Singapore Management University

Institutional Knowledge at Singapore Management University

Research Collection School Of Computing and Information Systems

School of Computing and Information Systems

2-2007

Mining generalized associations of semantic relations from textual web content

Tao JIANG

Ah-hwee TAN
Singapore Management University, ahtan@smu.edu.sg

We WANG

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research

Part of the Computer Engineering Commons, and the Databases and Information Systems Commons

Citation

JIANG, Tao; TAN, Ah-hwee; and WANG, We. Mining generalized associations of semantic relations from textual web content. (2007). *IEEE Transactions on Knowledge and Data Engineering*. 19, (2), 164-179.
Available at: https://ink.library.smu.edu.sg/sis_research/5228

This Journal Article is brought to you for free and open access by the School of Computing and Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Computing and Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email cherylds@smu.edu.sg.

Mining Generalized Associations of Semantic Relations from Textual Web Content

Tao Jiang, Ah-Hwee Tan, and Ke Wang

Abstract—Traditional text mining techniques transform free text into flat bags of words representation, which does not preserve sufficient semantics for the purpose of knowledge discovery. In this paper, we present a two-step procedure to mine generalized associations of semantic relations conveyed by the textual content of Web documents. First, RDF (Resource Description Framework) metadata representing semantic relations are extracted from raw text using a myriad of natural language processing techniques. The relation extraction process also creates a term taxonomy in the form of a sense hierarchy inferred from WordNet. Then, a novel generalized association pattern mining algorithm (*GP-Close*) is applied to discover the underlying relation association patterns on RDF metadata. For pruning the large number of redundant *overgeneralized patterns* in relation pattern search space, the GP-Close algorithm adopts the notion of *generalization closure* for systematic overgeneralization reduction. The efficacy of our approach is demonstrated through empirical experiments conducted on an online database of terrorist activities.

Index Terms—RDF mining, association rule mining, relation association, text mining.

1 Introduction

With the explosive growth of the World Wide Web, we face an increasing amount of information resources, of which most are represented in free text. As text data are inherently unstructured and difficult to directly process by computer programs, there has been great interest in text mining techniques [1] for helping users to quickly gain knowledge from the Web. Text mining technologies usually involve two subtasks [2]: *text refining*, which transforms free text into an intermediate representation form which is machine-processable, and *knowledge distillation*, which deduces patterns or knowledge from the intermediate form.

Existing techniques mainly transform text documents into simplistic intermediate forms, e.g., term vectors and bags of keywords. As terms are treated as individual items in such simplistic representations, terms lose their semantic relations and texts lose their original meanings. For example, in Fig. 1, two short text documents with different meanings can be represented in a similar bag of keywords, e.g., {France, Defeat, Italy, World Cup, Quarter Final}. In the original documents, "France" and "Italy" have different roles in the events of "Defeat." However, the semantic relations depicting the conceptual roles are lost in the bag-of-keyword representations. Therefore, the original meanings of the documents in Fig. 1 cannot be discriminated against any more. Based on such simplistic representations of text, text mining techniques can only discover shallow patterns, such as term associations, deviations, and document clusters, which are statistical patterns of terms, not knowledge about text semantics. In this paper, we aim to overcome the limitation of text mining technologies to discover knowledge based on the detailed meanings of the text. For this purpose, we need an intermediate representation that expresses the semantic relations between the concepts in texts.

Resources Description Framework (*RDF*), proposed by the World Wide Web Consortium (W3C), is a language specification for modeling machine-processable and human-readable semantic metadata to describe Web resources on the Semantic Web [3]. The basic element of RDF is RDF statements, which are triplets in the form of <subject, predicate, object>. An RDF statement can express that there is a relation (represented by the predicate) between the subject and the object. In [4], Berners-Lee further illustrated that RDF can be interworkable with *conceptual graphs*, [5], [6], which serve as an intermediate language for translating natural languages into computer-oriented formalisms. As the full conceptual graph standard is complex for large-scale applications, simplified conceptual graphs are used in many existing practices, such as [7]. In our work, we also use a set of simplified conceptual graphs containing only three kinds of predicates, i.e., "agent," "theme," and "modifiedBy," for representing the key semantics of text. As an example, Fig. 2 shows such a simplified conceptual graph translated from the first sentence in Fig. 1. We treat each directed arc in the conceptual graph as a semantic relation consisting of a subject (the start node of the arc), a predicate (the label or type of the arc), and an object (the end node of the arc). Each of these relations can be encoded using an RDF statement. For example, the three semantic relations in the conceptual graph in Fig. 2 can be represented using the RDF statements <Defeat, agent, France>, <Defeat, theme, Italy>, and <Defeat, modifiedBy, World Cup Quarter Final>. More details on the semantics of the three predicates will be introduced in Section 3.

- *T. Jiang and A.-H. Tan are with the School of Computer Engineering, Nanyang Technological University, Block N4-B3b-06, 50 Nanyang Avenue, Singapore 639798. E-mail: {jian0006, asahtan}@ntu.edu.sg.*
- *K. Wang is with the Department of Computing Science, Simon Fraser University, 8888 University Drive, Burnaby, British Columbia, Canada V5A 1S6. E-mail: wang@cs.sfu.ca.*

Manuscript received 30 Sept. 2005; revised 14 Apr. 2006; accepted 2 Aug. 2006; published online 19 Dec. 2006.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDESI-0429-0905.

1. France defeated Italy in the World Cup quarter final.
2. France was defeated by Italy in the World Cup quarter final.

Fig. 1. Two short text documents with similar keywords but of distinct semantic meanings.

Using RDF as the intermediate representation, our proposed knowledge discovery process consists of two stages (Fig. 3). During the semantic relation extraction stage, text documents are processed using a myriad of natural language processing (NLP) techniques, including pronominal coreference resolution, part-of-speech (POS) tagging, and sentence structure parsing. A set of predefined syntactic patterns is then used to extract semantic relations (composing the conceptual graphs) from the tagged text sentences. The extracted relations are encoded in RDF statements. In addition, a term taxonomy is constructed on the fly based on WordNet and domain-specific lexicons. The term taxonomy, described using RDF Schema [8] (a vocabulary specification for RDF), is, in turn, used in the subsequent stage.

During the association rule mining stage, relation association patterns are discovered from RDF metadata extracted from text. A problem for mining semantic relations is that a relation is seldom repeated in many documents. Therefore, statistically significant patterns can hardly be extracted. To overcome this limitation, generalizations of the semantic relations are needed. Most existing generalized pattern mining algorithms, such as Cumulate [9], are designed for mining patterns on atomic items, not relations. Therefore, the existing methods cannot be directly applied to mining generalized relation patterns from RDF metadata. Even if we treat each relation as an item, the existing algorithms do not work efficiently on the RDF data. We observe that generalizing a semantic relation is complex as the relation can be generalized in many different ways. For example, <Defeat, agent, France> can be generalized by generalizing "Defeat" into "Is Unbeaten," generalizing "France" into "European Football Team," or their combination. This implies that the generalized pattern search space can be very large and there can be many redundant overgeneralized patterns [10]. Using the existing generalized pattern mining approaches, such as the Cumulate algorithm [9], all generalized patterns including the overgeneralized ones will be extracted. It is not only computationally inefficient but also causes much redundancy in the mining results. Therefore, pruning the generalized pattern search space and reducing the overgeneralized patterns are the new challenges for mining patterns of semantic relations

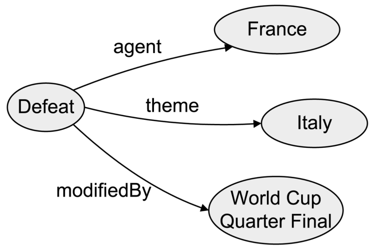


Fig. 2. A simplified conceptual graph translated from the first sentence in

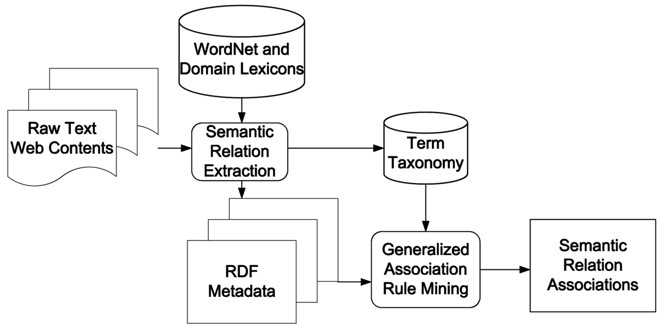


Fig. 3. Overview of the knowledge discovery process.

from RDF metadata. In this paper, we present a novel generalized association pattern mining algorithm, called GP-Close (Closed Generalized Pattern Mining) for discovering generalized association patterns of semantic relations from RDF data. The proposed algorithm with efficient *pattern space pruning* and full *overgeneralization reduction* can be applied to any RDF database with the existence of an RDF vocabulary.

The rest of the paper is organized as follows: Section 2 introduces the related work. Section 3 presents the semantic relation extraction procedure. Section 4 formulates the problem statement for generalized association pattern mining of RDF metadata and presents the GP-Close algorithm. Section 5 reports our experiments based on a document set in the terrorist domain downloaded from the online database of International Policy Institute for Counter-Terrorism (ICT). Concluding remarks are given in the final section.

2 Related Work

2.1 Resource Description Framework and RDF Schema

Resource Description Framework (RDF) [11] is a specification proposed by the World Wide Web Consortium (W3C) for describing and interchanging semantic metadata. The basic element of RDF is RDF statements, each consisting of a subject, a predicate, and an object. In this paper, a triplet of the form <subject, predicate, object> is used to express an RDF statement. At the semantic level, an RDF statement can be interpreted as "the subject has an attribute (represented by the predicate) whose value is given by the object" or "the subject has a relation (represented by the predicate) with the object." RDF is mainly a language specification addressing syntactical aspects. Based on RDF, RDF Schema [8] (RDFS) is further proposed to define RDF vocabularies for constructing RDF statements. In this study, we use RDF and RDFS to encode concepts and semantic relations which are extracted from textual Web content.

2.2 Semantic Modeling and Computing on the Semantic Web

In the Semantic Web area, semantic modeling and computing are two essential issues that determine how data semantics can be helpful for building a more meaningful and intelligent Web. A lot of recent research efforts focus on these two problems. In [12], an infrastructure for supporting human-centric Semantic Web is presented. In such a Human Semantic Web, relation-oriented conceptual modeling plays an important role. In [13], a vision of developing *powerful semantic* techniques which combine knowledge composition and statistical analysis for scalable and flexible applications is presented.

Besides the principles for advanced semantic modeling, new methods are also proposed for advanced computing and reasoning based on the existing semantic modeling technologies, such as RDF. Liu et al. [14] propose a method for discovering taste fabric from the Web social networks. In [15], a rule-based system, called DR-DEVICE, for defeasible reasoning over RDF metadata is presented. In this system, the incomplete and inconsistent information can be efficiently handled.

We can see that there is a trend in the Semantic Web literature, which is to leverage loose and lightweight computing and reasoning techniques, instead of formal reasoning, for enabling various flexible applications. Our method for mining semantic relation association patterns is sympathetic to this trend. In addition, besides knowledge discovery, our proposed techniques can also be used for supporting other applications in the Semantic Web, such as processing versatile Web queries [16] for retrieving information from both the conventional Web and the Semantic Web.

