carlo | monte ??rlo	0.7	0.727	0.878	0.927
		data mining | data minin	0.941	0.875	0.969	0.981
	Differe nt delimite	computer science | computerscience	0.896	0.916	0.979	0.987
		serial episodes | serial&&episodes	0.827	0.818	0.936	0.961
		data cube | data_cube	0.75	0.777	0.925	0.955
	Different endings	network structure | network structures	0.969	1.0	0.981	0.988
		time complexity | time complexities	0.896	0.833	0.981	0.951
		value | values	0.888	0.833	0.918	0.951
Different Se-	Common parts (words)	database | military base	0.400	0.500	0.410	0.410
		brainstorm | stormy weather	0.363	0.428	0.509	0.509
		iron clad | iron maiden	0.444	0.636	0.804	0.882
		jellyfish | fish tank	0.352	0.307	0.614	0.614
		four delegates | delegated authority	0.451	0.666	0.557	0.557

[6] The test set and computed term similarity values are publicly available at https://github.com/OntoElect/Data/blob/master/STG/Test-Set.xls

		string theory | string format	0.583	0.571	0.812	0.887
	Very few character differences	deprecate against | depreciate against	0.909	1.0	0.903	0.941
		alternately move | alternatively move	0.933	0.916	0.9	0.94
		affect them | effect them	0.9	0.758	0.906	0.906

Table 3: Average string similarity measure values for different categories of term pairs from the test set

Case / Category	**Items in Test Set**	**Sørensen-Dice**	**Jaccard**	**Jaro**	**Jaro-Winkler**
Different strings (FN)	**6**	**0.03**	**0.45**	**0.55**	**0.55**
Identical strings (FP)	**3**	**1.00**	**1.00**	**1.00**	**1.00**
Similar Semantics (PP)	**32**	**0.71**	**0.72**	**0.63**	**0.70**
- Unnecessary (extra) characters	7	0.8401	0.8820	0.8714	0.8784
- Common parts (words)	6	0.7122	0.7280	0.6375	0.7043
- Typos	6	0.7797	0.8637	0.8863	0.9220
- Different delimiters	6	0.7860	0.8473	0.9125	0.9442
- Different endings	7	0.8911	0.9135	0.9410	0.9590
Different Semantics (PN)	**18**	**0.89**	**0.89**	**0.89**	**0.91**
- Common parts (words)	11	0.4336	0.5221	0.6161	0.6408
- Very few character differences	7	0.8826	0.8845	0.8914	0.9059
Total:	**59**				

Table 4: Term similarity thresholds chosen for experimental evaluation

Method	**Term Similarity Thresholds**			
	Min	*Ave-1*	*Ave-2*	*Max*
Sørensen-Dice	0.71	0.76	0.83	0.89
Jaccard	0.72	0.77	0.83	0.89
Jaro	0.63	0.72	0.80	0.89
Jaro-Winkler	0.70	0.77	0.84	0.91

The average values of all four chosen similarity measures for each category have been computed using all the test set term pairs falling into this category. These values are presented in Table 3. Term similarity thresholds have to be chosen such that full and partial negatives are regarded as not similar, but full and partial positives are regarded as similar. Hence, for the case of partial positives, the thresholds have to be chosen as minimal of all the case categories, and for the partial negatives – as the maximal of all the case categories. The values of case thresholds are shown in bold in Table 3. These are further used as the margins for relevant threshold intervals in our experiments. These intervals have been evenly split by the four threshold points, as presented in Table 4. The requirements for partial positives and negatives unfortunately contradict to each other. For example, if a threshold is chosen to filter out partial negatives, also some of the partial positives will be filtered out. Therefore, subsuming that partial negatives are rare, it has been decided to use the thresholds for partial positives.

4 OntoElect and the Refinement of the THD Algorithm

OntoElect, as a methodology, seeks for maximizing the fitness of the developed ontology to what the domain knowledge stakeholders think about the domain. Fitness is measured as the stakeholders' "votes" – a measure that allows assessing the stakeholders' commitment to the ontology under development, reflecting how well their sentiment about the requirements is met. The more votes are collected, the higher the commitment is expected to be. If a critical mass of votes is acquired (say 50%+1, which is a simple majority vote), it is considered that the ontology meets the requirements satisfactorily.

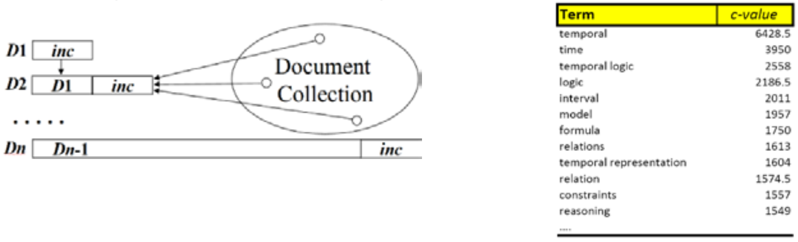
Unfortunately, direct acquisition of requirements from domain experts is not very realistic. The experts are expensive and not willing to do the work, which falls out of their core activity. That is why the OntoElect approach focuses on the indirect collection of the stakeholders' votes by extracting these from high quality and reasonably high impact documents authored by the stakeholders.

An important feature to be ensured for knowledge extraction from text collections is that the dataset needs to be representative to cover the opinions of the domain knowledge stakeholders satisfactorily fully. OntoElect suggests a method to measure the terminological completeness of the document collection by analysing the *saturation* of terminological footprints of the incremental slices of the document collection [2].

The approach followed in our work is finding the terminological core of a document collection by measuring terminological saturation [2, 3]. This measurement is done using our terminological difference measure (*thd*, [2]) which is a variant of a Manhattan distance measure (see e.g. [18]) or Minkovski's distance with p=1 [38].

The full texts of the documents from a collection are grouped in datasets in the order of their timestamps. As pictured in Fig. 1 (a), the first dataset D_1 contains the first portion of (*inc*) documents. The second dataset D_2 contains the first dataset D_1 plus the second incremental slice of (*inc*) documents. Finally, the last dataset D_n contains all the documents from the collection.

At the next step of the OntoElect workflow, the bags of multi-word terms B_1, B_2, …, B_n are extracted from the datasets D_1, D_2, …, D_n together with their significance (*c-value*) scores, using UPM Term Extractor software [36]. An example of an extracted bag of terms is shown in Fig. 1 (b).



Term	c-value
temporal	6428.5
time	3950
temporal logic	2558
logic	2186.5
interval	2011
model	1957
formula	1750
relations	1613
temporal representation	1604
relation	1574.5
constraints	1557
reasoning	1549
….	

(a) (b)

Fig. 1: (a) Incrementally enlarged datasets in OntoElect; (b) An example of a bag of terms extracted by UPM Term Extractor [36].

At the subsequent step, every extracted bag of terms B_i, $i = 1, \ldots, n$ is processed as follows. Firstly, an individual term significance threshold (*eps*) is computed to cut off those terms that are not within the majority vote. The sum of *c-value*s with individual values above *eps* form the majority vote if this sum is higher than ½ of the sum of all *c-value*s. Secondly, insignificant term candidates are cut-off at *c-value* < *eps*. Thirdly, the normalized scores are computed for each individual term: *n-score* = *c-value* / max(*c-value*). Finally, the result is saved in the bag of retained significant terms T_i. After this step only significant terms, that represent the majority vote, are considered. T_i are then evaluated for saturation by measuring pair-wise terminological difference between the subsequent bags T_i and T_{i+1}, $i = 0, \ldots, n$-1. So far in OntoElect, it has been done by applying the baseline THD algorithm[7] [2] presented in Fig. 2.

```
Algorithm THD. Compute Terminological Difference between Bags of Terms
Input:
 Ti, Ti+1 – the bags of terms with grouped similar terms.
      Each term Ti.term is accompanied with its T.n-score.
      Ti, Ti+1 are sorted in the descending order of T.n-score.
 M –  the name of the string similarity measure function to compare terms
 th – the value of the term similarity threshold from within [0,1]
Output: thd(Ti+1, Ti), thdr(Ti+1, Ti)
1.  sum := 0
2.  thd := 0
3.  for k := 1, |Ti+1|
4.    sum :=  sum + Ti+1.n-score[k]
5.    found : = .F.
6.    for m := 1, |Ti|
7.       if (Ti+1.term[k] = Ti.term[m]) | if (M(Ti+1.term[k], Ti.term[m], th))
8.          then
9.            thd += |Ti+1.n-score[k] - Ti.n-score[m]|
10.           found := .T.
11.    end for
12.    if (found = .F.) then thd += Ti+1.n-score[k]
13. end for
14. thdr := thd  / sum
```

Fig. 2: Baseline THD algorithm [2] for measuring terminological difference in a pair of bags of terms and its refinement. Baseline THD uses string equalities for comparing terms (dashed rounded rectangle in line 7). The refinements are shown in solid rounded rectangles. Refined THD has two more input parameters (*M* and *th*) and uses *M* for comparing terms (line 7).

In fact, the THD algorithm accumulates the *n-score* differences, in the *thd* value for the bag T_{i+1}, if there were the same terms in T_i and T_{i+1}. If there was no the same term in T_i, it adds the *n-score* of the orphan to the *thd* value of T^{i+1}. After *thd* has been computed, the relative terminological difference *thdr* receives its value as *thd* divided by the sum of *n-scores* in T_{i+1}.

Absolute (*thd*) and relative (*thdr*) terminological differences are computed for further assessing if T_{i+1} differs from T_i more than the individual term significance threshold

[7] The baseline THD algorithm is implemented in Python and is publicly available at https://github.com/OntoElect/Code/tree/master/THD

eps. If not, it implies that adding an increment of documents to D_i for producing D_{i+1} did not contribute any noticeable amount of new terminology. Hence, the subset D_{i+1} of the overall document collection may have become terminologically saturated. However, to obtain more confidence about the saturation, OntoElect suggests that more subsequent pairs of T_i and T_{i+1} are evaluated. If stable saturation is observed, then the process of looking for a minimal saturated sub-collection could be stopped.

```
Algorithm STG. Group similar terms in the bag of terms
Input:
 T  – a bag of terms. Each term T.term is accompanied with its
      T.n-score. T is sorted in the descending order of T.n-score.
 M – the name of the string similarity measure function to compare
     terms
 th – the value of the term similarity threshold from within [0,1]
Output: T with grouped similar terms
    sum := 0
    for k = 1,|T|
      term :=  T.term[k]
      n-score := T.n-score[k]
      count := 1
      for m = k+1,|T|
           if M(term, T.term[m], th)
              then
                n-score += T.n-score[m]
                count += 1
                remove(T[m])
      end for
      T.n-score[k] := n-score / count
    end for
```

Fig. 3: Similar Term Grouping (STG) algorithm

Our task is to modify the THD algorithm in a way to allow finding not exactly the same but sufficiently similar terms by applying string similarity measures with appropriate thresholds, as explained in the previous Section 3. For that, the preparatory similar term grouping step has been introduced to avoid duplicate similarity detection. For each of the compared bags of terms T_i and T_{i+1} the similar term grouping (STG) algorithm is applied at this preparatory step – see Fig. 3. After term grouping is accomplished for both bags of terms, the refined THD algorithm (Fig. 2 – rounded rectangles) is performed to compute the terminological difference between T_i and T_{i+1}.

5 Evaluation

This section reports on our evaluation of the refined THD algorithm against the baseline THD [2]. This evaluation is performed using the workflow of the OntoElect Requirements Elicitation Phase [3] and three document collections from different domains: TIME, DMKD-300, and DAC-cleaned. Section 5.1 outlines the set-up of our evaluation experiments. The document collections are presented in Section 5.2. The results of our evaluation experiments are discussed in Section 5.3.

5.1 The Set-up of the Experiments

The objective of our experiments is to find out if using the refined THD algorithm yields quicker terminological saturation compared to the use of the baseline THD algorithm. We are also looking at finding out which string similarity measures best fit for measuring terminological saturation.

For making the results comparable, the same datasets, created from the document collections as described in Section 5.2, are fed into both the refined and baseline THD algorithms. For each document collection, we apply:

1. The refined THD – sixteen times – one per individual string similarity measure *M* [8] (Section 3) and per individual term similarity threshold *th* (Table 4); and
2. The baseline THD – one time as it does not depend on a term similarity threshold

The values of: (i) the number of retained terms; (ii) absolute terminological difference (*thd*); and (iii) the time taken to perform similar terms grouping by the STG algorithm (*sec*) are measured.

Finally, to verify if our SSM implementations, and hence the STG and refined THD algorithms, are correct, we check if the refined THD algorithm implementation returns the results which are satisfactorily similar to that of the baseline THD when the terms similarity threshold is set to 1.00. This threshold value straightforwardly means that only equivalent strings have to be regarded as similar terms.

All the computations are run using a Windows 7 64-bit PC with: Intel® Core™ i5 CPU, M520 @ 2.40 GHz; 8.0 Gb on-board memory; NVIDIA Geforce GT330M GPU.