2.3 Association Rule Mining and Frequent Pattern Mining

Association Rule Mining (ARM) [17] since its introduction has become one of the key data mining techniques in the field of Knowledge Discovery in Database (KDD). Given a set of items $\mathcal{I}$ and a large database of transactions $\mathcal{D}$, where each transaction is a set of items $T \subseteq \mathcal{I}$ with a unique identifier *tid*, an association rule is an implication of the form $X \Rightarrow Y$, where $X, Y \subseteq \mathcal{I}$ (called itemsets or patterns) and $X \cap Y = \emptyset$. A transaction T supports an itemset X if $X \subseteq T$. The *support* of an itemset X, denoted by $supp(X)$, is the fraction of transactions in $\mathcal{D}$ that support X. Additionally, the *support* of an association rule $X \Rightarrow Y$, denoted by $supp(X \Rightarrow Y)$, is defined as $supp(X \cup Y)$. The *confidence* of $X \Rightarrow Y$, denoted by $conf(X \Rightarrow Y)$, is defined as $supp(X \cup Y)/supp(X)$. The problem of association rule mining is to discover all rules that have supports and confidences greater than some predefined minimum support (*minsup*) and minimum confidence (*minconf*). Mining association rules consists of two subtasks. The first task, known as *frequent itemset mining* (or *frequent pattern mining*), generates all itemsets that have supports higher than a minimum support (*minsup*) threshold. In the second task, association rules are generated based on the discovered frequent patterns. The rule generation methods are relatively straightforward and have been discussed extensively in [18]. Therefore, most recent efforts focus on frequent pattern mining, such as [19], [20], [21], [22], [23]. In this paper, we focus on mining frequent generalized association patterns.

For discovering associations between items across different levels of a taxonomy, generalized association rule mining is proposed [9]. In particular, users may require the generated rules to have a large support to avoid trivial knowledge being discovered. In this case, a large portion of the rules that include only the leaves of the item taxonomy may be filtered away. However, useful knowledge may still be found by generalizing the elemental patterns to an abstract level. For example, a rule like "$Outwear \Rightarrow HikingBoot$" may be extracted from the fact that the people usually bought Jackets and Ski Pants with Hiking Boots, while the specialized rules "$Jacket \Rightarrow HikingBoot$" and "$SkiPant \Rightarrow HikingBoot$" may not be extracted due to their low supports. Therefore, generalized association rule mining allows users to extract a small set of useful rules instead of generating a large set of trivial ones. This property is especially important for mining associations from large text data sets, in which the support of most items (words) is very low.

2.4 Association Rule Mining in Text Databases

Mining association rules between words in text documents has been done in [24] and [25]. These efforts have shown that text databases cannot be efficiently analyzed by standard association mining algorithms. This is because the characteristics of text databases are quite different from those of relational and transactional databases. First, the number of distinct words in a text database is usually quite large (large size of $\mathcal{I}$) and so is the number of words in each document (long transactions). The large number of words implies a large number of possible patterns (sets of words) both in a document collection and in each individual document. Thus, a text AR mining algorithm needs to explore a much larger search space to find interesting patterns. Moreover, the document frequency of each word is usually very low. For example, 68.8 percent of the 47,189 words occurs in only three or less of 3,568 articles in a sample drawn from the 1996 TReC data collection [25]. Low *minsup* was thus used in the existing work for mining text association rules. However, this will cause a large set of trivial patterns discovered. The work presented in [24] and [25] aims to mine shallow word patterns on text and is fundamentally different from our task of mining relation associations on RDFs.

For discovering more detailed knowledge from text, y Gómez et al. [26] presented a method for extracting knowledge from text based on *conceptual graphs*, [5], [6]. At the knowledge discovery stage, the conceptual graphs are first clustered into a hierarchy. Then, pattern mining techniques, such as association rule mining, can be applied to this hierarchical structure. This method shows that meaningful and detailed patterns can be discovered from text using the conceptual graph representation, which is in spirit similar to our method. However, y Gómez et al.'s method requires the conceptual graphs to be clustered before the mining process. This is not only costly initially, but also involves many efforts for maintenance and update of the clustered structure, in particular, for mining dynamic Web content.

3 Semantic Relation Extraction

In this section, we introduce the procedure for semantic relation extraction in details. As shown in Fig. 4, the raw textual Web content is sequentially preprocessed by a pronominal coreference resolution module and a POS (part-of-speech) tagging and syntax parsing module. For eliminating the ambiguities of the pronouns in text, we employ

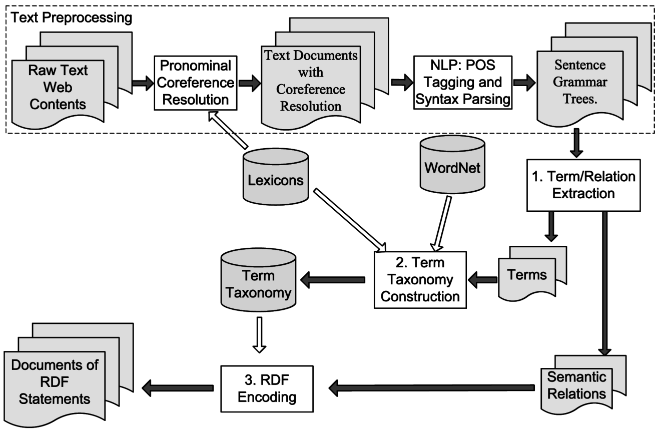

Text Preprocessing
Raw Text Web Contents
Pronominal Coreference Resolution
Text Documents with Coreference Resolution
NLP: POS Tagging and Syntax Parsing
Sentence Grammar Trees.
Lexicons
WordNet
1. Term/Relation Extraction
2. Term Taxonomy Construction
Terms
Term Taxonomy
Documents of RDF Statements
3. RDF Encoding
Semantic Relations

Fig. 4. An overview of the procedure for semantic relation extraction.

the pronominal coreference resolution function of Gate [27], an NLP toolkit developed by the University of Sheffield. In addition, domain-specific name lexicons are embedded in Gate for identifying name entities (NE) in text. After coreference resolution, we replace each resolved pronoun with the origin term that it refers to. The text documents are then tagged and parsed by two NLP tools, namely, Brill's rule-based part-of-speech (POS) tagger [28] and Collins' parser [29]. After preprocessing, each parsed document contains a set of sentence grammar trees. Based on the sentence grammar trees, simplified conceptual graphs containing semantic relations are extracted and encoded by the following three modules.

1. **Term and Relation Extraction**. Based on the preprocessing results, a set of predefined rules adopted from [30] is used for extracting semantic relations from the sentence grammar trees. When extracting semantic relations, we first identify the important terms describing the major concepts, i.e., noun phrases (NP) and verb phrases (VP), in the grammar trees, followed by three major types of relations between these terms. The three relation types are introduced in Section 3.1.
2. **Term Taxonomy Construction**. The terms (NP/VP) extracted from the sentence grammar trees are incrementally clustered into a term taxonomy with the assistance of WordNet [31]. The atomic clusters in the term taxonomy are groups of synonyms. When a new term is inserted into the term taxonomy, we first try to find whether there is a synonym group it can join. If there is such a synonym group, we simply insert the new term into the synonym group and the structure of term taxonomy will not change; otherwise, it will be inserted as a new synonym group and the structure of the term taxonomy will change.
3. **RDF Encoding**. The term taxonomy and the semantic relations extracted from the sentence grammar trees are encoded as an RDF vocabulary and RDF statements, respectively.

In the following sections, we describe the three modules in detail.

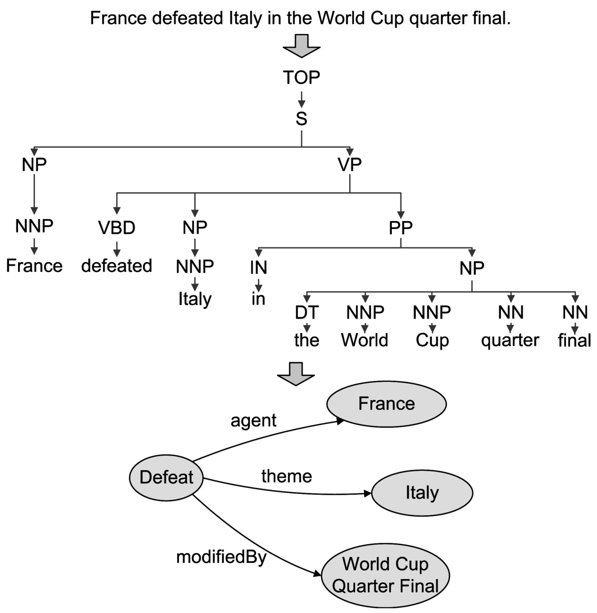


Fig. 5. Conversion of text to semantic relations shown in the form of a conceptual graph.

3.1 Extraction of Terms and Relations from Sentence Grammar Trees

As noun phrases (NP) and verb phrases (VP) convey the main meaning of text, we first extract those terms of NP or VP from the sentence grammar trees. Then, three types of relations between those terms (NP/VP) are identified based on their syntactic dependencies using a set of rules [30]. Similarly to Sowa's conceptual graphs [5], [6] as used in many knowledge-based systems [7], [26], the definitions of the three types of relations are given below:

- $< A, \text{agent}, B >$, where A can be a VP and B can be an NP/VP. The relation indicates that B is the *agent* that performs the *action*A.
- $< A, \text{theme}, B >$, where A can be a VP and B can be an NP/VP. The relation indicates that B is the *theme* (i.e., *recipient*, *object*, or *target*) of the action A.
- $< A, \text{modifiedBy}, B >$, where A can be an NP/VP and B can be an NP/VP. This relation indicates that A is modified by B through a proposition.

Fig. 5 shows an example. A sentence is first parsed into a grammar tree structure. We then extract three relations, i.e., <Defeat, agent, France>, <Defeat, theme, Italy>, and <Defeat, modifiedBy, World Cup Quarter Final>, based on the obtained grammar tree. These three relations form a simplified conceptual graph as in Fig. 5.

We note that other forms of expressions can be used for modeling relations. For example, from the sample grammar tree in Fig. 5, we can also extract a relation <France, defeat, Italy>, where the action, its agent, and its theme are combined into one relation. However, we observe that, in many sentences (in particular those in passive voice), the agents or themes of an action may be missing. For example, the sentence "France was defeated" does not contain the agent of the action. In this case, we cannot extract a full relation containing action, agent, and theme. However, using the three relation types that we adopt, useful relations

TABLE 1
A Set of Sample Terms Represented by Bags of WordNet Senses

Order	Terms	Bags of Senses
1	Midfield Player	$S_{Midfield\_Player}$ = {08451381_midfield, 08405214_center, 10283858_player, 09476765_contestant, 10260253_performer, ... , 10246540_participant}
2	Player	S_{Player} ={10283858_player, 09476765_contestant, 10260253_performer, ... , 10246540_participant}
3	Attack Player	S_{Attack_Player} = {00453042_attack, 00452701_play, 10283858_player, 09476765_contestant, 10260253_performer, ... , 10246540_participant}
4	Winner	S_{Winner} = {09619712_achiever, 09476765_contestant, 10621652_winner, 09968260_gambler, 10621801_victor, 00007626_person}

can still be extracted, even though the agents or themes of the actions are missing in the sentences.

3.2 Term Taxonomy Construction

Our purpose in building a term taxonomy is to hierarchically group similar terms into meaningful clusters. Based on such clusters, semantic relations that consist of similar terms can be generalized for deducing statistically significant patterns during the knowledge mining stage.