5.2 Experimental Data

The document collections used in our experiments are all composed of the papers published at the peer-reviewed international venues in three different domains:

- The TIME collection contains the full text papers of the proceedings of the Time Representation and Reasoning (TIME) Symposia series[9] published between 1994 and 2013
- The DMKD-300 collection is composed of the subset of full text articles from the Springer journal on Data Mining and Knowledge Discovery[10] published between 1997 and 2010
- The DAC-cleaned collection comprises the subset of full text papers of the Design Automation Conference[11] published between 2004 and 2006

[8] The functions for all the four selected SSMs have been implemented in Python 3.0 and return real values within [0, 1]. These functions are publicly available at: https://github.com/OntoElect/Code/tree/master/STG/core/methods

[9] http://time.di.unimi.it/TIME_Home.html

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

[11] http://dac.com/

The **chronological** order of adding documents is chosen for generating experimental datasets from the documents of all the three collections using our Dataset Generator [37]. The characteristics of all the document collections and generated datasets are summarized in Table 5.

Table 5: The characteristics of the used document collections and datasets

Document Collection	Paper Type and Layout	No Doc	Noise	Processing	Inc	No Datasets
TIME	conference, IEEE 2-column	437	manually cleaned	manual conversion to plain text, automated dataset generation	20 papers	22
DMKD-300	journal, Springer 1-column	300	not cleaned, moderately noisy	automated [37]	20 papers	15
DAC-cleaned	conference, IEEE 2-column	506	quite noisy	automated, stop terms removal [37]	20 papers	26

5.3 Results and Discussion

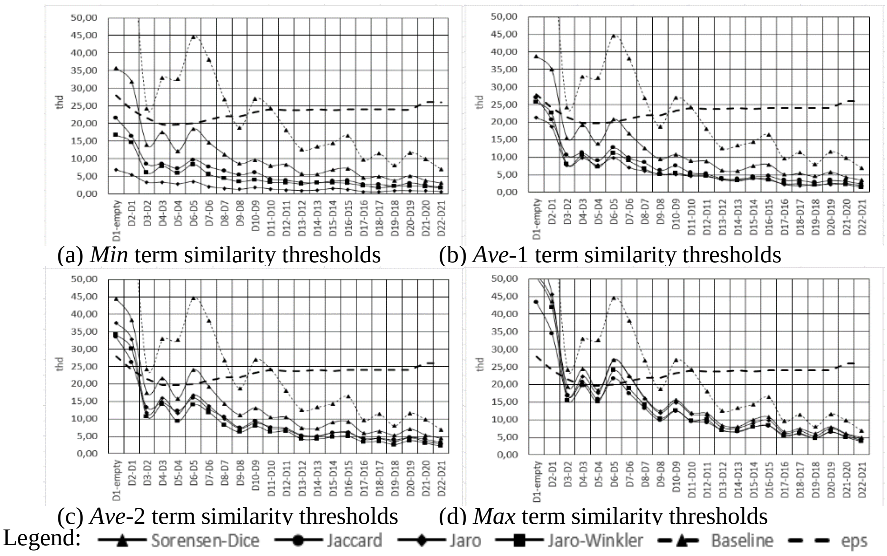
The measurements, taken in our experiments for different collections and terms similarity threshold points, are not presented in the paper in a tabular form due to page limits. Instead, the results are presented diagrammatically in figures below and made available in full, including values, publicly online[12].

The results of our measurements of terminological saturation (*thd*) are pictured in Fig. 4–6. Saturation (*thd*) measurements reveal that the refined THD algorithm detects terminological saturation faster than the baseline THD algorithm, no matter what the chosen term similarity measure (*M*) or similarity threshold (*th*) is. If the results for different measures are compared, then it may be noted that the respective saturation curves behave differently, depending on the similarity threshold point.

Overall, as one could see in Fig. 4–6 (a) – (d), the use of the Sørensen-Dice measure demonstrates the least volatile behaviour along the terms similarity threshold points. Sørensen-Dice also results in making the refined THD algorithm to detect saturation slower than the three other measures for *Min*, *Ave*-1, and *Ave*-2. For *Max*, it is as fast as Jaro and slightly slower than Jaccard and Jaro-Winker.

One more observation is that, integrally, all the implemented term similarity measures coped well with retaining significant terms from all the three document collections. This is indicated by the co-locations of terminology contribution peaks at the diagrams (a) – (d) in Fig. 4–6. One can see in Fig. 4–6(d), for the *Max* threshold point, that all the string similarity methods curves follow the shape of the baseline THD curve quite closely. Hence, they have the peaks exactly at the same *thd* measurement points where the baseline has, pointing at more new significant terms. The most sensitive to terminology contribution peaks was Sørensen-Dice.

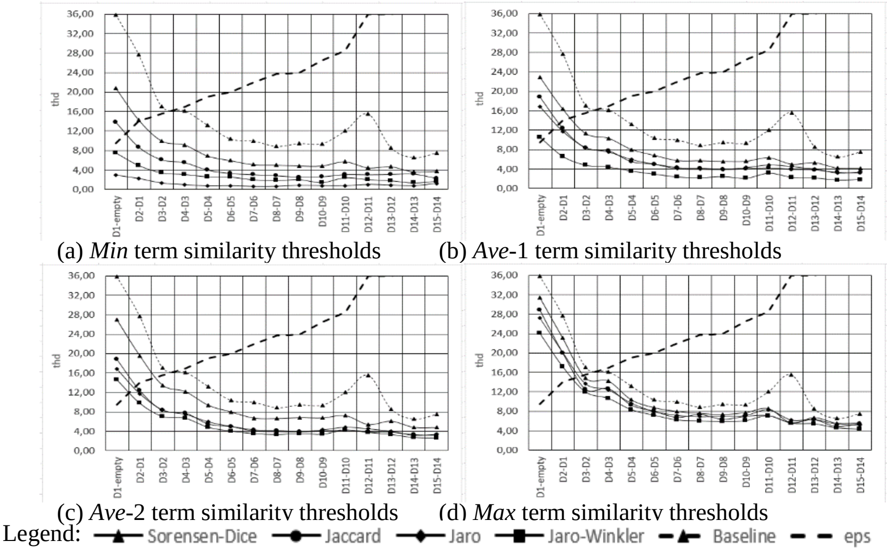
[12] https://github.com/OntoElect/Experiments/tree/master/STG. File names are {TIME, DMKD-300, DAC-cleaned}-Results-Alltogether-{min, ave, ave2, max, 1}-th.xlsx.



(a) *Min* term similarity thresholds (b) *Ave*-1 term similarity thresholds

(c) *Ave*-2 term similarity thresholds (d) *Max* term similarity thresholds

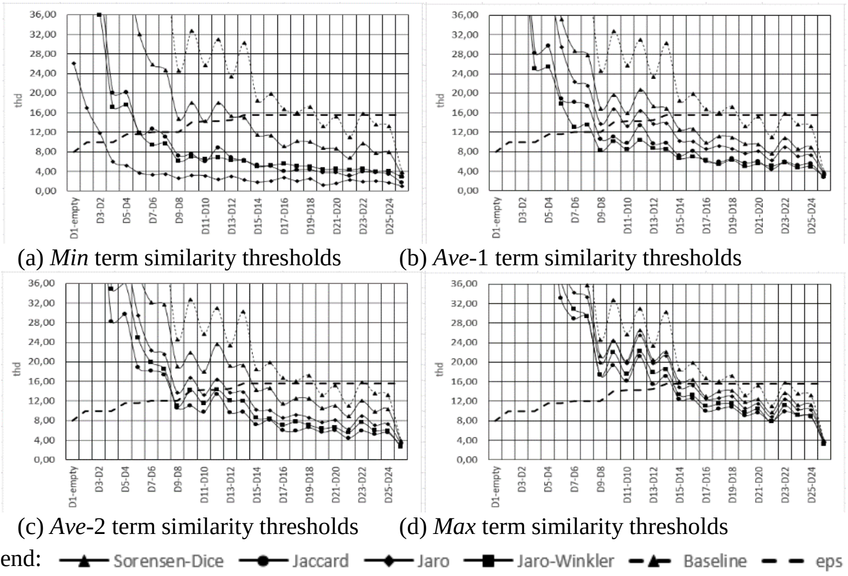
Fig. 4: Terminological saturation measurements on TIME for different similarity threshold points



(a) *Min* term similarity thresholds (b) *Ave*-1 term similarity thresholds

(c) *Ave*-2 term similarity thresholds (d) *Max* term similarity thresholds

Fig. 5: Terminological saturation measurements on DMKD-300 for different similarity threshold points



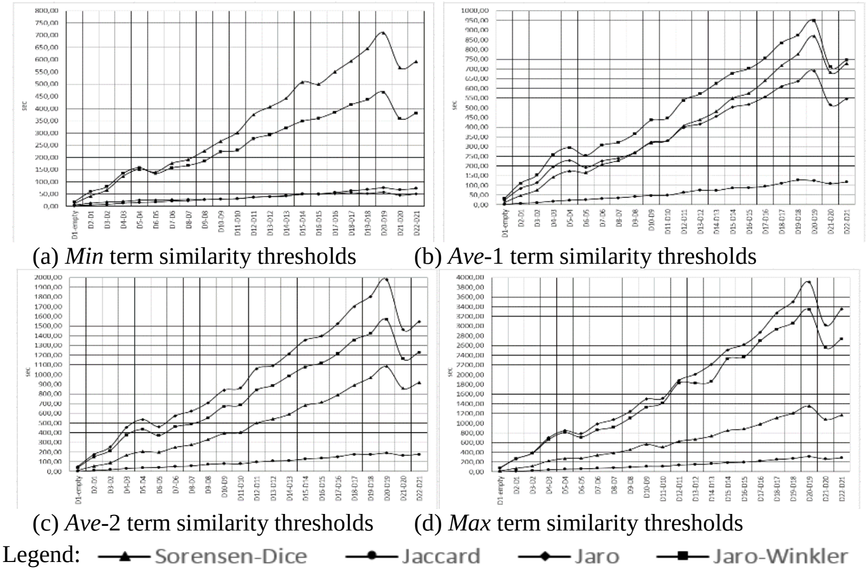
(a) *Min* term similarity thresholds (b) *Ave*-1 term similarity thresholds

(c) *Ave*-2 term similarity thresholds (d) *Max* term similarity thresholds

Legend: Sorensen-Dice Jaccard Jaro Jaro-Winkler Baseline eps

Fig. 6: Terminological saturation measurements on DAC-cleaned for different similarity threshold points

The diagrams in Fig. 7–9 show the times spent by the STG algorithm to detect and group similar terms for different chosen term similarity thresholds. One particular diagram corresponds to a particular terms similarity threshold point (*Min*, *Ave*-1, *Ave*-2, and *Max*).

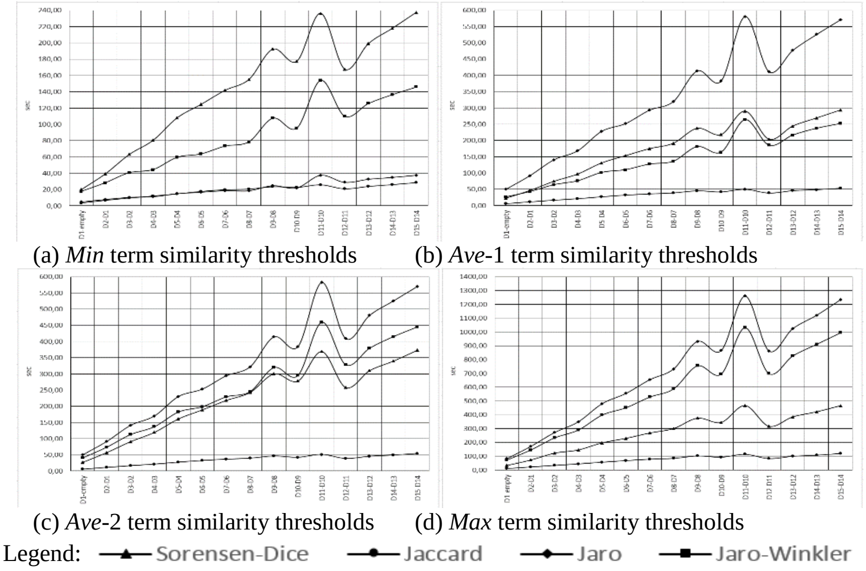


(a) *Min* similarity thresholds (b) *Ave*-1 term similarity thresholds

(c) *Ave*-2 term similarity thresholds (d) *Max* term similarity thresholds

Legend: Sorensen-Dice Jaccard Jaro Jaro-Winkler

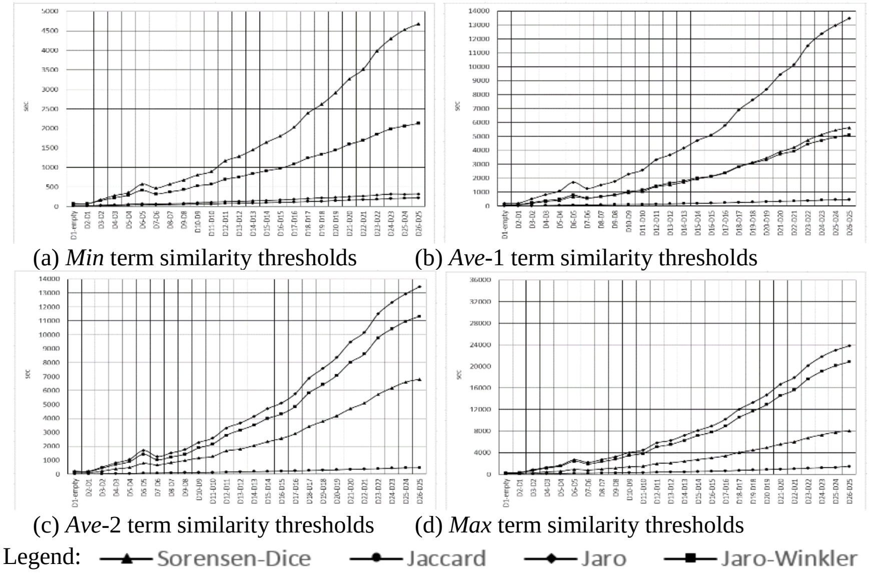
Fig. 7: Time (sec) spent by the STG algorithm for grouping similar terms on TIME bags of terms