3.2.1 Existing Work in Term Taxonomy Construction

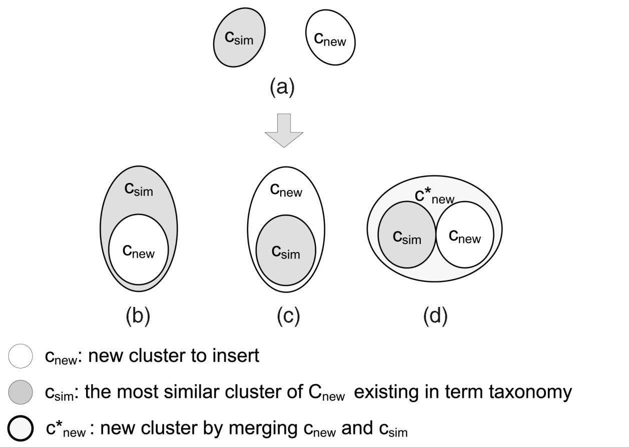
Recently, there has been an increasing amount of attention on automatic taxonomy construction in the field of ontology engineering [32], [33], [34], [35]. Existing methods for taxonomy construction mainly fall into two categories, symbolic approaches and statistics-based methods, and both have their limitations. Symbolic approaches, which directly find taxonomic relations from text using lexico-syntactic patterns [32], can hardly exhaustively extract all possible taxonomic relations, especially when some commonly known domain-specific information, such as "goalkeeper is a kind of soccer player," does not explicitly exist in the text documents. Statistics-based methods usually employ bottom-up (agglomerative) or top-down (divisive) hierarchical clustering methods to build term hierarchies based on statistical context features of terms (such as frequencies of surrounding terms) [35], [33]. The disadvantages of these methods include the poor traceability of the taxonomy construction process and the difficulty in labeling nonleaf nodes (inner clusters) of the taxonomies. The extracted taxonomies are thus difficult for human users to understand. Furthermore, both symbolic and statistics-based approaches require a large domain-specific text corpus, which is usually unavailable, for taxonomy construction. In addition, the costs of computation and update of taxonomies may be very large in such a corpus. Besides the symbolic and statistics-based methods, a more recent work presented in [36] focuses on deriving produces and services ontologies, including concept taxonomies, based on the existing industrial categorization standards. The work itself is interesting. However, as well-defined categorization standards do not widely exist, the reusability of the propose method is limited. In our opinion, most existing techniques for taxonomy construction are more suitable to bootstrap an ontology acquisition process but have limited usages in data mining tasks such as the one presented in this paper. In Section 3.2.3, we introduce a lightweight incremental clustering strategy for taxonomy construction. Instead of using a large text corpus, it utilizes the word sense hierarchies in WordNet [31] as the basis for building the term taxonomy. The constructed taxonomy is thus more understandable for human users. In addition, because it constructs the taxonomy in an incremental manner, the computation and update costs are minimal.

3.2.2 Term Representation and Similarity Measure Selection

A term (VP/NP) extracted from text is represented as a bag of senses in WordNet [31] in the form of $S = \{s_1, s_2, \ldots, s_n\}$, where each sense represents a meaning of a word and corresponds to a set of synonyms in WordNet. For each word in a VP/NP, we add all its WordNet senses into the bag representation. For each sense added in the bag, we recursively add their *hypernyms* and *derivationally related senses* into the bag. However, adding all senses of a term and their related senses (hypernyms and derivationally related senses) can generate a very large bag which will slow down the process of term taxonomy construction. Therefore, we impose a restriction on the WordNet search depth ($WNSD$) when building the bag of senses for a term. A set of sample terms represented by bags of senses is listed in Table 1. A WordNet sense (e.g., *10283858_player*) is expressed using its ID (e.g., *10283858*) in WordNet conjuncted with its representative word (e.g., *player*). If two terms have the same set of senses, they are called *synonyms*. The extracted NP/VPs thus can be classified into groups of synonyms.

For clustering the terms extracted from the text, we need a method to measure the semantic similarity between the terms. There have been many term similarity measures proposed in the existing literatures, such as, Jiang and Conrath's measure [37], Leacock and Chodorow's measure [38], and Seco et al.'s measure [39]. Some measures uniquely rely on the topology information in concept taxonomies such as WordNet [38], [39], while some others use both concept taxonomy and large text corpus for combining the topology information and word statistics [37]. Evaluations and comparisons of various term similarity measure are presented in [40] and [39]. However, the above-mentioned similarity measures are seldom used in the existing work of term taxonomy construction. One reason can be that these similarity measures (in particular, those relying on the word statistics) are computationally intensive. Another reason may be that these measures usually do not take multiword terms (e.g., "Attack Player" in Table 1) into account. For handling multiword terms, an additional strategy must be used.

As we represent each term as a bag of senses, we calculate the term similarities based on the number of



c_sim
c_new
(a)
c_sim
c_new
c_new
c_sim
c*_new
c_sim
c_new
(b)
(c)
(d)
c_new: new cluster to insert
c_sim: the most similar cluster of C_new existing in term taxonomy
c*_new : new cluster by merging c_new and c_sim

Fig. 6. Three cases for merging the new cluster and its most similar cluster.

senses shared by two terms. However, our similarity measure is intrinsically equivalent to the ***cosine similarity measure*** which is widely used and has achieved satisfactory results in many term taxonomy construction tasks [35], [33]. The similarity measure is defined as

$$sim(t_1, t_2) = \frac{|S(t_1) \cap S(t_2)|}{\sqrt{|S(t_1)| \cdot |S(t_2)|}}, \qquad (1)$$

where t_i denotes a term and $S(t_i)$ denotes its corresponding bag of senses ($i = 1$ or 2). This measure is equivalent to the cosine similarity measure if we convert the bags of senses into sense vectors where each sense is a feature dimension and the feature values of the senses are set to 1 or 0, depending on whether a sense is present in a bag of senses.

3.2.3 Incremental Term Taxonomy Construction

In our work, the term taxonomy is dynamically built on the fly using an incremental hierarchical clustering strategy. Here, we treat each group of synonyms as an atomic cluster. Several clusters can be merged to form a larger cluster, which is treated as the parent (supercluster) of the merged clusters. A cluster is also represented as a bag of WordNet senses, containing the senses shared by all terms in this cluster. We can thus use (1) to measure the similarity between two clusters. The root of the term taxonomy corresponds to a cluster containing all terms. The bag of senses associated with the *root cluster* is set to $\emptyset$.

When a new term (NP/VP) is extracted, we first try to find an existing atomic cluster (composed of its synonyms) to which it can be directly assigned. If there is no such atomic cluster, we create one for the new term and add it into the term taxonomy using our incremental hierarchical clustering strategy. The clustering process can be summarized in the following steps:

- Step 1: We first find a cluster in the term taxonomy that is most similar to the new term.[1] If no similar cluster can be found (i.e., there is no existing cluster that has nonzero similarity with the new term), add the new term as a subcluster of the root cluster and the process is completed.

1. For simplicity, we use "new term" here to represent the atomic cluster created for the new term.

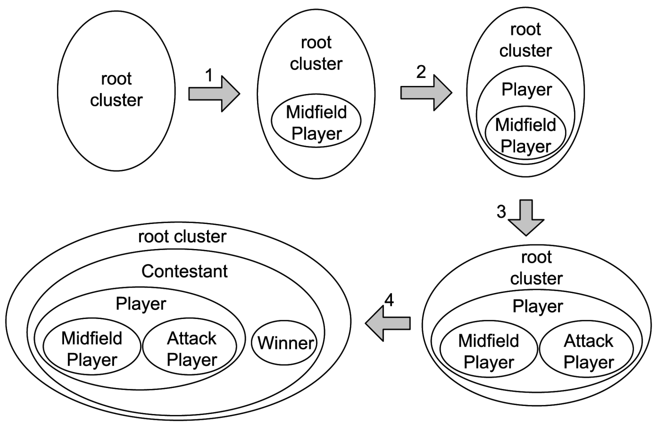


Fig. 7. Term taxonomy construction using the sample terms in Table 1.

- Step 2: The new term and its most similar cluster are merged to form a new cluster in three different ways:

  1. If the sense bag of the new term is a superset of the sense bag of its most similar cluster (i.e., the new term is more specific), we merge the new term into its most similar cluster as a subcluster (Fig. 6b) and the process is completed.
  2. If the sense bag of the new term is a subset of the sense bag of its most similar cluster (i.e., the new term is more general), we merge its most similar cluster into the new term as a subcluster (Fig. 6c) and go to Step 1 to recursively insert the merged cluster (the expanded cluster of the new term).
  3. Else merge the new term with its most similar cluster to form a new cluster (Fig. 6d) and go to Step 1 to recursively insert the merged cluster.

In Case 2 and Case 3 of Step 2 described above, the merged cluster is to be recursively inserted into the taxonomy. According to Step 1, we need to find a most similar cluster for the merged cluster. We know that the merged cluster is generated from and thus similar to the most similar cluster c_{sim} of the new term. Intuitively, the merged cluster is very likely to be similar to a supercluster of c_{sim}. Thus, a *local search* (i.e., searching the superclusters of c_{sim}) is adopted for locating the most similar cluster for the merged cluster.

In addition, we use another heuristic strategy to simplify the taxonomy structure. We observe that terms sharing little meaning may still be grouped into a cluster, e.g., terms "go" and "stop." To avoid grouping such irrelevant terms into a cluster, we define a *minimum similarity threshold* ($minsim$). If the similarity between a new cluster and its most similar cluster is below $minsim$, we directly insert the new cluster as a subcluster of the root.

We illustrate the term taxonomy construction process using the sample terms listed in Table 1.

- As shown in Fig. 7, when the first term "Midfield Player" (an atomic cluster) is inserted, the root cluster is the only cluster in the taxonomy. Therefore, we cannot find a most similar cluster for "Midfield Player." Thus, "Midfield Player" is directly added as a subcluster of the root cluster.

- When the term "Player" is inserted, "Midfield Player" is found as the most similar cluster. As the sense bag of "Player" is a subset of the sense bag of "Midfield Player" (i.e., "Player" is more general), "Midfield Player" is merged into "Player" as a subcluster. Note that we need to recursively insert the expanded "Player" cluster into the term taxonomy. As the root cluster is the only supercluster of "Midfield Player," we cannot find a most similar cluster that has nonzero similarity to the expanded "Player" cluster. "Player" is thus added as a subcluster of the root cluster.
- When the term "Attack Player" is inserted, the "Player" cluster is identified as the most similar cluster. As "Attack Player" is more specific than "Player," "Attack Player" is directly inserted as a subcluster of the cluster "Player."
- Finally, when the term "Winner" is inserted, the most similar cluster "Player" is identified. "Player" and "Winner" are merged into a new cluster, labeled by "contestant" (as sense 09476765_contestant appears in both sense bags of "Player" and "Winner"). We then recursively add the "contestant" cluster into the term taxonomy. As the root cluster is the only supercluster of "Player," we cannot find a cluster that has nonzero similarity to "contestant." Therefore, the "contestant" cluster is inserted as a subcluster of the root.

3.3 RDF Encoding

The term taxonomy is encoded using RDFS as a part of our RDF vocabulary (a schema file) for describing semantic relations. Each term cluster in the taxonomy is mapped to as RDFS class. For any two clusters c_1 and c_2 where c_2 is a subcluster of c_1, the RDFS class of c_2 is defined as a subclass of the RDFS class of c_1 using the "rdfs:subClassOf" predicate.

To encode semantic relations in RDF, the three predicates, i.e., *agent*, *theme*, and *modifiedBy*, are defined as *RDF predicates* (instances of *rdf:Property*) in our RDF vocabulary.[2] In addition, we treat each NP/VP extracted from text as an *RDF resource* with a Unified Resource Identifier (URI). As atomic clusters (synonym groups) are defined as RDFS classes, each NP/VP is thus defined as an instance of the RDF class corresponding to its synonym group.

4 Mining Generalized Associations from RDF Metadata

In the last section, we introduce the semantic relation extraction process that generates RDF metadata describing semantic relations together with an RDF vocabulary defining the three RDF predicates (*agent*, *theme*, and *modifiedBy*) and a hierarchy of RDF classes (encoded term taxonomy). In this section, we present the *GP-Close* algorithm for mining frequent generalized association patterns based on the extracted RDF metadata and RDF vocabulary.