(a) *Min* term similarity thresholds (b) *Ave*-1 term similarity thresholds

(c) *Ave*-2 term similarity thresholds (d) *Max* term similarity thresholds

Fig. 8: Time (sec) spent by the STG algorithm for grouping similar terms on DMKD-300 bags of terms



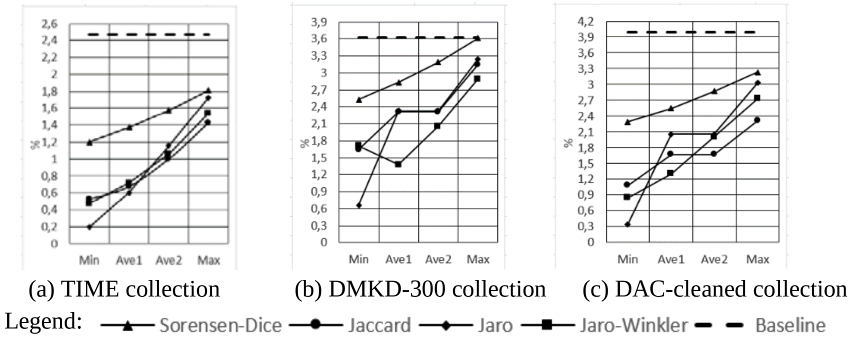
(a) *Min* term similarity thresholds (b) *Ave*-1 term similarity thresholds

(c) *Ave*-2 term similarity thresholds (d) *Max* term similarity thresholds

Fig. 9: Time (sec) spent by the STG algorithm for grouping similar terms on DAC-cleaned bags of terms

It needs to be mentioned that the introduction of string similarity measures in the computation of terminological difference (THD algorithm) increases the computational complexity quite substantially. As it could be noticed in Fig. 7–9 (a) – (d), the times grow with the value of the terms similarity threshold (*th*) and reach thousands of seconds for *Max* threshold values. It is worth acknowledging that Sørensen-Dice and Jaccard are substantially more stable to the increase of *th* than Jaro and Jaro-Winkler. Sørensen-Dice takes, however, times more time than Jaccard. From the other hand, Jaccard is not very sensitive to terminological peaks and retains significantly less terms than Sørensen-Dice.

Fig. 10 pictures the proportions of the retained to all extracted terms when saturation has been detected, computed at different terms similarity threshold points, for the bags of terms extracted from our three document collections. It is clear from Fig. 10 that Sørensen-Dice yields the second highest proportions for all the collections and used term similarity thresholds, after the baseline, which does not group terms.



(a) TIME collection (b) DMKD-300 collection (c) DAC-cleaned collection

Fig. 10: The proportions of retained to all extracted terms for different term similarity measures per document collections

Finally, the terms similarity threshold is set to 1.00 and the refined THD implementation is evaluated for all three collections for the pairs of the bags of terms in a few pair vicinity of the saturation points. The task is to check if the refined THD with similar terms grouping: (i) detects terminological saturation at the same point as the baseline THD, therefore, *thd* values are measured; and (ii) retains the same number of significant terms as the baseline THD, therefore, the numbers of retained terms are measured. We are also interested in comparing the time taken to accomplish term grouping (STG).

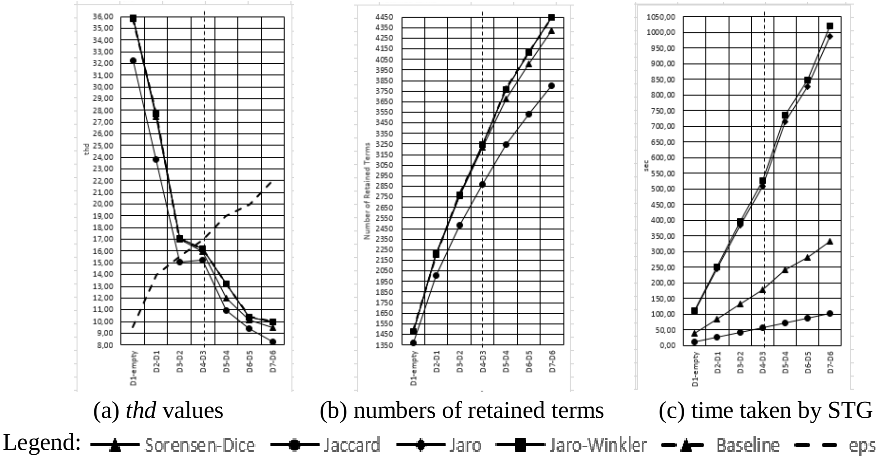
The results for the DMKD-300 collection are presented graphically in Fig. 11. The results for the TIME and DAC-cleaned collections[13] are very much similar to these for DMKD-300 and do not change our conclusion and recommendation.

It may be seen in Fig. 11 (a) and (b) that Jaro and Jaro-Winkler implementations fully repeat the baseline THD results, both in the measured *thd* values and numbers of retained significant terms. Sørensen-Dice behaves similarly to Jaro and Jaro-Winkler up to the saturation point. After that, it returns slightly lower *thd* and retains slightly

[13] These results could be accessed at https://github.com/OntoElect/Experiments/tree/master/STG. File names are {TIME, DMKD-300, DAC-cleaned}-Results-Alltogether-1-th.xlsx.

less significant terms. This behaviour is acceptable as the measurements after the saturation point are of marginal interest. Jaccard implementation however appears to return significantly lower *thd* values and significantly less retained terms at all measurement points – before and after detecting saturation. Jaccard also detects saturation one measurement point earlier than the rest of the SSMs, which is not correct for this threshold (1.00).

Fig. 11 (c) reveals that, for being accurate in measurements at the very high threshold of 1.00, Jaro and Jaro-Winkler take too much of a computational overhead. Sørensen-Dice and Jaccard however remain more stable to the increase of the *th*, similarly as it was before for *Ave1*, *Ave2*, and *Max* threshold points.



(a) *thd* values (b) numbers of retained terms (c) time taken by STG

Fig. 11: Evaluation of the refined THD implementation at *th* = 1.00 on DMKD-300 bags of terms. Vertical dashed lines mark terminological saturation point.

The summary of our experimental findings is collected in Table 7 in the form of the rankings. We rank the performance of all the evaluated SSMs and the baseline THD on a scale from 1 (the best) to 5 (the worst) for every document collection and every terms similarity threshold point (*Min*, *Ave*1, *Ave*2, *Max*) within each collection. We also look at the average rankings for all four thresholds points.

The aspects we look at in this ranking are: (i) the fastness of detecting terminological saturation, the faster – the better (Fig. 4 – 6); (ii) the number of retained significant terms, the more – the better (Fig. 10); and (iii) the time taken by the method to accomplish the computation, the less – the better (Fig. 7 – 9).

Table 8 contains the values of performance indices for different SSMs and the baseline THD regarding the four terms similarity thresholds points and their average values. This is done for two cases: (a) taking into account the execution time criterion (Less Time Taken in Table 7); and (b) not taking the execution time criterion into account. It has been done to analyse the value of using an SSM if the computational overhead is not important. The values were calculated by summing all the ranks for different collections and criteria taken from the corresponding threshold point rows of Table 7.

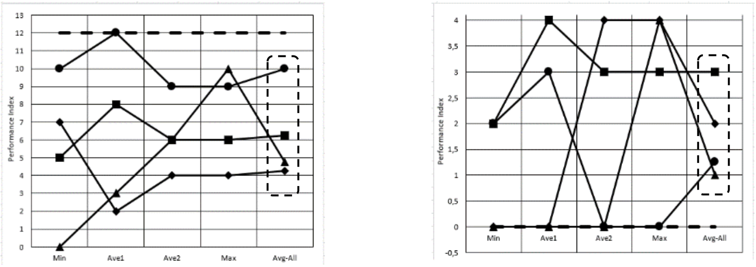
Table 7: The ranking of the evaluated SSMs

		Rank (1-5)				
Criterion	**String Similarity Threshold**	**Baseline THD**	**Sørensen-Dice**	**Jaccard**	**Jaro**	**Jaro-Winkler**
TIME Collection						
Faster detection of saturation	Min	5	4	1	1	1
	Ave1	5	4	1	1	1
	Ave2	5	4	1	1	1
	Max	5	3	1	3	1
	Average	5	3.75	1	1.5	1
More significant terms retained	Min	1	2	3	5	4
	Ave1	1	2	4	5	3
	Ave2	1	2	5	3	4
	Max	1	2	5	3	4
	Average	1	2	4.25	4	3.75
Less time taken	Min	1	5	3	2	4
	Ave1	1	4	2	3	5
	Ave2	1	3	2	5	4
	Max	1	3	2	5	4
	Average	1	3.75	2.25	3.75	4.25
DMKD-300 Collection						
Faster detection of saturation	Min	5	4	3	1	1
	Ave1	5	4	1	1	1
	Ave2	5	4	1	1	1
	Max	5	1	1	1	1
	Average	5	3.25	1.5	1	1
More significant terms retained	Min	1	2	3	5	4
	Ave1	1	2	4	5	3
	Ave2	1	2	5	3	4
	Max	1	2	5	3	4
	Average	1	2	4.25	4	3.75
Less time taken	Min	1	5	2	3	4
	Ave1	1	4	2	5	3
	Ave2	1	3	2	5	4
	Max	1	3	2	5	4
	Average	1	3.75	2	4.5	3.75
DAC-cleaned Collection						
Faster detection of saturation	Min	5	4	3	1	2
	Ave1	5	4	1	3	1
	Ave2	5	4	1	3	1
	Max	5	4	1	1	1
	Average	5	4	1.5	2	1.25
More significant terms retained	Min	1	2	3	5	4
	Ave1	1	2	4	3	5
	Ave2	1	2	5	3	4
	Max	1	2	5	3	4
	Average	1	2	4.25	3.5	4.25
Less time taken	Min	1	5	2	3	4
	Ave1	1	4	2	5	3
	Ave2	1	3	2	5	4
	Max	1	3	2	5	4
	Average	1	3.75	2	4.5	3.75

These sums have further been subtracted from the highest rank value[14] in order to revert to “the higher – the better” scale in Table 8. Performance indices are also pictured in Fig. 12.

Table 8: The performance indices of the evaluated SSMs with and without accounting for taken execution time

Threshold	Baseline THD	Sørensen-Dice	Jaccard	Jaro	Jaro-Winkler
(a) Execution time criterion is taken into account					
Min	12	0	10	7	5
Ave1	12	3	12	2	8
Ave2	12	6	9	4	6
Max	12	10	9	4	6
Average (Table 7)	12	4.75	10	4.25	6.25
(b) Execution time criterion is not taken into account					
Min	0	0	2	0	2
Ave1	0	0	3	0	4
Ave2	0	0	0	4	3
Max	0	4	0	4	3
Average (Table 7)	0	1	1.25	2	3



(a) Accounting for execution time (b) Without accounting for execution time

Legend: ▲ Sorensen-Dice ● Jaccard ◆ Jaro ■ Jaro-Winkler ‒ ‒ Baseline

Fig. 12: Performance indices of the evaluated SSMs per terms similarity thresholds with (a) and without (b) taking their execution time ranks into account. The points in the rounded dashed rectangles represent the averages for all the thresholds.

Regarding the evaluation of correctness at *th* = 1.00, the SSM, that behaves both correctly, up to the saturation point, and computationally efficiently, is Sørensen-Dice. Jaro and Jaro-Winkler, though are correct, take too much of the computational overhead at this high *th* value. Jaccard is not fully correct.

Let us now summarize the comparative analysis of the performance of all the SSMs in the two cases, (a) and (b), presented in Table 8 and Fig. 12.

[14] The rank value is the sum of all ranks for a method within a particular threshold in Table 7. The highest rank value indicates the lowest performance. For case (a) it equals to 12, which is for Sørensen-Dice at *Min* threshold. For case (b) it equals to 4.