2. Note that the technologies used in our work are not limited by the predefined types of relations. New types of relations can be included by expanding the set of rules for relation extraction. In addition, our generalized relation association mining algorithm can be applied on any RDF document collection with the existence of a RDF vocabulary.

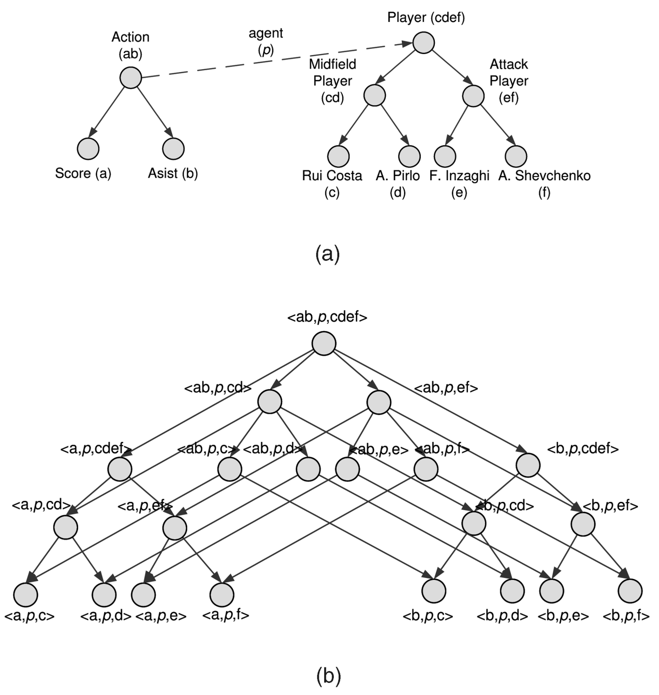


Fig. 8. A sample RDF vocabulary and the inferred relation hierarchy. (a) A sample RDF vocabulary. (b) The generalized relation hierarchy.

4.1 Problem Statement

First, we present the necessary notations and the problem statement for mining generalized associations on RDF metadata with the assistance of an RDF vocabulary. For simplicity, the mining task is defined based on a simplified view of the RDF model.

Definition 1. *Let $\mathcal{V} = \{\mathcal{E}, \mathcal{P}, \mathcal{H}, domain, range\}$ define an* ***RDF vocabulary*** *in which $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$ is a set of entity identifiers, $\mathcal{P} = \{p_1, p_2, \ldots, p_n\}$ is a set of predicate identifiers, and $\mathcal{H}$ is a directed acyclic graph. An edge in $\mathcal{H}$ represents an* ***is-a*** *relationship between a pair of entities. If there is an edge from e_1 to e_2, we say e_1 is a parent of e_2 and e_2 is a child of e_1. We call $\hat{e}$ an* ***ancestor*** *of e if there is a path from $\hat{e}$ to e in $\mathcal{H}$. The* ***function***, *$domain : \mathcal{P} \rightarrow 2^{\mathcal{E}}$, relates a predicate to a set of entities that can be its subject (defining the domain of the predicate). The* ***function***, *$range : \mathcal{P} \rightarrow 2^{\mathcal{E}}$, relates a predicate to a set of entities that can be its object (defining the range of the predicate).*

Fig. 8 a shows a sample RDF vocabulary

$$\mathcal{V} = \{\mathcal{E}, \mathcal{P}, \mathcal{H}, domain, range\},$$

where

$$\mathcal{E} = \{a, b, c, d, e, f, ab, cd, ef, cdef\},$$
$$\mathcal{P} = \{p\},$$
$$domain(p) = \{a, b, ab\},$$

and $range(p) = \{c, d, e, f, cd, ef, cdef\}$.

The above definition simplifies the model of an RDF vocabulary in two aspects:

- We treat instances and RDFS classes both as entities. Correspondingly, we treat "rdf:type" and "rdfs:sub ClassOf" predicates as an "is-a" relation between entities without discrimination. Through this way, we can represent instances and RDF classes in one taxonomy so that the mining task can be simplified. In our study, each entity corresponds to a term in text (an instance) or a cluster of terms in the term taxonomy (an RDFS class).
- For simplicity, we do not consider the hierarchy of RDF predicates at the current stage. In Section 5, we will show that the predicate hierarchy can be easily incorporated into the generalized association mining framework.

Definition 2. *Given an RDF vocabulary*

$$\mathcal{V} = \{\mathcal{E}, \mathcal{P}, \mathcal{H}, domain, range\},$$

*we define a **relation** (RDF statement) r on $\mathcal{V}$ as a triplet $< x, p, y >$, where $x, y \in \mathcal{E}$, $p \in \mathcal{P}$, $x \in domain(p)$, and $y \in range(p)$. We call x, p, and y the **subject**, the **predicate**, and the **object** of r, respectively. A relation $\hat{r} =< x_1, p_1, y_1 >$ is called a **generalized relation** (**ancestor**) of another relation $r =< x_2, p_2, y_2 >$, if and only if:*

1. $\hat{r} \neq r$,
2. $p_1 = p_2$,
3. x_1 *is an ancestor of* x_2 *or* $x_1 = x_2$, *and*
4. y_1 *is an ancestor of* y_2 *or* $y_1 = y_2$.

*We use $\mathcal{G}(r)$ to denote the set of relations containing r and all its generalized relations. We use $\mathcal{R}^{\mathcal{V}}$ to denote the set of all relations on $\mathcal{V}$. Relations in $\mathcal{R}^{\mathcal{V}}$ and their generalization/specialization relationships form a **relation hierarchy** $H_r^{\mathcal{V}}$.*

For example, Fig. 8b shows the generalized relation hierarchy containing all relations that can be derived from the sample RDF vocabulary in Fig. 8a.

Definition 3. *A **relationset** (**pattern**) is a set of relations $X \subseteq \mathcal{R}^{\mathcal{V}}$ where X does not contain both a relation and its ancestor. We call X a **generalized relationset** of another relationset Y and Y a **specialized relationset** of X, if and only if: 1) $X \neq Y$, 2) $\forall r \in X, \exists r^* \in Y$ such that $r = r^*$ or r is an ancestor of r^*, and 3) $\forall r^* \in Y, \exists r \in X$ such that $r = r^*$ or r is an ancestor of r^*. Given a set of RDF documents $\mathcal{D}$, where each document consists of a set of relations, we say an RDF document supports a relationset X if it contains X or a specialized relationset of X. The **support** of a relationset X (supp(X)) in the RDF document set $\mathcal{D}$ is defined as the proportion of the RDF documents that support X.*

For example, given the sample RDF vocabulary and the relation hierarchy in Fig. 8, $\{< \mathrm{a, p, ef} >, < \mathrm{b, p, c} >\}$ is a relationset and it is also a generalized relationset of $\{< \mathrm{a, p, e} >, < \mathrm{b, p, c} >\}$. Given a sample set of RDF documents in Table 2, the support of $\{< \mathrm{a, p, ef} >, < \mathrm{b, p, c} >\}$ is 50 percent as document 1 and document 2 contain its specialized relationsets.

Problem Statement. *Given an RDF vocabulary and a set of RDF documents, we aim to extract* frequent relationsets (frequent relation association patterns*) whose supports are larger than a predefined* minimum support *(minsup).*

TABLE 2
A Sample Set of RDF Documents

id	RDF Documents
1	<a, *p*, e> <a, *p*, f> <b, *p*, c>
2	<a, *p*, e> <b, *p*, c>
3	<a, *p*, d>
4	<a, *p*, f>

Currently, we use *support* as the criterion for generalized relationset pruning. Though alternative interestingness measures are available in the field of association rule mining [41], [42], support is still one of the most widely used as it represents the *statistical significance* of a pattern [42]. In fact, no matter which interestingness measure is used, the extracted patterns must be statistically significant, i.e., satisfying the minimum support. Our pattern mining and pruning approach, to be introduced in the subsequent sections, is only based on support. After the frequent (statistical significant) patterns are identified, other interestingness measures [41], [42], such as *confidence*, ϕ-coefficient, or J-Measure, can be further applied to measure the *strength* of the patterns. In addition, all the information needed for calculating such pattern strength is the supports of the frequent patterns. Therefore, strength calculation can be treated as a postprocessing of the frequent patterns and it will not influence the frequent pattern mining process.

4.2 Overgeneralization Problem

First, we introduce the overgeneralization problem using an example. With the sample set of RDF documents shown in Table 2 and the RDF vocabulary in Fig. 8a, we can identify the set of all frequent generalized relationsets with a minimum support of 50 percent given in Table 3.

Among those frequent relationsets, we can find some interesting patterns. For example, $\{< \mathrm{a}, p, \mathrm{e} >, < \mathrm{b}, p, \mathrm{c} >\}$, i.e.,

$$\{< \mathrm{Score, agent, F.Inzaghi} >, < \mathrm{Assist, agent, RuiCosta} >\}$$

has a support of 50 percent. This pattern may be explained as "Rui Costa often assists F. Inzaghi to score." On the other hand, its generalized relationset $\{< \mathrm{a}, p, \mathrm{ef} >, < \mathrm{b}, p, \mathrm{c} >\}$, i.e.,

$$\{< \mathrm{Score, agent, AttackPlayer} >, \\ < Assist, agent, RuiCosta >\}$$

also has the same support of 50 percent (see Fig. 9a). Intuitively, with the same support, a specialized pattern is more interesting than its generalized pattern as the information conveyed by the specialized pattern is more precise. Therefore, the pattern "Rui Costa always assists attack players to score" is overgeneralized and not interesting. Based on this observation, we define overgeneralization as follows:

Definition 4. *A frequent relationset X is **overgeneralized** if there exists a specialized relationset Y of X with*

TABLE 3
Frequent Generalized Relationsets in the Sample RDF Documents ($minsup = 50\%$)

Support	Frequent Generalized Relationsets ($minsup$=50%)
50%	{<a, *p*, e>} {<a, *p*, f>} {<b, *p*, c>} {<a, *p*, e> <b, *p*, c>} {<ab, *p*, c>} {<ab, *p*, e>} {<ab, *p*, f>} {<b, *p*, cd>} {<ab, *p*, cd>} {<b, *p*, cdef>} {<a, *p*, e> <b, *p*, cd>} {<a, *p*, e> <b, *p*, cdef>} {<a, *p*, e> <ab, *p*, c>} {<a, *p*, e> <ab, *p*, cd>} {<a, *p*, ef> <ab, *p*, c>} {<a, *p*, ef> <ab, *p*, cd>} {<a, *p*, ef> <ab, *p*, e>} {<a, *p*, ef> <ab, *p*, f>} {<a, *p*, ef> <b, *p*, c>} {<a, *p*, ef> <b, *p*, cd>} {<a, *p*, ef> <b, *p*, cdef>} {<a, *p*, cdef> <ab, *p*, c>} {<a, *p*, cdef> <ab, *p*, cd>} {<a, *p*, cdef> <ab, *p*, e>} {<a, *p*, cdef> <ab, *p*, f>} {<a, *p*, cdef> <b, *p*, c>} {<a, *p*, cdef> <b, *p*, cd>} {<a, *p*, cdef> <b, *p*, cdef>} {<a, *p*, cdef> <ab, *p*, ef>} {<ab, *p*, e> <ab, *p*, c>} {<ab, *p*, e> <ab, *p*, cd>} {<ab, *p*, e> <b, *p*, c>} {<ab, *p*, e> <b, *p*, cd>} {<ab, *p*, e> <b, *p*, cdef>} {<ab, *p*, ef> <ab, *p*, c>} {<ab, *p*, ef> <ab, *p*, cd>} {<ab, *p*, ef> <b, *p*, c>} {<ab, *p*, ef> <b, *p*, cd>} {<ab, *p*, ef> <b, *p*, cdef>} {<ab, *p*, c> <b, *p*, cd>} {<ab, *p*, c> <b, *p*, cdef>} {<ab, *p*, cd> <b, *p*, cdef>}
75%	{<a, *p*, ef>} {<ab, *p*, ef>}
100%	{<a, *p*, cdef>} {<ab, *p*, cdef>}

$supp(X) = supp(Y)$.