In case (a), when the computation time is taken into account in the comparative rating, the summary is the following. Probably surprisingly, Jaccard, which is computationally the most lightweight SSM (Fig. 7–9), demonstrates the best overall performance. In this case, it still does not outperform the baseline THD because: (i) it takes extra time for STG; and (ii) it retains less significant terms[15]. Jaccard is the best balanced on all evaluation criteria, compared to the rest of the evaluated SSMs. One important drawback of Jaccard is that it does not perform fully correctly at *th* = 1.00. Therefore, the use of Jaccard may be recommended in the cases of low terms similarity thresholds (like *Min* or *Ave*1) and hard constraints on the time of computation. Performance indices are also good for Sørensen-Dice and Jaro-Winkler which both work acceptably correctly at *th* = 1.00. These two SSMs appear to be mutually complementary in the terms that: (i) Jaro-Winkler is better than Sørensen-Dice at lower terms similarity thresholds, like *Min* or *Ave*1; (ii) Sørensen-Dice outperforms Jaro-Winkler at higher terms similarity thresholds, like *Ave*2 or *Max*. Jaro in case (a) is a clear negative outlier and is not recommended for use.

In case (b), when the computation time is not taken into account in the comparative rating, the summary is different. As it is clearly seen in Fig 12(b), all the SSMs outperform the baseline THD on average and at *Max* threshold. Jaro-Winkler is the best performing for *Min* and *Ave*1 thresholds, but gives up to Jaro at *Ave*2 and *Max*. It is also outperformed by Sørensen-Dice at *Max*. However, Jaro-Winkler appears to be most balanced in performance regarding all the four thresholds, which is highlighted by the *Avg-All* value. Jaccard in case (b) is a clear negative outlier and therefore cannot be recommended for use.

If the assessments for the cases (a) and (b) are combined, the following recommendation could be given. At an expense of a substantially higher execution time, the THD algorithm refined by Jaro-Winkler (at all thresholds except *Max*) or Sørensen-Dice (at *Max* threshold) are our recommended choices for measuring terminological saturation. Jaro-Winkler is the first choice, because it is the most balanced in performance for all the evaluated thresholds.

6 Conclusions and Future Work

In this paper, we investigated if a simple string equivalence measure, used in the baseline THD algorithm, could be outperformed if a carefully chosen string similarity measure is used instead.

Overall, we found out that the use of STG, even at high terms similarity thresholds, rewards quite substantially in reducing the volume of processed data. The numbers of these gains are provided in the Terminological Core part of Table 9. Depending on how fast saturation is achievable in different collections, the use of STG allowed lowering the size of a terminological core by 22 to 46 percent.

[15] Which should be so as the baseline THD does not group terms. Hence, any alternative method that does similar terms grouping retains less significant terms.

It is also remarkable that, in general, the numbers of retained significant terms, due to their grouping, were also decreased substantially, by 44 to 72 percent depending on the collection. At the same time, the individual term significance thresholds (*eps*) were very slightly changed. This hints that the use of STG did not result in a noticeable loss of significant terms.

Because of applying our THD algorithm refinement, using all four evaluated SSMs, terminological saturation has been detected faster. Hence, in that sense, the refined THD with STG outperformed the baseline method. Three of the SSMs gave also acceptably correct results at *th* = 1.00. A somewhat discouraging result was, however, that the use of SSMs for STG causes a substantial computational overhead. Therefore, none of the methods involving STG outperformed the baseline THD integrally if execution time is an important criterion for assessing performance – case (a) in Table 8 and Fig.12. If execution time is not very important and may be disregarded, the result is substantially different – case (b) in Table 8 and Fig.12. Overall, putting together the findings in these two cases, the recommendation was made to use the THD algorithm refined by Jaro-Winkler (at all thresholds except *Max*) or Sørensen-Dice (at *Max* threshold) for measuring terminological saturation. Jaro-Winkler was recommended as the first choice, because it is the most balanced in performance for all the evaluated thresholds.

Table 9: The gains of the use of STG and refined THD

	Saturation Point	**Terminological Core**			**Terms**			
		No Papers	**Volume, Mb**	**% Baseline**	**Extracted Terms**	**Retained Terms**	**% Baseline**	***eps***
	TIME (Max)							
Baseline	D11	220	6.55	**100.00**	287887	7110	**100.00**	23.77
Jaccard	D6	120	3.53	**53.89**	190263	2717	**38.21**	21.00
Sorensen-Dice	D7	140	4.17	**63.66**	200176	3629	**51.04**	22.00
Jaro-Winkler	D6	120	3.53	**53.89**	190263	2717	**38.21**	21.00
	DMKD-300 (Max)							
Baseline	D3	60	3.14	**100.00**	89617	7110	**100.00**	17.00
Jaccard	D2	45	2.46	**78.34**	67913	2135	**30.03**	15.50
Sorensen-Dice	D2	45	2.46	**78.34**	67913	2453	**34.50**	15.50
Jaro-Winkler	D2	45	2.46	**78.34**	67913	1963	**27.61**	15.50
	DAC-cleaned (Max)							
Baseline	D23	460	12.40	**100.00**	514364	20558	**100.00**	15.51
Jaccard	D14	280	7.46	**60.16**	320473	7406	**36.02**	15.51
Sorensen-Dice	D16	320	8.54	**68.87**	356749	11528	**56.08**	15.51
Jaro-Winkler	D14	280	7.46	**60.16**	320473	8736	**42.49**	15.51

The plans for our future work are implied by the presented results. Firstly, we would like to admit that the test set of term pairs (Table 2) is not big enough to consider the choice of the thresholds fully reliable. Therefore, we will extend the test set in short term and apply a variation of a clustering technique to check our thresholds. Secondly, we would like to explore the ways to improve the performance of the Sørensen-Dice and Jaro-Winkler measures implementations, as their high computational complexity is the only obstacle to outperform the rest of the evaluated SSMs and, possibly, the baseline. To put it more generally, we plan to explore the ways to improve the performance of similar terms grouping, as the times taken by the STG algorithm are too long. Thirdly, we are interested in finding out if a similar terms grouping algorithm, using Sørensen-Dice or Jaro-Winkler, would be plausible for grouping features while building feature taxonomies. This task is on the agenda for the second (Conceptualization) phase of OntoElect [3, 39].

Acknowledgements

The research leading to this publication has been performed in part in cooperation between the Department of Computer Science of Zaporizhzhia National University, the Ontology Engineering Group of the Universidad Politécnica de Madrid, the Applied Probability and Informatics Department at the RUDN University, and Springer-Verlag GmbH. The first author is funded by a PhD grant awarded by Zaporizhzhia National University and the Ministry of Education and Science of Ukraine. The second author is supported by the FPI grant (BES-2017-082511) under the DATOS 4.0: RETOS Y SOLUCIONES - UPM project (TIN2016-78011-C4-4-R) funded by Ministerio de Economía, Industria y Competitividad of Spanish government and EU FEDER funds. The fourth author acknowledges the support of the "RUDN University Program 5-100". The authors would like to acknowledge the contributions by Alyona Chugunenko and Rodion Popov for their research contributions leading to this publication. In particular, they helped develop the approach for term grouping and implement the software for it. The collection of full text Springer journal papers dealing with Knowledge Management, including DMKD-300, has been provided by Springer-Verlag. The authors would also like to express their gratitude to anonymous reviewers whose comments and suggestions helped improve the paper.