In Table 3, 41 frequent patterns (highlighted with underlines) are overgeneralized. It means that almost 89 percent (41 out of 46) of the patterns are not useful. In a real RDF data set, the proportion of overgeneralized patterns may be even higher. The existence of overgeneralized patterns not only implies redundancy in the mining result, but also seriously increases the computation cost. A strategy for pruning overgeneralized pattern is thus needed for efficiently mining generalized associations in RDF-like databases.

4.3 Overgeneralization Reduction: A Generalization Closure-Based Approach

In this section, we introduce our method for overgeneralization reduction based on the notion of generalization closures. Informally, a generalization closure of a relationset X, denoted as $\varphi(X)$, is an RDF relation set containing all relations in X and all their generalized relations. We can see that an RDF document which supports a relationset X must also support its generalization closure and vice versa, i.e., $supp(X) = supp(\varphi(X))$. As an example, Fig. 9b shows the generalization closures of the relationsets X and Y in Fig. 9a.

The formal definition of generalization closure is given below.

Definition 5. *Given an RDF vocabulary*

$$\mathcal{V} = \{\mathcal{E}, \mathcal{P}, \mathcal{H}, domain, range\},$$

we define a function φ on $2^{\mathcal{R}^{\mathcal{V}}}$: $\varphi(X) = \bigcup_{r \in X} \mathcal{G}(r)$, where X is a relationset and $X \subset \mathcal{R}^{\mathcal{V}}$. ($\mathcal{G}(r)$ is a set of relations that contains r and all its generalized relations, see Definition 2.) φ is a closure operator,[3] *[43] called the* ***generalization closure operator****. $\varphi(X)$ is called the* ***generalization closure*** *of X. Given two relationsets X and Y, we say $\varphi(Y)$ can* ***subsume*** *$\varphi(X)$ if $\varphi(X) \subset \varphi(Y)$ and $supp(\varphi(X)) = supp(\varphi(Y))$.*

Referring to Fig. 9, we can see that if a relationset X is a generalized relationset of Y, $\varphi(X)$ must be a proper subset of $\varphi(Y)$, i.e., $\varphi(X) \subset \varphi(Y)$. In fact, for any relation in $\varphi(X)$ (either a generalization of a relation in X or an identical of a relation in X), it must be either a generalization of a relation in Y or an identical of a relation in Y, i.e., it must be in $\varphi(Y)$.

It follows that if a relationset X is an overgeneralization of a relationset Y, both $\varphi(X) \subset \varphi(Y)$ and $supp(\varphi(X)) = supp(\varphi(Y))$ hold, i.e., $\varphi(X)$ is not closed. A pattern (relationset) is closed if it does not have a proper superset having the same support [21], [23], [44]. We say a generalization closure $\varphi(X)$ of a relationset X is *closed* if there does not exist a relationset Y such that $\varphi(Y)$ can subsume $\varphi(X)$. Based on the above observations, the following lemma holds.

Lemma 1. *Given a frequent relationset X, if $\varphi(X)$ is* closed*, X is $\underline{not}$* overgeneralized*.*

Proof. Suppose X is overgeneralized. It follows that there is a specialized pattern Y of X, where $supp(Y) = supp(X)$ (see Definition 4). In addition, as X is a generalized pattern of Y, $\varphi(X) \subset \varphi(Y)$ holds. Note that the generalization closures of X and Y have the same support with X and Y, respectively, i.e., $supp(X) = supp(\varphi(X))$ and $supp(Y) = supp(\varphi(Y))$. Therefore,

Pattern ***X*** (supp=50%):
{<Score, *agent*, Attack Player>, <Assist, agent, Rui Costa>}

Pattern ***Y*** (supp=50%):
{<Score, *agent*, F. Inzaghi>, <Assist, agent, Rui Costa>}

(a)

Generalization Closure of ***X*** (supp=50%):
{<Score, *agent*, Attack Player>, <Action, agent, Attack Player>, <Score, agent, Player>, <Assist, *agent*, Rui Costa>, <Action, *agent*, Rui Costa>, <Assist, *agent*, Midfield Player>, <Action, agent, Midfield Player>, <Assist, agent, Player>, <Action, agent, Player>}

Generalization Closure of ***Y*** (supp=50%):
{<Score, *agent*, F. Inzaghi>, <Score, *agent*, Attack Player>, <Action, agent, F. Inzaghi>, <Action, agent, Attack Player>, <Score, agent, Player>, <Assist, *agent*, Rui Costa>, <Action, *agent*, Rui Costa>, <Assist, *agent*, Midfield Player>, <Action, agent, Midfield Player>, <Assist, agent, Player>, <Action, agent, Player>}

(b)

Fig. 9. Illustrations of the overgeneralization and the generalization closure. (a) An illustration of overgeneralization problem: X is an overgeneralized pattern of Y. (b) The generalized closures of the pattern X and Y in Fig. 9a.

3. In [43], φ is generally used to represent a closure operator. For simplicity, this paper uses φ to refer to the generalization closure operator, FCA.

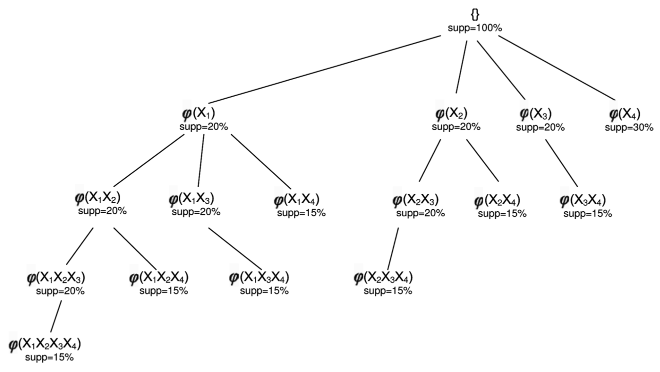


Fig. 10. Closure enumeration tree.

$\varphi(X) \subset \varphi(Y)$ and $supp(\varphi(X)) = supp(\varphi(Y))$, i.e., $\varphi(X)$ is not closed. It is contradictory to the statement that $\varphi(X)$ is closed. □

Lemma 1 shows that, if we extract only those patterns of which the generalization closures are closed, all overgeneralized patterns will be pruned. This motivates us to mine *closed generalization closures* by traversing the closure search space for overgeneralization reduction.

4.4 Mining Closed Generalization Closures

In the last section, we show that the task of overgeneralization reduction can be converted into a closed pattern mining problem by using the notion of generalization closures. In this section, we introduce the mining process for discovering closed generalization closures in an intuitive way. The detailed algorithms will be presented in the next section.

Note that, given two generalization closures $\varphi(X)$ and $\varphi(Y)$, $\varphi(X) \cup \varphi(Y)$ is also a generalization closure, i.e., $\varphi(X \cup Y)$. Thus, we can start with the generalization closures of 1-frequent relationsets (frequent relations) and gradually enumerate larger closures by merging the smaller ones. Fig. 10 shows a closure enumeration (search) tree based on four 1-frequent relationsets (X_1, X_2, X_3, and X_4), wherein the closures are enumerated in a depth-first search (DFS) manner.

In the closure enumeration tree in Fig. 10, each node is a unique generalization closure. Its children or descendants are the closures that expand it, i.e., its proper supersets. We can see that if a closure and its children have the same support, this closure is not closed and, thus, can be pruned. We prune such a nonclosed closure by replacing it with the union of its *equal-support children* (i.e., its *child-closures* that have the same support). For example, in Fig. 10, $\varphi(X_1)$ and its two child-closures, $\varphi(X_1X_2)$ and $\varphi(X_1X_3)$, have the same support of 20 percent, so $\varphi(X_1)$ is replaced by $\varphi(X_1X_2X_3)$, the union of $\varphi(X_1X_2)$ and $\varphi(X_1X_3)$ (see Fig. 11).

For a nonclosed generalization closure $\varphi(X)$, the union of its equal-support children is the largest expansion of $\varphi(X)$ that can preserve $\varphi(X)$'s support. Therefore, this union is *locally closed* in the subtree rooted at $\varphi(X)$. For example, in Fig. 10, $\varphi(X_1X_2X_3)$ is the largest expansion of $\varphi(X_1)$ that maintains the support of 20 percent. It is thus locally closed in the subtree with the root $\varphi(X_1)$. If a node in the closure enumeration tree does not have equal-support children, the node itself is locally closed (see $\varphi(X_2X_3)$ or $\varphi(X_3)$ in Fig. 10 for an example).

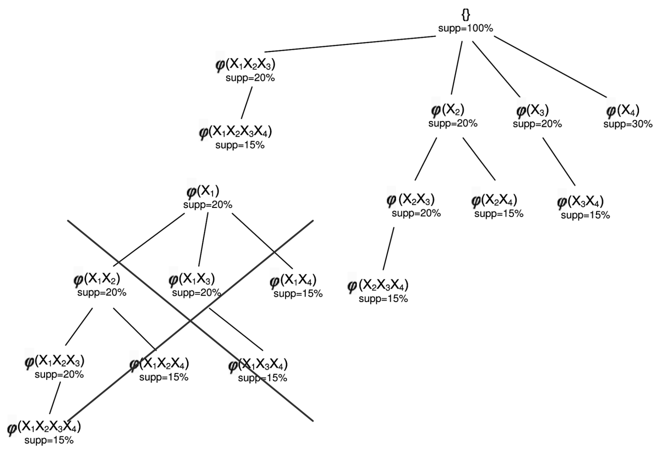


Fig. 11. Pruning nonclosed closure in a closure enumeration tree.

The locally closed closures are candidates to be *globally closed*. For determining whether a locally closed closure is globally closed, we need to examine whether there is an identified (globally) closed closure containing it and having the same support (i.e., subsuming it). If there is no such identified closed closure, it is globally closed. Therefore, by recursively traversing the closure enumeration tree, the entire set of the closed generalization closures can be discovered.

The closed generalization closure mining process is inspired by the existing closed pattern mining approaches, such as [21], [23], [44]. However, the existing approaches are designed for mining *atomic* or *element* patterns (itemsets), not *generalized* patterns. Specifically, they do not involve a taxonomy in their mining tasks and, therefore, do not use taxonomic information for pruning pattern search space.

In the next section, we present the *GP-Close* algorithm which is developed for mining closed generalization closures. Besides using the closed pattern mining technique described above, the GP-Close algorithm employs additional pruning methods for reducing pattern search space. In particular, we will show that *taxonomy-based pruning techniques* can play an essential role in saving computation cost.

4.5 GP-Close Algorithm

In this section, we present an algorithm, called GP-Close (Closed Generalized Pattern Mining) that discovers the set of closed generalization closures instead of all frequent generalized patterns.

4.5.1 *Algorithm Design*

The pseudocode of GP-Close algorithm is presented in Algorithm 1 and Algorithm 2. GP-Close algorithm first initializes the enumeration tree to contain only the root closure, i.e., an empty set with the support of 100 percent, and a set of closures of 1-frequent relationsets as the child-closures of the root (see Algorithm 1, lines 1-2). Starting from the root closure of the empty set, the closure enumeration process (Algorithm 2) recursively traverses the closure enumeration tree to discover closed generalization closures.

Algorithm 1 GP-Close.

```
Input:
  RDF database: $\mathcal{D}$
  Generalized relation lookup table: $GRT$
  Support Threshold: $minsup$
Output:
  The set of all closed frequent generalization closures: $\mathcal{C}$
1: $root = \emptyset$; $root.supp = 1$ //initialize closure enumeration
     tree root
2: $root.children = \{\varphi\{r\} | r \in \mathcal{R}^{\mathcal{V}} \wedge supp(r) \geq minsup\}$
     //Constructing closures of 1-frequent relationsets (by
     looking up $GRT$) as child-closures of $root$
3: Sort($root.children$) //sort child-closures in a
     specialization-first (length-decreasing/
     support-increasing) manner
4: Closure-Enumeration($root$, $\mathcal{C} = \emptyset$)
5: return $\mathcal{C}$
```

Algorithm 2 Closure-Enumeration.

```
Input:
  A node in the closure enumeration tree : $n$
  A set of discovered frequent closed generalization
    closures: $\mathcal{C}$
Output:
  The expanded set of frequent closed generalization
    closures: $\mathcal{C}$
1: If $\exists c^{ast} \in \mathcal{C}$ where $c^*$ subsumes $n$, return $\mathcal{C}$. //Subtree
     Pruning
2: Children-Prune($n.children$). //Child-Closure Pruning
3: $c$ = Closed-Closure($n$) //Generate a locally closed
     generalization closure $c$ and $c$ must also be globally
     closed according to our subtree pruning strategy.
4: $\mathcal{C} = \mathcal{C} \cup \{c\}$ //Insert $c$ into the frequent closed
     generalization closure set.
5: for each child-closure $child_i$ of $n$ where
   $child_i \in n.children$ do
6:    Generating the child-closures of $child_i$ by merging
      $child_i$ with one of its subsequent siblings:
      $child_i.children = \{gc_{ij} \mid gc_{ij} = child_i \cup child_j;$
      $child_j \in n.children \wedge i < j\}$
7:    Closure-Enumeration($child_i$, $\mathcal{C}$) //Recursively visit
      the child-closure of the current tree node $n$
8: end for
9: return $\mathcal{C}$
```

When a tree node n is visited, we first check whether the closure n can be subsumed by an identified closed generalization closure. If so, the current tree node n and all its descendants can be pruned (Algorithm 2, line 1). This is known as the *subtree pruning* strategy. If n cannot be subsumed, our *child-closure pruning* strategy is applied for pruning n's child-closures (Algorithm 2, line 2). (See Section 4.5.2 for more details of our pruning strategies.)

Then, for the current tree node n, a locally closed generalization closure c is generated (Algorithm 2, line 3). As discussed in the last section, there are two cases to generate the locally closed closures. In the first case, a tree node n is locally closed (i.e., $c = n$) if it does not have any equal-support children. The other case is that the current tree node n has equal-support children and the union of its equal-support children is locally closed. For the second case, we prune the current tree node n by replacing it with the union of its equal-support children c (see the last section). If the locally closed closure c cannot be subsumed by an identified closed closure, it is deemed to be a globally closed closure. In the next section, we will show that if a tree node n cannot be pruned by our subtree pruning strategy (Algorithm 2, line 1), the locally closed closure c generated based on n must be globally closed. Therefore, we can directly insert it into the frequent closed generalization closure set (Algorithm 2, line 4).

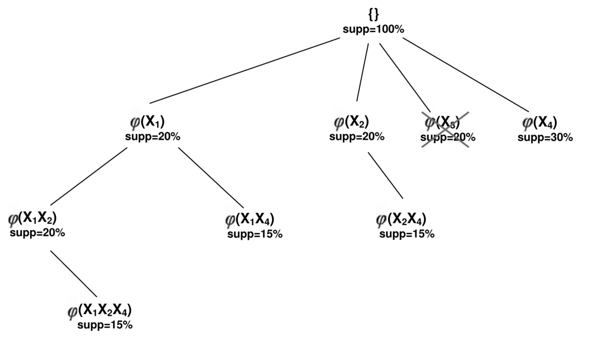
Finally, Algorithm 2 (lines 5-8) recursively visits the child-closures of the current tree node n. Before recursively visiting a child-closure $child_i$ of n, we need to first generate the child-closures of $child_i$ (Algorithm 2, line 6) as these child-closures are needed for determining whether $child_i$ is locally closed when $child_i$ is visited. When the child-closures of $child_i$ are generated, their supports are counted (see Section 4.5.3) and the infrequent child-closures are removed.

4.5.2 Pruning Child-Closures and Subtrees

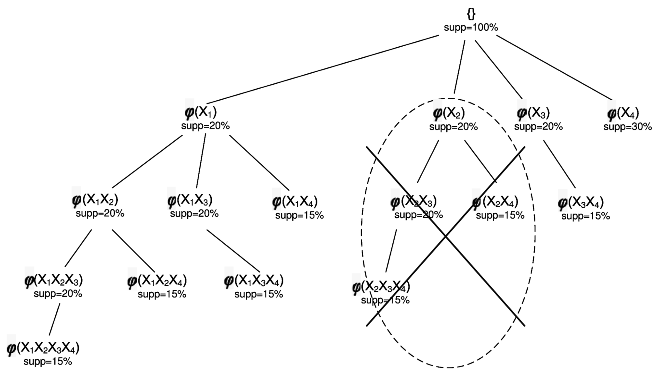
Besides the basic closed pattern pruning strategy, we further employ two additional pruning techniques to reduce the pattern search space.

First, we note that a full closure enumeration tree, whose root has n child-closures, has a total of 2^n nodes. Thus, pruning one child-closure of the tree root will reduce *half of the pattern search space*. For example, as the tree in Fig. 10 has four child-closures under the root, the total number of nodes in the tree is 16 (2^4). Suppose X_3 is a generalized relationset of X_2. This implies $\varphi(X_3) \subset \varphi(X_2)$. As $supp(X_3) = supp(X_2) = 20\%$, we conclude that X_3 is an overgeneralization of X_2. Therefore, $\varphi(X_3)$ is subsumed by $\varphi(X_2)$, i.e., any pattern containing $\varphi(X_3)$ must also contain $\varphi(X_2)$. Therefore, $\varphi(X_3)$ can be pruned. Fig. 12a shows the pruning results. Upon pruning of $\varphi(X_3)$, half of the tree nodes are gone. Therefore, early removal of redundant child-closures of a (sub)closure enumeration tree is highly desirable. In Algorithm 2, line 2, the function *Children-Prune* prunes the redundant child-closures of the current tree node visited. We call this pruning technique the *child-closure pruning* technique. This is a taxonomy-based pruning technique, which is very efficient for pattern search space pruning and is not yet used in other closed pattern mining approaches.

Second, we note that all descendants of an enumeration tree node are its expansions (proper supersets). If the tree node can be subsumed by an identified closed closure, i.e., the tree node is not closed, it follows that traversing the (sub)enumeration tree rooted at this node cannot generate new closed generalization closures. Therefore, the entire (sub)tree can be pruned (Algorithm 2, line 1). For example, in Fig. 12b, the closure $\varphi(X_2)$ is subsumed by the closed closure $\varphi(X_1X_2X_3)$ and all its descendants can be subsumed by the descendant of $\varphi(X_1X_2X_3)$. Thus, the subtree with the root $\varphi(X_2)$ can be pruned. In addition, we can see that if a tree node n cannot be subsumed by any identified closed closure, the locally closed closure generated based on n cannot be subsumed by any identified closed closure, i.e.,



(a)



(b)

Fig. 12. Pruning the closure enumeration tree. (a) Child-closure pruning. (b) Subtree pruning.

it is globally closed. This is because if there is an identified closed closure that can subsume the locally closed closure which is a superset of n, it must also subsume n. It contradicts the statement that n cannot be subsumed.

There are three cases in which one closure can subsume another:

1. A *specialized closure* can subsume a generalized closure if they have the same support.
2. A closure can be subsumed by one of its *superclosures* if they have the same support.
3. A closure can be subsumed by a *superset of its specialized closures* if they have the same support.

Based on the above observations, a support-increasing and length-decreasing strategy is adopted for dynamic closure sorting (Algorithm 1, line 3). This sorting method implies that specialized closures will be enumerated first. This increases the occurrences of subsumption Cases 1 and 3 so that there is a higher probability that a subtree can be pruned. In this way, the tree traversing space will be further reduced.

4.5.3 Hybrid Support Counting

For simplifying the pseudocode in Algorithm 2, we do not explicitly show the pattern support counting process. In fact, our support counting method is basically a *transaction ID set* (*tidset*)-based approach [19], [23]. Here, each generalization closure is associated with a tidset containing the IDs of the RDF documents that support the closure. Then, if a closure is expanded by merging with another closure, the support of the expanded closure can be calculated by intersecting the tidsets of the two original closures.

Note that tidsets may not be able to fit into the physical memory, especially when the number of RDF documents is large and the database is dense. A hybrid counting strategy is used in GP-Close for handling data sets under different circumstances. The hybrid support counting is a combination of the database (DB) scan method with the use of a hash-tree structure [18] and the tidset-based counting. It allows a user to define a tidset buffer with a maximum buffer size. Support counting is initially performed by scanning DB. During the DB scanning and support updating, the algorithm tries to build tidsets for candidate closures if these tidsets can fit into the preallocated buffer. The constructed tidsets are then used for subsequent tidset-based support counting. If the tidsets cannot fit into the tidset buffer, the algorithm will still use DB scans for subsequent support counting.

4.6 Correctness and Complexity Analysis

For guaranteeing the correctness of our closed generalization closure-based pruning approach, the following theorem needs to be introduced:

Theorem 1. *The support of all frequent relationsets can be derived from the set of all frequent closed generalization closures.*

Proof. Based on the fact that each relationset X can be mapped into a generalization closure $\varphi(X)$ that has the same support as X, it follows that the support of X can be derived from $\varphi(X)$ by one of the following ways:

1. If $\varphi(X)$ is a closed closure, $supp(X) = supp(\varphi(X))$.
2. Otherwise, there exists a closed closure $\varphi(Y) \supset \varphi(X)$ and does not exist a closed closure $\varphi(Z)$ with $\varphi(X) \subset \varphi(Z) \subset \varphi(Y)$. It follows that $supp(X) = supp(\varphi(X)) = supp(\varphi(Y))$. □

Theorem 1 thus ensures that using closed generalization closures for overgeneralized pattern pruning will not cause any loss in information.

As the GP-Close algorithm adopts a hybrid support counting strategy, its running time depends on whether DB scans or tidsets are used for support counting. Here, we analyze two extreme cases, i.e., using tidsets only for support counting after 1-frequent relationsets are generated (unlimited tidset buffer) and using DB scans only for support counting (no tidset buffer).

Theorem 2. *The running time of GP-Close is between $O(|C| \cdot (l_{gc} \cdot log|C| + l_{gc} + l_{tidset}))$ (lower bound) and $O(|C| \cdot (l_{gc} \cdot log|C| + l_{gc} + C_{|doc|}^{l_{gc}} \cdot |D|))$ (upper bound), where l_{gc} is the average length of generalization closures, l_{tidset} is the average length of tidsets, $|C|$ is the number of frequent closed generalization closures, $|doc|$ is the average number of (generalized) relations in an RDF document, and $|D|$ is the total number of RDF documents.*

Proof. Note that, when traversing the generalization closure enumeration tree, we only visit those nodes based on which a closed generalization closure will be generated.

TABLE 4
Summary of Semantic Relation Extraction Results

RDF Data set	Documents	N_r	N_r^*	N_{gr}	n_{gr}	*NoC*	*NoR*
ICT-SB	106	1938	1665	48691	29	1578	1988
ICT-CB	128	2224	1814	52673	29	1790	2198

This is guaranteed by our subtree pruning strategy. Therefore, GP-Close performs $O(|C|)$ *full tree node accesses*. Each full access of a tree node n consists of three major operations, i.e., subtree pruning (Algorithm 2, line 1), child-closure pruning (Algorithm 2, line 2), and generating the child-closures for n's children (involving support counting) (Algorithm 2, line 6).