References

1. 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, online
2. 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)
3. Ermolayev, V.: OntoElecting requirements for domain ontologies. The case of time domain. EMISA Int J of Conceptual Modeling 13(Sp. Issue), pp. 86--109 (2018)
4. 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)
5. 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)

6. 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)
7. 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)
8. 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)
9. Astrakhantsev, N.: ATR4S: toolkit with state-of-the-art automatic terms recognition methods in scala. arXiv preprint arXiv:1611.07804 (2016)
10. Medelyan, O., Witten, I. H.: Thesaurus based automatic keyphrase indexing. In: Marchionini, G., Nelson, M. L., Marshall, C. C. (eds.) Proc. ACM/IEEE Joint Conf on Digital Libraries, JCDL 2006, pp. 296--297, Chapel Hill, NC, USA, ACM (2006)
11. Ahmad, K., Gillam, L., Tostevin, L.: University of surrey participation in trec8: Weirdness indexing for logical document extrapolation and retrieval (wilder). In: Proc. 8th Text REtrieval Conf, TREC-8 (1999)
12. Sclano, F., Velardi, P.: TermExtractor: A Web application to learn the common terminology of interest groups and research communities. In: Proc. 9th Conf on Terminology and Artificial Intelligence, TIA 2007, Sophia Antipolis, France (2007)
13. Frantzi, K. T., Ananiadou, S.: The c/nc value domain independent method for multi-word term extraction. J. Nat. Lang. Proc. 6(3), pp. 145--180 (1999)
14. Kozakov, L., Park, Y., Fin, T., Drissi, Y., Doganata, Y., Cofino, T.: Glossary extraction and utilization in the information search and delivery system for IBM Technical Support. IBM System Journal 43(3), pp. 546--563 (2004)
15. Astrakhantsev, N.: Methods and software for terminology extraction from domain-specific text collection. PhD thesis, Institute for System Programming of Russian Academy of Sciences (2015)
16. Bordea, G., Buitelaar, P., Polajnar, T.: Domain-independent term extraction through domain modelling. In: Proc. 10th Int Conf on Terminology and Artificial Intelligence, TIA 2013, Paris, France (2013)
17. Badenes-Olmedo, C., Redondo-García, J. L., Corcho, O.: Efficient clustering from distributions over topics. In*: Proc.* K-CAP 2017, ACM, New York, NY, USA, Article 17, 8 p. (2017)
18. Gomaa, W. H., Fahmy. A. A.: A Survey of Text Similarity Approaches. Int J Comp Appl 68(13), pp. 13--18 (2013)
19. Yu, M., Li, G., Deng, D., Feng, J.: String similarity search and join: a survey. Front. Comput. Sci. 10(3), pp. 399--417 (2016)
20. Miller, G.A., Beckwith, R., Fellbaum, C.D., Gross, D., Miller, K.: WordNet: An online lexical database. Int. J. Lexicograph. 3(4), pp. 235--244 (1990)
21. Arnold, M., Ohlebusch, E.: Linear Time Algorithms for Generalizations of the Longest Common Substring Problem. Algorithmica 60(4), pp. 806--818 (2011)
22. *Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10 (8), pp. 707--710 (1966)*
23. Hamming, R. W.: Error detecting and error correcting codes. Bell System Technical Journal 29(2), pp. 147--160 (1950)
24. Monger, A., Elkan, C.: The field-matching problem: algorithm and applications. In: Proc. 2nd Int Conf on Knowledge Discovery and Data Mining, pp. 267--270, AAAI Press (1996)

25. Jaro, M. A.: Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida. J Amer Stat Assoc 84(406), pp. 414--420 (1989)
26. Winkler, W. E.: String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage. In: Proc. Section on Survey Research Methods. ASA, pp. 354--359 (1990)
27. Dice, L. R.: Measures of the amount of ecologic association between species. Ecology 26(3), pp. 297--302 (1945)
28. Sørensen, T.: A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons. Kongelige Danske Videnskabernes Selskab 5 (4), pp. 1--34 (1948)
29. Jaccard, P.: The distribution of the flora in the alpine zone. New Phytologist 11, 37--50 (1912)
30. Huang, A.: Similarity Measures for Text Document Clustering. In: Proc. 6th New Zealand Computer Science Research Student Conference (NZCSRSC2008), Christchurch, New Zealand, pp. 49--56 (2008)
31. Singhal, A.: Modern Information Retrieval: A Brief Overview. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24 (4), pp. 35--43 (2001)
32. Lu, J., Lin, C., Wang, W., Li, C., Wang, H.: String similarity measures and joins with synonyms. In: Proc. 2013 ACM SIGMOD Int Conf on the Management of Data, pp. 373--384 (2013)
33. Lee, H., Ng, R. T., Shim, K.: Power-law based estimation of set similarity join size. Proc. of the VLDB Endowment 2(1), pp. 658--669 (2009)
34. Tsuruoka, Y., McNaught, J., Tsujii, J., Ananiadou, S.: Learning string similarity measures for gene/protein name dictionary look-up using logistic regression. Bioinformatics 23(20), pp. 2768--2774 (2007)
35. Qin, J., Wang, W., Lu, Y., Xiao, C., Lin, X.: Efficient exact edit similarity query processing with the asymmetric signature scheme. In: Proc. of the 2011 ACM SIGMOD Int Conf on Management of data, pp. 1033--1044. ACM New York, USA (2011)
36. Corcho, O., Gonzalez, R., Badenes, C., Dong, F.: Repository of indexed ROs. Deliverable No. 5.4. Dr Inventor project (2015)
37. Kosa, V., Chaves-Fraga, D., Naumenko, D., Yuschenko, E., Badenes-Olmedo, C., Ermolayev, V., Birukou, A.: Cross-evaluation of automated term extraction tools by measuring terminological saturation. In: Bassiliades, N., et al. (eds.) ICTERI 2017. Revised Selected Papers. CCIS, vol. 826, pp. 135--163 (2018)
38. Minkowski, H.: Geometrie der Zahlen. Bibliotheca Mathematica Teubneriana, Band 40 Johnson Reprint Corp., New York-London, 256 pp. (1968) – in German
39. Moiseenko, S., Ermolayev, V.: Conceptualizing and formalizing requirements for ontology engineering. In: Antoniou, G., Zholtkevych, G. (eds.) Proc. ICTERI 2018 PhD Symposium, Kyiv, Ukraine, May 14-17, CEUR-WS **vol. 2122**, pp. 35--44 (2018) online