Subtree pruning is to check whether n can *be subsumed* by an identified closed closure, i.e., whether there is an identified closed closure that has the same support with n and also contains n. The cost of finding the (hashed) closed closures having the same support is $O(log|C|)$. The cost of checking whether a closed closure contains n is $O(l_{gc})$. Therefore, the total cost of subtree pruning is $O(l_{gc} \cdot log|C|)$.

Note that, in a closure enumeration tree, each nonleaf tree node on the average has two child-closures. The child-closure pruning on a tree node n involves checking whether one child-closure can *be subsumed* by another child-closures. Referring to the analysis of the subsumption checking for subtree pruning, we can see that the total cost of child-closure pruning is $O(l_{gc} \cdot log2 \cdot 2)$ or $O(l_{gc})$.

Finally, generating the child-closures for n's children means merging any pair of n's children to create larger closures. As n typically has two children, the algorithm, on average, generates only one child-closure for n's children. The main cost here is counting the support of the new closure. If we use tidsets for support counting, the cost of intersecting two tidsets is $O(l_{tidset})$; if we use DB scans, the cost of scanning $|D|$ documents (each having $C_{|doc|}^{l_{gc}}$ subsets with the length of l_{gc}) and updating the supports of the generalization closures stored in a hash-tree is $O(C_{|doc|}^{l_{gc}} \cdot |D|)$ [18].

Based on the above analysis, we can see that the overall computation cost of GP-Close is between $O(|C| \cdot (l_{gc} \cdot log|C| + l_{gc} + l_{tidset}))$ and

$$O(|C| \cdot (l_{gc} \cdot log|C| + l_{gc} + C_{|doc|}^{l_{gc}} \cdot |D|)).$$

When the document set is large (i.e., $|D|$ and l_{tidset} is large), the running time of GP-Close is approximately between $O(|C| \cdot l_{tidset})$ and $O(|C| \cdot C_{|doc|}^{l_{gc}} \cdot |D|)$. □

5 Experiments

Our experiments were performed on a desktop PC running Windows XP with a P4-2.6G CPU and 1 G RAM. The GP-Close algorithm was implemented using Java (JDK 1.4.2). Two variants of GP-Close with different sizes of tidset buffer were used in the experiments, namely, GP-Close-*0* with a tidset buffer of 0 KB and GP-Close-*50000* with a tidset buffer of 50,000 KB. GP-Close-*0* is used to illustrate the worst-case scenario wherein the support can only be calculated by using DB scans. On the other hand, GP-Close-*50000* is to show the best-case scenario in which tidsets can always fit into the main memory. Therefore, experimenting with GP-Close-*0* and GP-Close-*50000* can inform us about the performance boundary of our algorithm. We also implemented the Cumulate algorithm [9], an original algorithm for mining generalized association rules, as a reference algorithm for performance evaluation and comparison.

5.1 Semantic Relation Extraction

The data sets used in our experiments were collected from the online database of the International Policy Institute for Counter-Terrorism (ICT). The contents of the online documents of varying length were mainly descriptions of car bombing and suicide bombing events. We apply semantic relation extraction to extract semantic relations from the ICT suicide bombing (ICT-SB) documents and the ICT car bombing (ICT-CB) documents with a WordNet search depth ($WNSD$) of 2 and a minimum similarity threshold ($minsim$) of 0.1.

The statistics of the semantic relation extraction and term taxonomy construction are summarized in Table 4. We see that most of the extracted RDF relations are distinct, heightening the necessity of mining generalized patterns. In addition, Table 4 shows that the set of generalized relations which can be derived from the term taxonomy is much larger than the original set of relations stored in RDF metadata. This implies that there is a very large generalized relationset search space.

As Cumulate is not designed for mining RDF data, for a fair comparison, the extracted RDF metadata are stored as binary transaction files resided in the Microsoft new technology file system (NTFS). We assign a unique identifier (*rid*) for each (generalized) relation. Each RDF document is thus encoded as a transaction containing a set of *rid*s. The generalized relation hierarchy (see Fig. 8b for an example) is precomputed and stored as a generalized relation lookup table (*GRT*) in a binary file. Both Cumulate and GP-Close algorithms depend on this table for fast look up of generalized relations. Note that, to incorporate a hierarchy of RDF predicates in mining, we need to involve the predicate hierarchy for computing the GRT.

5.2 Mining Generalized Associations on RDF Metadata

5.2.1 Number of Patterns

We first do a comparison on the number of the patterns extracted by the two algorithms, Cumulate and GP-Close. Fig. 13 shows that the number of closed generalization closures can be one to two orders of magnitude smaller than

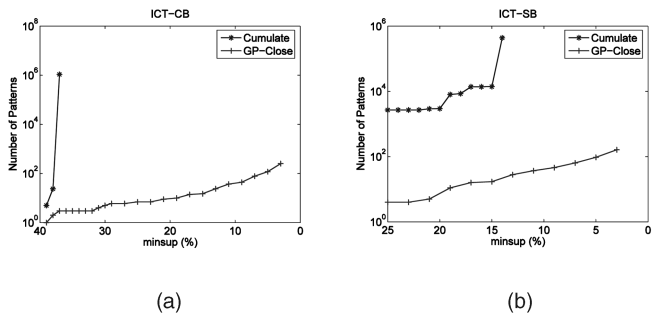


Fig. 13. Number of patterns. (a) ICT-CB. (b) ICT-SB.

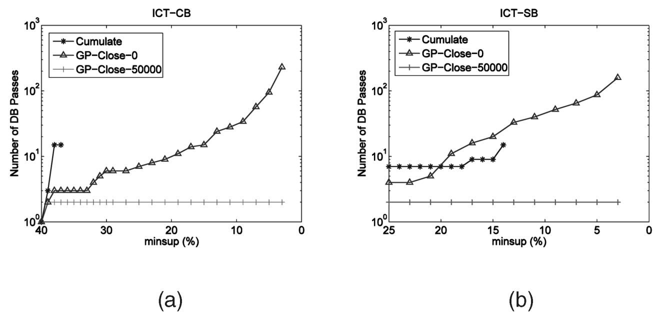


Fig. 15. Database scans.(a) ICT-CB. (b) ICT-SB.

the number of frequent relationsets discovered by Cumulate. This is especially so with a low *minsup*.

We can see that as *minsup* decreases, the number of frequent relationsets increases rapidly. The reason is that when *minsup* is low, larger and more specialized relationsets will be discovered. Note that the more specialized and larger a frequent relationset is, the more generalized relationsets it contains. And, all these generalized patterns are also frequent. This may not be a problem when mining generalized item associations [9] as an item has no inner structure and it usually has only one parent. However, when mining relation associations, the problem becomes critical as a relation is a triplet which consists of a subject, a predicate, and an object, which means a relation can be generalized in many different ways. Therefore, when relationsets become larger and more specialized, the number of their generalized patterns will increase dramatically.

As the Cumulate algorithm generates all frequent patterns, we can expect that the execution time of the Cumulate algorithm also increases very fast when *minsup* decreases. For the GP-Close algorithm, as it generates only a small set of the closed generalization closures, its execution time is expected to be less sensitive to *minsup*.

5.2.2 Time Efficiency

Fig. 14 shows the execution times of the algorithms by varying the minimum support (*minsup*). As expected, Cumulate can work properly only with high *minsup*. When *minsup* is high, the performance of the algorithms are close. The GP-Close-*0* is slightly slower than GP-Close-*50000* due to the fact that it involves more IO accesses. However, when *minsup* is low, both versions of the GP-Close algorithm can run more than one to two orders of magnitude faster. This can be explained by the number of patterns discovered by the various algorithms, i.e., Cumulate generates many more patterns than GP-Close.

5.2.3 Number of DB Scans

Fig. 15 shows the number of DB scans performed by the various algorithms. The overall trend is that the number of DB passes increases when *minsup* decreases. For GP-Close-*50000*, the tidsets can always fit into the tidset buffer after two DB scans. For GP-Close-*0*, DB scanning is the only way for it to calculate the support. Notice that, at each level of the closure enumeration tree, there may exist multiple branches. Thus, GP-Close-*0* needs to scan the database once for support counting at each of these branches. The number of DB scans performed by GP-Close-*0* increases rapidly due to the fact that more branches of the closure enumeration tree are visited when *minsup* decreases. However, for low *minsup*, though GP-Close-*0* scans DB for many more times than that of Cumulate, its execution time is still much shorter than Cumulate. Based on this observation, we can see that, when *minsup* is low, the bottleneck of the algorithms lies in CPU computation instead of IO access.

5.2.4 Scalability

The scalability of the GP-Close algorithm is also evaluated by replicating the ICT-CB database 100 to 500 times with an incremental step of 100, i.e., the largest data set contains more than 50,000 RDF documents. For introducing variations in the replicas, we randomly modify 1 percent of the RDF statements by replacing them with noisy statements. Then, the GP-Close algorithm is tested on the replicated

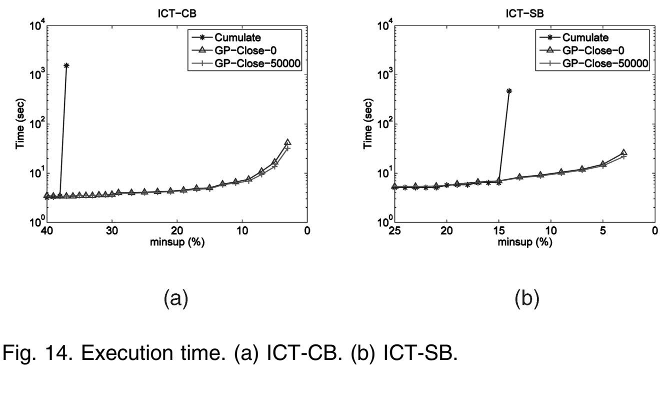


Fig. 14. Execution time. (a) ICT-CB. (b) ICT-SB.

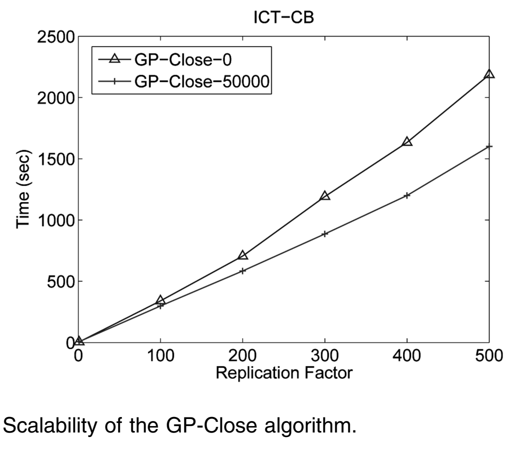


Fig. 16. Scalability of the GP-Close algorithm.

data sets with a fixed *minsup* of 10 percent. As shown in Fig. 16, the execution time of the GP-Close algorithm increases linearly with the replication factor.

5.3 Analysis of Patterns

For evaluating the quality of the patterns discovered, we analyze the patterns generated using GP-Close on the ICT-CB data set with a *minsup* of 7 percent. We examine each of the 78 generalized patterns (relationsets) to verify 1) if it is previously known and 2) if it is significant.

We observe that 71.8 percent (56 out of 78) of the patterns are commonsense patterns already known by people. For example, the pattern {<explode, agent, bomb>, <wound, theme, people>} (with support of 7.0 percent) describes a commonsense fact that, when a bomb explodes, people get wounded. Though such patterns may not be significant knowledge for human users, they may still be useful for the tasks of clustering or classification of Web documents as they reflect the underlying semantic structures of a particular domain. Ten out of 78 (12.8 percent) of the patterns are identified as previously unknown and not useful. These patterns usually involve general terms, such as "group" or "city." For example, {<kill, theme, group>} can be interpreted as "something related to a certain group is killed," with support of 12.6 percent. However, terms like "group" or "city" are too general to inform detailed semantic knowledge. The remaining 15.4 percent (12 out of 78) of the patterns are previously unknown and potentially useful. An example of such patterns is {<detonate, theme, truck>}, with support of 11.8 percent. In the ICT-CB (car bombing) data set, this pattern can be interpreted as "truck is often used by terrorists for carrying out car bombing." Another example is {<claim, agent, Terrorist_Group_001>, <claim, theme, responsibility>}, with support of 7.8 percent. This pattern can be explained as "the terrorist group with the identifier *Terrorist_Group_001* often claims responsibility for a car bombing event."

6 Conclusion

This paper has proposed a systematic approach for discovering knowledge from free-form textual Web content. Specifically, we present an automatic semantic relation extraction strategy to extract RDF metadata from textual Web content and an algorithm known as GP-Close for mining generalized patterns from RDF metadata. The experimental result shows that the GP-Close algorithm based on mining closed generalization closures can substantially reduce the pattern redundancy and perform much better than the original generalized association rule mining algorithm Cumulate in terms of time efficiency. The pattern analysis based on human validation shows that the proposed method is promising and useful.

Acknowledgments

The reported work is supported in part by the I^{2}R-SCE Joint Lab on Intelligent Media. The authors would like to thank the anonymous reviewers for providing many invaluable comments to the previous versions of the paper. They also thank Rohan Kumar Gunaratna and Teng-Kwee Ong at the Institute of Defence and Strategic Studies, Singapore, for providing their advice on data sources and the terrorist domain.

References

[1] J. Dörre, P. Gerstl, and R. Seiffert, "Text Mining: Finding Nuggets in Mountains of Textual Data," *Proc. Int'l Conf. Knowledge Discovery and Data Mining,* pp. 398-401, 1999.
[2] A.-H. Tan, "Text Mining: The State of the Art and the Challenges," *Proc. Pacific Asia Conf. Knowledge Discovery and Data Mining (PAKDD '99) Workshop Knowledge Discovery from Advanced Databases,* pp. 65-70, 1999.
[3] T. Berners-Lee, J. Hendler, and O. Lassila, "Semantic Web," *Scientific Am.,* vol. 284, no. 5, pp. 35-43, 2001.
[4] T. Berners-Lee, "Conceptual Graphs and Semantic Web—Reflections on Web Architecture," http://www.w3.org/DesignIssues/CG.html, 2001.
[5] J.F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley Longman, 1984.
[6] J.F. Sowa, "Conceptual Graphs: Draft Proposed American National Standard," *Proc. Int'l Conf. Computational Science,* pp. 1-65, 1999.
[7] N. Guarino, C. Masolo, and G. Vetere, "Ontoseek: Content-Based Access to the Web," *IEEE Intelligent Systems,* vol. 14, no. 3, pp. 70-80, May/June 1999.
[8] W3C, W3c RDF Schema Specification, http://www.w3.org/TR/rdf-schema/, 2005.
[9] R. Srikant and R. Agrawal, "Mining Generalized Association Rules," *Proc. Conf. Very Large Databases,* pp. 407-419, 1995.
[10] A. Inokuchi, "Mining Generalized Substructures from a Set of Labeled Graphs," *Proc. Int'l Conf. Data Mining,* pp. 415-418, 2004.
[11] W3C, W3c RDF Specification, http://www.w3.org/RDF/, 2005.
[12] A. Naeve, "The Human Semantic Web Shifting from Knowledge Push to Knowledge Pull," *Int'l J. Semantic Web Information Systems,* vol. 1, no. 3, pp. 1-30, 2005.
[13] A.P. Sheth, C. Ramakrishnan, and C. Thomas, "Semantics for the Semantic Web: The Implicit, the Formal and the Powerful," *Int'l J. Semantic Web Information Systems,* vol. 1, no. 1, pp. 1-18, 2005.
[14] H. Liu, P. Maes, and G. Davenport, "Unraveling the Taste Fabric of Social Networks," *Int'l J. Semantic Web Information Systems,* vol. 2, no. 1, pp. 42-71, 2006.
[15] N. Bassiliades, G. Antoniou, and I. Vlahavas, "A Defeasible Logic Reasoner for the Semantic Web," *Int'l J. Semantic Web Information Systems,* vol. 2, no. 1, pp. 1-41, 2006.
[16] F. Bry, C. Koch, T. Furche, S. Schaffert, L. Badea, and S. Berger, "Querying the Web Reconsidered: Design Principles For Versatile Web Query Languages," *Int'l J. Semantic Web Information Systems,* vol. 1, no. 2, pp. 1-21, 2005.
[17] R. Agrawal, T. Imielinski, and A.N. Swami, "Mining Association Rules between Sets of Items In Large Databases," *Proc. ACM SIGMOD Conf.,* pp. 207-216, 1993.
[18] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," *Proc. Conf. Very Large Databases,* pp. 487-499, 1994.
[19] A. Savasere, E. Omiecinski, and S.B. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases," *Proc. Conf. Very Large Databases,* pp. 432-444, 1995.
[20] H. Toivonen, "Sampling Large Databases for Association Rules," *Proc. Conf. Very Large Databases,* pp. 134-145, 1996.
[21] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering Frequent Closed Itemsets for Association Rules," *Proc. Int'l Conf. Database Theory,* pp. 398-416, 1999.
[22] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," *SIGMOD Record,* vol. 29, no. 2, pp. 1-12, 2000.
[23] M.J. Zaki and C.-J. Hsiao, "Charm: An Efficient Algorithm for Closed Itemset Mining," *Proc. Siam Conf. Data Mining,* 2002.
[24] R. Feldman and H. Hirsh, "Mining Associations in Text in the Presence of Background Knowledge," *Knowledge Discovery and Data Mining,* pp. 343-346, 1996, http://citeseer.ist.psu.edu/feldman96mining.html.
[25] J.D. Holt and S.M. Chung, "Multipass Algorithms for Mining Association Rules in Text Databases," *Knowledge Information System,* vol. 3, no. 2, pp. 168-183, 2001.

[26] M.M. y Gómez, A.F. Gelbukh, and A. López-López, "Text Mining at Detail Level Using Conceptual Graphs," *Proc. Int'l Conf. Complex Systems*, pp. 122-136, 2002.

[27] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan, "GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications," *Proc. 40th Anniversary Meeting Assoc. Computational Linguistics,* 2002.

[28] E. Brill, "A Simple Rule-Based Part of Speech Tagger," *Proc. Conf. Applied Natural Language Processing,* pp. 152-155, 1992.

[29] M. Collins, "A New Statistical Parser Based on Bigram Lexical Dependencies," *Proc. Conf. Assoc. Computational Linguistics,* pp. 184-191, 1996.

[30] C. Barriere, "From a Children's First Dictionary to a Lexical Knowledge Base of Conceptual Graphs," PhD dissertation, 1997.

[31] G.A. Miller, "Wordnet: A Lexical Database For English," *Comm. ACM,* vol. 38, no. 11, pp. 39-41, 1995.

[32] M.A. Hearst, "Automatic Acquisition of Hyponyms from Large Text Corpora," *Proc. 14th Conf. Computational Linguistics,* pp. 539-545, 1992.

[33] A. Maedche, V. Pekar, and S. Staab, "Ontology Learning Part One—On Discovering Taxonomic Relations from the Web," citeseer.ist.psu.edu/maedche02ontology.html, 2002.

[34] P. Cimiano, A. Hotho, and S. Staab, "Comparing Conceptual, Divisive and Agglomerative Clustering for Learning Taxonomies from Text," *Proc. European Conf. Artificial Intelligence,* pp. 435-439, 2004, citeseer.ist.psu.edu/630486.html.

[35] S.A. Caraballo, "Automatic Construction of a Hypernym-Labeled Noun Hierarchy from Text," *Proc. 37th Ann. Meeting Assoc. for Computational Linguistics on Computational Linguistics,* pp. 120-126, 1999.

[36] M. Hepp, "Products and Services Ontologies: A Methodology for Deriving Owl Ontologies from Industrial Categorization Standards," *Int'l J. Semantic Web Information Systems,* vol. 2, no. 1, pp. 72-99, 2006.

[37] J.J. Jiang and D.W. Conrath, "Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy," *Proc. Int'l Conf. Research on Computational Linguistics,* 1997, http://arxiv.org/pdf/cmp-lg/9709008.

[38] C. Leacock and M. Chodorow, "Combining Lexical Context and Wordnet Similarity for Word Sense Identification," *WordNet: An Electronic Lexical Database,* 1998.

[39] N. Seco, T. Veale, and J. Hayes, "An Intrinsic Information Content Metric for Semantic Similarity in Wordnet," *Proc. European Conf. Artificial Intelligence,* pp. 1089-1090, 2004.

[40] A. Budanitsky, "Semantic Distance in Wordnet: An Experimental, Application-Oriented Evaluation of Five Measures," *Proc. Workshop WordNet and Other Lexical Resources,* citeseer. ist.psu.edu/budanitsky01semantic.html, 2001.

[41] R. Hilderman and H. Hamilton, "Knowledge Discovery and Interestingness Measures: A Survey," citeseer.ist.psu.edu/hilderman99knowledge.html, 1999.

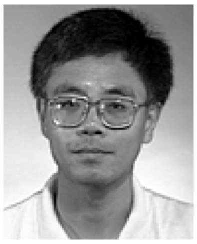
[42] P.-N. Tan, V. Kumar, and J. Srivastava, "Selecting the Right Interestingness Measure for Association Patterns," *Proc. Eighth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 32-41, 2002.

[43] B. Ganter and R. Wille, *Formal Concept Analysis: Mathematical Foundations.* Springer-Verlag, 1997.

[44] J. Wang, J. Han, and J. Pei, "Closet+: Searching for the Best Strategies for Mining Frequent Closed Itemsets," *Proc. Ninth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining,* pp. 236-245, 2003.

Tao Jiang received the BS degree in computer science and technology from Peking University in 2000. He is a PhD student in the School of Computer Engineering, Nanyang Technological University. His research interests include text mining, semantic Web mining, and multimedia information fusion in the Semantic Web.

Ah-Hwee Tan received the BS degree (first class honors) and the MS degree in computer and information science from the National University of Singapore, and the PhD degree in cognitive and neural systems from Boston University. He is an associate professor with the School of Computer Engineering, Nanyang Technological University. He is also the director of the Emerging Research Lab and the deputy program director of the MSc Program in Information Systems. He was a research manager and senior member of the research staff with the Institute for Infocomm Research, where he led research and development projects in knowledge discovery, document analysis, and information mining. Dr. Tan is an editorial board member of *Applied Intelligence*, a member of the ACM, and a senior member of the IEEE.



Ke Wang received the PhD degree from the Georgia Institute of Technology. He is currently a professor in the School of Computing Science at Simon Fraser University. Before joining Simon Fraser, he was an associate professor at the National University of Singapore. He has taught in the areas of database and data mining. His research interests include database technology, data mining and knowledge discovery, machine learning, and emerging applications, with recent interests focusing on the end use of data mining. This includes explicitly modeling the business goal and exploiting user prior knowledge. He has published in database, information retrieval, and data mining conferences, including SIGMOD, SIGIR, PODS, VLDB, ICDE, EDBT, SIGKDD, SDM, and ICDM. He is an associate editor of the *IEEE Transactions on Knowledge and Data Engineering* and has served on program committees for international conferences including DASFAA, ICDE, ICDM, PAKDD, PKDD, SIGKDD, and VLDB.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.